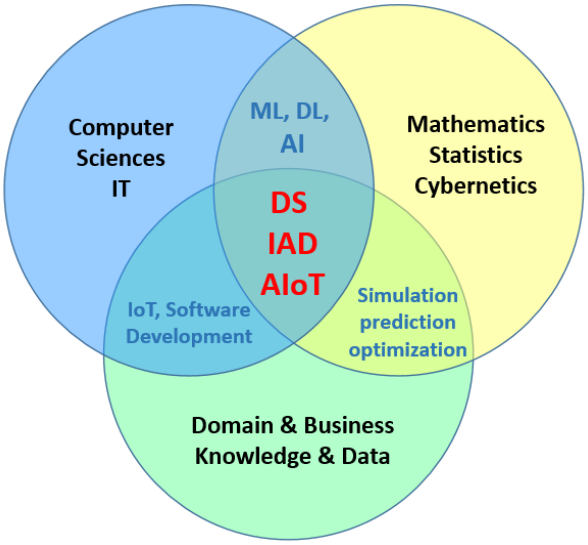
Vitalii B. Mokin, Mykola G. Pradivliannyi



Machine Learning, Intelligent Data Analysis and Artificial Intelligence of Things

Ministry of Education and Science of Ukraine
Vinnytsia National Technical University

Vitalii B. Mokin, Mykola G. Pradivliannyi

MACHINE LEARNING, INTELLIGENT DATA ANALYSIS AND ARTIFICIAL INTELLIGENCE OF THINGS

Electronic textbook

Vinnytsia
VNTU
2024

UDC [004.65-047.44+004.8/.9](075.8)
M78

Recommended for publication by the Academic Council of Vinnytsia National Technical University of the Ministry of Education and Science of Ukraine (protocol # 11 from 27.06.2024).

Reviewers:
O. V. Bisikalo, Doctor of Technical Sciences, Professor
R. N. Kvyetny, Doctor of technical sciences, Professor
V. V. Lytvyn, Doctor of technical sciences, Professor

Mokin, V. B.

M78 Machine Learning, Intelligent Data Analysis and Artificial Intelligence of Things : electronic textbook [Electronic resource] / V. B. Mokin, M. G. Pradivliannyi – Vinnytsia : VNTU, 2024. – (PDF, 230 p.)

The textbook contains theoretical information about the main concepts, methods, and tools of Data Science, Machine Learning, Artificial Intelligence, and Intelligent Data Analysis, the Artificial Intelligence of Things, as well as practical recommendations for the application of modern technologies in solving numerous applied tasks and problems of system analysis. A list of test questions for checking acquired theoretical knowledge and practical skills is provided.

The textbook for foreign students who study the specialties 124 "System Analysis" and 126 "Information Systems and Technologies" of the II and III levels when learning the following subjects: "Internet of things and intelligent data analysis", "Information technologies of monitoring and analysis of the state of complex systems", "Information technologies of monitoring and data analysis", "Information intelligent technologies", "System analysis", "Smart Technologies" as well as for students undergoing the industrial and pre-diploma practice, and for the pedagogical practice of graduate students. It can also be useful for students of other areas: management, finance, construction, cyber security, bio-, electrical and mechanical engineering, agriculture, biology, medicine, education, etc. The textbook provides many examples of tasks in these areas.

UDC [004.65-047.44+004.8/.9](075.8)

CONTENTS

INTRODUCTION

Machine learning of models and intelligent data analysis with the use of these models are becoming more and more relevant and widespread. Areas based on the use of large language models and services such as ChatGPT from OpenAI are developing especially rapidly. However, the effectiveness of solving a problem depends on its correct formulation, chosen information technologies and model architecture, methods of data analysis and visualization as well as prediction results using these models. ChatGPT and its analogs solve many problems, but not all of them – you still need a data scientist who will correctly set the problem, formulate a request, and verify the answer (ChatGPT often "hallucinates", i.e. synthesizes a random, and not a correct answer), will use it to solve the problem. Solving problems became much easier, processes became more intelligent and not routine. This textbook is intended to provide a comprehensive introduction to the above issues presenting suggestions, recommendations and peculiarities useful for the application of the latest technologies in system analysis.

The purpose of this textbook is to acquaint undergraduates and graduate students with the basic knowledge and skills in Machine Learning, Intelligent Data Analysis, the Internet of Things, and the Artificial Intelligence of Things necessary for solving real problems of varying complexity, as well as to help find the optimal choice of information technologies and services for automating this process.

The material of the textbook can also be useful for students of the second higher education in systems analysis, information systems and technologies, and students of other specialties who do not have enough basic knowledge and skills in this field. The textbook will also be interesting and useful for students who study the specialties 124 "System Analysis" and 126 "Information Systems and Technologies" of the II and III levels when learning the following subjects: "Internet of things and intelligent data analysis", "Information technologies of monitoring and analysis of the state of complex systems", "Information technologies of monitoring and data analysis", "Information intelligent technologies", "System analysis", "Smart Technologies" as well as for students undergoing the industrial and pre-diploma practice, the pedagogical practice for graduate students. It can also be useful for students of other areas: management, finance, construction, cyber security, bio-, electrical and mechanical engineering, agriculture, biology, medicine, education, etc. The textbook provides many examples of tasks in these areas.

The textbook contains the following chapters:

1. General Aspects of Setting and Solving Tasks in Data Science and Intelligent Data Analysis.
2. Data Preprocessing and Exploratory Data Analysis.
3. Feature Engineering.

4. Training and Tuning of Machine Learning Models.
5. Intelligent Data Analysis.
6. Internet of Things.

Machine learning models, the training of which is described in chapters 1-4, are not an “end in themselves”. As a rule, they are built to solve applied problems using intelligent data analysis technologies that is using special technologies that take into account the specifics of data, and using pre-trained machine learning models. Therefore, although the material presented in chapters 1-4 allows you to solve problems of arbitrary complexity, it is more effective to use them together with the material of chapters 5, 6: images and videos (section 5.1), natural language texts (sections 5.2, 5.3), time series (section 5.4), IoT systems (Chapter 6). All programs in this tutorial are in Python. Appendix A lists the basic Python commands, data types, and basic Python libraries that you should know as a minimum to be able to read later chapters. There are practical exercises and examples of their solutions at the end of chapters 1, 2, 4, and 5. There are generalized topics given as possible topics for practical and laboratory tasks in the all chapters 1-6.

Appendices B-G provide auxiliary resources for all chapters 1-6: Appendix B for building your dataset in the Kaggle environment, Appendix C – examples of setting problems from Machine Learning and Intelligent Data Analysis, Appendix D – lists of IT infrastructure, Appendix E presents libraries and methods for automatic Exploratory Data Analysis: Pandas_profiling, AutoViz, SweetViz with many examples, Appendix F highlights libraries SHAP, LIME for the model interpretation with many examples, Appendix – with Neural Network architectures.

When presenting the material, it will be taken into account that it is now easy to find the content and parameters of any command, operator, function, or library in ChatGPT, therefore such material will be presented minimally – instead, at the end of each chapter, infographics with the names of such commands or libraries which should be mastered independently will be provided. Each chapter will provide reviews of examples for solving real problems, taking into account the authors' many years of experience, and problems of a training nature from the Kaggle, platform of data scientists, which as of May 2024 already contains more than 21 million accounts, 60 million notebooks and 5 million datasets from data scientists around the world (see Kaggle metadata).

In addition, there will be links to ready-made Python programming code with an illustration of the material presented on the example of solving such real or training tasks. Each chapter will end with a list of test questions.

The authors of the textbook have extensive experience in solving real problems in the fields of medicine and biology, ecology and meteorology, economics and trade, electronics and the Internet of Things, energy and electromechanics, agriculture, management, finance, construction, cyber security, bioengineering, education, transport and control of drones, recognition of data from remote sensing of the Earth, including aerial photography, etc.

using artificial intelligence technologies, machine learning, and intelligent data analysis technologies.

In addition to real tasks, the authors have significant achievements in the ratings of the Kaggle platform. Professor Vitalii Mokin has the title of Kaggle Notebooks Grandmaster (the first in Ukraine to receive this title, at that time there were only about 60 of them in the world), and he reached the 10th place in the Kaggle world rating for notebook development! His profile has more than 850 Python notebooks, incl. about 200 public ones and these are the ones that were used as examples in this textbook.

The main text was written by Professor Vitalii B. Mokin, but Mykola G. Pradivliannyi did a creative translation of most of the material into English and selected examples for various tasks, as well as summarized material from various web resources. Approximately 80% of the material – Vitalii B. Mokin, 20% – Mykola G. Pradivliannyi.

Author's Kaggle Notebooks are dedicated to solving all the types of problems discussed in this textbook. Many drawings and explanations were taken from them. Readers can copy them and adapt them to their tasks using this textbook. For many of these notebooks, there are also Ukrainian-language lectures by Vitalii B. Mokin on his YouTube channel "AI-ML-DS Training course on Python".

The team of authors would like to thank such data scientists for their useful advice and comments, which allowed to significantly improve the level of the material of the textbook:

- Doctor of Technical Sciences, Professor of the Department of System Analysis and Information Technologies (SAIT) of VNTU, Prof. Oleksandr Mokin for valuable advice on various aspects;
- David Groozman for his advice on topics of Data Science, Artificial intelligence and Data Engineering;
- Boris Sorochkin for his advice on development methods, programming, and systems analysis;
- Mykhailo Dratovanyi for his help in designing the textbook [1], the material from which was used as the basis for this textbook;
- Dmytro Shmundiak for exceptional quality consultations when writing the material on the analysis of anomalies in time series in chapter 5;
- Kaggle Grandmaster Yaroslav Isaienkov for sharing valuable experience and expertise when writing the material on the GANs in chapter 5;
- Kaggle Grandmaster Leonid Kulyk for sharing valuable experience and expertise when writing the material on the Diffusion Models in chapter 5;
- Volodymyr Kopniak for help in writing the material on the analysis of time series and the text on the heteroscedasticity of series in chapter 5;
- PhD Arsen Losenko for his materials on forecasting time series using Prophet (Facebook Prophet) in chapter 5;
- Kostiantyn Bondalietov for his advice in web scrapping and NLP research in chapter 5;

- Borys Varer and Serhii Levitskiy for their advice on chat-bots and LLM models in chapter 5;

- Dmytro Honcharenko for the assistance in writing Chapter 6 on the Internet of Things and in developing practical tasks for it.

This is the first edition of the textbook. Please feel free to send comments, remarks, and recommendations for its improvement. The e-mail of the Department of SAIT of VNTU sait@vntu.edu.ua.

1 GENERAL ASPECTS OF SETTING AND SOLVING TASKS IN DATA SCIENCE AND INTELLIGENT DATA ANALYSIS

1.1. Basic concepts of data science, machine learning, artificial intelligence, and intelligent data analysis

In all spheres of our life it is necessary to be able to process information optimally and correctly: collect it, analyze it, forecast it, use it to make informed decisions, etc. The entire complex of approaches, techniques, methods and tools for solving the above problems is called "Data Science". The main components of DS are the following [1]:

- *Data engineering*, which includes methods and means of data collection and management, their preliminary analysis and processing for use in machine learning tasks;
- *Machine learning* – a complex of techniques, methods and technologies for solving applied problems, which is preceded by "training" models on computers (machines) on certain data;
- *Artificial intelligence* (or artificial intelligence technologies) is a set of techniques, methods and machine learning technologies that simulate various aspects of human cognition, such as learning, problem solving, reasoning, perception and decision-making;
- «*Intelligent Data Analysis*» – solving applied problems of analyzing various data using a set of techniques, methods and intelligent technologies.

Let's define what intelligent models are. There are many definitions. We suggest the following: an intelligent model is an information model built for the efficient solution of the analytical problem, capable of learning from experience and generalizing knowledge to process new data and scenarios. This definition reflects the main difference of this kind of models – they allow to predict data, events, generate new knowledge and information with high accuracy. If an information system or technology makes a simple comparison of the input data with a database of samples (fingerprints, DNA, faces, cadastral information, etc.), then it is not intelligent, but if it can predict how a specific face will change in 10 years or how it looked 10 years earlier, then this is already an evidence of its intellectuality within this definition.

However, it is worth noting the following important features [1]:

1. Machine learning (ML) focuses primarily on training machine models with information from datasets, while data engineering (DE) focuses on creating and managing these datasets. That is why they are called so. But some publishers and employers often include ML in DE or DE in ML when posting data engineer or ML developer jobs.
2. Intelligent data analysis (IDA), as a rule, consists in the use of specially developed information technologies and pre-trained intelligent models for solving applied analytical problems. Although, some authors often consider

ML/AI as one of the initial stages of IDA or IDA as one of the final stages of ML or AI.

This textbook suggests the following way of structuring these terms, concepts, methods and technologies (Fig. 1.1) [1]:

1. Collection and processing of data is carried out, including big data (Big Data), from IoT or information systems and the formation of datasets for data analysis – data engineering (DE), which can be considered as a section of data science (DS) or as an independent section.
2. Based on the results of DE, Exploratory Data Analysis (EDA) is carried out, which involves the applied application of certain sections of mathematics, including statistics, to data. This analysis may also involve trial building of models to analyze certain patterns in the features. Moreover, such models can also be multilayers neural networks (DL).
3. Based on the results of EDA, machine learning (ML) models, deep learning (DL) and artificial intelligence (AI) models are carried out.
4. Pretrained ML/DL/AI models are used for intelligent data analysis (numbers, text, images, video, speech, sounds) (IDA) aimed at solving various applied analytical problems.

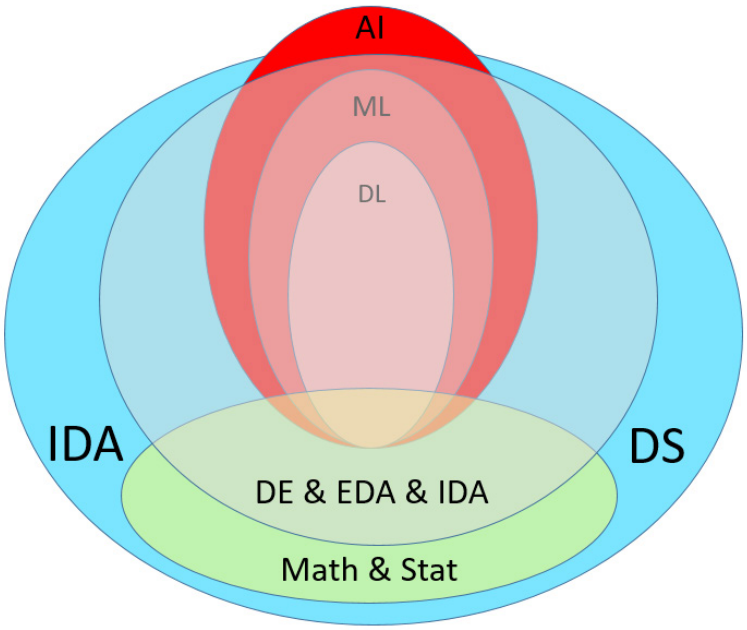


Figure 1.1 – Basic concepts and their combination in the field of ML/DL/AI/DE/DS/IDA

They are all united by data science (DS), including DE, although they all share some aspects of the software-hardware plan that DS does not, so they only overlap.

DS is a more general concept than mathematics, including statistics, and DE, EDA, IDA, and therefore it covers them completely.

Application software and integrated development environments are used to automate all these operations – IDE.

This structure corresponds to the division of tasks within the IT companies into data engineers and data scientists. In turn, data scientists either create new ML/DL/AI models, or, more often, use ready-made models, but develop technologies for their application to solve new problems. The most valuable nowadays is the ability to use IDA itself, but this is impossible qualitatively without understanding the fundamentals of ML/DL/AI model building, so it is important to master all these aspects of DS.

Regarding Data Science, a slightly different scheme is common in the literature. With some of our improvements and a combination with Internet of Things (IoT) and IDA, the corresponding scheme is shown in Fig. 1.2.

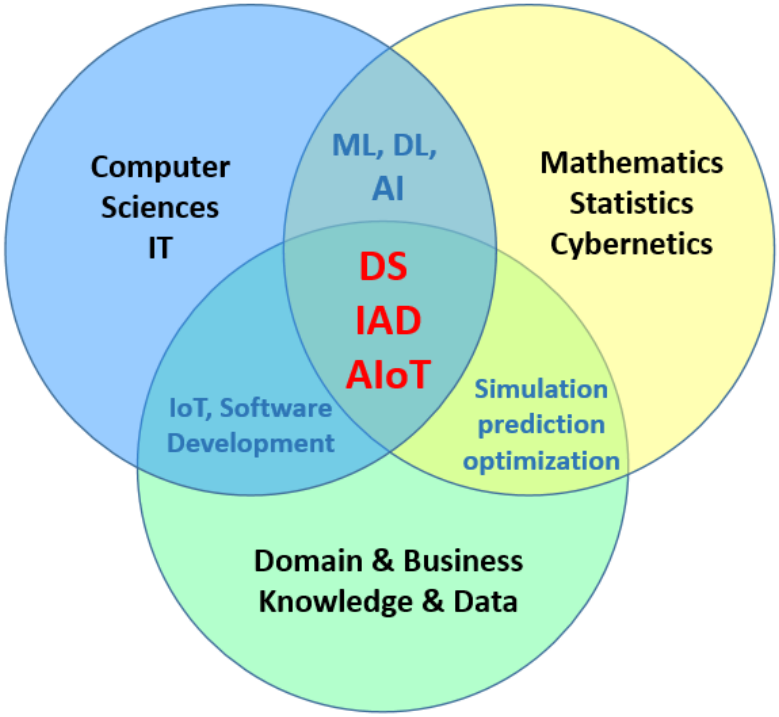


Figure 1.2 – Basic concepts and their combination in the field of Data Science, IoT and IDA

Fig. 1.2 shows that informatics (IT and Computer Sciences) in combination with knowledge and data about the subject area (which is commonly called as "Domain Knowledge & Data") and knowledge and data about performance indicators, user or customer requirements, business-strategies and limitations, etc. (Business Knowledge & Data) form "IoT Software Development". And the combination of mathematics, including statistics, and cybernetics, including theory of control and optimization of systems, with this knowledge and data it is traditional research in the field of mathematical simulation, prediction and optimization of processes and systems. The combination of IT, including information and information-measurement systems, as well as computer sciences, with a mathematical apparatus, made it possible to create machine learning and artificial intelligence (ML & AI). And at the junction of all of them, there were formed:

- Data Science (DS) as a theoretical complex;
- Intelligent Data Analysis (IDA) as an applied application of DS for various tasks and tasks;
- Internet of Things with Artificial Intelligence (AI&IoT = AIoT) as a full-cycle intelligent information system: observation, storage, processing, intelligent analysis and decision support (Industry 4.0).

It is important to realize that in applied terms, data science, machine learning, artificial intelligence, and intelligent technologies are, first of all, information technologies that process information from input to output according to certain algorithms aimed at increasing the amount of information I, primarily – for the better systematization and formalization, detection and prediction of new knowledge and regularities. These algorithms, depending on the uncertainty of the input data, the structure of the models or the branching and multivariation of the algorithm, have different complexity. Therefore, for a better understanding of the material, the manual will use the appropriate infographics, which will demonstrate how each block of information technologies can be located in such a coordinate system *S(I)* (Fig. 1.3).

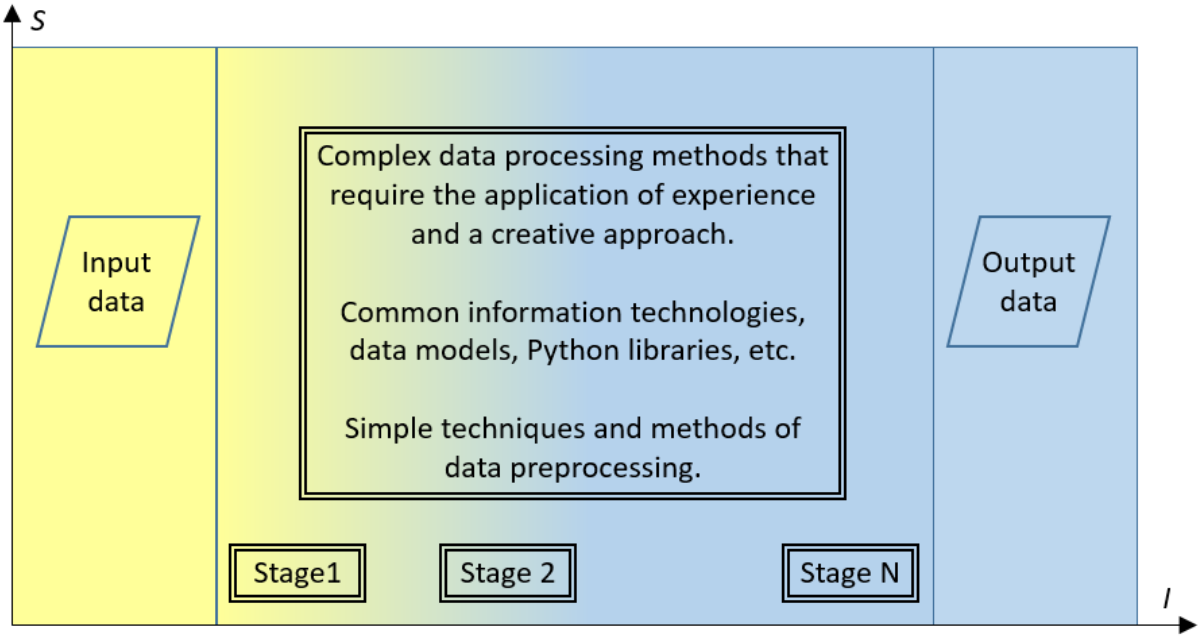


Figure 1.3 – Infographics for visualization of information technologies for the transformation of input data into output, depending on the complexity *S* of the algorithm and the amount of information they add

If there are many blocks on the diagram, then for a better understanding of their sequence along the abscissa axis, the projection of these blocks onto it can be added in the form of green vertical lines from the center of the blocks to this axis.

Each section will present examples of setting and solving both real problems and training problems in the Kaggle platform from the relevant topic. The textbook is focused on the use of the Python programming language (see Appendix A with some infographics and links to documentation and problem sets,

which will allow you to speed up its learning, if you already have knowledge of at least some other programming language and the basics of algorithmization).

1.2 Setting the task of data analysis. Search for information on it. Building a dataset, its division into training, validation, and test datasets

The vast majority of data analysis problems fall into 2 classes [1]:

1. Tasks of exploratory data analysis, where the data itself is enough to analyze patterns, dependencies, primary statistical analysis, etc. without using complex models.
2. Data analysis and prediction tasks that require the use of complex models to make high-precision predictions, after which either the analysis of the predictions made is carried out, or the model itself is analyzed, whose good predictive function has proven its adequacy.

The tasks of the second class, in turn, are divided into problems that can be effectively solved using pre-trained models, and problems that require the construction of a new model.

This textbook is dedicated to the most complex case, when it is necessary to carry out both exploratory data analysis and intelligent data analysis, for which it is still necessary to build models based on datasets that have yet to be created from data that has yet to be found.

For the first class, the task is, as a rule, a requirement to carry out an *intelligence analysis of data* to detect the presence and identification of patterns, including statistical; detection of wrong, anomalous and problematic data; detection of connections between data, their grouping and clustering; visualization of the obtained conclusions in an easy-to-understand form. There may be broader task statements.

Most tasks of the second class are optimization tasks. It is worth distinguishing between the tasks of machine learning of intelligent models and tasks of IAD. Even if it is not explicitly formulated, optimization can be carried out in functions of libraries that are used to improve the accuracy of model training.

Classical optimization tasks in systems theory are usually formulated as follows: for given input data, control variables, under the influence of controlled and uncontrolled disturbances, ensure the optimum of the optimization criterion under certain restrictions.

In machine learning tasks, as a rule, the input data and the so-called "target" are distinguished. There can be many targets, or all data can be a target in turn. Often, all this data is located in one table. The optimization criterion is a numerical indicator (metric or error). There may be no restrictions. Basically, their role is to limit the use of only values from the dataset and prohibit the use of others. But there may also be certain physical restrictions, for example, the prohibition of fractional values (in the case of using regression models to predict or forecast the number of objects, all values should be rounded to a whole and only then determine the metric – see the example of determining the number of

survivors on the "Titanic" in a notebook) or the number of patients with coronavirus cannot be negative, etc.

So, the classic task of *machine learning of an intelligent model* is usually formulated as follows: for a given dataset (data tables, text files, audio, video files or images), build and train an intelligent model that will provide the optimum (maximum or minimum) of the given metric (criteria) for a given target feature(s). Additional restrictions may be applied, but are not required.

The tasks of intelligent analysis can be the optimization of the use of various models, methods and technologies to identify complex relationships and regularities in various datasets, including those with numerical, textual, graphical static and/or dynamic information, with the aim of forming predictions or obtaining insights for effective decision-making and problem solving in various fields. There may be other (more general or more narrow) statements of IDA tasks.

There is no single algorithm for setting and solving all such tasks, although below in this section there is a generalized algorithm and recommendations for the most complex options for setting the problem, but unfortunately, it do not cover all possible options.

Algorithms may differ in the stage at which task solving begins. Competitions and training examples already have datasets and task statements that need to be solved. In real problems [1]:

Option 1. The data set may be a known one. This may be, for example, data from medical tests and only the desire to improve the treatment efficiency or evaluate the spread of the disease in the country more accurately. But it still remains unknown how to do it and what indicators to use. That means that the exact formulations of the tasks need to be done independently.

Option 2. The task statement may be known. For example, this may be to increase the accuracy of the daily forecast of the officially published number of coronavirus patients in the country, but what factors to take into account remains unknown, i.e. the dataset still needs to be created, and after creating, in is necessary to clarify the task statement, since not all data usually can be found, according to the requirements.

Option 2 occurs more often and is the most generalized, and option 1 is its simplified case. Therefore, we will pay more attention to option 2.

Ready-made datasets can be searched on web platforms (GitHub, Kaggle, Hugging Face, etc., where there are hundreds of thousands of them and they are publicly available). If you could not find the required dataset, or if it is found, but it does not look the way you want, you should create your own dataset. It often makes sense to create your own dataset, even when there are different existing datasets, but you still need to apply many pre-processing operations to them each time. Then you apply them, select everything you need, save it once in its optimal form, and then only use it in the future.

When collecting data, it is very important to consider those features that impact or potentially affect the target feature and ignore those that definitely do

not. For example, there is a web page with the so-called "False Correlations". These are regularities between which there is no physical or semantic connection, but statistical analysis shows that the correlation is present (Fig. 1.4).

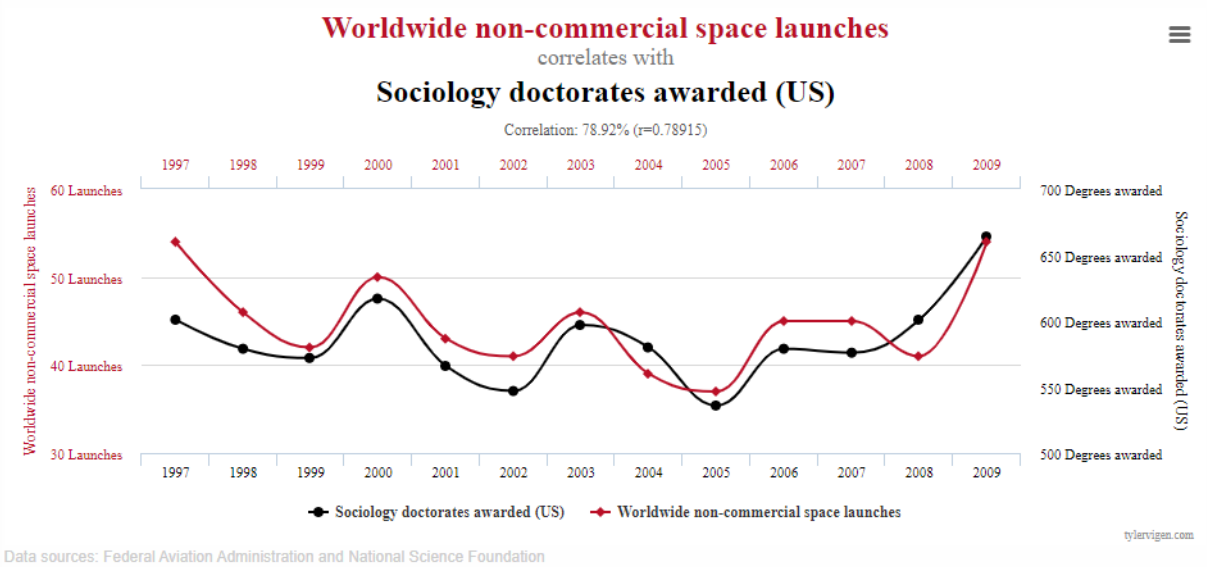


Figure 1.4 – 0.79 Correlation of Non-Commercial Space Launches and Sociology PhDs defended on «False Correlations»

Appendix B provides an algorithm and recommendations for building your own dataset in the Kaggle environment. Also, you can create datasets as files on your local computer (for use in Python programs on the same computer), on GitHub, on your Google Drive, in the Amazon S3 service, etc.

An important aspect in machine learning tasks is the formation of training, validation and test datasets. Models are trained on the training dataset. It is also called "training". Optimal model is chosen on the validation dataset. And the testing one is been used to apply only the optimal model.

In real tasks, all data are usually divided into training and validation or into training and testing, but the testing acts as the validation one. Usually, the training dataset contains from 70% to 90% of the total data, and the validation dataset from 10% to 30%. The most popular options are: 75/25% or 80/20%. The 90/10% option is used if the total amount of data is too small and the model cannot learn using 70-85%, and the option with 70/30% (or even 60/40%) when the data is very uniform and the model gives a super high accuracy of 100% matches, even using 70/30%.

Usually, the validation data are selected from the general randomly, but for time series, when it is necessary to predict the future, then the last values of the series are validating, so that the optimal model most accurately reproduces the data located in time immediately before the test ones.

For classic machine learning tasks, cross-validation is usually used, when only the percentage of data that should be selected for training and validation, and the number of cross-validations (abbreviated as "cv") and other parameters (percentage of overlap, etc.) are set (Fig. 1.5).

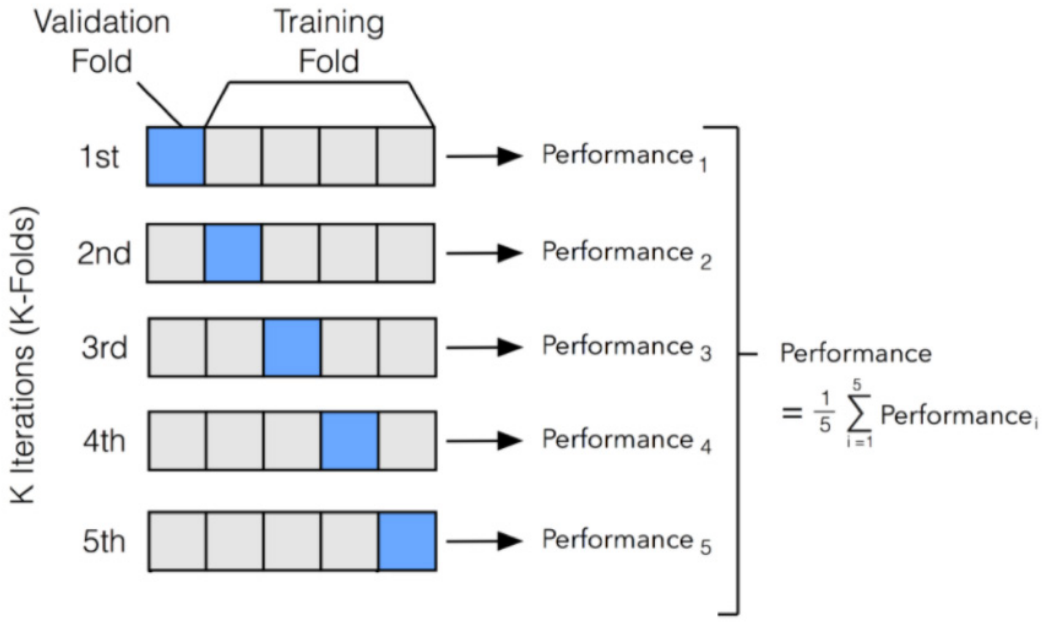


Figure 1.5 – Cross-validation of training and validation data with cv = 5, without data overlap (from GitHub)

That is, all data is divided into blocks, which one by one become a validation dataset, and the rest are training data. If data overlap is specified, then each value can be multiple times in the validation dataset. In some cases this method allows to significantly increase the accuracy of the model.

It is also important that the validation and test data have the same ranges of feature values as in the training data, otherwise the model may be underfitted. For example, if the training dataset contains only used cars with a value of $10,000 to $20,000, and the validation and test dataset contains $300,000 or more up to 2 million dollars, then such a model is unlikely to be effective. For this, it is important to analyze and compare the distribution laws of these datasets, which will be discussed in more detail in the next Chap. 2.

- A dataset, as a separate set of files on a disk or in a cloud environment, is not always necessary if the data can be obtained through an Application Program Interface (API) or simply from web resources. To do this, it is enough to make a request in the program, download the data and start processing. For example, you can:

- download a web page, extract the text from it and analyze it (this will be discussed in Chap. 5);

- download the cryptocurrency rate via API from the cryptocurrency exchange (this will be discussed in Chap. 5);

- in Kaggle, you can use another notebook as a dataset in your notebook (Python program) and tighten its results directly, without the separate saving to the cloud as a dataset.

However, using a dataset is more convenient, because you can store in one place different versions of the data, general, training, validation and test

parts of the dataset, versions in different languages and with different coding, their description, comments on them, programs for their processing, etc.

1.3 Definition of the target feature, types of tasks and metrics of machine learning. Clarifying the statement of the tasks

After the dataset is formed, it is necessary to clarify the formulation of the task or tasks that can be solved on its basis.

To clarify the statement of the tasks, it is important that the customer (he can be the researcher himself) first describe the tasks, as he sees it, in 3-5 sentences, add a dataset or datasets with data, briefly describe the desired expected results. With this in mind, one should clearly define the type of feature or features to be targeted, select a metric or metrics that can be used for that type, and determine if and which constraints there are. Therefore, you should get answers to the following questions [1]:

1. What is a target feature and what type is it?

Incorrect understanding of what a target is or a misunderstanding of its type or nature leads to the impossibility of correctly and qualitatively solution of the task. This question, in turn, is divided into a number of sub-questions.

1.1. What are the permissible values of the target characteristic?

It is necessary to find or obtain examples of the values of the target characteristic, to analyze its dimensions, ideally to find out the full set of possible values (if there are few of them) or – theoretically achievable statistical indicators (minimum, maximum, average value, etc.), if there may be many values .

1.2. Is the target feature the only one, can it be reduced to one, or is it a task with many target features?

The most common and simple option is the option with one target feature or when the task can be reduced to a number of separate tasks, where each time there will be a different single target feature and each such task must be solved separately. A more difficult option are tasks where several interrelated target features must be determined at once and cannot be divided into separate ones – for such tasks, special models should be used, for example, neural networks with many outputs.

2. Is the task a classification or regression?

Classification tasks are machine learning tasks, where a limited number of classes are predicted, each of which has a sufficient amount of data to study the patterns of their formation. Accordingly, regressive tasks are tasks where values of the target feature are predicted, for studying the regularities of each range of data, there may not be enough input data, and then they are simply predicted by the model, i.e. the identified dependencies are spread ("regressed") to other values. Note, that regression models should not be confused with regression tasks – this is a different classification! For example, a logistic regression model is often used to solve classification tasks.

In practice, tasks where the target feature is a fractional number are immediately classified as regression tasks, and tasks where it is an integer and this number is 2-20 or less are classification tasks. In other options, a more thorough study is required. Tasks with 100 classes and a billion data can be considered as a classification problem, and if there are only 200 pieces of data, then it is better to solve it as a regression tasks, otherwise the model will not learn enough.

Tasks with the so-called "binary target" are most effectively solved, that is, when the target attribute takes only 2 values (0 or 1, True or False, sick or healthy, whether a passenger on the Titanic survived or died, etc.). A classic example is forecasting not the values themselves, (for example, sales volumes, currency exchange rates, weather parameters, etc.), but whether there will be an increase in the next step (value 1) or not (0)?

The importance of classifying the problem by the target feature will become clear in chapter 4, where it is noted that most names of machine learning models consist of two parts, the first of which is the actual name of the model, and the second is the type of problem ("Classifier" or "Regressor") , for example: RandomForestClassifier and RandomForestRegressor.

3. This is the task of analysis, prediction or forecasting?

There are tasks where you just need to analyze the available data and the result is various valuable conclusions and recommendations. And there are tasks where it is necessary to predict the results. However, there are tasks where no prediction is carried out, but only *analysis*, so it makes sense to separate the "analysis" task type as a separate one.

In general, forecasting tasks are a subspecies of predicting tasks, but in *forecasting* problems it is important to consider time dependence and use special time series models. *Prediction* tasks are more common, but in them, on the contrary, the feature of each moment of time, if it is present, must be removed, otherwise the model, as they say, will be retrained or "overfitted". It will diligently predict facts that occurred strictly at those points in time that were in the training dataset.

It will not generalize and analyze the nature of the phenomenon itself and will not be able to correctly work with test data at previously unobserved moments of time. For example, we accidentally receive a photo and have to determine whether it is a cat or a dog – this is a prediction task and it should not take into account the time of receiving the photo. And if we forecast the exchange rate on the market, the environmental quality indicator, the number of cars at the traffic light, the amount of electricity produced by the solar panel with minute averaging, the expected harvest in the field, etc., then this is a forecasting task and time must be taken into account. Time series analysis and forecasting will be discussed in more detail in subsection 5.4, and other materials will focus mainly on predicting problems.

4. In the development of the previous question: what is the type of data formalization and what is the type of the given problem? The algorithm for solv-

ing the given tasks will depend significantly on the answers to these questions, that is, the choice among the following options:

- *data tables* that are not sequences of data over time (not time series), and classic classification or regression problems of multivariate prediction, when one or more target features (columns of the table) should be determined for certain input features – the universal models of this type are more detailed in Chapter 4;

- *image or video:* tasks of analysis, recognition and/or generation of images – tasks in which it is necessary to determine to which class the given images (or video as a sequence of images) belong, or to find specific objects or to analyze them according to other criteria – this is detailed in chapter 5;

- *text:* tasks of analysis, processing and/or generation of natural language text – tasks in which natural language text should be analyzed, classified or otherwise processed – this is explained in more detail in chapter 5;

- *time series of data:* tasks (as a rule, regression) in which a given target characteristic should be predicted as a result of a change in another or many others, or the same one, but – at earlier moments in time – this is explained in chapter 5.

Of course, this classification does not cover all the variety of data and problem statements. There is also *speech* as sound signals; sounds that are not speech signals; game algorithms (chess, checkers, Go, etc.) and others. But for these other types of tasks, a significant amount of special knowledge in the subject area, in the field of data engineering, programming, signal theory, etc., is necessary, this should be the material of a separate textbook for each of them. More briefly, this question sounds like this: tables, images, videos, text, time series or other?

Now we will consider the types of metrics for checking the optimality of the target feature.

A metric in machine learning is an optimization criterion that should be given an optimal value at various stages of data processing (clustering, building models, data classification, etc.).

For prediction tasks, the following are the most popular (for more details and formulas, see subsection 1.5):

- for classification tasks: "accuracy_score", ROC-AUC, if the dataset is unbalanced, i.e. some class significantly prevails, then – "F1_score" or "F2_score";

- for regression problems: «r2_score», «MAE», «MAPE», «SMAPE»;

- for clustering problems: «siluette_score» and other.

There are many others in the sklearn library – see in the documentation and in its description «User Guide».

In the following sections, other metrics will be mentioned in the context of the tasks that will be described there.

After determining the target characteristic, the type of task (data type) and the metric, it is necessary to clarify the statement of the task. Appendix B pro-

vides examples of such problem statements, both real and from Kaggle prize competition.

Fig. 1.6 presents an infographic of the algorithm for setting the tasks in the *S(I)* coordinate system.

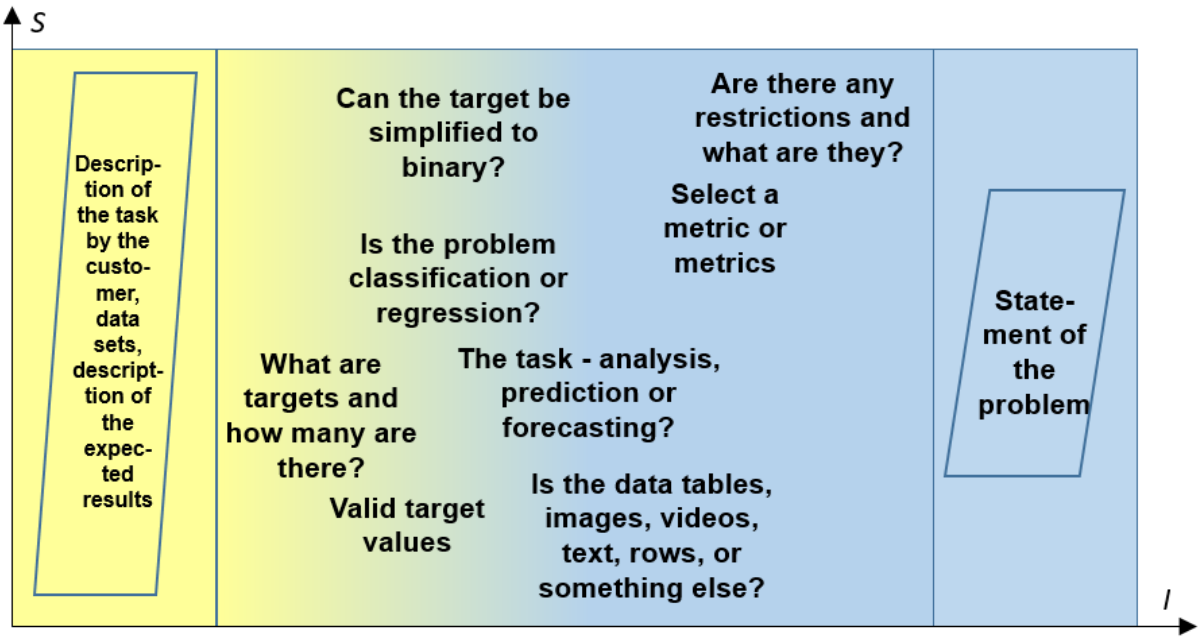


Figure 1.6 – Infographics of the algorithm for setting the tasks machine learning

1.4 Generalized algorithms for solving machine learning and IDA tasks and IT infrastructure for their implementation

To solve the tasks of machine learning of an intelligent model, in general, the following algorithm is used [1]:

1. Collect data. Build a dataset.
2. Perform data preprocessing and cleaning.
3. Carry out an intelligence analysis of the data and determine which models should be built next. If the result is satisfactory and it is clear which models should be built, then proceed to point 5, otherwise, to point 4. If item 4 fails to improve the data, then go to item 1 and find additional data.
4. Carry out a feature engineering. After that, repeat point 3, if the new features contain raw data, then go to point 2.
5. Choose the architecture of promising models. Carry out their tuning on the training dataset.
6. Carry out model diagnostics on the validation dataset and analyze outliers (anomalies). If, according to the results of diagnostics, it turned out that all models were over- or undertraining, then repeat point 5 with other tuning parameters. If it did not help after *N* attempts, then go to point 4, and build different diagrams of the features importance according to the constructed models and analyze these features. If one or more models meet the metric requirements, then go to point 7.

7. Choose an optimal model (perhaps an ensemble of models) that satisfies the requirements of the problem in terms of metrics and constraints.

8. Apply the optimal model and make predictions (forecasting), analyze emissions and detected regularities, ensure visualization and reproducibility of the obtained results.

Fig. 1.7 shows the infographics of this algorithm in the *S(I)* coordinate system.

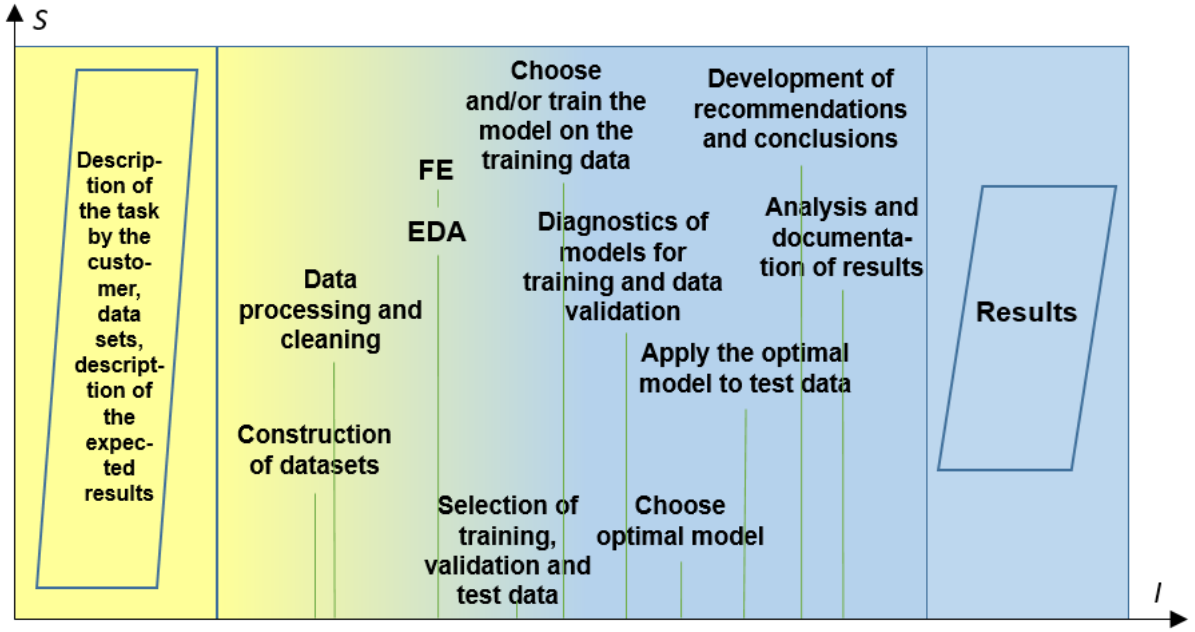


Figure 1.7 – Infographics of the generalized algorithm for solving the tasks of machine learning of an intelligent model

The following sections will be devoted to the stages of this algorithm.

As it was mentioned above, pre-trained machine learning models are used at the stage of intelligent data analysis (IDA). In addition, as a rule, the input is not raw data, but ready-made datasets to which IDA should be applied. In most cases, it makes sense to perform data preprocessing and cleaning, since models, as a rule, work with numerical data, and in IDA data can be images, videos, text, etc. The new stages are the transformation of the input data into a format acceptable to the optimal pretrained models and the post-processing of the results to obtain a higher quality analysis. Fig. 1.8 presents an infographics of the generalized algorithm for solving the IDA problems.

In general, it is rarely sufficient to use only pretrained models for high-quality IDA. More often, they are used only at certain stages, and then new machine learning models are built to generalize the results of their application. Therefore, to solve a real task, you should combine the algorithms of Fig. 1.7 and 1.8.

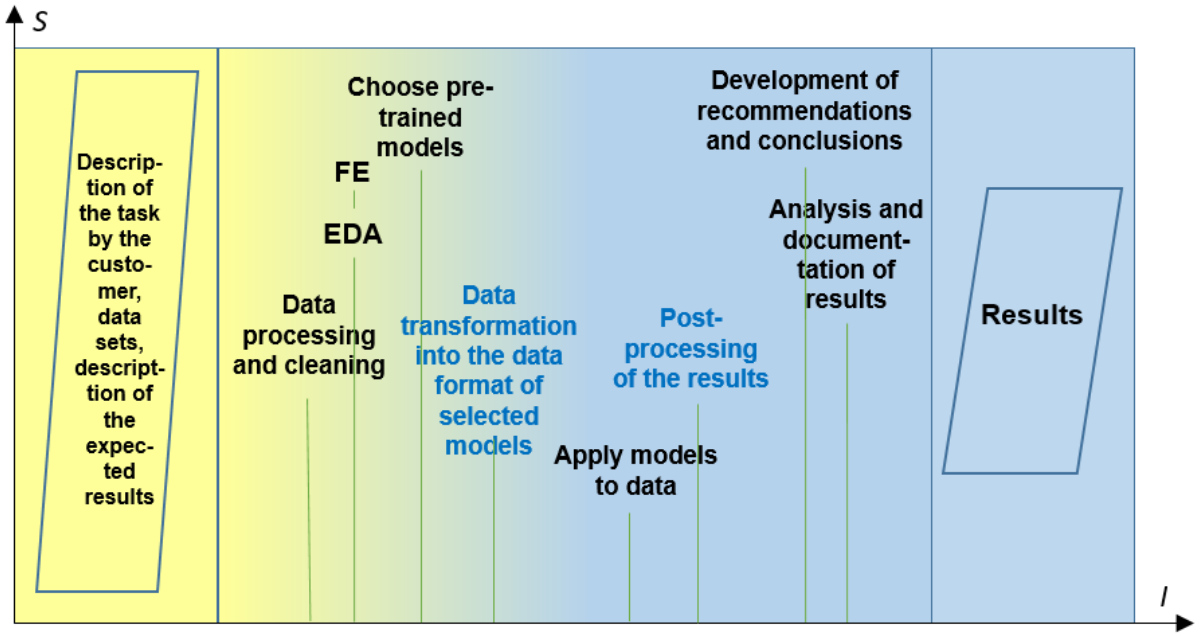


Figure 1.8 – Infographics of the generalized algorithm for solving the problem of intelligent analysis using pretrained machine learning models (intelligent models)

1.5 Examples of setting tasks of intelligent data analysis in applied areas

1.5.1. Cyber Security and Encryption

1. IEEE-CIS Fraud Detection. Can you detect fraud from customer transactions? (2019).

The IEEE-CIS Fraud Detection competition is devoted to developing machine learning models that can accurately detect fraudulent customer transactions. Using Vesta's real-world e-commerce dataset, which includes a variety of features from device type to product characteristics, create and benchmark models to enhance the accuracy of fraud detection. Your goal is to improve the system's efficacy, reducing false positives and fraud losses for businesses while enhancing the customer experience. Submissions will be evaluated based on the area under the ROC curve (AUC-ROC).

Data size: 6,4 Gb.

2. ALASKA2 Image Steganalysis. Detect secret data hidden within digital images (2020).

An efficient and reliable statistical method must be developed to detect secret data hidden in digital images (Fig. 1.9).

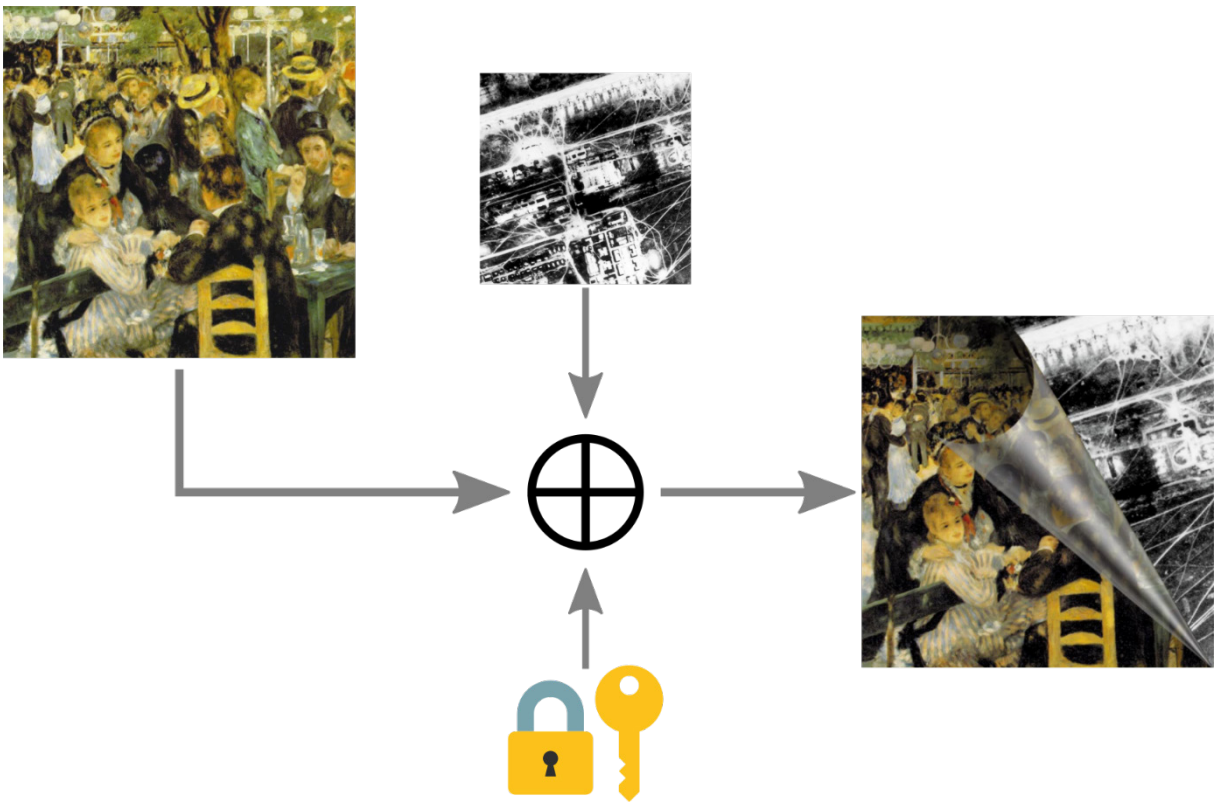


Figure 1.9 – The Kaggle competition "ALASKA2 Image Steganalysis. Detect secret data hidden within digital images"

These images, captured with up to 50 different cameras and processed in various ways, aim to reflect real-world conditions. Successful methods will use robust detection algorithms with minimal false positives, and submissions will be evaluated based on the weighted AUC to emphasize reliable detection with a low false alarm rate.

Data size: 32 Gb.

3. The Learning Agency Lab - PII Data Detection. Develop automated techniques to detect and remove PII from educational data (2024).

The Learning Agency Lab is hosting a competition to develop automated techniques for detecting and removing personally identifiable information (PII) from educational data. The goal is to create a model that can accurately identify PII in student writing, which will reduce the cost and increase the scalability of releasing educational datasets for research and tool development. Current methods, such as manual review and Named Entity Recognition (NER), are either too costly or insufficiently accurate. Submissions will be evaluated based on a classification metric that prioritizes recall over precision to ensure comprehensive PII detection.

Data size: 110 Mb.

4. TalkingData AdTracking Fraud Detection Challenge. Can you detect fraudulent click traffic for mobile app ads? (2018).

This contest aims to detect fraudulent click traffic for mobile app ads. Companies face significant volumes of fraudulent traffic, leading to misleading

data and financial losses. Participants are tasked with developing an algorithm to predict whether a user will download an app after clicking on a mobile ad, using a dataset of approximately 200 million clicks over four days. Submissions are evaluated based on the area under the ROC curve (AUC) between the predicted probabilities and the actual outcomes.

Data size: 11.3 Gb.

5. Microsoft Malware Prediction. Can you predict if a machine will soon be hit with malware? (2019).

The task involves predicting the probability of a Windows machine getting infected by various families of malware based on its properties, using a dataset provided by Microsoft. This dataset includes telemetry data and infection reports from Windows Defender. Each row represents a unique machine, with the ground truth labeled as "HasDetections". The objective is to use the training data to predict the "HasDetections" value for each machine in the test data. Submissions will be evaluated based on the area under the ROC curve between the predicted probabilities and the observed labels.

The training and test datasets of this contest contain more than 80 features and 8-9 million rows each (Fig. 1.10).

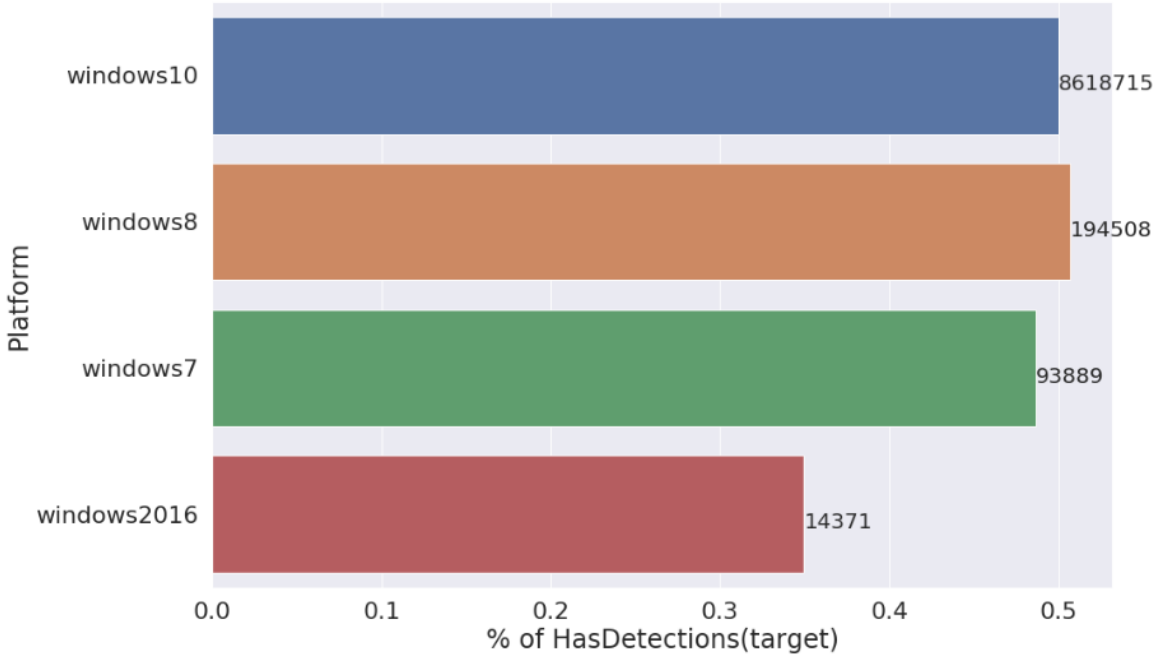


Figure 1.10 – Statistics for feature "Platform" (platform name) in the training dataset of the contest “Microsoft Malware Prediction” (from the notebook)

One of the authors of this manual (Prof. Vitalii Mokin) received a bronze medal for participating in this competition.

Data size: 8.5 Gb.

1.5.2. Electronics and Telecommunications

1. Google Smartphone Decimeter Challenge 2022. Improve high precision GNSS positioning and navigation accuracy on smartphones (2022).

The goal of this competition is to compute smartphones location down to the decimeter or even centimeter resolution. It is necessary to develop a model based on raw location measurements from Android smartphones collected in open sky and light urban roads using datasets collected by the host. It helps produce more accurate positions, bridging the connection between the geospatial information of finer human behavior and mobile internet with improved granularity (Fig. 1.11).

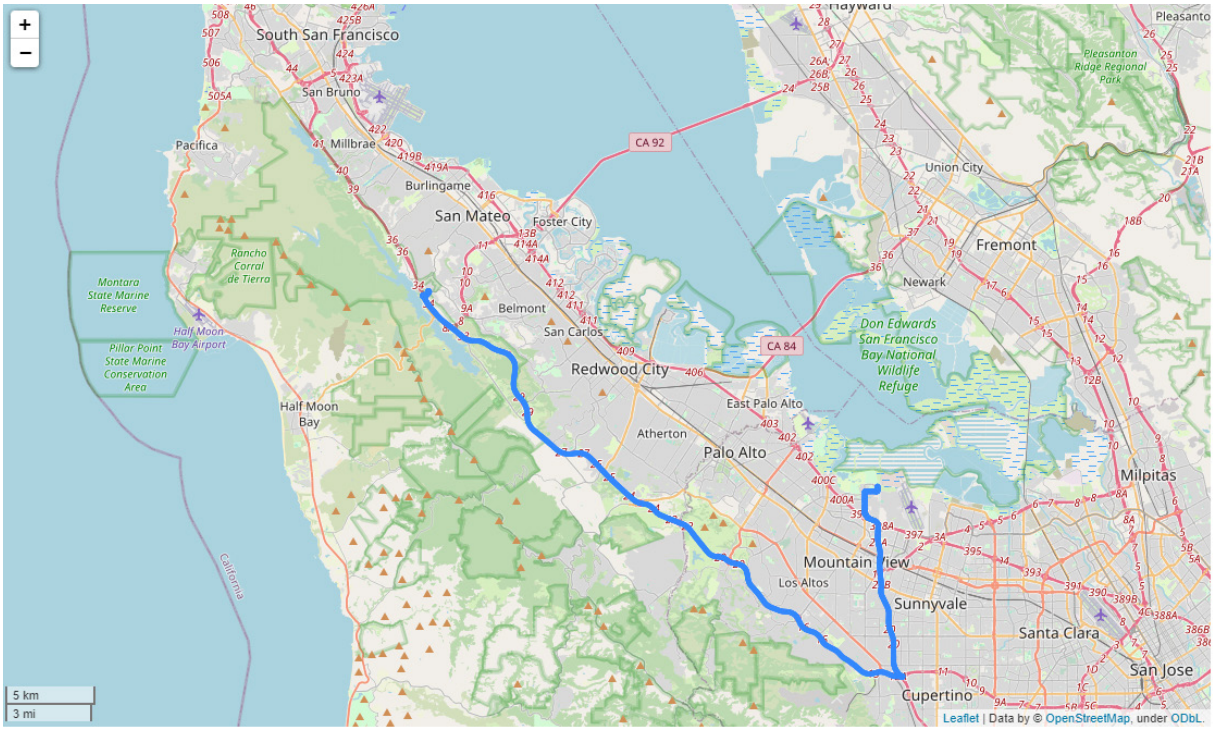


Figure 1.11 – Smartphone 2022: A look at the maps (from the notebook)

Data size: 22.9 Gb.

2. SETI Breakthrough Listen - E.T. Signal Search. Find extraterrestrial signals in data from deep space (2021).

Anomalous signals must be found in scans of Breakthrough Listen targets. Since there are no confirmed alien signals for training, simulated signals ("needles") are included in the data (Fig. 1.12).

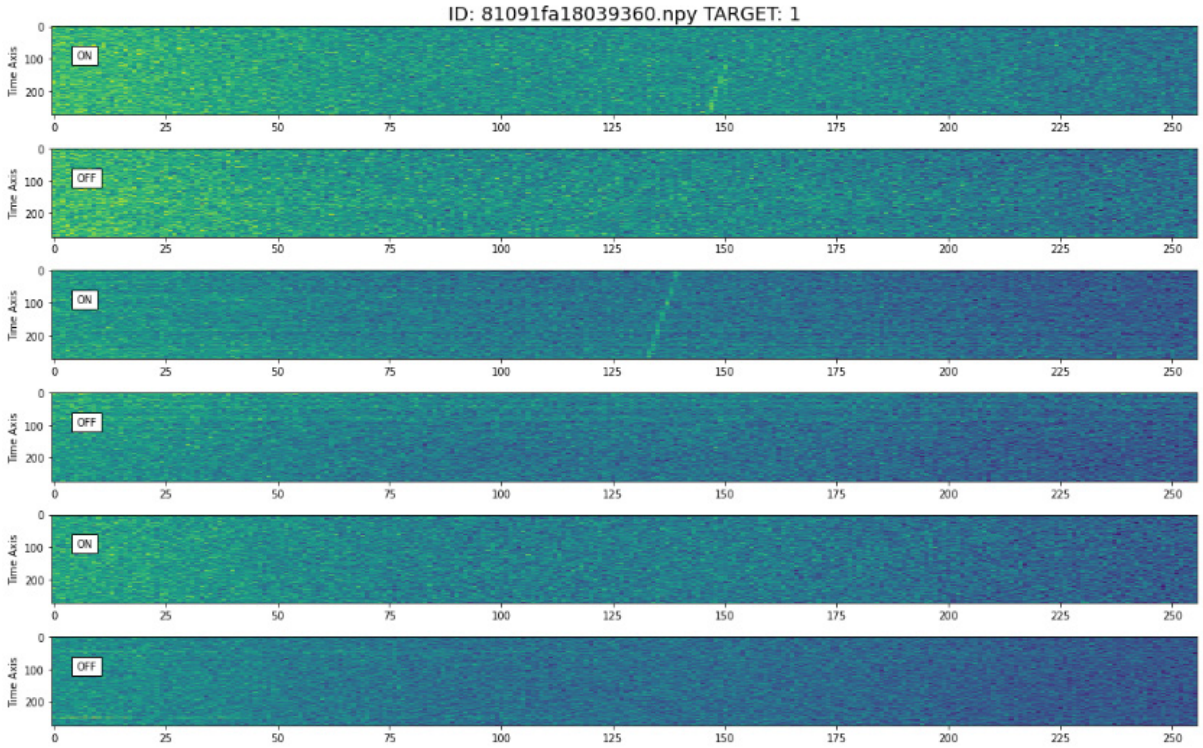


Figure 1.12 – The simulated signals ("needles") are included in the data from the Green Bank Telescope (from the notebook)

The data consist of two-dimensional arrays, and submissions are evaluated based on the area under the ROC curve between predicted probability and observed target.

Data size: 156 Gb.

3. Indoor Location & Navigation. Identify the position of a smartphone in a shopping mall (2021).

The task is to predict the indoor position of smartphones using real-time sensor data provided by XYZ10 and Microsoft Research. There is the dataset of nearly 30,000 traces from over 200 buildings to improve the accuracy of indoor positioning solutions, which is crucial for enhancing location-based apps and services (Fig. 1.13).

Figure 1.13 – Indoor Location & Navigation: identification the position of a smartphone in a shopping mall (the illustration from the notebook)

Submissions will be evaluated based on the mean position error.
Data size: 60 Gb.

1.5.3. Automation and Robotics, UAV, Transport, Mechanical Engineering

1. CVPR 2018 WAD Video Segmentation Challenge. Can you segment each objects within image frames captured by vehicles? (2018).

The task is devoted to image segmentation of movable objects, such as cars and pedestrians, at the instance level within image frames captured by vehicles (Fig. 1.14).

Using a unique dataset provided by Baidu Inc., participants will help improve computer vision algorithms for environmental perception in autonomous driving, with evaluation based on mean average precision (mAP) across various IoU thresholds. The challenge includes annotations for seven object types: car, motorcycle, bicycle, pedestrian, truck, bus, and tricycle.

Data size: 102.6 Gb.

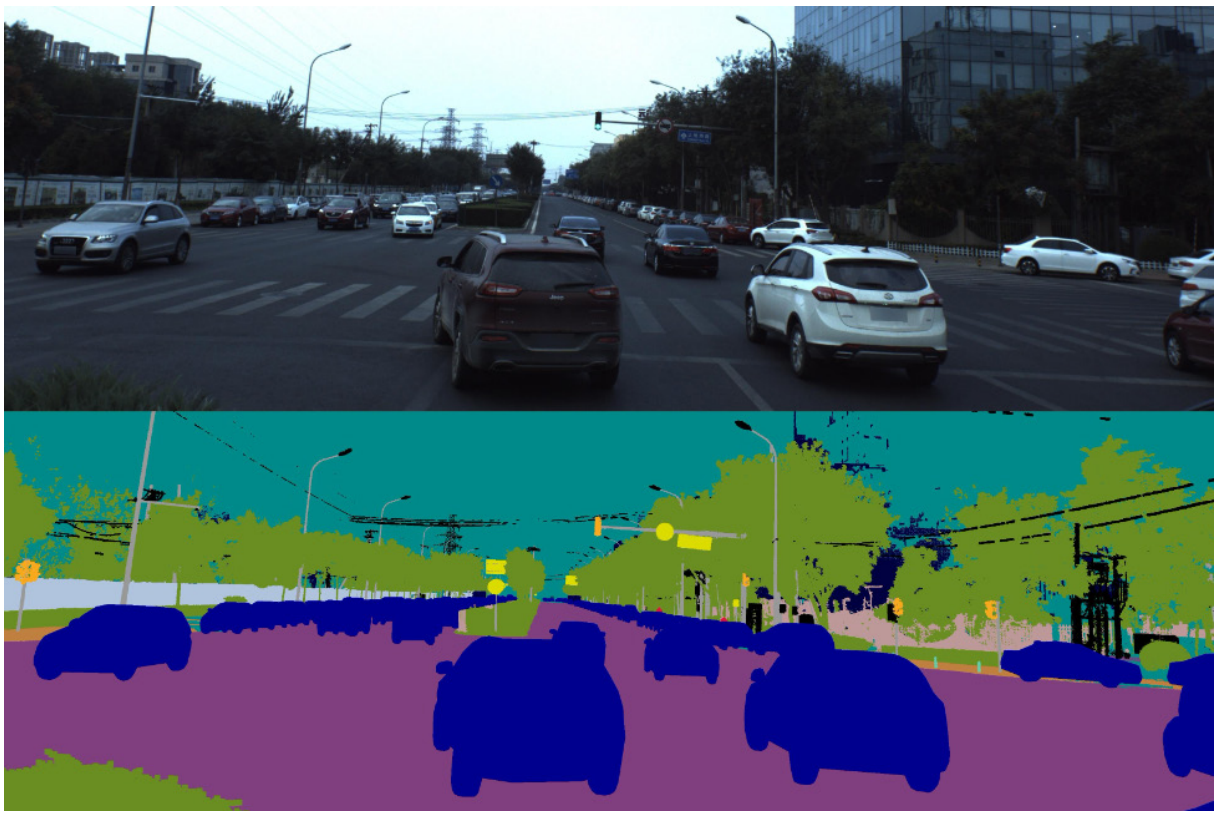


Figure 1.14 – The segmentation of movable objects within image frames captured by vehicles (from the contest)

2. ECML/PKDD 15: Taxi Trip Time Prediction (II). Predict the total travel time of taxi trips based on their initial partial trajectories (2015).

The task involves predicting the total travel time of taxi trips in Porto, Portugal, using initial partial trajectories. This prediction aims to enhance the efficiency of electronic taxi dispatching systems by helping dispatchers assign drivers to pick up requests more effectively. Submissions will be evaluated based on the Root Mean Squared Logarithmic Error (RMSLE), emphasizing accurate estimation of trip durations using trajectory data.

Data size: 0.5 Gb.

3. New York City Taxi Fare Prediction. Can you predict a rider's taxi fare? (2018).

Your goal is to predict the fare amount (including tolls) for a taxi ride in New York City based on the given pickup and dropoff locations, pickup time, and number of passengers. You need to build a model to minimize the RMSE in fare prediction. This competition was the first where one of the authors (Vitalii Mokin) participated in a Kaggle competition (Fig. 1.15).

Data size: 5.7 Gb.

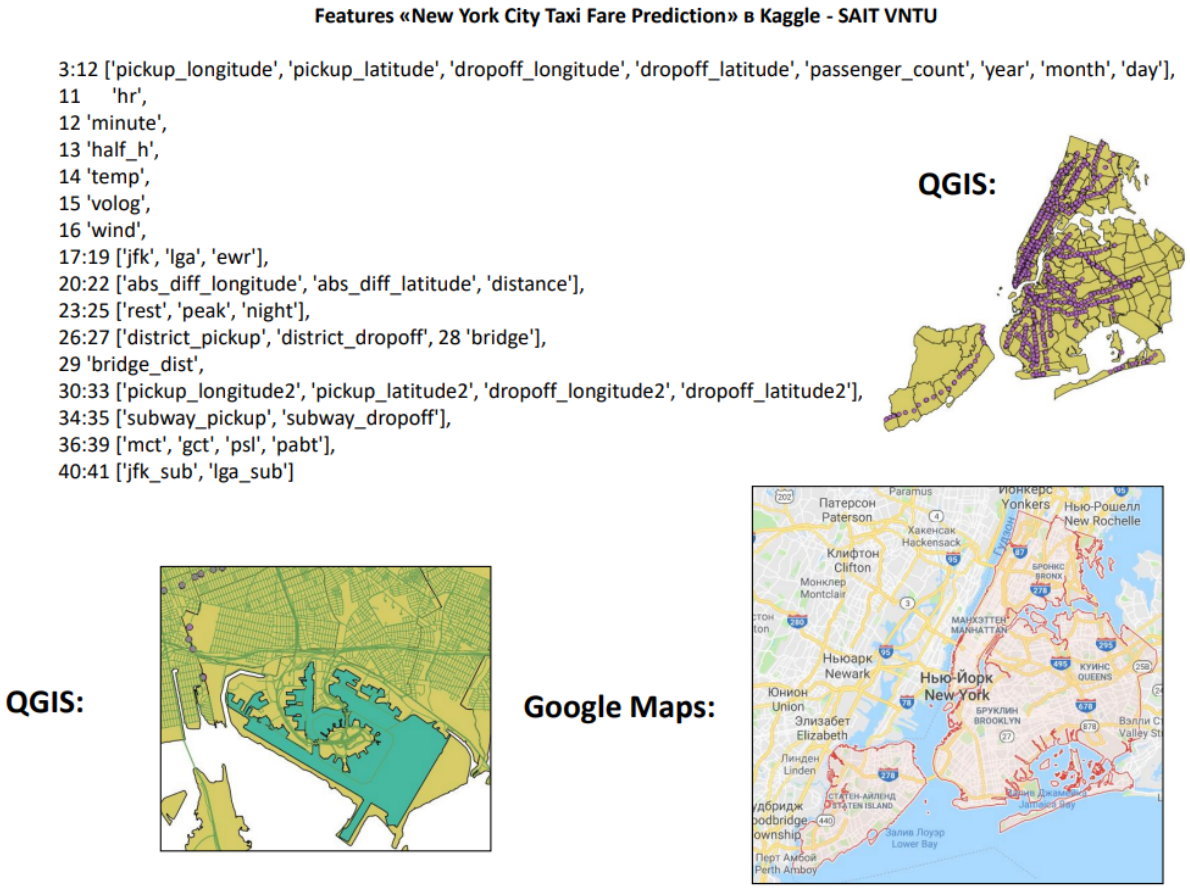


Figure 1.15 – Feature engineering in the Kaggle competition "New York City Taxi Fare Prediction" was performed by the team "SAIT VNTU" of Prof. Vitalii Mokin (2019)

4. Logical Rhythm 2k22 Motorbike Cost. Help predict the price of the motorcycles and be an awesome stuntman (2022).

Predict the price of motorcycles based on various features and become an awesome stuntman in the process. Submissions will be evaluated using Root-Mean-Squared-Log-Error (RMSLE). Use the provided dataset, which includes columns like model_name, model_year, kms_driven, owner, location, mileage, power, and price, to train your model and predict prices.

Data size: 720Kb.

5. Lyft Motion Prediction for Autonomous Vehicles. Build motion prediction models for self-driving vehicles (2020).

In this task, you develop motion prediction models for self-driving vehicles, aiming to predict the trajectories of surrounding traffic agents such as cars, cyclists, and pedestrians (Fig 1.16).

Figure 1.16 – The Kaggle competition
"Lyft Motion Prediction for Autonomous Vehicles"

Utilizing the largest Prediction Dataset released, you will apply your data science and machine learning skills to build and test your models. The goal is to address the challenge of multi-modality and ambiguity in traffic scenes, using either unimodal models for single predictions or multi-modal models for multiple hypotheses. Your work will contribute to advancing autonomous vehicle technology and making transportation safer and more accessible.

Data size: 23.7 Gb.

4. Peking University/Baidu - Autonomous Driving. Can you predict vehicle angle in different settings? (2020).

Self-driving cars, despite significant advancements, still face challenges in accurately perceiving objects in traffic, leading to consumer and legislative hesitation. Your task is to develop an algorithm capable of estimating the absolute pose (6 degrees of freedom) of vehicles from a single image in real-world traffic. This will improve computer vision and aid in the broader adoption of autonomous vehicles, potentially reducing the environmental impact of our growing societies. Submissions will be evaluated based on the mean average precision between the predicted and actual vehicle pose.

Data size: 6.3 Gb.

5. Lyft 3D Object Detection for Autonomous Vehicles. Can you advance the state of the art in 3D object detection? (2019).

This competition challenges participants to advance the state of the art in 3D object detection for autonomous vehicles. Leveraging a large-scale dataset featuring raw sensor inputs from high-end autonomous vehicles, participants

will develop and optimize algorithms to improve perception, prediction, and planning. The goal is to democratize access to high-quality data, fostering innovation in higher-level autonomy functions and ultimately contributing to the development of safer, more efficient self-driving technology. Success will be measured based on mean average precision at various intersections over union (IoU) thresholds.

Data size: 125.8 Gb.

6. Passenger Screening Algorithm Challenge. Improve the accuracy of the Department of Homeland Security's threat recognition algorithms (2017).

This contest seeks to enhance the USA Department of Homeland Security's threat recognition algorithms to reduce high false alarm rates that cause significant delays at airport checkpoints. The challenge invites the data science community to improve the accuracy of these algorithms using a dataset of images from the latest scanning equipment. Participants are tasked with predicting the probability of threats in 17 body zones for each scan, aiming to improve passenger experience and maintain security (Fig. 1.17).

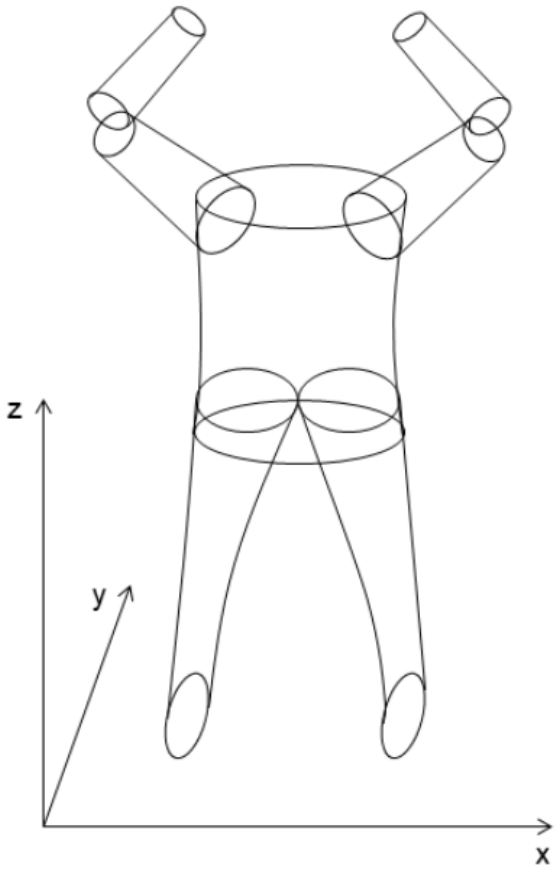


Figure 1.17 – The passenger's body surface is shown in a 2-dimensional representation using a cylindrical coordinate system for 7 body parts: 2 legs, 1 trunk, 2 biceps, and 2 forearms (from the notebook)

No one understands the need for both thorough security screenings and short wait times more than the U.S. Transportation Security Administration (TSA). They are responsible for all U.S. airport security, screening more than two million passengers daily.

This competition is one of the leaders among Kaggle competitions in terms of the value of the prize fund: $1,500,000.

Data size: 134 Gb.

7. Mercedes-Benz Greener Manufacturing. Can you cut the time a Mercedes-Benz spends on the test bench? (2017).

Daimler, one of the world's largest manufacturers of premium cars, is challenging participants to optimize the testing time for Mercedes-Benz vehicles. With a focus on maintaining safety and efficiency, the competition requires developing powerful algorithmic solutions to predict the testing time for various car feature combinations. The goal is to reduce the time vehicles spend on the test bench, thereby lowering carbon dioxide emissions without compromising on quality. Submissions will be evaluated based on the R^2 score.

Data size: 0.35 Mb.

8. Porto Seguro's Safe Driver Prediction. Predict if a driver will file an insurance claim next year (2017).

In this task, you are challenged to develop a model that predicts the probability of a driver filing an auto insurance claim in the next year for Porto Seguro. This improved prediction will help tailor insurance pricing more accurately, making it fairer and more accessible for cautious drivers. Your model's performance will be assessed using the Normalized Gini Coefficient.

Data size: 300.5 Mb.

9. Blue Book for Bulldozers. Predict the auction sale price for a piece of heavy equipment to create a "blue book" for bulldozers (2013).

The task is to predict the auction sale price of heavy equipment, specifically bulldozers, based on usage, equipment type, and configuration. The goal is to create a "blue book" for bulldozers, helping customers value their heavy equipment fleets at auction. The data comes from auction results and includes details on usage and configurations. The evaluation metric is the RMSLE (root mean squared log error) between actual and predicted auction prices.

Data size: 213.8 Mb.

1.5.4. Architecture and the Building Construction

1. Google Landmark Recognition Challenge. Label famous (and not-so-famous) landmarks in images (2018).

The Google Landmark Recognition Challenge invites participants to develop models that accurately identify landmarks in images from a dataset lacking large annotated resources. Unlike traditional image classification challenges, this competition focuses on recognizing a wide array of landmarks, totaling 15,000 classes, with varying amounts of training data per class. Submissions are evaluated using Global Average Precision (GAP), emphasizing precision across diverse landmark categories (Fig. 1.18).

Figure 1.18 – Examples of landmark categories (from the notebook)

The data is no longer available – see them in the updated version below.

2. Google-Landmarks Dataset. Label famous (and not-so-famous) landmarks in images (2022).

The Google-Landmarks Dataset aims to label famous and lesser-known landmarks in images, assisting users in identifying and organizing their vacation photos. It addresses the challenge of landmark recognition by predicting labels directly from image pixels. This dataset, divided into training, test, and index sets, supports two main computer vision tasks: landmark recognition, where each test image is assigned a landmark label, and retrieval, where relevant index images are identified for each test image to aid in further analysis and understanding of landmark features (Fig. 1.19).

Figure 1.19 – Examples of landmark features (from the notebook)

Data size: 1 Gb.

3. ASHRAE - Great Energy Predictor III. How much energy will a building consume? (2019).

In the ASHRAE Great Energy Predictor III competition, participants are tasked with developing accurate models to predict metered energy consumption (chilled water, electric, hot water, and steam) for over 1,000 buildings across a three-year period. The goal is to improve current fragmented estimation methods and support pay-for-performance financing, where payments are based on the difference between actual and predicted energy use. The data includes building metadata, weather information, and energy consumption measurements, with evaluation based on Root Mean Squared Logarithmic Error. Accurate models will encourage investments in building efficiency improvements.

