FAKULTA
PŘÍRODOVĚDNĚ-HUMANITNÍ
A PEDAGOGICKÁ TUL

Bakalářská práce

Comparison of supervised machine learning methods used for classification

Studijní program: B0541A170015 Matematika
Autor práce: **Věnceslav Chumchal**
Vedoucí práce: Mgr. Martin Schindler, Ph.D.
Katedra aplikované matematiky

Liberec 2023

FAKULTA
PŘÍRODOVĚDNĚ-HUMANITNÍ
A PEDAGOGICKÁ TUL

Zadání bakalářské práce

Comparison of supervised machine learning methods used for classification

Jméno a příjmení:	**Věnceslav Chumchal**
Osobní číslo:	P20000724
Studijní program:	B0541A170015 Matematika
Zadávající katedra:	Katedra aplikované matematiky
Akademický rok:	2020/2021

Zásady pro vypracování:

Student nastuduje a popíše základní metody strojového učení s učitelem (supervised machine learning methods), zaměří se zejména na metody používané pro klasifikaci jako např. logistická regrese, support vector machines, rozhodovací stromy, náhodné lesy, k-means, k-nearest neighbors a umělé neuronové sítě. Vybrané metody budou porovnány pomocí různých metrik jednak na vhodně zvolených simulovaných datech příp. i na datech reálných. Výpočty budou provedeny pomocí Python a R knihoven.

Rozsah grafických prací:
Rozsah pracovní zprávy:
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Angličtina

Seznam odborné literatury:

SHALEV-SHWARTZ, Shai and Shai BEN-DAVID. *Understanding Machine Learning: From Theory to Algorithms* [online]. Cambridge: Cambridge University Press, 2014. Retrieved z: doi:10.1017/CBO9781107298019
BURGER, Scott. *Introduction to Machine Learning with R*. O'Reilly Media, 2018. ISBN 9781491976449.
GIUDICI, Paolo a Silvia FIGINI. *Applied Data Mining for Business and Industry*. 2nd Edition. Wiley, 2009. ISBN 978-0-470-74582-3.
ANDĚL, Jiří. *Statistické metody*. Páté vydání. Praha: Matfyzpress, 2019. ISBN 978-80-7378-381-5.

Vedoucí práce: Mgr. Martin Schindler, Ph.D.
Katedra aplikované matematiky

Datum zadání práce: 4. června 2021
Předpokládaný termín odevzdání: 30. července 2021

L.S.

Mgr. Martin Schindler, Ph.D.
vedoucí bakalářské práce

doc. RNDr. Miroslav Koucký, CSc.
vedoucí katedry

V Liberci dne 4. června 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

25. dubna 2023 Věnceslav Chumchal

SROVNÁNÍ KLASIFIKAČNÍCH METOD STROJOVÉHO UČENÍ S UČITELEM

ABSTRAKT

Tato práce je stručný úvod do aplikovaného strojového učení, včetně aplikace na reálná data. Nejprve zadefinujeme základní pojmy strojového učení. Následně popíšeme obecnou klasifikační úlohu, vybrané klasifikační algoritmy, a metriky. V praktické části vyzkoušíme jeden z pracovních postupů používaných při aplikaci strojového učení na konkrétní úlohu, včetně podrobného popisu příslušné datové sady. Na závěr uvedeme empirické srovnání použitých algoritmů.

Klíčová slova: strojové učení; empirické srovnání; klasifikace; ML workflow;

COMPARISON OF SUPERVISED MACHINE LEARNING METHODS USED FOR CLASSIFICATION

ABSTRACT

This work briefly introduces applied machine learning, including application to actual data. First, we define the basic concepts of machine learning. We then describe a general classification task, selected classification algorithms, and metrics. In the practical part, we will test one of the workflows that apply machine learning to a specific task, including a detailed description of the corresponding dataset. Finally, we conclude with an empirical comparison of the algorithms used.

Keywords: machine learning; empirical comparison; classification; ML workflow;

ACKNOWLEDGEMENTS

Děkuji svému vedoucímu Mgr. Martinu Schindlerovi Ph.D. za to, že se mě ujmul a byl mi nápomocen. Taktéž děkuji panu děkanovi prof. RNDr. Janu Pickovi, CSc. a vedoucí studijního oddělení, paní Miroslavě Kretschmerové za poskytnutí dat k vypracování praktické části práce a konzultace k nim. Ke korektuře této práce byl mimo jiné použit software Grammarly. Tímto děkuji jeho autorům za tento velmi užitečný nástroj a univerzitě za pořízení a poskytnutí profesionální licence všem studentům. Mé poděkování rovněž patří všem, kteří mi pomohli dosáhnout znalostí a schopností potřebných k sepsání této práce.

CONTENTS

ACRONYMS

AUC Area Under the ROC Curve. 28, 31, 40–43

eCDF empirical Cumulative Distribution Function. 37

EDA Exploratory Data Analysis. 30, 31

ERM Empirical Risk Minimization. 14

FPR False Positive Rate. 28

i.i.d. independent and identically distributed. 15

KDE Kernel Density Estimation. 38

KNN K-Nearest Neighbors. 23, 24, 39

ML Machine Learning. 11, 12, 15–18, 28, 30, 45

MLP Multi-Layer Perceptron. 24, 39, 43, 45

PAC Probably Approximately Correct. 14–16

RF Random Forest. 21, 22, 39, 40, 42–45

ROC Receiver Operating Characteristic curve. 28, 29, 41, 43

SVM Support Vector Machines. 22, 23, 39

TN True Negative. 27

TP True Positive. 27

TPR True Positive Rate. 27, 28

INTRODUCTION

Why bother with machine learning, even as a scientist from a distant field? Is it some niche area only for enthusiasts? Is it more statistics, informatics, or something else?

In the last decade we can see advances in many application areas of machine learning such as natural language processing - with an idea how to represent words as vectors [1], how to process structure of text and meaning of words in various context with transformer architecture [2] or development of general language model capable of generating text indistinguishable from human one [3], image generation - with architecture using two or more neural networks to create unseen images [4], improvements over this to generate image sets with high variation by controlling strength of each feature [5], object detection - with models capable of doing inference in real time on standard hardware [6], content recommendations - with successful application on sophisticated large scale environment as YouTube is [7], games - with reinforcement learning algorithm beating human in Go [8] or system able to defeat e-sports professionals in fast paced online game Dota 2 [9], bio-informatics - with program able to predict 3D structure of proteins with atomic accuracy [10], knot theory - with usage of deep learning to find patterns in data to give a direction where is fruitful to pursue with conventional methods [11], linear algebra - with the first try to develop faster matrix multiplication algorithms using only machine learning [12], computational modeling - saving computational cost with deep learning meta-model [13], particle physics - with relatively long usage of machine learning for analyzing LHC data [14], and these are just examples.