Data size: 2.6 Gb.

1.5.5. Electrical Engineering

1. Global Energy Forecasting Competition 2012 - Wind Forecasting. A wind power forecasting problem: predicting hourly power generation up to 48 hours ahead at 7 wind farms (2012).

The Global Energy Forecasting Competition 2012 focuses on developing models to predict hourly wind power generation up to 48 hours ahead at seven wind farms using historical data and wind forecasts (Fig. 1.20).

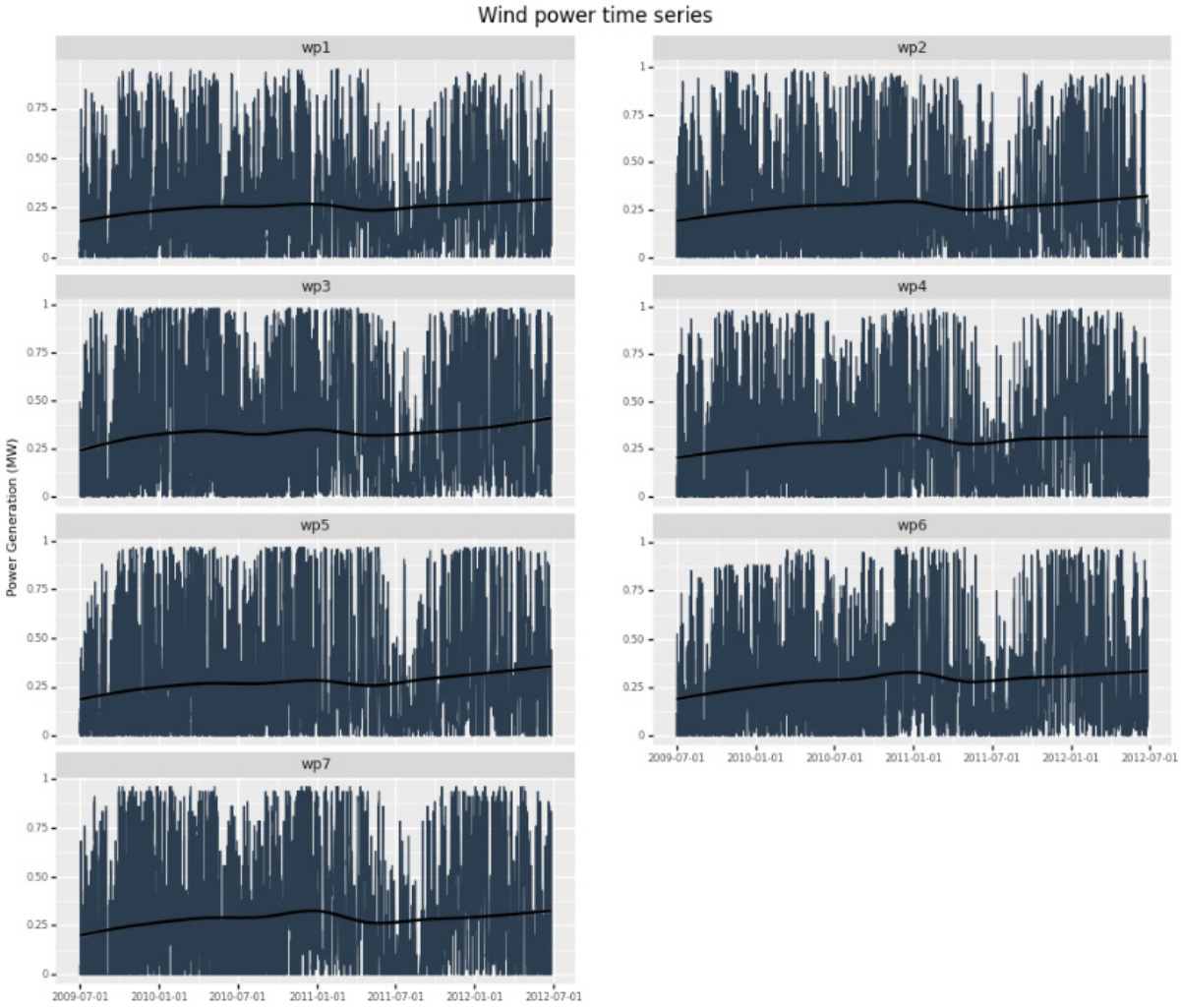


Figure 1.20 – Wind power time series (from the notebook)

The accuracy of these models is evaluated based on RMSE. The data is available for periods ranging from the 1st hour of 2009/7/1 to the 12th hour of 2012/6/28.

Data size: 27 Mb.

2. Global Energy Forecasting Competition 2012 - Load Forecasting. A hierarchical load forecasting problem: backcasting and forecasting hourly loads (in kW) for a US utility with 20 zones (2012).

Develop models to backcast and forecast hourly electricity loads (in kW) for a US utility. The participants are required to backcast and forecast at both the zonal level (20 series) and system (sum of the 20 zonal level series) level, a total

of 21 series. Data (loads of 20 zones and temperature of 11 stations) history ranges from 2004/1/1 to 2008/6/30. Given the actual temperature history, the 8 weeks are set to be missing and are required to be backcasted.

Data size: 9 Mb.

3. AMS 2013-2014 Solar Energy Prediction Contest. Forecast daily solar energy with an ensemble of weather models (2013).

The contest challenges participants to forecast daily solar energy production at 98 Oklahoma Mesonet sites using ensemble weather models. With renewable energy sources like solar fluctuating based on weather conditions, accurate forecasts are crucial for balancing energy resources and minimizing costs. Contestants utilize numerical weather predictions from the GEFS. The competition evaluates predictions against Mean Absolute Error (MAE), aiming to identify models that best predict short-term solar energy production.

Data size: 3 Gb.

4. ASHRAE - Great Energy Predictor III. How much energy will a building consume? (2019).

In the ASHRAE Great Energy Predictor III competition, participants are tasked with developing accurate models to predict metered energy consumption (chilled water, electric, hot water, and steam) for over 1,000 buildings across a three-year period. The goal is to improve current fragmented estimation methods and support pay-for-performance financing, where payments are based on the difference between actual and predicted energy use (Fig. 1.21).

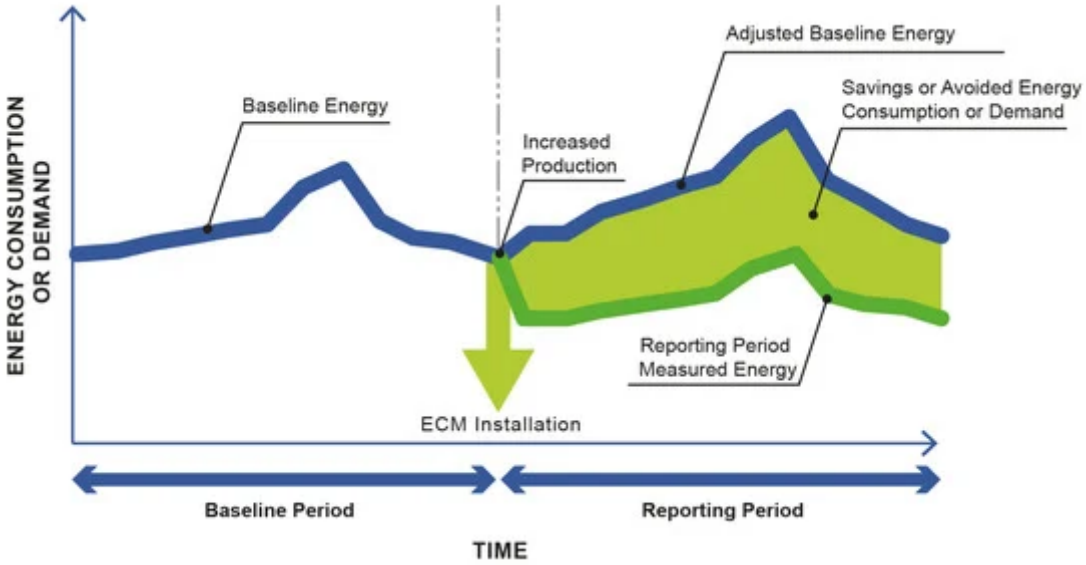


Figure 1.21 – How much energy will a building consume? (From the notebook)

The data includes building metadata, weather information, and energy consumption measurements, with evaluation based on Root Mean Squared Logarithmic Error. Accurate models will encourage investments in building efficiency improvements.

Data size: 2.6 Gb.

5. Enefit - Predict Energy Behavior of Prosumers. Predict Prosumer Energy Patterns and Minimize Imbalance Costs (2024).

The competition is devoted to addressing the issue of energy imbalance caused by prosumers, who both consume and generate energy. The goal is to develop a predictive model that accurately forecasts prosumer energy patterns to minimize imbalance costs, which pose logistical and financial challenges to energy companies. Effective solutions will reduce operational costs, improve grid reliability, and promote efficient integration of prosumers into the energy system. Submissions will be evaluated based on the Mean Absolute Error (MAE) between predicted and actual energy use.

Data size: 1.1 Gb.

6. VSB Power Line Fault Detection. Can you detect faults in aboveground electrical lines? (2019).

Your task is to detect partial discharge patterns in signals from medium voltage overhead power lines, using data acquired with a new meter designed at the ENET Centre at VŠB. Effective classifiers developed from this data will enable continuous monitoring of power lines for faults, reducing maintenance costs and preventing power outages. Submissions will be evaluated based on the Matthews correlation coefficient (MCC) between the predicted and observed responses.

Data size: 12.6 Gb.

7. Belkin Energy Disaggregation Competition. Disaggregate household energy consumption into individual appliances (2013).

The task involves disaggregating household energy consumption into individual appliances. Participants are to develop a system that not only displays total power consumption but also breaks it down by appliance in real time, providing personalized energy-saving recommendations. The challenge is to accurately sense and identify the energy usage of various appliances using machine learning, specifically by examining Electromagnetic Interference (EMI) signatures (Fig 1.22).

There are a few lab-quality videos that may help you grasp the big picture: the video of the signal and the video of the technology applied to energy monitoring.

Submissions will be evaluated based on their accuracy in a multi-label classification task, measured by the mean Hamming Loss.

Data size: 21.5 Gb.

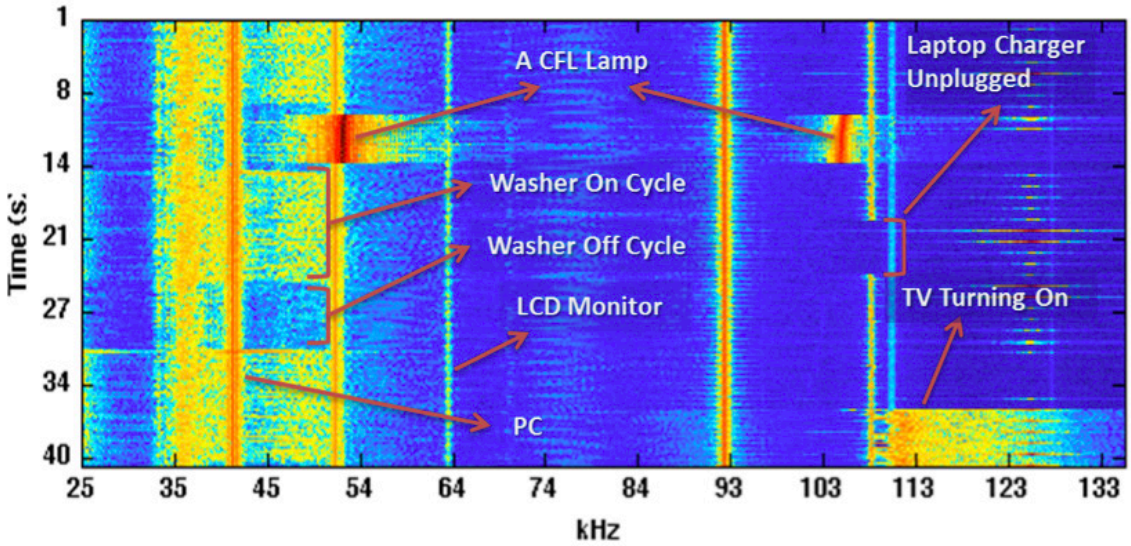


Figure 1.22 – The example of EMI captured from a home: signatures of various appliances in the frequency domain (Kaggle competition "Belkin Energy Disaggregation Competition")

1.5.6. Bioengineering, Medicine

1. Mayo Clinic - STRIP AI. Image Classification of Stroke Blood Clot Origin (2022).

The goal of this competition is to classify the blood clot origins in ischemic stroke. Using whole slide digital pathology images, it is necessary to build a model that differentiates between the two major acute ischemic stroke (AIS) etiology subtypes: cardiac and large artery atherosclerosis (Fig. 1.23).

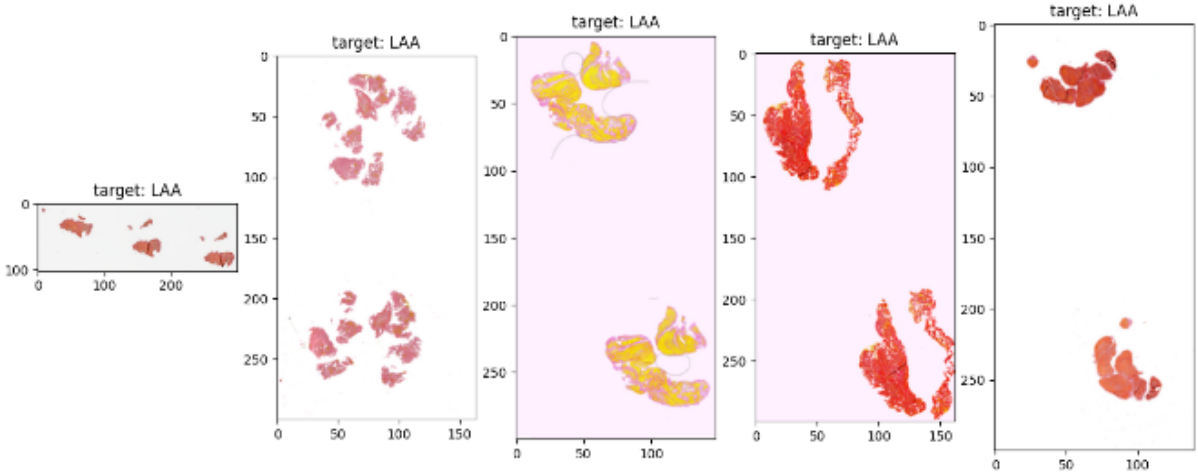


Figure 1.23 – Data of the Kaggle Competition "Mayo Clinic - STRIP AI" (from the notebook)

Data size: 395.36 Gb.

2. UW-Madison GI Tract Image Segmentation. Track healthy organs in medical scans to improve cancer treatment (2022).

Develop a deep learning method to automate the segmentation of the stomach and intestines in daily MRI (Magnetic Resonance Imaging) scans to

expedite radiation therapy for gastrointestinal cancer patients, enhancing treatment precision and reducing patient discomfort (Fig. 1.24).

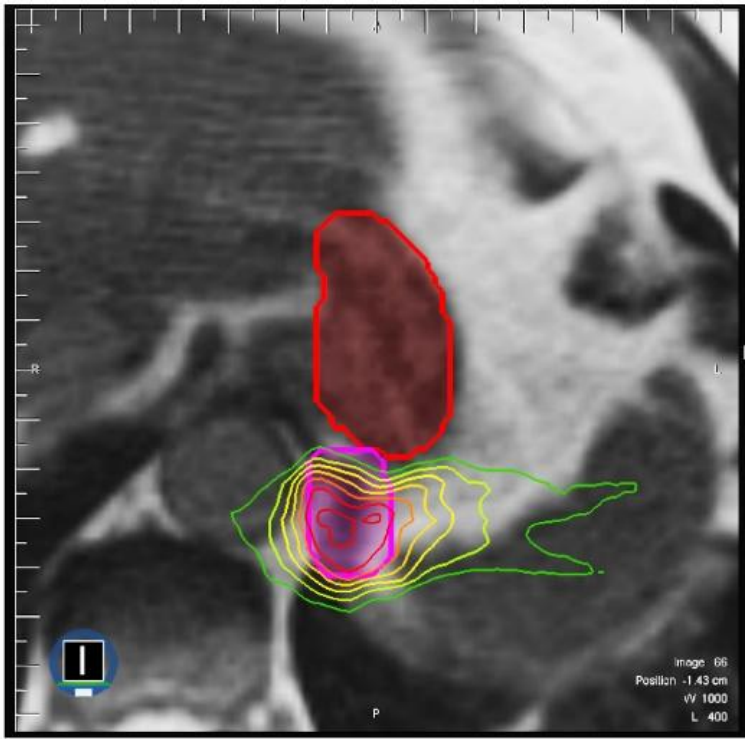


Figure 1.24 – Kaggle competition "UW-Madison GI Tract Image Segmentation": segment the stomach and intestines on MRI scans

In these scans, radiation oncologists must manually outline the position of the stomach and intestines to adjust the direction of the X-ray beams to increase the dose delivery to the tumor and avoid the stomach and intestines. This is a time-consuming and labor-intensive process that can prolong treatments from 15 minutes a day to an hour a day. The contest helps automate this process.

Data size: 2.47 Gb.

3. Google Brain - Ventilator Pressure Prediction. Simulate a ventilator connected to a sedated patient's lung (2021).

The task involves simulating a ventilator connected to a sedated patient's lung, with the best submissions considering lung attributes like compliance and resistance. The goal is to develop algorithms that can generalize across different lung characteristics, ultimately reducing the cost and clinician burden associated with mechanical ventilation. The competition is judged based on the mean absolute error between predicted and actual pressures during the inspiratory phase.

Data size: 0.7 Gb.

4. Grasp-and-Lift EEG Detection. Identify hand motions from EEG recordings (2015).

The task is to develop a model that accurately identifies specific hand motions (grasping, lifting, replacing objects) from electroencephalography (EEG) recordings. This is crucial for advancing brain-computer interface (BCI) prosthetic devices, aiming to restore independence to patients with neurological dis-

abilities who have lost hand function. The competition evaluates submissions based on the mean column-wise Area Under the Curve (AUC) of ROC curves, requiring calibrated probabilities across multiple subjects and series to ensure consistent scaling of predictions.

Data size: 1.1 Gb.

5. Stanford Ribonanza RNA Folding. Create a model that predicts the structures of any RNA molecule (2023).

The competition challenges participants to develop a model that accurately predicts the structures of RNA molecules based on experimental chemical reactivity data (Fig. 1.25).

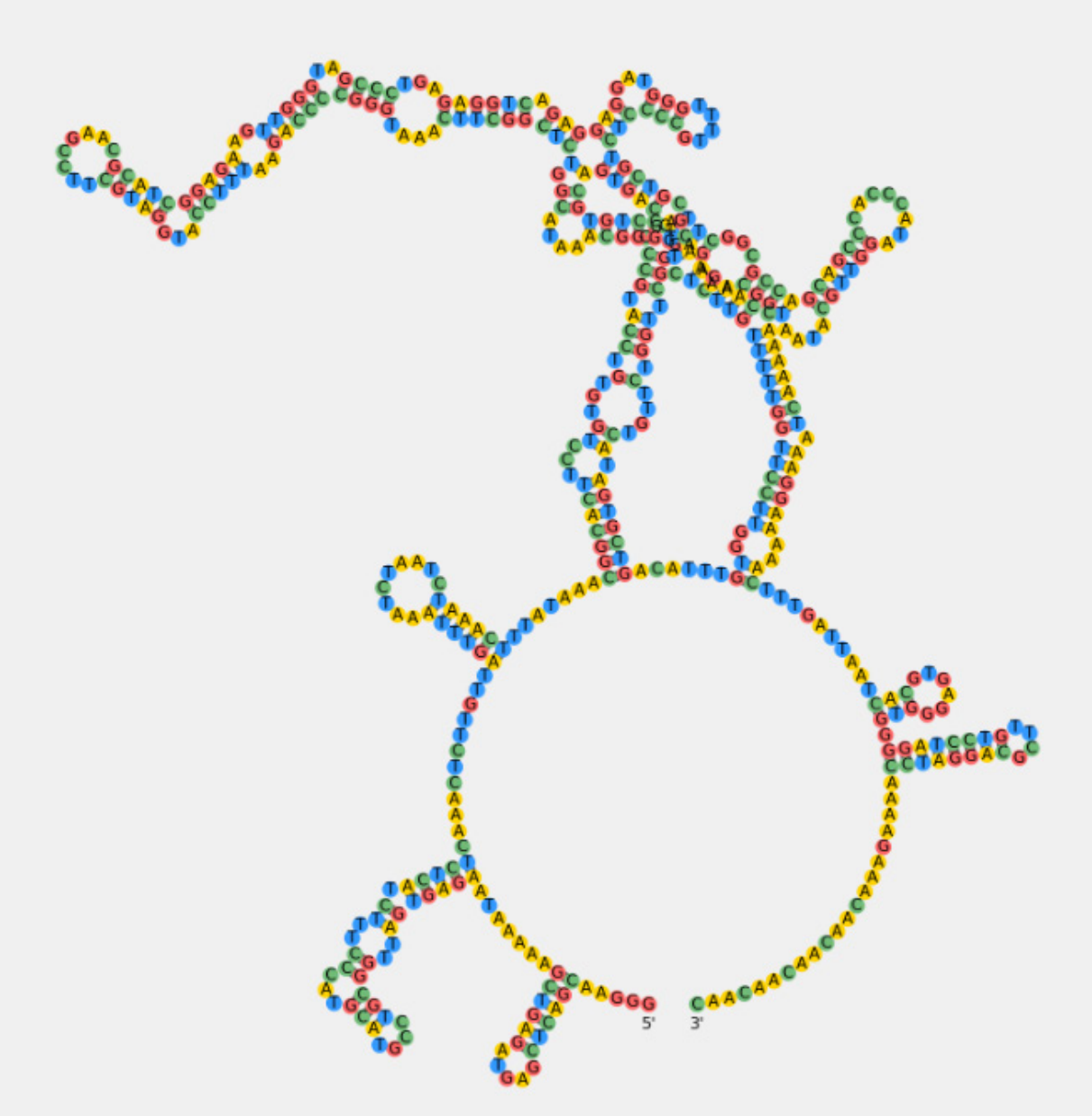


Figure 1.25 – Predicted structure of RNA molecules (from the notebook)

This predictive model is crucial for advancing medical research by enabling the discovery of RNA-based therapies and addressing antibiotic resistance,

while also offering insights into fundamental biological processes and potential applications in biotechnology and climate change mitigation. Submissions are evaluated based on their Mean Absolute Error (MAE) in predicting chemical reactivity profiles across RNA molecules.

Data size: 130.4 Gb.

6. Mechanisms of Action (MoA) Prediction. Can you improve the algorithm that classifies drugs based on their biological activity? (2020).

Your objective is to enhance an algorithm that classifies drugs based on their biological activity, specifically their Mechanism of Action (MoA). This involves developing a multi-label classification model using a unique dataset combining gene expression and cell viability data across 100 different cell types. You will train your model on a provided training dataset to predict MoA labels for a test set. The performance of your algorithm will be evaluated using the average logarithmic loss function on the drug-MoA annotation pairs. Successful development of this algorithm will aid in advancing the drug discovery process by accurately predicting compounds' MoAs based on their cellular signatures.

One of the authors of this manual (Prof. Vitalii Mokin) received a bronze medal for participating in this competition.

Data size: 216 Mb.

1.5.7. Management, Economy, Finance

1. Loan Default Prediction - Imperial College London. Constructing an optimal portfolio of loans (2014).

The task involves predicting loan defaults and estimating the associated losses, aiming to go beyond binary classification by predicting both default likelihood and loss severity. This approach bridges traditional banking's focus on economic capital reduction with asset management's risk optimization for financial investors. Evaluation in this competition is based on mean absolute error (MAE), emphasizing accurate prediction of loss amounts incurred from loan defaults.

Data size: 0.6 Gb.

2. RecSys2013: Yelp Business Rating Prediction. RecSys Challenge 2013: Yelp business rating prediction (2013).

The contest focuses on predicting Yelp business ratings based on a detailed dataset from Phoenix, AZ, including over 10,000 businesses, 8,000 check-in sites, 40,000 users, and 200,000 reviews. Participants are tasked with creating a model to predict future user ratings of businesses which is evaluated using the root mean squared error (RMSE) metric to measure accuracy.

Data size: 0.18 Gb.

3. CPROD1: Consumer PRODucts contest #1. Identify product mentions within a largely user-generated web-based corpus and disambiguate the mentions against a large product catalog (2012).

The CPROD1 contest focuses on identifying and disambiguating consumer product mentions within user-generated web content against a large product

catalog. Participants must develop state-of-the-art methods to automatically recognize product mentions in a diverse collection of web content and accurately match them to specific products in a catalog of over fifteen million items.

Data size: 1.6 Gb.

4. Indoor Location & Navigation. Identify the position of a smartphone in a shopping mall (2021).

The task is to predict the indoor position of smartphones using real-time sensor data provided by XYZ10 and Microsoft Research. There is the dataset of nearly 30,000 traces from over 200 buildings to improve the accuracy of indoor positioning solutions, which is crucial for enhancing location-based apps and services. Submissions will be evaluated based on the mean position error.

Data size: 60 Gb.

5. Optiver - Trading at the Close. Predict US stocks closing movements (2024).

In this competition, you are tasked with developing a model to predict the closing price movements for hundreds of Nasdaq-listed stocks. Using data from the order book and the closing auction, your model will assess supply and demand dynamics, adjust prices, and identify trading opportunities, especially during the critical final ten minutes of trading. This project provides an opportunity to handle real-world data science problems, similar to those faced by professionals at Optiver, and aims to improve market efficiency and accessibility. Submissions will be evaluated based on the Mean Absolute Error (MAE) between the predicted return and the observed target.

Data size: 0.65 Gb.

6. GoDaddy - Microbusiness Density Forecasting. Forecast Next Month's Microbusiness Density (2023).

The goal of the GoDaddy Microbusiness Density Forecasting competition is to predict the monthly density of microbusinesses in specific areas using U.S. county-level data. Participants will develop models to provide accurate forecasts, which will aid policymakers in understanding and supporting microbusinesses. The competition aims at utilizing advanced data science techniques to improve current econometric models and better inform policy decisions. Submissions will be evaluated based on the Symmetric Mean Absolute Percentage Error (SMAPE) between the predicted and actual values.

Data size: 11.4Mb.

7. OTTO – Multi-Objective Recommender System. Build a recommender system based on real-world e-commerce sessions (2022).

The task of this competition is to build a "Multi-Objective Recommender System" to predict e-commerce clicks, cart additions, and orders based on real-world user session data. The goal is to improve the shopping experience by providing tailored recommendations, thereby increasing customer satisfaction and retailer sales (Fig. 1.26).

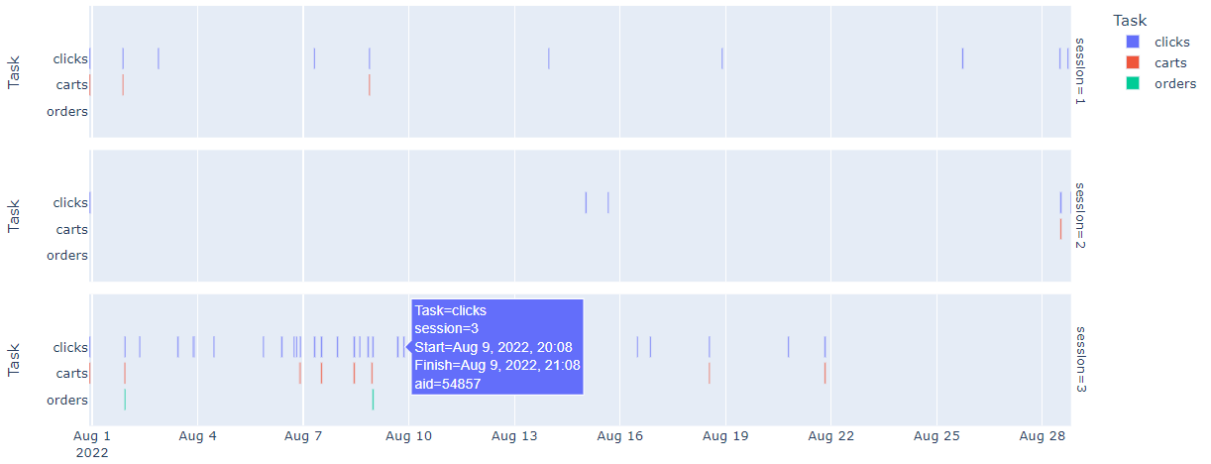


Figure 1.26 – E-commerce clicks, cart additions, and orders based on real-world user sessions data (from the notebook)

This involves creating a single model to optimize multiple objectives simultaneously and will be evaluated based on Recall@20 for each action type, with weighted averages for the three recall values.

Data size: 11.9 Gb.

8. American Express - Default Prediction. Predict if a customer will default in the future (2022).

In this competition, participants are tasked with developing a machine learning model to predict customer credit defaults. By leveraging industrial-scale datasets that include time-series behavioral data and anonymized customer profiles, the goal is to create a model that surpasses the current production model used by American Express. The evaluation metric for the competition is the mean of the Normalized Gini Coefficient and the default rate captured at 4%. Successful models could improve customer experiences and optimize lending decisions.

Data size: 50.3 Gb.

9. JPX Tokyo Stock Exchange Prediction. Explore the Tokyo market with your data science skills (2022).

The task involves predicting future returns for stocks on the Tokyo Stock Exchange. Participants build models to rank approximately 2,000 stocks from highest to lowest expected returns and be evaluated on the Sharpe Ratio of their daily spread returns. The competition provides historical and real-time financial data, and the goal is to identify undervalued stocks to buy and overvalued stocks to sell. Successful models foster learning and potentially increase retail investor interest in quantitative trading.

Data size: 1.33 Gb.

10. Ubiquant Market Prediction. Make predictions against future market data (2022).

In the Ubiquant Market Prediction competition, your task is to develop a model that accurately forecasts the return rates of investments. You will train and test your algorithm using historical price data, aiming for high precision in

your predictions. Successful models will enhance the capability of quantitative researchers to predict returns, aiding investors in making informed decisions. Submissions will be evaluated based on the mean Pearson correlation coefficient for each time ID.

This dataset is no longer available for download but there are many public notebooks with solutions worth exploring.

11. G-Research Crypto Forecasting. Use your ML expertise to predict real crypto market data (2022).

Participants in this competition are tasked with using machine learning to predict short-term returns for 14 popular cryptocurrencies. The competition provides a dataset of high-frequency market data, including various price and trading metrics, dating back to 2018. Participants must build models that forecast returns, with submissions evaluated based on a weighted version of the Pearson correlation coefficient. The challenge involves handling volatile and non-stationary data while avoiding overtraining to achieve persistent predictive accuracy.

Data size: 3.12 Gb.

See the article by Prof. Vitalii Mokin with co-authors "Information Technology for the Cryptocurrency Rate Forecasting on the Basics of Complex Feature Engineering" about this contest and other decisions regarding cryptocurrencies (Ukrainian language).

12. M5 Forecasting - Uncertainty. Estimate the uncertainty distribution of Walmart unit sales (2020).

This competition challenges participants to estimate the uncertainty distribution of Walmart's unit sales for various products sold across the USA. Using hierarchical sales data from Walmart, including item-level details, department categories, and store information from three states, competitors must forecast daily sales for the next 28 days and quantify the uncertainty of these forecasts. The robust dataset also includes explanatory variables like price, promotions, and special events. Participants are encouraged to use both traditional forecasting methods and machine learning to enhance forecast accuracy, with the goal of advancing forecasting theory and practice.

Data size: 0.5 Gb.

One of the authors of this manual (Prof. Vitalii Mokin) received a bronze medal for participating in this competition.

13. Recruit Restaurant Visitor Forecasting. Predict how many future visitors a restaurant will receive (2018).

The objective of this competition is to develop a predictive model to estimate the number of visitors a restaurant will receive on future dates, using reservation and visitation data. This task is crucial for helping restaurants optimize their ingredient purchases and staffing schedules, mitigating the challenges posed by unpredictable factors like weather and local competition. Recruit Holdings provides access to valuable datasets to support the development of accurate

forecasts. Submissions will be assessed based on the root mean squared logarithmic error (RMSLE).

Data size: 27.3 Mb.

14. Porto Seguro's Safe Driver Prediction. Predict if a driver will file an insurance claim next year (2017).

In this task, you are challenged to develop a model that predicts the probability of a driver filing an auto insurance claim in the next year for Porto Seguro. This improved prediction will help tailor insurance pricing more accurately, making it fairer and more accessible for cautious drivers. Your model's performance will be assessed using the Normalized Gini Coefficient.

Data size: 300.5Mb

15. Instacart Market Basket Analysis. Which products will an Instacart consumer purchase again? (2017).

In this competition, participants are tasked with predicting which previously purchased products will be included in a user's next order using anonymized customer order data. The dataset includes over 3 million grocery orders from more than 200,000 users, detailing the sequence of products, order times, and intervals between orders. Competitors will be evaluated based on their predictions' mean F1 score. This analysis helps Instacart enhance its recommendation systems and improve user experience.

Data size: 205.8 Mb.

16. House Prices - Advanced Regression Techniques. Predict sales prices and practice feature engineering, RFs, and gradient boosting (this competition runs indefinitely with a rolling leaderboard).

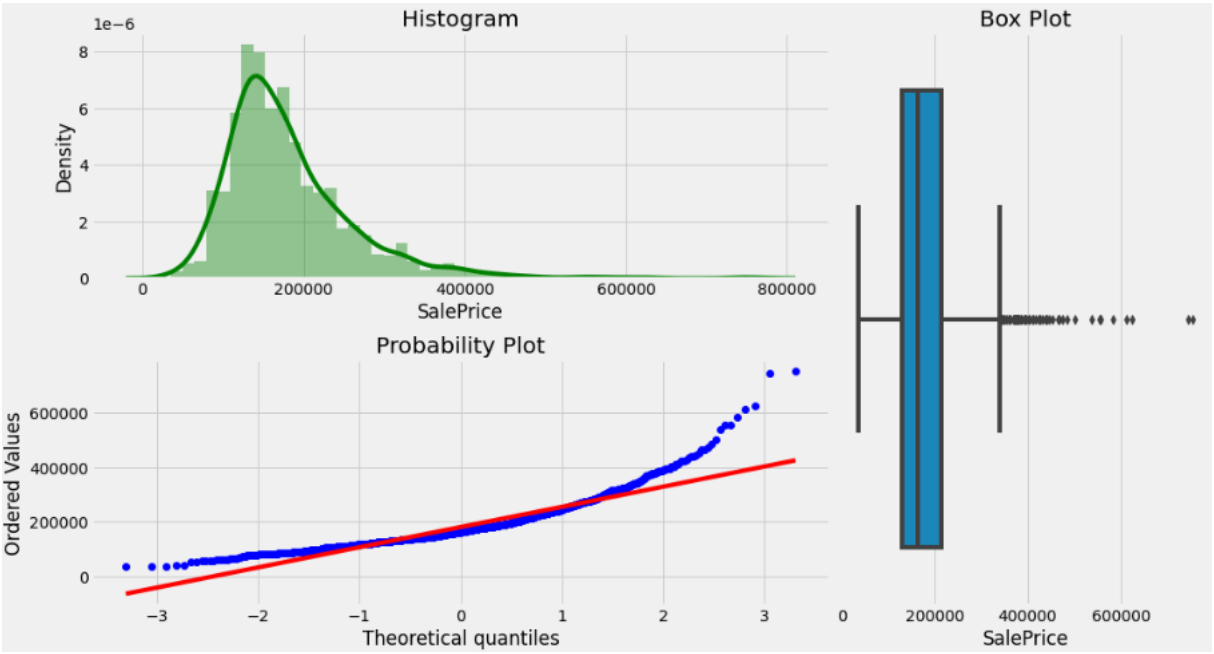


Figure 1.27 – The Histogram, the Probability Plot and the Box Plot for the feature 'SalePrice' in the training dataset of this contest (from the notebook)

In this competition, your task is to predict the final sales price of homes in Ames, Iowa, using a dataset with 79 explanatory variables that describe various aspects of residential properties (Fig. 1.27).

You need to create a model that can accurately estimate the sales price for each house, given its unique features. Submissions will be evaluated based on the Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted prices and the logarithm of the actual sales prices, ensuring that errors in predicting both expensive and inexpensive homes are treated equally.

This is the competition from the "Getting Started competitions" which Kaggle data scientists created for people who have little to no machine learning background. They are a great place to begin if you are new to data science or just finished a MOOC and want to get involved in Kaggle.

Data size: 1 Mb.

17. Store Sales - Time Series Forecasting. Use machine learning to predict grocery sales (this competition runs indefinitely with a rolling leaderboard).

Your task is to build a machine learning model to predict grocery sales for thousands of items sold at different Favorita stores. You'll work with a dataset containing dates, store and item information, promotions, and unit sales to improve the accuracy of sales forecasts. The goal is to reduce overstocking and stock outs, thereby minimizing food waste and improving customer satisfaction. The competition uses Root Mean Squared Logarithmic Error as the evaluation metric.

This is the competition from the "Getting Started competitions" too.

Data size: 125 Mb.

1.5.8. Agricultural Engineering, Environment, Biology

1. Beyond Visible Spectrum: AI for Agriculture 2023. Boosting automatic crop type classification using Sentinel satellite data and self-supervised learning (2023).

In the contest challenge, participants are tasked with developing self-supervised learning (SSL) models for automatic crop type classification using a massive remote sensing dataset, including multispectral and SAR data (Fig. 1.28).

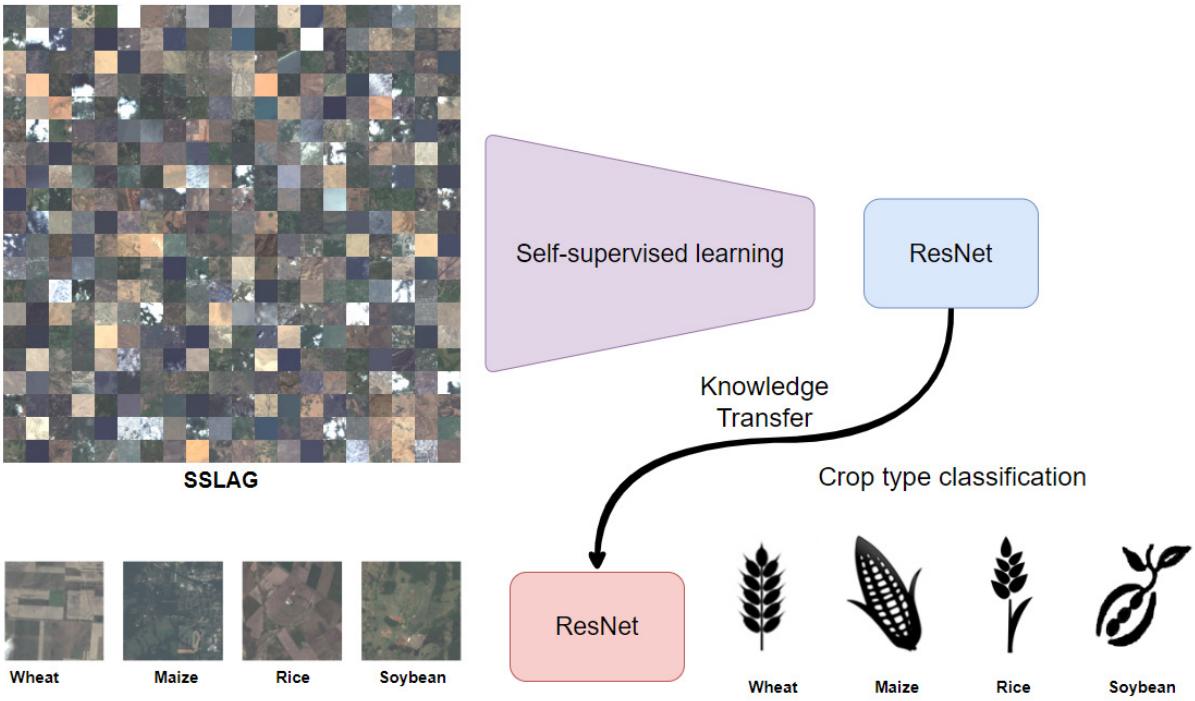


Figure 1.28 – Beyond Visible Spectrum: Boosting automatic crop type classification using Sentinel satellite data and self-supervised learning

The goal is to improve crop classification accuracy without relying on labeled training data, addressing the challenge of time-consuming ground-truth sample collection.

Data size: 2.34 Gb.

2. Beyond Visible Spectrum: AI for Agriculture 2023. Automated Crop Disease Diagnosis from Hyperspectral Imagery 2nd (2023).

In the contest challenge, participants are tasked with developing models for the accurate diagnosis of yellow rust disease in crops using hyperspectral imagery. The goal is to enhance precision management by leveraging detailed spectral-spatial information for better diagnostic accuracy, with the performance of models evaluated based on categorization accuracy.

Data size: 0.9 Gb.

3. Wids datathon (Optimizing Agricultural Production). Develop a model to identify the most profitable crop to grow in a Specific agricultural region (2023).

Develop a machine learning model to identify the most suitable crop to grow in a specific agricultural region using data on weather, soil conditions, and crop growth. Utilize datasets containing information on rainfall, climate, and fertilizer data to optimize agricultural production by leveraging the precision agriculture approach, which includes the use of GPS, drones, and sensors for data collection. The goal is to enhance efficiency, reduce costs, and improve crop yields by analyzing factors such as soil nitrogen, phosphorus, potassium content, temperature, humidity, soil pH, and rainfall.

Data size: 150 Kb.

4. ML Olympiad – AgriSol. Use TensorFlow to build an image classification model to predict crop diseases (2022).

Data: 116933 files, size 1.8 Gb. Test contains 33 test images for later for prediction purpose.

5. BirdCLEF 2024. Bird species identification from audio, focused on under-studied species in the Western Ghats, a major biodiversity hotspot in India (2024).

The task is to apply machine-learning techniques to identify under-studied bird species in the Western Ghats of India using audio data. Participants will develop computational solutions to recognize bird species by their calls, focusing on endemic and endangered species, as well as nocturnal species. This aims to leverage passive acoustic monitoring and machine learning for more efficient and effective avian biodiversity assessment, supporting conservation efforts in this biodiverse region (Fig 1.29).

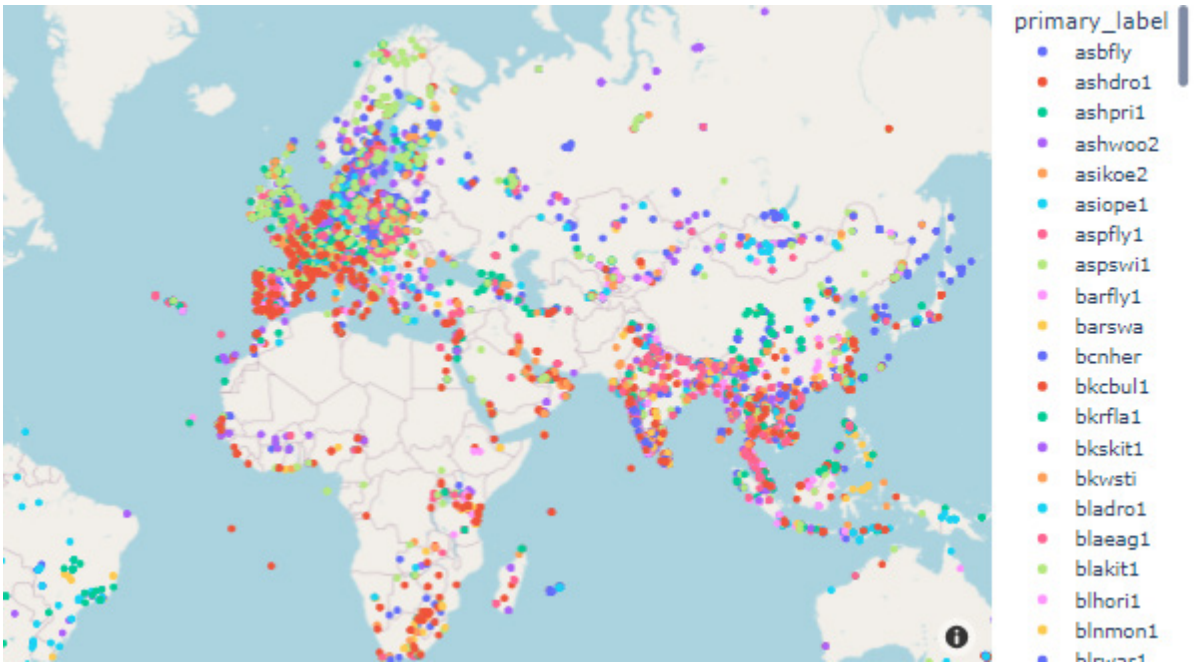


Figure 1.29 – Geographical Distribution of Bird Species (from the notebook)

The competition evaluates solutions using a macro-averaged ROC-AUC metric adapted to skip classes with no true positive labels, facilitating accurate population trend assessments and adaptive conservation strategies.

Data size: 23.4 Gb.

1.5.9. Education and Social spheres

1. Visualize the State of Public Education in Colorado. Using 3 years of school grading data supplied by the Colorado Department of Education and R-Squared Research, visually uncover trends in the Colorado public school system (2013).

Visualize the State of Public Education in Colorado by analyzing and presenting trends from three years of school grading data (2013) provided by the

Colorado Department of Education and R-Squared Research. Use the Colorado School Grades platform to create accessible and easy-to-understand visualizations that help community members, parents, students, and educators make informed decisions and engage in local school improvement efforts.

Data size: 7 Mb.

2. The Learning Agency Lab - PII Data Detection. Develop automated techniques to detect and remove PII from educational data (2024).

The Learning Agency Lab is hosting a competition to develop automated techniques for detecting and removing personally identifiable information (PII) from educational data. The goal is to create a model that can accurately identify PII in student writing, which will reduce the cost and increase the scalability of releasing educational datasets for research and tool development (Fig. 1.30).

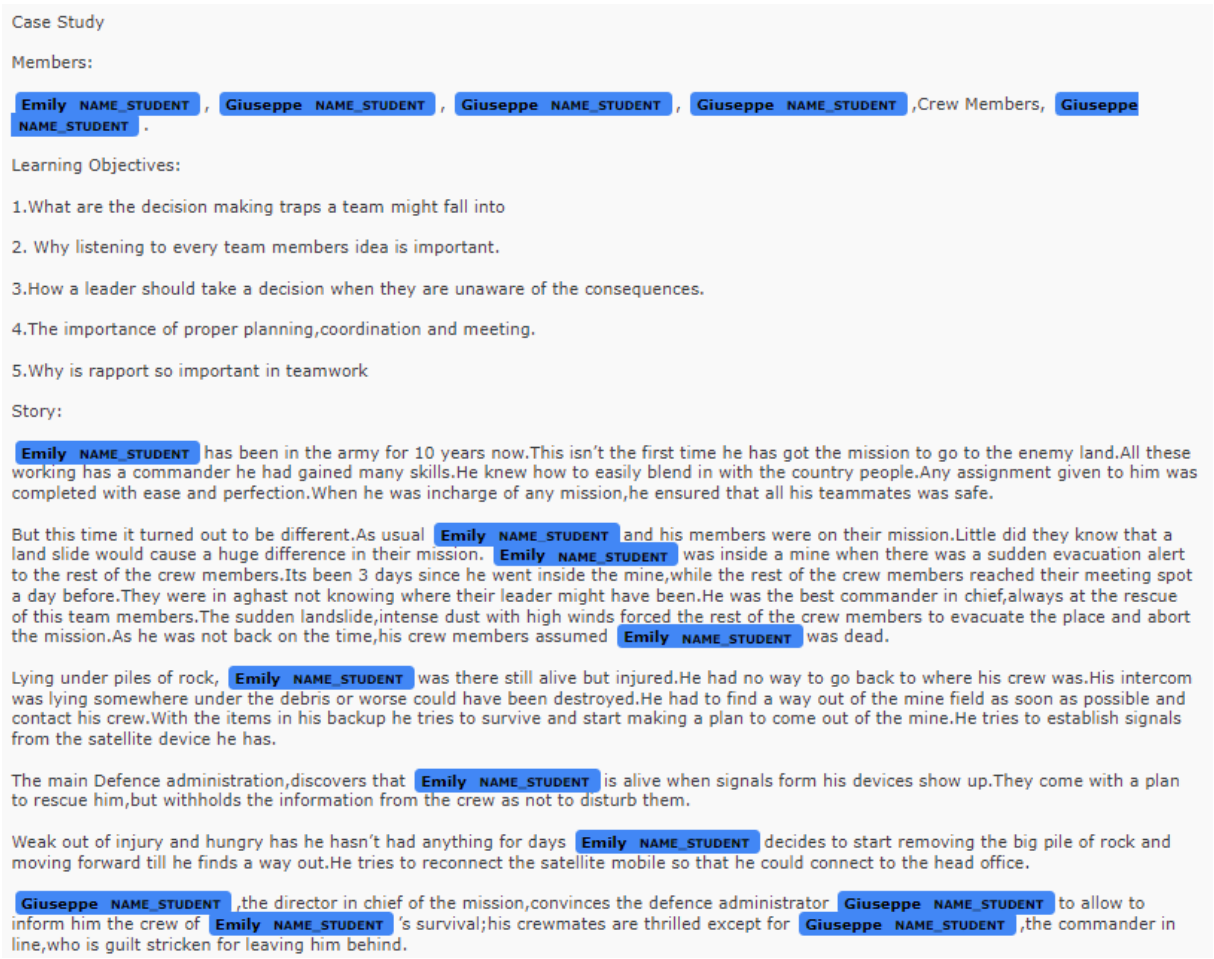


Figure 1.30 – Visualization of an example of hiding personal information in found patterns in the text (from the notebook)

Current methods, such as manual review and named entity recognition (NER), are either too costly or insufficiently accurate. Submissions will be evaluated based on a classification metric that prioritizes recall over precision to ensure comprehensive PII detection.

Data size: 110 Mb.

3. CommonLit - Evaluate Student Summaries. Automatically assess summaries written by students in grades 3-12 (2023).

The task is to develop a model that automatically evaluates the quality of summaries written by students in grades 3-12. This model should assess how well a student captures the main ideas and details of a source text, as well as the clarity, precision, and fluency of the summary. Using a collection of real student summaries, the goal is to assist teachers in evaluating student work efficiently and help learning platforms provide immediate feedback. Submissions are scored using the Mean Columnwise Root Mean Squared Error (MCRMSE) metric.

Data size: 3.45 Mb.

4. Learning Equality - Curriculum Recommendations. Enhance learning by matching K-12 content to target topics (2023).

The goal of this competition is to streamline the process of matching K-12 educational content to specific curriculum topics using an accurate and efficient model. Participants will develop models trained on a diverse library of educational materials organized by topic taxonomies, especially in STEM subjects. The challenge lies in aligning these materials to various national curricula, a process currently done manually and requiring significant resources. Submissions will be evaluated based on their mean F2 score, calculated for each predicted row and then averaged.

Data size: 0.9 Gb.

5. Feedback Prize - English Language Learning. Evaluating language knowledge of ELL students from grades 8-12 (2022).

The Feedback Prize – English Language Learning competition aims to evaluate the language proficiency of 8th-12th grade English Language Learners (ELLs). Participants will use a dataset of essays written by ELLs to develop proficiency models that provide accurate feedback on language development, expediting the grading cycle for teachers (Fig. 1.31).

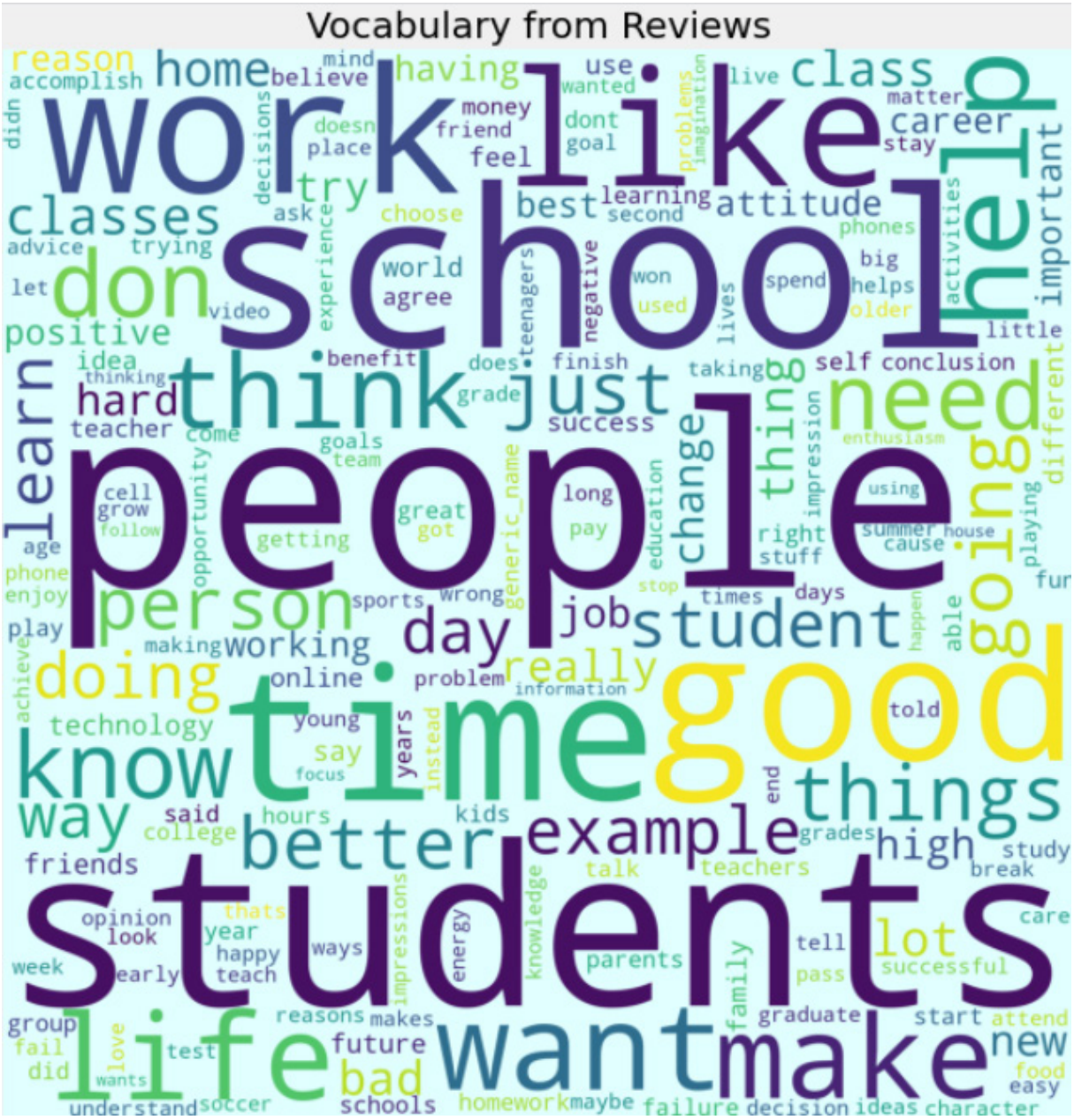


Figure 1.31 – WordCloud "Vocabulary from Reviews" for this contest (from the notebook)

This initiative addresses the lack of tailored feedback in existing automated tools, which often fail to meet the unique needs of ELL students. The goal is to enable ELLs to receive more appropriate learning tasks, ultimately enhancing their English language proficiency. Submissions are scored using the MCRMSE.

Data size: 9.3 Mb.

Practical exercises

1) Calculate the result of the "**print (True and not False or False)**" operation on Python. Experiment with other options for using parentheses and operations "not", "and", "or". Remember that the expressions in parentheses are processed first, then "not", then equivalent in status "and" and "or".

2) Practice applying operations **sorted, min, max, len** to lists of numbers (Fig. 1.32 and see Appendix A).

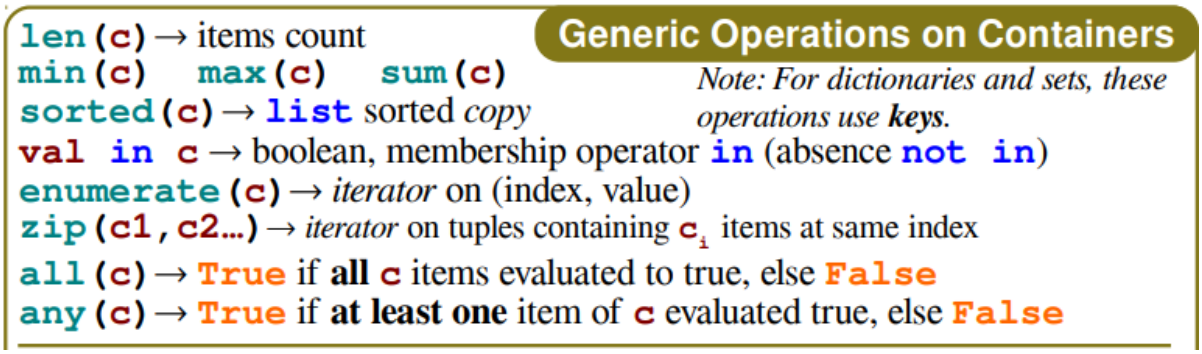


Figure 1.32 – Common operations

3) Practice correctly applying operations lst[3:7], lst[-4:-1], lst[3:], lst[:-4] and other for text, for example lst = “learning” (Fig. 1.33 and see Appendix A).

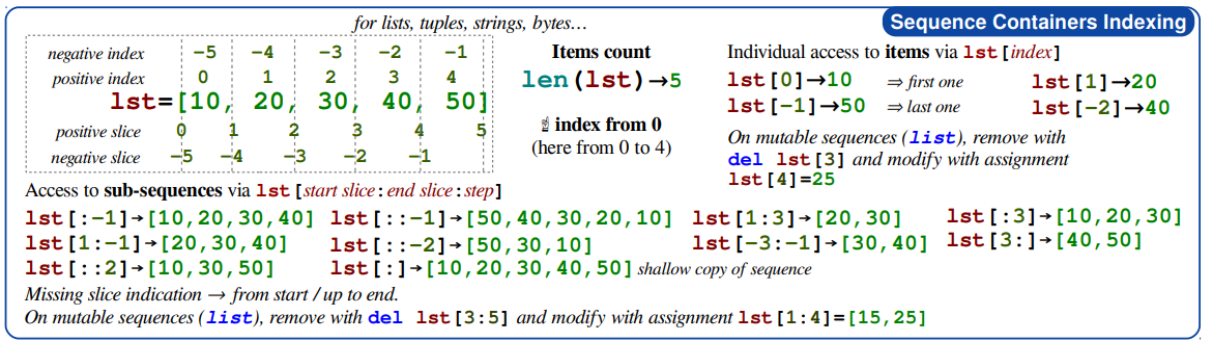
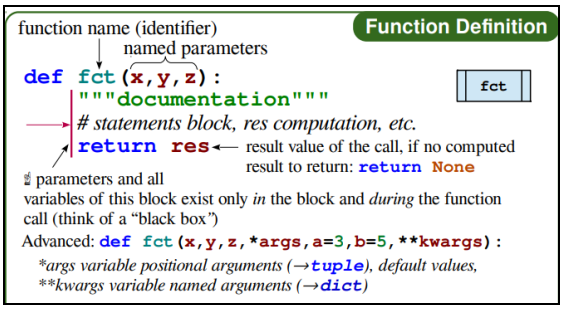
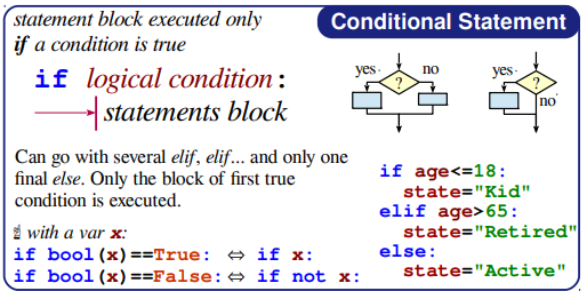


Figure 1.33 – Indexing of a list

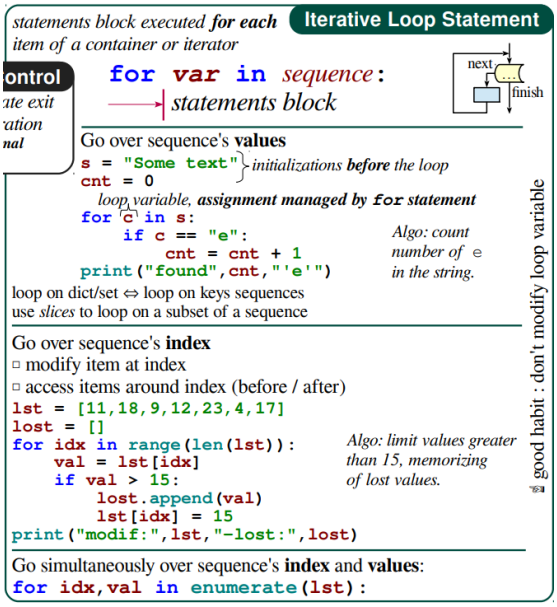
4) Practice correctly writing and applying user’s function ("def...return"), condition ("if .... else"), loop ("for...in..."), understand the options for using "range" in the loop "for...in…" (for examples: range(5), range(1,5), range(2,12,3), etc.) (Fig. 1.34 and see Appendix A).



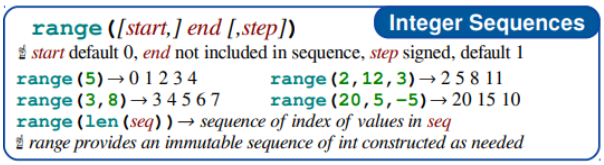
a)



b)



c)



d)

Figure 1.34 – Basic structures of the Python code and its elements: a) user's function, b) conditions, c) loop "for"; d) range for loop "for"

Note that in Python, it is very popular to write a condition in this concise form:

```
x = 0 if a > 0 else 1
```

instead of:

```
if a > 0:
    x = 0
else:
    x=1
```

Possible topics of practical tasks

Topic No. 1. Formulation of the machine learning task using the example of real tasks and tasks of competitions of Artificial Intelligence Kaggle platform

The purpose of the lesson is to get acquainted with the formulation of tasks and selection of information technologies that were used for solving the tasks of the competitions of the international platform of artificial intelligence Kaggle or real tasks using the example of one of the Kaggle datasets.

Lesson plan:

1. Select a competition or Kaggle dataset that has at least one notebook, the author of which is an expert, master, or grandmaster of Kaggle. Provide the title (the main one and additional), web link, an author or organization that owns the Kaggle dataset or data of the Kaggle contest.

2. Describe the composition of the data tables (or one main table) of the dataset (column names, for which years). It is worth providing graph(s) from notebooks that illustrate exactly what data is in the dataset. If the data is geographically referenced, provide a map that illustrates this information.

3. Characterize the tasks that can be solved on the basis of this dataset (or from the competition task, from the "Task" section of the dataset, or come up with it yourself).

4. Indicate and characterize which Python libraries and/or information technologies were used in the notebooks of the competition or Kaggle dataset with the best rating (either with the highest places in the competition or with the most votes for the notebook). For example, Plotly library for building interactive graphs, Xgboost library for model building, Folium library for building interactive map, IT analysis of image recognition based on PyTorch, etc.

Examples of datasets:

- notebook with links and description of datasets of Prof. Mokin V.B. at Kaggle in the field of water quality monitoring;
- other public datasets of Prof. Mokin V.B. in Kaggle:
- a popular contest «Titanic - Machine Learning from Disaster» for newcomers to Kaggle (it is important to note that the notebooks of Kaggle competitions cannot be used in labs № 2-8, because the rules of the competition prohibit sharing them with the teacher – they can be used only if the authors make public notebooks immediately, that is, available to everyone Internet users).

Topic No. 2. Generalized formulation of the problem and construction or selection of a dataset for it.

The purpose of the lesson is to acquire the knowledge and skills to create your own dataset and read it using a Python program.

Lesson plan:

1. Clarify the formulation of the problem in order to understand the most necessary data for its solution.

2. Analyze available public datasets in Kaggle, GitHub.

3. Study the data source (title, author, content, size of data and their description). Find examples or understand in the documentation or description how to import data in Python.

4. Explore the possibilities and master the basic skills of working with a given IDE (PyCharm, VSCode or Spyder in Anaconda) or a "Jupyter Notebook" (JNB) type shell (SageMaker of Amazon, Google Colab, Kaggle notebook editor or in Jupyter Notebook or JupiterLab in Anaconda) to create Python programs.

5. Create a Python program in IDE/JNB from point 3 to download data from point 1 using techniques from point 2.

- *Examples of JNB notebooks with techniques for uploading data in various ways to Kaggle, including via API:*

- 50 Tips: Data Science (tabular data) for beginner
- 50 Advanced Tips: Data Science for tabular data

Topic No. 3.

Formation of an integrated dataset for analyzing system state data from various sources (API, CSV files, etc.) in Python in IDE or Jupyter Notebook.

The purpose of the lesson is to study data storage systems and learn the skills to read data from these systems using a Python program.

Lesson plan:

1. Study the data source (title, author, content, amount of data and their description).

2. Study examples of how to import data into Python from the following sources:

- information system or IoT system;
- Kaggle dataset (CSV, JSON, etc. formats);
- GitHub dataset;
- web system with API.

3. Explore the possibilities and master the basic skills of working with a given IDE (PyCharm, VSCode or Spyder in Anaconda) or a "Jupyter Notebook" (NB) type shell (SageMaker of Amazon, Google Colab, Kaggle notebook editor or in Jupyter Notebook or JupiterLab in Anaconda) to create Python programs.

4. Create a Python program in IDE/JNB from point 3 to download data from point 1 using techniques from point 2. Combine all data into one or more dataframes.

Examples of JNB notebooks with techniques for uploading data in various ways to Kaggle, including via API:

- 50 Tips: Data Science (tabular data) for beginner
- 50 Advanced Tips: Data Science for tabular data

Test questions

1) What does the concept of data science include? Give a short definition.

2) What stages does the process of collecting information and building a dataset for analysis include?

3) What target features can be used in machine learning tasks? Give examples.

4) What are the main types of machine learning problems? Give a brief description of each species.

5) What metrics are used to evaluate the quality of machine learning models? Give examples of metrics.

6) What does the generalized algorithm for solving the machine learning problem contain? List the basic steps.

7) What stages does the generalized algorithm contain for solving the problem of intelligent data analysis? Describe each stage.

8) What infrastructure is used to solve machine learning and data analysis problems? Give examples of infrastructure.

9) What are examples of setting tasks for machine learning and intelligent data analysis?

2 DATA PREPROCESSING AND EXPLORATORY DATA ANALYSIS

2.1 Data cleaning and preprocessing

Most data sets require cleaning before use [1]:

- Replacing the marks "missing", "-", "same", ">0.2", "<10" with a number or with the value "np.nan" («Not a Number»);
- Replacing words from a textual description with values from a fixed set (for example, with possible words of medical diagnosis);
- Removal or replacement of html tags, special character codes, web addresses, emoticons, etc., although emoticons should be replaced with text in sentiment analysis tasks (see examples in functions «remove_emoji», «remove_punctuations», «convert_abbrev_in_text» in the notebook);
- in the case of receiving data from a pdf file, the text from the footers (page numbers, etc.) can get there, which needs to be deleted.

The input data is often called "Raw Data". And after cleaning: «Cleaned Data».

More complex are preprocessing operations (see many preprocessing operations in author's articles [2, 3]):

1. Transformation of formats (replacement of "float64" by "int8", "str" by "bool", etc.) to optimize memory for data storage. By default, datasets are read by the read_csv command with float64 and object data types. Therefore, they must be converted into the most economical formats. This can be done in two ways:
   - immediately when reading, specify the required data types – see «Tip 2.4» from [4]);
   - if the data formats are not known in advance, then you can first read the data, then change the logical types to "bool", and for numerical data, use «Tip 5.1» from [5].

   A special function can be used to transform text into numerical data sklearn.preprocessing.LabelEncoder (see «Tip 5.3» from [4]).

   Date is often read as "str" or "object". But it is better to save it in datetime format. The most common variant is "2023-10-30", which is coded in Python as «%Y-%m-%d» – see «Tip 5.6» from [4]).
2. Elimination of duplicates. The presence of complete duplicate rows of the table distorts the statistics, so they must be identified and removed (see. «Tips 5.5» from [4]).
3. Cleaning text data. For simple cases, the re library is used, and for more complex ones, the powerful NLTK library, described in more detail in [6]).
4. Replace very small or very large values np.inf and negative (-np.inf) values with np.nan, since it is better to work with only one kind of problematic values (see «Tip 4.5» from [4]).
5. Imputation of missing numerical data. Most machine learning models (except for Prophet and some others) require the absence of missing (np.nan)

value of data. For this, they use "imputing": "SimpleImputer", "KNNImputer" and "IterativeImputer" of the Sklearn library.

6. Formation of a new class from textual missing values. For missing text values, the value is replaced by some number that is definitely not in the table, forming a new class. For example, if all numbers are positive, then the missing numbers are replaced by (-1) (see "Tip 4.5" from [4]).

7. Filtering of abnormal values. It will be described below.

2.2 Clustering and data dimensionality reduction

After performing data preprocessing, they are often clustered, or reduced in dimension, sometimes it is necessary to look for associations, and only then the results of these operations are analyzed more thoroughly.

Data clustering – it is the process of grouping similar objects into classes or clusters based on their characteristics. The main goal of clustering is to find hidden structures in the data and highlight groups of objects that are similar to each other without a prior known distribution or classification. This is a classic task "unsupervised".

Search for *association* consists in finding connections and relationships between different elements in a data set. The main goal of *association search* is to find association rules that indicate which elements often occur together or with similar characteristics. This may include identifying items that are frequently purchased together; events that take place under similar conditions, etc. Association is a task "supervised". Usually, such methods as Apriori, Eclat, FP-growth and others are used for these tasks. For example, see the Kaggle notebook «Apriori Association Rules | Grocery Store».

Dimensionality Reduction – it is a process of reducing the amount of data or features (dimensionality) in a data set. The goal of this process is to reduce the number of features that should be considered during data analysis, retaining as much useful information as possible about the structure of the data and the relationships within it. This operation allows:

- increase the efficiency and speed of calculations;
- reduce the risk of overtraining due to the reduction data noise;
- improve visualization (for example, it is possible to reduce the multi-dimensional feature space to 2- or 3-dimensional, which can be displayed and analyzed visually).

The following methods are popular:

- Principal Component Analysis (PCA);
- t-Distributed Stochastic Neighbor Embedding (t-SNE);
- UMAP;
- Autoencoders (see below), etc.

The clustering operation is most often used in machine learning, so let's consider it in details.

Basic *functions of clustering*:

– detection of data structure to search for new important regularities;

– simplifying complex data to reduce the dimensionality of the data or to decompose it into smaller datasets or tables (for example, see the notebook);

– filling in missing data with statistical averages by cluster or class.

All clustering methods can be conditionally divided into the following types [7]:

– Partitioning methods;
– Hierarchical methods;
– Density-based;
– Graph-based methods;
– Model-based clustering.

The following are the most popular *methods of clustering*:

1. *Kmeans*. Divides the data into k clusters, where k is a predetermined number.

Work algorithm: k input data are randomly selected as centroids of future clusters. Next, the distance from each point to each of the clusters is determined, and then the point belongs to the cluster to which this centroid is the closest. At the next iteration, another point is selected among the points of each cluster, which is better suited to the role of the centroid, and all operations are repeated. It continues until the distance between the old and new centroid becomes less than a certain threshold. Other criteria can be the maximum number of iterations or inertia (sum of squared distances between objects and the centroid of their cluster) (Fig. 2.1).

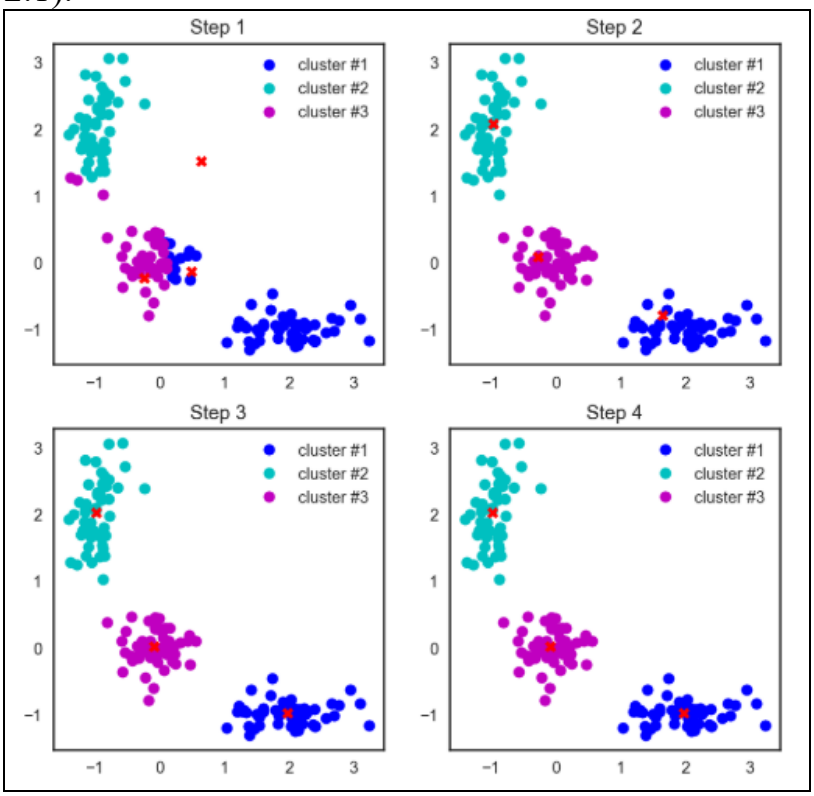


Figure 2.1 – Stages of the clustering method Kmeans

A number of approaches and metrics are used to determine the optimal number of clusters (Sklearn has nearly 20 ones). The most popular are the use of such criteria [8]:

- «Silhouette score» (sklearn.metrics.silhouette_score) – the extent to which points within one cluster are similar to each other compared to points in other clusters;

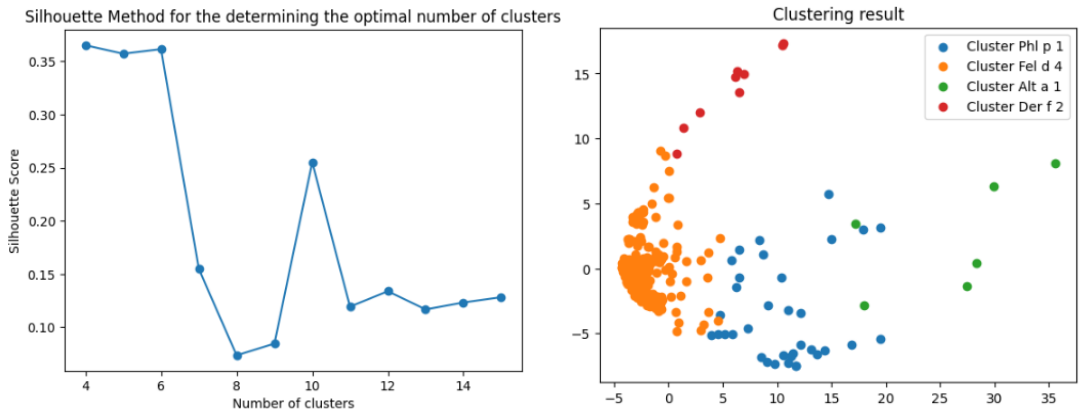
- «Calinski and Harabasz Score» (sklearn.metrics.calinski_harabasz_score) analyzes the ratio of the sum of variance between clusters and variance within clusters for all clusters;

- «Davies and Bouldin Score» (sklearn.metrics.davies_bouldin_score) – compares the distance between clusters with the size of the clusters themselves;

- «Adjusted Rand Index» (ARI);

- «Adjusted Mutual Information» (AMI).

In fig. 2.2 gives an example of analyzing the sensitivity of patients to various allergens based on real data from the author's notebook. As can be seen in fig. 2.2a, the optimal number of clusters is 4.



a) b)

Figure 2.2 – Illustration of choosing the optimal number of clusters in the Kmeans method using the Silhouette criterion for analyzing the sensitivity of patients to various allergens based on real data from the author's notebook: a) curve of the Silhouette criterion, depending on the number of clusters; b) the result of applying the Kmeans clustering method with 4 clusters

The modern version of the Kmeans method implemented in the Sklearn library uses an improved method called "Kmeans++". Its main differences are as follows:

1) instead of the distance between the point and the centroid, the square of this distance is determined;

2) only the first centroid is selected, and each subsequent one is selected taking into account the probability of being selected as a centroid, proportional to the square of the distance from the point to the nearest already selected centroid;

3) the criterion of the method is the minimization of the inertia.

KMeans is very computationally expensive and requires a lot of memory. MiniBatch is used for large data or with limited computing resources. This method does not work with all data, but only with a certain random sample. Centroids are updated after each such mini-batch. It is interesting that this method can give sometimes no worse results than the KMeans method on all data, but in much less time and can be effective for a relatively small amount of data.

To increase the speed of the method, it is recommended to perform data preprocessing using PCA.

There is an option for time series: **tslearn.clustering.TimeSeriesKMeans**. The author's notebook provides an example of clustering by this method of the exchange rate of about 80 cryptocurrencies with a capitalization of more than a billion (in US dollars) as of April 2022.

2. The DBSCAN method (Density-Based Spatial Clustering of Applications with Noise) is clustering based on data density under noisy conditions. Each point can be the centroid of the cluster (the "main" point) if there is a given minimum number of points within a certain radius from it (Fig. 2.3).

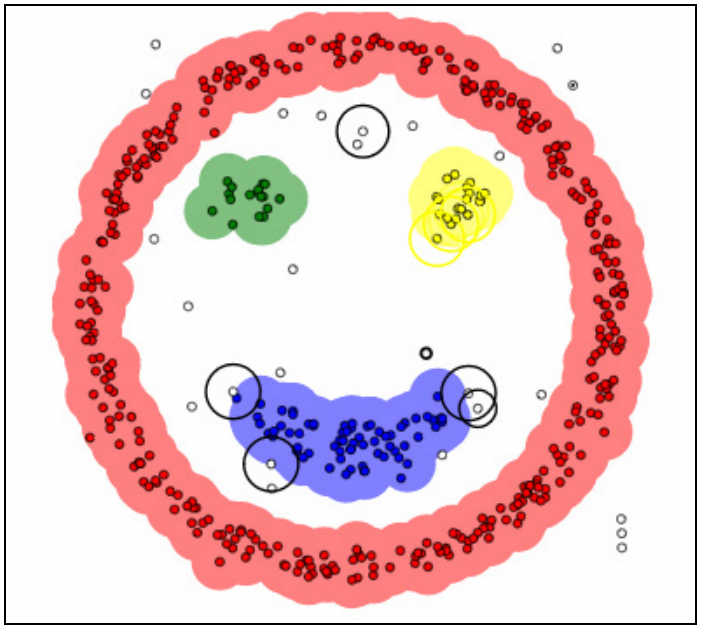


Figure 2.3 – Operation of the method DBSCAN (in dynamic)

3. *Hierarchical clustering methods*. The method of *agglomerative clustering* is the most popular of these methods: it starts with individual objects and successively iteratively pairs them together into clusters, depending on the method of aggregation (by the smallest, by the largest, by the average distance between them or others). It does not require the number of clusters. After forming a hierarchical tree of pairs of points, the cut level should be set and the method will immediately return clusters that will correspond to this level (Fig. 2.4).

```
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist
distance_mat = pdist(pca_df)
Z = hierarchy.linkage(distance_mat, 'single')
```

```
plt.figure(figsize=(10, 5))
dn = hierarchy.dendrogram(Z, color_threshold=3)
```

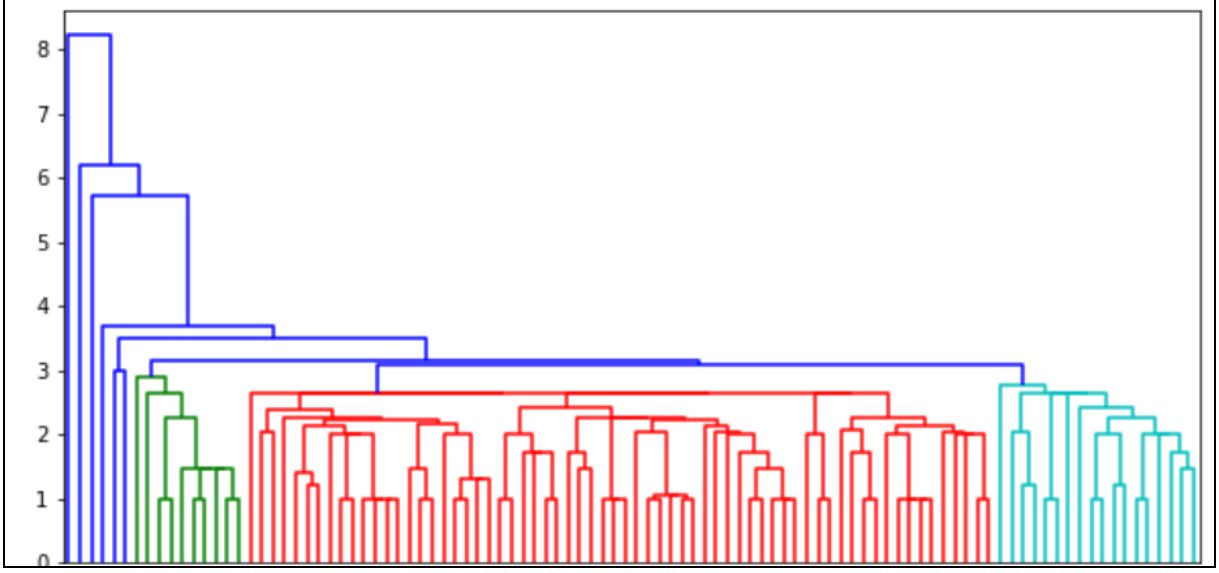


Figure 2.4 – Method of agglomerative clustering

4. *Clustering and dimensionality reduction methods* "UMAP" ("Uniform Manifold Approximation and Projection") and t-SNE ("t-Distributed Stochastic Neighbor Embedding"). The goal of both methods is to reduce the dimensionality of the data while preserving important local structures and dependencies between the data. UMAP uses distances in a low-dimensional space to find similarities between points, while t-SNE uses the probabilities of having these similarities

A bright demonstration of the possibilities of clustering and dimensionality reduction methods «UMAP», «t-SNE» and «PCA» is «Embedding Projector» for an interactive visualization of how these methods work on typical and user datasets. Also, see author's notebooks «MNIST Digits Original : 2D t-SNE with Rapids», «MNIST Original : 2D tSNE, 3D UMAP with RAPIDS» (Fig. 2.5).

The Sklearn library contains a nice comparison table for different clustering methods.

See in notebook «Titanic Top 3% : cluster analysis» clustering of Titanic passengers using 11 methods (see Fig. 2.6).

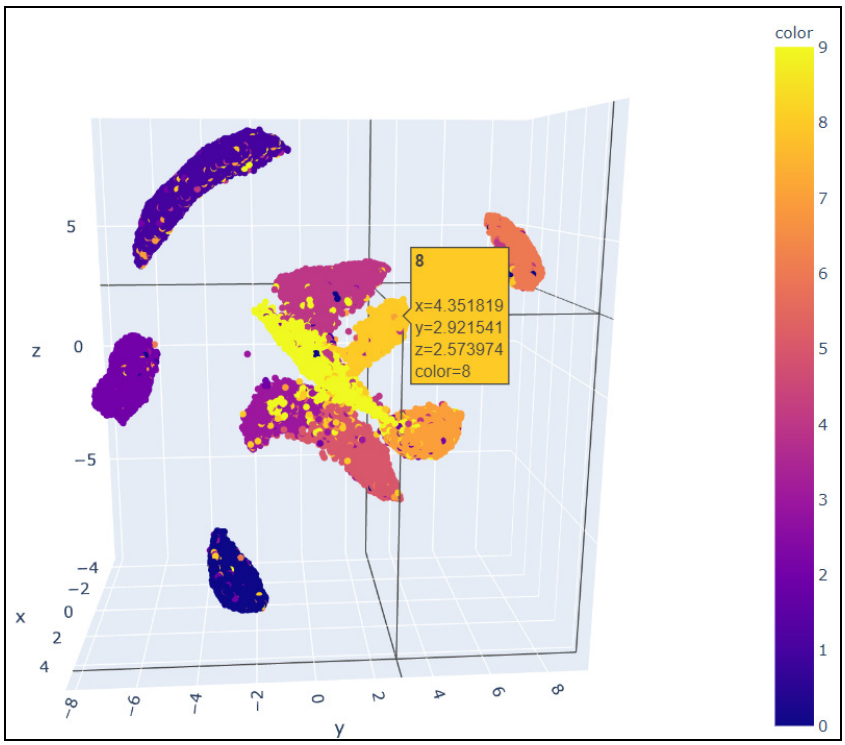


Figure 2.5 – The result of applying the UMAP clustering method to the MNIST handwritten Arabic numerals dataset («MNIST Original : 2D tSNE, 3D UMAP with RAPIDS»)

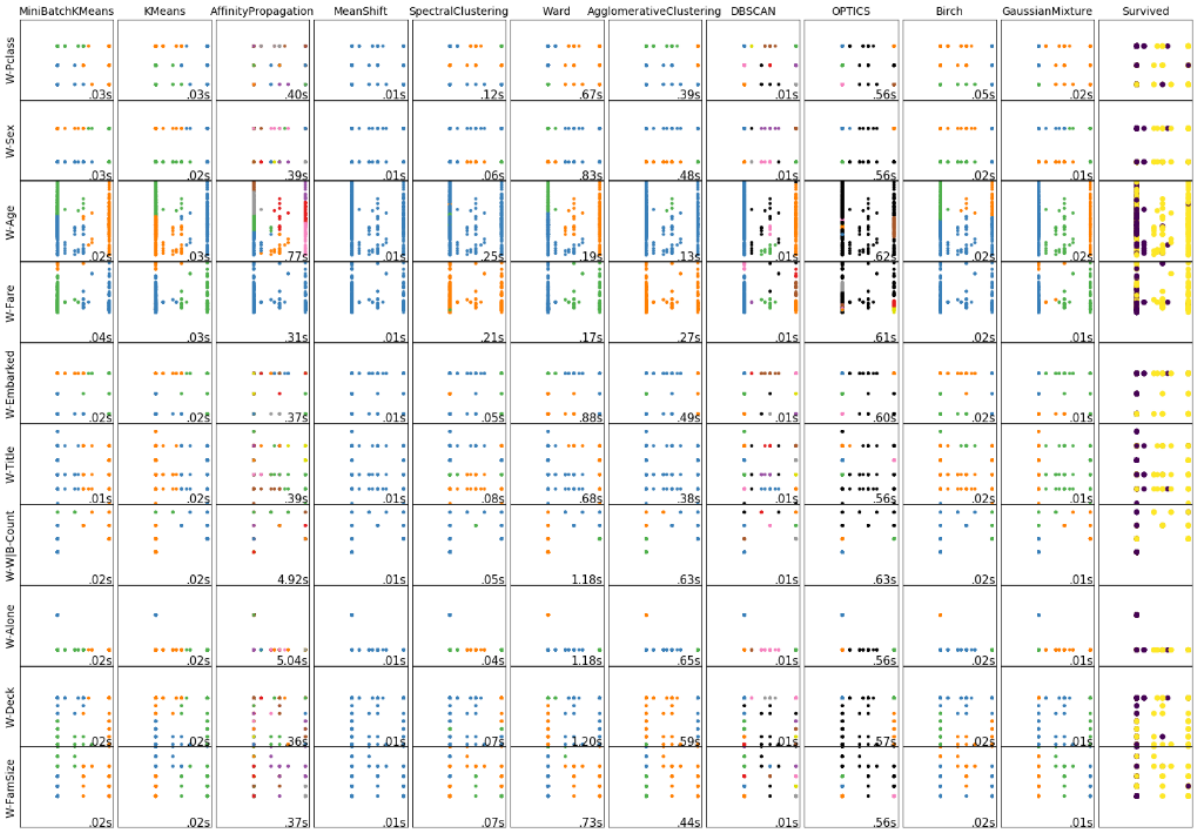


Figure 2.6 – Clustering of Titanic passengers using 11 methods and their comparison with the values of the target characteristic («Titanic Top 3% : cluster analysis»)

Notebook "Titanic Top 3%: cluster analysis" contains a universal function that immediately performs clustering by a given method with given parameters (new methods can be easily added in clustering_algorithms).

Fig. 2.7 provides an infographics of the toolkit mentioned in subsections 2.1 and 2.2 in the *S(I)* coordinate system.

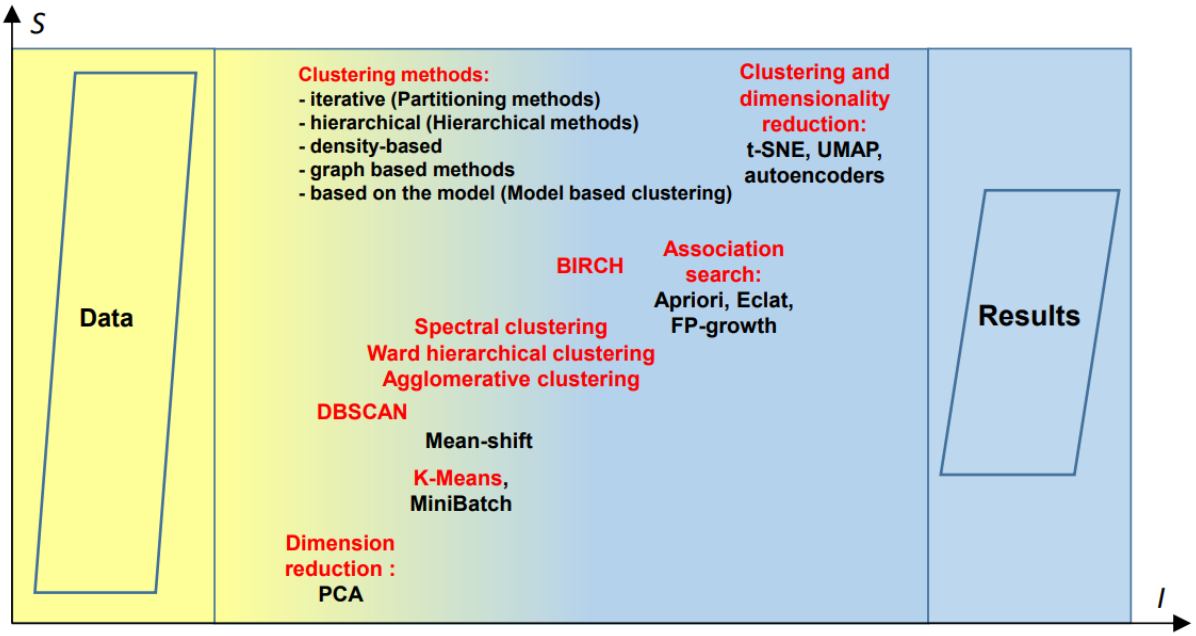


Figure 2.7 – Infographics of data preprocessing and clustering

2.3 Exploratory data analysis

Exploratory data analysis (EDA) – it is an analysis of the main properties of the data, finding general regularities, distributions and anomalies in them using relatively simple models.

The purpose of EDA is the following:

- maximum study and "understanding" of data;
- identification of main structures and systematization of data;
- detection of deviations and anomalies (outliers);
- testing the main hypotheses;
- construction and researching data using relatively simple models (regressions, decision trees).

EDA methods are applied both to all data and to their clusters and separately to training, validation and test data:

- analysis of probability distributions of variables;
- construction and analysis of correlation matrices;
- factor analysis;
- discriminant analysis;
- multidimensional scaling, etc.

Depending on the specifics of the task and research results, EDA may include the following stages:

1. *Calculation of quantitative indicators* in the dataset (see section 2.1):

- the total number of rows and columns and the number of missing values in each column (see «Tip 4.3» from [4]);

- identifying rows where there is a significant percentage of missing values in various attributes and possibly removing such rows or filling in these missing values;

- format and examples of values in each column – see "Tip 5.2" from [5].

2. *Drawing various plots* for the analysis of regularities regarding the values of each feature and their combinations, etc. (libraries matplotlib, seaborn, etc. – see notebooks Plotting with pandas, matplotlib, and seaborn, Data-Visualization-Using-MATPLOTLIB-SEABORN-PLOTLY, Visualization Matplotlib vs Seaborn).

3. *Building descriptive statistics*: characteristics are analyzed for each feature: min, max, mean, std, counts, quantiles (quartiles) (P25(Q1), P50(Q2) and P75(Q3), rarely – P05, P10, P90 and P95); number of missing values, number of unique values.

4. *Advanced primary statistical analysis* of each feature and their combinations. For each feature, a distribution law should be constructed and a hypothesis regarding its type should be checked whether it is normal (Gaussian) [9, 10].

As a rule, distribution laws are built for each class separately, for example, see the example in Fig. 2.8 from notebook) (Fig. 2.8) and analyze whether there is no need to balance them. Feature balancing (FE stage) is described in Chap. 3.

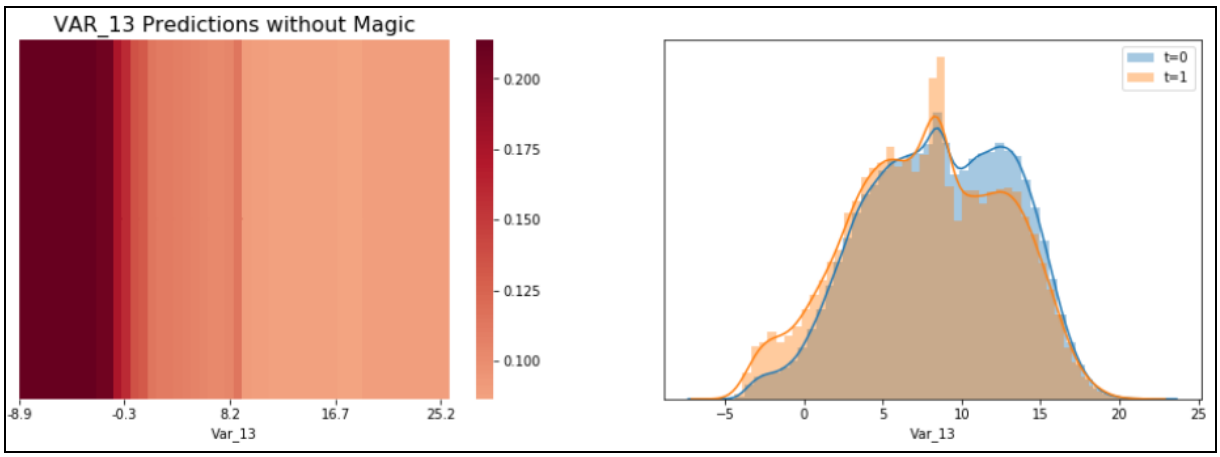


Figure 2.8 – Distribution laws for different target values t=0 and t=1 from the notebook

5. If the dataset contains training, validation and test data or at least 2 of these 3 options, then *their characteristics are compared*, first of all, distribution laws – this is a very important step that is recommended to be done every time (Fig. 2.9).

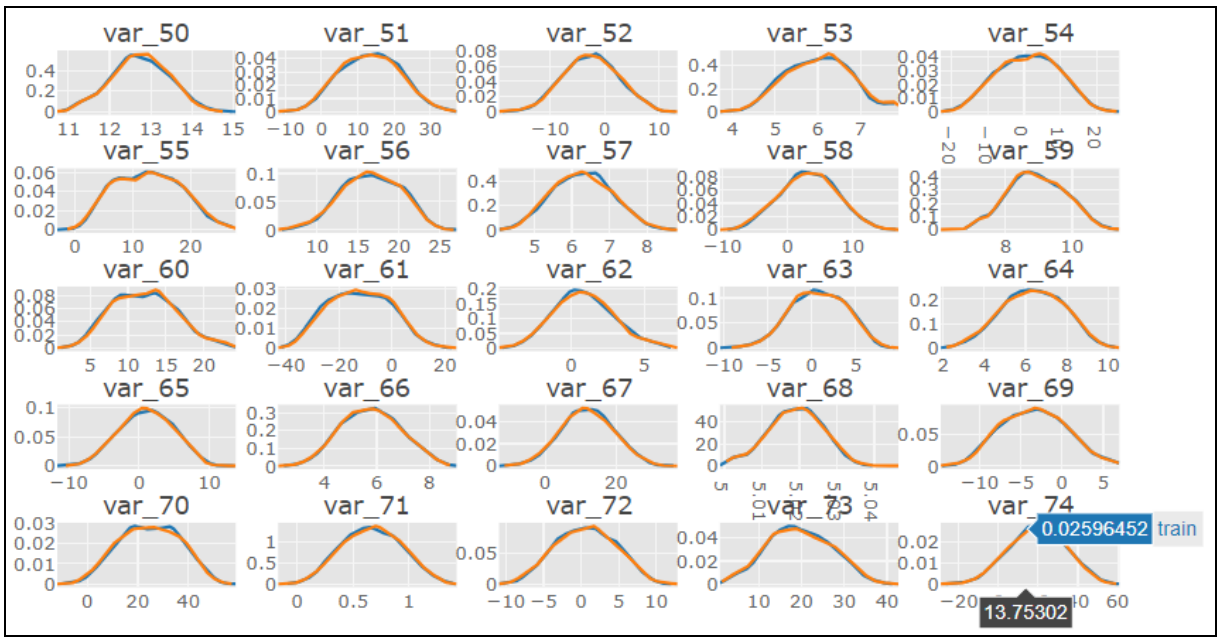


Figure 2.9 – Comparison of distribution laws training and test datasets [9]

Analysis of plots in Fig. 2.9 shows that the distribution laws are normal and very similar, so the datasets are good for further processing. The function plotly.create_distplot [9] was used to construct distribution laws. Also, you can use other functions, such as seaborn.distplot, as in notebook.

6. *Correlation analysis* (determining the presence of dependence and strength of influence between characteristics). For example, the intercorrelations of features and the detection of those that are most dependent on each other are analyzed. The result can be both a matrix of numbers and a graph of the "heatmap" type or a hybrid variant. Out of every two strongly correlated features, one should be removed if there are enough features. Sometimes, as for example, with the analysis of the "Open", "High", "Low", "Close" features of the cryptocurrency exchange rate, they are usually not removed, although they are highly correlated, as they contain very valuable information, and there are few such features in these datasets [11].

7. *Regression analysis* – construction and analysis of simple models (linear or logistic regression, decision trees, etc.) to study certain regularities between characteristics to confirm the presence and determine the nature and form of influence of one indicator on others.

8. *Analysis of outliers and data anomalies*. This can be done in three ways:

1) by quantiles, when filtering feature values where the maximum or minimum value is times bigger than P90 (or P95) or less than P10 (or P05), respectively, then all values bigger than P90 or less than P10, respectively, are discarded – see [11];

2) visually – plots are built, as a rule, using interactive plots of the plotly library, and anomalies are investigated by value or by the first and/or second change of values, news on the Internet is studied, whether it is really an anomaly that has some explanation (for example, when a large companies or the govern-

ment of the country bought something or, on the contrary, sold something, or something, or an outbreak of a disease, or a natural disaster, etc.) and then this value refers to abnormal (see Crypto - BTC : Advanced EDA, COVID in UA: Prophet with 4, Nd seasonality);

3) using special libraries for time series, which will be detailed in Chap. 5.

9. Analysis of patterns of data using methods of clustering, factor analysis and dimensionality reduction – see subsection. 2.2.

10. Analysis of the variability of features, that is, whether there is a sufficient number of different variants of the values of each feature. Features that take a single value should be removed, as they will prevent the model from learning.

11. Grouping of data by certain features and analysis of how other features are clustered relative to this one (see subsection 2.2).

12. For time series: detection of seasonality of values and identification of periods of these fluctuations, checking of series for stationarity and heteroscedasticity (see below subsection 5.4).

A more complete overview of these methods and their classification is presented in articles [2, 3] of one of the co-authors.

In addition to these methods, various system analysis methods can be used to identify important patterns between features and identify features that are most closely related: Bayesian modeling [12-16], associative data analysis, statistical modeling, etc.

The ultimate goal of EDA is to answer such questions:

1. Is the data ready for building models or does it need additional processing?

2. What models should be built to solve the given problem, according to what metrics and with what initial values of parameters and hyper-parameters?

The easiest way to build descriptive statistics for a Python dataframe is the "describe" method of the pandas library (Fig. 2.10).

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.202211	0.523008	0.381594	32.204208
std	0.486592	0.836071	13.549459	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	21.000000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	26.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	36.750000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Figure 2.10 – Statistics of the describe function of the pandas library for the data of the contest regarding the passengers of the Titanic from the author's notebook

Appendix E lists some specialized Python libraries that allow you to perform automatic IDA.

In addition, to build analytical plots according to your own script, you can use universal libraries that have built-in functions for EDA: Matplotlib, Seaborn, Plotly, Pandas (see the notebook «EDA for tabular data: Advanced Techniques»).

Fig. 2.11 presents infographics of the toolkit mentioned in the Chap. 2 in general, in the coordinate system *S(I)*.

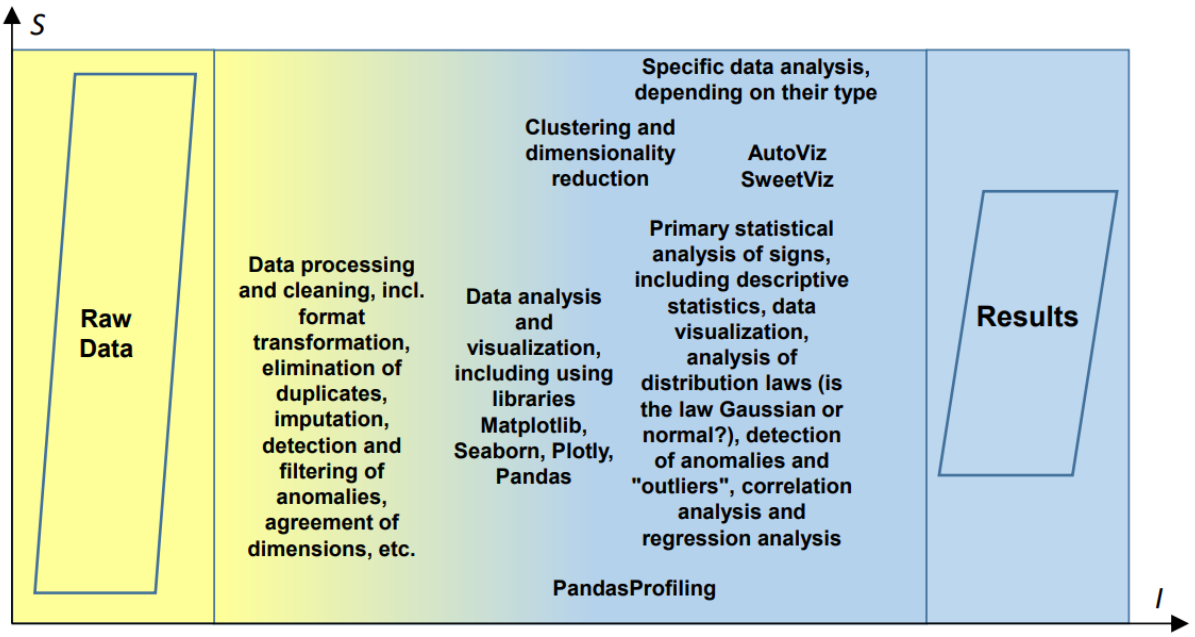


Figure 2.11 – Exploratory data analysis (EDA) infographics

Practical exercises

1) Find the median for the given list of numbers. The median is the average value of a sorted list of numbers. If the number of numbers is even, the median is the average of the two middle numbers.

Python: **np.median(np.array(numbers_list))**

But it can be useful to be able to find the median manually (Fig. 2.12).

List of numbers: 20 7 2 10 9 5 1
Sort a list and find the median:
1 2 5 7 9 10 20 => median = 7
1 2 5 7 9 10 => median = (5+7)/2 = 6

Figure 2.12 – Determining the median of the list of numbers

2) Find the mode for the given list of numbers. The mode is the meaning of the list that occurs most often. The list can have several modes if they occur equally often (Fig. 2.13).

10,12,12,23,23,23,23,38,45,45,45 => mode – 23
10,12,12,23,23,23,38,45,45,45 => modes – 23 and 45

Figure 2.13 – Determining the modes of the list of numbers

3) There are three points: A(x1, y1), B(x2, y2), and C(x3, y3). Calculate the distance between the nearest points that can form a cluster using the **AgglomerativeClustering** method with the parameter **linkage='single'**.

For example, there are three points: A(1, 1), B(2, 3), and C(5, 6). Calculate the distance between the nearest points (Fig. 2.14).

1. Distance between A and B:

$$\text{distance}(A, B) = \sqrt{(2-1)^2 + (3-1)^2} = \sqrt{1+4} = \sqrt{5} \approx 2.236$$

2. Distance between A and C:

$$\text{distance}(A, C) = \sqrt{(5-1)^2 + (6-1)^2} = \sqrt{16+25} = \sqrt{41} \approx 6.403$$

3. Distance between B and C:

$$\text{distance}(B, C) = \sqrt{(5-2)^2 + (6-3)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.243$$

Figure 2.14 – Distances between the nearest points A, B, C

As you can see on Fig. 2.14, the smallest distance is between points A and B. Also, answer: 2.236.

Possible topics of practical tasks

Topic No. 1. "Exploratory data analysis and visualization of analysis results in Python").

The purpose of the lesson is to study information technologies and Python libraries for intelligence analysis and data visualization, and to master practical skills in their application using the example of one of the Kaggle datasets or data downloaded via the API.

Lesson plan:

1. Find a dataset with real or realistic data with a description that is interesting for analysis. It is optimal to find a Kaggle dataset in which there are public notebooks with medals (at least bronze ones). Describe it.

2. Choose the Python libraries that will be used for exploratory analysis and data visualization (EDA): Matplotlib, Seaborn, Plotly, Pandas, Sklearn, etc., and specify what exactly for.

3. To review notebooks or articles regarding the dataset from point 1 using bibliographies from point 2 for its EDA. Provide at least 5 graphs, with a description of exactly what laws they illustrate and what exactly is visible on them.

4. Develop your own notebook that carries out a similar or other study (it is optimal to take an existing well (with a gold or silver medal) Kaggle notebook and adapt it to another dataset and draw conclusions about the EDA results yourself.

Samples of notebooks with EDA:
- EDA for tabular data: Advanced Techniques
- Heart Disease – Multiple Clustering by 12 methods
- MNIST Original : 2D t-SNE, 3D UMAP with RAPIDS
- MNIST Digits Original : 2D t-SNE with Rapids
- Automatic EDA with Pandas Profiling 2.9 (09.2020)
- Titanic Top 3% : cluster analysis
- Heart Disease – Automatic AdvEDA & FE& 20 models
- Autoselection from 20 classifier models & L_curves
- Biomechanical features - 20 popular models
- Suspended substances prediction in river
- AI-ML-DS Training. L1T: NH4 – linear regression

Also, you can use all notebooks from datasets COVID-19 in Ukraine: daily data or Forecasting Top Cryptocurrencies.

Test questions

1) What does the process of data preprocessing involve and why is it important before further analysis?
2) Name several methods of data cleaning and give examples of situations where they can be used.
3) What clustering methods are used to group similar objects? Give examples of their use.
4) How can data dimensionality reduction techniques help in further analysis and modeling?
5) What is exploratory data analysis (EDA) and what tasks does it solve in the data analysis process?
6) What visualization tools can be used for EDA? Give examples of plot types.
7) What are the main steps involved in the data analysis performed by the Kaggle competition winners?
8) What intelligent techniques are used for data analysis in Kaggle competitions?
9) How to identify anomalies or outliers in data during EDA, and how does this affect further analysis?
10) Why is it important to understand the distribution of the target feature during exploratory data analysis?

3.1 Main tasks and stages of feature engineering

Feature engineering (FE) is the analysis and processing of features of a dataset, including removal of uninformative ones and synthesis of new ones.

The purpose of FE is:

– detection of uninformative features that can be removed, reducing the noise of the solution and increasing the speed of the model;

– identification and removal of features that have a deterministic dependences on other;

– detection of highly correlated pairs of features;

– detection and removal of "*leaks*", when some feature contains the target value or can be obtained from it on the basis of deterministic dependencies, that is, the model can simply accurately calculate the target based on it;

- analysis of the importance of each feature using both relatively simple models (regressions, decision trees) and more complex models, in case of passing this stage again after the stage of building complex models;

- synthesis of new ("secondary") more informative features based on the values of the existing ones;

- improvement of features values.

Depending on the specifics of the task and research results, FE may contain the following stages:

1. *Synthesis of fundamentally new features* based on knowledge of the subject area. To do this, they study the content of the features, the statement of the task, study analogues of solving similar problems in GitHub, Kaggle, in professional articles in Google Scholar and in other special sources.

2. *Analysis of data types* of all features. Typically, using the function for dataframes using df.info() of the pandas library. See "Type 5.1" and "Type 5.2" from [4]).

3. *Identification of the most informative features* that satisfy the following minimum requirements:

- is not a leak;
- is not a constant;
- does not have missing values (after preprocessing);
- is not a feature highly correlated with others.

An additional requirement is the high importance of the feature, but this is discussed in the subsection 3.3.

4. *Discretization of informative features* by forming a small set (3-10) of values of a numerical feature instead of a large number or fractional values in a certain range, for example, by dividing an integer by a certain number – see of advice "Tip 5.5" in [5].

5. Formal *synthesis of new* ("*secondary*" or "*synthetic*") features from those available according to such an algorithm (see the example in "Tip 5.5" in [5]):

1) convert the value of the available informative features to a type «str»;

2) combine the values of 2, 3 or more characters through some symbol ("_"or "-"), and then:

$$df[i + "\_" + j] = df[i].astype('str') + "\_" + df[j].astype('str'); \quad (3.1)$$

3) encode the newly formed values with the numbers of the values in the list.

The most interesting thing is to use some non-linear function (multiplication, division, root, power, etc.) rather than addition, then the new feature will be fundamentally new, which most models will not be able to synthesize on their own. Example: synthesis of technical indicators of cryptocurrency in subsection 2.4 of the notebook Crypto - BTC : Advanced Analysis & Forecasting. Another interesting example is the synthesis of features in the "Google Analytics Customer Revenue Prediction" (Google Online Store) contest, where it was necessary to analyze when a user browsing the pages of an online store will finally buy something and for what amount. In Fig. 3.1 see the "pageviews/hits" attribute, which is the ratio of the number of viewed web pages divided by the total number of hits.

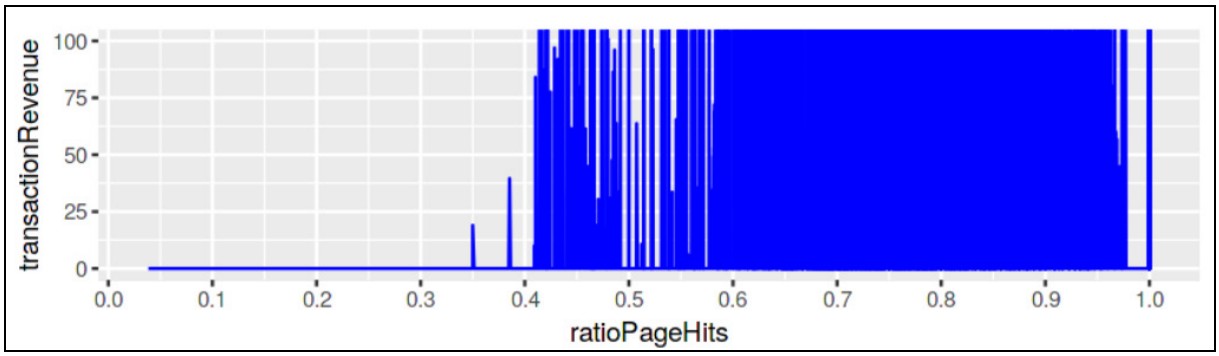


Figure 3.1 – The relationship between the total revenue from transactions in the Google online store "transactionRevenue" and "pageviews/hits" – the number of views divided by the number of operations (button clicks)

As can be seen from Fig. 3.1, it is already possible to obtain a solution to the problem with considerable accuracy, since it is clearly visible after which threshold value the client will almost certainly buy something, and up to which – not.

The FE stage also often includes some reprocessing operations to save memory by specifying the format of features and transforming formats (see clauses 1 and 2 in subsection 2.1).

6. *Reduction of the dimension (number)* of features (see subsection 2.1).

7. *Factor analysis* is the identification of hidden (latent) factors that can explain the observed relationships between features in the dataset.

8. Feature engineering can include *feature value operations* (they can also be applied at the preprocessing stage). For example, see "Type 4.4" in [4].

9. It is advisable to *divide features with date and time into separate ones*: year, month, day, hour... with different increments (quarter, season, half hour...). And the date and time itself is often removed over time to avoid overtraining.

10. *Balancing* of different target classes in the training dataset using the Synthetic Minority Oversampling TEchnique (SMOTE) method or others – see an example in the article on the analysis of coronavirus patients in Great Britain [17].

The mentioned FE operations require some creativity and are usually carried out alternately with the EDA stage, in a cycle. But there are FE operations that are already carried out with the final dataset, which at the EDA stage is defined as meeting the minimum requirements and promising for building a model. These are operations of standardization and normalization of data – they are devoted to the following subsection 3.2.

In addition, there is an important stage of FE as feature importance analysis. This requires machine learning models – either simplified (at the first stage) or – already after the stage following FE – the model building stage. We will consider these operations in more detail in subsection 3.3.

3.2 Standardization and normalization of features

Most machine learning models try to adapt to all features equally, but those features that have a greater dispersion, a greater variety of values and deviations from them will be more influential. And then the model can "overfit" for them. To avoid this, the data should be standardized (Fig. 3.2a).

The Sklearn library has a number of methods for this:

1. *Standardization* (sklearn.preprocessing.StandardScaler) (see Fig. 3.2b), which is applied to all values of the feature x, except for target, with obtaining the centered value $\tilde{x}$ according to the formula (the denominator and the numerator are optional – you cannot subtract the average $\tilde{x}$ or divide by variations σ_x):

$$\tilde{x} = \frac{x - m_x}{\sigma_x}. \qquad (3.2)$$

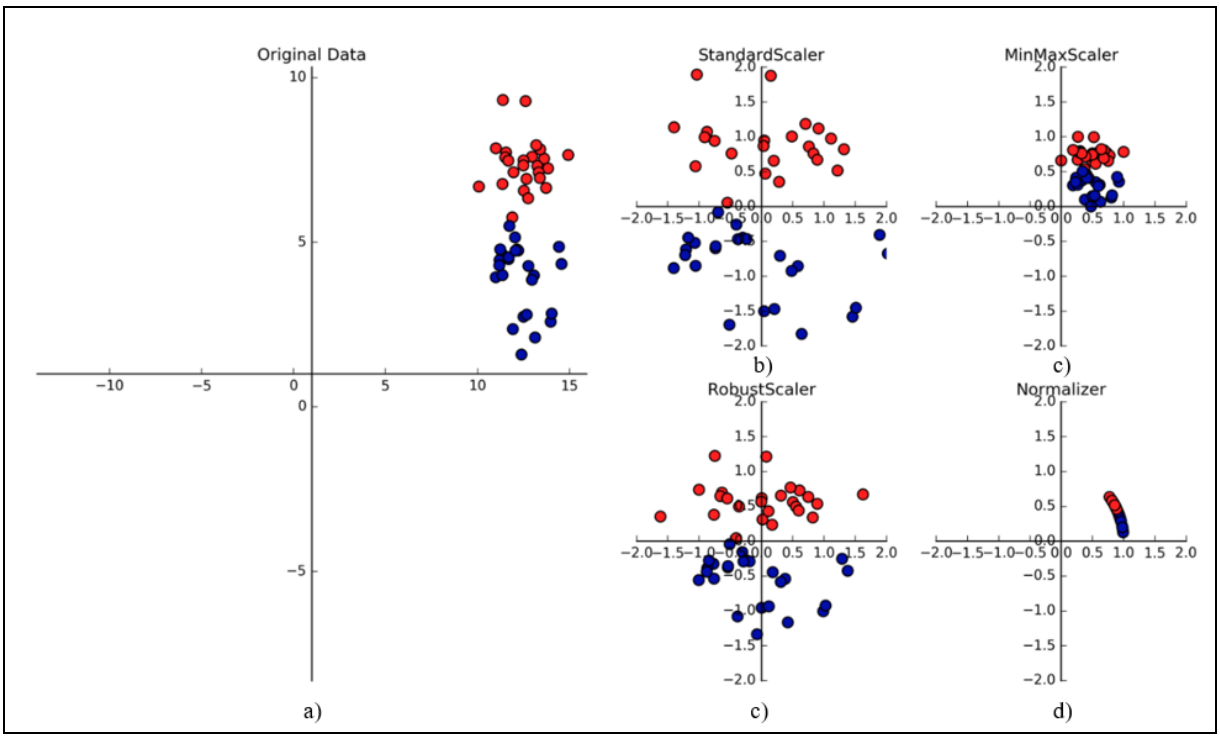


Figure 3.2 – Typical methods of standardization and normalization of features using Python libraries Sklearn – from the documentation: a) sample data; b) result of normal standardization; c) scaling result; d) the result of robust standardization; e) result of normalization

2. *Minimax scaling* (sklearn.preprocessing.MinMaxScaler) (see Fig. 3.2c) of the feature x shall be performed by its minimum x_{min} and x_{max} maximum values

$$x_m = \frac{x - x_{min}}{x_{max} - x_{min}}. \quad (3.3)$$

It is usually widely used for displaying different features on one graph in the range from 0 to 1 along the y-axis.

1. Standardization according to formula (3.2) can lead to significant data distortions, if there are significant anomalies. In such cases, robust standardization sklearn.preprocessing.RobustScaler is used (see Fig. 3.2d).

2. Some clustering methods are more effective for heterogeneous data if they are first normalized (see Fig. 3.2e). 3 normalization options are possible, depending on the selected norm: based on the sum of vector modules, based on the Euclidean norm (length of the vector), which is the default option), based on the maximum value of the vector.

Usually, all these standardization and normalization functions are adjusted (scaler.fit_transform) to training data, and (scaler.transform) is applied to both training and test data, otherwise there will be a data leak, that is, the model will become inoperable under real conditions when the test data unknown in advance.

3.3 Construction of feature importance diagrams and automation of feature selection based on Sklearn, SHAP, LIME libraries. Interpretability of models

As mentioned above, it is important not only to build a machine learning model, but also to be able to draw correct conclusions based on it. Often, an adequate model is not needed for prediction, but for providing reasonable conclusions based on it. Therefore, the interpretability of models is important. And it is often carried out on the basis of a relative comparison of the importance of features under certain conditions.

For the first application of FE, simple models are used to analyze the importance of features, for example, linear regression or decision trees from the Sklearn library, and feature importance (FI) is determined. A corresponding FI-diagram is built according to the decrease of this importance and analyzed (Fig. 3.3).

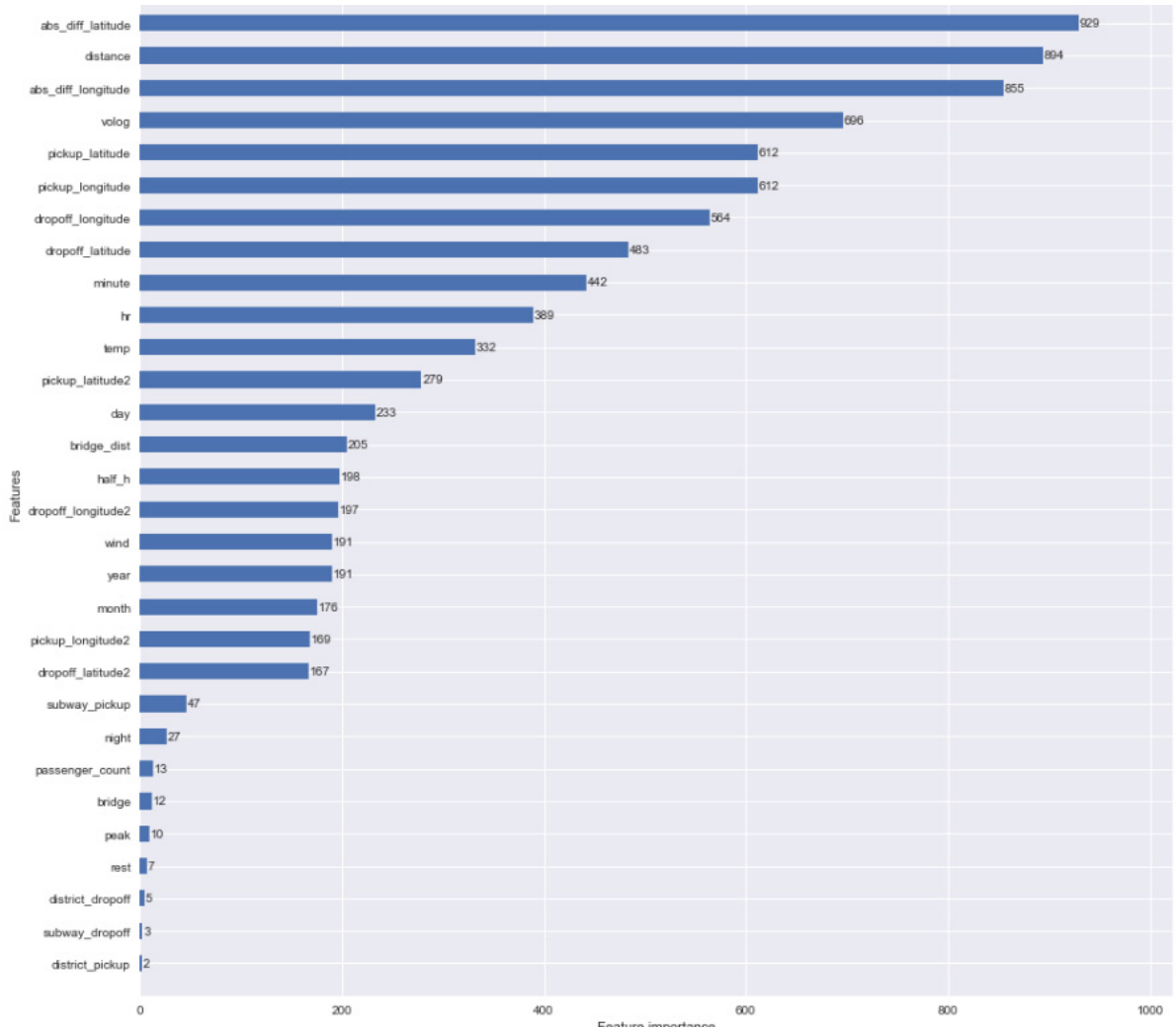


Figure 3.3 – An example of a feature importance diagram for a decision tree in the problem of predicting the cost of a New York taxi ride in the 2018 Kaggle Author's Solution Competition (see Fig. 1.15)

The following conclusions should be drawn from the diagram of the importance of features:

1. If a feature occurs very often (perhaps in all lines of the dataframe), but its importance is very low, then it should be removed as uninformative, which only introduces additional noise (see Fig. 3.3 on the feature "passenger_count" – that is the number of passengers).

2. If the feature is very important, then it should remain.

3. If the feature has low importance, but it is rare, then its value should be analyzed only by the accuracy of the model (if it increases with it, then it should be left, if not, then it can be removed).

At the FE stage, not only methods for analyzing the importance of features are important, but also automated methods for selecting the best features. For this, there are special automation methods of "Feature Selection" (FS) from the Sklearn library [18]. Most of them use a machine learning model chosen by the analyst, which will be explained in more detail in the next chapter.

1. Selection of features by the *Pearson correlation coefficient* (corr in the pandas library – for linear dependencies or features distributed according to the normal law) or *Spearman* (corr(method='spearman') in the pandas library – that is, for non-linear dependencies or features not distributed according to the normal law by law).

2. Selection of features using the *SelectFromModel* method using a specific machine learning model, usually linear.

3. The *SelectKBest method* – selects the *K* best features that have the greatest impact using a given criterion. Criteria:

- for classification problems: F-statistics, for categorical features – Chi-2 (χ2-criterion);

- for regression problems: ANOVA ("ANalysis Of VAriance"), Pearson's or Spearman's correlation coefficient, etc.

4. Feature selection using the *Recursive Feature Elimination* (RFE) method. The algorithm gradually removes less important features (a given number or percentage) until a given number is reached. Any given models are used.

5. Selection of features using *VarianceThreshold* — features with low variability (a small number of unique values) are removed. Hence, the name: threshold for variance.

Tips 5.6-5.13 in [5] provide examples of the application of these FE methods.

More effective tools for exploring and visualizing the importance of features are library methods based on game theory and on approximation by simplified models around a specific example of data. Let's consider them in more detail.

The most interesting and detailed explanations of features with good visualization are provided by methods of the SHAP, LIME library (see Appendix F).

In Kaggle, competitions are often held to solve certain problems, where in order to win, you need to be able to analyze the importance of features and be able to process and generate them (see section 1.5):

1. FE for predicting the risk of loan default from Home Credit.
2. Development of methods for automatic detection of personally identifiable information (PII) in educational data from The Learning Agency Lab.
3. Highlighting key characteristics for predicting the behavior of prosumers in the energy sector from Enefit.
4. Processing and aggregation of data from the order block and the final auction to predict stock price movements in the final minutes of trading from Optiver.
5. Isolation of important features for predicting chemical effects on various cell types from Open Problems – Single-Cell Perturbations and others.

Fig. 3.4 presents an infographics of the toolkit mentioned in section 3 as a whole, in the *S(I)* coordinate system.

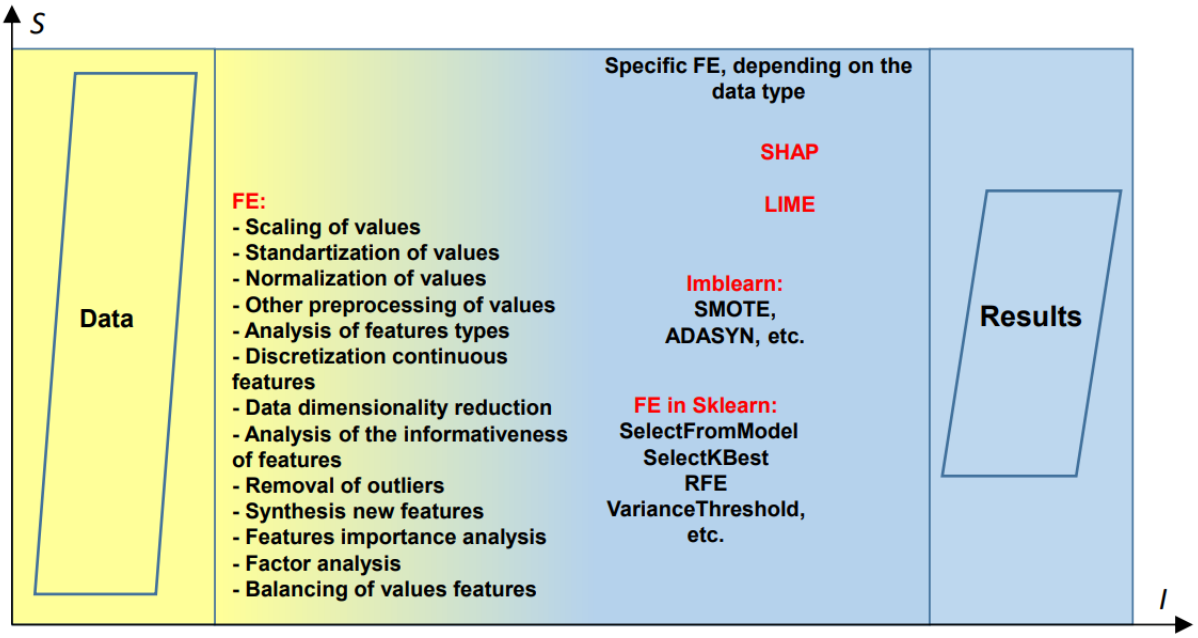


Figure 3.4 – Infographics of all the main operations and technologies of feature engineering (FE)

Possible topics of practical tasks

<u>Topic.</u> Analysis of the importance of features.

The purpose of the class is to learn information technologies and Python libraries for analyzing the importance of features and performing FeatureEngineering of data and mastering practical skills of using some of them on the example of one of the Kaggle datasets or on data downloaded via the API.

Lesson plan:

1. Select a dataset (see Chapter 1).
2. Carry out primary EDA of the dataset and define tasks for FE.

3. Characterize the stages of feature processing (which existing ones are deleted and why, which new ones are created and for what, which ones are transformed and why).

4. Give a diagram of the importance of the features of one of the models and what conclusions can be drawn from it.

5. Indicate which optimal features were selected based on the results of clauses 3, 4 (these clauses can be applied alternately and repeatedly: 3, 4, 3, 4...). Also, a combined (or complex) diagram of the importance of features, built according to several models at the same time, can be given – see example in notebooks: FE - Feature Importance – Advanced Visualization (for the classification problem)

- FE-FI for Regression Task – Advanced Visualization (for the regression problem).

Notebook samples:

The following notebooks are recommended for this work:

- for the classification task: FE - Feature Importance – Advanced Visualization;
- for the regression task: FE-FI for Regression Task – Advanced Visualization;

Also, you can see other examples of notebooks with feature importance charts:

- Autoselection from 20 classifier models & L_curves
- Biomechanical features - 20 popular models
- Suspended substances prediction in river
- WQ SB river : EDA and Forecasting
- Heart Disease – Automatic AdvEDA & FE & 20 models

Test questions

1) What does feature engineering (FE) involve and why is it important in data analysis and model building?

2) What are the main stages of feature engineering and what problems do they solve?

3) What methods can be used to synthesize new features based on existing data?

4) What is the standardization and normalization of features, and why is it important before building a model?

5) How can you build a feature importance diagram, and what is its role in choosing the most important features for the model?

6) What Python libraries can be used to automate feature selection, and what methods do they provide for this?

7) How to ensure the interpretability of machine learning models from the point of view of feature engineering?

8) What strategies are used to process categorical features during feature engineering?

9) What are the possible problems that can occur during feature engineering and how can they be avoided or solved?

4.1 Types of machine learning models and their advantages

Machine learning models are divided into the following *main classes* (see documentation of the library Sklearn):

1) Linear models (the package “sklearn.linear_model” – about 20 models and methods), the most common:
   - LinearRegression;
   - Ridge;
   - Lasso;
   - LogisticRegression;
   - Stochastic Gradient Descent;
2) Models based “Support Vector Machine” (the package “sklearn.svm” – several models, but with many variations), the main:
   - SVC;
   - LinerSVC;
3) Models based on the “Neighbors Method” (the package “sklearn.neighbors” – near 10 models), the main: NearestNeighbors;
4) Forecasting methods based on the “Gaussian Process” and when not the values themselves are forecast, but their probabilistic characteristics (the package sklearn.gaussian_process – several models, but with many variations), the main: GaussianProcess;
5) The Naive Bayes model (the package "sklearn.naive_bayes" – several models), the main: GaussianNB;
6) Decision Trees;
7) Ensembles of models:
   - ensemble of Decision Trees:
   - the baging;
   - the boosting;
   - the stacking;
   - the voting;
8) Neural Networks and their ensembles:
   - Perceptron (Sklearn);
   - MLP – Multi-Layer Perceptron (Sklearn);
   - Neural Networks of frameworks Keras, TensorFlow and PyTorch.

The last 2 classes are the most effective and popular, but in order to understand them better, you must first master the first six.

The *personal experience* of the authors and the results of studying many solutions and sources in various fields of application allow to state:

- linear models work well for small or very noised datasets;
- logistic regression works quickly in classification tasks and is often used for post-processing in solutions using Neural Networks;
- Decision Trees allow a good understanding of data patterns, have good possibilities for visualization, selection of features according to their importance;
- ensembles are the most effective, but very sensitive to the quality of the models they consist of;
- Neural Networks and their ensembles require larger datasets than linear models or Decision Trees but are more efficient for big data.

4.2 Training of machine learning models and their regularization

Each model of machine learning, for example, **name_model** (with parameters **model_params**) from the package **name_package** of the library **sklearn** using the dataframe **train** and **target** (a column of data frame as type «series») is built in the same way:

```
From sklearn.name_package import name_model
model = name_model(model_params)
model.fit(train, target)
```

and then the *prediction* (*predict*) of the values **y_pred** or *prediction of the probability* (*predict_proba*) of these values by the **model** according to the **test** dataframe is carried out in the following way:

```
y_pred = model.predict(test)
```

Next, we will give only values for each model: **name_package, name_model** and examples **model_params**.

Almost all models, except for LinearRegression, Lasso, GaussianNB, in the Sklearn library have an option for the classification problem (the word "**Classifier**" is added at the end of the name) and for the regression problem ("**Regressor**"): DesicionTreeClassifier and DecisionTreeRegressor, etc. There is no LinearClassifier, LassoClassifier, or GaussianNBClassifier model variant.

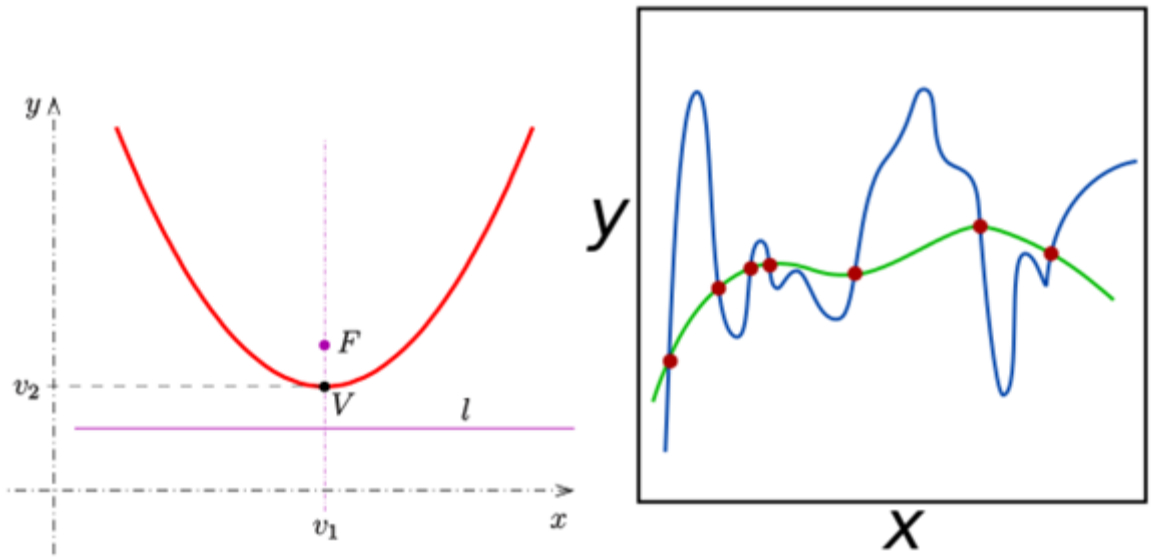
LogisticRegression is the only classification model that can't be used to solve regression problems. It is intended only for classification tasks, despite the name.

In all models, as a rule, there is a **random_state** parameter of the "int" type, which must be fixed, for example, random_state=42, or 0, 1, 2, or another integer. This ensures the *reproducibility of the results*, else calculation result can't be repeated.

And there is also the **verbose** parameter, which is usually equal to 1 (display intermediate results of calculations) or 0 (do not *display*).

All models other than LinearRegression typically have specific **model_params**. In machine learning, these are called *hyperparameters* – parameters that are not learned by themselves during model training, but define the architecture or configuration of the model. They are set before training begins and determine how the model should learn and how it should adapt to the data.

It is important to configure hyperparameters in such a way as to prevent or minimize the risk of overtraining. To do this, *regularization* is used – a technique in machine learning to reduce the values of model parameters to improve its generalization ability and avoid overtraining. It is best to explain its essence in the example of identifying polynomial regression for one feature (Fig. 4.1).



a) b)

Figure 4.1 – Examples of polynomial regression construction: a) parabola (polynomial of order n=2), which exactly passes through n+1=3 points; b) polynomials of the (m=6) and lower orders that are close to m+1=7 points

As you know, a straight line can be precisely drawn through n=2 points, that is, a polynomial of order n-1=1. Through n=3 points – a parabola, that is, a polynomial of order n-1=2 (Fig. 4.1, a), through n=7 points – a polynomial of order n-1=6 (Fig. 4.1, b, blue line). However, this very precise curve will differ significantly from the basic trend line between the known points. Theoretically, the parameters can be sign-changing and have a value of 10^6or more. Because of this, when test data with values that differ from the training dataset arrive, the model will give significantly inadequate results. To avoid this, artificial limitation of parameter values is introduced. It is necessary to limit both positive and negative values, and therefore, use 2 types of functions of the 1st or 2nd orders with the corresponding designation L1 or L2:

- L1: module function:

$$\min_w \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha||w||_1 \tag{4.1}$$

- L2: square function:

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2 \tag{4.2}$$

Almost all machine learning models have 3 parameters: L1 and L2, which take the value True or False, or one (for example, "*penalty*"), which takes the value L1, L2, or None. The third parameter is the degree of regularization (usually denoted as "*alpha*" if larger values mean more regularization, or as "*C*" if vice versa) – a positive float number. There are models with both types of regularization at the same time, for example, ElasticNet.

Let's consider how the models are trained and how the effectiveness and accuracy of their training is controlled.

4.3 Tuning of models' hyperparameters and controlling their training's effectiveness

The key stage of machine learning is actually *model training* (hence the name). The purpose of training is to achieve some *goals* (in the direction of decreasing importance):

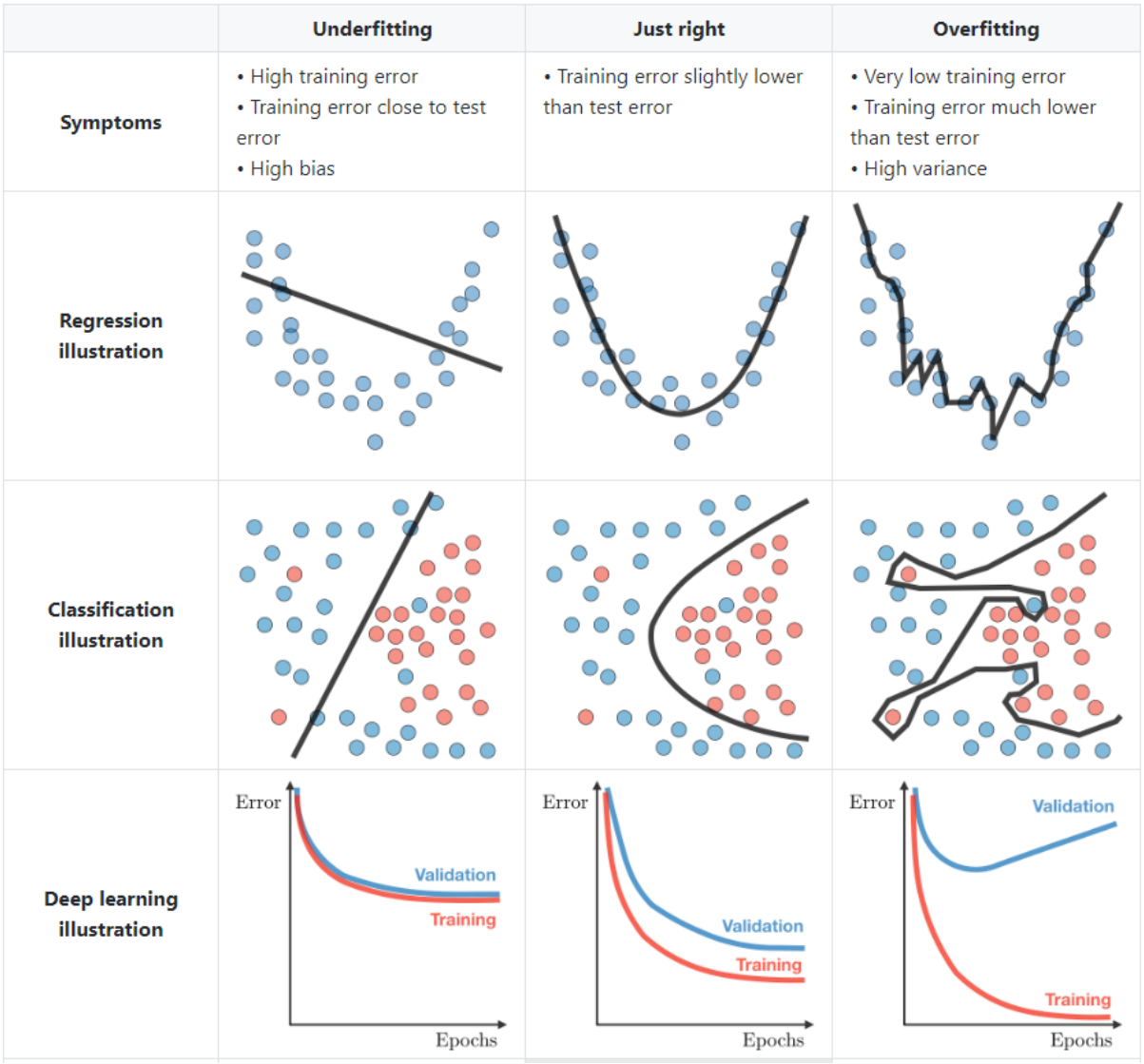
- achieving a better value of the metric on the validation dataset **val_loss**;
- there is no or minimal difference between the value of the metric on the training **train_loss** and validation **val_loss** datasets;
- shorter duration of calculations;
- lower cost of calculations, as well as lower requirements for the necessary computing power in the form of GPU or TPU.

That is, among models with the same very good values, for example, **val_loss**, you need to choose the one in which **train_loss=val_loss** (or almost so), and if it also coincides, then – the one that makes predictions faster (and training too) and is also cheaper, that is, it requires less power for calculations.

The *main goal* is to achieve a better metric value on the **val_loss** validation dataset, but it is important that the loss value on the training data is also good and does not significantly differ from **val_loss** (at least by no more than 5–10%). According to the ratio of these values, 3 types of machine learning results are distinguished (Fig. 4.2):

- *Undertraining (very bad learning)*: the value of loss is "bad" or unsatisfactory, then the value of **val_loss** no longer has a special value, or **train_loss = val_loss**;
- *Just Right or Appropriate-Fitting (good learning):* both **val_loss** and **train_loss** are good or at least satisfactory (**val_loss** should be worse than **train_loss**, but slightly, for example by 3-10%);

- *Overtraining (bad learning):* **train_loss** is good but **train_loss** is "bad" or unsatisfactory or **train_loss** is better than **val_loss** by more than 10%.



a) b) c)

Figure 4.2 – Types of machine learning model results: a) Undertraining; b) Just Right or Appropriate-Fitting; c) Overtraining

Usually, in the case of a good implementation of the models and advice from subsections 4.1 and 4.2, it is possible to achieve a good loss, and then the main problem is overtraining itself, which takes place in very many cases. That is why it is often said that the main task of machine learning is *to avoid overtraining*! There are a number of *techniques* to achieve this.

1. Using cross-validation to select training and validation datasets (see "Tip 6.1" in [5]).

2. Using the sklearn.model_selection.GridSearchCV function, which allows you to perform a complete search of model parameter options in given lists of discrete values with given cross-validation (see "Tip 6.2" in [5]).

There is also a RandomizedSearchCV option – this is the same as GridSearchCV, but without the full selection of options. Only certain random combinations are taken – not as accurate, but works much more accurately and you can try more options, also set continuous ranges of values.

First of all, you need to configure:

- the architecture of the models, if they provide for the use of different components;
- for Decision Trees: maximum depth (max_depth) and/or number of leaves (num_leaves);
- learning_rate (lr);

In addition to tuning the hyperparameters of the model itself, other techniques for improving the accuracy of models can still be effective:

- parameters of cross-validation (cv);
- changing the size or number of batches (batch);
- method of selecting random features during model tuning (random_state).

During tuning, it is necessary to analyze various combinations of parameters to find the most successful ones and form a plan for their further change. In author's notebooks (GRU & LSTM mix & custom loss - tuning by 3D visual, Stock Embedding - FFNN - upgrade & 3D, MoA: Pytorch-RankGauss-PCA-NN upgrade & 3D visual) there are nice 5D (3D coordinates + shape + color) visualizations of how different parameters affect accuracy. According to those graphs, it is possible to improve this accuracy well.

There are special methods to automate the setup process. The most popular among them are the following:

- GridSearchCV from sklearn.model_selection;
- HyperOpt from the hyperopt library;
- Optuna from the optuna library.

Good examples of using GridSearchCV and HyperOpt (Fig. 4.3) are in the notebook for predicting Titanic survivors (in the Kaggle competition).

Good examples of using the Optuna method with detailed explanations and infographics are in the notebook.

Recommended model training algorithm:

1. Based on the results of the exploratory analysis, possible ranges of values are selected.
2. Apply the HyperOpt or Optuna method and significantly narrow the possible ranges of values and the number of hyperparameters that can be changed and that have the greatest impact on accuracy.
3. GridSearchCV is used to refine the global optimum for the values and hyperparameters selected in point 2.

Although, sometimes the algorithm ends at point 2, and sometimes, with a small number of possible value options, point 2 is skipped and only GridSearchCV is used.

```
0.9338137951554497
{'booster': 'gbtree', 'colsample_bytree': 0.65, 'eval_metric': 'auc', 'gamma': 0.78, 'learning_r
ate': 0.0349, 'max_depth': 7, 'min_child_weight': 4.4, 'missing': None, 'n_estimators': 600, 'ob
jective': 'binary:logistic', 'silent': 1, 'subsample': 0.53, 'tree_method': 'exact'}
0.9371656795353692
{'booster': 'gbtree', 'colsample_bytree': 0.9500000000000001, 'eval_metric': 'auc', 'gamma': 0.6
900000000000001, 'learning_rate': 0.0325, 'max_depth': 5, 'min_child_weight': 4.475000000000000
5, 'missing': None, 'n_estimators': 815, 'objective': 'binary:logistic', 'silent': 1, 'subsampl
e': 0.685, 'tree_method': 'exact'}
100%|██████████| 10/10 [00:20<00:00,  2.07s/it, best loss: 0.9260240223818323]
best:
{'colsample_bytree': 0.8, 'gamma': 0.595, 'learning_rate': 0.0015, 'max_depth': 2, 'min_child_we
ight': 8.575000000000001, 'n_estimators': 870, 'subsample': 0.5700000000000001}
CPU times: user 20.8 s, sys: 350 ms, total: 21.1 s
Wall time: 21.1 s
```

Figure 4.3 – An example of XGB Classifier model parameter tuning by the HyperOpt method: the best parameter combinations (the last 2 out of 10 are shown) and the selected optimal model parameter combination (see last dictionary {}) (from the notebook)

When analyzing model training accuracy, it is important not only to calculate the basic metric but also to analyze the learning curve and confusion matrix.

The *Learning Curve* displays the dependence of the metric on the amount of training data or on the number of training iterations. It shows: how best to choose cross-validation parameters. Is the data evenly distributed between batches? Is there overtraining or undertraining, and which steps of cross-validation increase this risk? A learning curve helps you evaluate how a model learns and how its performance changes over time or with different data. Usually, 2 learning curves are built: the *Training Curve* and the *Validation Curve*. It is valuable to analyze not only them but also their comparison. The optimal situation is when the validation curve at the end of training is close to the training curve, but has slightly worse accuracy than it, and they both achieve good accuracy values.

Confuse Matrix or *Error Matrix* is a square matrix, the size of which is equal to the number of classes of the target feature, so it is built only in classification problems. Each row of this matrix corresponds to the predicted classes, and each column corresponds to the true classes. In each cell, the relative number of correctly predicted corresponding classes is displayed (there may also be an absolute value of this number and how much data had to be predicted in total). An ideal confusion matrix is a single diagonal matrix, that is, a matrix in which all 1s are in the diagonal, and 0s in the other cells. For a binary target, the confusion matrix is shown in Fig. 4.4.

		Predicted condition	
	Total population = P + N	Predicted Positive (PP)	Predicted Negative (PN)
Actual condition	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

Figure 4.4 – The structure of the Confusion Matrix

Error "False Positive" (FP) is also called "error of the I kind" or "false alarm", and error "False Negative" (FN) is "error of the II kind" or "missing the target". A large FP value indicates overtraining, and a large FN indicates undertraining.

These errors determine 2 more important metrics Precision and Recall, which are used to determine the effectiveness of models for binary classification:

- *Precision* or "Positive Predictive Value" (PPV)) determines how accurate the positive (target = 1) predictions of the model are:

$$PPV = \frac{TP}{TP+FP}, \tag{4.3}$$

- *Recall* or "True Positive Rate" (TPR)) determines what part of all positive (target = 1) instances the model identified correctly:

$$TPR = \frac{TP}{TP+FN}. \tag{4.4}$$

If it is important to choose a model that has balanced values of both of these metrics, then the metric F_1 (*harmonic average*) is used:

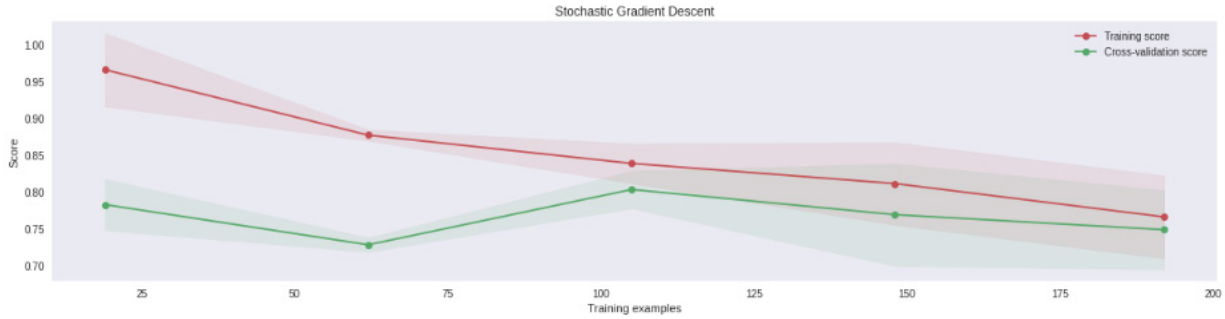
$$F_1 = 2\frac{PPV*TPR}{PPV+TPR} = \frac{2TP}{2TP+FP+FN}. \tag{4.5}$$

Metrics that are still popular in Kaggle competitions are:

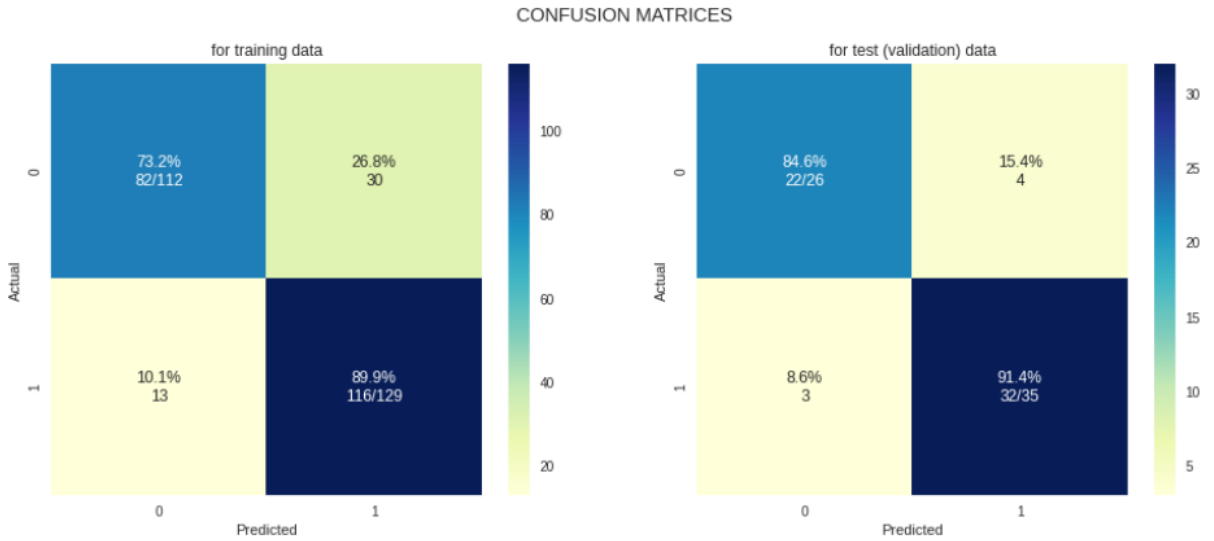
- ROC-AUC (Receiver Operating Characteristic – Area Under the Curve) – the area under the ROC curve, which graphically displays the dependence between TP and FPR shares at different threshold values in the classification model; the larger the AUC value (from 0 to 1), the better the model, where 1 indicates a perfect model and 0.5 indicates a random classification model;

- F_2 – for cases where it is more important to give more weight to Recall, especially in tasks where it is important to avoid false negative results as little as possible.

In fig. 4.5 and 4.6 are examples of learning curves and confusion matrices.



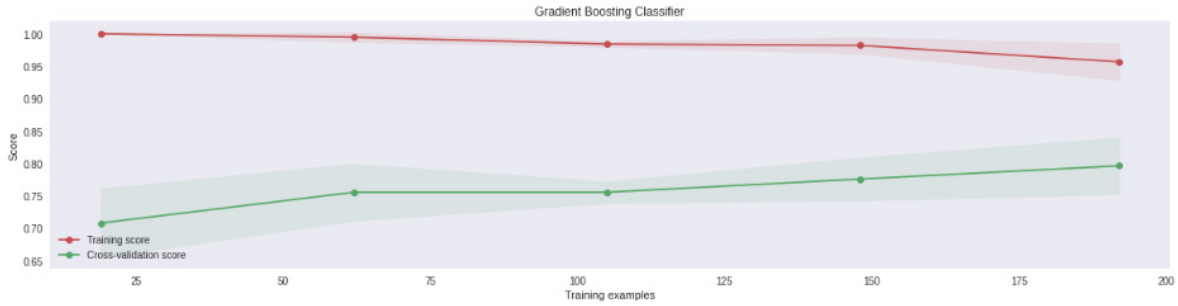
a)



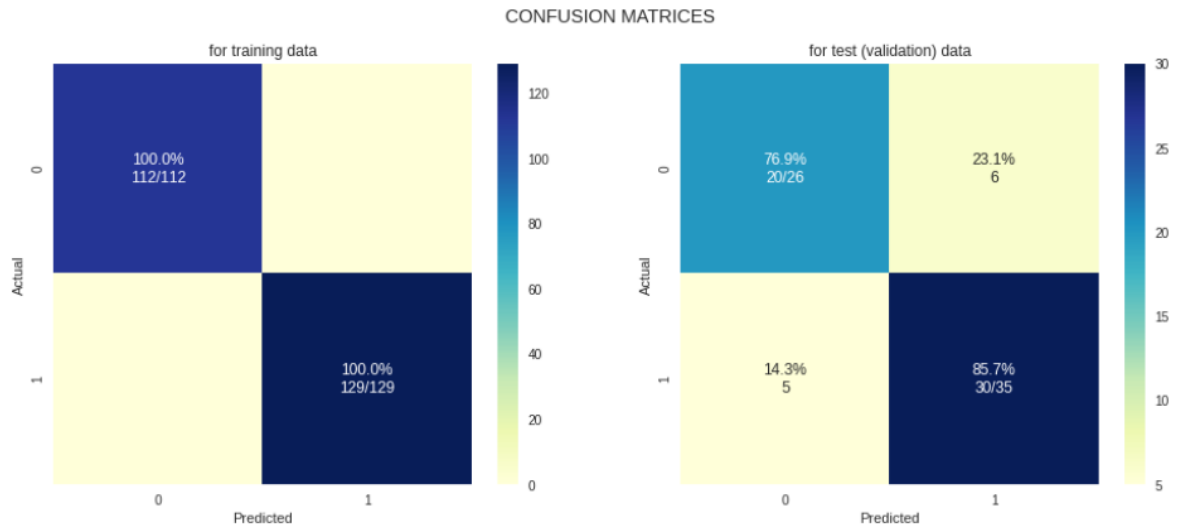
b)

Figure 4.5 – Analysis of the learning results of the "Stochastic Gradient Descent" model from the notebook: a) learning curves; b) confusion matrices

Analysis of Fig. 4.5 shows that the model learned normally, but the addition of batches worsened its accuracy, so it is advisable to revise the cross-validation parameters.



a)



b)

Figure 4.6 – Analysis of the learning results of the "GradientBoostingClassifier" model from the notebook:learning curves; b) confusion matrices

In fig. 4.6 the confusion matrix indicates clear overtraining. Moreover, values target=0 are predicted worse. To reduce overtraining, it is necessary to optimize the model parameters. The learning curve shows that the addition of batches allows you to increase the accuracy of prediction of validation data, but it is not enough, that is, it is advisable to optimize the cross-validation parameters as well.

It should be noted that the confusion matrix can be built not only for binary classification problems. In fig. 4.7 shows the confusion matrix for the problem of recognizing and classifying Arabic numerals.

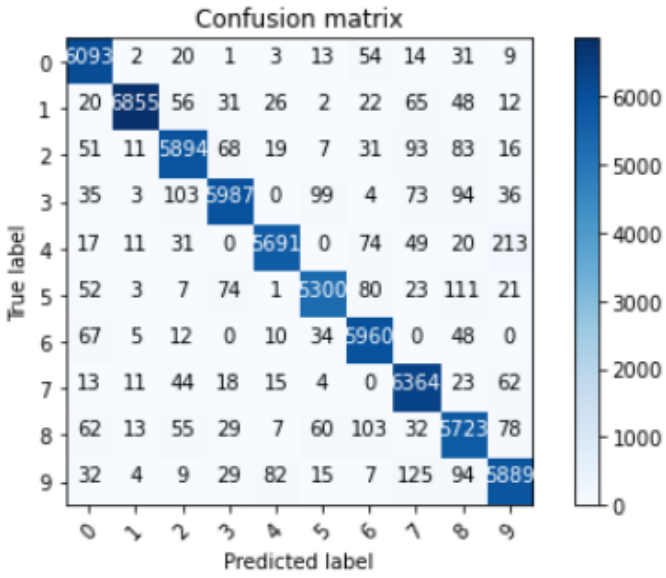


Figure 4.7 – Confusion matrix for the problem of recognizing and classifying Arabic numerals from the notebook

In fig. 4.8 presents an infographics of the toolkit mentioned in subsections 4.1-4.3 in the *S(I)* coordinate system.

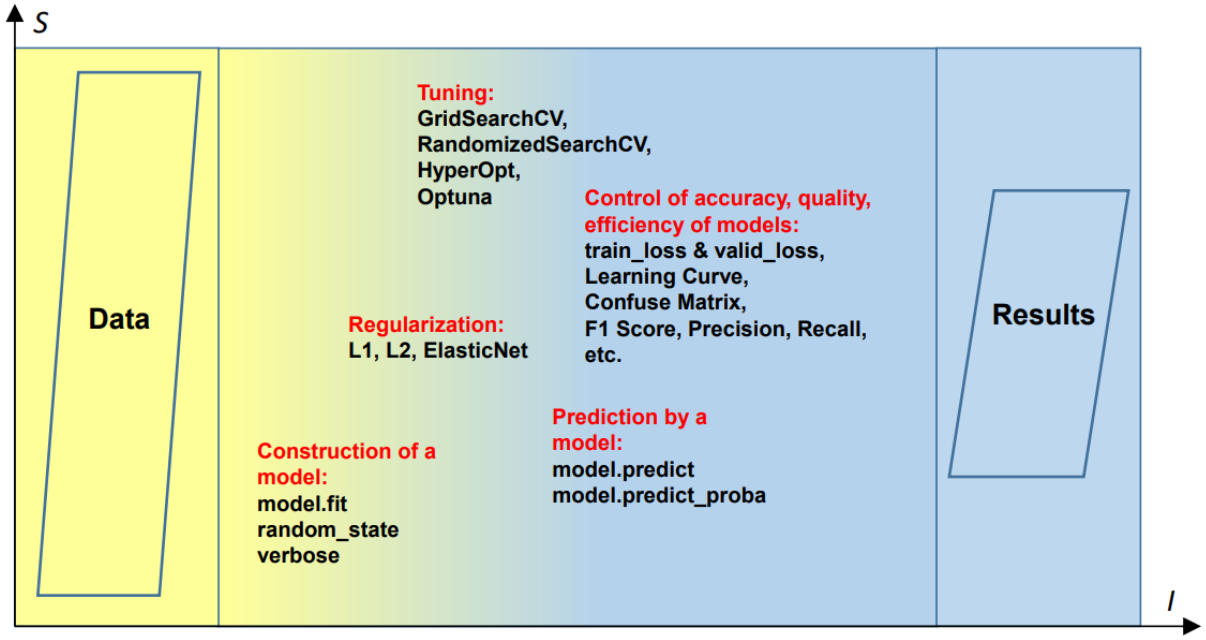


Figure 4.8 – Machine learning model tuning infographic

Let's consider the main models in more detail.

4.4 Linear Regression, Ridge and Lasso models. Logistic Regression

4.4.1 Linear Regression, Ridge and Lasso models

Linear Regression is a method that describes the relationship between *n* features (input data) $x_1 \dots, x_n$ and target y (output feature) using linear functions (Fig. 4.9).

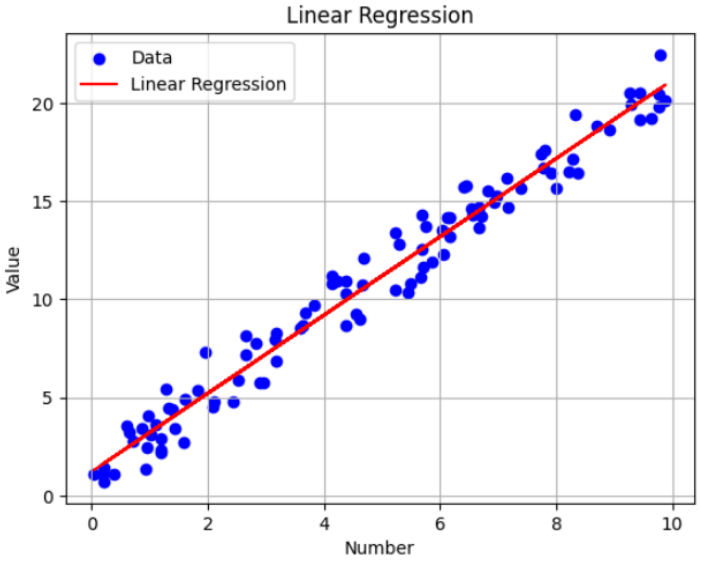


Figure 4.9 – Linear regression from notebook

It can be used both for features with integers (classification problem) and fractional numbers (regression problem), without changing the name:

name_package = linear_model

name_model = LinearRegression()

Mathematically, this model looks like this:

$$y = a_0 + a_1 x_1 + \cdots + a_n x_n + \alpha, \tag{4.6}$$

where a_0 is the constant component (shift),

$a_1, \dots, a_n$ – feature importance coefficients,

α – random noise with zero average and fixed dispersion, less than that of features.

The result is the a – models coefficients, which reflect the importance of the respective features and the value of the shift. Chapter 3 is devoted to their analysis.

Linear regression itself has no parameters, but has 2 popular varieties, depending on the type of regularization:

1. Lasso is a linear regression with L1 regularization (4.1).

name_package = linear_model

name_model = Lasso (regressor)

model_params = *alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*

The Lasso model, like linear regression, does not have a separate option for classification tasks. For them, it is used in the same form.

2. Ridge is a linear regression with L2 regularization (4.2).

name_package = linear_model

name_model = Ridge (regressor), RidgeClassifier

model_params = *alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None*

The parameters for the regressor and the Ridge classifier are the same. Higher *alpha* values mean higher regularization.

It is not recommended to set alpha = 0 in both types of models. Instead, it is better to use the LinearRegression model.

The main parameters that need to be varied to improve accuracy are:

- *tol* is the precision (positive float) at which the algorithm stops;
- *max_iter* is the maximum number of iterations if *tol* is not reached;
- *solver* is an optimization algorithm.