All of this is possible due to the development of new algorithms or novel usage of old ones and our increased computational resources and their exploitation with parallel computational frameworks like TensorFlow [15]. All of this leads to possible answers to the given questions. Machine learning is not only a research area; after decades of research, it applies to many areas of human interest. It is a mature toolkit with tools applicable to a wide range of problems [16, p. 8]. Moreover, it is up to the experts in a field where a particular problem resides to try this toolkit. This work aims to provide a basic understanding of machine learning and an example of the safe workflow solving a classification task in one of those fields using a few classical and a few more modern approaches so that others can start more quickly.

1 MACHINE LEARNING THEORY

What is machine learning? One way we could see it is a field with an essential goal of understanding and building methods that can "learn" from given data to make predictions on previously unseen data [17, p. 1].

This view leads to another point of view, where we can see machine learning as a new programming paradigm, a new way of communicating with computers when we do not just "tell" a computer what to do by "giving" it a sequence of instructions to follow strictly, but rather program some framework which can exploit and leverage given examples and respond based on them [18].

It may be easily understandable, but it is a vague definition of machine learning. So we will proceed with a more formal one, as anyone can find in textbooks. Moreover, because the main focus of this work is classification tasks, we will later restrict ourselves to this area. However, we first revisit basic concepts to ensure the reader is familiar with them.

1.1 FUNDAMENTAL TERMINOLOGY

We use the following terms in this thesis, and it is crucial to understand them.

1.1.1 Prediction vs. inference

Supposing there is a relationship between variable Y and $\boldsymbol{X} = (X_1, X_2, \ldots, X_M)$ that can be expressed with a function f as $Y = f(\boldsymbol{X}) + \epsilon$, where ϵ is a noise. When our goal is to find an estimate of f (let us mark it $\hat{f}$) to predict Y based on $\boldsymbol{X}$, we see this as a prediction task where we are typically not concerned about the exact form of $\hat{f}$. In another way, if we are concerned about it, we can see it as an inference task where our goal could be to find which X_s affects Y most to filter out the rest. [16, p. 18] ML task is a prediction or inference task or both of them.

1.1.2 Supervised vs. unsupervised learning

Supervised learning is used where we have both independent and dependent variables in the data. We want to learn the relationship between these two to predict the dependent one for a new, previously unseen observation

of independent variable(s). Unsupervised learning is for data with the absent dependent variable(s), so the main goal is finding patterns and structures in data. Examples of supervised learning are regression and classification. An example of unsupervised learning is clustering, which we can see as unsupervised relative to classification [16, p. 26].

1.1.3 Regression vs. classification

When we deal with structured data, we characterize them as quantitative or qualitative (categorical). When a task is to predict a variable from a set of continuous values, we call it regression. When a set is discrete (in practice, have hundreds of elements at most), we call it classification [16, p. 28].

1.1.4 Binary vs. multi-class classification

Binary classification is a task to categorize observation into two distinct classes. Multi-class is not limited in the number of classes and could be reduced to multiple binary classifiers. The classification model usually produces how confident it is about the observation belonging to a particular class, i.e., gives a number between 0 and 1 that can be treated similarly to a predicted probability of belonging to the category [19, p. 556].

1.1.5 Algorithm vs. model

In this work, we work with an algorithm as a procedure described as a sequence of tasks or with a pseudo-code that can have multiple implementations in specific programming languages. The model is already trained (executed) implementation of the algorithm. It means the model is tight to specific implementation and inputted data, and we see it as an output of an algorithm.

1.1.6 Parameter vs. hyperparameter

Model parameters are set in the learning process. They are defined by the algorithm, given data, and sometimes by some (pseudo)random process, and the practitioner has no direct way to set them. Model hyperparameters are the opposite because they are set by the practitioner (or by another algorithm, we use specifically for this purpose) and can be viewed as parameters of the optimization algorithm we use in the learning process. We will give examples of hyperparameters when discussing selected ML methods in 2.1.

1.2 STATISTICAL LEARNING

Statistical learning is the basis of what we call machine learning nowadays. We briefly describe this field by defining a few terms and notations we will use in this and the following chapter. The primary source of this section (if not stated otherwise) is the Shalev textbook - Chapters 2, 3, and 5 [20].

1.2.1 Basic terminology

Domain set $\mathcal{X}$ - set of objects we want to label (i.e., categorize).

Feature vector - representation of an element from $\mathcal{X}$. It is a real vector in practical applications.

Label set $\mathcal{Y}$ - set of all possible categories or values we may predict. We discuss binary classification when the set contains two elements. Also called the response variable.

Training data $\mathcal{S}$ - finite sequence of unordered pairs from $\mathcal{X} \times \mathcal{Y}$ of size N. We also use validation and testing data in applications with the same form.

Prediction rule h - mapping $h : \mathcal{X} \mapsto \mathcal{Y}$, also called hypothesis or classifier. It is the outcome of our algorithm.

Hypothesis class $\mathcal{H}$ - set of hypotheses restricted from the search space of all possible hypotheses as there are reasons for doing this explained later.

Probability distribution $\mathcal{D}$ - arbitrary probability distribution over $\mathcal{X}$

Simple data-generation model - mapping $f : \mathcal{X} \mapsto \mathcal{Y}$, our labeling function for creating $\mathcal{S}$ by sampling a point from $\mathcal{X}$ according to $\mathcal{D}$ and then labeling it with f. We assume one such exists and can also have various assumptions about its "correctness." We are trying to figure out this function with our algorithm.

True error $L_{\mathcal{D},f}$ - the probability to draw a random instance $\boldsymbol{X} \in \mathcal{X}$ according to $\mathcal{D}$, such that $h(\boldsymbol{X}) \neq f(\boldsymbol{X})$ called error of classifier. A more formal description is

$$L_{\mathcal{D},f}(h) := P_{\boldsymbol{X}\sim\mathcal{D}}[h(\boldsymbol{X}) \neq f(\boldsymbol{X})] := \mathcal{D}(\{\boldsymbol{X} : h(\boldsymbol{X}) \neq f(\boldsymbol{X})\}) \tag{1.1}$$

0-1 loss - loss function, an example of a true error for binary classification

$$l_{0\text{-}1}(h, (\boldsymbol{X}, Y)) := \begin{cases} 0 & \text{if } h(\boldsymbol{X}) = Y \\ 1 & \text{if } h(\boldsymbol{X}) \neq Y \end{cases} \tag{1.2}$$

Training error L_S - the error the classifier gives rise to over the training sam-

ple. Also called *empirical error* or empirical risk.

$$L_S(h) := \frac{|\{k \in \{1, .., N\} : h(\boldsymbol{X}_k) \neq Y_k\}|}{N} \tag{1.3}$$

Empirical Risk Minimization (ERM) - learning paradigm (or rule) with the goal to find prediction rule h that minimizes $L_s(h)$.