There is another subspecies of these regressions and types of regularization, when both types of regularization L1 and L2 are applied simultaneously. This is a model of sklearn.linear_model. ElasticNet. It is useful when some features are highly correlated, but it is not desirable to remove them (the Lasso model will leave one of them, ElasticNet – all of them).

Fig. 4.10 presents an example of prediction of 5 different datasets by different types of linear regression with different regularization.

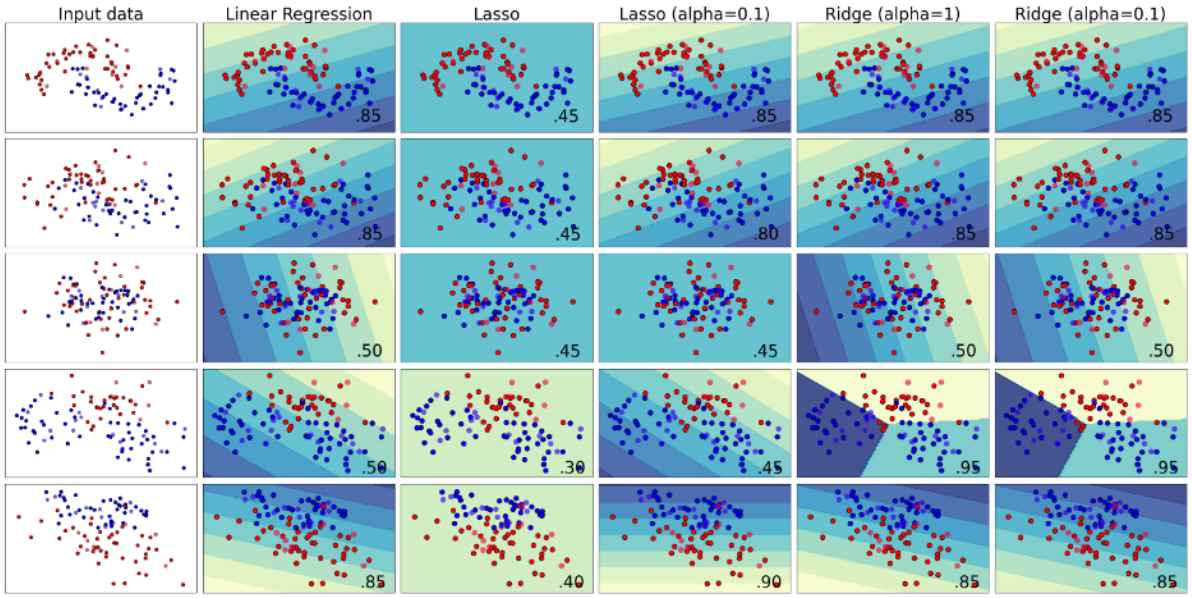


Figure 4.10 – Classification of data from 5 different datasets by different types of linear regression with different regularization (in the lower right corner – accuracy_score for test (validation) data, test data are circles outlined in black) [19]

As can be seen from Fig. 4.10, using Lasso as an example, a lower level of alpha regularization increases the accuracy of the models, but on some datasets, the accuracy of the models is very low. The threshold of minimum permissible accuracy is conventionally considered to be values greater than 0.5, since if you do not make any prediction, but simply randomly generate answers, then the probability, theoretically, will reach just 0.5. Therefore, if the accuracy is higher, then it is already better. A satisfactory level is usually 0.7. Excellent – 0.9. But there are problems where 0.9999 may not be enough, especially in Data Science competitions in Kaggle. If the accuracy is less than 0.5, then this is unacceptable!

On Fig. 4.10 one of the best models in terms of accuracy on test data is Ridge(alpha=0.1), at least on the first (0.85) and last (0.9) datasets.

4.4.2 Logistic Regression

Logistic Regression is a statistical regression method based on the logistic function, which is used to predict the target y in the case when the input variables $x_1, \ldots, x_n$ are categorical, that is, they can acquire a fixed number of values (2–10, rarely more). It is used only for classification tasks (Fig. 4.11).

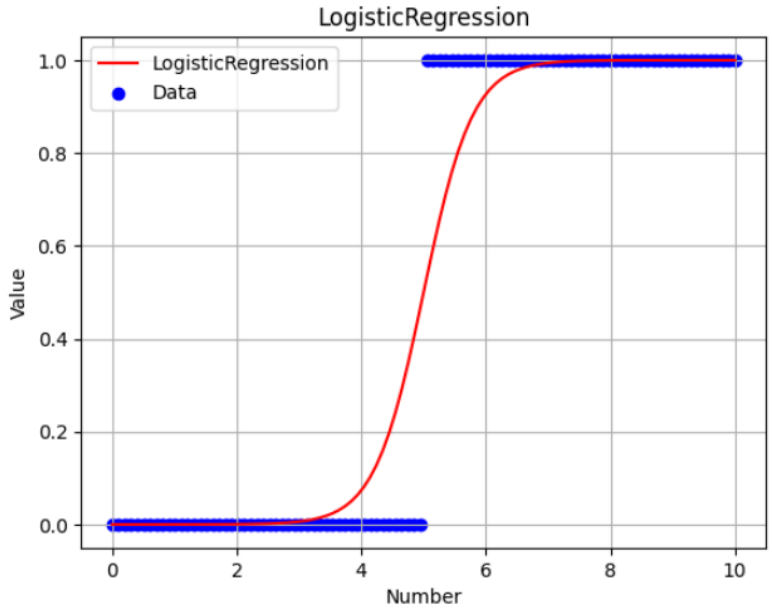


Figure 4.11 – Logistic regression from the notebook

As can be seen from the plot in Fig. 4.11 and from notebook, the probability of a given class model.predict_proba(x_values) is calculated for the input values. It is displayed on the graph in red, and then, according to simple conditions, it is assigned to a certain class. The most common option is 1 if bigger than 0.5, and – 0 otherwise (as in Fig. 4.11). It is important to realize that LogisticRegression can apply this rule to a large number of features at the same time, and for multi-class problems, when there are more than 2 classes. The main thing is that there are not too many of these classes, as a rule, no more than 10-20.

It can be used both for features with integers (classification task) and fractional numbers (regression task), without changing the name:

name_package = linear_model

name_model(model_params) = LogisticRegression (*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*)

Most often, the same parameters are changed as for the models of Ridge regression and Lasso regression (see above), but instead of *alpha*: *C* is the inverse of the regularization force (positive float): the lower the values, the stronger the regularization, i.e. the stronger the constraints of the a – coefficients.

Mathematically, this model allows you to compute the probability of each class for the input and then select the option with the highest probability *P*. For example, for binary targeting, the probability of class 1 ($Y = 1$) for one feature x is calculated using the formula.

$$P(Y = 1|X = x) = \frac{1}{1+e^{-k(a_0+a_1x)}}, \quad (4.7)$$

where a_1 is the feature importance factor x, and a_0 is shift.

The result is a -coefficients of the model, which reflect the importance of the relevant features and the value of the shift for analysis by FE methods.

Fig. 4.12 shows an example of prediction of 5 different datasets by logistic regression with different parameters.

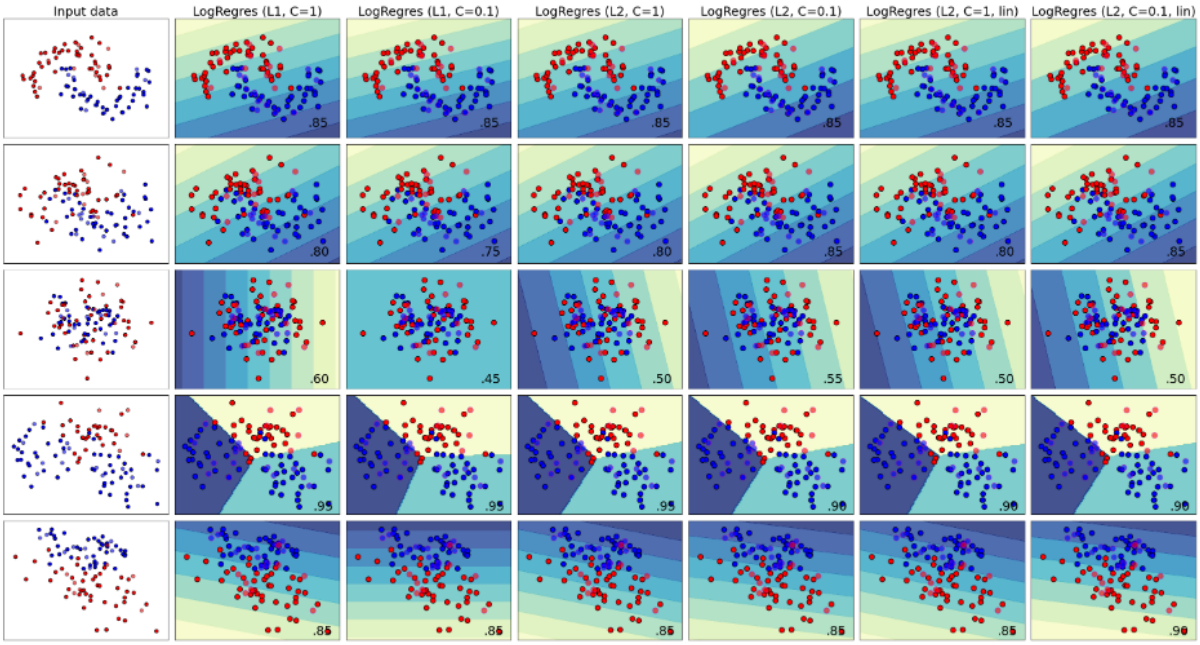


Figure 4.12 – Classification of data from 5 different datasets by Logistic regression with different parameters (in the lower right corner – accuracy_score for test (validation) data, test data are spaces circled in black) [19]

On Fig. 4.12 one of the best models in terms of accuracy on test data is LogisticRegression(L1, C=1). It is the best on the first (0.85), third (0.6) and fourth (0.95) datasets. In second (0.8) and fifth (0.85) it ranks second. In addition, it can be seen that regularization and the choice of metric (L1 or L2) gives different results. The first dataset is invariant to parameter changes, the second dataset is the choice of L2 regularization with stronger regularization (lower C values) increases accuracy. On the third, L1 gives better accuracy, but still low (0.6). In fourth, both L1 and L2 are better. On the 5th, a good accuracy of 0.9 is provided by L2 with stronger regularization (C=0.1) and the solver='liblinear' optimization method (see the "lin" parameter on the graph). Conclusion: there is no universal advice, in each case you need to carry out individual diligent tuning of the model.

4.5 SGD, SVM, k-NN, GP, NB models

4.5.1 Stochastic Gradient Descent

Stochastic Gradient Descent is an iterative method for optimizing gradient descent using stochastic approximation. It is used to speed up the search for a target by using a limited sample (batch) of training data, which is selected randomly at each iteration. It can be considered as a stochastic approximation to the gradient descent optimization method, the actual gradient for the entire training dataset is replaced by its score (Fig. 4.13). The method significantly reduces the volume computing, allows to work with big data when data is loaded in batches.

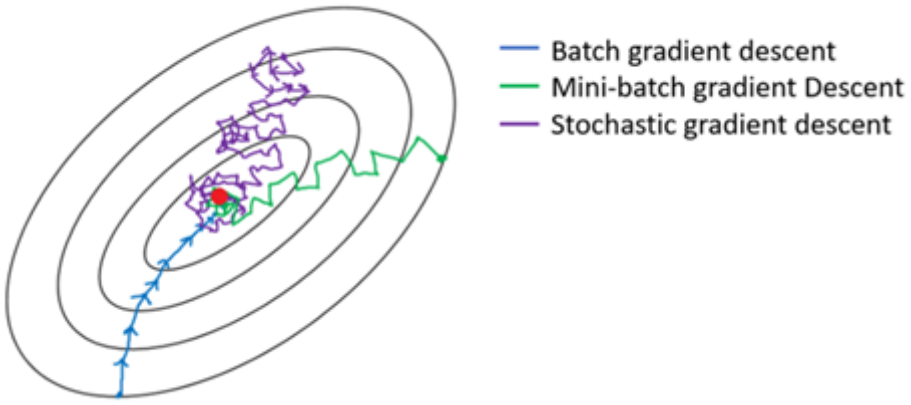


Figure 4.13 – Illustration of the Stochastic Gradient Descent method from the article

name_package = linear_model
name_model(model_params):
- SGDClassifier(*loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*);
- SGDRegressor(*loss='squared_error', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, random_state=None, learning_rate='invscaling', eta0=0.01, power_t=0.25, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, warm_start=False, average=False*).

The main popular parameter that is varied to improve accuracy, in addition to the above-mentioned *tol*, *alpha*, *max_iter*, is *learning_rate* – the speed of learning at each epoch (iteration).

An example of predicting 5 different datasets will be in the next paragraph, along with other methods.

4.5.2 Support Vector Machine

Support Vector Machine is a method of data analysis using a model that assigns new data to one or another category (class, cluster) by drawing the widest "corridor" between support vectors, which are the "walls" of neighboring classes in multidimensional space, and then assigns the data to each of these classes (Figure 4.14).

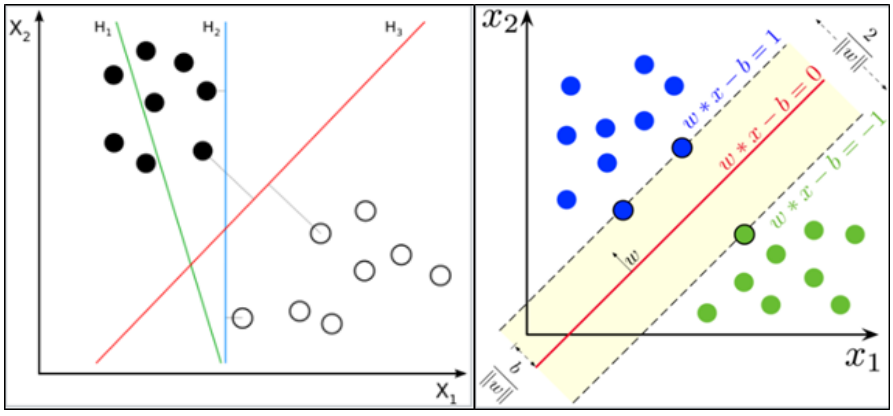
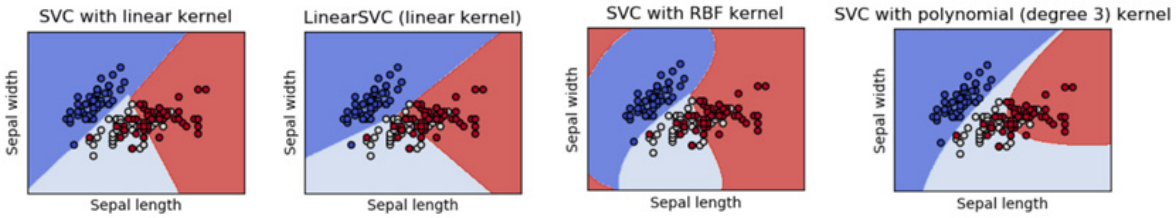


Figure 4.14 – Illustration of drawing a "corridor" between support vectors ("walls" of this "corridor") in the method of support vectors for a two-dimensional plane in the case of two input features

To complicate the algorithm, a preliminary nonlinear transformation of data is carried out using one of the types of *kernels* (Fig. 4.15). In addition to those shown in Fig. 4.15, kernels can also be: "sigmoid", "precomputed".



a) b) c) d)

Figure 4.15 – Types of kernels of the Support Vector Method with illustrations of their application from documentation: a) the linear core of the "linear" of the SVC method; b) the linear core of the LinearSVC method; c) "rbf" (radial basis functions); d) 3rd-order polynomial kernel "poly"

The type of kernel is a hyper parameter.

The LinearSVC method is similar to SVC(*kernel*="linear"), but uses a slightly different model and works better for larger datasets.

name_package = svm

name_model(model_params):

- Classifiers:

- SVC*(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*);

- LinearSVC(*penalty='l2', loss='squared_hinge', *, dual='warn', tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*).

- Regressors:

- SVR(**, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1*);

- LinearSVR(**, epsilon=0.0, tol=0.0001, C=1.0, loss='epsilon_insensitive', fit_intercept=True, intercept_scaling=1.0, dual='warn', verbose=0, random_state=None, max_iter=1000*).

The main parameter that is varied to improve accuracy, in addition to the above-mentioned *tol*, *C*, *max_iter*, in the SVC and SVR models is *kernel*.

Fig. 4.16 shows an example of prediction of 5 different datasets by the Stochastic Gradient Method and the Support Vector Method with different parameters.

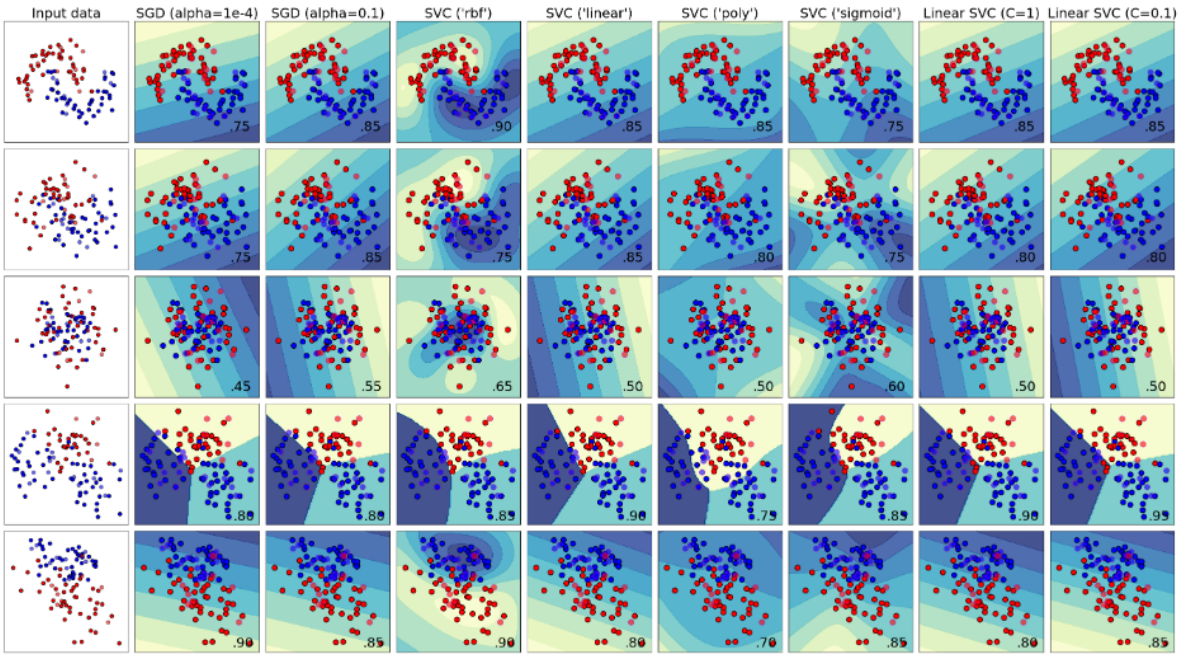


Figure 4.16 – Classification of data from 5 different datasets using the Stochastic Gradient Method (SGD) and the Support Vector Method (SVC) with different parameters (in the lower right corner – accuracy_score for test (validation) data, test data are spaces circled in black) [19]

As shown in Figure 4.16, increasing regularization (larger alpha) for the Stochastic Gradient (SGD) model increases accuracy on the first 3 datasets. For the SVC model, the "RBF" kernel provides the highest accuracy for all datasets except for the second one, where the "linear" kernel is the best. Increasing regularization (less than *C*) for the LinearSVC model increases accuracy on the last 2 datasets. In general, the best in terms of accuracy on test data are:

- on the first dataset (0.9): SVC with the kernel "RBF";
- on the second dataset (0.85): SGD (alpha=0.1), SVC with the kernel "linear";
- on the third dataset (0.65): SVC with the "RBF" core;
- on the fourth dataset (0.95): LinearSVC (C=0.1);
- on the fifth dataset (0.9): SGD (alpha=10-4), SVC with the kernel "RBF".

So, the best in this example may be considered SVC with the kernel "RBF" and LinearSVC (C=0.1).

4.5.3 K-nearest neighbor method

K-nearest neighbor method (KNN) is a simple non-parametric classification method, where objects with the shortest distance are selected to classify objects in the multidimensional feature space, they are allocated to a separate class. Objects of different classes are analyzed at a given distance *k* and the object is attributed to the one that is the most numerous among them.

Fig. 4.17 provides an example that explains the essence of the method. The test sample (green circle) should be classified as either blue squares or red triangles. If k = 3 (circle of a solid line), then it belongs to red triangles because there are 2 triangles inside the inner circle and only 1 square. If k = 5 (dashed line circle), then it is referred to as blue squares (3 squares versus 2 triangles inside the outer circle).

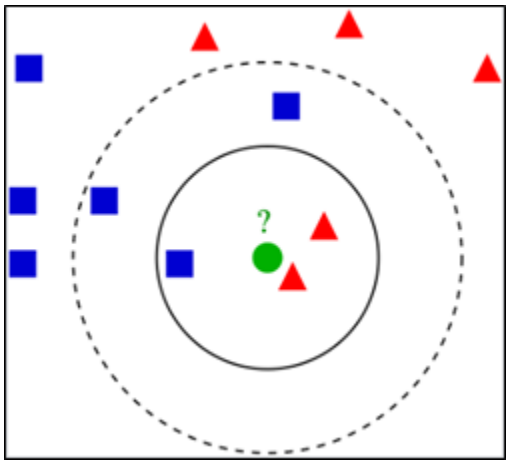
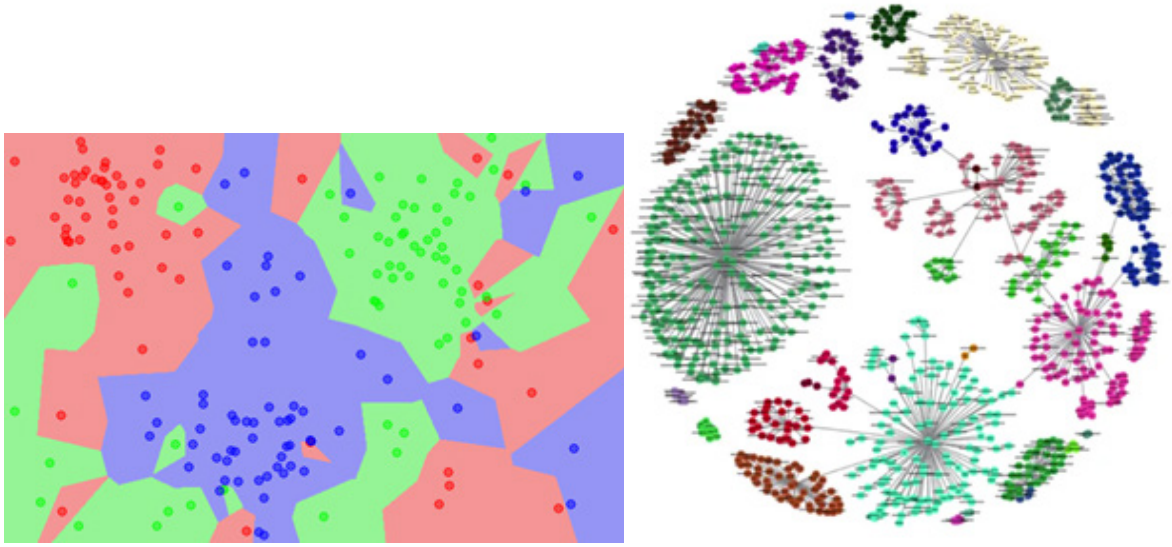


Figure 4.17 – Illustration to explain the principle of operation of the k-nearest neighbor method

Fig. 4.18 shows examples of how the method works.



a) b)

Figure 4.18 – Examples of k-nearest neighbor method: a) example1; b) example2

name_package = neighbors
name_model: KNeighborsClassifier, KNeighborsRegressor
model_params (*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*)

For the classifier and the regressor, the parameters are the same, except for the first one, which specifies the "neighbors" and it is the one that needs to be configured for the model, first of all:

- For the classifier: *n_neighbors* – positive natural int;
- For a regressor: *radius*is – the radius in which you need to look for "neighbors" – positive float.

The *weights* parameter allows you to set different weights for points, depending on the distance to the specified one.

It has advantages and disadvantages, as in clustering methods, since their principles of operation are similar. It is effective for data in the field of economics when there are many different factors that are poorly tracked, and there may not be a single adequate model. You just need to cluster the data, and then make predictions using the same algorithm.

This model does not have a *random_state* parameter.

An example of predicting 5 different datasets will be in one of the following paragraphs, along with other methods.

4.5.4 Forecasting methods based on the Gaussian process

Forecasting methods based on the Gaussian process predict not the values of the data themselves, but their probabilistic characteristics (average value and mean square deviation). The prediction is probabilistic, and therefore empirical confidence intervals can be easily calculated (Figure 4.19).

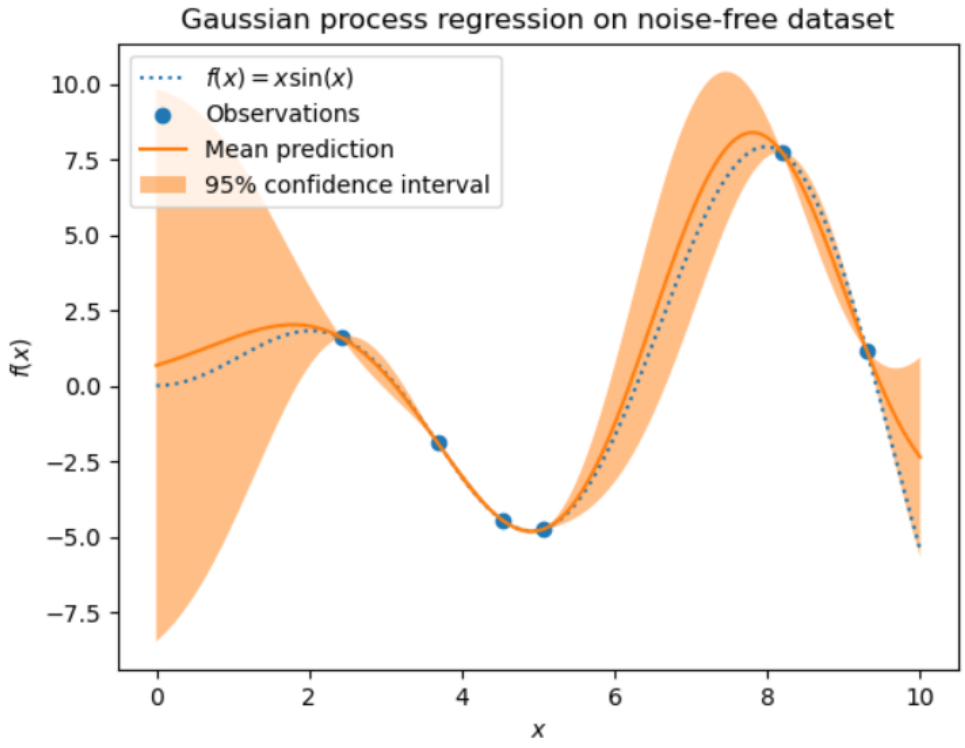


Figure 4.19 – An example of a prediction method based on the Gaussian process with documentation

You can specify kernels for additional data transformation. The method works with all data at once, so it is not suitable for big data.

name_package = gaussian_process
name_model(model_params):
- GaussianProcessClassifier(*kernel=None, *, optimizer='fmin_l_bfgs_b', n_restarts_optimizer=0, max_iter_predict=100, warm_start=False, copy_X_train=True, random_state=None, multi_class='one_vs_rest', n_jobs=None*);
- GaussianProcessRegressor(*kernel=None, *, alpha=1e-10, optimizer='fmin_l_bfgs_b', n_restarts_optimizer=0, normalize_y=False, copy_X_train=True, n_targets=None, random_state=None*).

The main popular parameter, which is varied to improve accuracy, is *the kernel* – (RBF, ConstantKernel, DotProduct, ExpSineSquared, Matern, possible combinations of ConstantKernel and DotProduct), which defines the covariance function of Gaussian processes.

An example of predicting 5 different datasets will be in the next paragraph, along with other methods.

4.5.5 Naive Bayes model

Naïve Bayes Classifier is a probabilistic classifier that uses Bayes' theorem to determine the probability of data belonging to one of the classes, usingthe "naive" hypothesis about the statistical independence of features (Fig. 4.20).

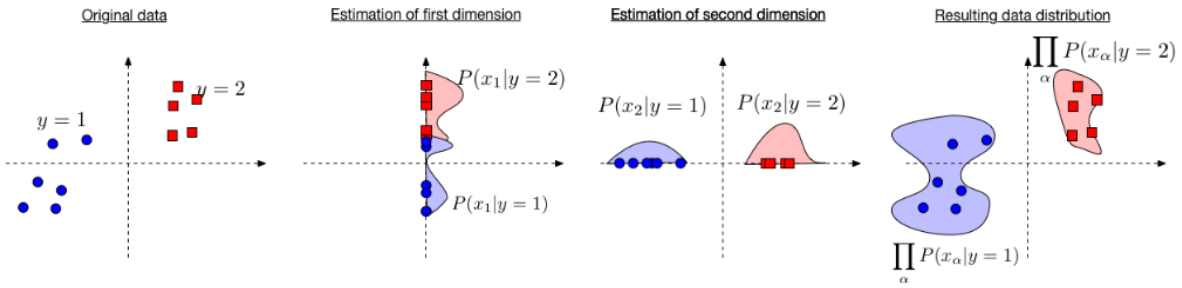


Figure 4.20 – Example of work of the "naive" Bayes model

The method has high speed, minimal memory usage and can give results when other methods simply cannot run. However, it is effective only if the hypothesis about the independence of features is really similar to the true one. At the first stage of the development of spam filtering algorithms, it was this method that gave good results, and then spammers learned to add useful text to the title and content, and this method ceased to be effective.

name_package = naive_bayes
name_model(model_params): GaussianNB(*, *priors=None, var_smoothing=1e-09*).

The method can be run without parameters: GaussianNB(). Interestingly, this model does not even have a random_state parameter .

Figure 4.21 shows an example of prediction of 5 different datasets by the k-nearest neighbor method, a classifier based on Gaussian processes, and a naive Bayesian classifier with different parameters.

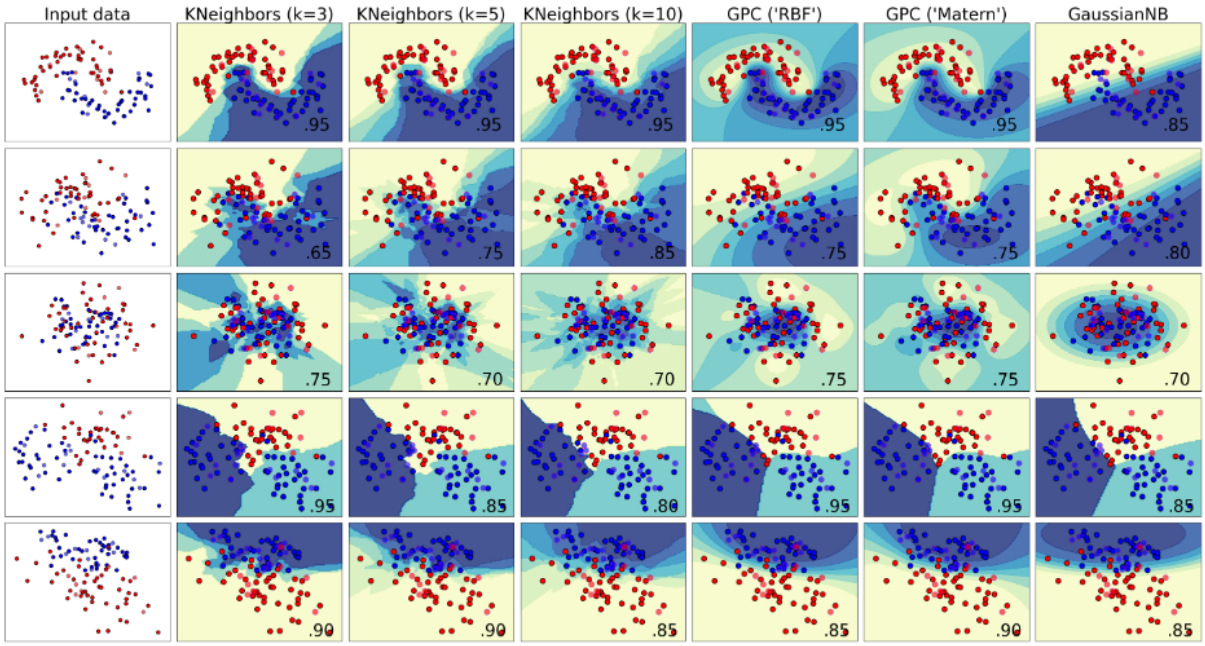


Figure 4.21 – Classification of data from 5 different datasets using the k-nearest neighbor method (KNeighbors), a classifier based on Gaussian processes (GPC) and a naïve Bayesian classifier (GaussianNB) (in the lower right corner – accuracy_score for test (validation) data, test data are spaces circled in black) [19]

As shown in Fig. 4.21, the smaller the k in the k-nearest neighbors method (KNeighbors), the higher the precision, which is quite expected. In fact, **bigger** k is an analogy of a higher level of regularization. And for the second dataset, it works, because the best model is KNeighbors with k=10.

In the GPC classifier, the best results are provided by the Matern core.

A model based on a naive Bayesian classifier is not the best on any dataset.

So, the best in this example are KNeighbors with k=3 and 10 and GPC with the Matern kernel.

4.6 Decision Trees. Comparative analysis of models on an example

4.6.1 Decision Trees

Decision Tree in Machine Learning is a structured decision-making model based on branching conditions if... then... else, which looks like a "tree" and is made up of nodes, branches, and leaves. Each node is a solution for a specific subset of data, the branches indicate the direction to be taken to reach the new node or the final result in the form of leaves. Usually, you specify the maximum amount of data that can be in a leaf. This sets the condition by which the nodes differ from the leaves.

Decision trees are handy for understanding patterns, for visualizing a solution that is easy to understand. To avoid overtraining, conditions of the “pruning” are set for trees. Theoretically, a tree with a depth *of n* would have 2n leaves. For example, if the depth is 6, then it would be 64 leaves. And then one of two "cropping" options is specified: or at a depth of 6 require, so that there is no more, for example 40 < 64 leaves, or if the number of leaves is 64, it is required that the maximum depth does not exceed, for example, 8 > 6. And then the algorithm is forced to build a "pruned" tree, which will be much more generalized than an "unpruned" one. Model parameters:

name_package = tree
name_model(model_params):
- DecisionTreeClassifier*(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*),
- DecisionTreeRegressor*(*, criterion='squared_error', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0*)

The parameters of the classifier model and the regressor differ only inthemetric (criterion):

- Classifier: {"gini", "entropy", "log_loss"};
- Regressor: {"squared_error", "friedman_mse", "absolute_error", "poisson"}.

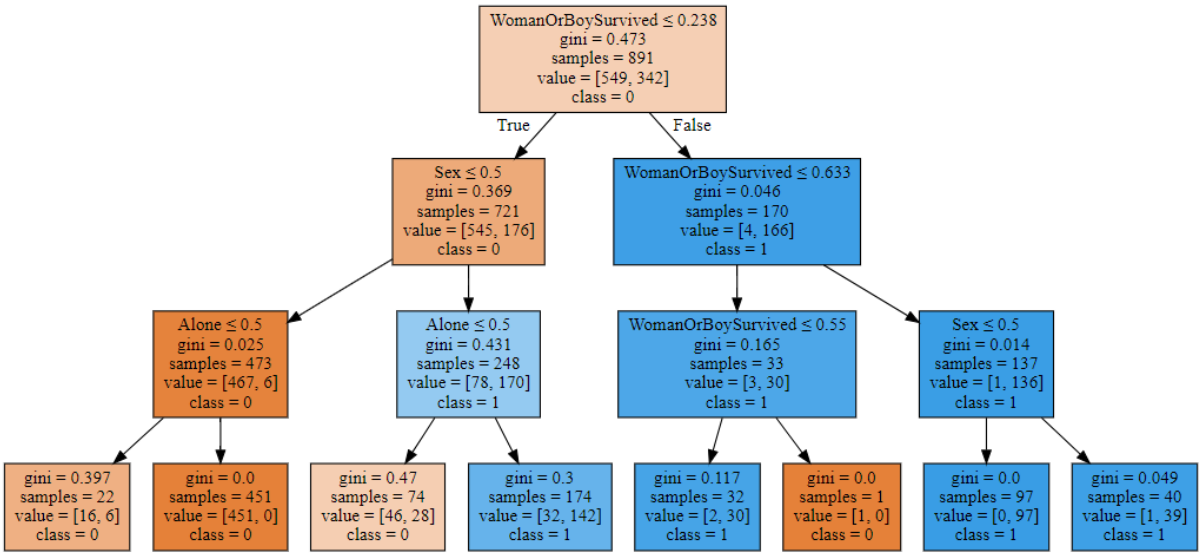
The developers of the model strongly recommend specifying the parameter max_depth as some kind of integer, starting from 2.

The main advantage of a decision tree is good interpretability. There is a method of mapping the decision tree as a tree with all the conditions, and then it can be replaced by a sequence of conditions if ... then… else. This technique was used by one of the authors of the manual during the Kaggle competition to predict the survivors of the Titanic ("Titanic - Machine Learning from Disas-

ter"), in its public notebook "Titanic – Top score : one line of the prediction (Figure 4.22).

```
import graphviz
from sklearn.tree import export_graphviz
dot_data = export_graphviz(model, out_file=None, feature_names=train_x.columns,
                           class_names=['0', '1'], filled=True,
                           rounded=False,special_characters=True, precision=3)
graph = graphviz.Source(dot_data)
graph
```

a)



b)

Figure 4.22 – Visualization of the decision tree for the competition "Titanic - Machine Learning from Disaster" with parameters max_depth=3, min_samples_leaf=2: a) code for visualization of the model decision tree using the Graphviz library; b) the result of running the code

Fig. 4.22, b presents clearly visible branches, nodes and leaves. Nodes are rectangles into which an arrow enters and from which 2 arrows come out. The arrow only enters the leaves. The color of the rectangle corresponds to the "class" (0 or 1 – the person did not survive or survived, respectively). Color saturation is inversely proportional to the value of the "gini" criterion (the closer it is to 0, the more "pure" it is, that is, it contains data of only one class).

Fig. 4.22 is a classic decision tree visualization. As you can see, it "grows" in depth, not in height, and therefore its depth max_depth, not height, is limited in the parameters. Although, by analogy with biology, the first vertex is called root. This may be unusual for biologists, but that's quite common for machine learning. Probably, the reason lies in the features of visualization. As a rule, on computers, the first point on the screen is at the top left. In the case of showing the course of "growth" of the tree in the cycle, it is advisable to start from top to bottom. Therefore, everyone is used to the fact that it "grows"

downwards, although, in most cases, the tree is first calculated programmatically and mathematically, and then visualized, and this problem would no longer exist, but everyone is used to this approach.

The convenience of decision tree interpretability lies in the fact that it can be simply "read": if ..., then..., and then, if ..., then... And so on until the end. And then the whole tree can be replaced with one condition, which will be the solution to the problem. And so it was done in the notebook "Titanic – Top score : one line of the prediction (2019) (Figure 4.23). Of course, this solution is preceded by the FE stage, where new features are synthesized that give such a beautiful and simple solution. This solution gives an accuracy of 0.80383, which as of April 2024 gives a level of "Top4%" (place 540 out of 15.5 thousand teams, although, in fact, higher, since a large part of the teams have an accuracy of 1.0, uploading answers known from the history of the Titanic, contrary to the rules of the competition).

```
# The one line of the code for prediction : LB = 0.80382 (Titanic Top 6%)
test_x['Survived'] = (((test_x.WomanOrBoySurvived <= 0.238) & (test_x.Sex > 0.5) & (test_x.Alone
> 0.5)) | \
                      ((test_x.WomanOrBoySurvived > 0.238) & \
                       ~((test_x.WomanOrBoySurvived > 0.55) & (test_x.WomanOrBoySurvived <= 0.63
3))))
```

Figure 4.23 – One line of code for predicting test data in the problem ("Titanic - Machine Learning from Disaster", which corresponds to the decision tree in Fig. 4.22,b and gives an accuracy of the "Top4%" level of the competition

The construction of decision trees is based on Shannon's information theory and probability theory, which is well described in the article and in the documentation.

Fig. 4.24 shows an example of predicting 5 different datasets by decision trees with different parameters.

As is shown in Fig. 4.20, the metric "log_loss" is clearly better suited to the binary classification problem than the metric "gini". A minimum number of samples per leaf is a way to regularize the model, but results in degraded accuracy (however, for larger datasets, this can have the opposite effect). Increasing the maximum depth of the decision tree allows for increased accuracy, except for the second dataset, where it leads to overtraining.

So, the best models in Fig. 4.24 are "DT(d=3, gini, s=1)" (first place on the first, second and fifth datasets) and "DT(d=8, log_loss, s=2)" (first place on all but the second).

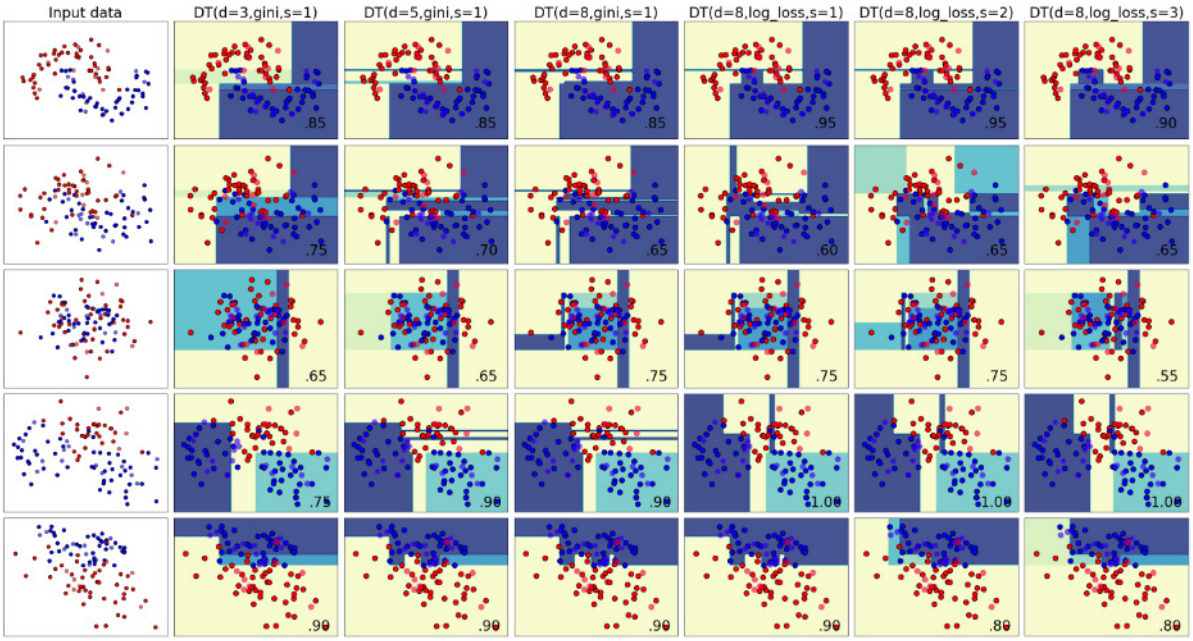


Figure 4.24 – Classification of data from 5 different datasets by decision trees with different parameters: maximum depth (d), "gini" or "log_loss" metric, minimum number of samples per leaflet (s) (in the lower right corner – accuracy_score for test (validation) data, test data are spaces circled in black) [19]

4.6.2 Comparative analysis of models on an example

Fig. 4.25 shows an example of prediction of 5 different datasets by the 9 best models of subsections 4.4-4.6.

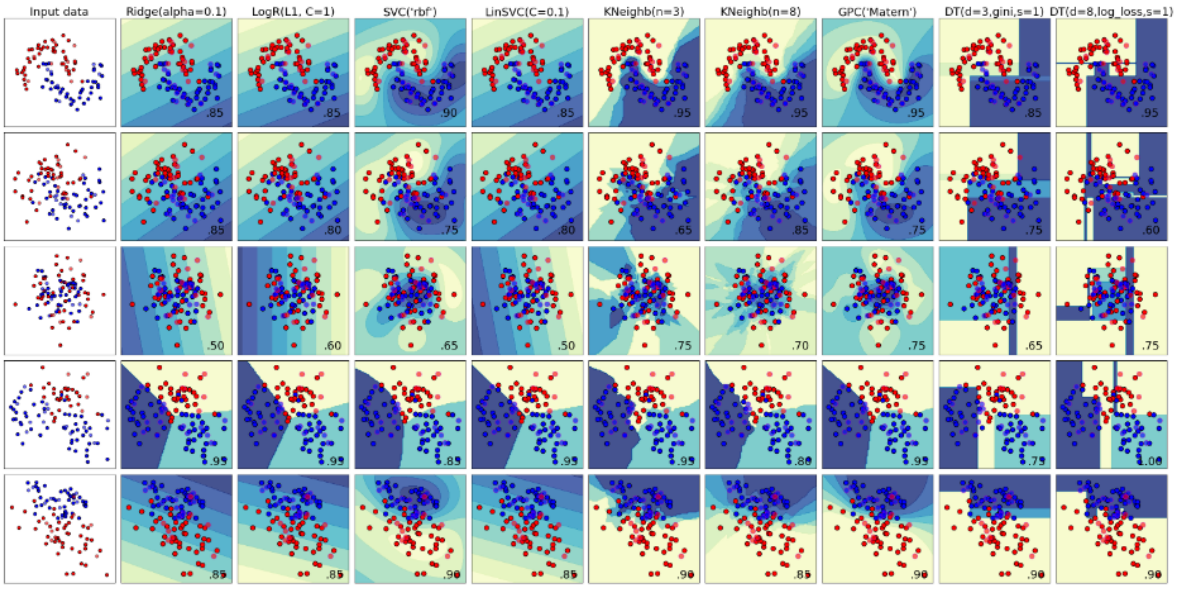


Figure 4.25 – Classification of data from 5 different datasets by 9 models (in the lower right corner – accuracy_score for test (validation) data, test data are circles outlined in black) [19]

As can be seen from Fig. 4.25, the best in terms of accuracy on test data are:

- on the first dataset (0.95): KNeighbors with k=3 and 10, GPC with the "Matern" core and DT(d=8, log_loss, s=2);
- on the second dataset (0.85): Ridge(alpha=0.1), KNeighbors with k=3;
- on the third dataset (0.75): KNeighbors with k=3, GPC with the core "Matern", DT(d=8, log_loss, s=2);
- on the fourth dataset (1,0): DT(d=8, log_loss, s=2);
- on the fifth dataset (0,9): SVC with the kernel "RBF", KNeighbors with k=3, GPC with the kernel "Matern", DT(d=3, gini, s=1), DT(d=8, log_loss, s=2).

So, the best in this example are KNeighbors with k=3, GPC with the Matern kernel, and DT(d=8, log_loss, s=2).

Let's take a closer look at them.

4.7 Randomized ensembles of trees: Random Forest and others

Some of the most common classes of machine learning models that produce good results based on the same type of model are ensembles of randomized decision trees. In them, subsets of features are randomly (randomly) selected and trees are built for each combination, and then generalized in a certain way. The most famous among them are the following ensembles:

- RandomForest (RF) – in the classification problem (RandomForestClassifier),the best prediction is determined by voting on the predictions of trees in the ensemble, and in the regression problem (RandomForestRegression) – by average in the values;
- ExtraTrees (ET) – analyzes a number of randomized decision trees (extra (very) randomized) on different subsets of the dataset and uses averaging to improve prediction accuracy;
- IsolationForest is good at detecting anomalous values when building a forest of trees.

For most tasks, RandomForest presents the best results, although in reality they are rarely the best. ExtraTrees sometimes gives even better solutions, but, in most cases, is prone to significant overtraining.

Fig. 4.26 shows an example of prediction of 5 different datasets by the RandomForest model with different parameters.

As shown in Fig. 4.26, the increase in the number of decision trees in ensemble n is expected to increase accuracy. Similarly, there is a greater maximum depth d, with the exception of the second and third datasets, on which overtraining is observed. An increase in the number of samples in leaf (s), i.e., greater regularization, succeeds just on the second dataset. Also, on the second dataset, the accuracy increases if the number of decision trees in the ensemble is increased by 10 times.

So, in Fig. 4.26 The best models are "RF(d=8, n=1000)" (1st place on datasets 1, 4, 5), "RF(d=5, s=5)" (1st place on datasets 2, 3, 5).

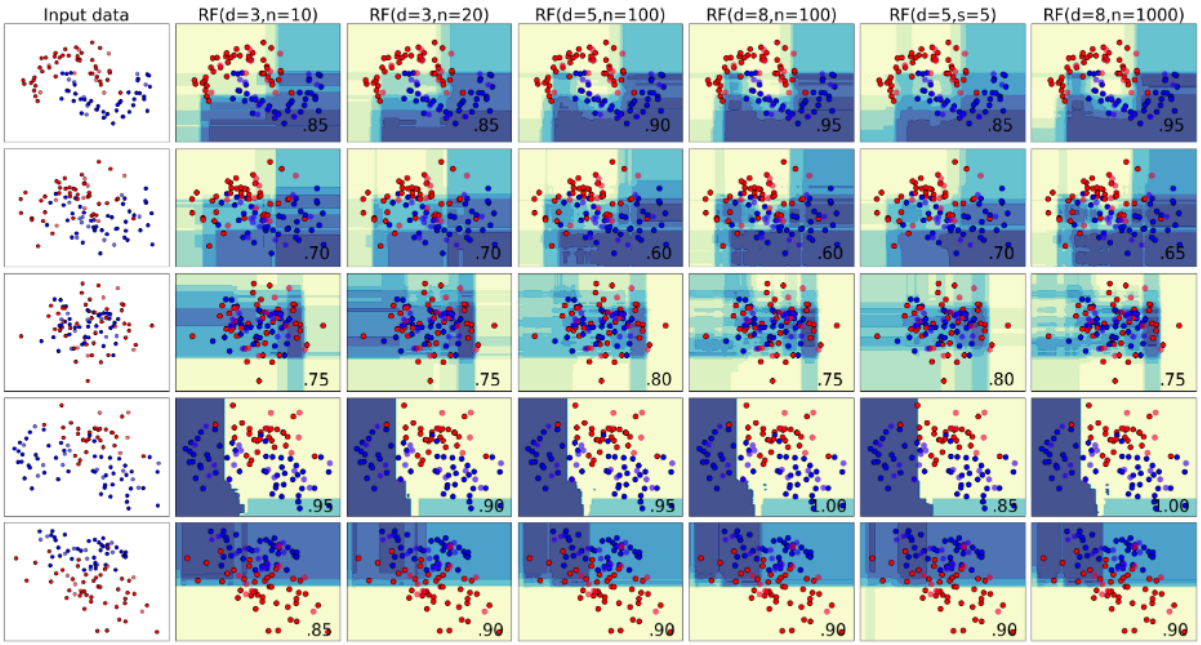
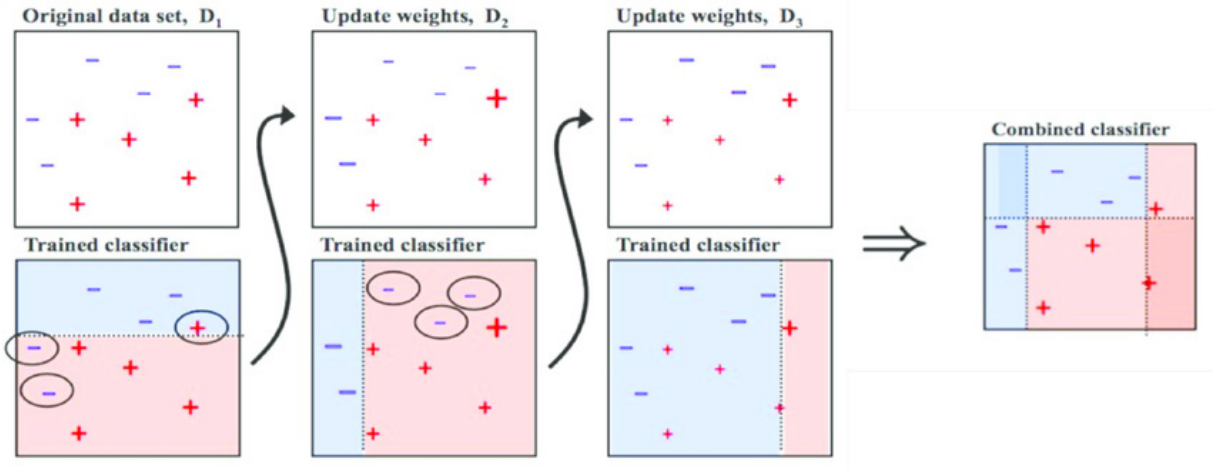


Figure 4.26 – Classification of data of 5 different datasets by the RandomForest model with different parameters: maximum depth (d), number of decision trees in the ensemble (n), minimum number of samples in a leaf (s), metric everywhere – "log_loss" (in the lower right corner – accuracy_score for test (validation) data, test data are circles outlined in black) [20]

4.8 Boosting models

Data boosting is a method of machine learning of an ensemble of weak base models, which consists in the fact that models are built sequentially in such a way that each corrects the errors of the previous one. This principle is well illustrated by Fig. 4.27.



(a) b) c) d)

Figure 4.27 – Illustration of the boosting (AdaBoost) of 3 decision trees: a) the first iteration, b) the second, c) the third; d) the answer is the consolidation of results

As can be seen from Fig. 4.27, the task is to distinguish blue "-" and red "+" (classification problem). The first model (it can also be a decision tree with max_depth=1, i.e., the usual condition "if... then… else") divides the data into 2 classes. Next, errors are analyzed (they are also called outliers in this context); In the lower figure (Fig. 4.27, a) they are circled with ellipses. The following model tries, first of all, to classify this data circled by an ellipse. At the same time, new errors are generated and analyzed (see circled by an ellipse in the lower figure (Fig. 4.27, b), the third iteration is performed in the same way (Fig. 4.27, c). And then, if it is decided that there are enough iterations, the models from all 3 iterations are consolidated and the final decision is obtained. In Fig. 4.27, d the accuracy (accuracy_score) on training data reaches 100%. Normally, there are more features and elements, they are more mixed, and therefore powerful methods and many different technologies and thousands of iterations are used.

Among the most well-known boosting models are:

- AdaBoost (AdaptiveBoosting) (see Fig. 4.27);
- Gradient Boosting Machines (GBM) or simply "GradientBoosting";
- XGBoost (eXtremeGradientBoosting) from Google;
- Microsoft's LightGBM is another fast and efficient library for gradient boosting. It uses a special algorithm to optimize calculations and supports categorical features.

The most effective XGBoost and LightGBM are the main competitors for data prediction and tend to perform better than the Sklearn library models, but they contain a lot of parameters and you need to be able to configure them. In addition, these two models have the ability to work with categorical and textual features directly, without over-processing, which improves accuracy compared to Sklearn models, which still require re-processing, which often leads to the loss of valuable information.

Sklearn's library models tend to be quicker and easier to set up, but well-tuned XGBoost and LightGBM are more accurate.

The XGBoost and LightGBM models require the data to be pre-transformed into a special format using xgb methods. DMatrix and lgbm. Dataset, respectively. An example of such a transformation is given in notebook.

XGBoost models are considered to be more accurate than LightGBM-based models, but, in practice, it is much more difficult to find the same combination of parameters that will prove this claim. LightGBM-based models are easier and faster to set up than XGBoost-based models, and therefore are used more often.

There are even faster and simpler variants of XGBClassifier, XGBRegressor, LGBMClassifier, LGBMRegressor, which do not require preliminary data transformation and are used as regular Sklearn models, but they do not always work or give an acceptable result.

To avoid overtraining XGBoost models, developers strongly recommend setting the parameter max_depth (maximum depth of the decision tree), and for

LightGBM – num_leaves (maximum number of leaves (nodes)). It is also important to correctly specify the metric of these models. Fig. 4.28 shows an example of XGBoost documentation on this subject, but this list is several pages long.

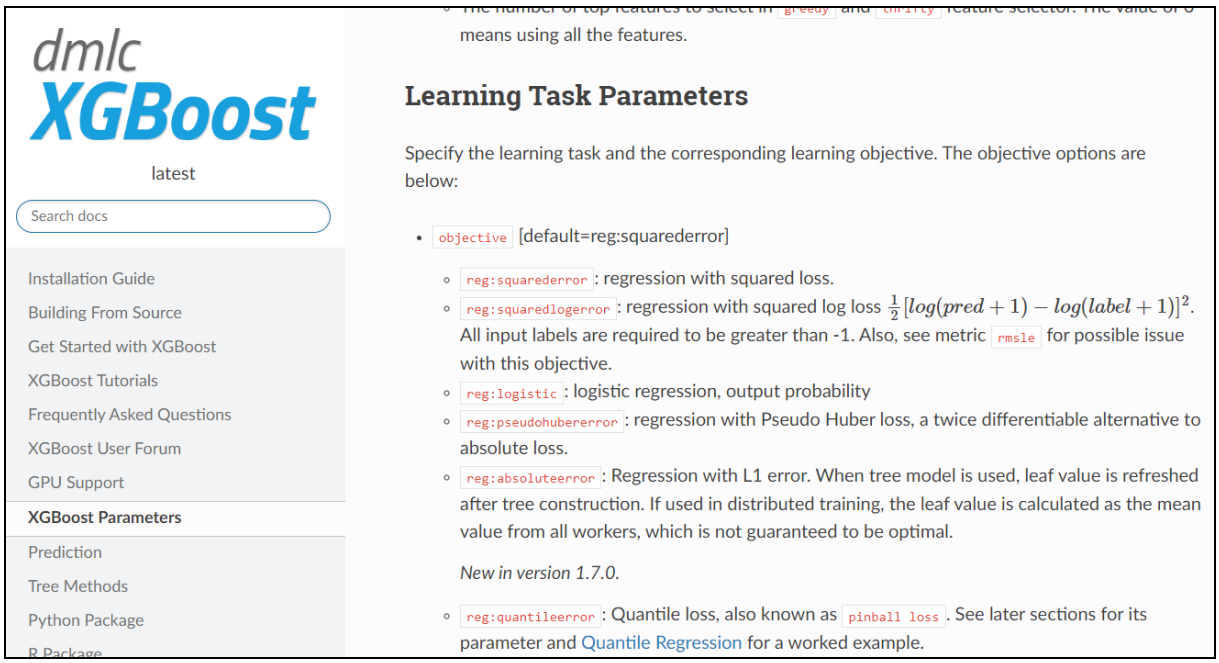


Figure 4.28 – Several variants of the metric parameter "objective" in the XGBoost model

A similar few pages for metric variants in LightGBM are in Fig. 4.29.

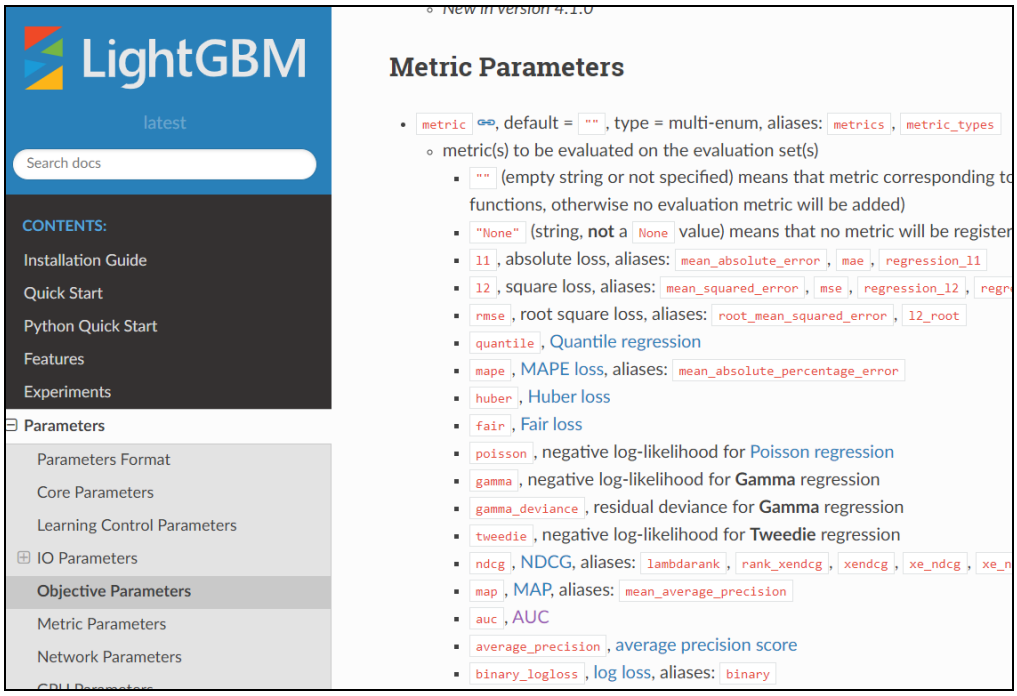


Figure 4.29 – Part of the metric variants for the "objective" parameter in the LightGBM model

Fig. 4.30 shows an example of prediction for 5 different datasets by boosting models with different parameters.

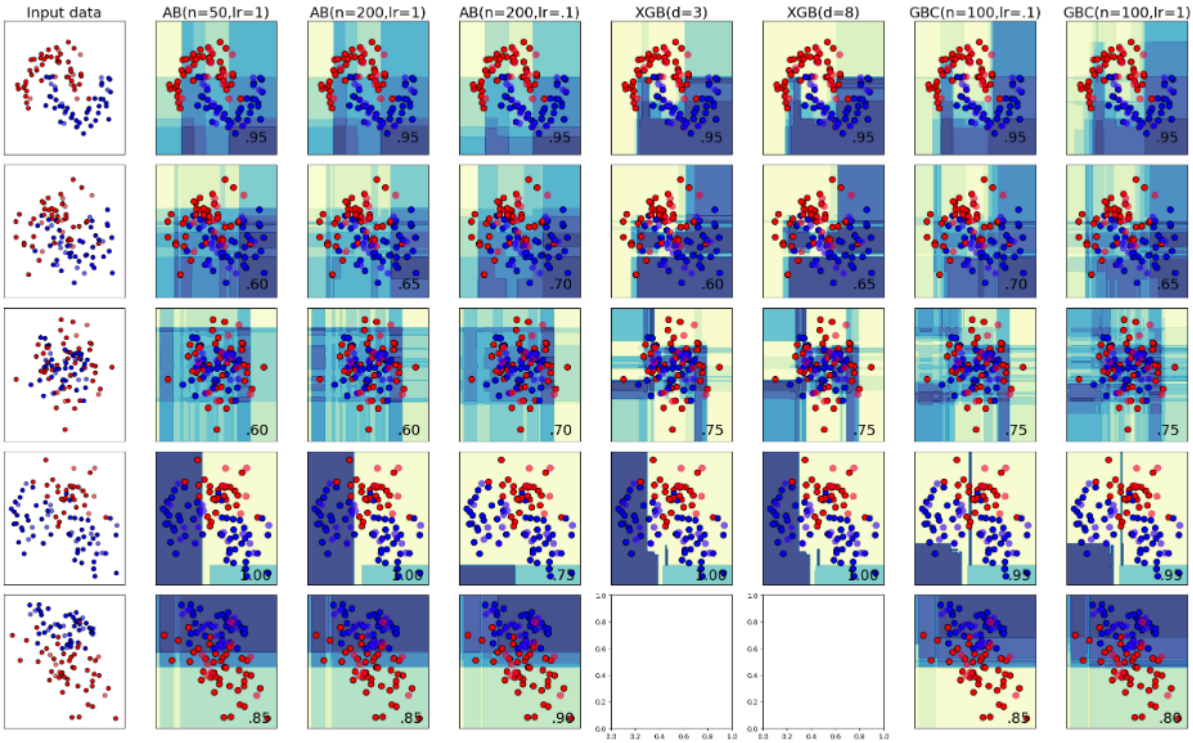


Figure 4.30 – Classification of data from 5 different datasets by boosting models: AdaBoostClassifier (AB), XGBoostClassifier (XGB), GradientBoostingClassifier (GBC) with different parameters: maximum depth (d), number of decision trees in the ensemble (n), learning rate (lr) (in the lower right corner – accuracy_score for test (validation) data, test data are circles outlined in black) [20]

As can be seen from Fig. 4.30, increasing the number of decision trees n in the AdaBoostClassifier ensemble increases accuracy as expected. Similarly, greater maximum depth d in the XGBoostClassifier model and lower lr learning rate in the GradientBoostingClassifier and AdaBoostClassifier models.

So, in Fig. 4.30 the best models are "AB(n=200,lr=.1)" (1st place on datasets 1, 2, 5), "XGB(d=8)" (1st place on datasets 1, 3, 4) and "GBC(n=100,lr=.1)" (1st place on datasets 1, 2, 3).

In the XGBoost model for the 5th dataset, some combinations of parameters give an error, so its graphs in Fig. 4.30 are not given. It is better to try to use the original xgb method for this dataset, rather than the simplified version of XGBClassifier. It was noted above that the last one does not always work or gives acceptable accuracy.

4.9 Ensembles of models. Comparative analysis of model ensembles on an example

The sklearn library provides good opportunities both for using the most common ensemble models of decision trees, and for building your own ensembles from any models.

There are a number of models (let's call them aggregators) in the **sklearn.ensemble** library that allow you to form ensembles from other sklearn library models. All of them have a special parameter estimator or estimators for this. These models are as follows:

- Bagging (**Bootstrap agg**regating) – trains a model *of one* type (estimator, by default it is a decision tree) based on subsets of data, and then aggregates their predictions (for classifiers – voting, for regressors – averaging); in fact, RandomForest and ExtraTrees are also decision tree bagging, but, unlike these models, the Bagging aggregator can work with other types of evaluation models;

- *Stacking* – training several estimator models defined in the form of a list of estimators and using their predictions as input data for the main final_estimator estimator model (it is also called *a* meta-classifier or meta-regressor, depending on the type of task);

- *Voting* – training several estimator models defined in the form of a list of estimators and aggregating their predictions by voting in one of two ways, depending on the voting parameter:

  - *hard voting* (voting = "hard"), when the class that provides for the majority of models is chosen element-by-element (as a weighted average with the same weights) – this is effective when the models are quite diverse;

  - *soft voting* (voting = "soft"), when the weight of votes depends on the model's confidence in its prediction: the final class is chosen based on the sum of probabilities for each class, i.e. the one with the largest sum and is effective when the models are comparable in accuracy.

The n_jobs parameter of the Voting aggregator allows you to implement parallelization of calculations, which can significantly speed them up. In the weights parameter, you can specify an array of weights for each model's predictions, and then these weights will be taken into account during the vote (works for both voting values). This is effective when some models are more confident than others.

A variant of ensemble formation for a regression problem using VotingRegressor(voting ="hard"), where weights is given as an array of weights, is often replaced by the usual weighted average. See for example, a notebook with 4 models in a competition with the prediction of the survivors of the Titanic.

Fig. 4.31 shows an example of prediction for 5 different datasets by ensembles selected in subsections 4.8 and 4.9 of the best models:

- Decision tree bagging ("Bagging(DT)");
- Bagging RandomForest models ("Bagging(RF)");
- Staking based on predictions using the method of support vectors, Ridge and logistic regression with the final classifier RandomForest ("Stack(SVC,RD,LgR) RF");
- Stacking based on predictions by a decision tree, a model based on Gaussian processes and a model based ont he k-nearest neighbor method with the final classifier RandomForest ("Stack(DT,GP,KN) RF");
- Ensemble based on soft voting between RandomForest predictions, a model based on Gaussian processes, and a model based on the k-nearest neighbor method ("Voting(RF,GP,KN,soft)");
- An ensemble based on hard voting between RandomForest predictions, a model based on Gaussian processes, and a model based on the k-nearest neighbor method ("Voting(RF,GP,KN,hard)").

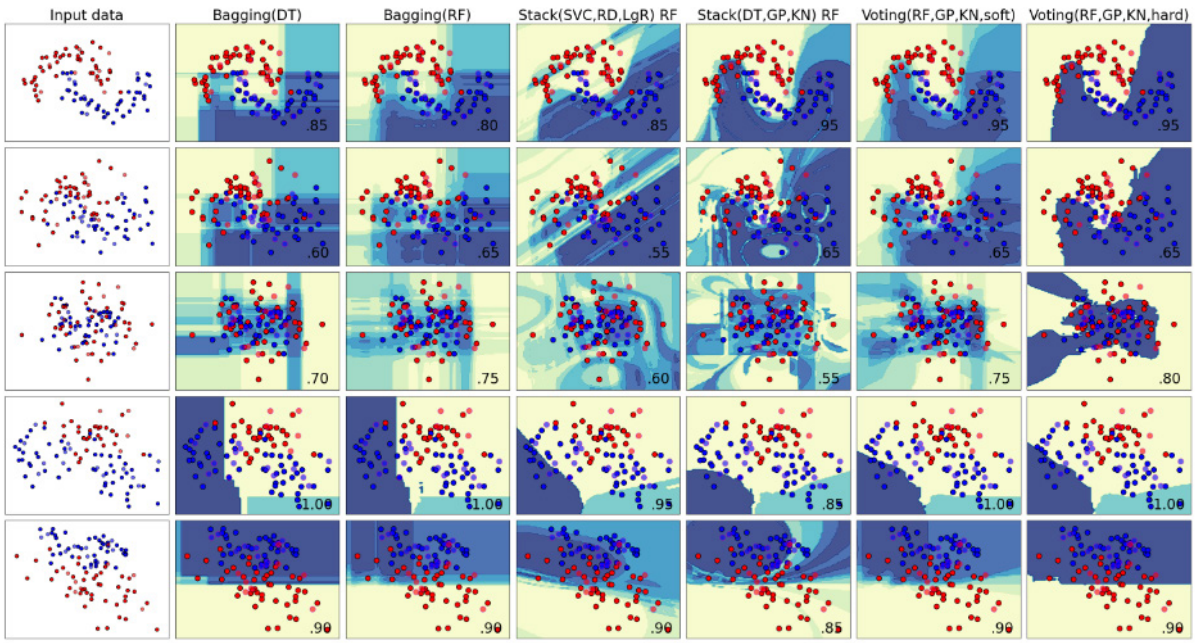
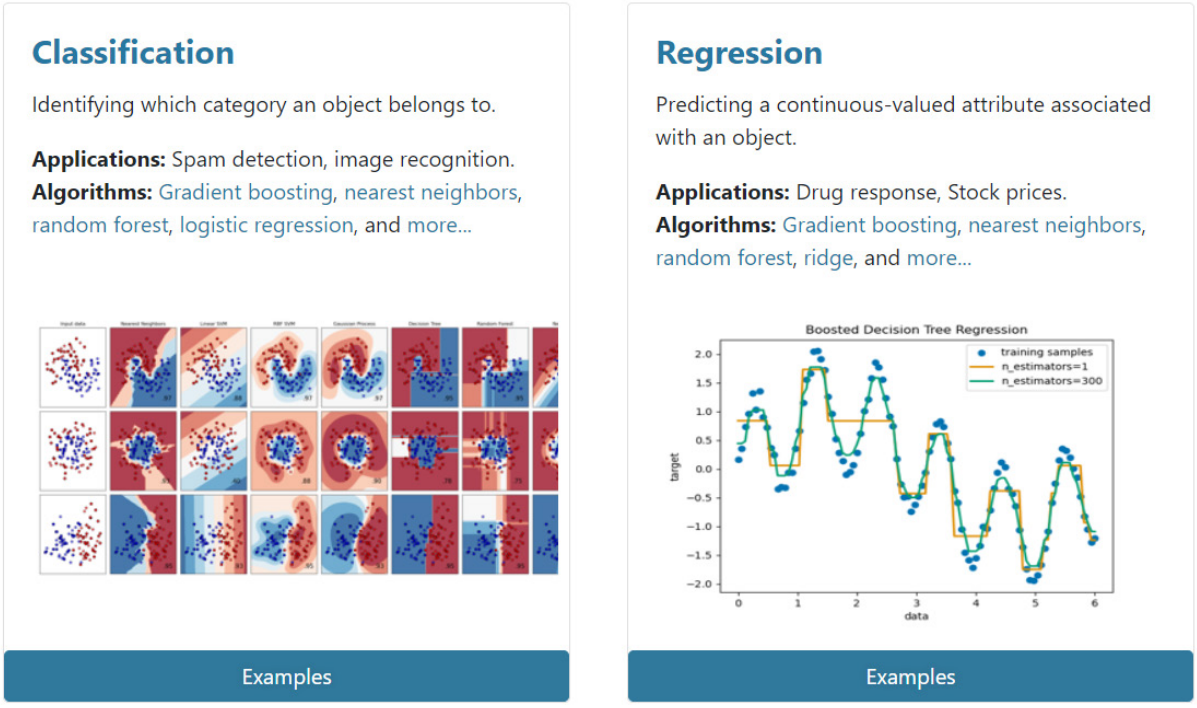


Figure 4.31 – Classification of data of 5 different datasets by ensembles of models: bagging, stacking and prediction voting with different combinations of models and parameters (in the lower right corner – accuracy_score for test (validation) data, test data are circles outlined in black) [20]

As shown in Fig. 4.31, using simple DT decision trees instead of random RF forests during bagging degrades accuracy for all but the first dataset. The best results are given by the ensemble based on a hard vote on the predictions of the models.

So, in Fig. 4.31 the best models are "Bagging(RF)" (1st place on datasets 2, 4, 5), "Voting(RF,GP,KN,hard)" (1st place on ALL datasets), the staking models are obviously overfitted.

For other types of models, see the Sklearn documentation (Fig. 4.32).



(a) b)

Figure 4.32 – Sklearn Library Machine Learning Models:
a) classifiers; b) regressors

Notebook [20, Chapter 5] lists the best of the best models built in subsections 4.4-4.9 for each of the given 5 datasets according to the accuracy_score metric. Models with overtraining risk, in which the difference between the error on training and test data exceeds 0.1, are filtered out. Fig. 4.33 shows the prediction results of the 4 best models for each dataset. As for the first dataset (No. 0), GPC, GBC, DT, AB models have the same accuracy.

It is important to remember that this is just an example of building models and analyzing them. The conclusions drawn from them cannot be extended to all other similar tasks. Each task requires its own analysis using the tuning methods from section 4.3.

	model	num_dataset	acc_train	acc_test	diff
0	Voting(RF,GP,KN,soft)	0	1.000	0.95	0.050
1	Voting(RF,GP,KN,hard)	0	1.000	0.95	0.050
2	RF(d=8,n=1000)	0	1.000	0.95	0.050
3	RF(d=8,n=100)	0	1.000	0.95	0.050
4	KNeighbors (k=10)	1	0.838	0.85	-0.012
5	LogRegres (L2, C=0.1, lin)	1	0.775	0.85	-0.075
6	SGD (alpha=0.1)	1	0.762	0.85	-0.088
7	Ridge (alpha=1)	1	0.762	0.85	-0.088
8	RF(d=5,n=100)	2	0.850	0.80	0.050
9	Voting(RF,GP,KN,hard)	2	0.838	0.80	0.038
10	RF(d=5,s=5)	2	0.812	0.80	0.012
11	Voting(RF,GP,KN,soft)	2	0.838	0.75	0.088
12	XGB(d=8)	3	1.000	1.00	0.000
13	XGB(d=3)	3	1.000	1.00	0.000
14	RF(d=8,n=1000)	3	1.000	1.00	0.000
15	RF(d=8,n=100)	3	1.000	1.00	0.000
16	Voting(RF,GP,KN,soft)	4	0.988	0.90	0.088
17	Voting(RF,GP,KN,hard)	4	0.988	0.90	0.088
18	RF(d=5,n=100)	4	0.988	0.90	0.088
19	DT(d=5,gini,s=1)	4	0.988	0.90	0.088

Figure 4.33 – Results of prediction by the best data models of 5 different datasets [20]

Figure 4.34 shows an infographics of the tools mentioned in subsections 4.4-4.9 in the *S(I)* coordinate system.

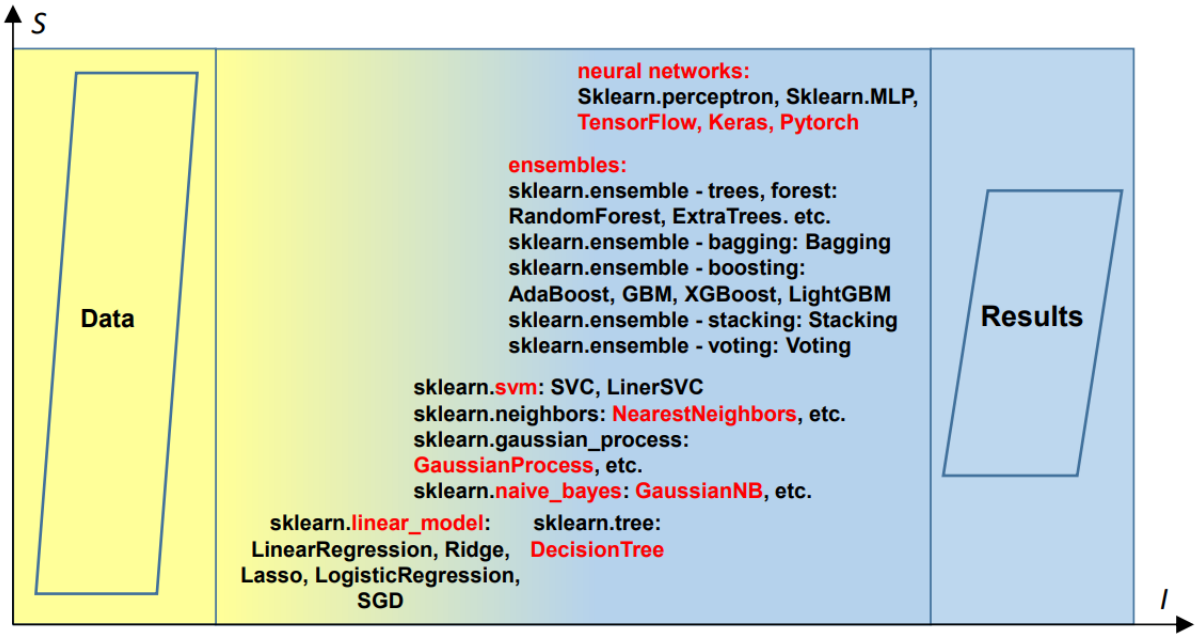


Figure 4.34 – Infographics of machine learning models and their ensembles (excluding neural networks)

4.10 Neural Network (NN) training and analyzing its accuracy. Deep Learning (DL) Concepts

A neural network in machine learning is a layer of neurons and connections between them. *A neuron* is a function with many inputs and one output, the value of which is formed depending on the *function of activating* the neuron. As soon as the output of this function crosses a certain threshold, the output is 1, otherwise 0. *Connections* are channels through which neurons send meaning to each other. Each connection has its own *weight* – a parameter by which the value in the channel is multiplied (Fig. 4.35). It is the adjustment of these *weights* (but not only them) that is carried out during the tuning (tuning) of the neural network.

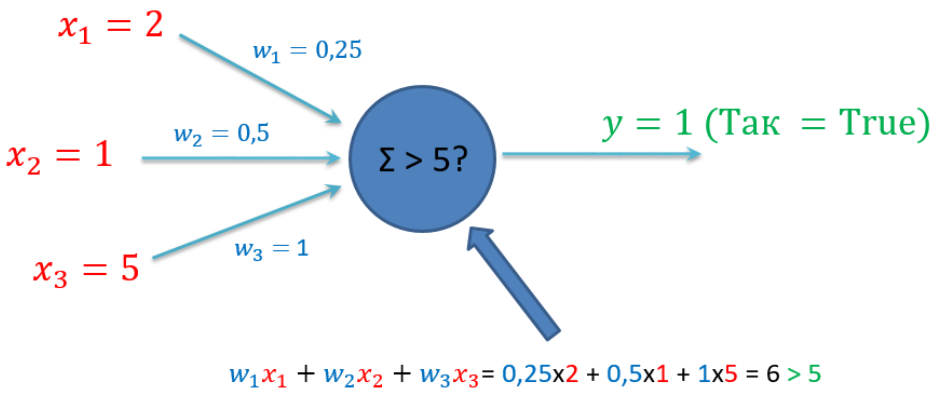


Figure 4.35 – Illustration of the operation of one neuron of a neural network in machine learning

Layers are formed from neurons. Within a layer, they are not related, but are related to the previous and next layers (Fig. 4.36).

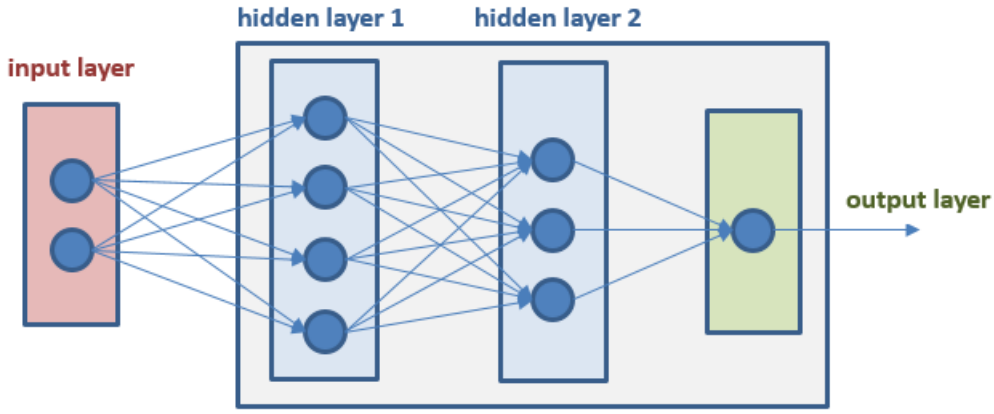


Figure 4.36 – Neural network with 4 layers, in particular with 2 hidden

The input data matrix (usually a 4-dimensional matrix, which is called a "*tensor" in machine learning*) goes to the first layer. The data is directed from left to right. Therefore, one of the most common libraries for working with neural layers is called TensorFlow. Google acquired TensorFlow and integrated it

into the Keras library, so now it is enough to work only with the latter. Its main competitor is PyTorch. Professionals, as a rule, use PyTorch, and for educational purposes, it is better to master Keras, so it is also easier and faster.

Usually, the user "sees" (can programmatically send signals or read them) only inputs and outputs, and the values in the nodes of the neural network are hidden from him, so such layers are called *hidden*. If the number of hidden layers of the neural network is greater than 1, then this is "Deep Learning" (DL).

Activation function ("excitation function" or "squashing function", "transfer function") of an artificial neuron is the dependence of the output signal of an artificial neuron on the input signal. Typically, this function displays real numbers in the interval [0, 1] or [-1, 1]. The most popular types of neural network activation functions are shown in Fig. 4.37. There is an even larger list of "Activation Functions", which is constantly updated from published articles, is available at the link (as of April 2024, the list has 73 functions).

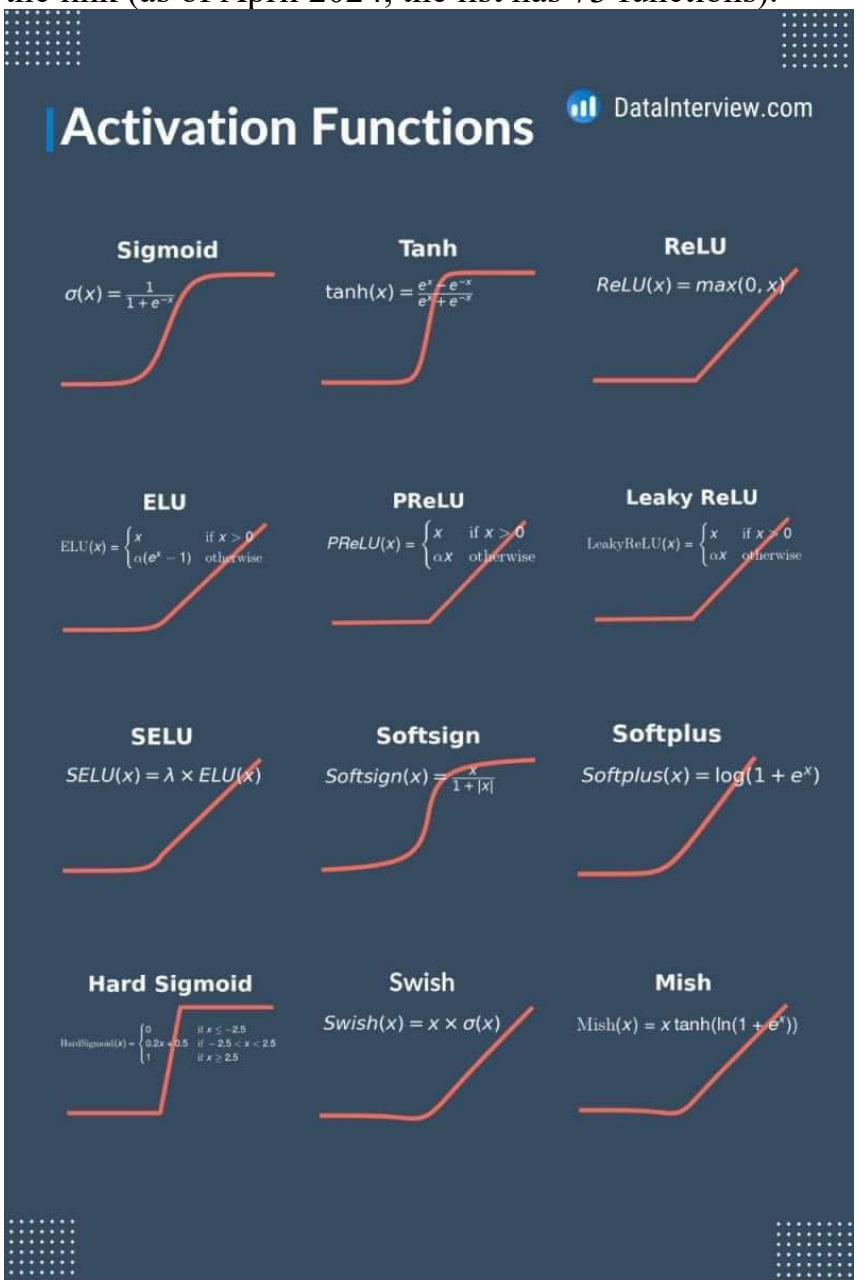


Figure 4.37 – The Most Popular Activation Functions of Neural Networks

In all hidden layers, ReLU is the most popular, and in the source layer, especially for classification tasks, **sigmoid** (if all target values are positive) or hyperbolic tangent (if there are negative values) is used.

Implementation of a simple neural network from Fig. 4.36 using the Keras library (TF) looks quite concise (Fig. 4.38).

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

model = Sequential()
model.add(Dense(units=4, activation='relu', input_dim=len(train.columns)))
model.add(Dense(units=3, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

optimizer = Adam(lr=0.001)
model.compile(loss='mse', optimizer=optimizer, metrics=['mse'])
```

Figure 4.38 – Implementation of the neural network from Fig. 4.36 in Keras (TF) from notebook

Similar code in PyTorch will look much larger (Figure 4.39).

```
import torch
import torch.nn as nn
import torch.optim as optim

class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.output_layer = nn.Linear(hidden_size2, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu1(x)
        x = self.layer2(x)
        x = self.relu2(x)
        x = self.output_layer(x)
        x = self.sigmoid(x)
        return x

# Set network parameters
input_size = len(train.columns)
hidden_size1 = 4
hidden_size2 = 3
output_size = 1

# Create an instance of the NeuralNetwork class
model = NeuralNetwork(input_size, hidden_size1, hidden_size2, output_size)

# Determine the losses and the optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Figure 4.39 – Implementation of the neural network from Fig. 4.36 in PyTorch

Each stage of neural network training is called *an epoch* (Figure 4.40).

```
Epoch 1/200
1680/1680 [==============================] - 378s 225ms/step - loss: 0.0116 - val_loss: 0.0058
Epoch 2/200
1680/1680 [==============================] - 374s 223ms/step - loss: 0.0057 - val_loss: 0.0056
Epoch 3/200
1680/1680 [==============================] - 373s 222ms/step - loss: 0.0056 - val_loss: 0.0057
Epoch 4/200
1680/1680 [==============================] - 375s 223ms/step - loss: 0.0055 - val_loss: 0.0056
Epoch 5/200
1680/1680 [==============================] - 373s 222ms/step - loss: 0.0055 - val_loss: 0.0056
Epoch 6/200
1680/1680 [==============================] - 372s 222ms/step - loss: 0.0055 - val_loss: 0.0056
Epoch 7/200
1680/1680 [==============================] - 374s 223ms/step - loss: 0.0054 - val_loss: 0.0055
```

Figure 4.40 – Fragment of an example of intermediate results of neural network training from a notebook

It is more efficient to use the variable step learning_rate (abbreviated: lr). The code from Fig. 4.38 with variable pitch is shown in Fig. 4.41.

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

def build_nn():
    model = Sequential()
    model.add(Dense(units=4, activation='relu', input_shape=(len(train.columns),)))
    model.add(Dense(units=3, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(loss='mse', optimizer='adam', metrics=['mse'])
    learning_rate_reduction = ReduceLROnPlateau(monitor='val_mse',
                                    patience=3,
                                    verbose=1,
                                    factor=0.5,
                                    min_lr=0.0001)
    return model
nn_model = build_nn()
```

Figure 4.41 – Neural network code from Fig. 4.36 in the Keras framework with variable pitch from notebook

Fig. 4.41 illustrates the **ReduceLROnPlateau** command, which monitors the change **monitor="val_loss"** against the MSE metric ("**val_mse**") and, if its value does not change within **patience=3** steps, then the **lr** value is multiplied by **factor=0.5**, i.e. halved. This will continue until the **lr** value decreases to **min_lr=0.0001**, and then stop decreasing. Fig. 4.42 shows an example of a variable step learning curve lr.

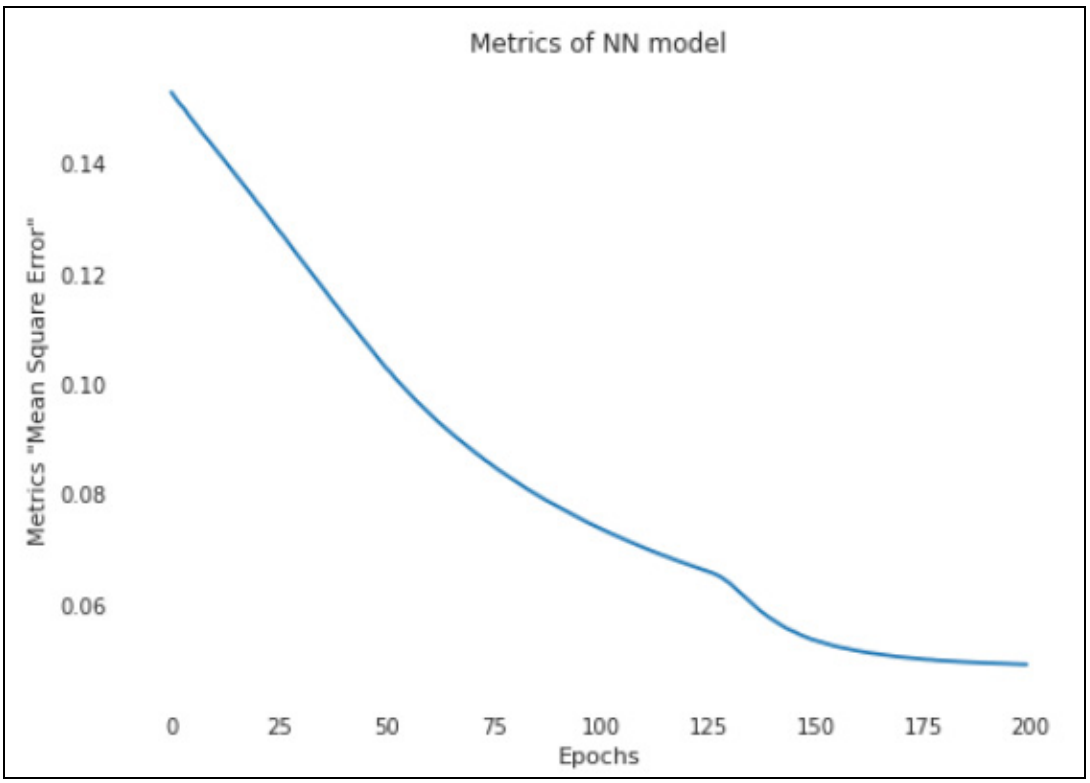


Figure 4.42 – Training a neural network with the architecture from Fig. 4.40 Variable Pitch LR from the notebook

To increase the generalization capacity of the neural network and reduce the risk of overtraining, the Dropout operation with a parameter from 0.1 to 0.4 can be used between its layers. This means that between 10% and 40% of neurons are randomly shut down during model weight training to increase its generalizing properties (Fig. 4.43).

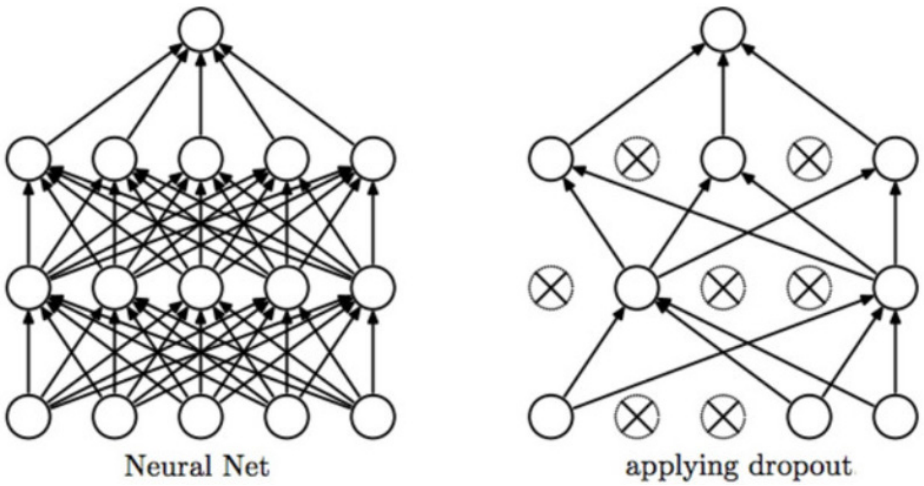


Figure 4.43 – Illustration of the Dropout operation to disable part of the neurons when training a neural network from a notebook

Fig. 4.44 shows a fragment of adding a Dropout to the code from Fig. 4.41.

```
model = Sequential()
model.add(Dense(units=4, activation='relu', input_shape=(len(train.columns),)))
model.add(Dropout(0.2))
model.add(Dense(units=3, activation='relu'))
```

Figure 4.44 – Adding Dropout for 20% of neurons to the code from Fig. 4.41 in notebook

The learning curve of a neural network with Dropout from Fig. 4.44 is shown in Fig. 4.45.

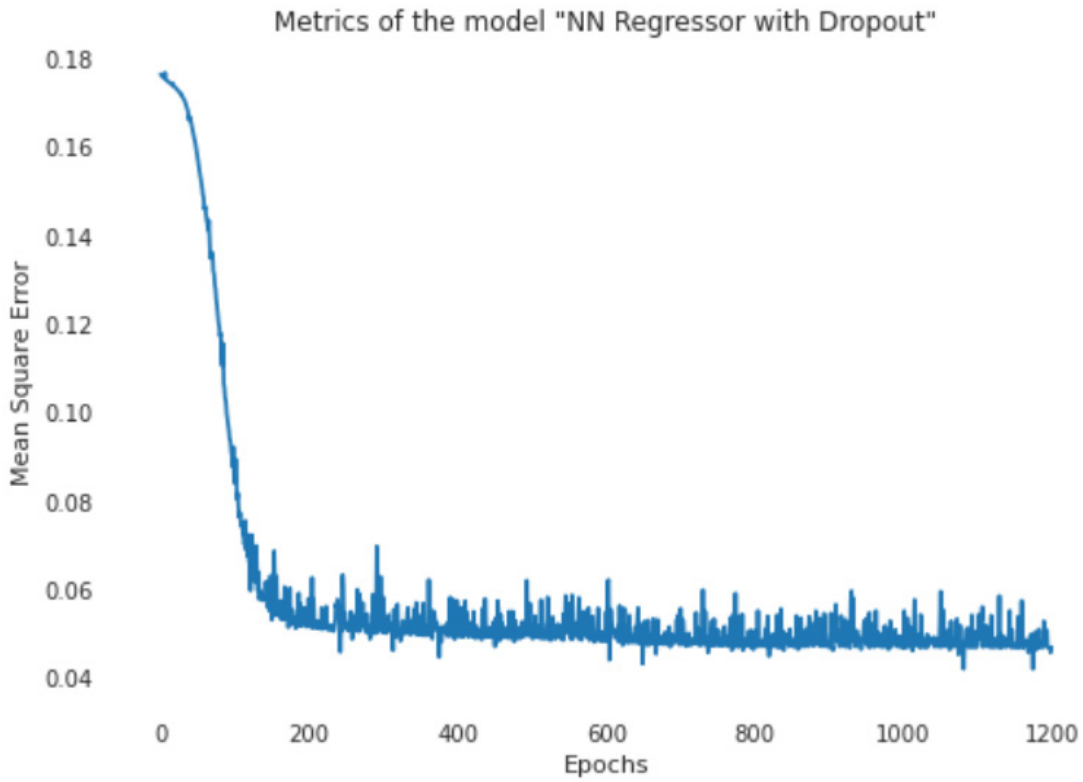


Figure 4.45 – Training a neural network with Dropout in the code from Fig. 4.44, which is added to the code from Fig. 4.41 in Notebook

As can be seen from Fig. 4.45, Dropout introduces significant noise into the learning process. It is more efficient in large amounts of data and in neural networks of complex architecture, which have a high risk of overtraining.

To analyze the process of learning a neural network online, there is a special service called TensorBoard. Callbacks (https://keras.io/callbacks/) are added to the code and what exactly needs to be tracked is specified (Fig. 4.46).

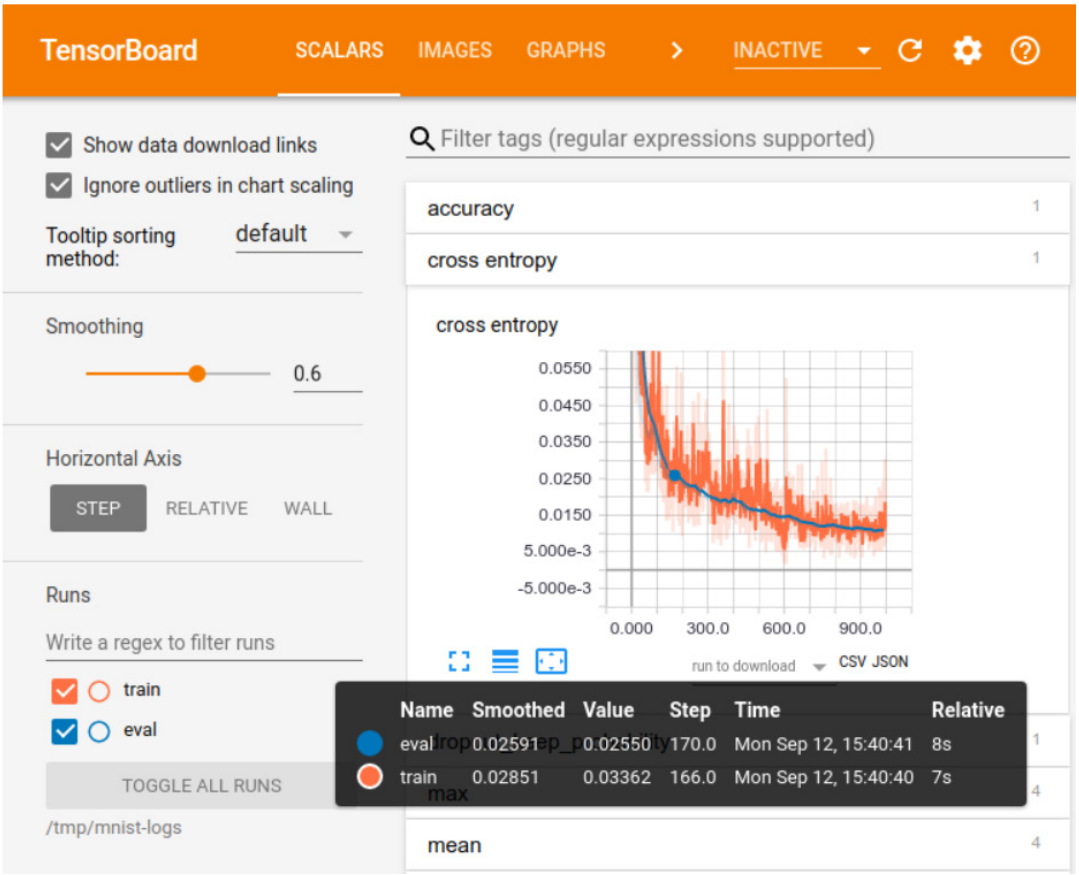


Figure 4.46 – An example of visualization of the current results of neural network training in the browser in online mode from [TensorFlow documentation]

Sometimes it is quite effective to use the model of a multilayer perceptron (neural network) from the Sklearn library: MLP ("Multi-layer Perceptron")

name_package = neural_network
name_model(model_params):

- Classifier:
  - MLPClassifier(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*);

- Regressor:
  - MLPRegressor(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, valida-*

tion_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no-change=10, max_fun=15000).

The main parameters of the MLP model, which vary to improve accuracy, are the number of hidden layers hidden_layer_sizes and the type of activation function ("identity", "logistic", "tanh", "relu").

Fig. 4.47 shows the learning results of the 4 models described in this subsection in the author's notebook.

model	train_score	train_mse	valid_score	valid_mse
MLP Regressor	0.70	0.057061	0.71	0.070000
NN Regressor with Dropout	0.73	0.050000	0.67	0.076795
NN Regressor with changed lr	0.72	0.048084	0.66	0.080000
NN Regressor	0.67	0.057061	0.66	0.080000

Figure 4.47 – Result of training 4 models in notebook

As can be seen from Fig. 4.47, the best on the validation dataset is the Sklearn MLPRegressor model, although the neural network built on the basis of the TensorFlow library with Keras, which uses Dropout, achieved higher accuracy on training data. Note that the notebook is not representative, since the models are built on a very small sample (hundreds of data), as for a regression problem. For good conclusions and high accuracy of neural networks, millions of data are needed!

Figure 4.48 shows an infographics of the tools mentioned in this section 4.10 in the *S(I)* notation.

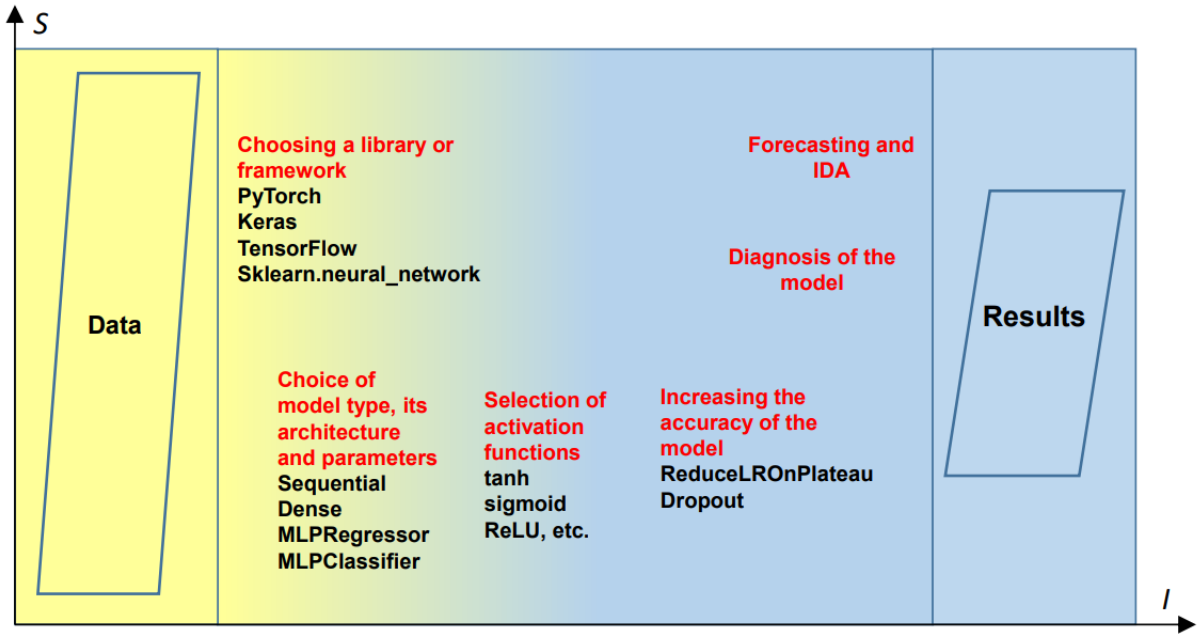


Figure 4.48 – Infographics of building and training neural network models

Practical exercises

1) There are 2 lists of binary numbers: target and prediction. Construct a confusion matrix for this task and calculate metrics:

- Precision;
- Recall;
- MSE;
- Accuracy_score.

See Chapter 4 for definitions of these metrics and how to calculate them in Python.

Example of a manual calculation for the target = [0, 1, 1, 1, 1, 0] and the prediction = [0, 1, 0, 1, 1, 0] see on the Fig. 4.49-4.53.

	Predicted Negative (0)	Predicted Positive (1)
Actual Negative (0)	True Negative (TN) = 2	False Positive (FP) = 0
Actual Positive (1)	False Negative (FN) = 1	True Positive (TP) = 3

- **True Negative (TN)**: The number of correctly predicted negative outcomes.
- **False Positive (FP)**: The number of incorrectly predicted positive outcomes.
- **False Negative (FN)**: The number of incorrectly predicted negative outcomes.
- **True Positive (TP)**: The number of correctly predicted positive outcomes.

Figure 4.49 – Confusion matrix

We draw attention to the fact that the confusion matrix in Fig. 4.49 differs from the same matrix in Fig. 4.4. There are "Predicted Positive" (1) in the left column and is "Predicted Negative" (0) in the right one in Fig. 4.4, and on Fig. 4.49 vice versa. Similarly, "Actual Positive" (1) is in the top row, and "Actual Negative" (0) is in the bottom in Fig. 4.4, and in Fig. 4.49 on the contrary. Fig. 4.4 is taken from the documentation, and Fig. 4.49 is how the matrix is displayed by the tools of the Sklearn library (as a rule, lists of values are shown in ascending order: [0 1]).

Precision measures the accuracy of positive predictions. It is calculated as:

$$\text{Precision} = \frac{TP}{TP+FP}$$

- Total Predicted Positive (1) = 3 (TP + FP)
- Precision = $\frac{3}{3+0} = 1.0000$ or 100%

Precision is 1.0000 because there are no false positives (FP = 0), meaning all positive predictions (1) were correct.

Figure 4.50 – Precision calculation

Recall measures the proportion of actual positives that are correctly identified by the classifier. It is calculated as:

$$\text{Recall} = \frac{TP}{TP+FN}$$

- Total Actual Positive (1) = 4 (TP + FN)
- Recall = $\frac{3}{3+1} = 0.7500$ or 75%

Recall is 0.7500 because there is 1 false negative (FN = 1), indicating that 3 out of 4 actual positives were correctly identified.

Figure 4.51 – Recall calculation

MSE measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. For binary classification, MSE compares the predicted values (0 or 1) with the actual values (0 or 1).

- Squared Errors:

$$\begin{aligned}(0-0)^2 &= 0\\(1-1)^2 &= 0\\(0-1)^2 &= 1\\(1-1)^2 &= 0\\(1-1)^2 &= 0\\(0-0)^2 &= 0\end{aligned}$$

- Mean Squared Error (MSE):

$$\text{MSE} = \frac{0+0+1+0+0+0}{6} = \frac{1}{6} = 0.1667$$

MSE is 0.1667, indicating the average squared difference between the predicted and actual values.

Figure 4.52 – MSE calculation

Accuracy measures the proportion of correct predictions among the total number of predictions. It is calculated as:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

- Total Predictions = 6 (TP + TN + FP + FN)
- Accuracy = $\frac{3+2}{6} = \frac{5}{6} = 0.8333$ or 83.33%

Accuracy is 0.8333, indicating that 83.33% of predictions were correct.

Figure 4.53 – Accuracy calculation

2) Calculate the output of a neuron with the ReLU activation function if the neuron's input values are u = [u_1, u_2] and the neuron's weights are w =[w_1, w_2]. An example of the solution is shown in Fig. 4.54, 4.55.

The weighted sum is computed as follows:

$$z = u_1 \cdot w_1 + u_2 \cdot w_2$$
$$z = 2 \cdot 0.4 + 3 \cdot 0.6$$
$$z = 0.8 + 1.8$$
$$z = 2.6$$

The ReLU (Rectified Linear Unit) activation function is defined as:

$$\text{ReLU}(z) = \max(0, z)$$

Applying ReLU to the weighted sum $z = 2.6$:

$$\text{ReLU}(2.6) = \max(0, 2.6)$$
$$\text{ReLU}(2.6) = 2.6$$

The output of the neuron with the ReLU activation function, given the inputs $u = [2, 3]$ and weights $w = [0.4, 0.6]$, is:

$$\text{Output} = 2.6$$

Figure 4.54 – Calculation of the output of a neuron with the activation function ReLU

We can calculate the weighted sum z as follows:

$$z = (2 \cdot 0.4) + (3 \cdot -0.6)$$
$$z = 0.8 + (-1.8)$$
$$z = 0.8 - 1.8$$
$$z = -1.0$$

Next, we apply the ReLU (Rectified Linear Unit) activation function. The ReLU function is defined as:

$$\text{ReLU}(z) = \max(0, z)$$

For our calculated z:

$$\text{ReLU}(-1.0) = \max(0, -1.0) = 0$$

Therefore, the output of the neuron with the ReLU activation function is:

$$\text{Output} = 0$$

Figure 4.55 – Calculation of the output of a neuron with the activation function ReLU and one negative input

Possible topics for practical and laboratory tasks

Topic No. 1. **Selection of methods and adjustment of machine learning models for solving problems of analysis and prediction** (or "**Building an intelligent model for predicting data on the state of a complex system and analyzing input data with its help**").

The purpose of the lesson is to study information technology and Python libraries for setting up machine learning models to solve problems of analysis and prediction and mastering practical skills in applying some of them using the example of one of the Kaggle datasets or from data uploaded via API.

Lesson plan

1. Find a dataset.

2. Describe the exact statement of the problem and determine whether it is a classification or regression problem.

3. Select the Python libraries that will be used to build machine learning models and use them to predict values, for example: Sklearn, Xgboost, Lightgbm, etc., and specify for what exactly, for which models.

4. Build at least 3 different models.

5. Make an ensemble of models or perform one of the types of generalization of their solutions.

6. Provide a table (for the classification task – also confusion matrices) with the achieved accuracy of models and their ensembles and/or generalized solutions on training and validation data, provide a graph of the predicted test data. Infer the best solution and justify it based on comparison and analysis of its accuracy on training and validation data.

For beginners in this field, it is recommended to take a notebook blank as a basis: AI-ML-DS Training. L2T: NH4 – Tree Regress models.

It is recommended to listen to the video with comments:

- Classifier Models of Tabular Data on the Example of the Titanic Competition - AI-ML-DS Training Course
- Tabular Data Regressor Models on the Example of Water Quality Modeling - AI-ML-DS Training Course
- AI Models: Data Processing, Tuning, and Model Accuracy Evaluation - AI-ML-DS Training Course
- Regressor Decision Trees on the Example of Water Quality Modeling - AI-ML-DS Training Course
- Real-World Problem Example - Water Quality Simulation - AI-ML-DS Training Course

Examples of notebooks:

- WQ SB river : EDA and Forecasting
- AI-ML-DS Training. L1T: Titanic – Decision Tree
- AI-ML-DS Training. L2T: NH4 – Tree Regress models
- AI-ML-DS Training. L4AT: Heart Disease prediction
- Heart Disease – Automatic AdvEDA & FE & 20 models

- Autoselection from 20 classifier models & L_curves
- Biomechanical features - 20 popular models
- Suspended substances prediction in river
- Merging FE &Prediction - xgb, lgb, logr, linr
- BOD prediction in river - 15 regression models

Topic No. 2. **Machine Learning and Intelligent Model Application in Kaggle Competition**

The purpose of the lesson is to study information technology and Python libraries for setting up machine learning models to solve problems of analysis and prediction and mastering practical skills in applying some of them using the example of one of the Kaggle datasets or from data uploaded via API.

Lesson plan.

1. Find a dataset.
2. Understand the problem statement and all its aspects (target feature, type of task, metric, data provided, whether external data can be used and with what license, etc.).
3. Form a team (individual participation in the competition is also possible). Organize its work.
4. Explore available solutions and tips in the "Code" and "Discussion" sections.
5. Conduct an exploratory analysis of the data, in particular the FE stage, and develop your own hypotheses for solutions.
6. Build models.
7. Submit a series of solutions, each time improving them.
8. At the end of the competition, study the decision of the winners.
9. Write an article with an overview of the problem, its solution, and your own hypotheses. Draw conclusions. What worked, what didn't. Publish your successful and interesting solutions. Make a link in the article to the notebooks and posts of others and to your own.

Test questions

1) What are the main types of machine learning models and their advantages?
2) What is model and hyperparameter training? How do they affect the process of building a model?
3) What is regularization and what role does it play in minimizing the risk of overtraining?
4) What is Linear regression, what are its features?
5) How does Logistic Regression work, in what areas is it applied?
6) What is Stochastic Gradient Descent, how is it used to train models?
7) how do Support Vector Methods work when they are applied?
8) How does the k-nearest neighbor method work, what are its features?

9) What are model ensembles, how do they help in improving forecasting results?

10) How is hyperparameter learning used to improve the efficiency of machine learning models, how is their effectivenes checked?

11) What is a neural network? Neuron? A neural network layer?

12) What is the activation function, what are the main types you know?

13) What is Deep Learning? How many minimum layers should a neural network have to implement deep learning?

14) How is the neural network trained and how is its accuracy evaluated?

15) What techniques are used to reduce neural network overtraining?

16) What tools and libraries can be used to create neural networks?

5 INTELLIGENT DATA ANALYSIS

The main purpose of intelligent models is to use them to solve problems of data analysis of various types, first of all, the main ones: numerical, textual, graphic. Also, this data can be various kinds of sensor signals, people's voices, geospatial data, data from virtual or augmented reality systems, and various complex data structures (graphs, ontological networks, etc.) [1-5, 9-11, 18, 21-38].

The most common concept of analysis basic data types is to first convert this data into numerical data, but with maximum preservation of its specifics and features, and then the problem is reduced to what is already known using machine learning models and exploratory data analysis (see Chapters 2-4).

5.1 Intelligent Analysis of Images and Videos

The most striking results of the application of IDA technologies are traditionally associated with the analysis of images and videos.

For training and testing algorithms, there are a number of typical datasets, for example, MNIST ("Mixed National Institute of Standards and Technology") – a database of 70 thousand samples of handwriting 10 Arabic numerals (60 thousand training and 10 thousand test). Data is available in the Kaggle (Figure 5.1).

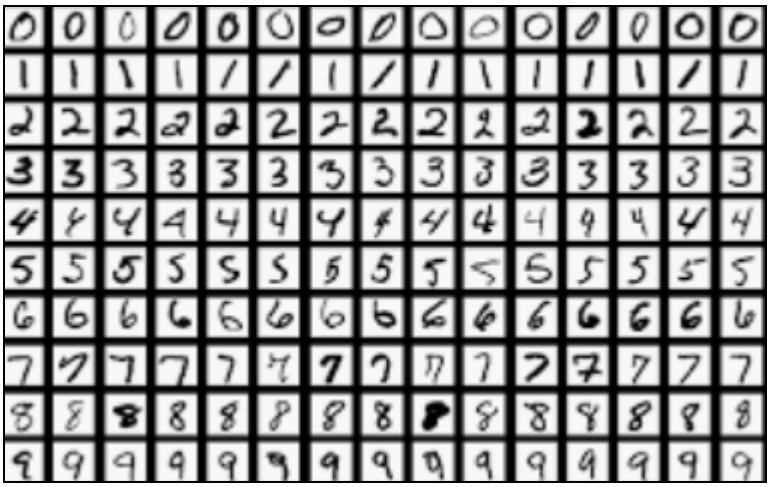


Figure 5.1 – Images of MNIST in Kaggle (monochrome, 28×28 pixels)

MNIST Fashion is a dataset of images of 10 categories of clothing items (Fig. 5.2).

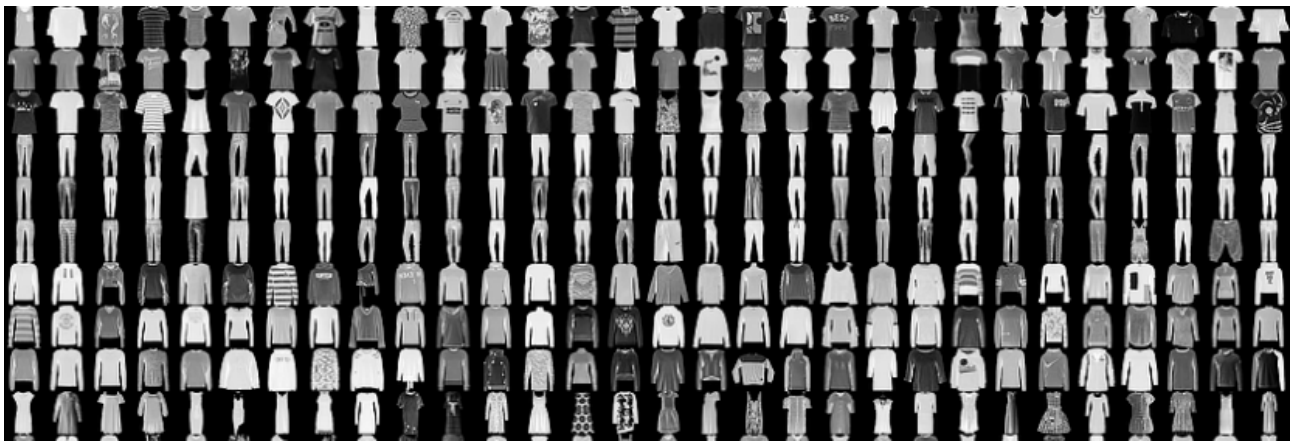


Figure 5.2 – Images of MNIST Fashion (monochrome, 28×28 pixels)

We will characterize typical tasks of graphic data analysis and methods and technologies for their solution.

5.1.1 Basic concepts, colors encoding, basic types, tensors

Normally, all videos are treated as a sequence of images.

Each image (denoted U) is cut into a collection of fragments (chunks or patches), such as squares of a certain size [39]. Occasionally, these are rectangles. Each patch is a matrix of pixels. Let X and Y specify the number of each pixel by the height and width of this matrix. For each pixel, you specify the number of C color channels. Generally: C = 3 in RGB format ("Red – Green – Blue"). Or it could be C = 1, if the image is monochrome. In the first case, the color is represented by a 6-digit number in the hexadecimal number system (Fig. 5.3). And for black-and-white images, the color is binary: 0 or 1, or, as in the MNIST dataset, numbers from 0 to 255.

#FFCCCC	#FFC0C0	#FF9999	#FF8080	#FF6666	#FF4040	#FF3333	**#FF0000**
#FFE5CC	#FFE0C0	#FFCC99	#FFC080	#FFB266	#FFA040	#FF9933	**#FF8000**
#FFFFCC	#FFFFC0	#FFFF99	#FFFF80	#FFFF66	#FFFF40	#FFFF33	**#FFFF00**
#FFFFE5	#FFFFE0	#FFFFCC	#FFFFC0	#FFFFB2	#FFFFA0	#FFFF99	#FFFF80
#E5FFCC	#E0FFC0	#CCFF99	#C0FFA0	#B2FF66	#A0FF40	#99FF33	**#80FF00**
#CCFFCC	#C0FFC0	#99FF99	#80FF80	#66FF66	#40FF40	#33FF33	**#00FF00**
#E5FFE5	#E0FFE0	#CCFFCC	#C0FFC0	#B2FFB2	#A0FFA0	#99FF99	#80FF80
#CCE5CC	#C0E0C0	#99CC99	#80C080	#66B266	#40A040	#339933	**#008000**
#CCFFE5	#C0FFE0	#99FFCC	#80FFC0	#66FFB2	#40FFA0	#33FF99	**#00FF80**
#CCFFFF	#C0FFFF	#99FFFF	#80FFFF	#66FFFF	#40FFFF	#33FFFF	**#00FFFF**
#E5FFFF	#E0FFFF	#CCFFFF	#C0FFFF	#B2FFFF	#A0FFFF	#99FFFF	#80FFFF
#CCE5E5	#C0E0E0	#99CCCC	#80C0C0	#66B2B2	#40A0A0	#339999	**#008080**
#CCE5FF	#C0E0FF	#99CCFF	#80C0FF	#66B2FF	#40A0FF	#3399FF	**#0080FF**
#CCCCFF	#C0C0FF	#9999FF	#8080FF	#6666FF	#4040FF	#3333FF	**#0000FF**

Figure 5.3 – Examples of RGB pixel colors in hexadecimal notation

Consequently, each fragment of the U image is encoded as a four-dimensional matrix

$$U = [<N>, <X>, <Y>, <C>], \quad (5.1)$$

where each cell contains a number that corresponds to the color of the corresponding pixel.

Such a multidimensional matrix of numbers (5.1) is called *a tensor*. If the images are large, then they are also divided into batches of these fragments. That is, it is already a 6-dimensional matrix (array) of data or a two-dimensional tensor.

And the black-and-white image of 28 by 28 pixels of the MNIST dataset is not divided into parts or fragments and is a one-dimensional tensor. The number of channels is C=1, and the numbers in the tensor cells are 0 (white) or integers up to and including 255 (black) (see Fig. 5.3): (1, X, Y, 1), where X and Y are integers from 0 to 27 inclusive. And the pixel color of an HD quality image will be a hexadecimal number, which will be decoupled into a pixel numbered for example (8, 16, 22, 2). These numbers mean the 9th fragment (numbering starts with zero), 16 and 22 are the coordinates in this fragment along different axes, the 2nd channel (green).

5.1.2 Typical tasks

The classic tasks of image and video processing and intelligent analysis are the following:

1. Image Classification – defining a class or category of images. It can be either a specific type (pollen on the microscope images [27], roofs on the aerial photography [34], face, car number, whether there is a person wearing a mask, a type of plant, military equipment based on satellite or aerial photography, etc.), or a classification of certain changes in the object (emotions on faces, destruction of houses, deforestation, emergence of crops in the fields, etc.).

2. Object Detection – detection and localization of objects in the image. Detecting vehicles on the streets (counting traffic, searching for traffic jams), calculation of pollen concentration in atmospheric air using a laser [33], detecting a person's face and their key points, detecting points for finger print analysis, etc.

3. Semantic Segmentation – as signing a class to each pixel in the image: such a definition of different classes in the image with high accuracy, as deter analysis the contours of a road and sidewalks. It is also used when analyzing video online using YOLO technology.

4. Image (Video) Generation – creation of new images (the so-called deepfakes). "Deep Fake" based on training datasets and custom deep learning models. Images can be created both from several images (GAN models, etc.) and from text descriptions (Stable Diffusion models, etc.). Images can also be generated using templates from well-known graphics packages, controlling their parameters [38].

5. Anomaly Detection – detection of unusual or anomalous patterns in images: a comet in space, oil spills at sea, damaged plants, or other problem areas that require a quick response. An example of solving such a problem is described in an article by one of the authors [39].

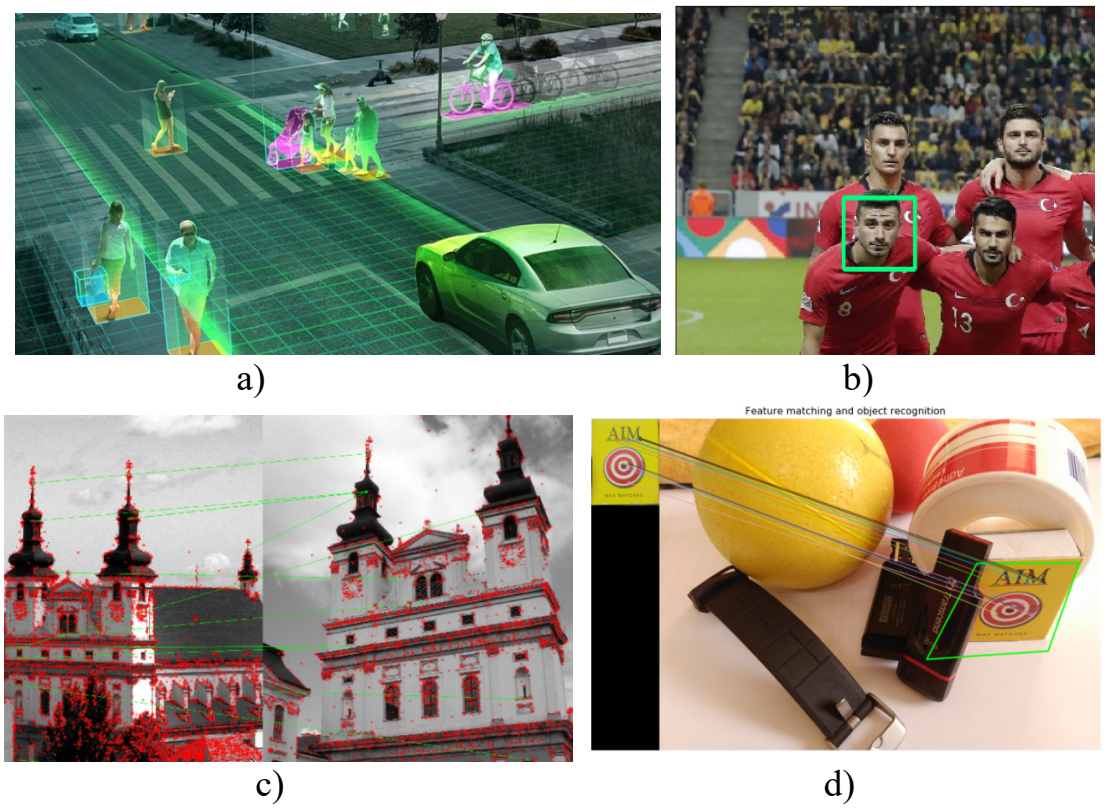
6. Face Recognition – identification of faces in images, including by turning their faces, wearing a mask, wearing makeup, fingerprint recognition on a smartphone, etc.

7. Image (Video) Enhancement and Generation by applying filters and transformations to enhance or generate new images (videos), improve the quality of images, create artistic and creative images, convert black and white images to color. To solve such problems, the OpenCV library is often used. Let's take a closer look at it.

5.1.3 Image preprocessing. OpenCV library

The library OpenCV (short for "Open Source Computer Vision Library") is designed to solve many problems: computer vision, video processing, image processing, etc.

The library contains more than 2500 algorithms optimized in speed and accuracy for various purposes (Fig. 5.4):



a) b) c) d)

Figure 5.4 – Examples of OpenCV library:
a) construction of 3D models and image identification;
b) face recognition; c) finding similar architectural forms;
d) finding similar images, taking into account their geometric alones transformations

- face detection and recognition, identification of objects in images or videos;

- classification of human actions in the video;

- tracking camera movements;

- tracking moving objects (for example, during football matches, the camera can automatically track the ball on the field);

- building 3D models, obtaining 3D point clouds from stereo cameras, which allows you to immediately build and analyze a model of the entire environment in dynamics around a UAV (car, plane or quadrocopter);

- connecting images together to produce a high-resolution image of an entire scene;

- finding similar images in the image database;

- removing red-eye from images, tracking eye movements;

- landscape recognition;

- setting markers to overlay them on augmented reality, etc.

5.1.4 Convolutional Neural Networks (CNN): principles of work and typical architecture

CNN (Convolutional Neural Network) is the most common type of neural network model for analyzing images and classifying objects on them.

Let's consider the application of CNN to the recognition and classification of digits of the MNIST dataset (Fig. 5.5, a, b).

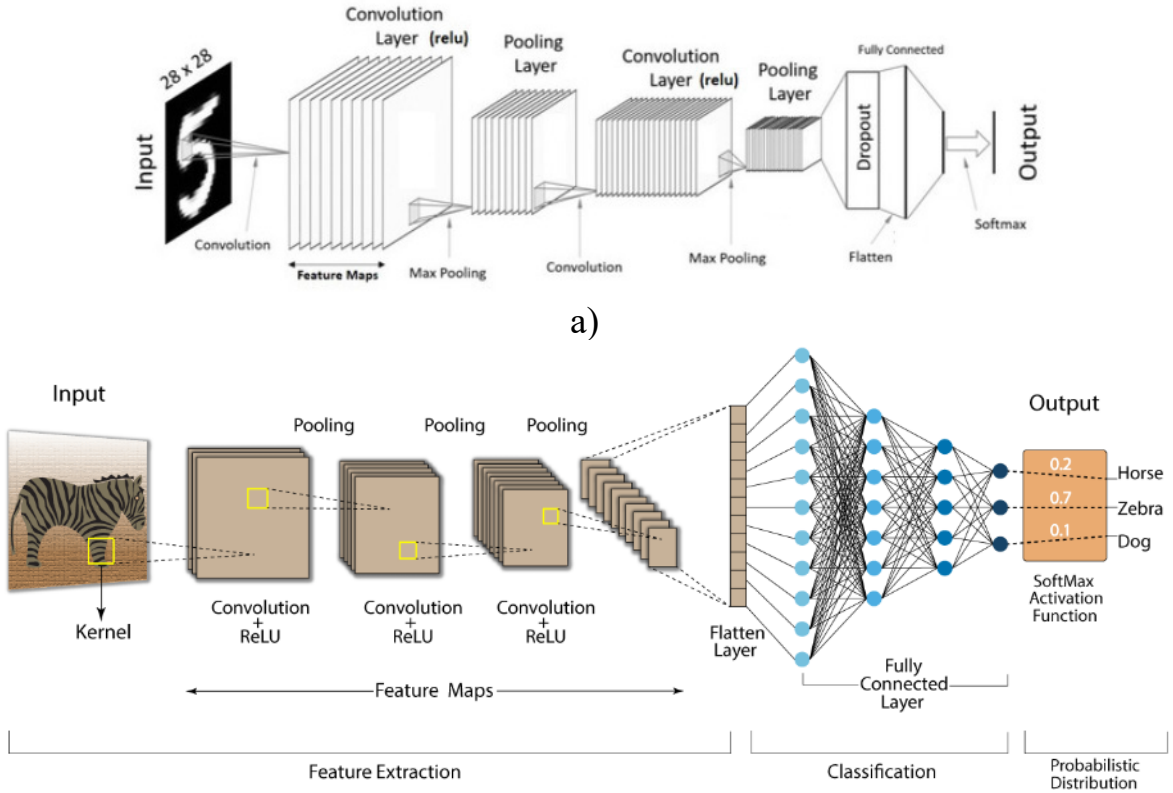


Figure 5.5 – Typical architecture of a CNN: a) to recognize the digits of the MNIST dataset [40]; b) to recognize the species of the animal [41]

The main element of CNN is a convolution. To use it, you need an input matrix of numbers **M×M** (can be rectangular) with the colors of the image pixels and a convolution kernel **W×W** (can also be rectangular), which allows you to process a submatrix of the same size **W×W**: the matrices are multiplied element-by-element and the result (one number) fits into the final matrix in the corresponding cell (Fig. 5.6).

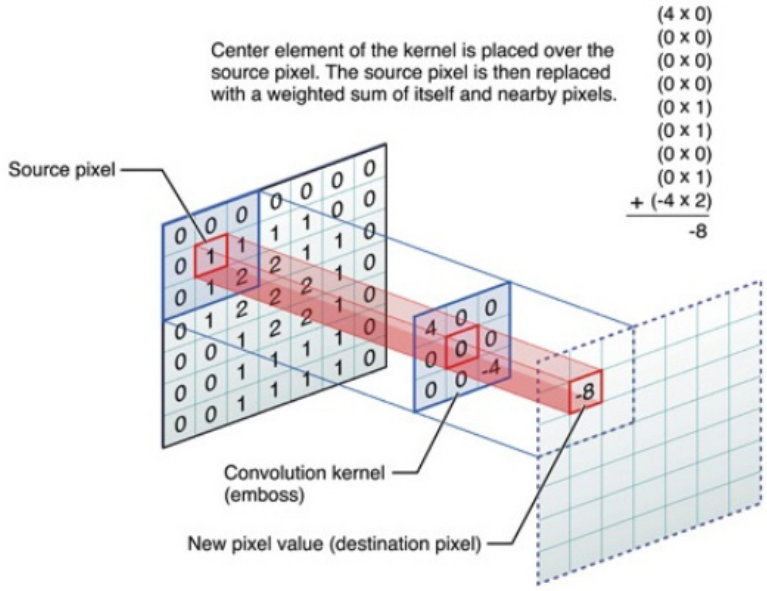


Figure 5.6 – Illustration of the convolution process [40]

The convolution operation is applied in a loop and "collapses" all the elements of the input matrix. Obviously, if you multiply the matrix **M×M** = 7×7 by the core matrix "kernel") **W×W** = 3×3, then this can only be done $M - W + 1 = 7 - 3 + 1 = 5$ once. This version of the convolution (the "padding" parameter is responsible for this) is called "value" (the notebook has a nice gif illustration that shows in dynamics how the final matrix is formed using the kernel in the "value" mode). But the more popular option is padding = "same", when the size of the final matrix is the same as the input one: **M×M**. For such convolution, zeros (0) are added to the input matrix in adjacent cells to ensure the possibility of applying the operation as many times as necessary (Fig. 5.7).

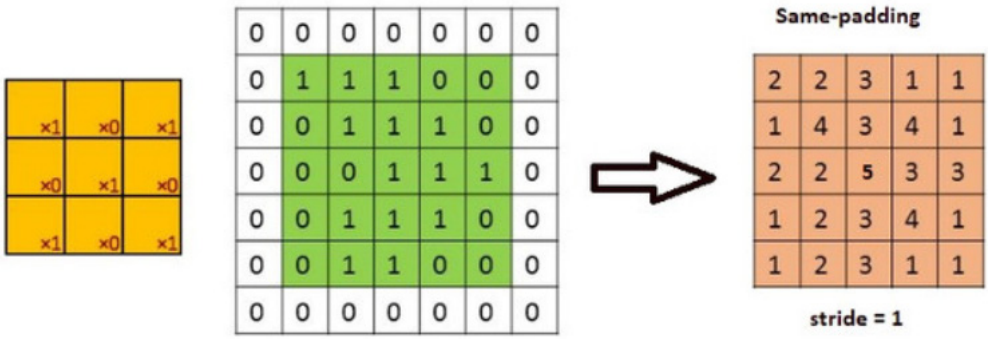


Figure 5.7 – Illustration of convolution mode with padding = "same" [40]

Applying a convolution using the Keras (TF) framework:

x = Conv2D(9, (3, 3), activation='relu', stride = (2, 2), padding='same')(x)

which means that a two-dimensional Conv2D 3×3 pixels convolution is applied to the input image 9 times, activation function – ReLU (Fig. 5.8), the kernel jumps 2 steps vertically and horizontally, and the result of the convolution will be the same size as the input image.

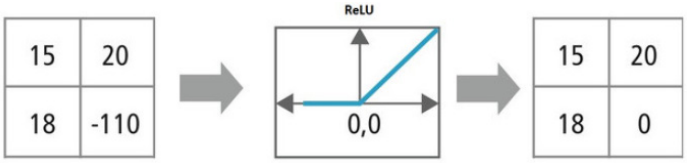


Figure 5.8 – Illustration of the use of the ReLU activation function [41]

Each such convolution allows you to find some important patterns in the input image and save them for the next stage of analysis. For example, in Fig. 5.9 shows an example of image segmentation.

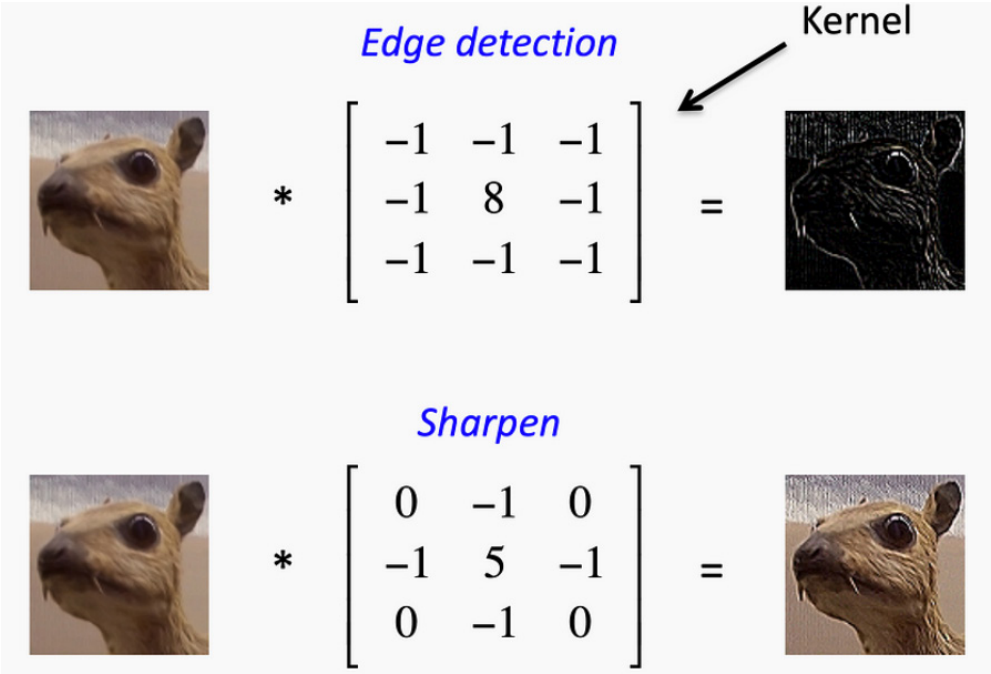


Figure 5.9 – Example of the use of different types of convolutions to the input image from notebook

As a result of using the convolution, the original image will be quite diverse. Therefore, in tandem with each convolution, as a rule, the MaxPooling or AveragePooling operation is used (see Fig. 5.5). As a result of this operation, the maximum or average value, respectively, of all elements of this submatrix is simply determined (Fig. 5.10):

MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding="valid")
MaxPooling2D(pool_size=(4, 4), strides=None, padding="valid")
AveragePooling2D(pool_size=(3, 3), strides=3, padding="same")

Where **pool_size** is the size of the submatrix of the input matrix to which the operation is applied, **strides** is the jump (**None** means 1).

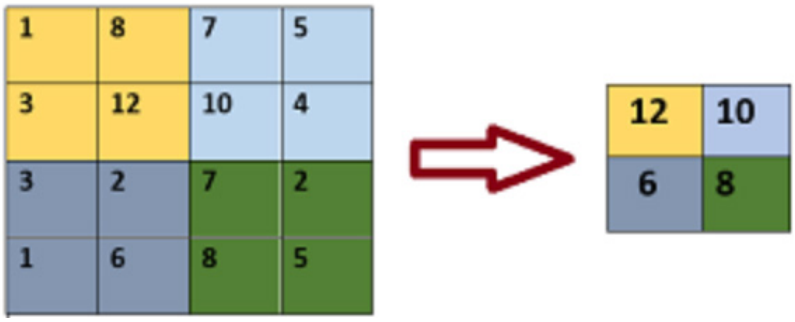


Figure 5.10 – Example of
MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid") [41]

The convolution is only the **Conv** operation, unlike **MaxPooling**, **AveragePooling**, because they do not use the convolution kernel. All of these 3 types of operations have options for 1D, 2D, 3D, depending on the dimension of the input.

Often, but not always, **Conv** layers are used with **padding = "same"** and MaxPooling, AveragePooling with **padding = "value"**. These operations are used in pairs (see Figure 5.5) or in blocks (2–3 convolutions and then Pooling). At their output, sometimes, they put a Dropout, mentioned in subsection. 4.10.

After the "Conv-Pooling" pairs with the Dropout, the "Flatten" layer is used, which converts all the resulting matrices into one one-dimensional data array (Fig. 5.11).

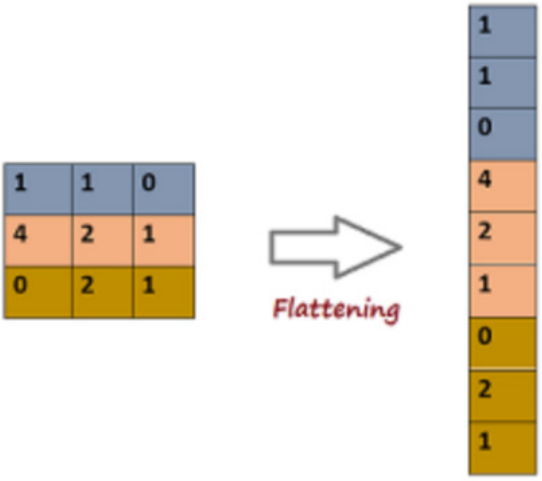


Figure 5.11 – An example of using the Flatten operation [41]

Data from Flatten is sent to a classic Fully Connected neural network (see Figure 5.5). If this is a classification task, then in the last layer of this neural network, as a rule, the Softmax activation function is used. It allows you to more accurately determine which class is the final one.

The main principle and advantage of CNN is that the researcher only forms the architecture, and the values of all matrices and convolution cores and other internal parameters are calculated automatically.

CNNs have become widely used, and not only in image recognition tasks. They work for text analysis, and for the analysis of tabular data, and for forecasting time series, since they are able to successfully process various numerical information.

5.1.5 Modern architectures of neural networks

Appendix G lists the neural network architectures that were relevant in 2016, and many of them are still in active use. Also in Appendix G it is noted that neural network models ResNet, ResNeXt, Efficient.Net and their variations have recently become popular and effective.

There is a web portal "Papers with Code", which registers and displays on one graph all known models and their accuracy, there is also a code and description of each model. Hence the name comes from it. Fig. 5.12 shows all models, including the best ones from 2016, as of April 2024.

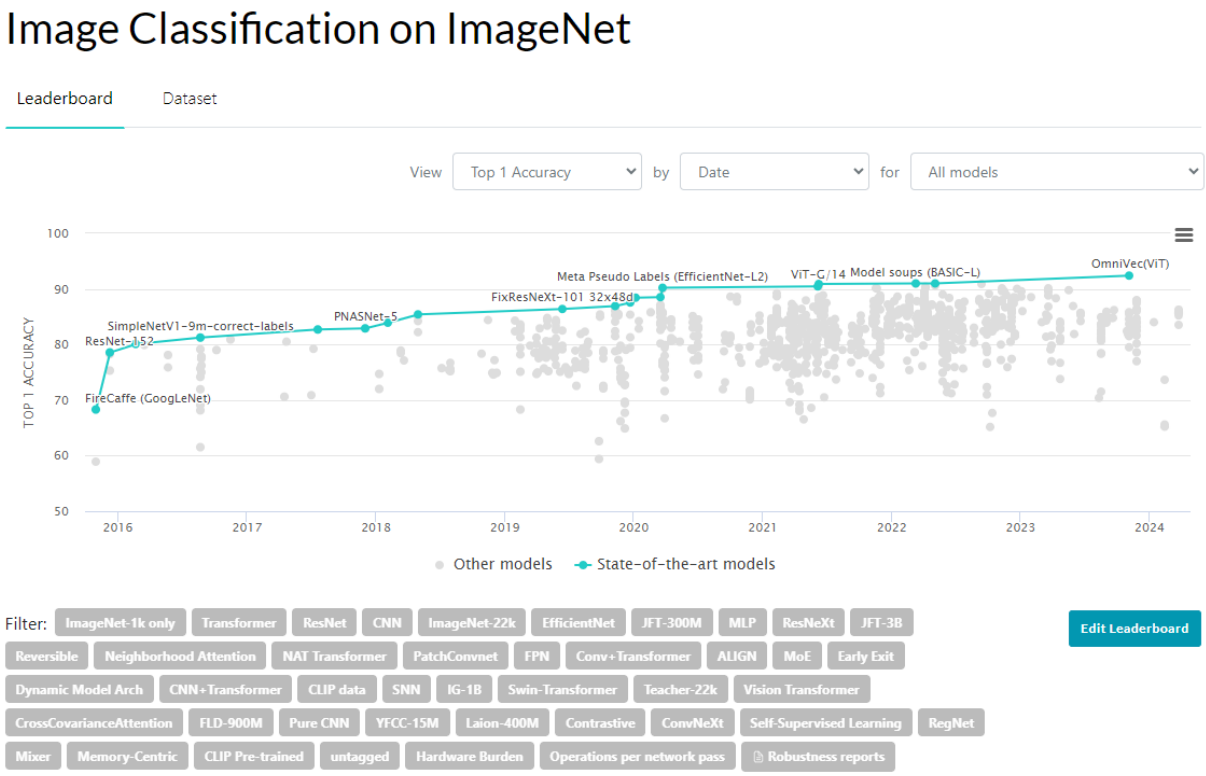


Figure 5.12 – Comparison of the accuracy of models for the classification of images of the ImageNet dataset on the portal "Papers with Code"

As can be seen from Fig. 5.12, the most accurate models for today are those of the ViTs ("Vision Transformers") class, which contain both convolutional, transformer and other blocks.

To train CNNs with complex architectures, the amount of input or variety of inputs is often not enough. Then the so-called augmentation is used, i.e. artificially enlarging the dataset. For example, using the OpenCV library, the image is rotated at different angles, rotated around one of the axes, deformed (compressed, stretched, etc.), added or removed noise, changed size or resolution, etc. (Fig. 5.13).

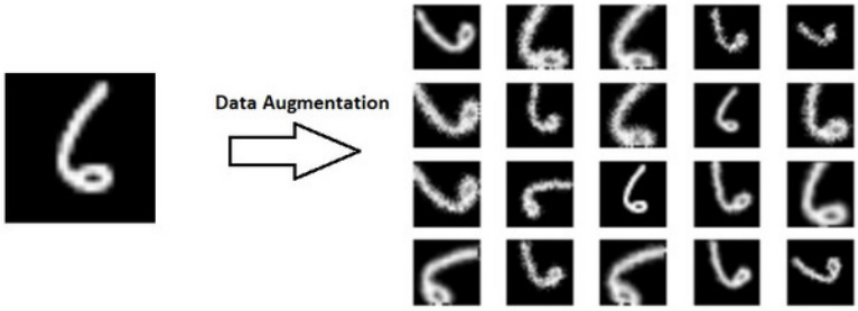


Figure 5.13 – Example of image augmentation from the notebook

5.1.6 Auto encoders in unsupervised tasks

As described above, there are unsupervised tasks where you need to find unknown patterns in a large dataset. One of the most common ways to solve this problem is to use "Convolution Auto Encoder" ("CAE").

The basic principle of operation of the autoencoder is illustrated in Fig. 5.14.

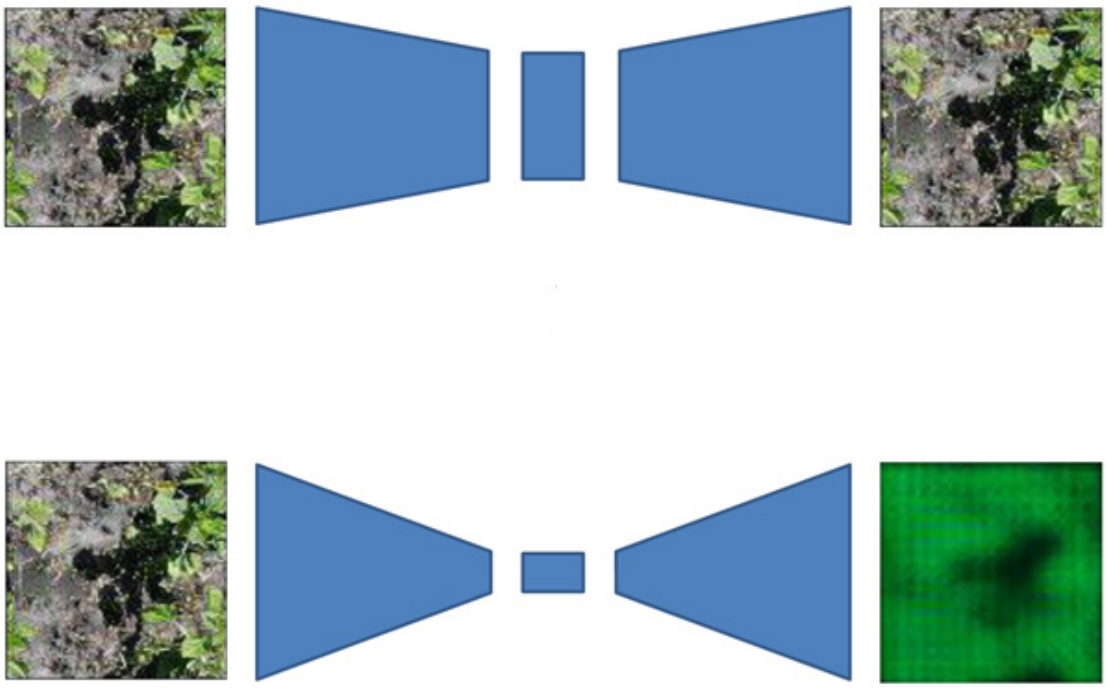


Figure 5.14 – The principle of operation of autoencoder (the upper figure is a small compression, the lower one is almost 400 times)

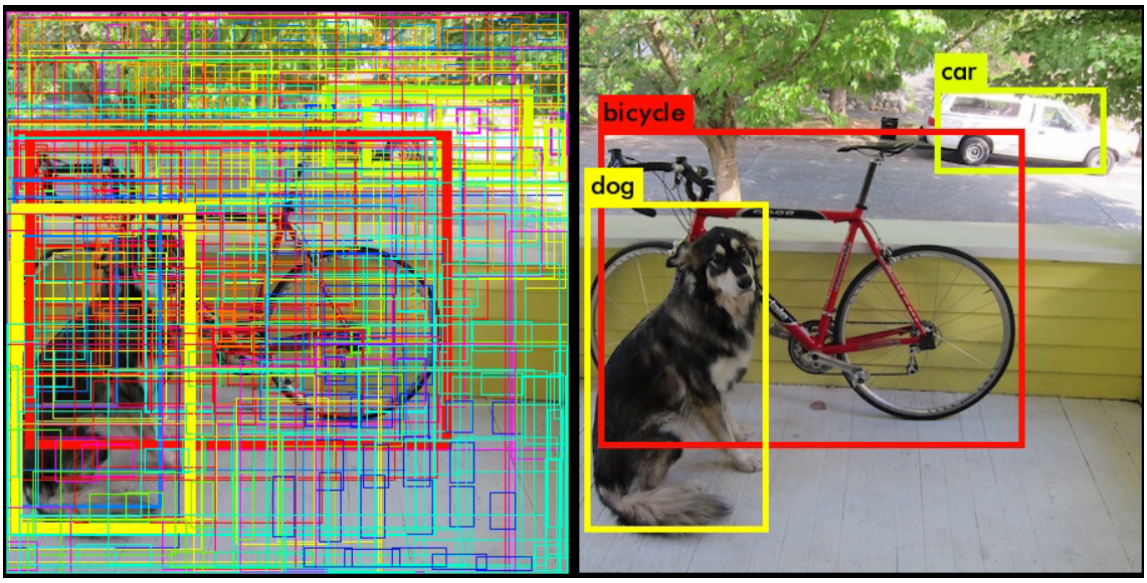
The input image is uploaded to a multilayer convolutional neural network, which transforms with a simultaneous reduction in dimensionality to a conditional "throat". And then, using a neural network with a mirrored architecture and the same parameters, it performs an inverse transformation from the same "throat" to the original image. The model is trained to ensure maximum similarity between the input and output images.

The article [39] describes an example of constructing a CAE by one of the authors of this book.

5.1.7 Videos analysis and recognition. YOLO.

YOLO ("You Only Look Once") is an intelligent real-time object recognition technology, especially effective in analyzing streaming video from video cameras, etc.

The basic principle of YOLO is to predict both bounding boxes with certain objects and the probabilities of assigning these objects to a certain class in each image in one model at once. The idea behind YOLO is that the network divides the image into a grid, and for each cell of that grid, it predicts bounding boxes and class probabilities that are compared to a certain threshold. Fig. 5.15 shows an example of object recognition with a threshold of 0.3.



a) b)

Figure 5.15 – Example of recognition of objects with a threshold of 0.3:
a) all possible potential objects; b) only the best objects
with an identification accuracy of at least 0.3

Currently, it is one of the main algorithms for recognizing objects in video streaming, on video cameras, etc. Fig. 5.16 shows an example of image recognition on city streets.

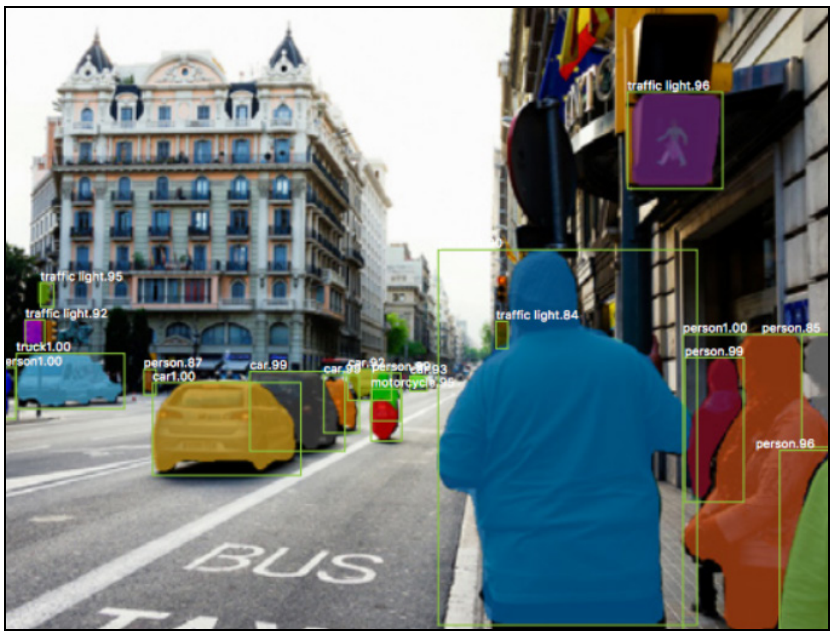


Figure 5.16 – Examples of work YOLO in the city

Usually, there are 5 variants of YOLO, depending on the number of parameters: "nano" (n), small (s), medium (m), large (l) and extremely large (x). In sources [42-44] there is an interesting overview of the differences between the YOLO versions since the first version in 2015 and there are interesting comparisons of the performance of these versions (Fig. 5.17).

Performance Comparison of YOLOv8 vs YOLOv5

Model Size	Detection#	Segmentation#	Classification*
Nano	+33.21%	+32.97%	+3.10%
Small	+20.05%	+18.62%	+1.12%
Medium	+10.57%	+10.89%	+0.66%
Large	+7.96%	+6.73%	0.00%
Xtra Large	+6.31%	+5.33%	-0.76%

#Image Size = 640 *Image Size = 224

Figure 5.17 – Comparison of the effectiveness of technologies YOLO8 and YOLO5

Figure 5.18 shows an infographics of the operations mentioned in section 5.1 above, in notation *S(I)*.

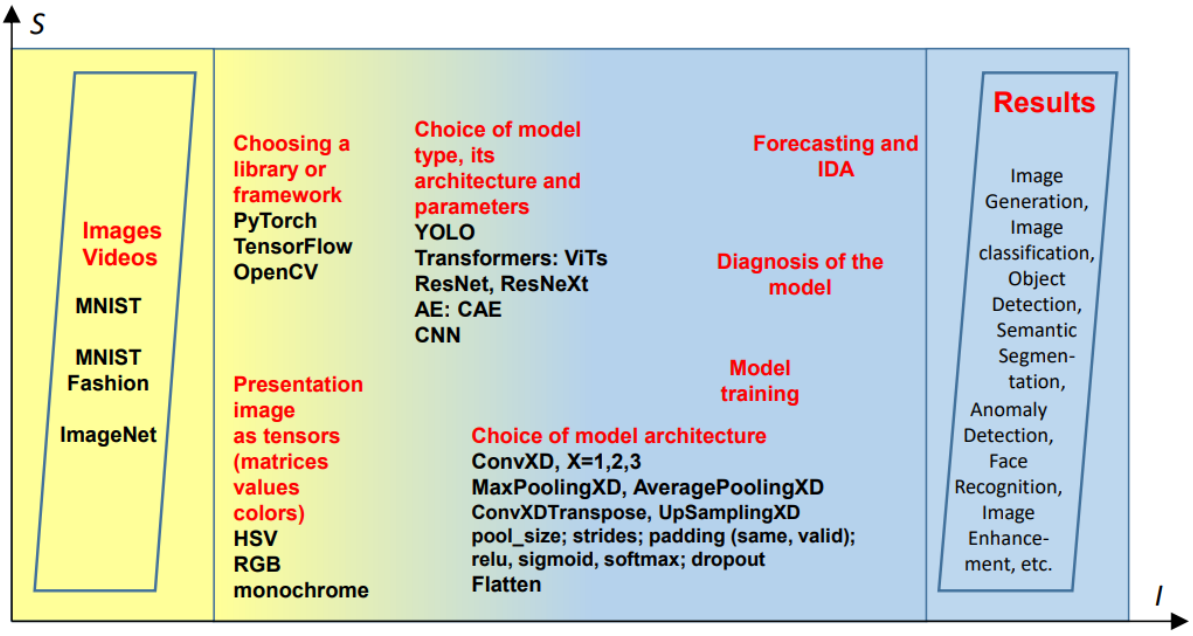


Figure 5.18 – Infographics with smart technologies for the preprocessing and classification of images and videos

5.1.8 Image generation and detection: GAN, VAE, Stable Diffusion

One of the newest areas of application of neural network technologies is the generation and detection of images and videos based on them.

Images and videos generated using deep neural networks are commonly referred to as "DeepFakes". Recently, this term has been extended not only to fully generated samples, but also to slightly altered real images or videos: faces, voices, sounds, backgrounds have been replaced and adjusted, any element of the image has been added, changed or removed.

Along with examples of the use of deepfakes for fraud and disinformation, there are also useful applications:

1) in films, animated films, commercials, music videos, and other entertainment industries;

2) to create educational content, in particular for various kinds of simulators;

3) to simulate realistic scenarios, also with the participation of people;

4) reconstruction of images from ancient or damaged documents;

5) generation of specified information in a convenient form for people with disabilities;

6) to generate virtual assistants and in the provision of various services;

7) for trying on clothes, hairstyles, makeup, etc.;

8) as avatars in social networks;

9) creation of samples for testing models and technologies, including deepfake detectors, etc.

Thanks to the efforts aimed at the development of intelligent information technologies for the task of generating and detecting deepfakes, the following are constantly being improved:

- intelligent prediction models;
- clustering methods;
- technologies for removing noise from images, videos, audio files;
- repair of corrupted videos;
- automatic colorization of black and white videos;
- improving the resolution of images and videos;
- file archiving;
- transforming files from one format to another;
- generating images and videos from text descriptions or, conversely, generating text from images and videos (see the Kaggle competition "Stable Diffusion – Image to Prompts»), etc.

Fig. 5.19 shows a series of images generated in Kaggle by one of the authors of the manual using the "StyleGAN2-ADA" model.

Figure 5.19 – Deepfakes generated using the "StyleGAN2-ADA" model from GitHub and from the Kaggle notebook, changing different style settings for a given photo

There are many different ways to generate deepfakes, but among them, it is worth highlighting the 3 best types of models and technologies, the principle of which is similar to the work of autoencoders (see subsection 5.1.6):

1. *GAN (Generative Adversarial Network)*. Each model has the following components: a generator that generates a deepfake of a given type from the noise distributed according to the normal law, and a discriminator (deepfake detector) that checks whether it is a fake or not. The generator learns and improves each time in such a way as to "outwit" the discriminator. The result is a deepfake that the discriminator does not consider to be a fake, or the probability that it is a fake is as low as possible. This is a competition between two powerful neural

networks, hence the name comes from here. Accordingly, the realism of fakes is very high, as well as the required computing resources and duration of work, compared to other approaches [45].

One of the most successful is the StyleGAN2 model, which does not so much generate a new image as focuses on changing certain styles of it (emotions on the face, background, lighting, skin color, face elements, etc.) (see Fig. 5.19).

2. *VAE (Variational Autoencoder)*. The principle of operation is the same as in convolutional autoencoders (see subsection 5.1.6), but at the throat there is a matrix, which is a probability distribution over the latent space. The VAE encoder compresses many images to the same set of multi-dimensional probability distributions, which are their counterpart in a sense. And then the VAE decoder randomly selects a number of parameters from this distribution and generates an output image from them. His work is described in more detail in the article [45]. One of the advantages is the high variety of images. Generally, VAE-based models are faster than GANs.

3. *Diffusion models*. Diffusion models are based on the formalization of the generation process in the form of a Markov chain. *Markov Chain* is a mathematical model of a stochastic process, where the future state of a system depends only on its current state, without taking into account its previous states. A given image undergoes a series of step-by-step transformations by adding Gaussian noise at each stage until it turns into white noise. And then the original image is reproduced from that white noise in reverse order. At each stage, the match is checked and, accordingly, the neural network that performs this reproduction is improved. Trained in this way, the model is able to generate quite realistic images from simple white noise. Generally, with each iteration, it adds different details, and the image gains more and more clarity. Provides high realism, but takes a bit of a long time to learn. Although some techniques for accelerating learning have emerged recently, however, it is still believed that GAN and VAE work faster. Diffusion models are described in more detail in the article [46].

Evaluating the performance of generative models includes assessing the quality and diversity of the samples generated. Three metrics are commonly used for this purpose: Inception Score (IS), Fréchet Inception Distance (FID), Precision and Recall for Distributions (PRD), Diversity Metrics (e.g., Mean Pairwise Distance or LPIPS - Learned Perceptual Image Patch Similarity), and Human eYe Perception Evaluation (HYPE). Each of these metrics provides unique information about the performance of generative models by criteria for quality, diversity, etc.

In addition to generating images, you need to highlight the video generation separately. It's not just a sequence of images. When generating deepfakes, you need to adjust the movements of the elements of the images to make them more realistic. Deepfake video detectors usually analyze the dynamics of movements for realistic behavior, in particular, from the point of view of the laws of physics and anatomical constraints. The movements of the arms, legs, especially of people, have many limitations. Many well-known misconceptions

are related to objects that cross parts of people's bodies, for example, a hand that passes through a microphone or deep into the surface of a table immediately emits a deepfake. It also analyzes how the person speaks and whether the sound really corresponds to what the person is saying (the most popular and simple way to deepfake is to replace the audio sequence of the video with another). Accordingly, video generators take this into account. As a rule, successful deepfake videos are quite short videos where the real video replaces the face of another person. At the same time, a person should look at the camera, and not stand sideways, be without glasses, a beard or a lush hairstyle (fitting facial hair is a significant problem), and almost not move. Then, even with the use of readily available models from GitHub, it is possible to generate quite realistic, high-quality deepfakes.

In 2020, Kaggle held a competition "Deepfake Detection Challenge", where it was necessary to detect a deepfake video. We recommend that you read the progress of solving the problem by its winners who took 1st place (Efficient Nets models), 3rd place (3 models EfficientNet-B7 with 3D CNN) and 5th place (SE-ResNeXT50, different 3D CNN).

Intelligent information technologies for analysis and generation (synthesis) of images allow solving many applied problems:

- analysis of CT tomography, fluoroscopy, analysis of ultrasound images, etc., for medical purposes;
- image recognition and conversion into information convenient for the perception of the blind and other people with disabilities;
- recognition of graphic information and its conversion into tabular or descriptive form;
- reconstruction of the image (inscriptions on ancient scrolls, etc.) from fragments from scans obtained in various ways, etc.

5.2 Intelligent Analysis of Text: Natural Language Processing and Generating

5.2.1 NLP: basic concepts, types of problems, data collection and preprocessing

Natural Language Processing (NLP) is one of the most popular types of data analysis problems nowadays.

NLP technologies refer to both text (in any language) and language, i.e., audio signals. But the most widespread use of NLP has recently become for the processing of natural language text.

A corpus is a large dataset (collection) of texts on a topic.

NLP solves the following *tasks*:

- translation from one language to another (audio, text);
- sentiment analysis;

– data searching or analysis, including web scraping (loading the page and removing blocks of useful text from it) and parsing (formalizing and structuring the text according to certain criteria and templates) (see, for example, the author's Kaggle notebook);
– data summarization;
– filling in missing text in sentences or continuing a set of sentences (e.g., T9 algorithm);
– text classification, deter analysis the author of the text (a human or AI?);
– filtering spam in the mail;
– detection of phishing sites;
– recognition of a text request in order to respond to it, as well as generating a response ("Quation-Answear" systems), support for the operation of systems for call centers;
– generating an image based on a text request;
– chatbots – generating information in dialog mode,
– creation of recommendation systems;
– NER (Name Entity Recognition) – recognition of named entities (geographical names, company names, names of people) and relationships between them based on natural language text;
– keywords mining for the georeferencing of the whole text [30];
– key phrases extraction from the text [32];
– augmentation of texts [35];
– search and selection in the text of certain parts of a sentence (nouns, verbs, adjectives, etc.) or parts in a word (prefixes, suffixes, etc.), etc.

An example of an NLP algorithm for, for example, text classification (see the implementation in Kaggle notebook by one of the authors in his dataset):

1. Find data.
2. Markup the data.
3. Clear data and perform other preprocessing.
4. *Tokenize* the text by dividing it into individual words or syllables.
5. Select and configure the model and choose the technology of its application.
6. Select and carry out post-processing.
7. Analyze outliers and, maybe, improve something at the previous stages.

Examples of text cleaning and preprocessing (this stage of bringing the text to a standard form is also often called *normalization*):

1. Convert all characters to lower case if the capital letter does not carry valuable information.
2. Spelling correction (examples: "goal", "goooooaaaall").
3. Remove punctuation marks.
4. Remove non-natural language characters (program code, numbers, hypertext links, emoticons, etc.).

5. Carry out *lemmatization*, that is, bringing all words to a single dictionary form (Fig. 5.20).

6. Removal of *stop words* – "the", "is", etc., which do not have specific semantics;

7. Apply *stemming* when words are reduced to the root by removing the variable part of the word form, by discarding unnecessary characters, usually suffixes or endings (see Fig. 5.20).

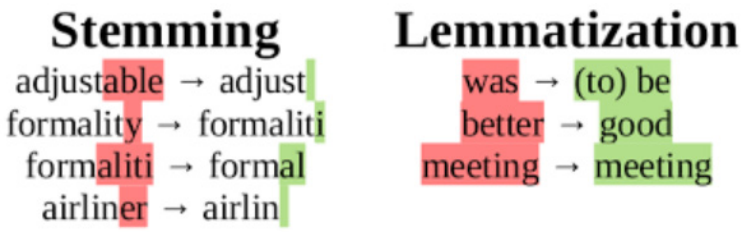


Figure 5.20 – Examples of stemming and lemmatization from Kaggle notebook

Examples of code for using many of these functions are available in the author's Kaggle notebook.

To automate these operations, the re (regular expression) and NLTK (Natural Language Toolkit) libraries are used. For more information on NLTK, see sorces [6, 47] and in documentation.

Large (powerful) language models usually do not require normalization operations.

5.2.2 Linguistic models and classification of natural language text

One of the most illustrative and popular NLP problems is the classification of natural language text.

5.2.2.1 Bag of Words.

One of the first and most popular NLP methods of text classification is based on "Bag of Words" (abbreviated: "BOW" or "BW"). The method consists in making a list-dictionary of unique words in the text. Then each sentence, paragraph or other part of the text is presented as a vector, in which 0 is placed if the word is a dictionary of unique words, and 1 – in the opposite case (Fig. 5.21). One of the main disadvantages of using BOW is ignoring the order of words and the relationships between them, which is very important for natural language text.

	good	movie	not	a	did	like
good movie	1	1	0	0	0	0
not a good movie	1	1	1	1	0	0
did not like	0	0	1	0	1	1

Figure 5.21 – Example of number vector formation in the BOW method from the Kaggle notebook

5.2.2.2 TF-IDF.

To increase the effectiveness of the BOW method, we decided to take into account the frequency of words appearing in the text in order to filter out commonly used words that are less valuable for analysis. This method is called "TF-IDF" because it ranks words by a metric that is the product of the "TF" and "IDF" scores:

- "Term Frequency" ("TF") is equal to the ratio of the number of times the term *T appears* in the document to the number of terms in the document;

- "Inverse Document Frequency" ("IDF") is equal to log (*N*/*n*), where *N* is the number of documents and *n* is the number of documents in which the term *T* appears.

An example of calculating TF and IDF indicators is shown in Fig. 5.22.

Sentence A. The car is driven on the road.

Sentence B. The truck is driven on the highway.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

a) b)

Figure 5.22 – An example of calculating TF and IDF indicators [48]: a) examples of 2 sentences; b) the result of the calculation of the indicator for these "TF" and "IDF"

As shown in Figure 5.22, words with a higher meaning of this criterion are more significant in terms of TF-IDF: "car", "truck", "road", "highway".

You can also practice this problem in the Kaggle contest "Natural Language Processing with Disaster Tweets (Predict which Tweets are about real disasters and which ones are not)", the task of which is to determine whether a given tweet is related to a disaster or not. An example is a notebook [49].

5.2.2.3 GloVe. Embeddings.

Most of the more effective NLP methods are based on the concept of embedding. In NLP, embedding is a number vector into which textual information tokens (words or syllables) are converted, taking into account the semantic meaning and typical relationships with other tokens in the natural language text of a given subject area.

Methods, models and technologies of GloVe, Word2Vec, transformers, in particular, BERT and others are based on embaddings. The way embeddings are calculated is different in different methods.

The most easy-to-understand concept is used in the GloVe (Global Vectors for Word Representation) method. A large corpus is taken (hundreds of thousands, millions, even billions of words – texts from Wikipedia, GitHub, etc.). Words are numbered and parsed in pairs. A matrix is created where each element (*i*, *j*) contains the number of times the word *j* occurs in the context of

the word *i*. The probability of this occurrence is then calculated. And then vectors are formed that characterize the probabilities of the appearance of other words in the context of a given one. This is a rather complex optimization algorithm, but it shows impressive results. An example of operations with the words "king" and "queen" is canonical, given, for example, in a blog. Fig. 5.23 shows the embedding of the word "king", determined by the GloVe method based on a corpus with 400 thousand unique English words from Wikipedia.

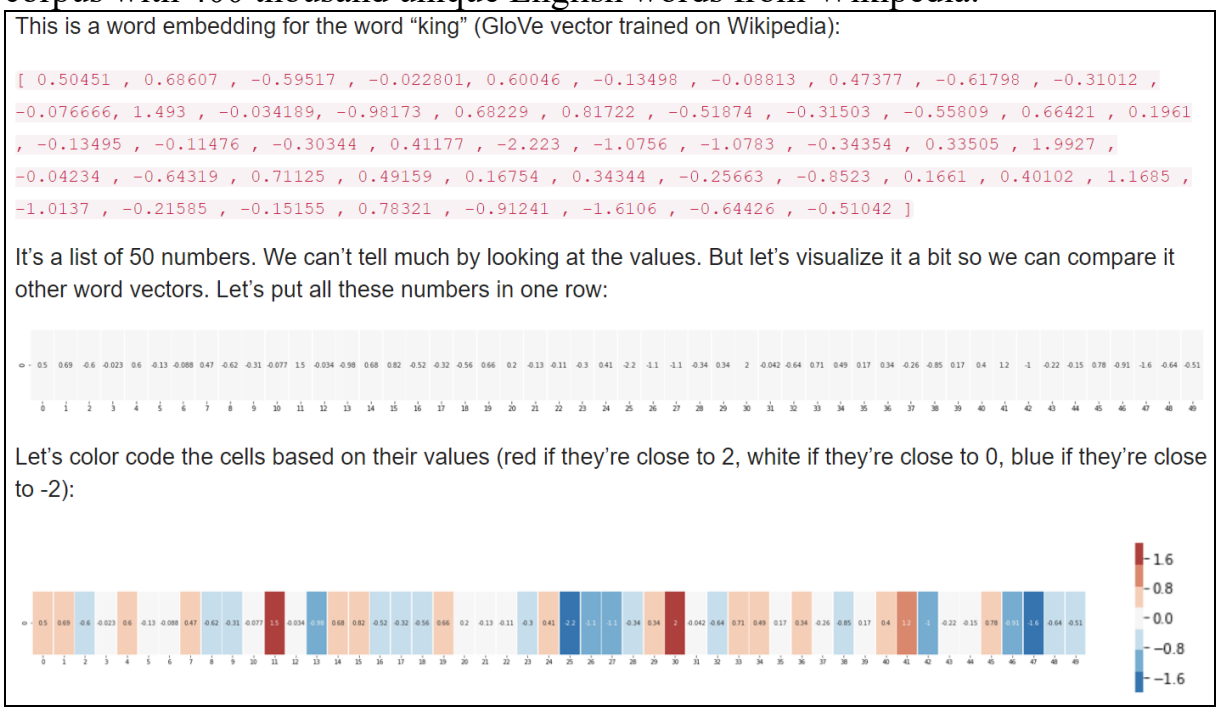


Figure 5.23 – Embedding of the word "king", defined by the GloVe method based on a corpus of 400,000 unique English words from Wikipedia (from the blog)

Figure 5.24 shows the canonical example of embedding operations.

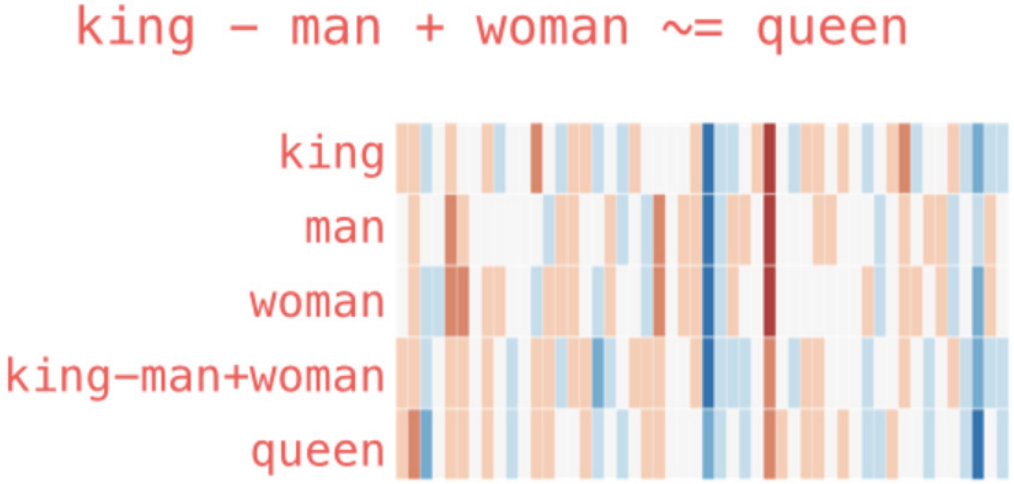


Figure 5.24 – Result of operations with embeddings "king" – "man" + "woman" and comparison of the result with embedding of the word "queen" (from the blog)

As can be seen from Fig. 5.24, if we apply element-by-element addition and subtraction of vectors according to the formula "king" – "man" + "woman", then the result, according to the blog, will be the closest to the word "queen" among all 400 thousand unique English words.

The article [50] gives another example: "Paris – France + Italy = Rome".

The GloVe method scales well on big data, but its effectiveness is manifested only with very large cases, and such training is long-term, and this is one of its main disadvantages.

5.2.2.4 Word2Vec.

More original and faster is the Word2Vec method. It combines 2 concepts: predicting the context by word and the word by context (Fig. 5.25):

- the CBOW (Continuous Bag-of-Words) method predicts a word by its context: the neural network takes each word as a target and analyzes the words in each sentence next to it (context), trying to predict this target word;
- skip-grams predict the context of a word (neighboring words) behind the word itself using its vector representation.

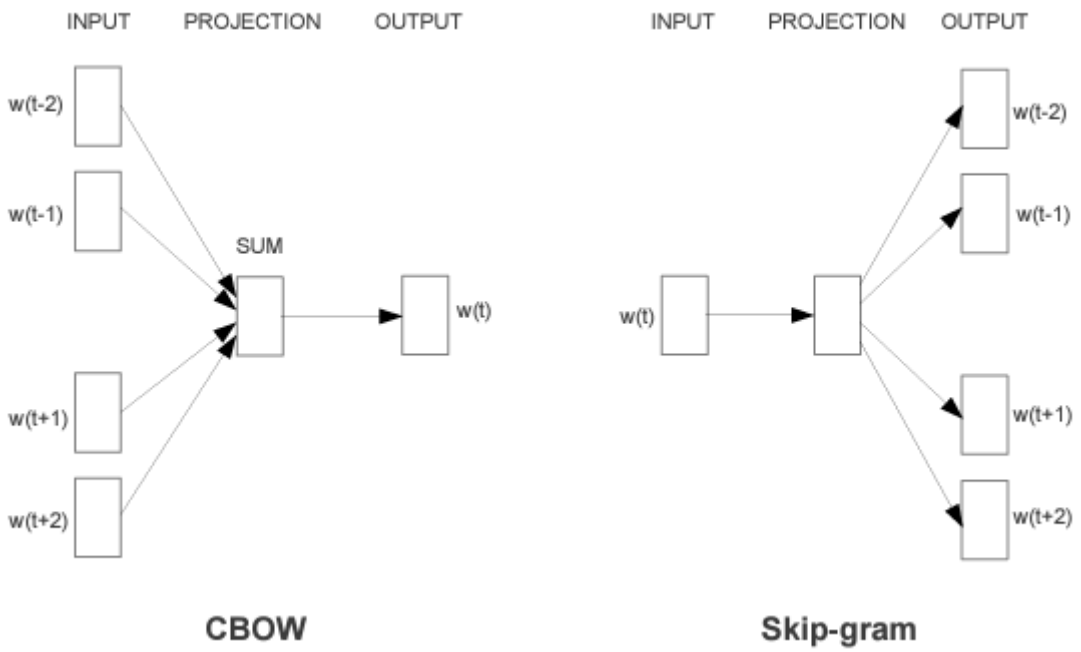


Figure 5.25 – Word2Vec method model architecture:
CBOW and skip-grams [50]

The way skip-grams are formed is similar to the way GloVe embeddings are formed, but, firstly, GloVe analyzes only pairs of words, and skipgrams analyze the entire adjacent context of many words, and secondly, GloVe analyzes the entire text globally, and skipgrams analyze only a given sentence or another part of the text each time.

It is also advisable to read the article [50], where this method was proposed back in 2013. Fig. 5.26 shows an example of foresight.

Relationship	Example 1	Example 2	Example 3
Country - Capital	France-Paris	Italy: Rome	Japan: Tokyo
Adjective - Comparative	big-bigger	small: larger	cold: colder
City - State	Miami-Florida	Baltimore: Maryland	Dallas: Texas
Person - Profession	Einstein-scientist	Messi: midfielder	Mozart: violinist
Chemical element - Symbol	copper-Cu	zinc: Zn	gold: Au
Company - Product	Microsoft-Windows	Google: Android	IBM: Linux
Company - CEO	Microsoft-Ballmer	Google: Yahoo	IBM: McNealy
Country - National dish	Japan-sushi	Germany: bratwurst	France: tapas

Figure 5.26 – Example from article [50]: predicting a word by context using Word2Vec trained on a corpus of 783 million words with skip-grams with a dimension of 300

5.2.2.5 Transformer

As mentioned above, the most up-to-date and powerful model for classifying data (both textual and graphic) is the Transformer model.

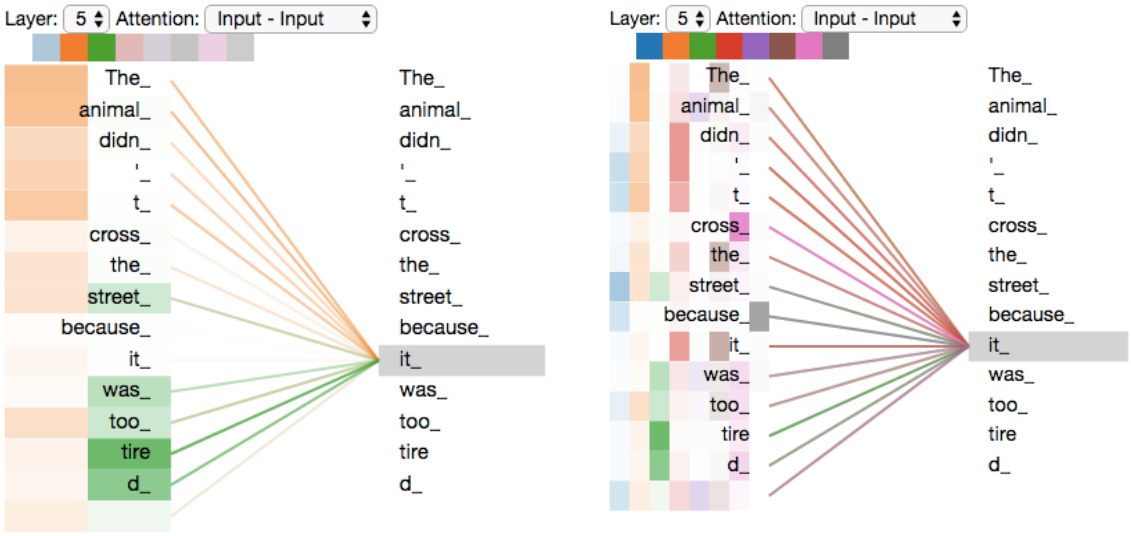
A very well-detailed and illustrated explanation of how transformers work in NLP is given in the blog. Its following important features can be distinguished:

1. Using the "encoder-decoder" architecture (the encoder transforms text to vector representation, and the decoder – vice versa: vectors to text).

2. The words themselves are processed separately, their position in the sentence is processed separately, which allows you to parallelize part of the calculations and significantly speed them up, due to the GPU or TPU;

3. Use of the Attention Heads Mechanism "attention") to account for the relationship between words in a sentence. To do this, a special neural network is used that evaluates the relationships between words and their importance for understanding the entire sentence and which words connect the parts of the sentence. This is the key feature of transformers, which have led to a revolution in the processing of natural language text, and subsequently in image processing.

The blog provides a good example for the sentence "The animal didn't cross the street because it was too tired". Fig. 5.27 shows a diagram of which words the word "it" is associated with, which attaches the last 4 words to the main sentence and is therefore important.



a) b)

Figure 5.27 – Analysis of which words the word "it" is most associated with in the sentence "The animal didn't cross the street because it was too tired" (from the blog): a) from 2 attention heads; b) with 8 attention heads

Typically, 8 different approaches ("attention heads") are used, as shown in Fig. 5.27, and then the results obtained are averaged and a single embedding is formed.

Transformer as a model is considered one of the best ("State-of-the-art") for both NLP problems and image-related tasks (this can also be seen in Fig. 5.12), where they are called "Vision Transformers" (ViTs). To do this, images are divided into fragments (patches) in the same way as text is divided into words, subwords or syllables. And then the entire powerful apparatus of transformers is used to find connections and dependencies between these patches. And this allows you to solve various problems: segmentation, detection and recognition of objects, as well as people, classification of objects and even generation of new images.

5.2.2.6 BERT.

A fairly modern and powerful solution is BERT (Bidirectional Encoder Representations from Transformers) models and technology. Bidirectionality means that the context of a word in a sentence is analyzed in both directions: both before and after it.

1. BERT uses sub-word tokenization ("WordPiece Tokenization"), which allows you to consider words at a lower level and take into account the morphological features of the language.

2. It uses its own tokenization system with additional marks, which depend on the problem to be solved (Fig. 5.28). The beginning and end of each sentence are also marked in a special way.

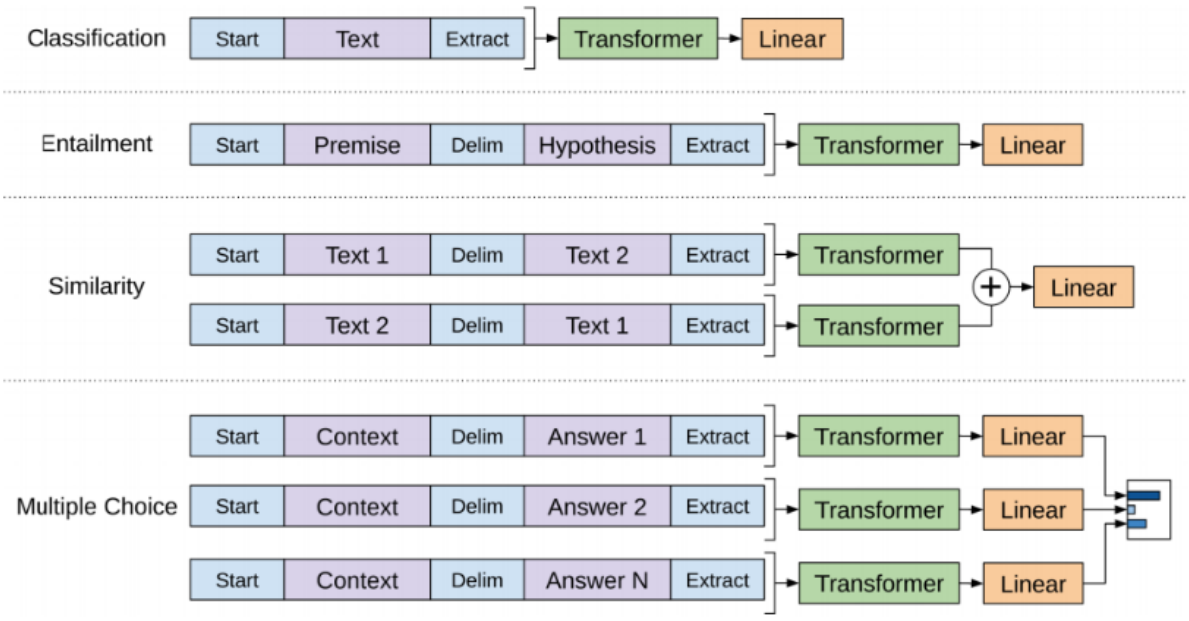


Figure 5.28 – Types of BERT tasks and features of token formation for them (from the blog)

3. The architecture of the model uses Dropout, which allows you to better predict missing words in a sentence (Fig. 5.29).

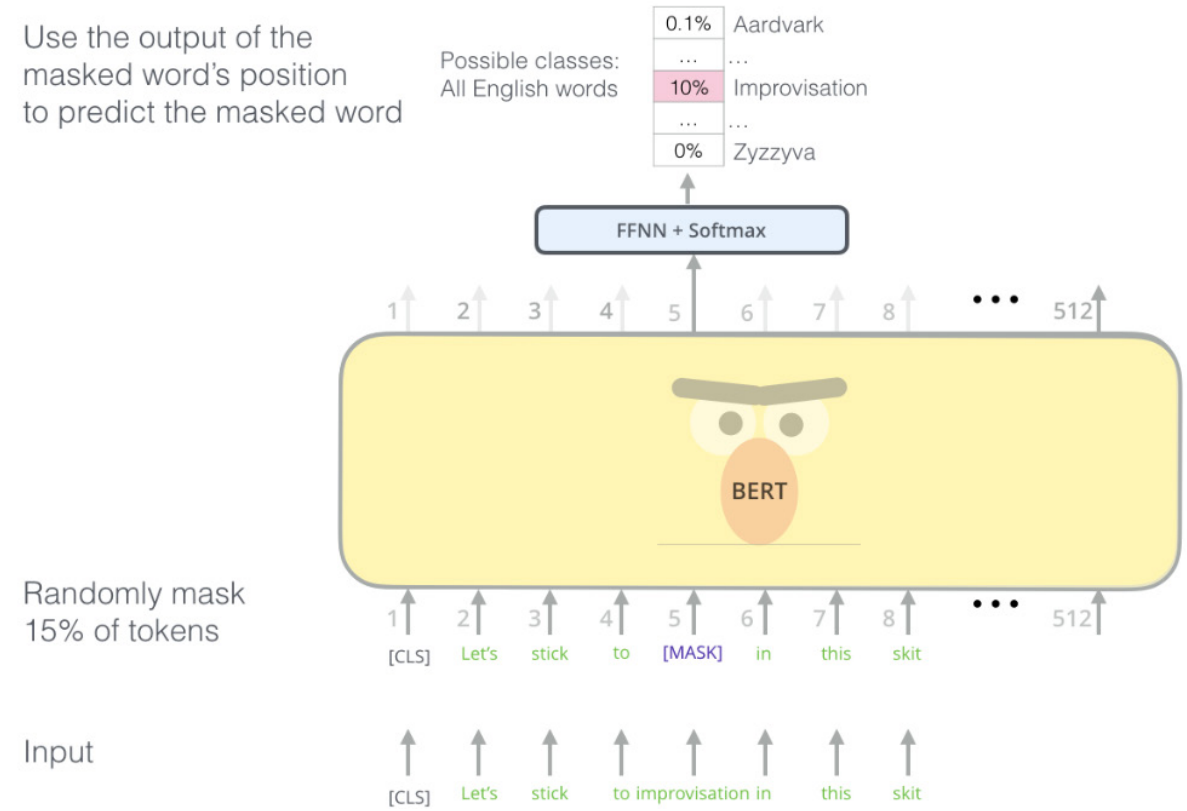


Figure 5.29 – Illustration of missing word prediction using BERT (from the blog)

A very nicely detailed and illustrated explanation of how BERT works is given in the blog.

5.2.2.7 Hugging Face (HF).

The Hugging Face (HF) collection contains a large number of pretrained language models BERT, GPT, T5, etc., which can be applied to different NLP tasks in different languages at once (Fig. 5.30).

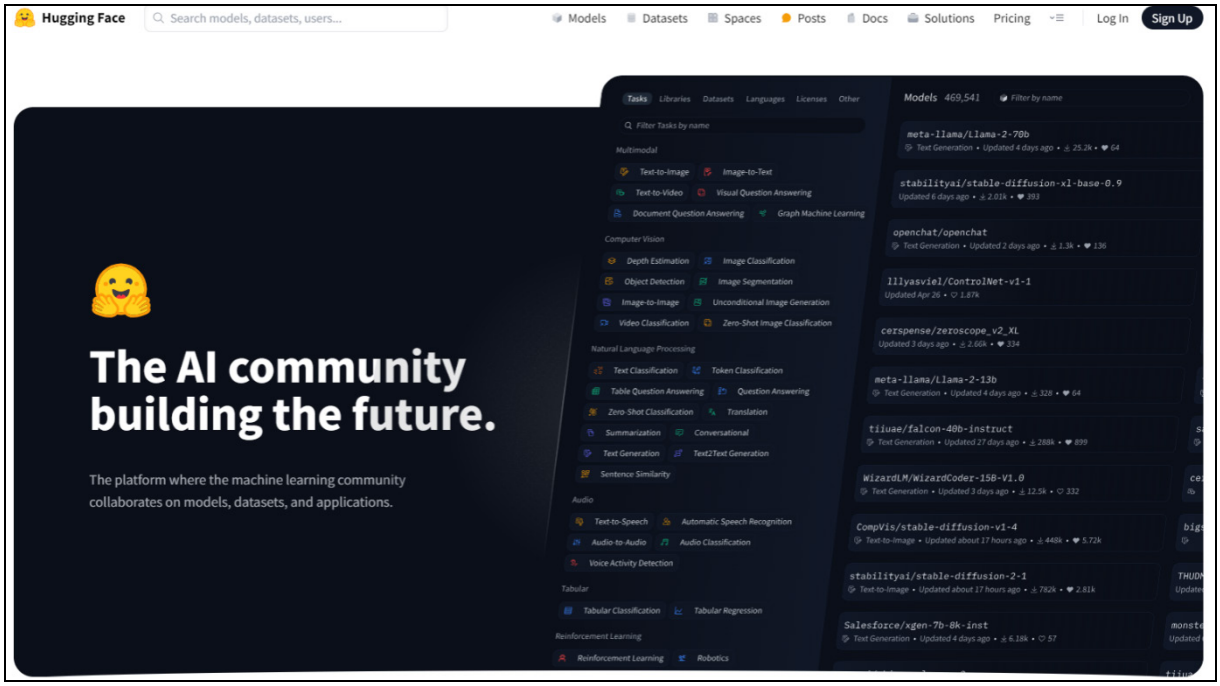


Figure 5.30 – Hugging Face (HF)

As of January 2024, HF contents:

- more than 600 thousand models, including more than 125 thousand different transformers;
- more than 140 thousand datasets.

Usually, the following algorithm is used to classify natural language texts using HF:

1. Download the data and separate the target from it.
2. Select and install the transformers library and pretrained models from the Hugging Face (HF) collection.
3. Perform data preprocessing, normalization and tokenization of text sentences (often synonyms) using the tools of the Hugging Face (HF) collection.
4. Apply the selected language model and get embeddings for the input from it.
5. Create additional features.
6. Combine embeddings from step 4 with other features from step 5 and target from step 1 to get a dataset for supervised machine learning.
7. Divide the data into training, validation, and test.
8. Train one of the models from Chap. 4 to predict a given target.

9. Predict test data.

10. Analyze the results. If the result is not satisfactory, then you can try to change something in the previous stages.

Examples of the use of this algorithm are given in the following notebooks, including author's ones:

1. Kaggle Contest Notebooks «Natural Language Processing with Disaster Tweets»;
2. Notebooks of the "NLP: Reports & News Classification" dataset;
3. Dataset notebooks «NLP with Disaster Tweets – cleaning data" for the Kaggle contest "Natural Language Processing with Disaster Tweets».
4. A very short notebook based on the simple transformers library: "Supershort NLP classification notebook" for Kaggle competition «Natural Language Processing with Disaster Tweets»).

Let's dwell on step 5 of the algorithm, which is somewhat specific for NLP problems.

5.2.2.8 FE in NLP tasks.

To effectively solve NLP problems, it is important to be able to take into account additional information, extract other features, in addition to embeddings. To do this, you can effectively use the NLTK library mentioned above.

Usually, features in NLP tasks are formed in one of the following ways:

1. Use of syntactic or grammatical features such as Part of Speech ("POS") or grammatical dependencies (for this you can use the NLTK, SpaCy, etc. libraries);
2. Statistical characteristics of syllables, words, sentences, paragraphs, documents in the corpus of texts in general (for this, you can use the NLTK, re, ordinary statistical analysis libraries, etc.); A good example is in the notebook, where the author tries to distinguish texts (essays) written by a machine (large language models) from texts written by students by features (in the Kaggle prize competition "LLM - Detect AI Generated Text");
3. The "TF-IDF" metric, which displays the frequency of unique words without taking into account commonly used words (see, for example, the comment on his decision by the participant who took 19th place (Top1.5%) in the Kaggle competition "Google AI4Code – Understand Code in Python Notebooks");
4. N-gram formation, etc.

N-grams in NLP are sequences of *N* elements (words or symbols) that occur in sentences, paragraphs, or other parts of the text. Normally, 2 numbers are given (N_1, N_2) where $N_2 \geq N_1$ which means searching for sequences that contain from N_1 to N_2 including elements.

For example, in Kaggle's "LLM - Detect AI Generated Text" competition, most public best solutions use (3, 5) and (3, 6)-grams using the **TfidfVectorizer (ngram_range=(3, 5))** command from the sklearn.feature_extraction.text package.

A notebook on keyword extraction, created with the help of one of the authors of the manual, used (N,N)-grams. One of the results is shown in Fig. 5.31 [32].

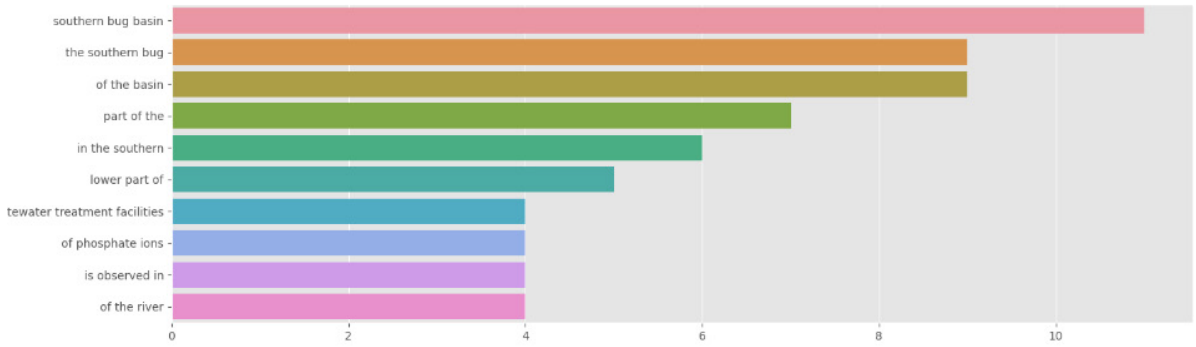


Figure 5.31 – Examples of (3, 3)-gram words and their number in the corpus of words [32]

5.3 Large Language Models (LLM) and Chatbots

A real revolution has recently been caused by the emergence of ChatGPT ("Chat Generative Pre-training Transformer") version 3.5 (Fig. 5.32), and later 4.0 from OpenAI.

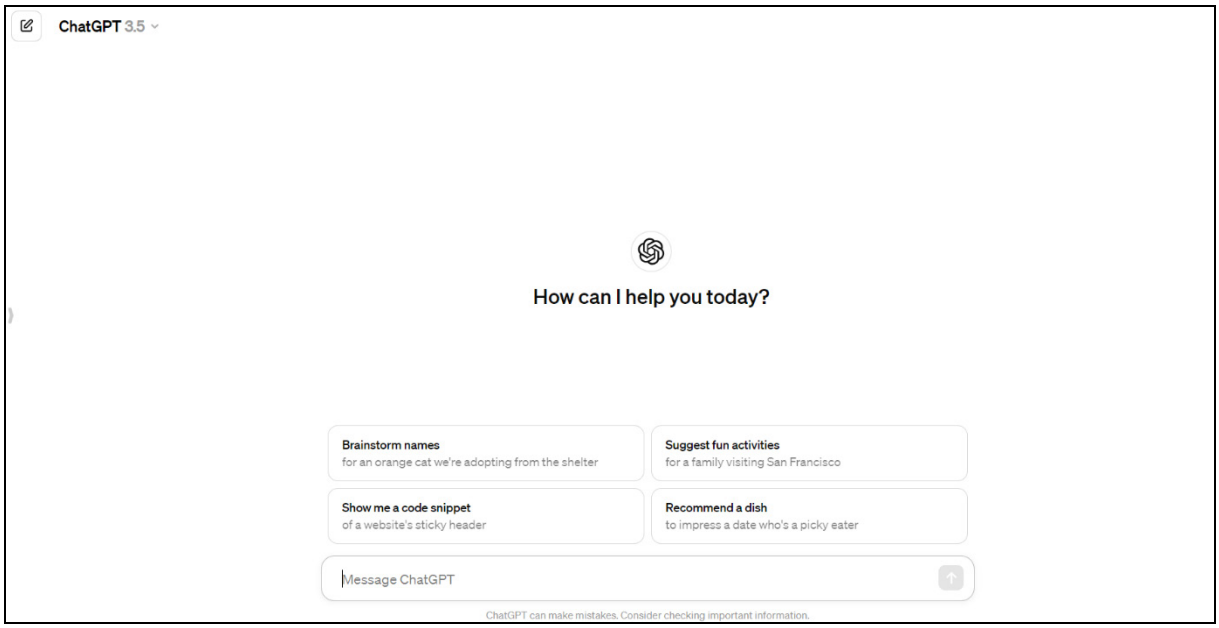


Figure 5.32 – ChatGPT 3.5

Millions of people have joined its testing. Billions of dollars began to be allocated for the development of similar services by all leading IT companies. Governments and individual companies initially tried to ban this service, but the "Pandora's box" has already been opened. Realizing that progress could not be stopped, they focused on developing rules and restrictions. For example, chat-

bots began to write more often: "I can't answer because it violates ... phrase the question in a different way." "I can't answer, it's better to contact ..." etc.

The following features of this system are especially valuable:

1) The ability to ask questions in your native language, although experience shows that the answers to questions are the most complete;

2) Detailed answers with the ability to ask clarifying questions and receive continuation of explanations in a dialogue mode;

3) You can edit a previously asked question, rather than writing a new one, and get a different answer;

4) You can click "Regenerate" ("Restart") and generate another answer option, and the system will ask if the new version is better, reminding that all users are its testers, and it constantly self-learns on them;

5) Generates Python programs at the level of a fairly professional programmer who knows all the most interesting programs from GitHub, Kaggle, StackOverFlow, etc.;

6) Allows native translation from one language to another in different styles;

7) Allows you to shorten your own text to a given number of words or sentences;

8) There are convenient buttons for copying the entire answer or just the program code;

9) All previous chains of questions and answers of the user are stored in their profile under the names assigned to them by the chatbot, but the user can rename them;

10) Availability of API access (paid, but inexpensive), which allows you to significantly expand the possibilities of its application in your applications;

11) Ability not to monitor the literacy of the query – compare the results of queries (good and short with errors in the text):

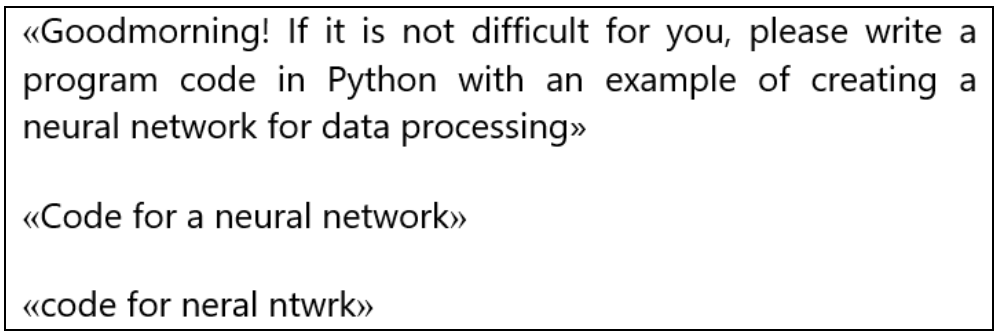


Figure 5.33 – Variants of prompts for the ChatGPT to obtain the simple Python code for the neural network building

12) The ability to ask a question, simply giving the input data that is available and the output that needs to be obtained, and he himself will guess how to write a program that will carry out such a transformation;

13) It can draw a primitive graph to illustrate the answer, but using pseudographics (from symbols from the keyboard), just to convey an idea, but there are also opportunities to integrate with other services that do it much better, or write code for high-quality visualization of the results in Python.

Of course, there are also disadvantages (there are fewer of them in the paid version than in the free version). "*Hallucinations"* are common, when the chatbot does not know the answer, but, taking advantage of the fact that it is a language model (a good "writer"), begins to quite realistically generate text in response, for example, web addresses of datasets or articles used in the answer, titles of books, etc., which have never existed, so all its answers need to be checked. In addition, there are errors in the answers for various reasons:

- due to problems in the material on which he studied and which also has errors;
- due to the problems of the chatbot model, which is constantly evolving;
- due to a misunderstanding of the request;
- due to user errors in the formulation of the request;
- due to the obsolescence of the material, especially when it comes to dynamically developing software libraries.

There are also limitations of the functionality itself in terms of the output format, but they can be eliminated by other services. Most of the typical disadvantages (duration of text generation, etc.) are absent in the paid version.

It is these shortcomings that still leave the existence of the profession of programmers relevant, because:

1) You need to be able to ask the right question in order to get a really effective answer, and for this you need to navigate the question and know exactly what to ask, for example, compare the answers to the queries:

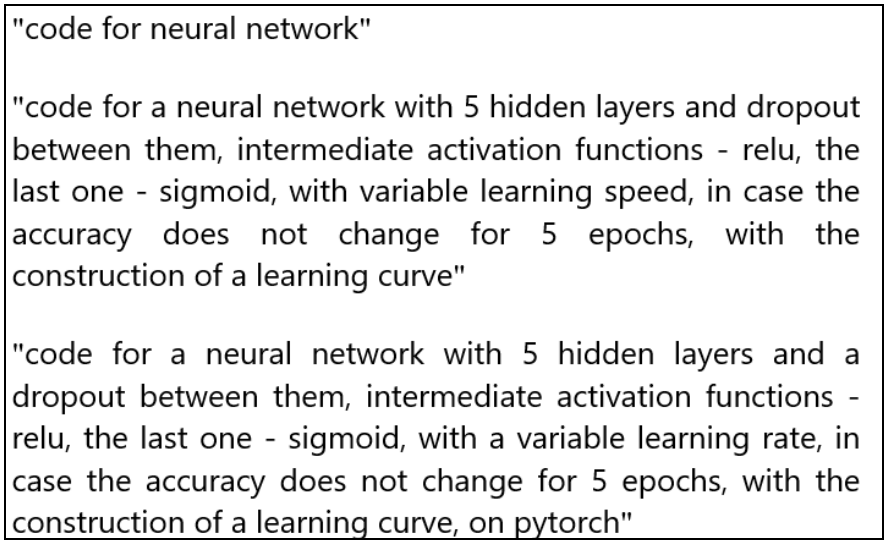


Figure 5.34 – Variants of prompts for the ChatGPT to obtain the more extend Python code for the neural network building

2) A chatbot will not write the entire program ready to be used along with other blocks, it will only give key blocks that an experienced programmer must combine;

3) You need to be able to recognize hallucinations and errors in the answer and guess what they need to be replaced with (theoretically, you can write to him what mistakes there are and he corrects himself, but sometimes he gets hung up – advises the previous first, also erroneous, answer);

4) Many companies categorically prohibit writing examples of data from these companies in a query so that it writes code for their processing – the programmer himself must come up with an analogue and then transfer it to the necessary data;

5) Many companies prohibit the use of chatbots for writing code at all, otherwise there are serious copyright problems for this code;

6) Someone needs to write new programs for such services, create new language models and datasets on which they will learn;

7) Often, answers are needed for a specific very narrow and modern field with its specific terminology, which the chatbot does not know enough about.

Chatbots like ChatGPT 3.5 are built on the use of the "Large Lingual Model" (LLM). Examples of LLMs at the moment are large multilingual models GPT3, GPT4, BERT, RoBERTa, XLNet, etc. The LLM for ChatGPT 3.5 is GPT3. Before GPT3, there were previous versions as well, such as GPT2. See, for example, the author's example [51] – a notebook using GPT2, BERT and XLNet for text generalization in 2022 in Kaggle, developed before the advent of ChatGPT 3.5.

ChatGPT has a "temperature" parameter that adjusts the degree of variety of responses.

Two relatively new directions in this area have emerged and are rapidly developing:

- *Prompt Engineering* is a set of knowledge and skills for generating a series of requests to the chatbot that will provide a truly relevant and most useful answer;
- *LLM Fine Tuning* is a technology of "learning" or continuing to teach a large language model on a given subject area, on a special corpus of words.

There are open-source LLMs "open-source" for example: LLAMA2 (https://ai.meta.com/llama/) from Meta (versions 7B, 13B and 70B), Falcon (https://falconllm.tii.ae/), Mistral (https://docs.mistral.ai/), MPT-7B (https://www.mosaicml.com/blog/mpt-7b) and many others [52, 53].

In the article of one of the authors of the manual, there is an analysis of the capabilities of LLM to generate training data for solving problems of the Kaggle competition "LLM — Detect AI Generated Text": GPT 3.5 Turbo; GPT 4; Mistral 7b Instruct; OpenAI text-ada-001; OpenAI text-babbage-001; OpenAI text-curie-001; OpenAI text-davinci-001; OpenAI text-davinci-002; OpenAI text-davinci-003; Google BARD (Gemini); Google PaLM; Claude Instant 1; Intel Neural Chat 7b v3.1; LLaMA 2 70b; Falcon 180b.

Figure 5.35 shows an infographics of the operations mentioned in sections 5.2, 5.3 on NLP problems in the *S(I)* notation.

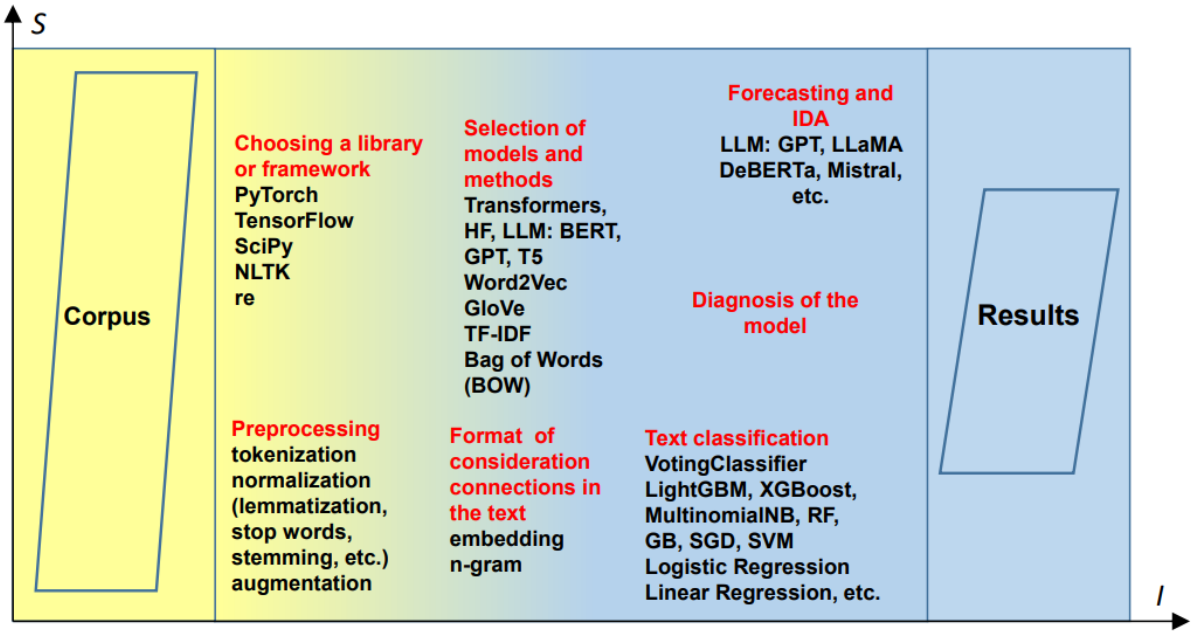


Figure 5.35 – Infographics of technologies for preprocessing and intelligent analysis of natural language text

5.4 Intelligent Analysis and Forecasting of Time Series

5.4.1 Basic concepts and types of problems

The problems of time series analysis and forecasting are among the most common that need to be solved. Often, it is important not only to build a model that can accurately and reliably predict data series, but the model itself can be valuable. The identified period and frequency of seasonality, the type of best model can say a lot about the series itself.

Important concepts are the spacing of the series and the forecasting horizon.

The spacing of the series is the difference between adjacent values. A number of models, such as ARIMA, require that the step be the same and that there are no missing values. For example, you can perform imputing from the Sklearn library.

The forecast horizon is the number of steps for which a forecast is made. It is important that these can be not only future values. This is the basis for the concept *of cross-validation diagnostics* of a series model that uses, for example, the Prophet library. The task is seen as a supervised task. The horizon is the values within the existing data interval – different parts of the data are selectively taken and used as validation, and then analyzed as the model predicts them well, using all other data for training (Fig. 5.36).

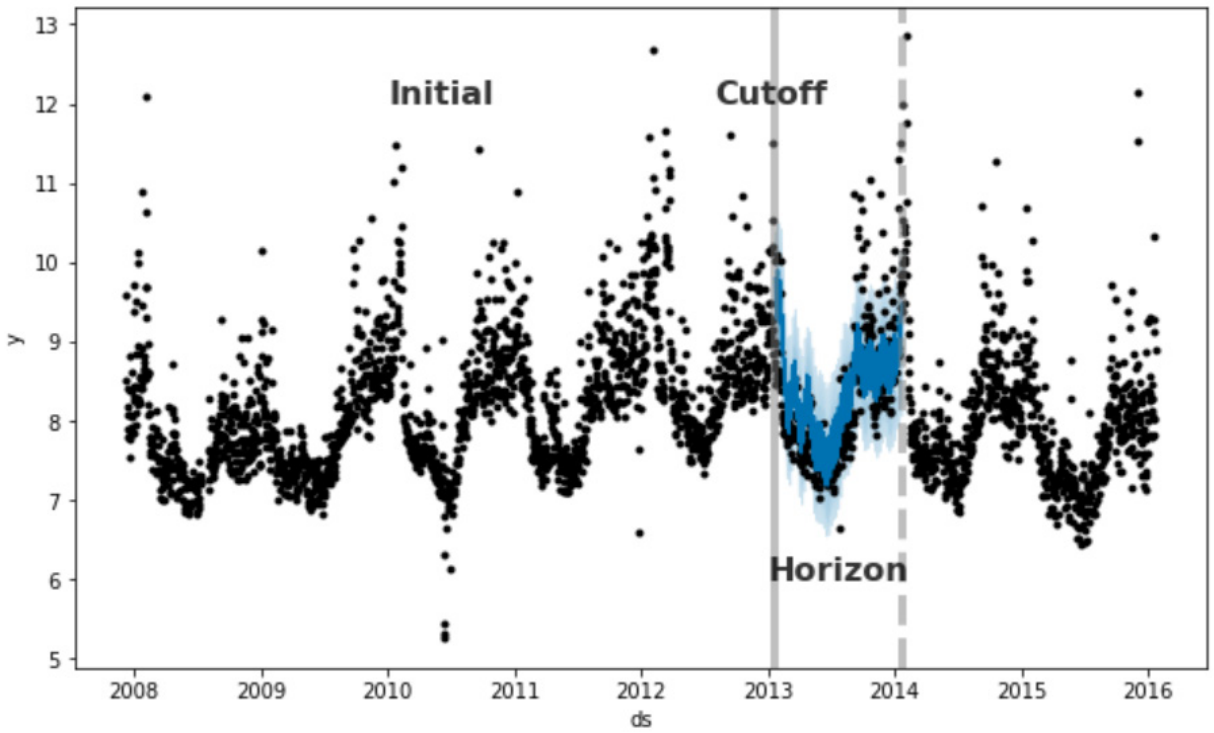


Figure 5.36 – Cross-validation in time series diagnostics with a horizon within the interval of known observational data [54]

Popular examples of time series forecasting problems are the following:

- forecasting of environmental parameters monitoring data: meteorological data, water status, atmospheric air status, solar activity, etc.;
- forecasting the exchange rate of cryptocurrencies or other as sets in financial markets;
- forecasting the values of device indicators in various technical systems;
- prediction of the results of analyzes of medical devices to predict the condition of patients;
- forecasting the number of patients during pandemics or epidemics in different regions to predict the extent of the disease and analyze possible directions of its spread;
- forecasting changes in radio signals, in particular of cosmic origin, etc.

5.4.2 EDA and FE for time series.

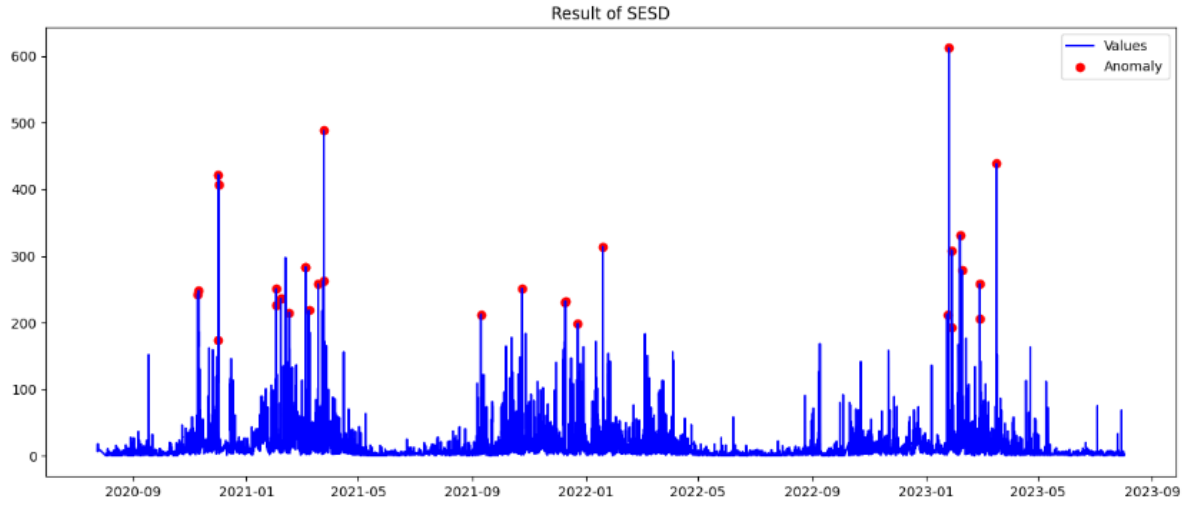
As a rule, the analysis of the time series is carried out in the following stages:

1. *Analysis of missed values and regularity of the step*.

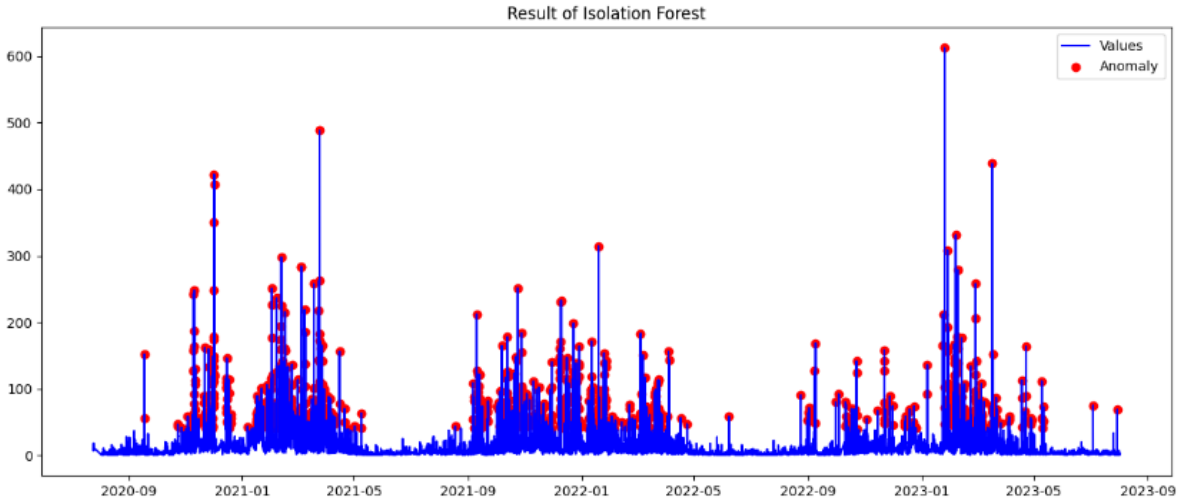
The ARIMA model requires regular data, while Prophet does not.

2. *Detection and analysis of series anomalies*.

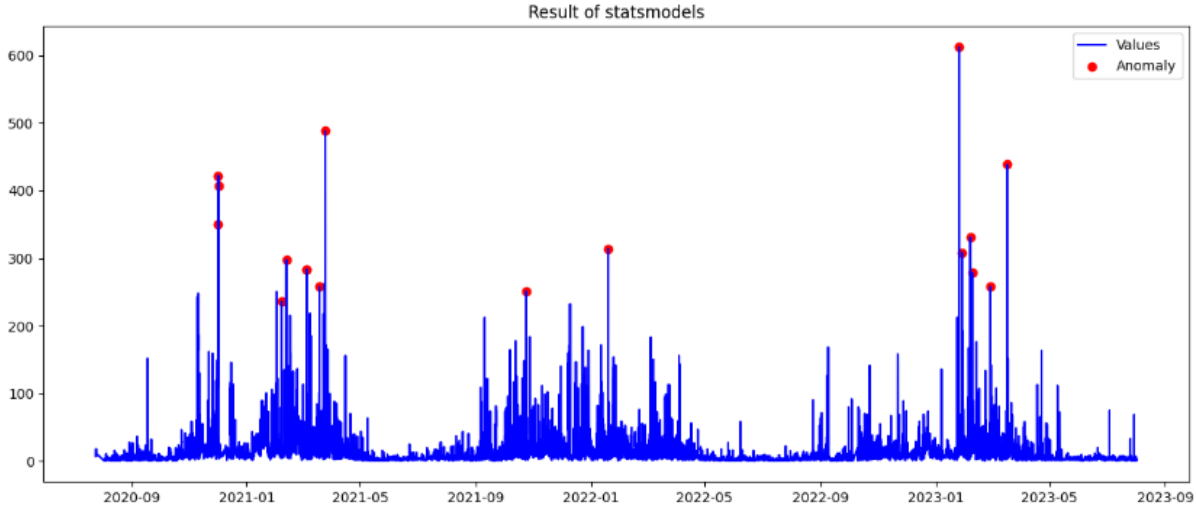
The simplest way is the one mentioned in Chap. 3 filtering by quantiles P10, P05, P90, P95, etc., but there are also special libraries [55] (Fig. 5.37):



a)



b)



c)

Figure 5.37 – Automatic identification of anomalies in the value of dust concentration in the ambient air using various libraries according to the data of the public monitoring network of the state of atmospheric air EcoCity from a notebook «Anomaly detection for air pollution» [55]: a) SESD; b) Isolation Forest; c) the method «seasonal_decompose» from the statsmodels library

- SESD (Seasonal Hybrid ESD) – the method with a combination of seasonal adaptation with an Exponential Smoothing and Extreme Studentized Deviate (ESD) to detect deviations from the expected data distribution;
- IsolationForest (library Scikit-learn) – the method for detecting anomalies by isolating anomalies in the data using a decision tree model in the entire feature space and without any regularization;
- statsmodels – a library that contains various statistical functions and models for anomaly detection, and a residual analysis method (seasonal_decompose).

In addition, anomalies can be found manually visually on the chart (it is worth using the interactive graphs of the plotly library), and then find confirmation of these anomalies. For example, the reasons for the abnormal fall in the bitcoin rate can usually be found in the news (Fig. 5.38).

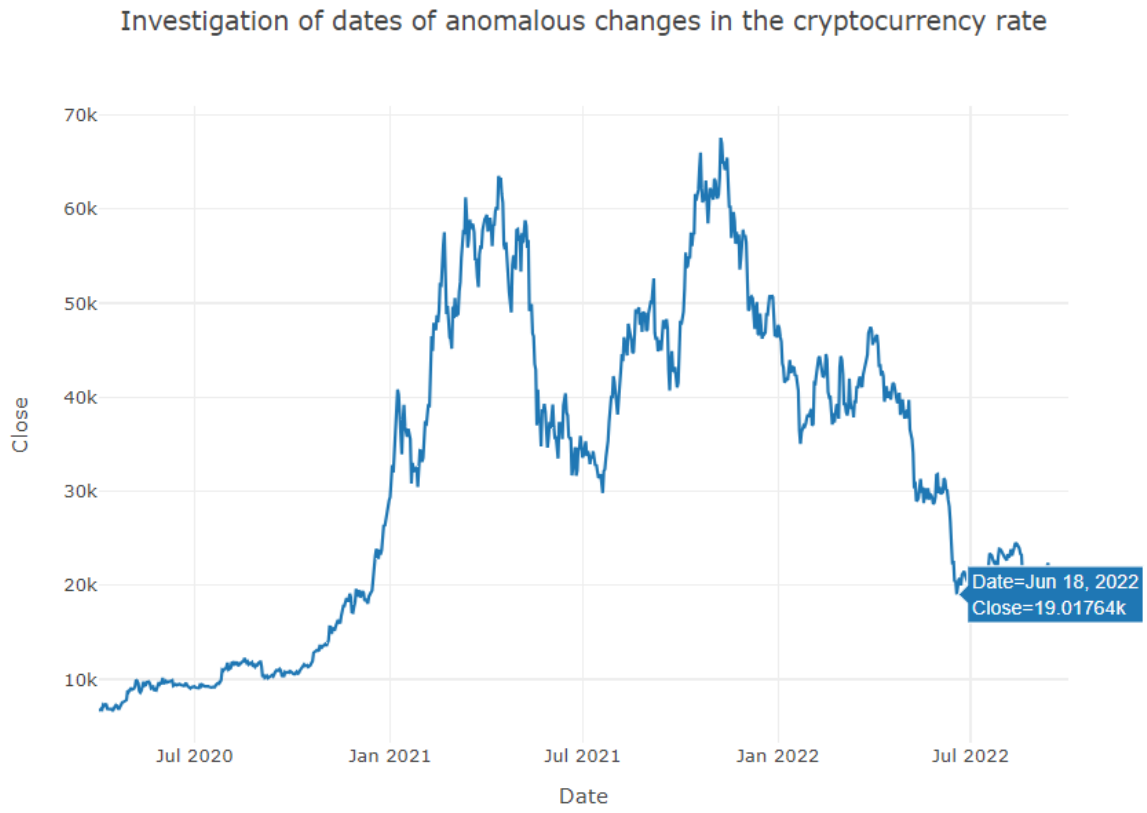


Figure 5.38 – Sharp drop in the bitcoin exchange rate in June 2022 [56] due to the collapse of the Celsius Network crypto exchange, the Tron stablecoin (USDD), etc. [57]

3. *Analysis of the law of distribution* of the series. Is it normal? Normalize data, if necessary.

To analyze the normality of the distribution law, there is a function **stats.probplot(x, dist=stats.norm)**. It calculates quantiles. If the distribution law is normal, then these quantiles line up. If this condition is not met, then the Box-Cox transformation (**stats.boxcox(x)**) ensures the alignment of these quantiles (Fig. 5.39).

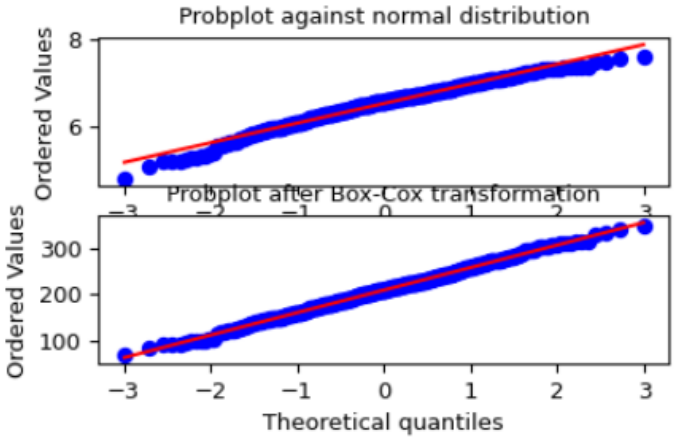


Figure 5.39 – The result of applying the Box-Coxmethod to the normalization of the time series

Also, it is important to compare the distribution laws of training and validation data. If they are too different, the model will not be effective.

4. *Analysis of the stationarity of the series*, although for ARIMA this step can be skipped – there it is analyzed automatically. Apply to stationarize this series, if necessary.

A stationary series is a series in which statistical indicators are invariable over time. Therefore, the statistical indicators of different samples of the series will be the same or comparable within a given error. To do this, the Dick-Fuller test is used, which is performed by the function **statsmodels.tsa.stattools.adfuller** (Fig. 5.40).