ERM$_\mathcal{H}$ learner - learner applying ERM rule over $\mathcal{H}$. Also called a predictor.

Why do we need to restrict our search space? One of the reasons is explained in the next section, but at least the same importance is this one - we need to avoid *overfitting*. Overfitting is when our ERM rule finds an excellent or perfect hypothesis with zero training error $L_S(h)$ but with a high true error. An example of such a case is when we have a probability distribution that can be drawn like in figure 1.1 with uniformly distributed instances labeled by f as **1** or **0**.

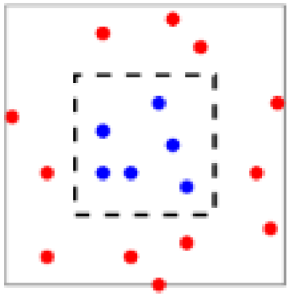


Figure 1.1: Uniform distribution of two classes [20, p. 36].

1 belongs to the first class and can be found inside the dashed square, and **0** belongs to the second class and can be found between the edges of dashed and thick squares. Then when we consider a hypothesis

$$h_S(\boldsymbol{X}) = \begin{cases} Y_k & \text{if } \exists k \in [N] s.t. \boldsymbol{X}_k = \boldsymbol{X} \\ 0 & \text{otherwise.} \end{cases} \tag{1.4}$$

we achieve $L_S(h_S) = 0$ no matter what sample we take, and that means this hypothesis could be chosen by an ERM algorithm. However, the true error is 1/2, as this is the classifier that predicts **1** only on a finite number of instances [20, p. 36].

1.2.2 PAC learnability

After defining basic terms and explaining overfitting, we can continue with the fundamental definition in statistical learning - Probably Approximately Correct (PAC) learnability.

Definition 1 (PAC learnability)**.** A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $N_{\mathcal{H}} : (0,1)^2 \mapsto \mathbb{N}$ and a learning algorithm with the following property: For every $\epsilon, \delta \in (0,1)$, for every $\mathcal{D}$ over $\mathcal{X}$, for every $f : \mathcal{X} \mapsto \{0,1\}$, if $\exists h^* \in \mathcal{H}$ such that $L_{(\mathcal{D},f)}(h^*) = 0$, then when running an algorithm on $N \geq N_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with the probability of at least $1 - \delta$,

$$L_{(\mathcal{D},f)}(h) \leq \epsilon \tag{1.5}$$

Parameter ϵ tells how far our hypothesis h is from optimal one h^* or how *approximately correct* it is. Parameter δ tells how likely the hypothesis will meet the accuracy requirement or how *probable* it is. We allow this lack of "precision" or "optimality" as our training set is finite, and even if it does faithfully represent distribution $\mathcal{D}$, it cannot reflect all fine details of $\mathcal{D}$ [20, p. 43].

An implicit assumption of the PAC learnability is the realizability assumption that there exists $h^* \in \mathcal{H}$ such that $L_{(\mathcal{D},f)}(h^*) = 0$ [20, p. 38]. That is only sometimes the case in practice, and a way how to deal with it is agnostic PAC learnability [20, p. 44].

Its definition is identical with the difference that we release the realizability assumption, as it is more natural to assume that there is a noise or some form of contradiction in our data. However, we must leave the previous definition of the distribution $\mathcal{D}$ and labeling function f. Let us redefine $\mathcal{D}$ as a joint distribution over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is still our domain set, and $\mathcal{Y}$ is still our set of labels. This definition allows us to have differently labeled equal elements from $\mathcal{X}$. First, we change the definition of the true error to

$$L_{\mathcal{D}}(h) := P_{(\boldsymbol{X},Y)\sim\mathcal{D}}[h(\boldsymbol{X}) \neq Y] := \mathcal{D}(\{(\boldsymbol{X}, Y) : h(\boldsymbol{X}) \neq Y\}) \tag{1.6}$$

Training error remains the same. We can follow the precise definition of agnostic PAC learnability.

Definition 2 (agnostic PAC learnability)**.** A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $N_{\mathcal{H}} : (0,1)^2 \mapsto \mathbb{N}$ and a learning algorithm with the following property: For every $\epsilon, \delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, when running an algorithm on $N \geq N_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$, the algorithm returns a hypothesis h such that, with the probability of at least $1 - \delta$,

$$L_{(\mathcal{D})}(h) \leq \min_{h' \in \mathcal{H}} L_{(\mathcal{D})}(h') + \epsilon$$

The definition was taken from [20, p. 46] but can be also found in [21, p. 276] or [22, p. 24]

1.2.3 No-free-launch theorem

There are a few essential theorems in statistical learning and, therefore, in ML, which tell us the theoretical limits of learning algorithms, so we know

what we can expect when applying ML in concrete scenarios. We have chosen the following one as it underpins the importance of the work provided in the practical part of the thesis.

Theorem 1 (No-free-lunch)**.** *Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain $\mathcal{X}$. Let N be any number smaller than $|\mathcal{X}|/2$, representing a training set size. Then there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ such that:*

1. *There exists a function $f : \mathcal{X} \mapsto \{0,1\}$ with $L_{\mathcal{D}}(f) = 0$*
2. *With probability of at least $\frac{1}{7}$ over the choice of $S \sim \mathcal{D}^N$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

We stated the theorem as in [20, p. 61], where you can also read the proof. An alternative formulation is available in [23, p. 76]. The critical fact is that for every algorithm, data and a related problem exist for which the algorithm is not sufficient or suitable. It is the reason why we must make a comparison of multiple algorithms. To select these algorithms, we need some prior knowledge about probability distribution $\mathcal{D}$ because when we deal with $\mathcal{X}$ of infinite size (as is usually the case) and try to beat it by the number of tested hypothesis, we will not succeed as states following corollary.

Corollary 1.1. *Let $\mathcal{X}$ be an infinite domain set, and let $\mathcal{H}$ be the set of all functions from $\mathcal{X}$ to $\{0,1\}$. Then, $\mathcal{H}$ is not PAC learnable.*

Proof can be found in [20, p. 64].

1.2.4 Bias-complexity tradeoff

The previous corollary leads to the need to restrict our hypothesis class $\mathcal{H}$ so that we avoid failing when learning our task but still include a hypothesis that has no error at all or at least the smallest error achievable (depending on the PAC setting).