```
result = adfuller(series.values)

print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if (result[1] <= 0.05) & (result[4]['5%'] > result[0]):
    print("\u001b[32mStationary\u001b[0m")
else:
    print("\x1b[31mNon-stationary\x1b[0m")
```

a)

```
ADF Statistic: -1.571470
p-value: 0.497907
Critical Values:
        1%: -3.438
        5%: -2.865
        10%: -2.568
Non-stationary
```

b)

```
ADF Statistic: -31.128922
p-value: 0.000000
Critical Values:
        1%: -3.438
        5%: -2.865
        10%: -2.568
Stationary
```

c)

Figure 5.40 – Dick-Fuller test on stationarity of the time series of the bitcoin exchange rate (feature "Close") for 2020-2022 from the author's notebook: a) Python code, b) the result of checking the series; c) the result of checking the first difference of the series

5. Analysis of the seasonality of the series: is there a periodicity and with what period? There may be several different seasons at the same time [58]. Remove seasonal components, if necessary (depending on the models used).

The main difference between time series models and other machine learning models is that they specialize in taking into account the seasonality of series. It is on periodic series that time series models demonstrate higher accuracy than the universal data models described in Chapter 4. There is a typical test to see if the *x* series has at least one seasonal component. For this, there is a function **statsmodels.tsa.seasonal.seasonal_decompose(*x*, *T*),** where *T* is the period of the series in its steps (hour, day, ...) (Fig. 5.41).

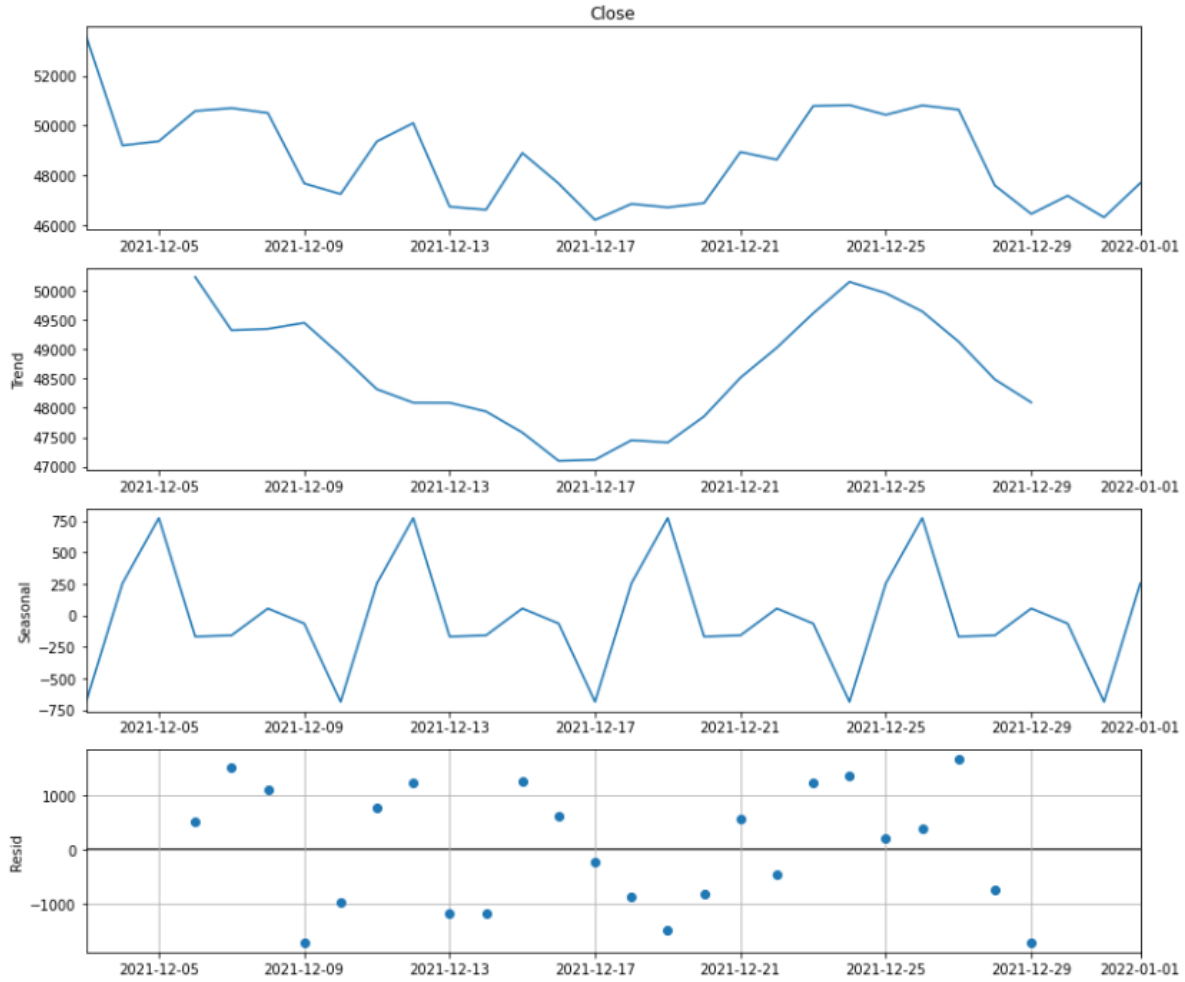


Figure 5.41 – The result of decomposing data on the bitcoin exchange rate in December 2021 using **seasonal_decompose** (from the author's notebook)

6. If the series was transformed at stages 3-5, then it is worth repeating steps 1-5.

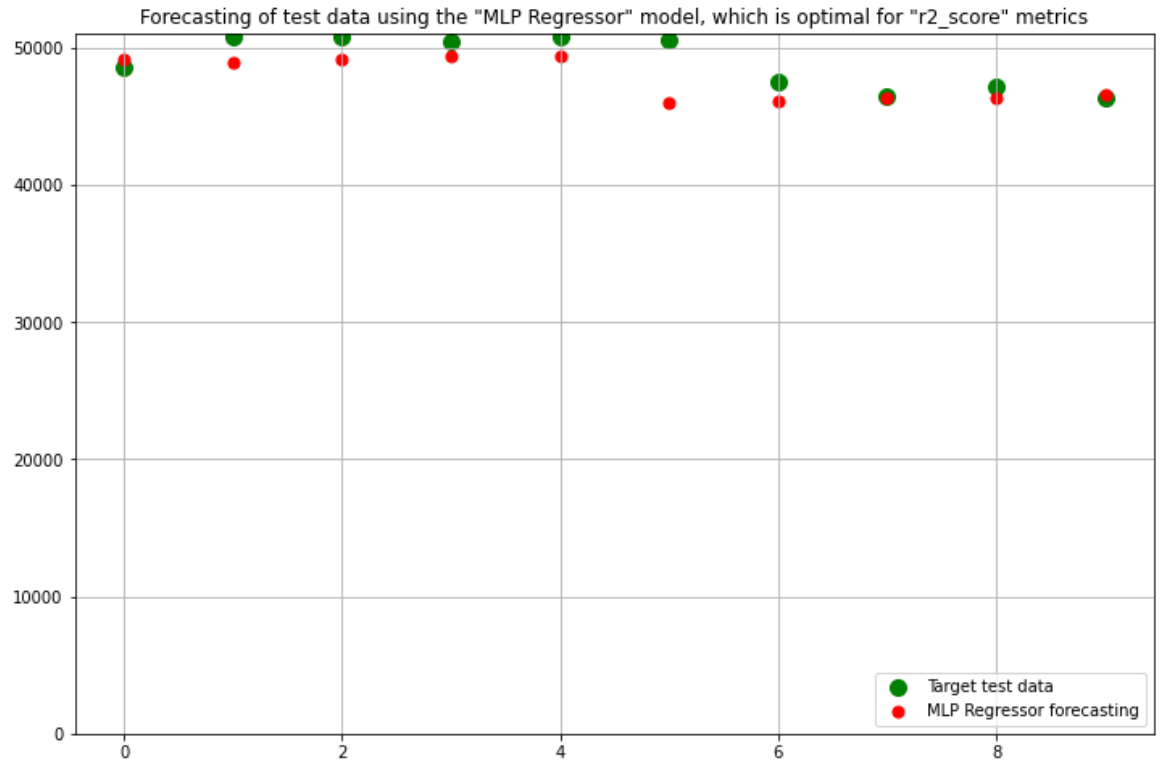
7. FE: Synthesize new features if section 4 models are used.

It is popular to use the TSFresh library, which allows you *to synthesize* up to 1200 features for a given time series. After analysis and filtering, as a rule, 50-100 features are left, which are quite interesting and effective for forecasting. For example, see author's examples of forecasting the average daily bitcoin exchange rate according to several years [11, 36, 59] in 2020-2021, which made it possible to predict the daily rate 10 days ahead with an accuracy of 2.44%, us-

ing about 60 features, almost 50 of which were obtained using the TSFresh library from the values of the "Close" feature (Fig. 5.42).

```
['index__sum_values',
 'index__abs_energy',
 'index__median',
 'index__mean',
 'index__root_mean_square',
 'index__maximum',
 'index__absolute_maximum',
 'index__minimum',
 'index__benford_correlation',
 'index__quantile__q_0.1',
 'index__quantile__q_0.2',
 'index__quantile__q_0.3',
 'index__quantile__q_0.4',
 'index__quantile__q_0.6',
 'index__quantile__q_0.7',
 'index__quantile__q_0.8',
 'index__quantile__q_0.9',
 'index__cwt_coefficients__coeff_0__w_2__widths_(2, 5, 10, 20)',
 'index__cwt_coefficients__coeff_0__w_5__widths_(2, 5, 10, 20)',
 'index__cwt_coefficients__coeff_0__w_10__widths_(2, 5, 10, 20)',
 'index__cwt_coefficients__coeff_0__w_20__widths_(2, 5, 10, 20)',
 'index__fft_coefficient__attr_"real"__coeff_0',
 'index__fft_coefficient__attr_"abs"__coeff_0',
 'index__value_count__value_0',
 'index__value_count__value_1',
 'index__range_count__max_1__min_-1',
 'index__count_below__t_0',
 'Close__sum_values',
 'Close__abs_energy',
 'Close__median',
 'Close__mean',
 'Close__root_mean_square',
 'Close__maximum',
 'Close__absolute_maximum',
 'Close__minimum',
 'Close__benford_correlation',
 'Close__quantile__q_0.1',
 'Close__quantile__q_0.2',
 'Close__quantile__q_0.3',
 'Close__quantile__q_0.4',
 'Close__quantile__q_0.6',
 'Close__quantile__q_0.7',
 'Close__quantile__q_0.8',
 'Close__quantile__q_0.9',
 'Close__cwt_coefficients__coeff_0__w_2__widths_(2, 5, 10, 20)',
 'Close__cwt_coefficients__coeff_0__w_5__widths_(2, 5, 10, 20)',
 'Close__cwt_coefficients__coeff_0__w_10__widths_(2, 5, 10, 20)',
 'Close__cwt_coefficients__coeff_0__w_20__widths_(2, 5, 10, 20)',
 'Close__fft_coefficient__attr_"real"__coeff_0',
 'Close__fft_coefficient__attr_"abs"__coeff_0',
```

a)



b)

Figure 5.42 – Forecasting the bitcoin exchange rate in December 2021 with an accuracy of 2.44% using the "MLP Regressor" model based on about 50 features synthesized by the TSFresh library (from the author's notebook): a) features synthesized using the TSFresh library; b) the result of a 10-day rate forecast

8. Determine forecast horizons and form training and validation datasets.

As a rule, test and validation data are selected at the end of the series if the prediction task is solved (see examples [11, 36, 59]).

9. Make forecasts and analyze the results.

5.4.3 Construction of time series models: ARIMA, Prophet

All models in Section 4 can be used to predict time series. Neural network recurrent models GRU and LSTM can also be used. But, more often, they use time series specific models ARIMA and Prophet (previous name: "Facebook Prophet").

The mathematical apparatus ARIMA (Autoregressive Integrated Moving Average) is described in detail in textbooks of one of the co-authors [60, 61]. To automate ARIMA in Python, one of two options is used:

1) ARIMA method of the package arima_model the "Time Series Analysis" library, which allows you to identify the model for a given series with a given order ARIMA(p,d,q), where p is the autoregressive order, d is the order of difference, q is the order of the moving average;

2) method auto_arima the pmdarima library, which itself selects the parameters of the SARIMAX model, where in addition to p, d, q there are also parameters of seasonal components (Fig. 5.43).

The main aspects of the Prophet model are described in the article [62] and in the documentation. Mathematically, the Prophet model for modeling and predicting the values of the series $y(t)$, depending on time t, is written as follows [54, 58, 62]:

- For the additive case:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t, \tag{5.2}$$

- For the multiplicative case:

$$y(t) = g(t)s(t)h(t)\epsilon_t, \tag{5.3}$$

where $g(t)$ is the trend of the series (logistic or piecewise linear approximation of data); $s(t)$ – seasonal component approximated by the Fourier series; $h(t)$ – a component that takes into account the impact of holidays or other anomalies that operate with a certain "window", that is, in the range of certain dates (steps); ϵ_t – "noise" error with zero mean.

For a more detailed example of effective setting of the parameters of the Prophet model on the example of forecasting the number of daily new cases of coronavirus in Ukraine in 2020 (Fig. 5.44 from the article [58]).