This tradeoff is called **bias-complexity tradeoff** and can be described by the decomposition of the error of an $\text{ERM}_{\mathcal{H}}$ predictor into two components

$$L_D(h_S) = \epsilon_{\text{app}} + e_{\text{est}} \tag{1.7}$$

where ϵ_{app} is **the approximation error** - the minimum risk achievable by $\text{ERM}_{\mathcal{H}}$ predictor. This error follows from restricting our class $\mathcal{H}$ and is also called *inductive bias*. The second term e_{est} is **the estimation error** - the difference between the approximation error and the error achieved by the $\text{ERM}_{\mathcal{H}}$ predictor. This error is the result of the minimization of the training error instead of the true error, which means our predictor is only an estimate of the predictor minimizing the true error. As was already said, a rich $\mathcal{H}$ might lead to *overfitting*, represented by a high value of estimation error. On the other hand, choosing small $\mathcal{H}$ might lead to a high value of the approximation error,

which is called *underfitting* [20, p. 65]. An underfitting can be immediately recognized by poor performance on the training sample, overfitting is more tricky than underfitting, and we have to apply some approaches to mitigate this issue, as we will see in the last chapter.

Here our journey into statistical learning ends, although there are other core concepts in the field as uniform convergence or VC-Dimension, which can be found in [20] or [24].

1.3 ROOTS IN STATISTICS AND INFORMATICS

In the last section of this chapter, we want to discuss what distinguishes machine learning from fields like statistics and informatics and where its place is relative to these fields.

The previous sneak-peak into statistical learning shows that machine learning is based on a rigorous statistical framework. Also, in the following chapters, we will see that many classical statistical algorithms are used nowadays and are called ML methods. Moreover, as said in [25, p. 2], part of machine learning is proving algorithms' guarantees, which is an informatics task.

Historical perspective is given in [26, p. 2], when they identify three main branches of research, particularly in classification, as *statistical*, *machine learning*, and *neural networks*. Although all of them emphasize different issues, they have three common objectives:

- to surpass a human decision maker, at least in consistency and explicitness
- develop approaches general as much as possible with the ability to solve a wide variety of problems
- deploy solutions to a real environment

On the other hand, when we talk about applied machine learning, we get to the field focused on programming, the most efficient implementation of algorithms, and overall usability in a business domain. We can use Burger's [27] and Witten's [28] books as examples of such focus. This area is more software engineering than any other.

SUMMARY

We presented the basic terminology of statistical learning with one of the goals to state the no-free-launch theorem that showed that it is impossible to have *one algorithm rule them all*. We need to find the best one based on our prior knowledge of distribution $\mathcal{D}$ for every different problem. In this process, we should vary between estimation and approximation error.

2 BINARY CLASSIFICATION

We have defined a few terms related to classification in the previous chapter. This chapter summarizes it and defines what a binary classification task means. Then we describe the algorithms we will apply to solve our problem and the metrics we will use to evaluate and compare trained models.

Our task is a binary classification, a supervised ML task. The goal is to establish a rule which we can use to classify a previously unseen observation into two distinct classes. In other words, find a hypothesis h^* that can reasonably approximate labeling function f. We have multiple variables as input and want one output variable. The intermediate result of an algorithm should be a value of confidence in the interval $[0, 1]$ that given observation can be classified as one of the classes (we usually choose the "positive" class to be this one) and then based on the threshold we do final categorization.

2.1 SELECTED ALGORITHMS

There are many classification algorithms in the world. Wikipedia lists 86 pages in the category Classification algorithms. Multipurpose R package *caret* contains 236 available models. We describe only selection often used in papers or real use cases for a mutual comparison. At least one of these algorithms often has enough "performance," as we can see in papers focused on comparison by Uddin et al. [29] or Yuvali et al. [30]. Our description is high-level, so rather be seen as a description of methods than concrete algorithms.

2.1.1 Binary logistic regression

We assume a reader knows linear regression as a way to fit a line (hyperplane in general) through data points to model the relationship between explanatory variable $\boldsymbol{X}_k \in \mathcal{X}$, $k = 1, ..., N$, N is the number of observations, and response $Y_k \in \mathcal{Y}$. To remind, with one-dimensional X_k, response function is

$$linear(x, w, b) = wx + b = E(Y|X_k = x) \tag{2.1}$$

When $\boldsymbol{X}_k$ is multidimensional (dim($\boldsymbol{X}_k$)=M), it is

$$linear(\boldsymbol{X}_k, \boldsymbol{W}) = W_0 + \sum_{s=1}^{M} W_s X_{k,s} \tag{2.2}$$

Such hyperplane "height" ranges from negative infinity to positive infinity. In binary classification, the only sensible output is 1 or 0 (or two elements set in general). We can use the sigmoid function with its definition in figure 2.1

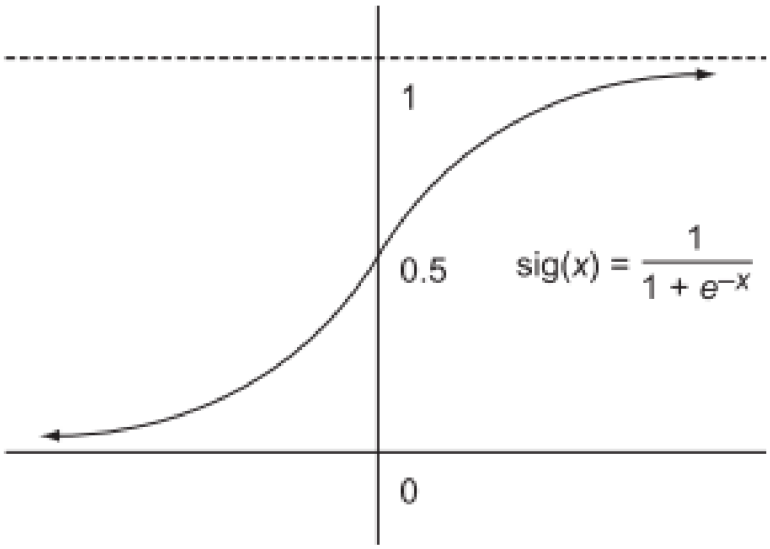


Figure 2.1: Graph of the sigmoid function [31, p. 101].

Sigmoid maps the explanatory variable (real feature vector) to interval $[0, 1]$ and relatively quickly converges to 0 when $\boldsymbol{X}_k$ goes to negative "infinity" or 1 when $\boldsymbol{X}_k$ goes to positive "infinity". Our model becomes $sig(linear(\boldsymbol{X}_k, \boldsymbol{W}))$ in binary logistic regression. Best-fit parameters produce a model with a linear separation between the two classes [31, p. 101].

The sigmoid function is derived based on the assumption that the logarithm of odds of the event of the occurrence of **1** class in k-th observation can be modeled linearly [32, p. 439], i.e.,

$$log(\frac{\pi_k}{1-\pi_k}) = W_0 + \sum_{s=1}^{M} W_s X_{k,s} \tag{2.3}$$

where $\pi_k = P(Y_k = 1)$ and is called fitted probability. We call this log function $logit$ [33, p. 68]. After some algebraic manipulation, we can get the sigmoid function and see

$$\pi_k = sig(linear(\boldsymbol{X}_k, \boldsymbol{W})) \tag{2.4}$$

The loss function (in condensed form and with penalization), which we use in binary logistic regression, is following

$$l(\mathcal{X}, \mathcal{Y}, \boldsymbol{W}) = \sum_{k=1}^{N} -Y_k \log(h(\boldsymbol{X}_k)) - (1 - Y_k) \log(1 - h(\boldsymbol{X}_k)) + \lambda * r(\boldsymbol{W}) \tag{2.5}$$

Function h is our hypothesis from the hypothesis class [20, p. 127]

$$\mathcal{H} = \{h : \boldsymbol{X}_k \mapsto sig(linear(\boldsymbol{X}_k, \boldsymbol{W})) = \pi_k, \boldsymbol{W} \in \mathbb{R}^N\} \tag{2.6}$$

Function r is the Ridge penalization (also called ℓ_2) with the purpose of reducing the size of coefficients

$$r(\boldsymbol{W}) = \sum_{s=1}^{M} W_s^2 \tag{2.7}$$

Constant λ specifies regularization strength and is one of the logistic regression hyperparameters. More details about regularization in logistic regression are in [34].

In other words, l = - log-likelihood + λ * regularization term. Function l is convex, so pleasant for various optimization algorithms, as we want to minimize it [31, p. 101]. The type of such algorithm (solver) is our choice, so it is another logistic regression hyperparameter. Comparison of solvers for logistic regression are in [35].

2.1.2 Decision tree

The decision tree technique is one of the most intuitive methods. The ultimate goal of the method (for binary classification) is to divide the dataset into two classes based on simple rules in each node, as you can see in figure 2.2. A tree is, in many algorithms, constructed by a top-down approach where the algorithm selects a feature and a separation condition in each node using a greedy approach when the algorithm tries to find the most ”pure” division. This condition can be a comparison against a selected value (in case of continuous feature) or equality with a mode (most frequent value) of feature values (in case of categorical feature) [32, p. 313]. Other division criteria can be found in [33, p. 71].

Ultimately, we get g leaves, each representing a group of observations. We can label this group as belonging to one of our classes with fitted success probability (for **1** class)

$$\pi_k = \frac{\sum_{s=1}^{M_g} \hat{Y}_{g,s}}{M_g} \tag{2.8}$$

where M_g is the size of a particular group [33, p. 71] and k is an observation.

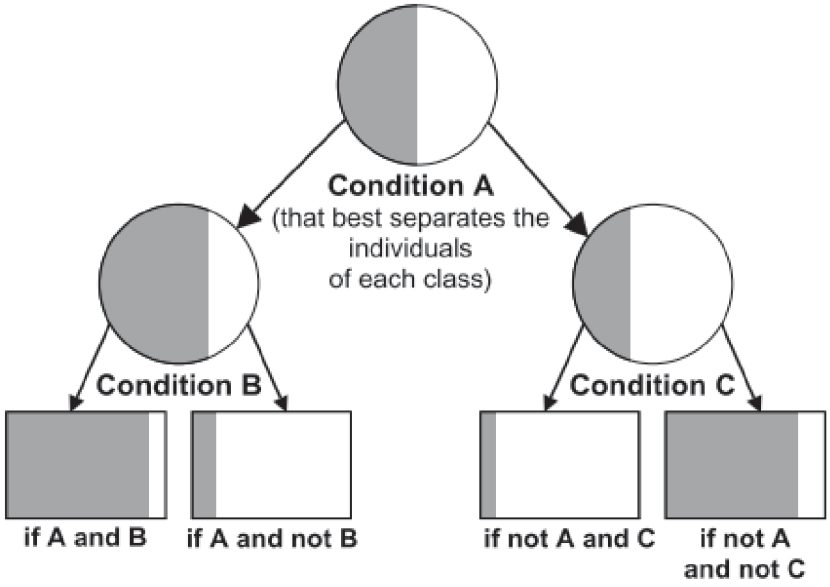


Figure 2.2: Illustration of decision tree model with its conditions [32, p. 314].

We should specify stopping criteria for g; preferably, we want it to be much less than the number of observations. It can otherwise lead to a complex tree, leading to higher odds of overfitting, as we have seen in the first chapter with rich hypothesis classes. Examples of stopping criteria are the maximal depth of the tree or the minimal proportion of samples that must be present in the node; otherwise, we set it as leave. These criteria are hyperparameters of the decision tree, as their values are up to us. The basic algorithm for inducing a decision tree can be found in [36, p. 333]

2.1.3 Random forest

Before explaining the Random Forest (RF) method, we have to introduce bootstrapping as a model training and selection method and the idea of bagging a decision tree, which is the RF method built on.

Bootstrapping is a resampling method often used where the dataset is small or we want to reduce the variance of error estimates or even obtain it. Because it is not always easily feasible with other sampling methods used for training and validation such as train-and-test or cross-validation [26, p. 108]. The idea is to sample with replacement from the training set to get set with the same size. We will obtain about 2/3 of the observations, some repeated multiple times. We use these for training and the rest for validation (estimating error). We do this B times and then calculate the mean and variance of obtained errors [26, p. 109].

More important right now is this idea applied to decision trees. Decision trees suffer from high variance. It means that when we split the training

set into two parts at random and train a separate model (decision tree) with each, we get results that could be quite different. However, by averaging a set of independent observations, we reduce variance. The idea is to train B models on separate training sets and then average results.

$$h_{avg}(\boldsymbol{X}_k) = \frac{1}{B}\sum_{b=1}^{B} h(\boldsymbol{X}_k) \quad k \in \{1, .., N\}, N = \text{ number of observations} \tag{2.9}$$

Because obtaining separate and still large enough B training sets is impractical, we use bootstrapping on a single training set to get and train trees with them. This approach is called *bagging* [16, p. 317].

The RF method is built on bagging with one important tweak. Instead of using all M predictors in splitting a node in a tree, a random sample of m predictors is chosen. We typically choose $m \approx \sqrt{M}$. The rationale is to *decorrelate* the trees because if there is a strong predictor, it will often be chosen for split at the top node. Furthermore, we get similar trees because the top split influences all other splits down a tree, resulting in highly correlated predictions. However, as was already said, we reduce variance by averaging these predictions assuming they are independent. Bagging ruins this assumption, and this is a way how to deal with it. RF method could also reduce test error [16, p. 320]. When training RF model, we have several hyperparameters to choose from. Except for the ones that apply to one decision tree as maximal depth or minimal portion of samples in node, we have to choose the number of estimators, i.e., trees used.