```
Performing stepwise search to minimize aic
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=6196.011, Time=2.35 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=6185.373, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=6187.379, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=6187.375, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=6183.686, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=6189.382, Time=0.04 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 2.564 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  345
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -3090.843
Date:                Fri, 07 Apr 2023   AIC                           6183.686
Time:                        12:05:59   BIC                           6187.527
Sample:                             0   HQIC                          6185.216
                                  - 345
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2       3.72e+06   2.24e+05     16.620      0.000    3.28e+06    4.16e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.76   Jarque-Bera (JB):                22.63
Prob(Q):                              0.38   Prob(JB):                         0.00
Heteroskedasticity (H):               0.82   Skew:                            -0.16
Prob(H) (two-sided):                  0.28   Kurtosis:                         4.21
===================================================================================
```

Figure 5.43 – Example of the result of the auto_arima work on identifying the SARIMAX model for predicting the bitcoin exchange rate (from the notebook)

Figure 5.44 – Example of forecasting the number of daily new cases of coronavirus in Ukraine (2020) using the Prophet (from the notebook)

See more examples of using the Prophet model and tuning its parameters with real data [37, 58, 63, 64].

The constructed model allows us to analyze the patterns inherent in the data. For example, it is possible to analyze environmental factors which increase *Alternaria* or *Cladosporium* spores [28, 29], find main factors that increase the risk of patient death of Coronavirus [31].

Fig. 5.45 shows an infographics of the operations and models mentioned in section 5.4 in notation *S(I)*.

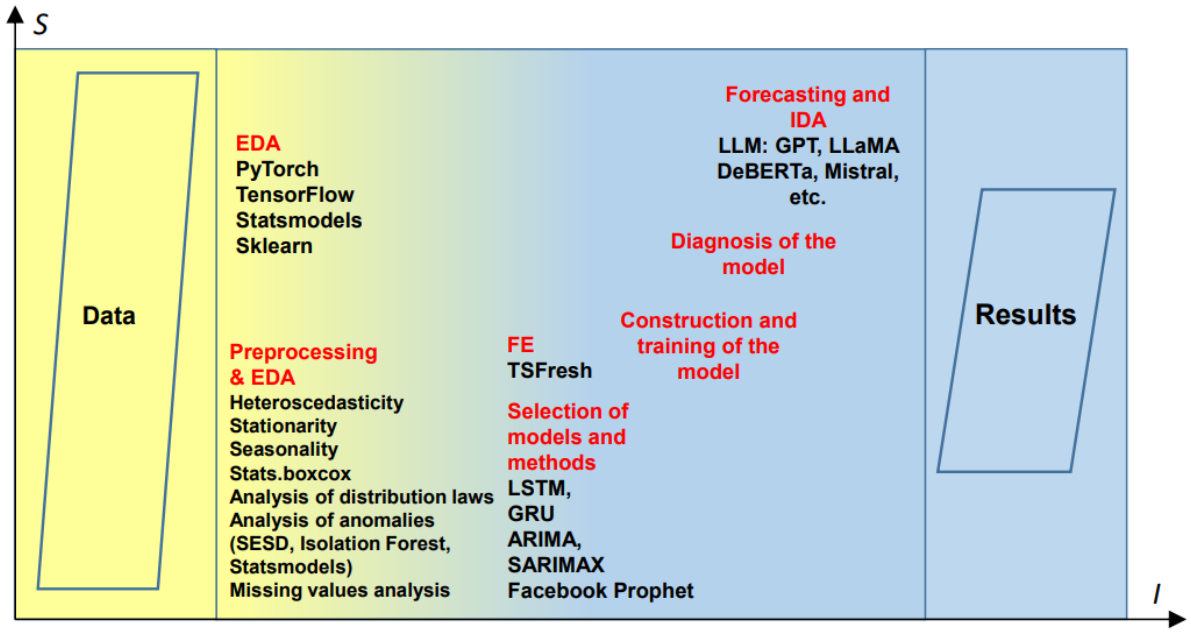


Figure 5.45 – Infographics of technologies for preprocessing and intelligent analysis of time series

Practical exercises

1) There is a 2x2 matrix A and there is a 2x2 kernel K. What will be the result of applying the convolution K to the matrix A using the Conv2D command in stride = (1, 1), padding='valid' mode? An example of the calculation is shown in Fig. 5.46.

Let's consider the following 2x2 matrix A and 2x2 kernel K:

Matrix A:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Kernel K:

$$K = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Position the kernel at the top-left corner of the matrix A:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Multiply corresponding elements and sum them:

$$(1 \cdot 1) + (2 \cdot 0) + (3 \cdot 0) + (4 \cdot -1) = 1 + 0 + 0 - 4 = -3$$

The result of applying the convolution operation with kernel K to matrix A using `Conv2D` with `stride = (1, 1)` and `padding='valid'` mode is a single value:

-3

Figure 5.46 – Application of convolution

2) There is a 2x2 matrix A and there is a 2x2 kernel K. What will be the result of applying MaxPooling2D (pool_size=(2, 2), strides=(2,2), padding="valid") to this matrix? An example of the calculation is shown in Fig. 5.47.

1. **Initial Window Position**: The 2x2 window covers the entire matrix A.

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

2. **Max Value Selection**: From this window, the maximum value is 4.

Figure 5.47 – Application of MaxPooling2D

3) There is a 2x2 matrix A and there is a 2x2 kernel K. What will be the result of applying AveragePooling2D(pool_size=(2, 2), strides=(2,2), padding="valid") to this matrix? An example of the calculation is shown in Fig. 5.48.

Let's consider a 2x2 matrix A:

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

Calculate the average:

$$\text{Average} = \frac{1+2+3+4}{4} = \frac{10}{4} = 2.5$$

Figure 5.48 – Application of AveragePooling2D

4) In the TF-IDF method, calculate the TF indicator for the word that occurs most often in a given sentence (without punctuation marks). An example of the calculation for the sentence “In the project, output the result in this way: sentence sentence sentence paragraph paragraph picture picture picture picture picture” is shown in Fig. 5.49.

$$\text{TF}(t) = \frac{\text{Number of times term } t \text{ appears in a sentence}}{\text{Total number of words in the sentence}}$$

1. **Count the occurrences of each word:**
   - "sentence": 3 times
   - "paragraph": 2 times
   - "picture": 5 times
2. **Count the total number of words in the sentence:**
   - Total words: $3 + 2 + 5 = 10$
3. **Calculate the TF for each word:**
   - For "sentence":

$$\text{TF}(\text{sentence}) = \frac{3}{10} = 0.3$$

   - For "paragraph":

$$\text{TF}(\text{paragraph}) = \frac{2}{10} = 0.2$$

   - For "picture":

$$\text{TF}(\text{picture}) = \frac{5}{10} = 0.5$$

Since the task specifies to calculate the TF indicator for the word that occurs most often, we focus on the word "picture," which appears 5 times.

Figure 5.49 – Calculation TF index for TF-IDF method

5) Calculate the TF-IDF indicator of the "TF-IDF" method for a given word in the given sentence (without punctuation marks) if the IDF of this word is a certain value for your corpus of words. An example of the calculation of the TF-IDF indicator for the word “picture” (IDF = 0.7) for the sentence “In the project, put the result in this way: sentence sentence sentence paragraph paragraph picture picture picture picture picture” is shown in Fig. 5.50.

To calculate the TF-IDF indicator for the word "picture" in the given sentence, we need to use th formula for TF-IDF:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

where:

- $\text{TF}(t, d)$ is the term frequency of term t in document d.
- $\text{IDF}(t)$ is the inverse document frequency of term t.

1. **Count the occurrences of each word:**
   - "sentence": 3 times
   - "paragraph": 2 times
   - "picture": 5 times
2. **Count the total number of words in the sentence:**
   - Total words: $3 + 2 + 5 = 10$
3. **Calculate the TF for "picture":**

$$\text{TF}(\text{picture}) = \frac{5}{10} = 0.5$$

Now, using the given $\text{IDF}(\text{picture}) = 0.7$:

4. **Calculate the TF-IDF for "picture":**

$$\text{TF-IDF}(\text{picture}) = \text{TF}(\text{picture}) \times \text{IDF}(\text{picture}) = 0.5 \times 0.7 = 0.35$$

Figure 5.50 – Calculation TF-IDF index

6) For a given time series in the form of a list of values at successive moments of time [10, 12, 11, 15, 14, 17], calculate the first (if you start counting from 1) rolling value, which will be the result of the following command: roll-

ing(window=3).mean() (or max, or min), which will not be equal to np.nan. An example of the calculation is shown in Fig. 5.51.

Let's go through the calculations:

- Time series: [10, 12, 11, 15, 14, 17]
- Rolling window size: 3

First, compute the rolling mean:

- For the first window $[10, 12, 11]$:

$$\text{Mean} = \frac{10 + 12 + 11}{3} = \frac{33}{3} = 11$$

Now, compute the rolling maximum for each window of size 3:

- For the first window $[10, 12, 11]$: Maximum = 12

Let's calculate the rolling minimum values:

- **For positions 1 to 3**: [10, 12, 11]
  - Minimum = 10

Figure 5.51 – Calculation rolling(window=3).mean() (or max, or min)

7) There is a sequence of time series values in the form of a list of numbers. Binaries it according to the principle "Will there be an increase in values?". Predict the next value of this new binary time series using the mean value method (with rounding to the nearest integer according to the rules accepted in mathematics). An example of the calculation is shown in Fig. 5.52.

Let's apply this to the series:

1. Compare each pair of consecutive values:
   - 10 vs 12: 12 > 10 ⟶ 1
   - 12 vs 11: 11 < 12 ⟶ 0
   - 11 vs 15: 15 > 11 ⟶ 1
   - 15 vs 14: 14 < 15 ⟶ 0
   - 14 vs 17: 17 > 14 ⟶ 1

Binary series based on increase prediction: $[1, 0, 1, 0, 1]$

Next, to predict the next value in this binary series using the mean value method:

2. Calculate the mean of the binary series:

$$\text{Mean} = \frac{1+0+1+0+1}{5} = \frac{3}{5} = 0.6$$

3. Round the mean value to the nearest integer:
   - Nearest integer to 0.6 is 1.

Figure 5.52 – Data binarization and simple time series forecasting

Possible topics for practical and laboratory tasks

***Topic No. 1*. Image Analysis and Classification** (or **"Intelligent Analysis of Graphical Data on the State of a Complex System Using Artificial Intelligence Technologies in Python").**

***Topic No. 2*. Natural Language Text Analysis and Classification Using Deep Learning and NLP Technologies**.

***Topic No 3*. Identification of the time series model and solution of the forecasting problem.**

***Topic No. 4*. Data analysis.**

The purpose of the work is to study information technologies and Python libraries for data analysis and master practical skills in applying machine learning models and artificial intelligence technologies on the example of one of the Kaggle datasets or from data uploaded via API.

Lesson plan:

1. Select a dataset.
2. Download data.
3. Conduct EDA, FE, and data preprocessing.
4. Choose information technology to solve the problem.
5. Apply information technology to convert input data into numerical data, depending on its type (image, video or text). For the rows, this stage can be skipped.
6. Train intelligent models using machine learning methods and choose the optimal one among them.
7. Apply intelligent models and analyze the results.
8. Provide a link to the created notebook, which contains all the results of the work, and draw conclusions.

Examples of notebooks that can be used:

- For Image Analysis and Classification:
  - MNIST model testing : typographic digits
  - MNIST model testing : user hand written digits
  - CNN over MNIST
  - Introduction to CNN Keras - 0.997 (top 6%)
  - Fashion MNIST Classification using CNNs
  - notebooks dataset MNIST models testing : handwritten digits
  - notebooks dataset MNIST models testing: typographic digits
- For Intelligent Analysis and Classification of Natural Text:
  - Data Science with DL & NLP: Advanced Techniques
  - NLP - EDA, Bag of Words, TF IDF, GloVe, BERT
  - NLP with Disaster Tweets - EDA and Cleaning data
  - NLP for UA : BERT Classification for Water Report
  - NLP for EN : BERT Classification for Water Report
- For time series analysis:
  - COVID in UA: Prophet with 4, Nd seasonality
  - COVID-19 UA: one region forecasting
  - AI-ML-DS Training. L1T : COVID in UA - Prophet
  - COVID-19 : Hospitalizations in Ukraine
  - COVID-19 in Ukraine: EDA & Forecasting
- other notebooks with Kaggle profile Prof. Vitalii Mokin: https://www.kaggle.com/vbmokin/code

Test questions

1) What types of information do you know about data analysis problems?
2) What are the typical datasets for training models in image classification?
3) What are tensors and how are they used in image processing?

4) Even examples of setting typical tasks for image processing?

5) What image upgrading operations can be performed using the OpenCV library?

6) What are convolutional neural networks, and how are they used for image analysis?

7) What complex neural network architectures do you know?

8) How are autoencoders used in unsupervising image processing tasks?

9) What is YOLO and for what purposes is this technology used?

10) What are generative adversarial networks (GANs) and variational autoencoders (VAEs), and how are they used to generate and detect deepfakes?

11) What intelligent information technologies are used to analyze and generate images and videos?

12) What does data processing contain in natural language text?

13) What are the basic concepts used in language models to solve NLP Problems?

14) What is "Bag of Words" and how is it used in text analysis?

15) How does TF-IDF work and what are its advantages compared to "Bag of Words"?

16) What is GloVe and what is the concept of embeddings in the context of language models?

17) What are the advantages of Word2Vec compared to GloVe?

18) What are transforming models and how are they used in solving NLP problems?

19) How does BERT work and how does it differ from other models for text analysis?

20) What approaches are used for feature engineering in the tasks of classification of natural language texts?

21) What are chatbots and how are they used in conjunction with large language models to improve communication with users?

22) What are the stages of exploratory analysis of time series data?

23) What techniques and libraries do you know for automatic detection of anomalies in time series?

24) What is stationarity in the context of time series analysis and why is it important?

25) How is seasonality determined in time series?

26) What is the purpose of the TSFresh library for time series forecasting?

27) What neural network models are used to predict time series?

28) What are the main parameters of the ARIMA model and how are they used for time series forecasting?

29) What are the main ways to identify an ARIMA model in Python?

30) What are the main components and types of the Prophet model do you know?

6.1 Basic concepts and concepts of the Internet of Things. Overview of LPWAN IoT technologies

6.1.1 Basic concepts and concepts of the Internet of Things

The "*Internet of Things*" (IoT) is a concept that describes a network of physical objects ("things") that have the ability to automatically collect and exchange data with each other, other networks, and the Internet.

IoT systems collect information from various types of sensors, using special IoT stations and routers. As a rule, information is stored in the cloud, where it is processed or transmitted to other networks or data warehouses (Fig. 6.1).

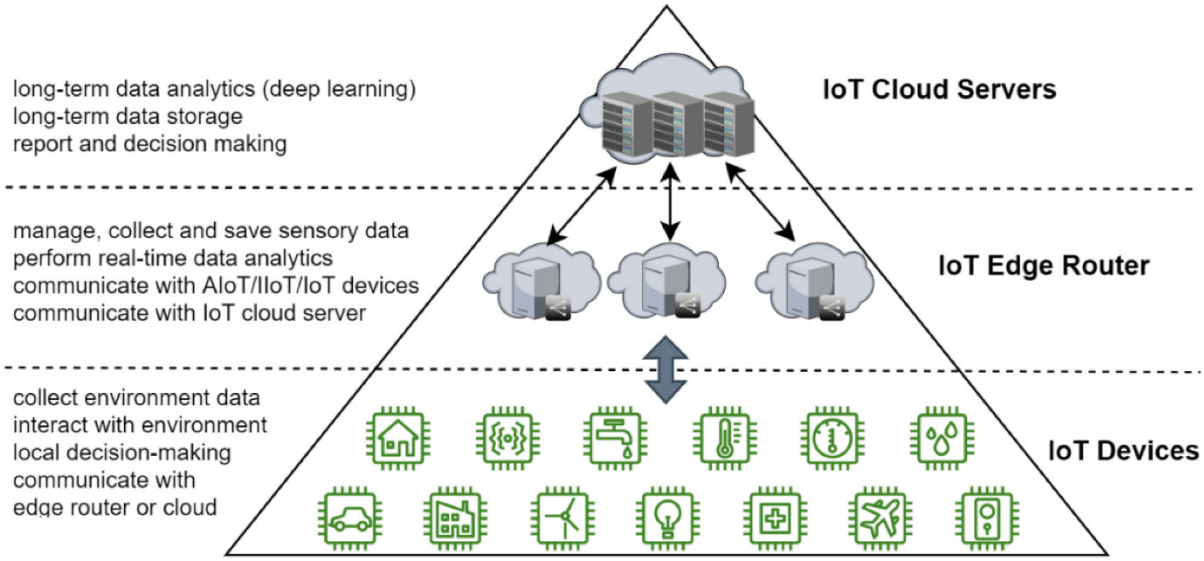


Figure 6.1 – IoT platform [65]

IIoT (Industrial Internet of Things) is the extension of the Internet of Things (IoT) technology into industrial sectors and applications (manufacturing, energy, transportation, healthcare, etc.). IoT involves connecting industrial equipment, machinery, sensors, and devices to the internet to collect and exchange data, monitor operations, and enable automation.

Artificial Intelligence of Things (AIoT) (AI + IoT) – artificial intelligence of things, which means the application of intelligent technologies to the processing of information collected using IoT systems to solve tasks of analysis, optimization and management processes and these things in real time. AIoT is now one of the main areas of IoT development, so a separate section 6.3 is devoted to it.

The concept of IoT is often associated with the term "*Digital Twin*" (DT), which is a virtual representation of a physical object, process or system in a digital environment. Unlike a model, the definition of which has a similar meaning, a digital twin contains a large amount of information about objects, which is collected in real time. But the digital twin can also be supplemented by mathematical and information models that are identified and updated based on the collected data. Generally, DT are created for complex production facilities, such as industrial production, grain elevators, bridges, etc.

The material in this section is based on the article [66].

Design and development of an information system based on IoT technology is the process of creating and consolidating software and hardware components that allow you to automatically collect, process and analyze data on physical indicators from various sources of information, primarily from sensors.

When building a system, it is necessary to take into account several important aspects, in particular:

- data transmission technology;
- data storage and processing technology;
- data security;
- standardization and interoperability.

It is important to choose:

- network technology;
- IoT platforms.

The main stages of designing and developing an information system based on IoT technology are as follows:

- requirements analysis and system specification;
- system architecture design;
- software selection and development;
- selection and development of the hardware component;
- integration and installation;
- testing, testing.

There are different types of wireless access media (Fig. 6.2).

Item	Data Rate (Mbps)	Range (m)	Frequency (MHz)	Tx (mA)	Standard
Wi-Fi 1-3 [35]	2–54	<200	2400–5000	312	IEEE802.11 a/b/g
Wi-Fi 4 [36]	300–600	<200	2400/5000	312	IEEE802.11n
Wi-Fi 5 [37]	6900	<200	5000	610	IEEE802.11ac
Wi-Fi 6 [37]	9600	<200	2400 and 5000	NA	IEEE802.11ax
ZigBee [38]	0.02–0.25	<100	400–2400	25.8	IEEE802.15.4
BLE [39]	1–2	<10	2400	5.8	Bluetooth SIG
WMBUS [40]	0.0048–0.032768–0.0100	<500	433–868	37	OMS
Wi-Fi Halow [41]	0.15–3.9	<1000	863–930	8.5	IEEE802.11ah
WMBUS [42]	0.0024–0.0048–0.0192	LPWAN	169	703	OMS
LoRa [43]	0.300	LPWAN	433–868–915	20–120	LoRa Alliance
SP-L2 SigFox [44]	0.0001–0.500	LPWAN	413–1055	32	SigFox's proprietary protocol
NB-IoT(5G)	0.10(DL)–0.02(UL)	LPWAN	700–800–900	167	3GPP
LTE-M	1	LPWAN	699–2200	202	3GPP

Figure 6.2 – Different types of wireless access media [65]

The main type of systems that are developed using the IoT concept are systems that are focused on collecting data from various IoT sensors, which can be located far from the network infrastructure, since systems in the city can be implemented on the basis of wired or WiFi communication. Such a system is called *a wireless network with low data transmission power over long distances* Low-power Wide-area Network (LPWAN) (see Fig. 6.2). Its advantages are long data transmission range, low power consumption, good signal quality, low costs for installation and operation of the network. LPWAN can use various modern communication protocols, information systems, and technologies. The most popular in the EU are LoRaWAN, Sigfox, NB-IoT.

6.1.2 LPWAN IoT technologies: LoRaWAN, Sigfox, NB-IoT

Let's consider the *technical aspects of* LoRaWAN, Sigfox, NB-IoT architecture.

LoRaWAN uses low-frequency LoRa technology for a large coverage area without large-scale infrastructure. As you know, LoRa (Long Range) is a patented technology for modulating a low-power data transmission network with a speed of 0.3-50 kb/s and a range of 1 to 15 km in the frequency range that requires licensing.

Sigfox is a low-power wireless network that uses the ISM standard. As you know, the ISM standard is a set of rules that specify the use of certain frequencies of the radio frequency spectrum for industrial, scientific, and medical (ISM) devices. These devices can use these frequencies without obtaining a license. Sigfox uses low-frequency "Ultra Narrow Band" (UNB) technology for great coverage and energy efficiency. The Sigfox protocol operates in "Half Duplex" mode, where the end device sends short messages to the base station with-

out acknowledging the transmission. This ensures energy-efficient operation, but limits the volume and frequency of transmission.

NB-IoT stands for "Narrow band Internet of Things" – this is a standard designed specifically for cellular network devices and services, allowing IoT systems to be formed using these networks. That is, LoRaWAN and Sigfox use their own station systems, while NB-IoT operates over cellular networks. At the same time, LoRaWAN requires frequencies to be licensed, while Sigfox and NB-IoT do not.

The specifics of LPWAN network technologies are shown in Table 6.1.

Table 6.1 The specifics of LPWAN network technologies

Indexes	Technologies		
	LoRaWAN	Sigfox	NB-IoT
Operating frequency	433, 868, 780, 915 MHz	865 – 924 MHz	700 – 900 MHz
Capacity	7.8 – 500 kHz	100 Hz	180 kHz
Maximum package size	up to 255 bytes	12 bytes	up to 160 bytes
Maximum data transfer speed	Up to 50 kbit/s («Class A»), up to 1 Mbit/s («Class C»)	up to 100 bps	up to 250 kbit/p/s
Range of base station/gateway (open space)	18 km	50 km (up to 100 km)	100 km
Range of base station/gateway (dense construction)	5 km	10 km	10-15 km
Network topology	mesh topology, star	star	star
Encryption	AES-128	AES-128	3GPP 128-256 bit
Transmission power	14 dBm, 27 dBm	14 dBm, 27 dBm	20 – 23 dBm
Energy consumption	from several μW to several mW	from several μW to several mW	from several mW to several tens of mW

6.2 Architecture of IoT systems. Types of its typical components. Optimization of the architecture of IoT systems

6.2.1 Architecture of IoT systems. Types of its typical components

Let's consider the architecture and components of networks built on the basis of LoRaWAN, Sigfox, NB-IoT technologies. All of them can be unified into a three-level architecture that includes devices/nodes, gateways/base stations, and a client-server part.

The basic architecture of a LoRaWAN network consists of three layers: nodes, gateways, and network servers (Fig. 6.3).

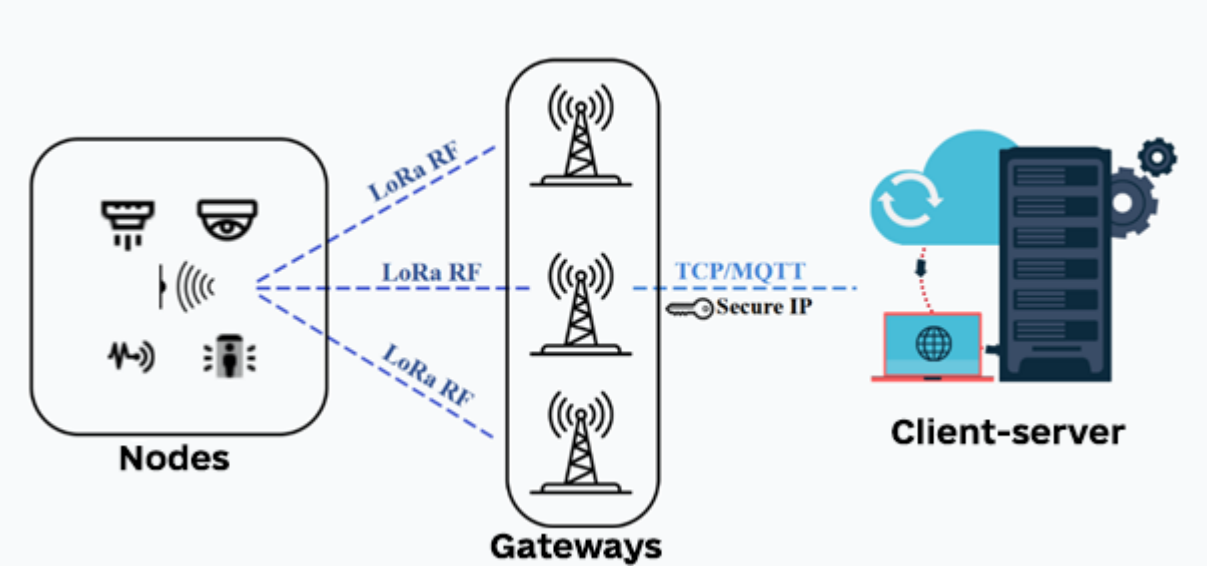


Figure 6.3 – LoRaWAN Network Architecture [66]

The Sigfox network consists of three layers: nodes, base stations, and accounting servers. Interaction in the Sigfox network takes place in only one direction (star type): from the node to the base station (Fig. 6.4).

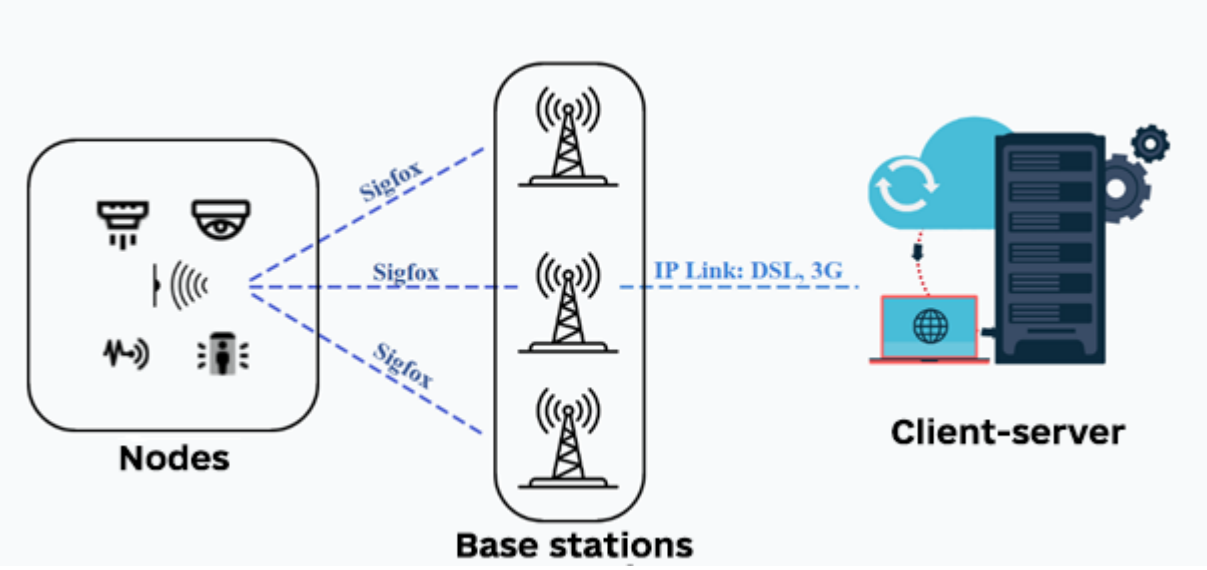


Figure 6.4 – Sigfox Network Architecture [66]

The basic architecture of the NB-IoT network also consists of three layers: devices, base stations (these often include another layer – the core of the network and the server part (Fig. 6.5).

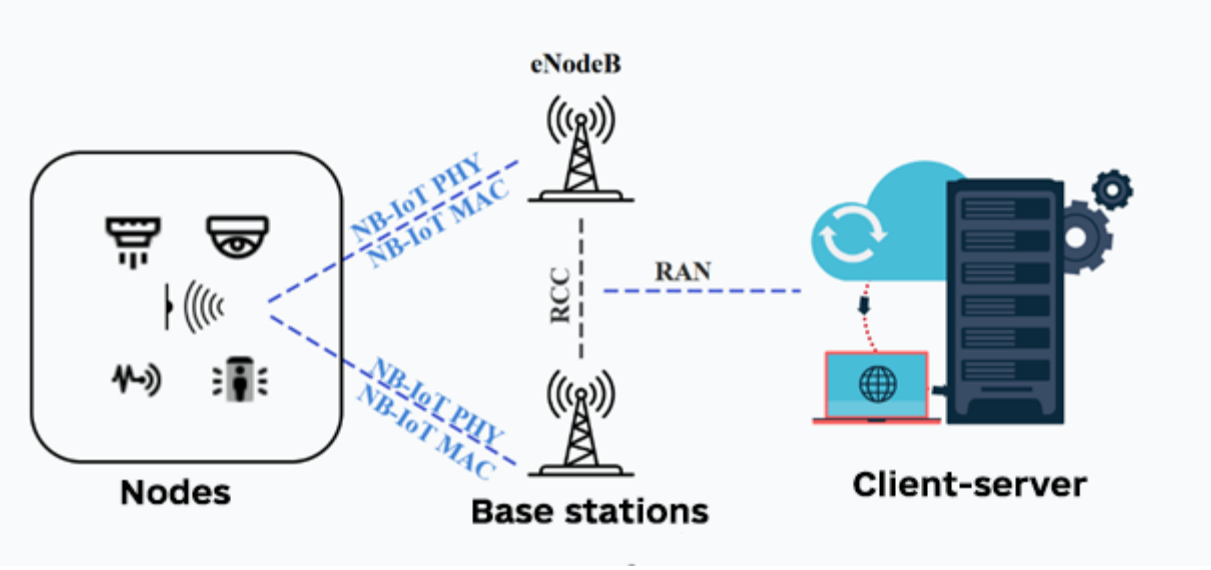


Figure 6.5 – NB-IoT Network Architecture [66]

6.2.2 Choosing an IoT platform for data collection, storage and analysis

IoT platforms provide tools for collecting, storing, processing, and analyzing data from connected devices, as well as for improving security and efficient management of the IoT network. The architecture of a typical IoT platform is shown in Figure 6.6:

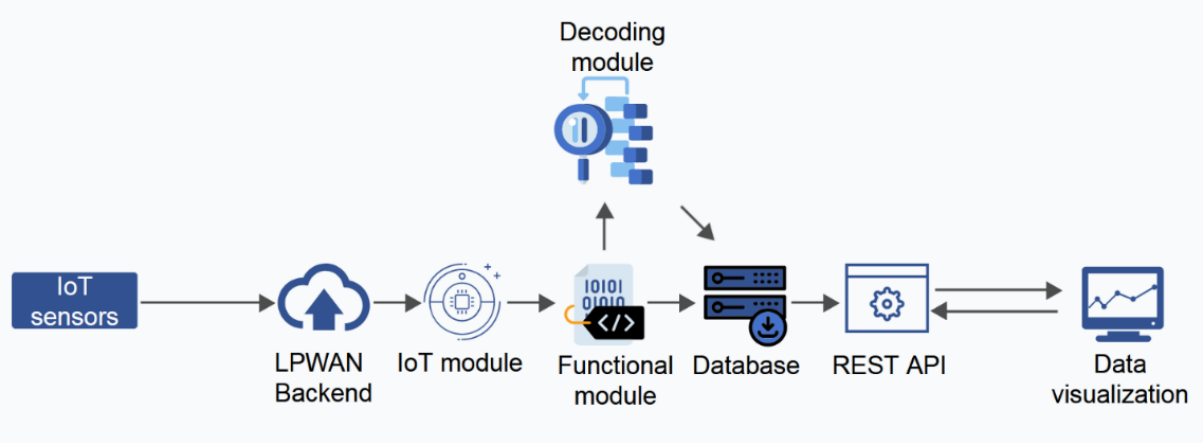


Figure 6.6 – IoT System Architecture [66]

The main components of the architecture are the following [66]:

1. Connected devices (IoT sensors) via data transmission standards (Sigfox, LoRaWAN, NB-IoT, etc.).

2. LPWAN Backend, which is responsible for maintaining and managing networks.

3. IoT module, which is designed to collect data from IoT devices.

4. Functional module, which is responsible for processing, analyzing and transmitting data.

5. The Decoding module is used to decrypt and interpret the data received from connected devices. IoT devices can use various data transfer formats, such as binary codes, JSON, or XML.

6. A database management system (Database) is used to store and manage a large amount of received data.

7. The REST API provides an application programming interface for external devices or systems to communicate and interact with the IoT platform.

8. Data visualization is responsible for displaying the data collected through the API. Visualization can include the creation of graphs, charts, maps, and other graphical representations of data for easy perception and analysis.

The most popular IoT platforms nowadays are:

- AWS IoT;
- Microsoft Azure IoT;
- Google Cloud IoT;
- ThingSpeak.

To create an information system based on the Internet of Things, first of all, it is necessary to determine the architecture of such a system. Architecture development consists of the following stages [66]:

- definition of functional requirements;
- selection of IoT devices, including sensors;
- selection of IoT communication technologies;
- choice of IoT platform;
- creation of a visualization system.

6.2.3 Optimization of the architecture of IoT systems

When designing a real IoT system, it is important to analyze the typical characteristics of each IoT technology and determine the optimal one in accordance with its requirements. Let's compare the characteristics of technologies according to the following criteria [66]:

X_1 – coverage range;
X_2 – frequency range;
X_3 – data transfer rate;
X_4 – cost of implementation;
X_5 – energy efficiency;
X_6 – reliability;
X_7 – speed of design;
X_8 – data confidentiality and security.

As an integral criterion, the classical criterion J_X is chosen, where w_i is the weight of the i-th criterion, which is determined by experts, depending on the conditions of the problem:

$$J_X = \sum_{i=1}^{N} w_i X_i. \quad (6.1)$$

The result of the multi-criteria analysis of expert assessments of the parameters of network technologies according to the above criteria is presented in Table 6.2.

Table 6.2. The result of the multi-criteria analysis of expert assessments of the parameters of network technologies

Criteria	X1	X2	X3	X4	X5	X6	X7	X8	*Jx*
Weights	0,2	0,1	0,1	0,15	0,15	0,1	0,15	0,05	
LoRa-WAN	1 – from 2 up to 15 km	1 – three ranges (433/868/915 MHz)	0,8 – 1,2 kbit/s	0,8	0,8	0,9	0,8	0,8	0,87
Sigfox	0,8 – from 2 up to 10 km	0,9 – two ranges	0,5 – 100 bit/s	1	1	1	1	0,8	0,89
NB-IoT	0,8 – 10 km	0,9 – two ranges	1 – 250 kbit/s	0.6	0.6	0,9	0,7	1	0,77

The task of choosing the optimal LPWAN technology can be as follows: minimize the cost of implementation while satisfying the following constraints [66]:

- The coverage range D_{min} must be not less than the specified value;
- The frequency range must correspond to the specified range from F_{min} to F_{max};
- The data transfer rate must be not less than the specified value R_{min};
- The cost of implementation C_{min} should be as low as possible;
- Energy efficiency E_{min} must not be less than the specified value;
- Reliability must be not less than the specified value N_{min};
- The design speed must be not less than the specified value P_{min};
- The confidentiality and security of data must be at least the specified value S_{min}.

The task can be solved as a linear programming problem, since all optimization criteria are linear. The optimal technology is the one for which criterion (6.1) will take the lowest value [66]:

$$J = \sum_{i=1}^{N} w_i X_{ij}, j = 1,2,3,\dots, \; AX \geq B, \tag{6.2}$$

where X_{ij} $(j = 1,2,3,..)$ is the value of the criteria for each of the IoT technologies, and X is their vector representation, A is the matrix of constraint coefficients on the criteria, B is the vector of constraints on the criteria.

To solve the taks, you can use various methods of linear programming (simplex method, inner point method, etc.). Let's use the PuLP library for Python to determine which of the technologies is the best choice when developing an architecture, taking into account energy efficiency, design speed and cost.

In Python, the solution of this problem was automated in the notebook "Selection of IoT technology" [67]. Input data is presented in the form of Table 6.2 (there may be a smaller number of criteria, i.e. columns). All values should be given as numbers so that they can be matched to each other. In a separate table, it is necessary to keep the weights of the criteria that are determined in accordance with the technical requirements for the system (Table 6.3). And in the other table, specify only the numerical values of the criteria (Table 6.4).

Table 6.3. Weights of the criteria

Architectures	X1	X2	X3	X4	X5	X6	X7	X8
Weights *w*	0,2	0,1	0,1	0,15	0,15	0,1	0,15	0,05

Table 6.4. Values of the criteria

Architectures	X1	X2	X3	Indicators	Technology	X6	LoRaWAN	Sigfox	*NB-IoT*
Operating Frequency	433, 868, 780, 915 MHz	865 – 924 MHz	700 – 900 MHz	Band width	7.8 – 500 kHz	100 Hz	180 kHz	Maximum Package Size	up to 255 bytes
12 bytes	up to 160 bytes	Maximum data transfer speed	up to 50 kbit/s ("Class A"), up to 1 Mbps ("Class C")	up to 100 bps	up to 250 kbps	Base Station/ Gateway Range (Open Area)	18 km	50 km (up to 100 km)	100 km
Base Station/ Gateway Range (Dense Build)	5 km	10 km	10-15 km	Network Topology	Mesh Topology, Star	star	star	Encryption	AES-128

The J_x *values obtained in the last column* indicate that Sigfox is the optimal technology for the Internet of Things-based physical parameters monitoring information system if a reliable, energy-efficient, relatively low-cost technology is required. If there is a need to increase the coverage range or increase the level

of privacy, then you need to change the parameters of Table 6.3 and re-solve the problem according to Table 6.4.

Other network technology parameters can be optimized in the same way.

6.2.4 The example of creating an IoT system

The article [66] provides an example of solving such a problem. The result of the information system for monitoring physical indicators based on the Internet of Things with the optimal architecture, created with the participation of one of the authors of this manual, is also described (Fig. 6.7).

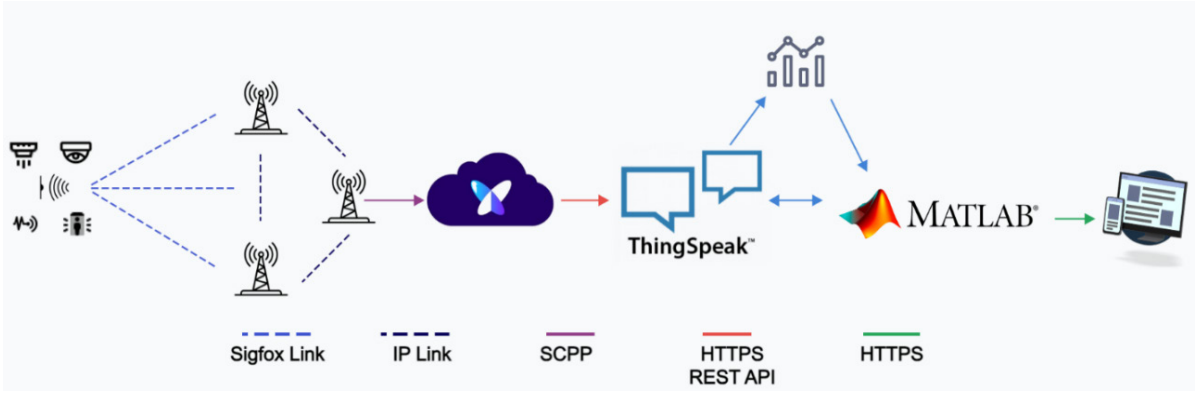


Figure 6.7 – Optimal architecture of the information system for monitoring physical indicators based on the Internet of Things, selected in the article [66]

As you can see in Figure 6.6, the best choice is to use the ThingSpeak IoT platform for data storage and analysis. The ThingSpeak server collects data from the Sigfox Backend using API queries and stores it in a database. The obtained data can be displayed using the web interface of ThingSpeak.

The article [66] describes an example of an IoT system implemented in practice with the architecture from Fig. 6.7. The device was implemented on the basis of the Arduino Mini microcontroller, SFM10R1 module using a temperature sensor SD18B20 and a Sigfox station (SMBS-T4), which is on the balance sheet of the Department of System Analysis and Information Technologies of the Faculty of Intelligent Information Technologies and Automation of VNTU. The block diagram of the hardware of the information system is shown in Figure 6.8:

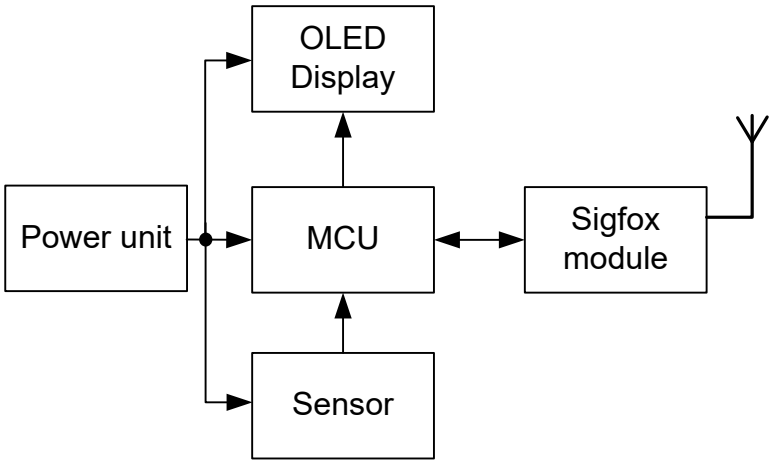


Figure 6.8 – Block diagram of the hardware part of the information system [66]

The microprocessor device generates commands to control the Sigfox SFM10R1 module via the RS-232 protocol. The temperature value over the I2C bus is read by the microcontroller and transmitted in the body of the AT command to send the data packet to the Sigfox network (AT$SF). When the transmitter receives this command, it generates a response about its successful execution.

Figure 6.9 shows an example of visualization of the data received from the created IoT system.

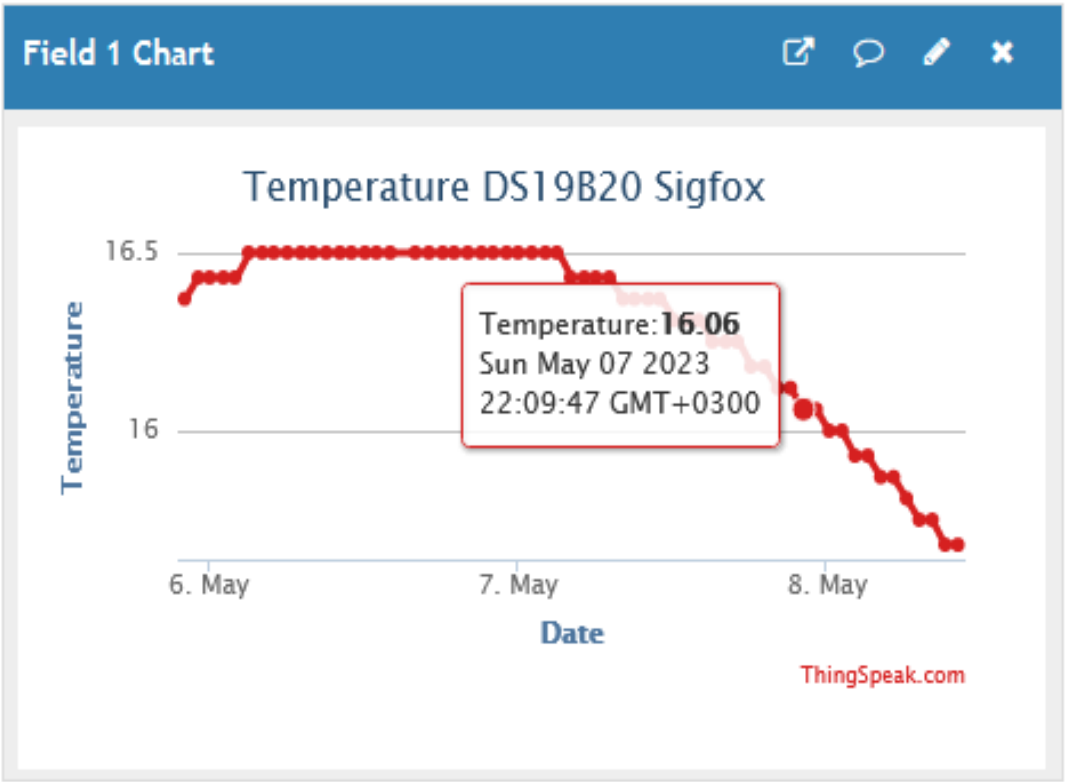


Figure 6.9 – Visualization of data obtained from the Sigfox temperature monitoring information system based on IoT technology, developed with the participation of the author of the manual [66]

6.3 Artificial Intelligence of Things (AIoT)

One of the most promising areas for the development of modern software and hardware technologies in the development of IIoT is the transition to AIoT, which not only collects and processes information, but also automatically makes decisions and implements them. Sometimes, IoT systems are also referred to as AIoT systems, which only process the information collected by IoT systems using AI subsystems and technologies. Such AIoT are of a recommendatory nature, for example, in a convenient form, they provide the process operator with a set of options for decisions that can be made and what consequences and benefits each of them will have. During 2022-2023, one of the co-authors of this manual (Vitalii Mokin) participated in the development of such a system as the scientific supervisor of the Vinnytsia National Technical University "Development of Information Technologies for Grain Elevator Optimization Using Neural Network Models and Reinforcement Learning Methods" together with INNOVINNPROM LLC, which implemented the project "Asset Performance Management System for grain processing industry SAKURA-APM PaaS SAKURA-IIoT based" (Fig. 6.10). This project has received funding from the European Union's Horizon 2020 research and innovation programme within the framework of the BOWI Project funded under grant agreement No 873155. The results of this project are described in more detail in YouTube video and in the article [68].

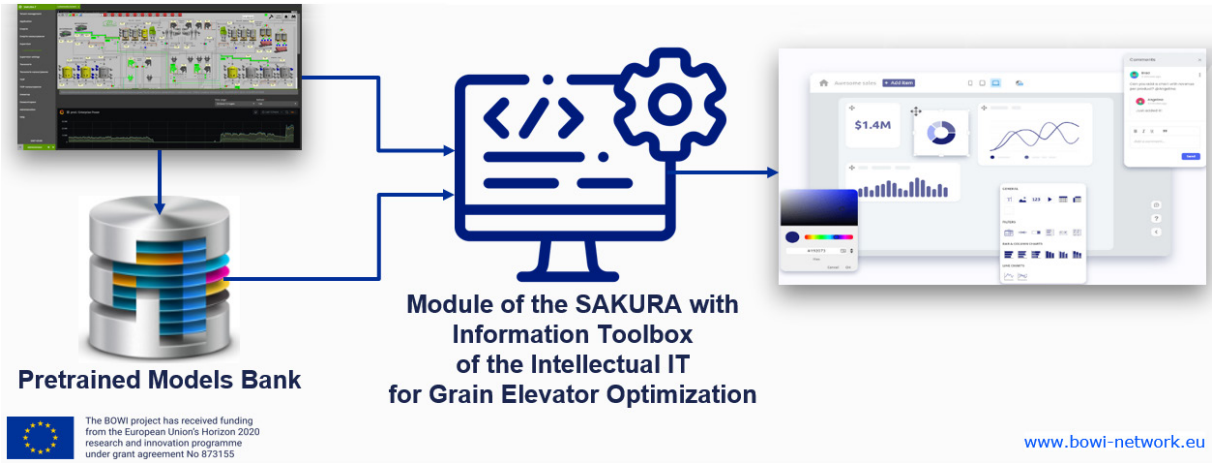


Figure 6.10 – AIoT in the project «Asset Performance Management System for grain processing industry SAKURA-APM PaaS SAKURA-IIoT based» (from YouTube video)

But more effective are AIoT subsystems, which also contain subsystems for automatic control of these production systems using another IoT subsystem (Fig. 6.11).

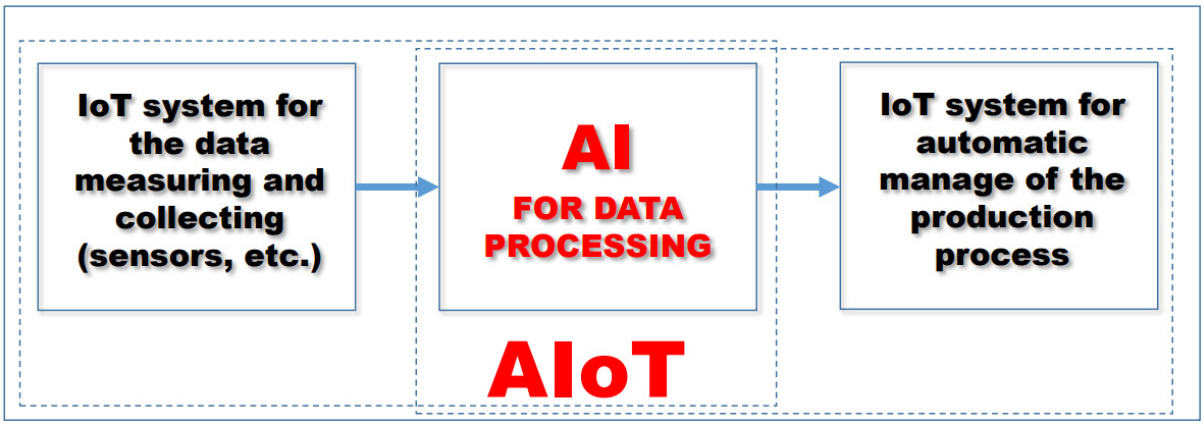


Figure 6.11 – Generalized architecture of the AIoT system

For example, Tesla and other vehicles with autopilot are classic AIoT with IoT and for collecting information, AI for processing it, and with IoT for applying the developed optimal solutions through the means of controlling this transport.

The article [65] provides a good and up-to-date overview of modern architectural, technological, software and hardware solutions used in AIoT. The basic idea is that the architecture from Fig. 6.11 should be able to work autonomously without copying data to the cloud (Fig. 6.12).

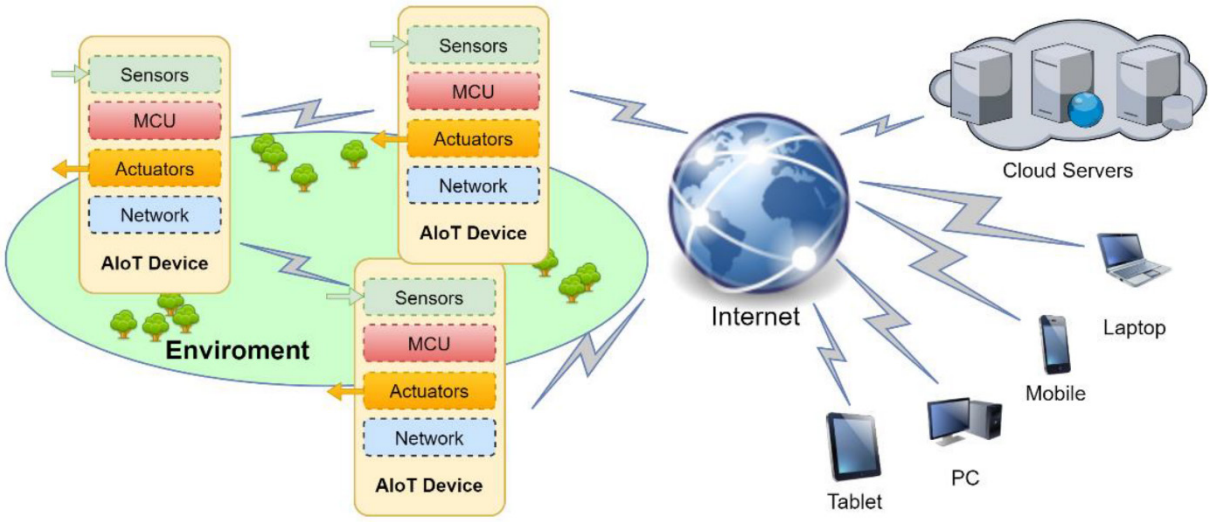


Figure 6.12 – Basic AIoT platform [65]

As can be seen in Fig. 6.12 The system consists of AIoT subsystems that independently collect information from sensors, process it, make decisions and transmit it directly to devices that execute these decisions. In addition, these subsystems exchange information with each other and with the Internet. Users have access to data via the Internet and it is possible to exchange data with the cloud. This architecture has the following main advantages:

- due to the autonomy of AIoT subsystems, the time for the implementation of optimal decisions is significantly reduced;
- due to data exchange with the Internet and the cloud, it is possible to store all the collected information and improve decision-making algorithms;
- due to the exchange of information between AIoT subsystems through IoT protocols without copying to the cloud, it is possible to speed up this process.

Of course, the advantages of architecture in Fig. 6.12 are also its disadvantages:

- errors or lack of information in the implementation of automatically made decisions, in the absence of human control, can lead to irreparable errors;
- the implementation of such AIoT subsystems requires very energy-efficient AI solutions, and cost-effectiveness, as a rule, means simplification and some opportunities;
- delays in the Internet network when transferring data to the cloud can lead to the fact that not all information will have time to be transmitted or transmitted with a significant delay, which can lead to the fact that external control may not be effective enough.

Fig. 6.13 shows an example of a hardware implementation of an AIoT subsystem for an intelligent irrigation system.

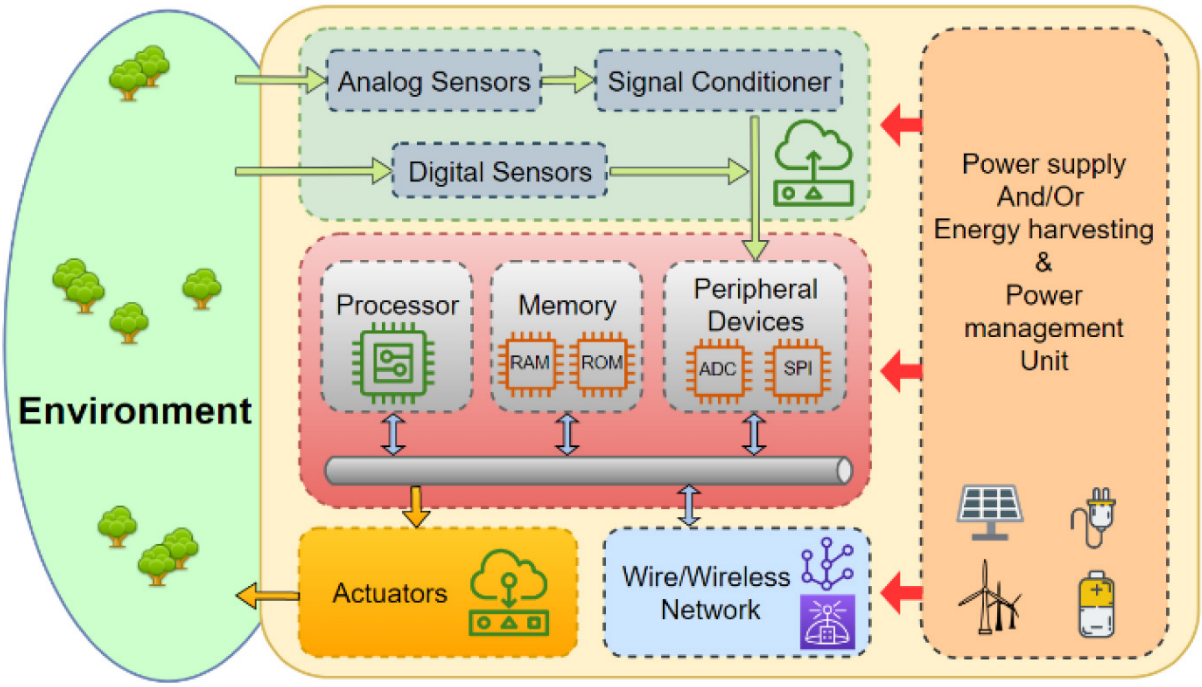


Figure 6.13 – Basic hardware architecture of AIoT/IIoT/IoT devices for a smart irrigation application [65]

Without the AIoT system, it would be impossible to study, for example, Mars. After all, the signal from the rover takes about 12.5 minutes to reach Earth. And during this time, he has to move somehow, perform some research,

maneuvers, go somewhere, adapt to the terrain. Therefore, its AIoT subsystem itself makes decisions and immediately implements them, and control from the control center from Earth can only make certain adjustments with a delay of 12.5 minutes. Under terrestrial conditions, the delay is usually much smaller, but for some processes, a delay of even a couple of seconds is critical.

It is for the development of such systems that it is important to create both accurate and energy-efficient models and technologies of machine learning and data analysis, as well as fast and effective algorithms for their adjustment, taking into account new data.

The most popular way to create energy-efficient machine learning models to solve data analysis problems for AI systems is to use the TinyML (Tiny Machine Learning) concept to transform intelligent models into more energy-efficient ones optimized specifically for AI subsystems. For example, such a transformation is provided by the TensorFlow Lite framework (Fig. 6.14).

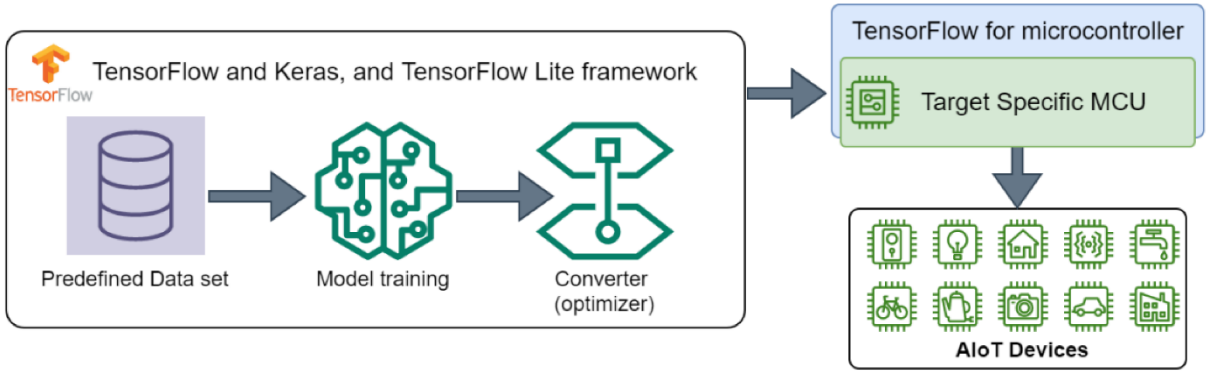


Figure 6.14 – TensorFlow framework for microcontrollers
(TinyML concept) [65]

Such a transformation can also be done using the Kaggle platform. First, a regular model is built and trained using the framework TensorFlow. And then it is transformed into a format optimized for IoT systems using the TinyMLGen library (tinymlgen). And it is still preserved. This is illustrated by the author's notebook "MNIST : TF Learning and TinyML Transformation" on the example of the task of recognizing Arabic numerals from the MNIST dataset (Fig. 6.15).

```
# Model transformation to TFLite model by TinyML concept
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

+ Code + Markdown

```
# Save the TFLite model
open("mnist_cnn_quantized.tflite", "wb").write(tflite_model)

!xxd -i mnist_cnn_quantized.tflite > mnist_cnn_quantized.cc
```

```
# Save the origin TF model using library tinymlgen
c_code = port(model, variable_name='mnist_cnn')

with open('mnist_cnn.h', 'w') as f:
    print(c_code, file=f)
```

Figure 6.15 – An example of transforming and saving an intelligent model built to recognize Arabic numerals from the MNIST dataset using the TensorFlow framework and the TinyMLGen library

Possible topics for practical and laboratory tasks

Topic No. 1. "Selection of the optimal LPWAN network technology for the implementation of an information and measurement system based on the Internet of Things".

The purpose of the lesson is a comparative analysis of LPWAN network technologies for the implementation of an information and measurement system based on the Internet of Things and mastering the possibilities for their optimal selection using the method of linear optimization and the notebook for its automation in Python in Kaggle.

Lesson plan:

5. According to the option provided by the instructor (see below), select the importance of criteria *w* in Table 5.3.

6. Type table 5.3 (MS Excel, Google.Table, etc.) and save it in CSV format. Download to Kaggle as a private dataset.

7. Copy the notebook "Selection of IoT technology" [67] to your Kaggle profile and pull the dataset from step 2 into it as a data source.

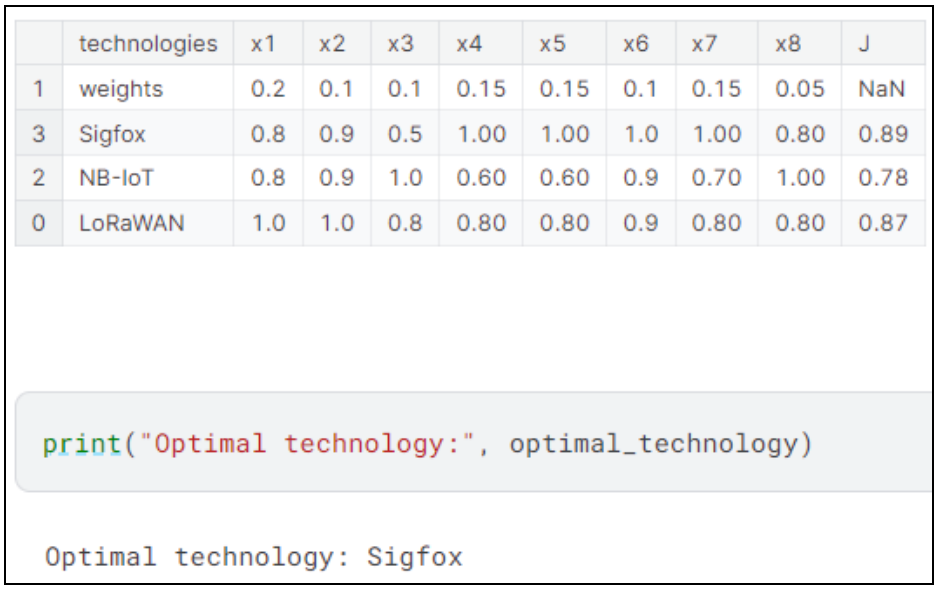
8. In cell No. 3 of the notebook with the selection of the table with weights, replace the path and name with the path and name to your table in the private dataset.

9. Run the notebook and analyze the result of its work.

10. Give a screenshot of the last cells with the results of the work (Fig. 6.16): the input table (p. 2) and the conclusion about which architecture from Table 6.4 is optimal.

11. If you wish, you can repeat the operations of pp. 1-5 to select optimal solutions at other stages of IoT system design (IoT platform, etc.), according to the example in the article [66].

12. The report must contain the full name, variant number and a screenshot from p. 6 in any form (file in docx or pdf). Send to the lecturer by mail.



	technologies	x1	x2	x3	x4	x5	x6	x7	x8	J
1	weights	0.2	0.1	0.1	0.15	0.15	0.1	0.15	0.05	NaN
3	Sigfox	0.8	0.9	0.5	1.00	1.00	1.0	1.00	0.80	0.89
2	NB-IoT	0.8	0.9	1.0	0.60	0.60	0.9	0.70	1.00	0.78
0	LoRaWAN	1.0	1.0	0.8	0.80	0.80	0.9	0.80	0.80	0.87

```
print("Optimal technology:", optimal_technology)
```

```
Optimal technology: Sigfox
```

Figure 6.16 – An example of the notebook "Selection of IoT technology" [67] with default data, which should be given in the report

Variants of tasks (hypothetical examples):

Option 1. Information and measuring system for monitoring the state of the reservoir.

Requirements:

- low-dynamic signal from sensors (for example, the content of heavy metals in the water of the reservoir);
- the distance from the sensors to the station is no more than 10 km;
- the territory outside the city, without buildings and, for the most part, flat (the station is installed on a hill near the reservoir);
- relatively low cost of system implementation;
- the energy efficiency of the system is quite high.

Option 2. Tracking the movement of freight transport.

Requirements:

- strongly dynamic signal from moving trucks;
- the distance from the sensors to the station is from 2 to 15 km;
- data transfer rate 1.2 kbit/s;
- relatively high reliability of the system and speed of its design are required.

Option 3. Remote monitoring and management of water supply in rural areas.

In rural areas, there are often problems with the reliability of the communication network due to remoteness and low population density. However, a reliable and stable communication network is necessary for effective monitoring and management of water supply systems.

Requirements:

- dynamic signal from sensors (water level in tanks, pressure in the system);
- the distance from the sensors to the station is up to 10 km;
- data transfer rate up to 250 kbit/s;
- the cost of implementation and energy efficiency are not key parameters;
- data privacy and security are an essential component of the system.

Option 4. Tracking the movement of goods in railway cars.

Requirements:

- low-variable signal (in fact, only a certain registration code of the cargo is transmitted);
- the signal should be triggered 15 km before the railway station;
- relatively low cost of system implementation;
- the energy efficiency of the system is quite high;
- high speed of its design.

Test questions

1) What is IoT?
2) What is AIoT?
3) What are the important aspects to consider when building an IoT system?
4) What stages of designing and developing an information system based on IoT technology do you know?
5) What is LPWAN?
6) What types of modern LPWANs do you know? Describe their main types.

7) What are the elements of the architecture of the main types of modern IoT LPWAN networks?

8) What are the elements of the IoT LPWAN architecture?

9) What modern IoT platforms do you know?

10) What are the stages of IoT system architecture design?

11) What criteria do you know for choosing a network technology when designing an IoT system?

12) What are the typical constraints for choosing a network technology when designing an IoT system?

13) Which method and Python of the optimization library can be used to select IoT network technology?

14) Which criteria are more important for each type of IoT technology? Which ones do they maximize and which ones do they minimize, compared to others?

15) What are the advantages and disadvantages of IoT system architecture with autonomous AIoT subsystems?

16) What concept can be used to transform an intelligent model from the TensorFlow Lite framework into a more cost-effective, IoT-optimize done?

REFERENCES

1. Data Science: Machine Learning and Data Mining: electronic textbook for combined (local and network) use [Electronic resource] / V. B. Mokin, M. V. Dratovanyi. – Vinnytsia: VNTU, 2024. – 258 p.
2. V. B. Mokin, A. V. Losenko, and M. V. Dratovanyi, “Intellectual Technology of Analysis and Price Forecasting of Used Cars”, Visnyk VPI, no. 6, pp. 62–72, Dec. 2019. https://doi.org/10.31649/1997-9266-2019-147-6-62-72.
3. Dratovanyi M. and Mokin V., “Intelligent Method with the Reinforcement of the Synthesis of Optimal Pipeline of the Data Pre-Processing Operations in the Machine Learning Problems”, *Scientific Works of Vinnytsia National Technical University*, no. 4, Jun. 2023. https://doi.org/10.31649/2307-5392-2022-4-15-24
4. Vitalii Mokin. Kaggle Notebook "50 Tips: Data Science (tabular data) - beginner v2" : website, 2024. URL: https://www.kaggle.com/code/vbmokin/50-tips-data-science-tabular-data-beginner-v2
5. Vitalii Mokin, Kaggle Notebook "50 Advanced Tips: Data Science (tabular data) v2": website, 2024. URL: https://www.kaggle.com/code/vbmokin/50-advanced-tips-data-science-tabular-data-v2
6. Bisikalo, O. V., Sevastyanov, V. M., and Bohach, I. V. Laboratory Practicum on the Discipline "Computer Linguistics" for Students of the Specialty 126 "Information Systems and Technologies": Electronic Laboratory Practicum of Combined (Local and Network) use. Vinnytsia: VNTU, 2022. 102 p.
7. Oleshchenko, L. M. Machine Learning: Computer Practicum on the Discipline "Machine Learning". Guide for students specialty 121 "Software Engineering" (educational program "Software Engineering of Multimedia and Information Retrieval Systems"). Electronic text data. Kyiv : KPI them. Igor Sikorsky, 2022. 92 p.
8. Machine Learning: A Textbook Intended for Students Studying at the First (Bachelor's) Level of Higher Education in the Specialties of the Field of Knowledge 12 "Information Technologies" / Basiuk T. M., Lytvyn V. V., Zakharia L. M., Kunanets N. E., Lviv: Publishing House "Novyi Svit - 2000", 2019. 335 p.
9. David Mark Albert. Kaggle Notebook «Santander-eda-to-model-comparison»: website, 2019. URL: https://www.kaggle.com/davidmarkalbert/santander-eda-to-model-comparison
10. Gabriel Preda. Kaggle Notebook "Santander EDA and Prediction": website, 2019. URL: https://www.kaggle.com/gpreda/santander-eda-and-prediction
11. Vitalii Mokin, Kaggle Notebook «Crypto - BTC : Advanced Analysis & Forecasting» : website, 2022. URL:

https://www.kaggle.com/code/vbmokin/crypto-btc-advanced-analysis-forecasting

12. Victoria V. Rodinkova, Serhii D. Yuriev, Mariia V. Kryvopustova, Vitalii B. Mokin, Yevhenii M. Kryzhanovskyi. Molecular Profile Sensitization to House Dust Mites as an Important Aspect for Predicting the Efficiency of Allergen Immunotherapy. *Frontiers in Immunology*, Mar 2022. - https://doi.org/10.3389/fimmu.2022.848616

13. V. Rodinkova, O. Kaminska, S. Yuriev, O. Sharikadze, V. Mokin, L. DuBuske, Bayesian Network Analysis Indicates a High Probability of Simultaneous Sensitization to Major Grass Allergens, Annals of Allergy, *Asthma & Immunology*, Volume 129, Issue 5, Supplement, 2022, Page S23, ISSN 1081-1206, https://doi.org/10.1016/j.anai.2022.08.569

14. Yuriev, S., Rodinkova, V., Mokin, V., Varchuk, I., et al. Molecular sensitization pattern to house dust mites is formed from the first years of life and includes group 1, 2, Der p 23, Der p 5, Der p 7 and Der p 21 allergens. *Clin Mol Allergy*, 21, 1 (2023). https://doi.org/10.1186/s12948-022-00182-z.

15. Rodinkova V., Yuriev S., Mokin V., Sharikadze O., Kryzhanovskyi Y., Kremenska L., Kaminska O., Kurchenko A. Sensitization patterns to Poaceae pollen indicates a hierarchy in allergens and a lead of tropical grasses. *Clinical and Translational Allergy*. 2023. № 13(8). e12287. https://doi.org/10.1002/clt2.12287

16. Rodinkova V., Yuriev S., Mokin V., etc. Bayesian analysis suggests independent development of sensitization to different fungal allergens, 2024, *World Allergy Organization Journal*, 17, 5,100908, https://doi.org/10.1016/j.waojou.2024.100908.

17. Vepa, A., Saleem, A., Rakhshan, K., Daneshkhah, A., Sedighi, T., Shohaimi, S., ... & Chakrabarti, P. (2021). Using machine learning algorithms to develop a clinical decision-making tool for COVID-19 in patients. *International journal of environmental research and public health*, 18(12), 6228. https://doi.org/10.3390/ijerph18126228.

18. Vitalii Mokin, Kaggle Notebook "Heart Disease - Automatic AdvEDA & FE & 20 models" : website, 2021. URL: https://www.kaggle.com/code/vbmokin/heart-disease-automatic-adveda-fe-20-models/notebook.

19. Vitalii Mokin, Kaggle Notebook «Tutorial : Classification models» : website, 2024. URL: https://www.kaggle.com/code/vbmokin/tutorial-classification-models

20. Vitalii Mokin, Kaggle Notebook "Tutorial : Ensembles of classification models": website, 2024. URL: https://www.kaggle.com/code/vbmokin/tutorial-ensembles-of-classification-models

21. Vitaliy B. Mokin. Development of the Geoinformation System of the State Ecological Monitoring // NATO Advanced Research Workshop on “Fuzziness and Uncertainty in GIS for Environmental Security and Protection". —

NATO Security through Science Series. *Geographic Uncertainty in Environmental Security*. Springer Netherlands, 2007. — P. 153-165. https://doi.org/10.1007/978-1-4020-6438-8_9.

22. Dniester River Basin : Environmental Atlas / O. Lysyk, V. Mokin, V. Bujac, Gh. Sirodoev etc. // [Zoï Environment network (Switzerland) ; UNEP/GRID-Arendal (Norway) ; The Republic of Moldova, Agency "Apele Moldovei" ; Ukraine, State Agency of Water Resources of Ukraine]. – 2012. – 45 p. – ISBN 978-2-940490-12-7. – https://doi.org/10.13140/RG.2.1.1835.0569. – was supported by governments of Finland, Sweden, Norway and was prepared on behalf of organizations participating in the project “Transboundary cooperation and sustainable management in the Dniester River basin: Phase III – Implementation of the Action Programme»implemented in the framework of Environment and Security Initiative – ENVSEC (ENVSEC comprises the Organization for Security and Cooperation in Europe (OSCE), Economic Commission for Europe (UNECE), UNEP, UNDP, NATO, Regional Environmental Centre for Central and Eastern Europe).

23. Mokin V. B. Method For Determining And Optimization Of Observability Of Multivariable Spatially Distributed Systems Using Geoinformation Parameter Space / V. B. Mokin, I. V. Varchuk // *Scientific Bulletin of National Mining University*. — 2015. — Issue 5. — Pages 105-111.

24. Control and minimization of allergenic plants impact on bronchial asthma morbidity, based on spatial-temporal data model / Tatyana Y. Vuzh ; Vitaliy B. Mokin ; Waldemar Wójcik; Baglan Imanbek // *Proc. SPIE 9816, Optical Fibers and Their Applications*, 2015, Volume 98161M (December 18, 2015); doi:10.1117/12.2229083.

25. Optimization of Hydrographic and Water-management Regionalization of Ukraine according to World Approaches and Principles of the EU Water Framework Directive / V. V. Grebin', Vitaliy B. Mokin, Ye. M. Kryzhanivskiy, S. A. Afanasyev. – *Hydrobiological Journal*, 2016, Volume 52, Issue 5. – pages 81-92.–DOI:10.1615/HydrobJ.v52.i5.90.

26. Information measuring systems with mobile devices for identification of air pollution parameters caused by transport / Vitalii B. Mokin; Georgii V. Goriachev; Dmytro Y. Dziuniak; Kostiantyn O. Bondalietov; Serhii O. Zhukov; Mariusz Duk; Saltanat Sailarbek // Proc. SPIE 10031, *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments*, 2016, 1003128 (September 28, 2016), 8 pages; doi:10.1117/12.2249202.

27. Vitalii B. Mokin. The Decision Support System for the Classification of Allergenic Pollen Types Based on Fuzzy Expert Data of Pollen Features on the Microscope Images / Vitalii B. Mokin, Oleksii M. Kozachko, Victoria V. Rodinkova, Olena O. Palamarchuk, Tetyana Y. Vuzh // Conference Proceedings, IEEE 2017, first Ukraine conference on electrical and computer engineering (UKRCON), May 29 – June 2, 2017 Kyiv. – P. 850-855. – doi 10.1109/UKRCON.2017.8100368.

28. Viktoriya Rodinkova. Environmental Factors Which Increase Alternaria spores in Central Ukraine / V. Rodinkova, V. Mokin, O. Bilous, Lawrence Dubuske, M. Dratovanyj // *Journal of Allergy and Clinical Immunology*. – Elsevier Science Ltd., 2018. – V. 141. – # 2. – p. AB30. - DOI: 10.1016/j.jaci.2017.12.096.

29. Rodinkova V. Main weather factors, which impact Cladosporium spore concentration in Ukraine / V. Rodinkova, V. Mokin, L. Kremenska, M. Dratovanyj // ICA 2018, 11th International Congress on Aerobiology 3-7 September 2018, Parma, Italy. – P. 142.

30. Mokin V. B., Horash M. A., Kryzhanovskyi Y. M., Vuzh T. E., "Information Intelligent Technology of the Automatic Georeferencing of the Ecological Text Natural-Language Information", *Scientific Works of Vinnytsia National Technical University*, no. 4, Jun. 2020. https://doi.org/10.31649/2307-5392-2020-4-32-41

31. V. B. Mokin, O. V. Kovalchuk and N. O. Muzyka, "Intelligent Technology for Predicting the Risk of Patient's Death from Coronavirus Based on PRINCIPLE-Methodology for Selecting Indicators Collected from Medical Devices," 2022 IEEE 41st International Conference on Electronics and Nanotechnology (ELNANO), 2022, pp. 451-455, https://ieeexplore.ieee.org/document/9927026.

32. Bondalietov, K., Mokin, V. (2023). Notation System for Comparing and Synthesis of Intelligent Key Phrase Extraction Methods for Ontological Models in Information Systems. In: Dovgyi, S., Trofymchuk, O., Ustimenko, V., Globa, L. (eds) Information and Communication Technologies and Sustainable Development. ICT&SD 2022. *Lecture Notes in Networks and Systems*, vol 809. Springer, Cham. https://doi.org/10.1007/978-3-031-46880-3_11

33. Tummon, F., Bruffaerts, N., Celenk, S., Mokin, V. et al. Towards standardisation of automatic pollen and fungal spore monitoring: best practises and guidelines, *Aerobiologia*, Volume 40, Issue 1, Pages 39-55, March 2024, https://doi.org/10.1007/s10453-022-09755-6.

34. O. V. Komenchuk, V. B. Mokin, Y. M. Kryzhanovsky, and V. O. Budiak, "Intelligent Technology of Buildings Plan Construction, Based on Aerial Photography of their Roofs", *Visnyk VPI*, no. 1, pp. 101–109, Feb. 2024. https://doi.org/10.31649/1997-9266-2024-172-1-101-109.

35. V. B. Mokin, K. O. Bondalietov, Y. M. Kryzhanovskyi, and V. O. Karavaiev1, "Method of Augmentation of Texts About the State of Water Bodies on the Base of Intellectual Referencing to Multi-Related Geoinformation Systems of Named Entities", *Visnyk VPI*, no. 3, pp. 55–65, Jun. 2023. https://doi.org/10.31649/1997-9266-2023-168-3-55-65

36. V. B. Mokin, S. O. Zhukov, L. M. Kupershtein, and Slobodianiuk O. V., "Information Technology for the Cryptocurrency Rate Forecasting on the Basics of Complex Feature Engineering", *Visnyk VPI*, no. 2, pp. 81–93, Apr. 2022. https://doi.org/10.31649/1997-9266-2022-161-2-81-93

37. V. B. Mokin, O. V. Slobodianiuk, O. M. Davydiuk, and D. O. Shmundiak, “Information Technology for Finding Possible Sources of Increased River Pollution Using the Prophet Model”, *Visnyk VPI*, no. 4, pp. 15–24, Sep. 2020. https://doi.org/10.31649/1997-9266-2020-151-4-15-24

38. V. B. Mokin and B. S. Biletskyi, “Intellectual Technology of Making Quality Posters”, *Visnyk VPI*, no. 6, pp. 73–82, Dec. 2019. https://doi.org/10.31649/1997-9266-2019-147-6-73-82

39. Mokin, V. B., Groozman, D. M., Dovgopolyuk, S. O., and Lototsky, A. O. System Analysis of the Size of a Fragment of Images of Aerial Photography of Agricultural Lands to Search for Anomalies in them by Machine Learning Methods. *Visnyk VPI*. 2019. № 3. P. 75-85. https://doi.org/10.31649/1997-9266-2019-144-3-75-85

40. Kaggle Notebook «Convolutional Neural Network (CNN) Tutorial» : website, 2020. URL: https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial/notebook

41. Rafet Can Kandar. Kaggle Notebook «Convolutional Neural Network (CNN) Tutorial» : website, 2021. URL: https://www.kaggle.com/code/rafetcan/convolutional-neural-network-cnn-tutorial/notebook

42. Arnab Mukherjee. YOLO: Algorithm for Object Detection Explained : website, 2023. URL: https://www.linkedin.com/pulse/yolo-algorithm-object-detection-explained-arnab-mukherjee/

43. Gaudenz Boesch. A Guide to YOLO v8 in 2024: website, 2023. URL: https://viso.ai/deep-learning/yolov8-guide/

44. Sovit Rath. YOLO v8 : Comprehensive Guide to State Of The Art Object Detection: website, 2023. URL: https://learnopencv.com/ultralytics-yolov8/

45. Ya. O. Isaienkov, O. B. Mokin. "Analysis of Generative Deep Learning Models and Features of Their Implementation on the Example of WGAN" Isaenkov Y. O., Mokin O. B., *Visnyk VPI.* 2022. Issue. 1, pp. 82–94. https://doi.org/10.31649/1997-9266-2022-160-1-82-94

46. L. R. Kulyk and O. B. Mokin, “Methods for Ensuring Consistent Generation in Diffusion Models”, *Visnyk VPI*, no. 4, pp. 75–85, Aug. 2024. https://doi.org/10.31649/1997-9266-2024-175-4-75-85.

47. Talakh, M. V., and Dvořák, V. V. Intellectual data analysis. Part 1. Chernivtsi: Tekhnodruk, 2022. 367 p.

48. Mayank Tripathi. How to process textual data using TF-IDF in Python: website, 2018. URL: https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/

49. Vitalii Mokin, Kaggle Notebook «NLP - EDA, Bag of Words, TF IDF, GloVe, BERT» : website, 2021. URL: https://www.kaggle.com/code/vbmokin/nlp-eda-bag-of-words-tf-idf-glove-bert

50. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations invector space. *arXivpreprint arXiv:1301.3781.* https://doi.org/10.48550/arXiv.1301.3781

51. Vitalii Mokin, Kaggle Notebook «NLP for WR: Summarizing using BERT, GPT2, XLNET» : website, 2022. URL: https://www.kaggle.com/code/vbmokin/nlp-for-wr-summarizing-using-bert-gpt2-xlnet/notebook

52. Glossary of LLM Terms. URL: https://vectara.com/glossary-of-llm-terms/

53. V. B. Mokin, B. Y. Varer, and S. M. Levitskyi, “Intelligent Technology for Detecting Text-Based Deepfakes Using Large Language Models”, *Visnyk VPI*, no. 1, pp. 110–120, Feb. 2024. https://doi.org/10.31649/1997-9266-2024-172-1-110-120

54. Prophet documentation, the chapter "Diagnostics". URL: https://facebook.github.io/prophet/docs/diagnostics.html

55. Shmundiak, D. O., Izhakovska, N. S., Lytvynenko, D. O., and Sudets, A. O. Analysis of the possibilities of Python libraries for detecting anomalous data in the problem of forecasting the state of atmospheric air: materials of the LII Scientific and Technical Conference of the Faculty of Intelligent Information Technologies and Automation of Vinnytsia National Technical University. Electron. text. data. 2023. Access Mode : https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/18957/15722

56. Vitalii Mokin, Kaggle Notebook "Crypto - BTC : Advanced EDA" : website, 2022. URL: https://www.kaggle.com/code/vbmokin/crypto-btc-advanced-eda/notebook

57. Wikipedia «Cryptocurrency bubble». URL: https://en.wikipedia.org/wiki/Cryptocurrency_bubble

58. V. B. Mokin, A. V. Losenko, A. R. Yascholt. "Information Technology Analysis and Predicting a Multiwave Number of New COVID-19 Disease Based on Prophet Model", *Visnyk VPI*. 2020. Issue. 6, pp. 65–75. https://doi.org/10.31649/1997-9266-2020-153-6-65-75

59. Vitalii Mokin, Kaggle Notebook «Crypto - BTC : Analysis & Forecasting» : website, 2023. URL: https://www.kaggle.com/code/vbmokin/crypto-btc-analysis-forecasting

60. Mokin, B. I., Mokin, O. B., and Mokin, V. B. Metodologiya ta organizatsiia naukovyi doslidzhennia. Guide. 3rd, changes and additional. [Electronic resource]. Vinnytsia: VNTU, 2023. – 230 p.

61. Functional analysis in information technologies: textbook / B. I. Mokin, V. B. Mokin, O. B. Mokin, Vinnytsia: VNTU, 2024, 130 p.

62. Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, *72*(1), 37-45. http://dx.doi.org/10.7287/peerj.preprints.3190v2

63. D. O. Shmundiak and V. B. Mokin, “Method of Harmonics Parameters Identification and Anomalies of a Periodic Time Series Based on Adaptive Decomposition”, *Visnyk VPI*, no. 6, pp. 46–56, Dec. 2023. https://doi.org/10.31649/1997-9266-2023-171-6-46-56.

64. V. B. Mokin, A. V. Losenko, A. R. Yascholt. "Informational Technology of Analysis and Forecasting of Number of New Cases of Coronavirus SARS-Cov-2 in Ukraine Based on the Prophet Model". *Bulletin of Vinnytsia Polytechnic Institute*. 2020. № 5. Pp. 71–83. https://doi.org/10.31649/1997-9266-2020-152-5-71-83.

65. Hou, K.M.; Diao, X.; Shi, H.; Ding, H.; Zhou, H.; de Vaulx, C. Trends and Challenges in AIoT/IIoT/IoT Implementation. Sensors 2023, 23, 5074, https://doi.org/10.3390/s23115074.

66. Goncharenko, D. V., Mokin, V. B., and Protsenko, D. P., Building an Information System for Monitoring Physical Indicators Based on Internet of Things Technology, *Information Technologies and Computer Engineering*, Issue. 57, iss. 2, pp. 99–108, Sep 2023. http://dx.doi.org/10.31649/1999-9941-2023-57-2-99-108.

67. Honcharenko D. Selection of IoT Technology [Electronic resource] / Dmytro Honcharenko – Mode of access to the resource: https://www.kaggle.com/code/honcharenkodmytro/selection-of-iot-technology.

68. Development of Intelligent Technologies for Energy-Saving Optimization of Grain Elevator Operation Using Neural Network Models and Reinforcement Learning Methods // Scientific progress: innovations, achievements and prospects. Proceedings of the 5th International scientific and practical conference. MDPC Publishing. Munich, Germany. 2023. Pp. 138-144.

69. Kononova, K. Y. Machine Learning: Methods and Models: Textbook for Bachelors, Masters and Doctors of Philosophy Specialty 051 "Economics". Kharkiv: V. N. Karazin Kharkiv National University, 2020. 301 p.

70. Shtovba S. D., Kozachko O. M. Machine learning: starter course : e-learning manual. Vinnytsia: VNTU, 2020. 81 p.

71. Gorokhovatsky, V. O., and Tvoroshenko, I. S. "Metody intellektnoho analizy ta obroblennia danyh" [Methods of intellectual analysis and data processing]. Guide. Kharkiv: KNURE, 2021. 92 p. Access Mode: https://openarchive.nure.ua/server/api/core/bitstreams/2e55d639-52fd-48d9-b7b7-14989f49f291/content

72. Savchenko, A. S., and Sinelnikov, O. O. "Metody ta sistemi instyvnosti instantnosti" [Methods and systems of artificial intelligence]. manual. Kyiv: NAU, 2017. 176 p. Access Mode: https://pdf.lib.vntu.edu.ua/books/2020/Savchenko_2017_176.pdf

73. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). "Why should i trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data analysis (pp. 1135-1144).