2.1.4 Support vector machines

Support Vector Machines (SVM) method is built on separating the feature space by hyperplane and then classifying based on the sign of such separation (a point is below or above the hyperplane), so we label classes either -1 or 1. It means SVM is a natural binary classifier, although it could be extended multi-class as you can find in [16, p. 355].

When classes are linearly separable, there is an infinite number of such hyperplanes. So the idea of *margin* is added to choose the ”best” hyperplane. This approach with linear separability assumption is called *maximal margin classifier* (MMC) [16, p. 342].

When classes are not linearly separable, we lessen the requirements by allowing some points to be inside the margin or on the other side of the hyperplane. We quantify such violations and add a limit to them. We call this method *soft margin classifier* or *support vector classifier* (SVC). An example of SVC is in figure 2.3. The term *support vector* comes from the realization that only points violating the rules of MMC affect the output of SVC, as points on the correct side of the hyperplane outside of the margins are not penalized, and it is not important how far away they are. A more detailed description of SVC constraints can be found in [16, p. 346].

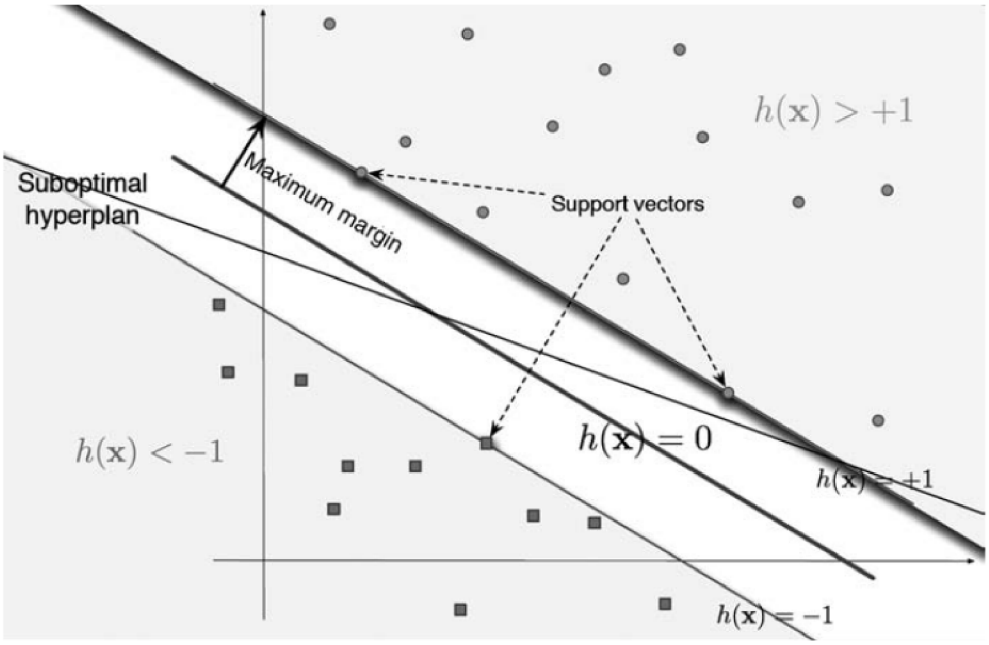


Figure 2.3: Illustration of support vector classifier or Support Vector Machines with linear kernel in another point of view [32, p. 503].

A natural extension of SVC is to add non-linear interactions of features such as products. The separation task is still linear in this extended feature space, although generally not in the original space. Nevertheless, this approach is only sometimes computationally feasible as there could be many interactions to try. The trick how to more easily generalize to the non-linear case is called *kernel trick*. As we deal with inner products when computing the distance between point and boundary [16, p. 351], by tweaking it, we can bring non-linearity. We construct a function of the inner product called *kernel*, and the following is an example called *polynomial kernel* of degree d (where $M = dim(\boldsymbol{X}_k) = dim(\boldsymbol{X}_l)$ and $k, l \in \{1, .., N\}$, $N =$ number of observations)

$$K(\boldsymbol{X}_k, \boldsymbol{X}_l) = (1 + \sum_{s=1}^{M} X_{k,s} X_{l,s})^d \tag{2.10}$$

Such non-linear kernels are the essence of SVM, and it is what distinguishes SVM from SVC [16, p. 352]. Other examples of kernels are in [32, p. 505]. The practitioner selects kernel, which means it is one of SVM hyperparameters.

2.1.5 K-nearest neighbors

K-Nearest Neighbors (KNN) classification differs from the previously discussed methods because it is a member of transductive. These methods

are ”memory-based” as they do classification based on previously classified individuals in opposite to methods that construct a model to do so [32, p. 302]. This idea is illustrated in figure 2.4.

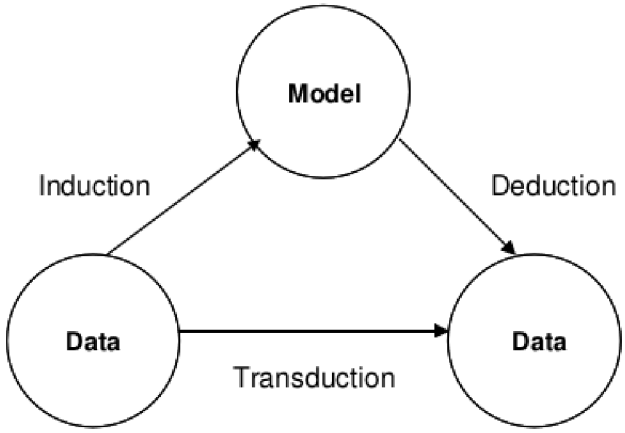


Figure 2.4: Transductive and inductive methods [32, p. 503].

The method's name tells us we classify based on k nearest neighbors. What do we mean by nearest? To define this, we have to choose a metric to compare distance, usually induced by p-norm with $p = \{1, 2\}$ (where $M = dim(\boldsymbol{X}_k) = dim(\boldsymbol{X}_l)$ and $k, l \in \{1, .., N\}$, $N =$ number of observations).

$$\rho_p(\boldsymbol{X}_k, \boldsymbol{X}_l) = (\sum_{s=1}^{M} |X_{k,s} - X_{l,s}|^p)^{\frac{1}{p}} \tag{2.11}$$

When choosing a metric, it is good to ask and answer the question, ”What does it mean when the distance between points double?” [28, p. 129]. We then order all neighbors of point $\boldsymbol{X}_k$ based on the chosen metric and return the *mode* among the K nearest [20, p. 259]. Because the practitioner must choose p and K, these are examples of KNN hyperparameters.

This approach's time complexity is proportional to the number of training observations, and we are just about classifying one test observation. There are ways to deal with this, like the kD-tree method storing a set of training points in the binary tree. The details are in [28, p. 130].

2.1.6 Multilayer perceptron

Many types of neural networks are used nowadays, such as convolutional, recurrent, or graph neural networks. In this work, we will describe and use one called *multilayer feed-forward network*. Other names are *dense, deep neural network* or Multi-Layer Perceptron (MLP). These are not exactly synonyms, but the differences are slight and out of our scope.

We describe a neural network by a directed acyclic graph $G = (V, E)$ with edges weighted by the function $\omega : E \mapsto \mathbb{R}$. We call nodes the graph neurons and model them with a function, $\sigma : \mathbb{R} \mapsto \mathbb{R}$. This function is called *activation* and examples used in practice are sigmoid function $sigma(z) = \frac{1}{1+\exp(-z)}$ or ReLU function $\sigma(z) = \max(z, 0)$. It is an example of a hyperparameter for a neural network. We link the output of one neuron to another as input, as illustrated in figure 2.5.

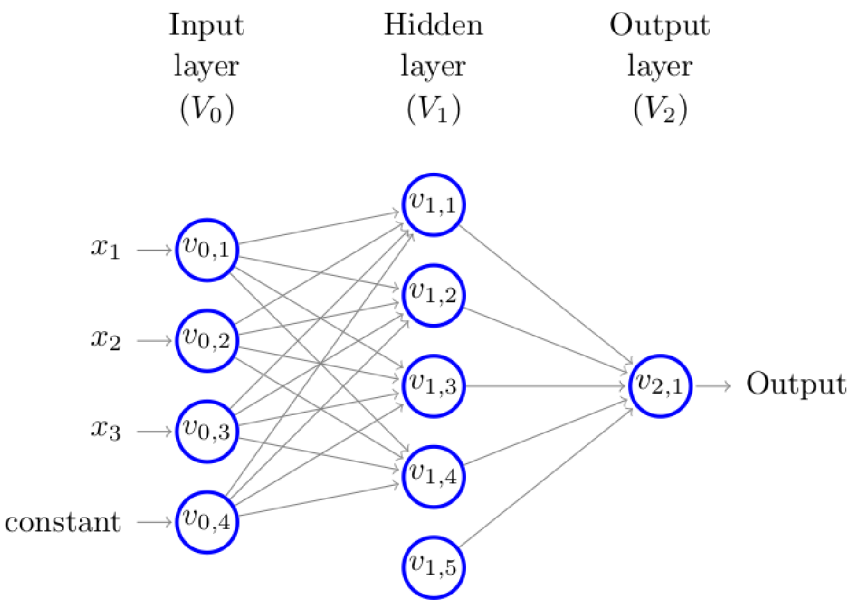


Figure 2.5: Graphical representation of the graph of neural network with one hidden layer V_1 [20, p. 270].

Input for the neurons in the first layer V_0 is an observation from the training set; other neurons get their input as a weighted sum (according to ω) of outputs of connected neurons, with a constant added in both cases. Each input always goes through an activation function before outputting to the next layer [20, p. 269].

We can compose our hypothesis class as

$$\mathcal{H} = \{h_{V,E,\sigma,\omega} : \omega \text{ is a mapping from } E \text{ to } \mathbb{R}\} \tag{2.12}$$

where $h_{V,E,\sigma,\omega} : \mathbb{R}^{|V_0-1|} \mapsto \mathbb{R}^{|V_T|}$ (where T is the number of layers) is a mapping of ω with fixed tripled (V, E, σ) that is called *architecture* of the network [20, p. 270].

Characteristics of neural networks that are seen as advantages are [32, p. 499]:

- ability to allow non-linear relations and complex interactions between variables

- no assumptions about variables following particular probability distribution
- applicability to a wide range of problems (most of the examples in the introduction of this work)

Some of the disadvantages are [32, p. 500]:

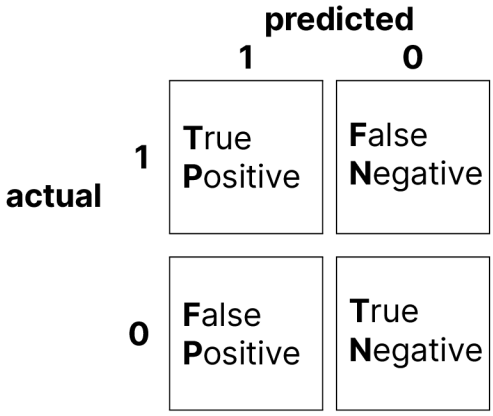
- convergence towards the best global solution is not guaranteed
- risk of overfitting if the number of observations is too small concerning the number of neurons
- requirement of an *a posteriori* analysis to discover the impact of different input variables on the output

These characteristics with other advanced topics are also discussed in Chapter 20 in [20].

2.2 SELECTED METRICS

We use a loss function in the process of training a model. The loss function is chosen based on the algorithm and specific mathematical properties. However, they are only sometimes best for comparison of different models, mainly if, in addition, models are based on different algorithms. Then we have to use different tools for comparison. General ones are described in this section, but sometimes we even need to define our domain or problem-specific metrics.

In supervised learning in the development stage, we always know the correct labels (or at least we have to trust what we have), so we only need to compare the output of our model with the values from the dataset, whether the model was correct or not. As models usually output fitted probability in $[0, 1]$ interval, we need to choose *threshold* value to map this interval to $\{0, 1\}$. The threshold value is in most implementations of metric evaluation implemented as 0.5, so when the fitted probability is at least 0.5, we classify the observation as a positive class; if less, then as a negative class. Threshold helps us represent our results as *confusion matrix* [31] [37, p. 209]. We use **1** for our positive class and **0** for our negative class.



The apparent goal is to have a model outputting only True Positive (TP) and True Negative (TN). Nevertheless, as we have seen in the first chapter, we usually assume something like this is not possible.

2.2.1 Accuracy

When we want to summarize the confusion matrix into one statistic, the most straightforward way is to calculate a ratio of correctly predicted outcomes to all outcomes. This ratio is called *accuracy* [31, p. 93].

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.13}$$

The usefulness of this metric decreases with the increasing imbalance in the ratio of the number of individual classes in the data. For example, when the ratio of presence of **1** and **0** class in data would be 99:1, we can predict **1** all the time and achieve 0.99 *accuracy*. So we have to come up with more sophisticated metrics.

2.2.2 Precision and recall

To answer how likely a positive prediction is correct compared to all positive predictions, we use a metric called *precision* [31, p. 94] [37, p. 781]. Its equation is

$$precision = \frac{TP}{TP + FP} \tag{2.14}$$

To answer how likely a positive prediction is correct compared to all true positives, we use a metric called *recall* [31, p. 94] [37, p. 781]. Synonym for *recall* is True Positive Rate (TPR). Its equation is

$$recall = \frac{TP}{TP + FN} \tag{2.15}$$

Trying to achieve high precision is proper when we want our model to output positive prediction only if it is sure. An example of doing such is if we would like to have a model for books recommendation. Because we have limited time, many books are on the market, and we want to avoid going through something uninteresting or unusable. We want our model only to give recommendations where is a high likelihood that we will like a book.