74. Josua Naiborhu. How to Interpret Black Box Models using LIME (Local Interpretable Model-Agnostic Explanations) : website, 2022. URL: https://www.freecodecamp.org/news/interpret-black-box-model-using-lime/

APPENDIX A
PYTHON BASICS: SYNTAX, DATA TYPES, BASIC COMMANDS AND BASIC LIBRARIES

We recommend that you familiarize yourself with the following Python infographic:

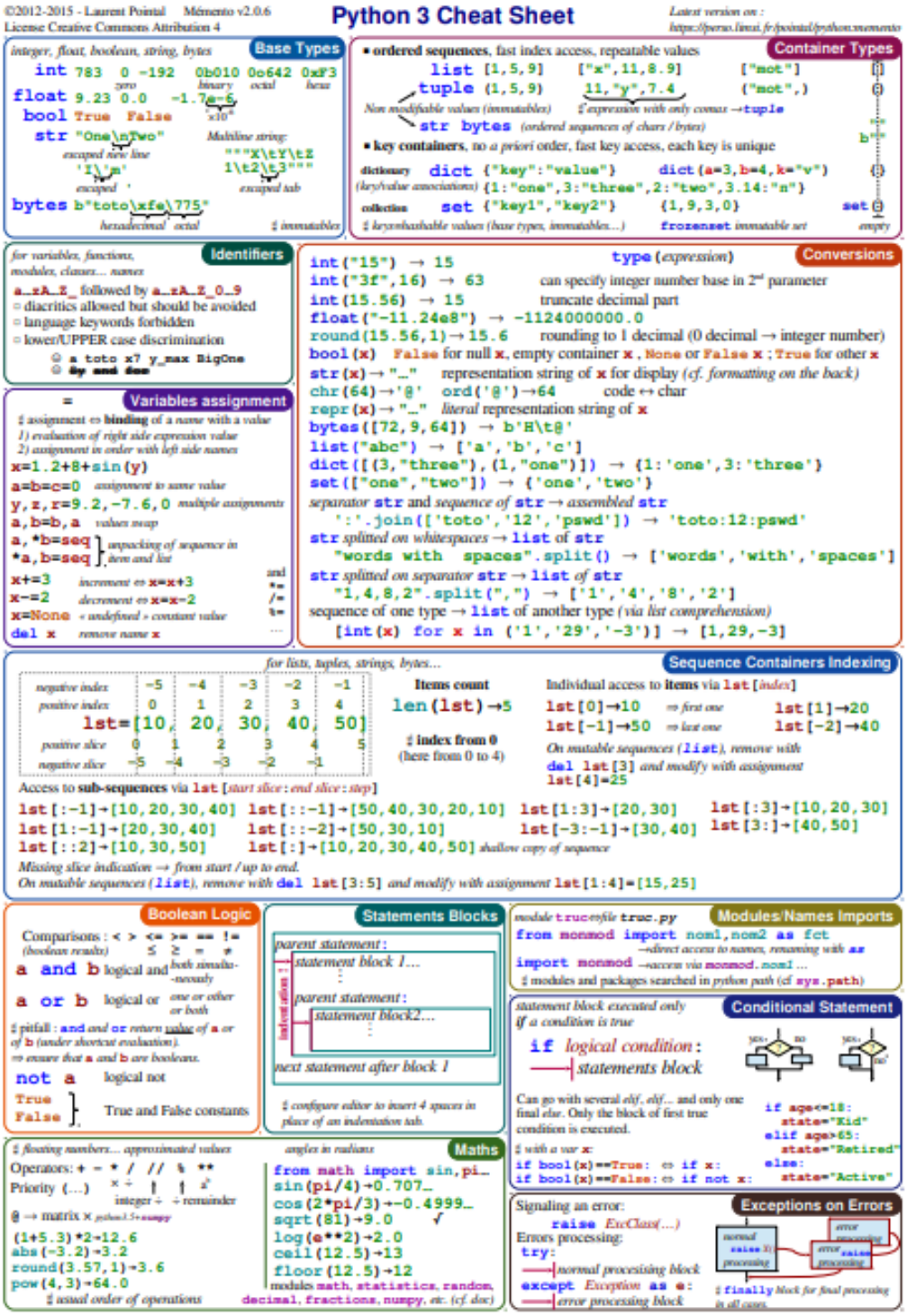


Fig. A.1 – Python syntax. Part I

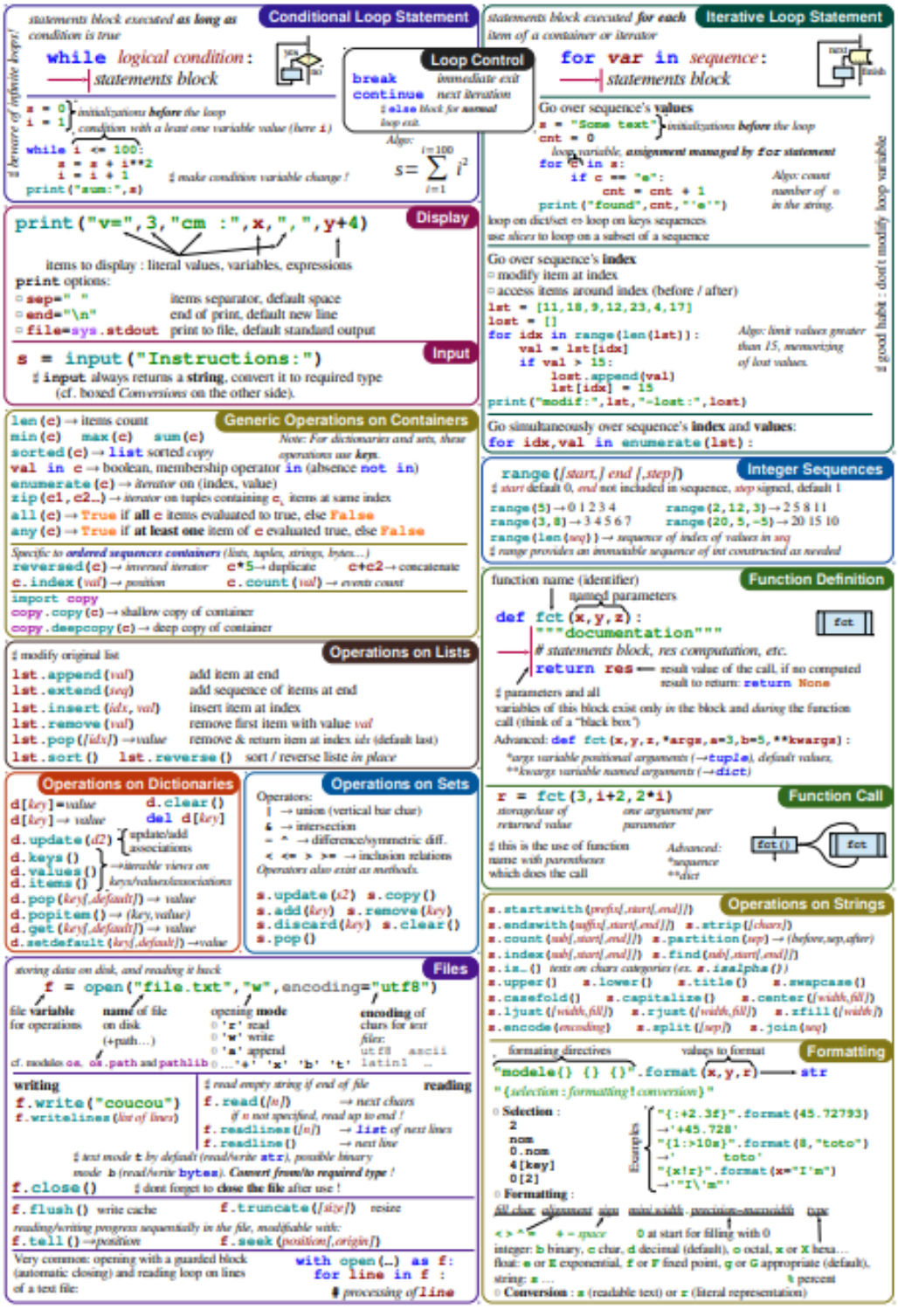


Fig. A.2 – Python syntax. Part II

Also, you should familiarize yourself with open materials about Python at documentation.

To write high-quality and readable programs, we recommend that you familiarize yourself with «PEP 8» – this is a document that defines the standards

for code design for the Python programming language ("PEP" – "Python Enhancement Proposal"): recommendations and rules for formatting code, variable names, placement of parentheses, indentation, etc.

An infographics about date and time operations is useful.

NumPy – is a fundamental package for scientific computing in Python. Operations with arrays and matrices are supported, including linear algebra operations, various mathematical and logical functions, sorting, selection, input/output, basic statistical operations, etc.

It is useful to study the following infographics of basic data operations in NumPy, Pandas, etc. libraries.:

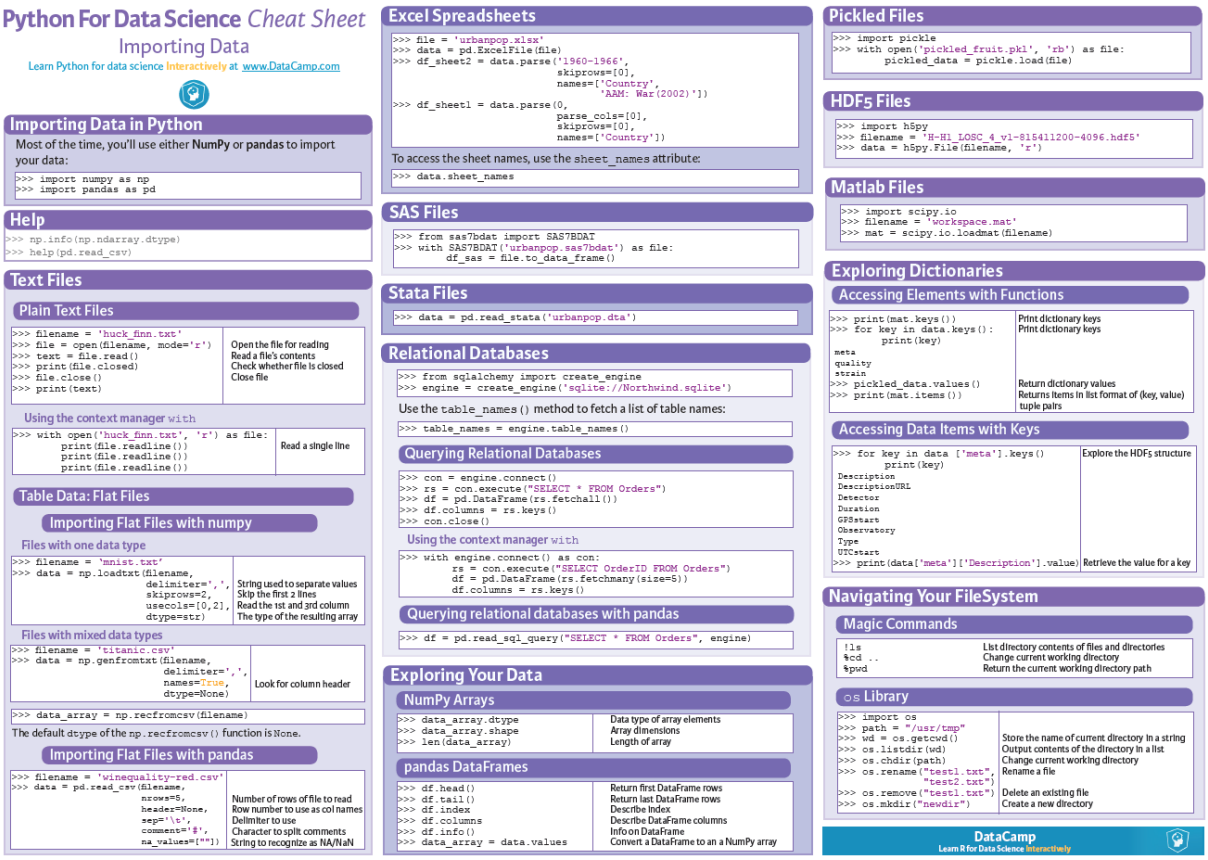


Fig. A.3 – Basic Python commands for importing data

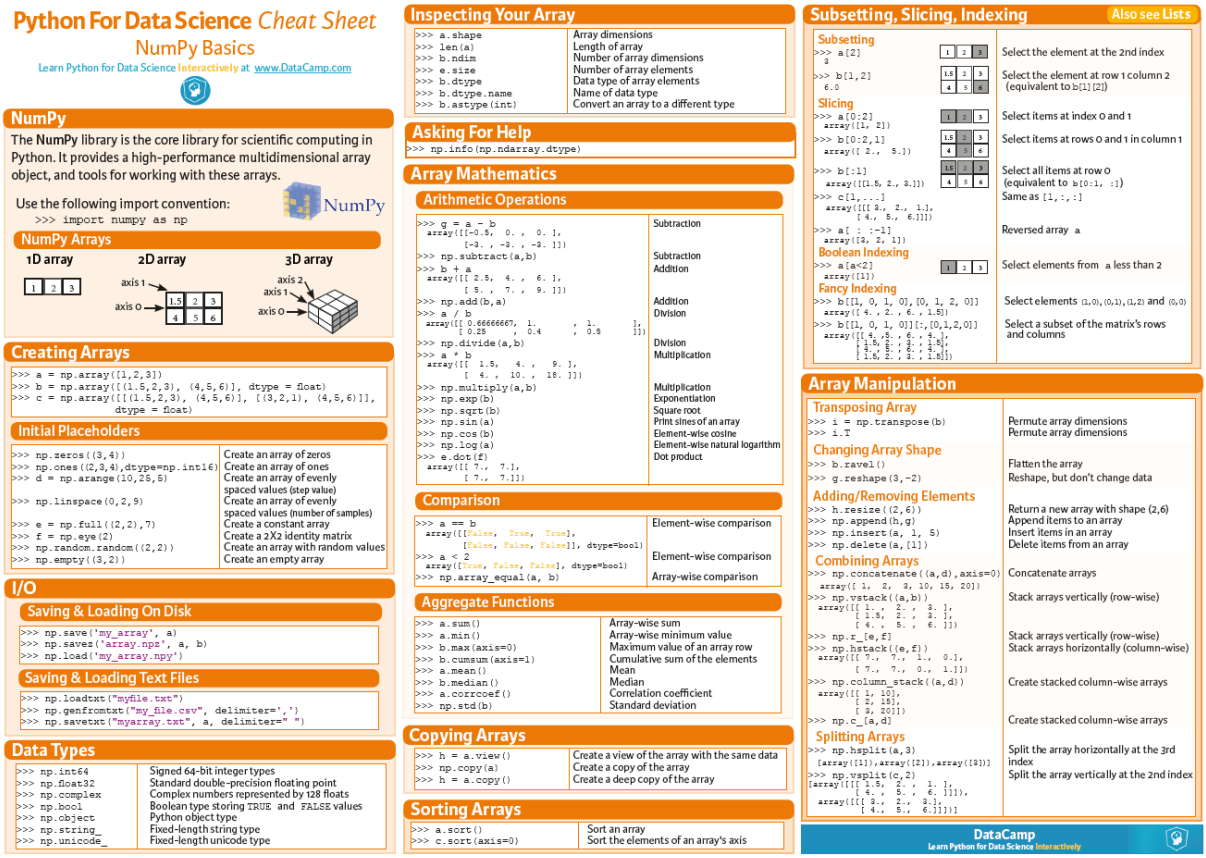


Fig. A.4 – The NumPy library

See more details about NumPy in textbooks:
- Tutorial;
- QuickStart;
- 100 numpy exercises.

Pandas – is the main basic high-speed Python library for working with tabular data in the form of dataframes.

Most Python data analysis libraries work with information in either NumPy or Pandas data format. For more details about operations in Pandas, see in infographics:

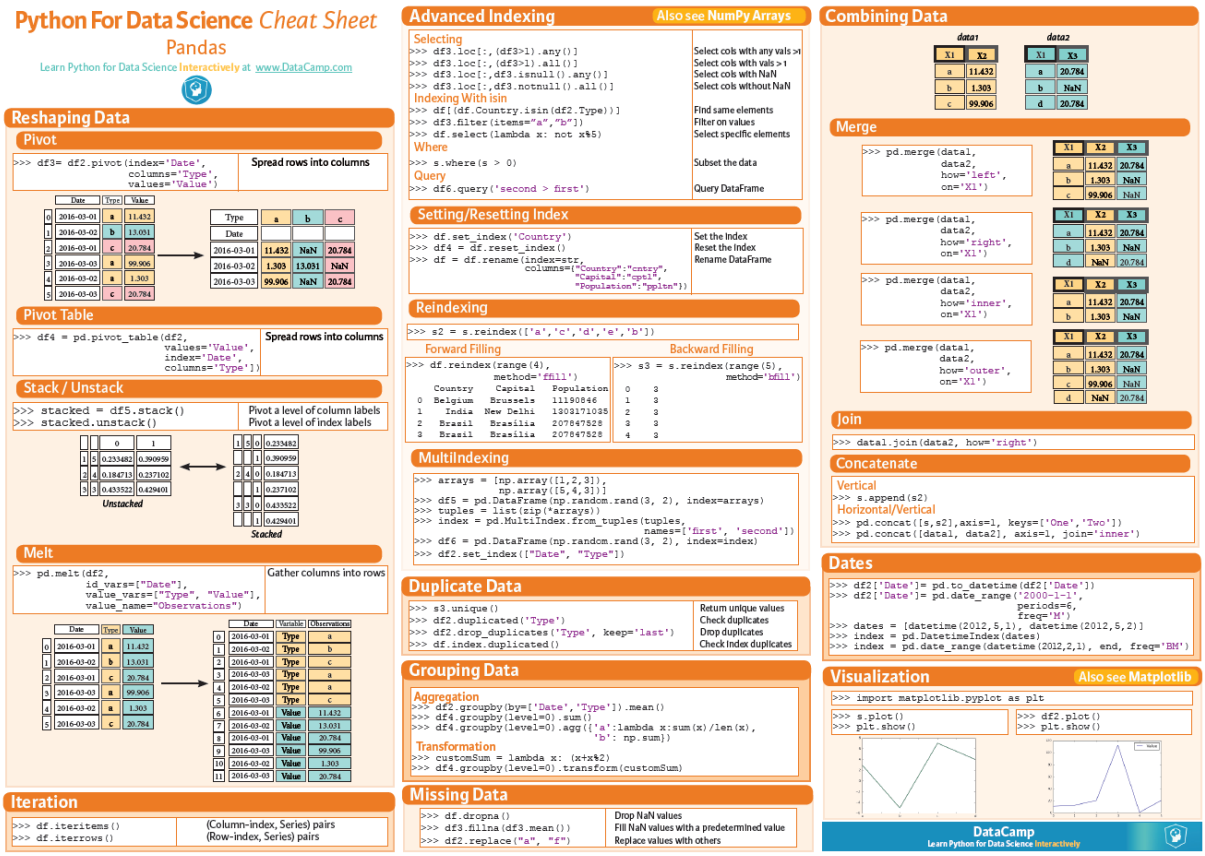


Fig. A.5 – Pandas library

Important Pandas dataframe operations are as follows 3:

- **concat** – joining along one of the axes (column or row, i.e. joining vertically or horizontally)
- **join** – joining on the left, right or other;
- **merge** – mergering according to various options.

For more details about these operations, see in documentation Pandas (it is useful to review this entire documentation file at least once).

APPENDIX B
BUILDING YOUR OWN DATASET IN THE KAGGLE ENVIRONMENT

To create *your own datasets* in the Kaggle environment, it is recommended to use the following algorithm:

1. Create a data table in an editor that can save files in csv format (for example, MS Excel, Google.Table, Calc Apache OpenOffice or LibreOffice, Spreadsheets WPS Office, etc.) and enter all the necessary data into it and perform formatting cleaning and data refactoring, to prepare for a convenient appearance for automatic processing:

- each table is on a separate sheet;
- do not merge cells;
- name all columns with words with small letters or numbers, try to avoid special characters;
- if there is more than one word in the name, then combine them with the symbol "_" and not with spaces): «body_temperature»;
- dates should be optimally submitted in the format "Short date format" (for example: 04.01.24), avoid the date option where zeros are omitted in the necessary digits "4.1.24", because there are no ready-made functions for them in Python and a new special function must be developed, instead editors like MS Excel have handy features that make it easy to convert 4.1.24 to 04.01.24.

2. Save each table in a separate csv file (remember that each sheet of the table is saved in a separate csv file);
3. Determine the character encoding or change it to a known one. Open the created csv-file in the "SublimeText" editor or similar. Make sure that all characters are read correctly, or re-encode them by clicking "File/Reopen with Encoding" in the menu and selecting the required encoding from the list, and then – save the file. Usually, choose UTF-8 or others. Then, in the case of reading the data in Kaggle, it will be known exactly which encoding to specify (see the example in tip "Tip2.3" in [4]). There are still services that allow you to identify text encoding, for example, the Python library chardet. In the case of working with datasets, it is important to know the encoding exactly to guarantee its correct reading, otherwise information may be lost or it may not be read at all.
4. Create a new dataset in Kaggle Dataset: click on the "New Dataset" button and drag the file created in point 2 into a new window.
5. Assign the main name to the dataset. It will appear later in the web link at the end of the address (letters through a hyphen and then it cannot be changed!), an additional clarifying name and describe the dataset, following further instructions and templates (see examples https://www.kaggle.com/vbmokin/datasets).

Regarding Kaggle, it is important to know that it uses a built-in plagiarism checker and does not allow you to save a dataset that is already in the system!

Therefore, if you intend to create your own dataset, it must be truly original or at least contain some processing of the existing one so that there is no overlap in the data.

We systematize the dataset construction operations in Kaggle in the form of infographics in the *S(I)* coordinate system (Fig. B.1).

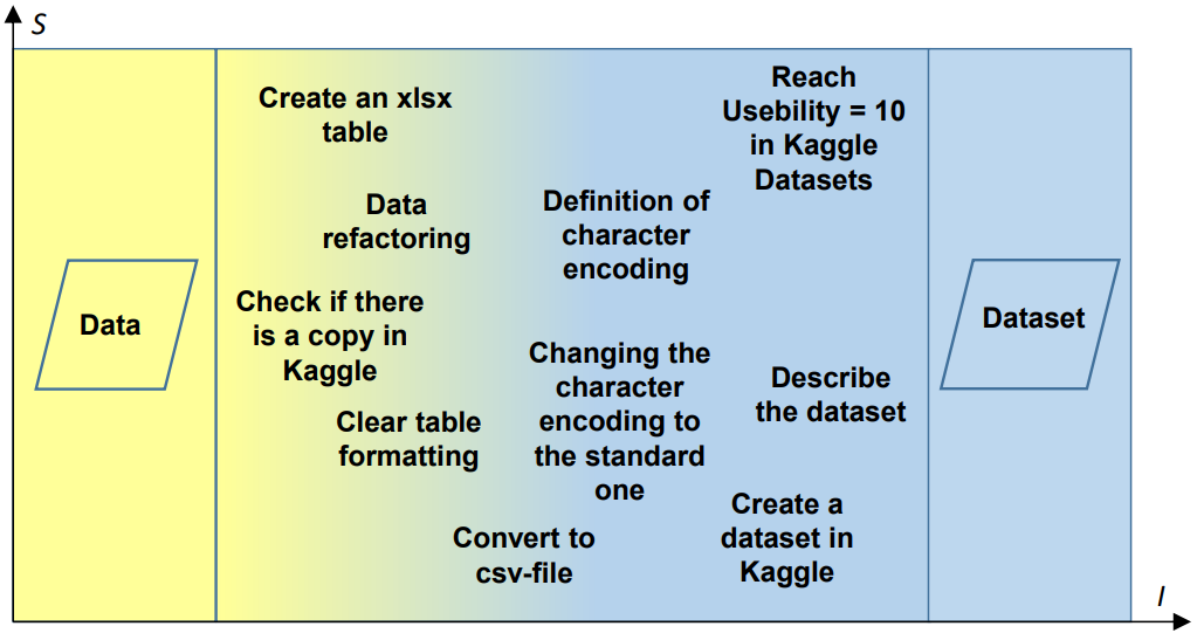


Figure B.1 – Infographics of dataset construction operations in Kaggle

It is important to make the most of open data. It is also useful because it is then easier to publish the results (with reference to the original sources of information).

APPENDIX C
EXAMPLES OF SETTING PROBLEMS FROM MACHINE LEARNING AND INTELLIGENT DATA ANALYSIS

We will give examples of setting tasks from machine learning and intelligent data analysis using the example of real tasks, as well as using the example of Kaggle datasets and Kaggle competitions.

1. **Retail Sales Forecasting**: Based on historical sales data, weather, holidays and other factors, a model is created to forecast future store sales. The results of such a model can help managers make decisions about inventory, marketing campaigns, etc.
2. **Text classification**: The task is to recognize the category of the text. For example, classifying e-mails as "spam" and "not spam", or analyzing sentiment in comments on social networks.
3. **Medical Data Anomaly Detection**: Using machine learning algorithms, medical data is analyzed to detect abnormalities or potential diseases at an early stage. This can be useful for diagnosing various diseases, such as cancer or heart disease.
4. **Recommender systems**: The challenge is to develop systems that recommend products, services or content to users based on their previous interactions. For example, movie recommendations based on a user's viewing history.
5. **Image recognition**: The goal is to recognize objects or patterns in images. This can be used to automatically recognize license plates in photographs or to analyze medical images to detect pathologies.

In addition, the Kaggle platform provides access to a variety of datasets and organizes competitions that can serve as examples of real-world machine learning tasks:

1) Titanic: Machine Learning from Disaster.
2) House Prices: Advanced Regression Techniques.
3) Sentiment Analysis on Movie Reviews.
4) COVID-19 Open Research Dataset Challenge (CORD-19).
5) GoDaddy Data Challenge.
6) Data Science for Good: City of Los Angeles.

APPENDIX D
IT INFRASTRUCTURE OF MACHINE LEARNING AND INTELLIGENT DATA ANALYSIS

Various IT infrastructures with the following components are used to implement machine learning and IDA solutions:

- programming languages: the availability of libraries and frameworks that can be used to solve machine learning problems depends on the choice of programming language;
- environments, web platforms (or cloud platforms) and services: seenvironments for developing and deploying machine learning models that provide access to computing resources and data;
- IDE (Integrated Development Environment) helps developers conveniently write and save code and debug machine learning models (usually integrates with GitHub to save and control program versions);
- databases and their management systems: used to store and structure data and files, as well as process and cache data requests according to specified criteria;
- frameworks, packages and libraries – provide access to functions, classes, methods that can be used to solve machine learning problems.

The choice of infrastructure components for solving machine learning problems depends on many *criteria*:

- customer requirements – sometimes the use of cloud resources is prohibited or, on the contrary, it is prohibited to store copies of data on a local computer;
- application conditions – sometimes, a machine learning model is required as part of an already deployed system, such as an IoT system or web service, and must be run as a separate module or in a separate container;
- available finances – for example, working with Amazon services requires considerable funds;
- knowledge and skills of programmers, including the date of engineers and data scientists – to ensure quality implementation, experienced personnel are needed, although, for a task with significant funding, suitable employees can be hired separately.

There may be other criteria, including requirements or restrictions.

The most popular *programming language* for machine learning and data analysis today is Python because it is easy to learn and use, has a large and active developer community, and offers a wide variety of machine learning libraries and frameworks.

In the past, solving problems of machine learning, artificial intelligence and intelligent data analysis were traditionally carried out in the languages Prolog, R, in the MATLAB package, etc. [69-72], but nowadays, especially in IT companies that are involved in the creation of programs and ready-to-

implement solutions, only Python is popular. R used to be a popular language for statistical machine learning because it offers powerful functions for data analysis and visualization. The majority of solution developers have already re-oriented themselves to Python, R, Prolog, MATLAB, etc. By the way, the authors of this manual used to program in R, as well as in MATLAB.

There are a number of *cloud platforms* and services that can be used to develop and deploy machine learning models. Some popular environments include:

- Amazon Web Services (AWS) – Provides a wide range of services for machine learning, including Amazon SageMaker, Amazon Lex, etc;

- Microsoft Azure Machine Learning is a machine learning platform that offers services such as model training, model deployment, and model management;

- Google Cloud AI Platform is a machine learning platform that offers services such as model training, model deployment, and model management;

- Colaboratory (or Google Colab) is a free environment for developing and deploying machine learning models in Python, which is available in the browser, and there are also free options for accessing computing capabilities using GPUs (although these capabilities are much greater on paid terms);

- Kaggle is a web-based platform for data scientists that offers competitions, a free code editor and computing power, and a forum to discuss problems.

The most popular IDE for machine learning in Python:

- PyCharm is an IDE developed by JetBrains and is free for higher education students;

- Visual Studio is a powerful but complex integrated development environment (IDE) from Microsoft;

- Anaconda (Fig. D.1) is a free Python distribution that includes many popular packages for scientific computing and machine learning, such as NumPy, pandas, Scikit-learn, TensorFlow, and PyTorch;

- Visual Studio Code (VS Code) – is a lightweight code editor from Microsoft;

- Jupyter Notebook – it is a popular environment for executing code in the browser.

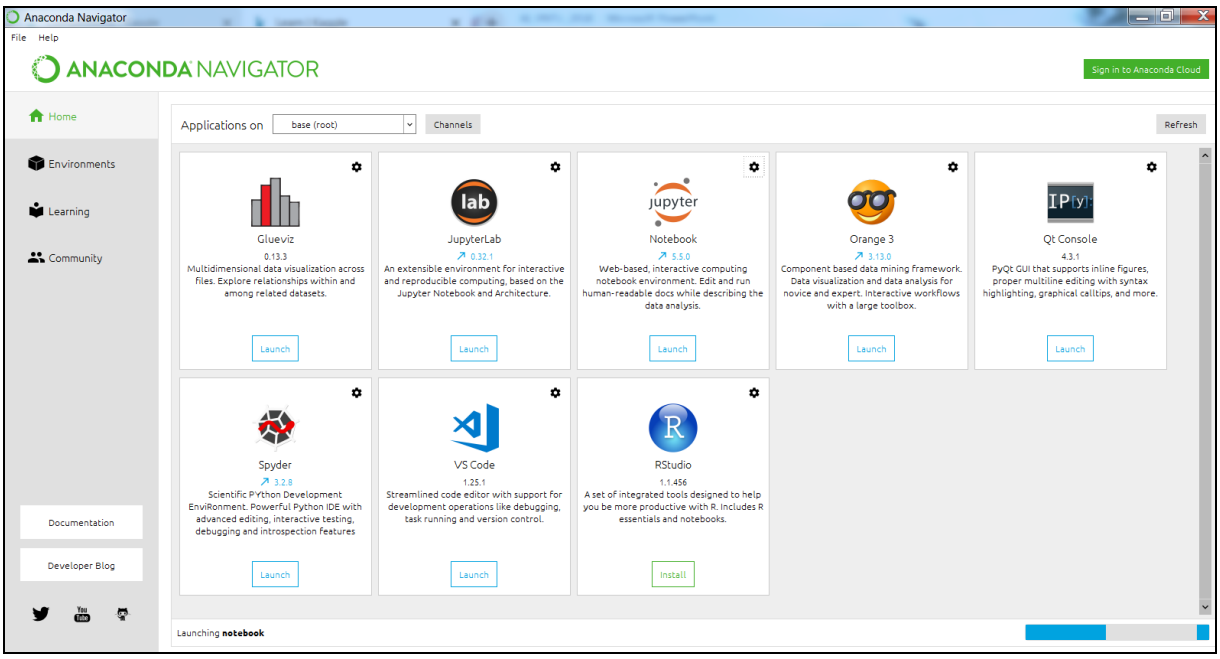


Figure D.1 – IDE Anaconda navigator interface

This manual is focused on the use of Jupyter Notebook (JN) (Fig. D.2, D.3), as the most universal environment. The authors, from their own experience, know that its code in .ipynb format can be edited in Jupyter Notebook Anaconda, Kaggle Editor, Amazon SageMaker, and Google Colab.

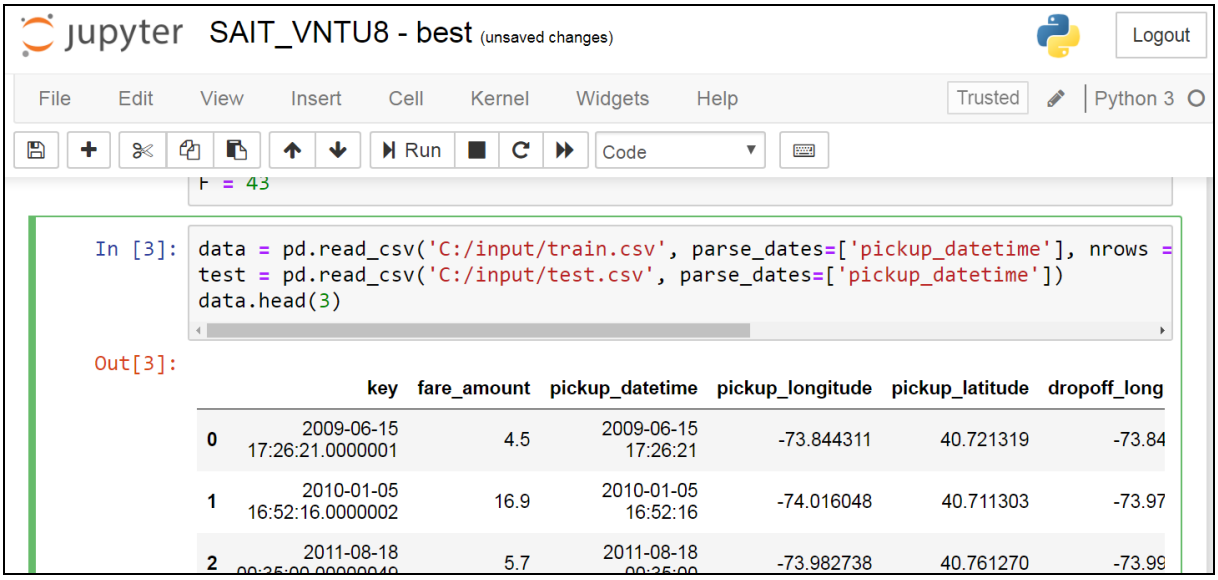


Figure D.2 – Interface Jupyter Notebook in IDE Anaconda

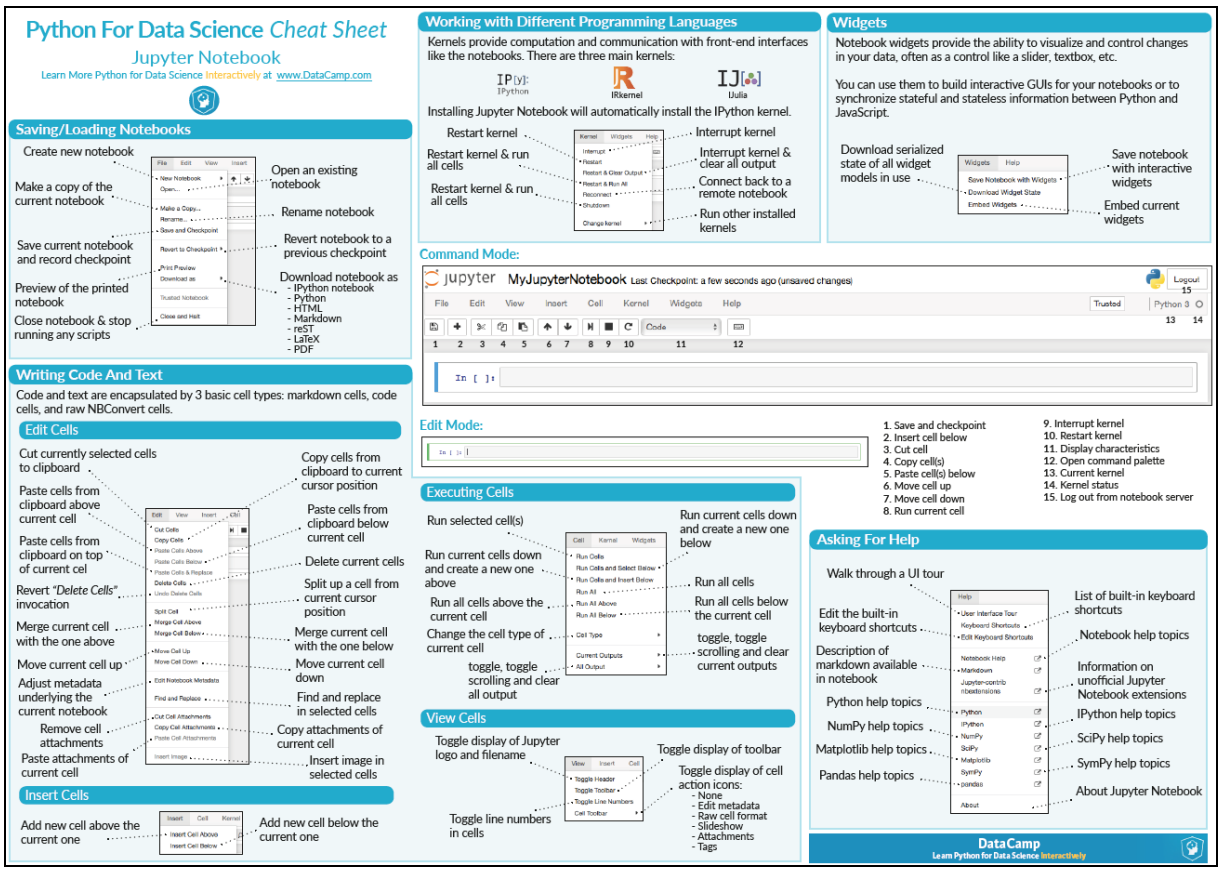


Figure D.3 – Basic elements of the Jupyter Notebook interface

Both well-structured *databases* (for example, relational or hierarchical) and unstructured ones that simply contain csv files, images, video or audio files can be used to store information in machine learning tasks. The second type is more common.

Nowadays, cloud unstructured databases have gained more popularity:

- Amazon S3 (abbreviation of "Simple Storage Service");
- Microsoft Azure Blob Storage (Microsoft);
- Google Cloud Storage;
- IBM Cloud Object Storage etc.

It is important to note that such databases use add-ons that structure the data and make SQL queries to them possible. For example, in AWS, Amazon Athena is used for this.

For local solutions, relational databases and the SQL language for working with them have become more popular.

There are a number of basic *frameworks and libraries* that can be used for machine learning:

- Scikit-learn – is a machine learning library that offers a wide variety of machine learning algorithms including classification, regression, clustering, dimensionality reduction and many others, will be covered in more detail in all subsequent chapters of the textbook;

- NumPy – is a library for scientific computing that provides access to high-performance computing, work with arrays, matrices, often used for data processing and preparation;

- Pandas – is a data analysis library that provides high-speed operations for processing structured data, such as dataframes (a table) and series (individual columns of a table);

- Matplotlib – is a data visualization library that provides graphs and charts;

- TensorFlow (TF) (integrated with Keras) – is a Google framework for machine learning, which specializes in deep learning;

- PyTorch – is Facebook's machine learning framework that also specializes in deep learning (competing with TF).

Other libraries, as well as packages (collections of interconnected modules), as a rule, use the functions, methods and classes of these basic libraries and frameworks, for example, the following are popular:

- seaborn – a data visualization library based on Matplotlib with quality infographics;

- plotly – library for interactive data visualization just in the browser;

- xgboost, lightgbm – libraries of high-performance boosting models;

- spaCy, NLTK – a library for natural language processing (NLP);

- opencv – a computer vision library used for image and video processing.

The notebooks mentioned in the author's reference notebook "Data Science for tabular data: Advanced Techniques" describe many examples of solving machine learning problems using the Python libraries and frameworks mentioned above.

APPENDIX E
LIBRARIES AND METHODS FOR AUTOMATIC EDA: PANDASPROFILING, AUTOVIZ, SWEETVIZ

Specialized Python libraries and methods that allow you to perform automatic EDA [56]:

1. PandasProfiling (PP) – everything is done by a single ProfileReport command.

The main principle of PandasProfiling is to output statistics and various information for each variable separately. It is also possible to analyze the relationship between arbitrary pairs of features on an interactive graph. It is valuable that at the beginning of the report general conclusions about the variables are given and different textual information is given in different colors: which features are highly correlated, which contain very few or very many unique values, which contain many missing values, etc. (Fig. E.1).

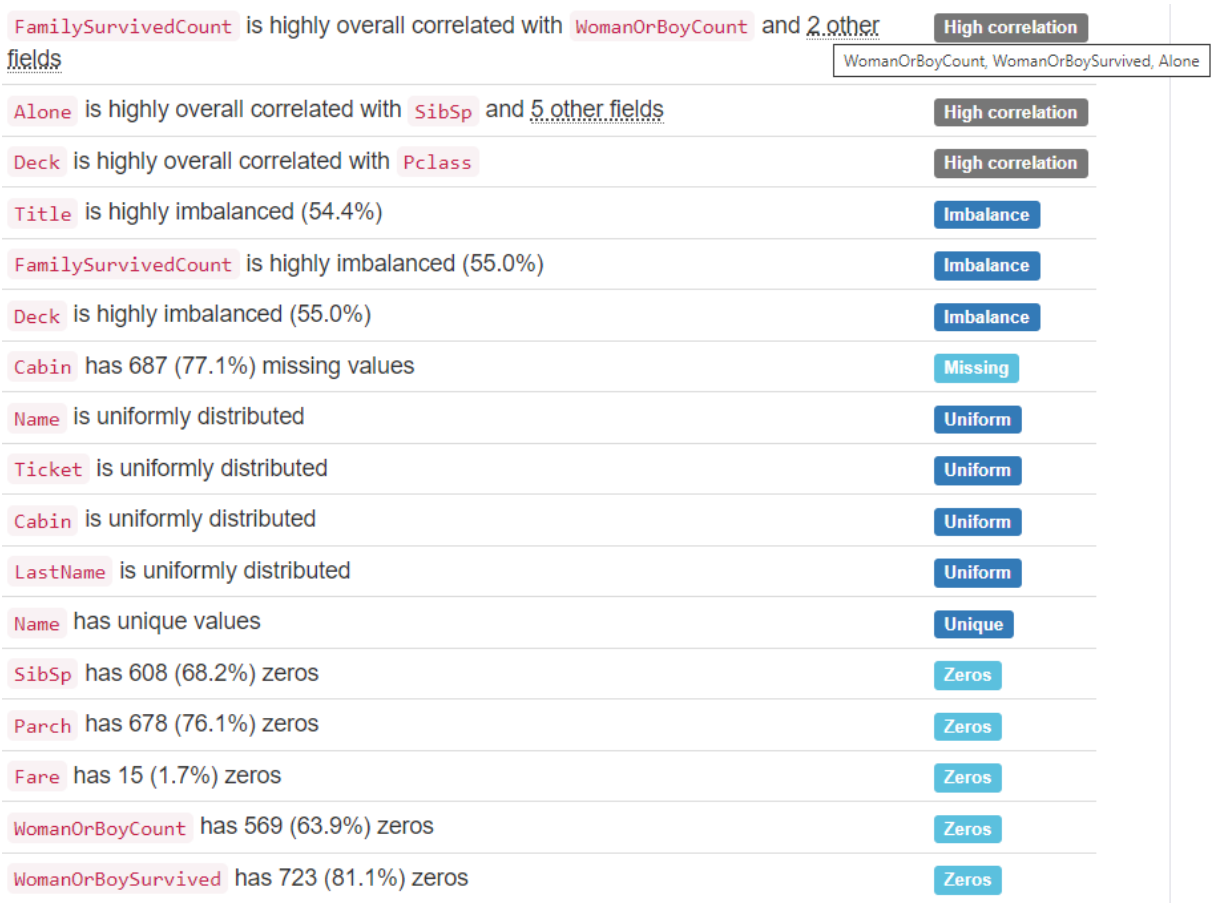


Figure E.1 – PandasProfiling statistical findings on features of the Titanic passenger competition training dataset from the author's notebook

2. AutoViz – automatically determines the graph type for each variable, depending on its characteristics. For example, numeric variables can be displayed as histograms, scatter plots, or line graphs, while categorical variables can be displayed as pie charts or bar graphs—trying to provide useful visualizations for each type of data. Can perform grouped analyses, for example, considering dependencies between variables or the distribution of variable values by a certain category. Can generate interactive graphs (Fig. E.2).

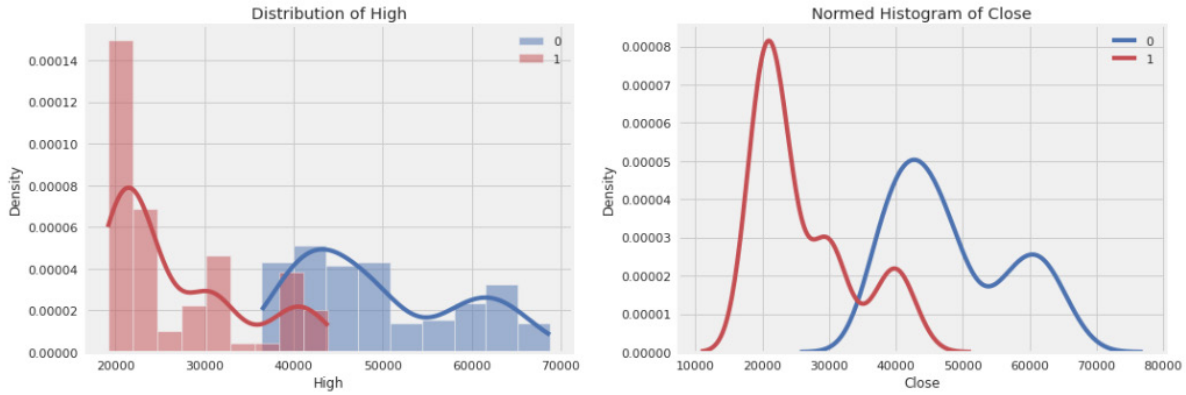


Figure E.2 – Comparison of datasets with the exchange rate of cryptocurrency for different dates corresponding to 2 classes: «0» – from 10.10.2021 to 06.04.2022, «1» – from 07.04.2022 to 03.10.2022 [56, section 6.3]

3. SweetViz – automatically constructs histograms for all numeric and categorical variables. It is possible to analyze the relationship between numerical and categorical features by building graphs of relativity. SweetViz provides convenient interfaces for comparing two different datasets, for example, training and testing, or for data collected at different times (Fig. E.3).

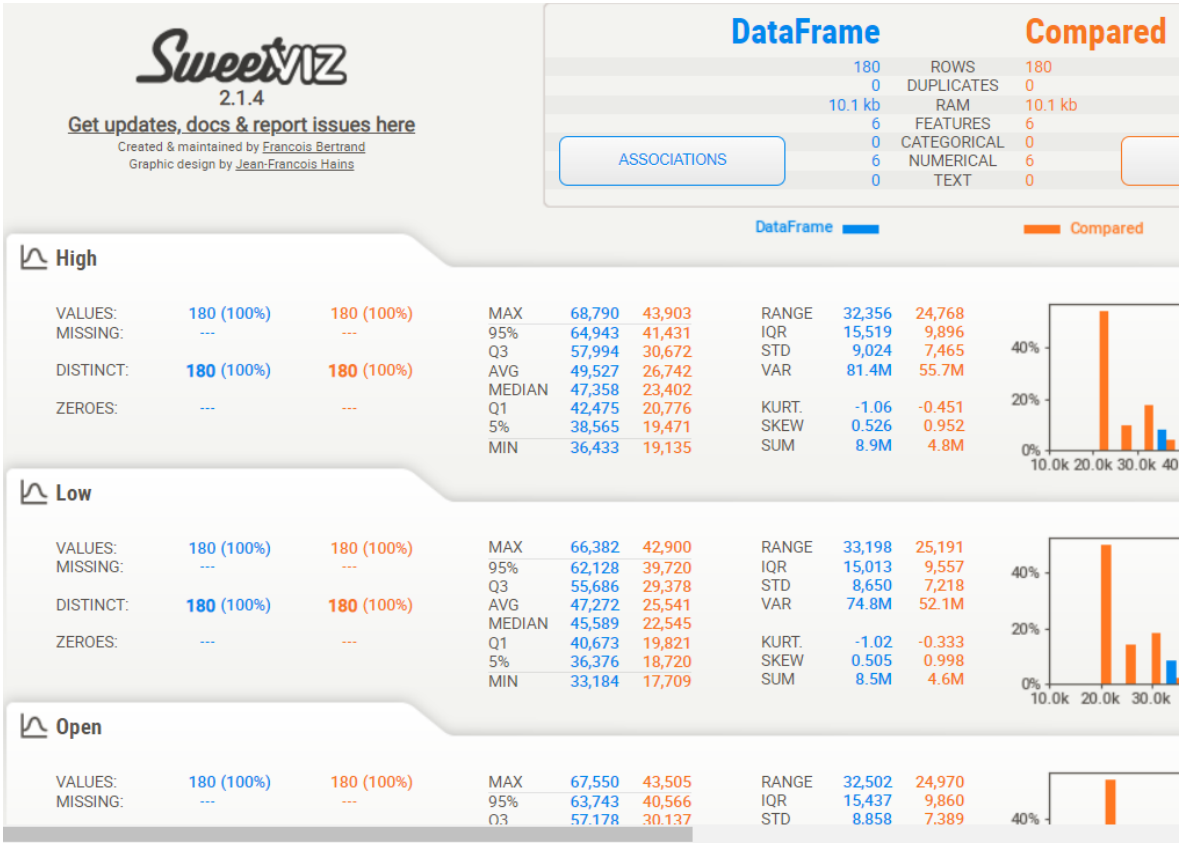


Figure E.3 – Comparison of datasets with the exchange rate of cryptocurrency for different dates corresponding to 2 classes: «Dataframe» – from 10.10.2021 to 06.04.2022, «Compared» – from 07.04.2022 to 03.10.2022 [56, section 6.2]

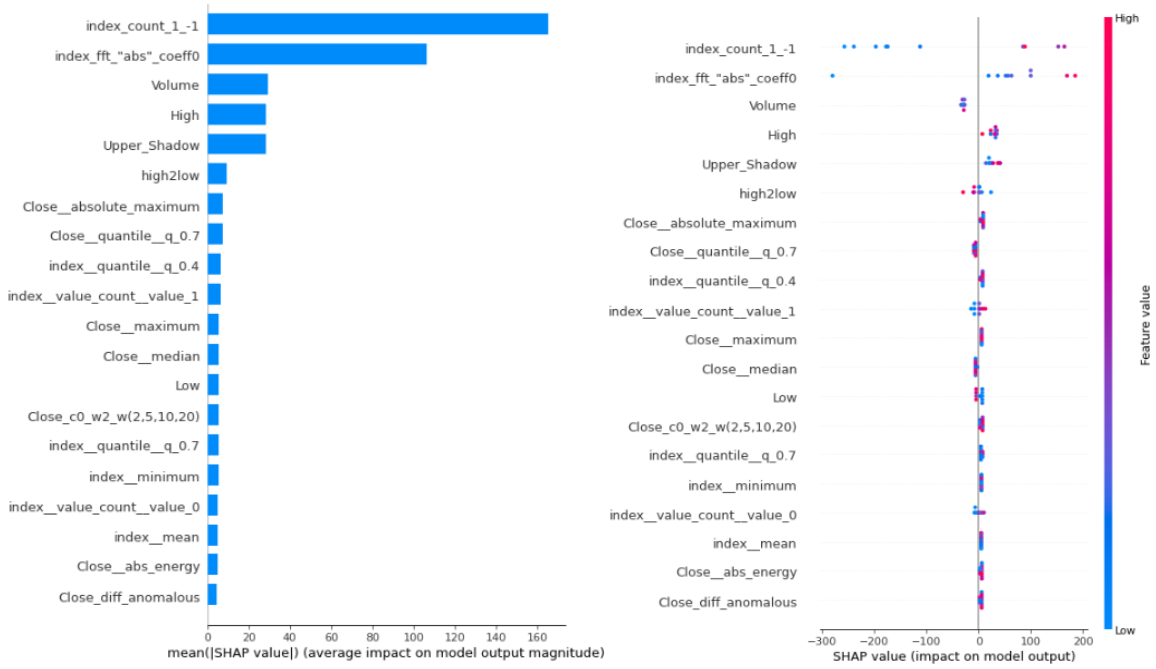
APPENDIX F
LIBRARIES SHAP, LIME FOR THE MODEL INTERPRETATION

SHAP ("SHapley Additive explanations") is a game-theoretic approach to explaining the predictions of any machine learning model using classical Shapley values from game theory. Shapley values determine how much a change in each feature affects the model's prediction, i.e. increases or decreases its value.

What is particularly valuable is that SHAP takes into account all possible subsets of features and allows consideration of interactions between them. This allows you to create interpretations that reflect not only the importance of individual features, but also their mutual influence. That is, it allows you to understand the cause-and-effect relationships and the contribution of each feature to the prognosis.

Such diagrams are effective and popular SHAP:

1. *Summary Plot* – a diagram that shows the contribution of each feature to each specific forecast (Fig. F.1).

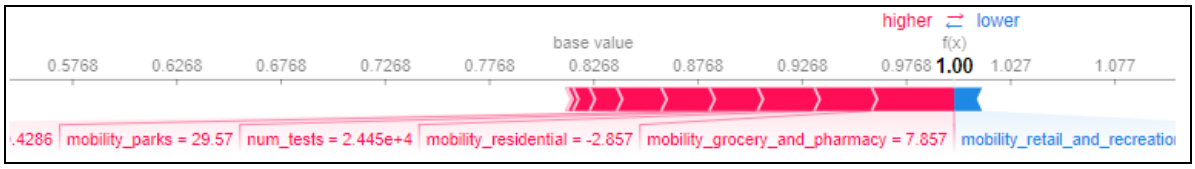


a) b)

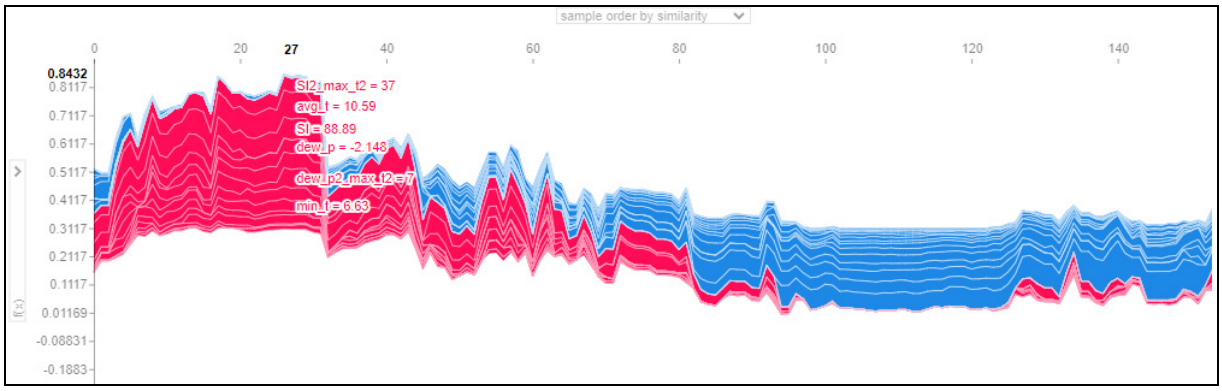
Figure F.1 – Summary Plot SHAP chart for Bitcoin exchange rate: a) with the parameter plot_type = "bar", b) with the parameter plot_type = "dot" [56]

2. *Force Plot* – a graph that displays the size and sign of the contribution of each feature for a specific forecast, taking into account the base forecast for the entire sample (Fig. F.2). Features that affect the increase of the target are displayed in red, and those that affect the decrease in blue. It is interesting that

such a graph can be built both for each prediction separately (Fig. F.2a) and for all values together, then it will have the form of an interactive graph, where you can choose different features (in the drop-down windows on the top and left) and ranges (mouse on the graph) and more carefully study the regularities (Fig. F.2b).



a)



b)

Figure F.2 – Force Plot SHAP charts for the increase in the number of coronavirus patients in Ukraine: a) for one date, October 20, 2020;
b) compilative interactive diagram based on the Tree Explainer method for a wider set of features for April-October 2020 (from the author's notebook)

Diagram F.2a shows that, according to Google trends, the negative red value of mobility_residential and the positive blue value of mobility_retail_and_recreation reduce the number of new coronavirus patients. Other indicators are shown in red and are positive, so their increase contributes to the increase in the number of new patients: mobility_grocery_and_pharmacy, number of tests, mobility_parks, etc. And on the diagram F.2b it can be seen that the regularities underwent significant changes during the entire range of observations, therefore, for forecasting in the medium and long term, these features can, rather, reduce the accuracy of forecasting.

3. *Dependency Plot* – a scatter diagram that shows how a change in the value of a specific feature affects the contribution of this feature to the forecast (Fig.. 3.6).

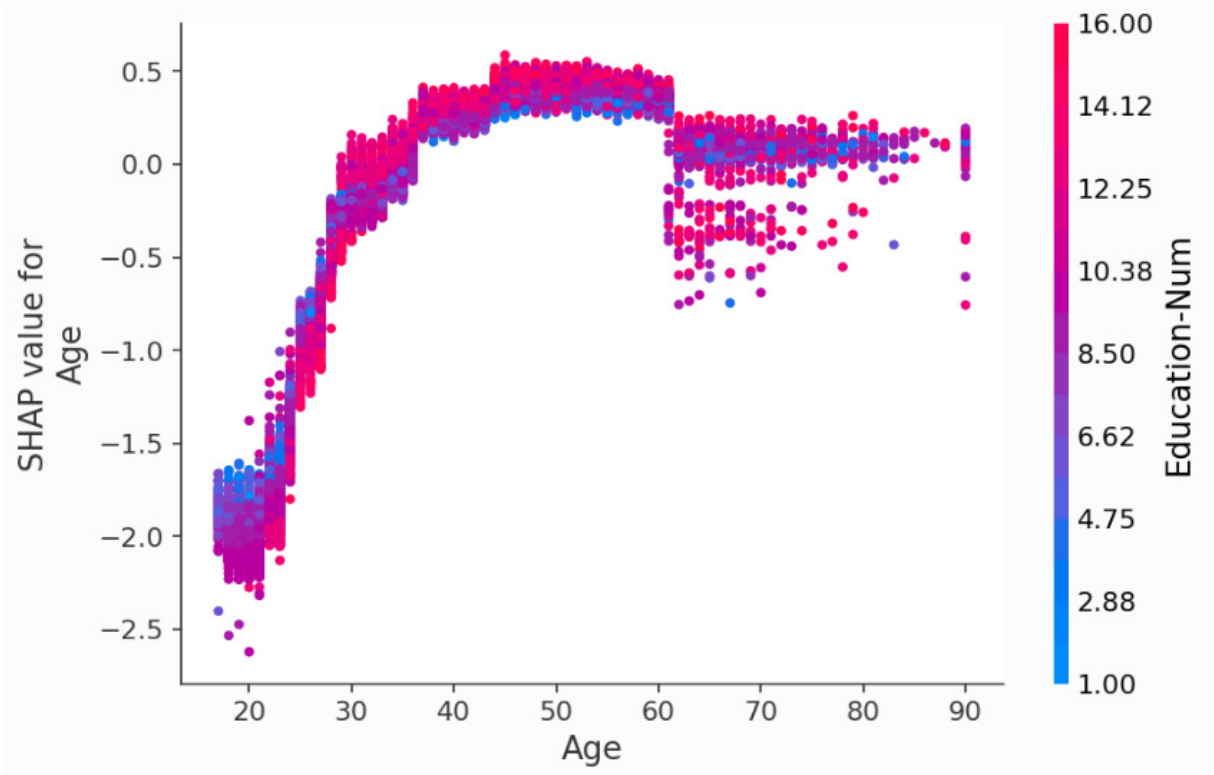


Figure F.3 – A Dependency Plot SHAP diagram that illustrates the logarithm of the odds of earning more than $50,000 a year, depending on the person's age and education

In fig. F.3 shows that the logarithm of the chances of earning more than 50,000 dollars per year increases significantly between the ages of 20 and 40, reaching the highest values, with the highest possible level of education, somewhere from 38 to 60, and the maximum – from 45 to, approximately, 53 years.

4. *Waterfall Plot*: A plot that illustrates the way the model arrives at a particular prediction, showing the contribution of each feature at each step as a waterfall of increments with different sign and color. The bottom of the waterfall diagram begins as the expected value of the model output given the input feature values plotted in gray numbers, and then each line in the diagram shows how the positive (red) or negative (blue) contribution of each feature moves the value from the expected model output to that prediction (Fig. F.4).

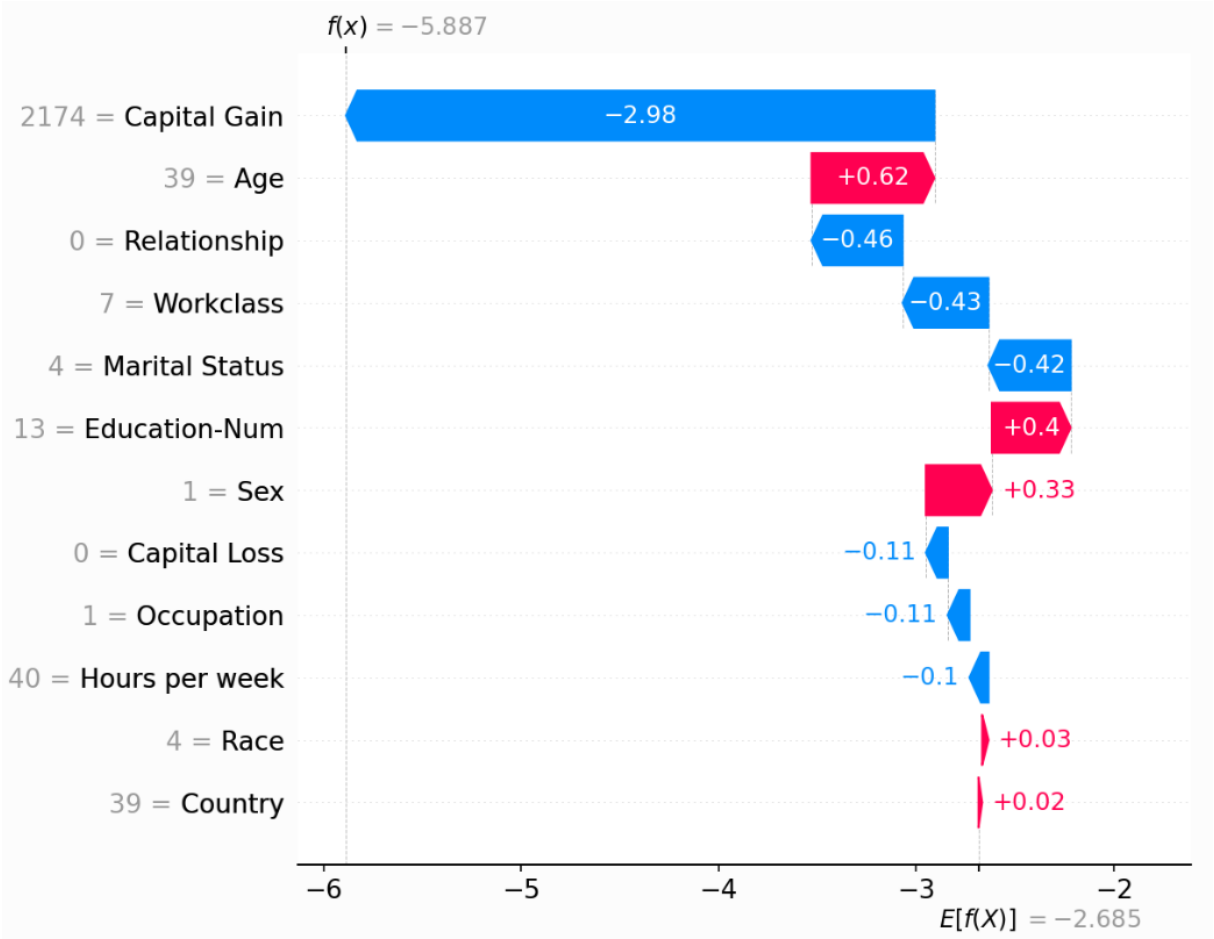


Figure F.4 – A Dependency Plot SHAP diagram that illustrates the nature and strength of the influence of certain features of the signs on the logarithm of the chances of earning more than 50 thousand dollars per year

In fig. F.4 shows that (see the description of parameter values in the "US Adult Income Dataset" dataset, for example, in the repository or elsewhere): the lack of capital growth had the greatest impact on the decrease in income; that this African American (Sex=1, Race=4) has never worked (Workclass); that he is ready to work only on manual work related to repairs (Occupation=1); that he is single (MaritalStatus = 4). Some growth is due to an age close to the optimum, when, on average, people earn the most (38-60 years), that he is male (Sex=1), that he has an education, albeit a small one (Education_Num=13: only 5-6th grades).

See other examples and their description in more detail: Data Science for tabular data: Advanced Techniques, Crypto - BTC : Analysis & Forecasting (sections 3.4), or in paper or in documentation.

The main drawback of the SHAP library is that it requires a lot of computation when analyzing large and complex neural network models or ensembles. Therefore, the LIME library is often used, which uses a number of simplifications and therefore requires less computational costs.

The basic principle of the LIME library is that any complex model is more easily approximated in the neighborhood of a particular example of data. The model is presented as a "black box". And its behavior at a given point is approximated by a linear model, and it is based on it that the explanation of the entire model as a whole is formed. In fig. F.5 gives an example that illustrates this principle.

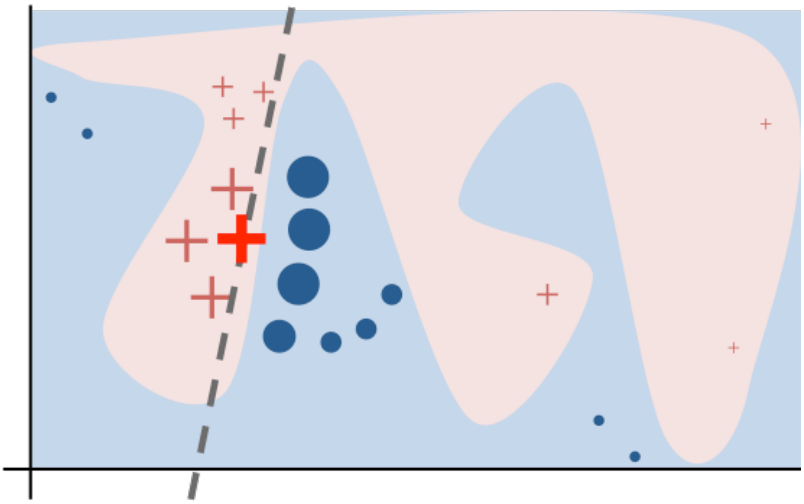


Figure F.5 – Illustration of the working principle of the method LIME [73]

The model-based function being explained is shown in blue and pink. It is obvious that it is non-linear. The large red cross is an example of the data at point X0. We take other data values in a certain neighborhood near X0, taking into account the degree of proximity to X0 in relation to the weight of these data. A collection of points (crosses) is formed, which is further approximated by a straight line (dotted line in Fig. F.5). This line allows us to characterize a certain section of regularities, but these are rather local rather than global regularities. This is the main drawback of the methods of this library – that it describes local regularities, not global ones [74].

Fig. F.6 and F.7 presents examples from the notebook as for the interpretation of the predictions made by the model for the passengers of the Titanic, whether they will survive or not according to the data of the well-known competition in Kaggle.

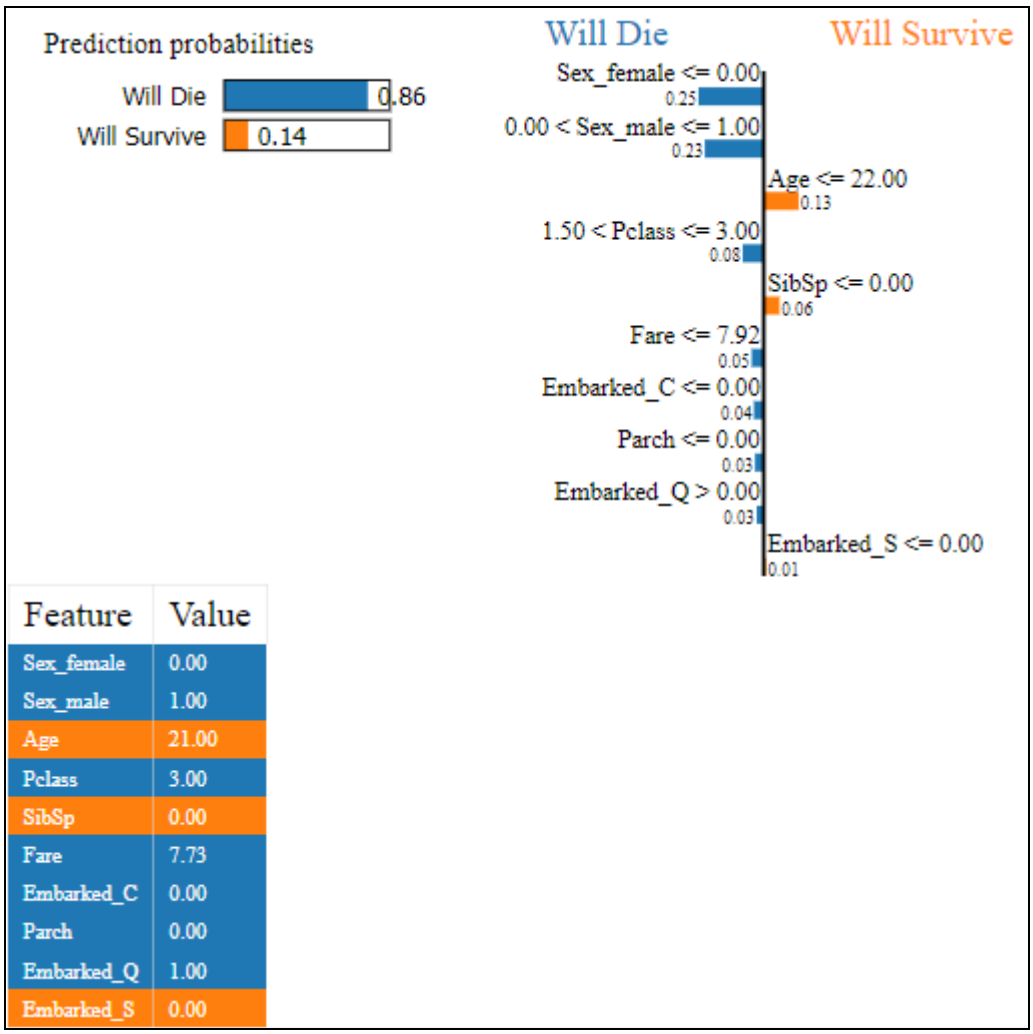


Figure F.6 – The example of predicting whether a Titanic passenger with a cheap 3rd class ticket who boarded at the last port – in Queenstown (Queenstown – "Q") will survive the disaster

As can be seen in fig. F.6, a passenger on the Titanic with a cheap 3rd class ticket, who disembarked at the last port – Queenstown, will die with a probability of 0.86. The fact that he was traveling in the hold with other 3rd class passengers and was a male left him virtually no chance of survival. In addition, as history knows, those who boarded at the last port (in Queenstown) took the least comfortable seats (far from the gangway from which the clean air came) that were still available. Although, the fact that he is young (Age=21) and that he has no brothers, sisters, or wife (SibSp=0) to worry about gives him some chances.

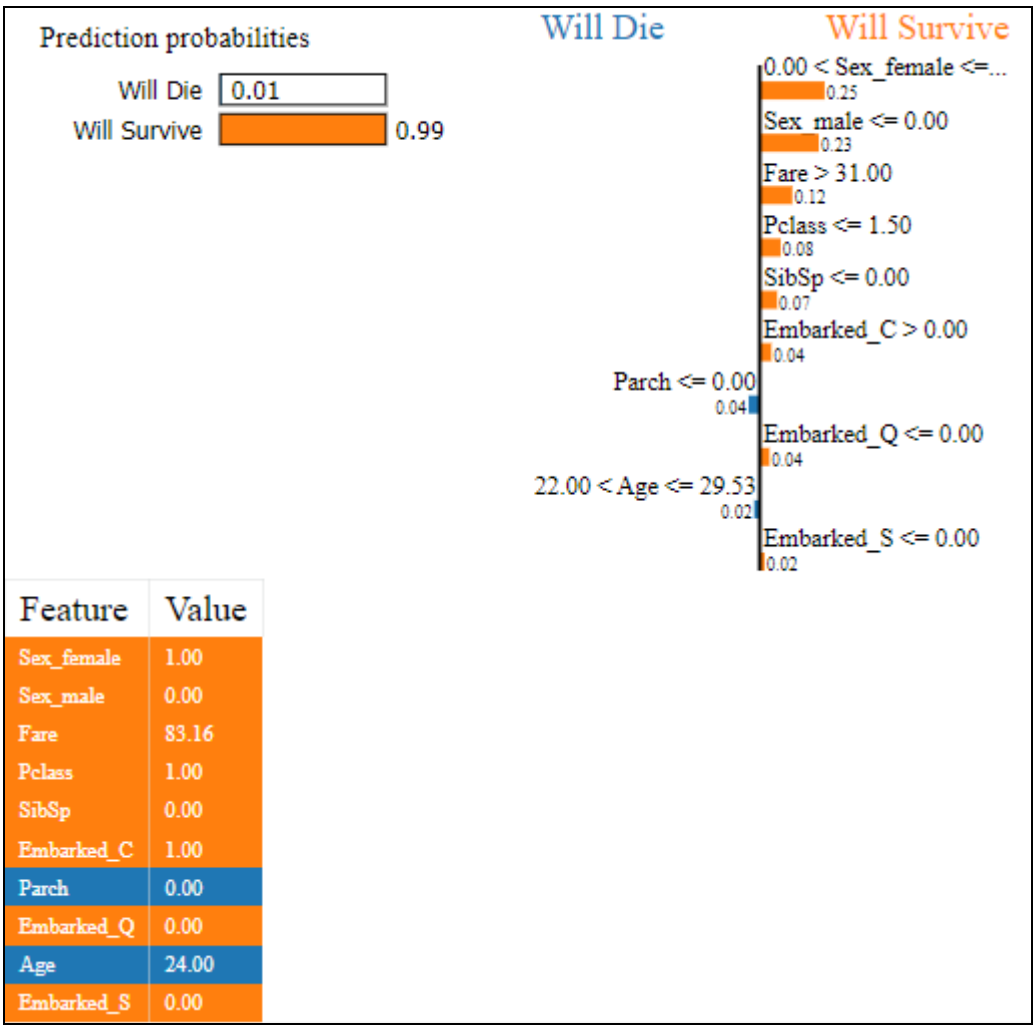


Figure F.7 – The example of predicting whether a Titanic passenger who has an expensive 1st class ticket will survive the disaster

And as can be seen from fig. F.7, the Titanic passenger has every chance to be saved. This is mostly facilitated by her female gender and an expensive 1st class ticket. It doesn't really matter in which port she sat down, since the 1st class passengers occupied separate beautiful cabins. It is somewhat surprising that her chances of survival are increased by the absence of brothers, sisters or a husband, because husbands and brothers saved women and sisters, respectively, in the first place. And it is also surprising that her chances of dying are affected by her young age (Age=24) and absence of parents (Parch=0). After all, the crew or other men would save such a young lady, and she would not have to take care of her parents. Obviously, these somewhat strange results demonstrate that, globally, the conclusions of the LIME library may not be entirely adequate. Specifically, such a woman in the training dataset would have a chance to die if she slept soundly or saved one of the children of her familiar family, that is, because of some features that are missing in this dataset. This is an example of the fact that it is difficult to draw global conclusions from one piece of data that may have signs of an anomaly.

APPENDIX G
NEURAL NETWORK ARCHITECTURES

In 2016, the Isaac Asimov Institute (USA) published an infographics of the main architectures of modern neural networks at that time (Fig. G.1).

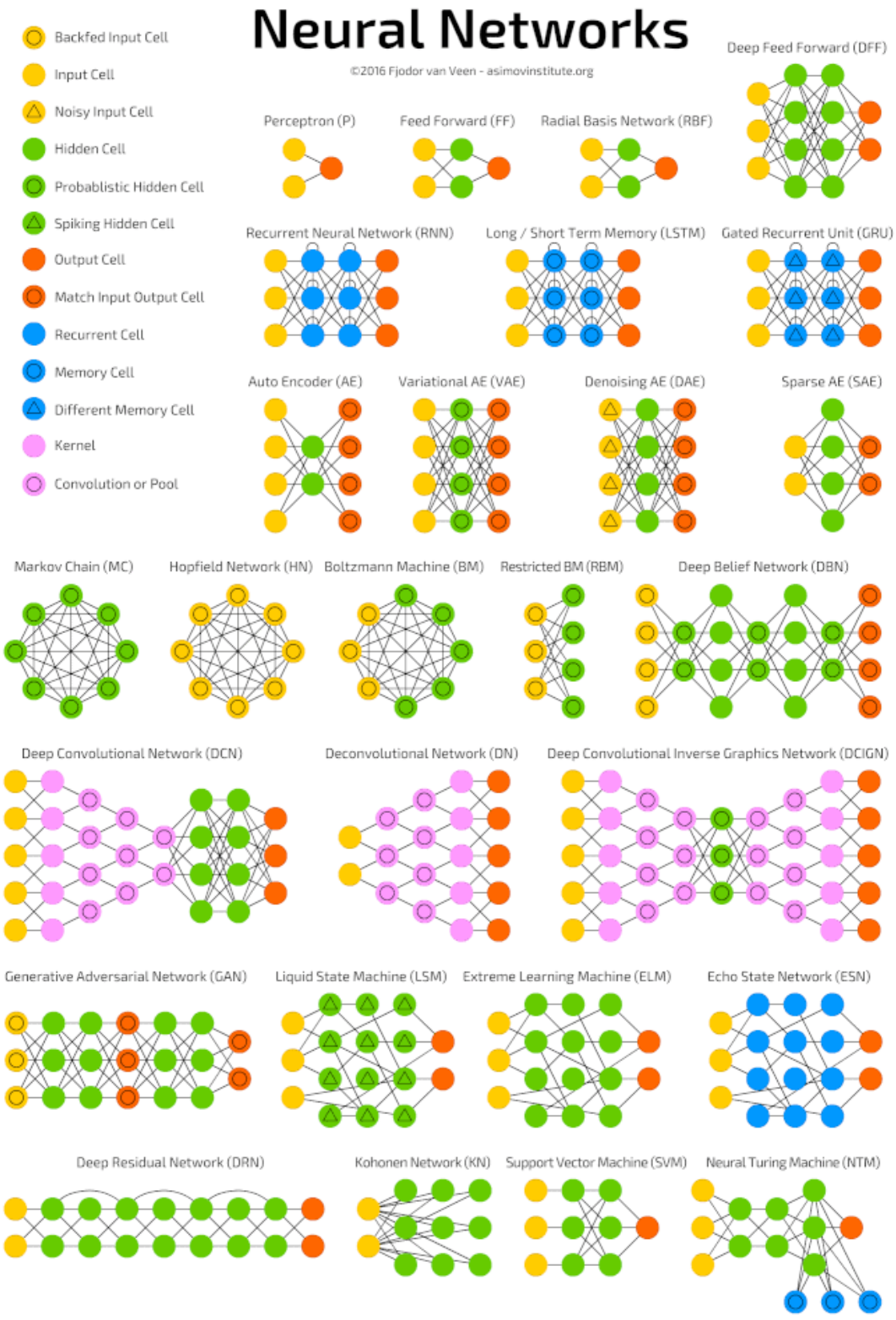


Figure G.1 – Neural network architectures as of 2016

More modern neural network architectures are the following (the number usually means the number of hidden layers):

- ResNet (ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152, ResNeXt, WideResNet architectures are popular);
- EfficientNet (EfficientNet-B0, EfficientNet-B1, EfficientNet-B2, EfficientNet-B3, EfficientNet-B4, EfficientNet-B5, EfficientNet-B6, EfficientNet-B7) and others.

Educational electronic publication

Vitalii B. Mokin
Mykola G. Pradivliannyi

MACHINE LEARNING, INTELLIGENT DATA ANALYSIS AND ARTIFICIAL INTELLIGENCE OF THINGS

Textbook

The manuscript was designed by *V. Mokin*

Editor: *M. Pradivliannyi*

The original layout was made by *V. Mokin*

Signed for publication December 12,2024.
Typeface Times New Roman. Deputy No P2024-080.

Publisher and manufacturer
Vinnytsia National Technical University,
Editorial and Publishing Department.
VNTU, GNK, room 114.
95 Khmelnytskyi highway,
Vinnytsia, 21021.
press.vntu.edu.ua;
Email: irvc.vntu@gmail.com
Certificate of the subject of publishing series DK
No 3516 dated 01.07.2009.

Електронне навчальне видання

Мокін Віталій Борисович
Прадівлянний Микола Григорович

МАШИННЕ НАВЧАННЯ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ ТА ШТУЧНИЙ ІНТЕЛЕКТ РЕЧЕЙ

Навчальний посібник (*англійською мовою*)

Рукопис підготував *В. Мокін*
Редактор: *М. Прадівлянний*
Оригінал-макет виготовив *В. Мокін*

Підписано до видання 18.12.2024 р.
Гарнітура Times New Roman. Зам. № Р2024-080

Видавець та виготівник
Вінницький національний технічний університет,
Редакційно-видавничий відділ.ВНТУ, ГНК, кімн. 114.
95 Хмельницьке шосе, Вінниця, 21021.
press.vntu.edu.ua; *email*: irvc.vntu@gmail.com
Свідоцтво суб'єкта видавничої справи серія ДК № 3516 від 01.07.2009 р.

Мокін, Віталій

Машинне навчання, інтелектуальний аналіз даних та штучний інтелект речей : навчальний посібник [Електронний ресурс] / В. Б. Мокін, М. Г. Прадівлянний – Вінниця: ВНТУ, 2024. – (PDF, 230 с.)

Посібник містить теоретичні відомості про основні концепції, методи та інструменти науки про дані (Data Science), машинного навчання, штучного інтелекту, інтелектуального аналізу даних, штучного інтелекту речей, а також практичні рекомендації щодо застосування сучасних технологій у вирішенні численних прикладних задач, задач та проблем системного аналізу. Наведено перелік контрольних питань для перевірки набутих теоретичних знань і практичних навичок.

Навчальний посібник призначений для іноземних студентів, які навчаються за спеціальностями 124 «Системний аналіз» та 126 «Інформаційні системи та технології» II та III рівнів при вивченні дисциплін «Інтернет речей та інтелектуальний аналіз даних», «Інформаційні технології моніторингу та аналіз стану складних систем», «Інформаційні технології моніторингу та аналізу даних», «Інформаційні інтелектуальні технології», «Системний аналіз», «Розумні технології», а також для студентів, які проходять виробничу та переддипломну практику і для педагогічної практики аспірантів. Він також може бути корисним для студентів інших напрямків: менеджмент, фінанси, будівництво, кібербезпека, біоінженерія, електротехніка та машинобудування, сільське господарство, біологія, медицина, освіта тощо. У посібнику подано багато прикладів завдань із цих напрямків.