On the other hand, achieving high recall is useful when we want our model to output positive prediction anytime there is some likelihood of such a case. An example can be a medical diagnosis, as it is better to diagnose a patient falsely as ill, even if we have to double-check and sometimes unnecessarily scare them, but still better than leaving someone untreated.

Another metric we use to compute a more complex one is False Positive Rate (FPR). Its equation is

$$FPR = \frac{FP}{FP + TN} \tag{2.16}$$

2.2.3 F1 score

A compromise between precision - P and recall - R is *F-measure*, the weighted harmonic mean of these two metrics.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} \quad \text{where } \alpha \in [0, 1] \tag{2.17}$$

With $\alpha = 1/2$, we got the balanced F-measure, commonly denoted as F_1 or F1 score in the ML community.

$$F_1 = \frac{2PR}{P + R} \tag{2.18}$$

Because it is high only if both precision and recall are high, it is useful when we want a balanced model [38, p. 1147].

2.2.4 ROC curve and AUC

In the introduction to this chapter, we talk about a binary classifier first producing a number and then categorizing inputted observation based on whether the number is greater than or less than a specified threshold.

All previous metrics measure performance of the classifier with a particular threshold set. We can continually adjust the threshold and plot TPR against FPR. We get a curve called Receiver Operating Characteristic curve (ROC). The baseline is given by an algorithm that would randomly guess (making a 50/50 guess).

If there is no clear distinction between the performance of models like in the right side of figure 2.6, we use Area Under the ROC Curve (AUC) as a quantitative measure of comparison. It means AUC is a numerical integral of ROC curve. An excellent classifier has AUC value higher than 0.9 [31, p. 95].

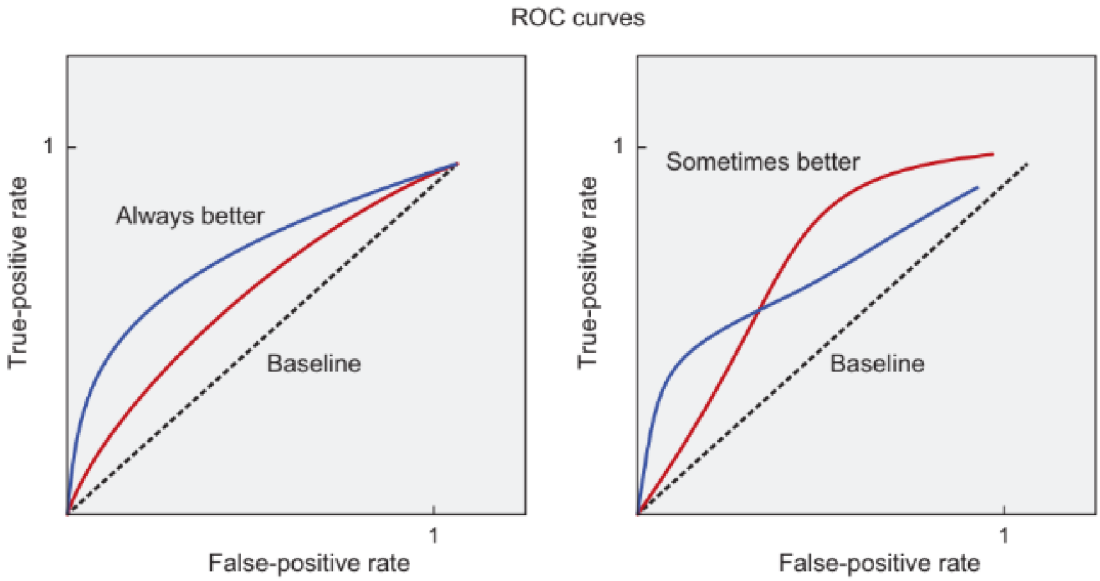


Figure 2.6: Examples of comparison of two algorithms with ROC [31, p. 95].

SUMMARY

We have gone through the selection of classification methods we will use in the next chapter with an intuitive explanation of the idea behind it and the definition of a loss function used for training a model with a given method. We also explained the most used metrics used for the evaluation of outputs of training.

3 MACHINE LEARNING WORKFLOW

In this chapter, the practical part of the thesis, we will roughly follow a model selection workflow proposed and presented by Cassie Kozyrkov, head of decision intelligence at Google Cloud, in her internal course to Google employees provided for free at her YouTube channel [39] [40]. An empirical comparison of algorithms on a given data is the main component of this workflow, just wrapped with preceding steps like the definition of requirements in means of our metrics (how well we expect a model to behave), preprocessing and splitting of a dataset, and following steps like presenting the results and deploying a chosen model, i.e., using it in a real environment.

3.1 PREPARATION

The zeroth step is to think whether we need a ML for solving a given task (or even if we can solve it with ML). Here we go from a different end as we want to apply selected algorithms and compare resulting models, so we have chosen a suitable task for our binary classification methods. That is opposite to what is usual in practice.

The first step is to set objectives, which means choosing our goal, a performance metric to quantify its model's ability to fulfill our goal, and the target performance score. This process involves choosing our action if the best model does not perform well enough. This step is crucial to make sound hypothesis testing on model performance, and the hypothesis is not postulated (bent over) based on the data but on the objective itself. When choosing a metric, we should also consider what mistakes are worse (as explained in examples in 2.2.2).

The second step is to get the data. The third is splitting data into training, validation, and testing parts. Then we can start with Exploratory Data Analysis (EDA), as a process to visualize training data and select predictors or engineer new ones (by combining original ones). We have done all these steps and describe our actions in the following subsections.

3.1.1 Task description

Our task is to predict whether an applicant for studies at our faculty has a chance to pass the first year successfully. Successfully means he/she will

get at least 35 ECTS credits at the end of the academic year. This task can be framed as a binary classification when the positive (1) class is getting at least 35 credits, and the negative (0) class is getting less than 35 credits.

3.1.2 Goal definition

We define two metrics for our case. The business metric will be *recall* (2.2.2, which we choose as we want to avoid false negatives, i.e., predicting an applicant as an unsuccessful student. We will leave a default threshold value ($= 0.5$) on the test dataset to obtain the recall value, as it is not clear how to optimize it without getting the classifier to predict only a positive class. The training metric will be AUC (2.2.4) for the sake of comparison of trained models when choosing optimal hyperparameters. The required performance is to score 0.9. The default action is not to use a model (in case we would have to decide) if the score is insufficient, so the null hypothesis is that the model scores 0.9, and the alternative hypothesis is that the model scores more than 0.9. We must postulate a null hypothesis before seeing the accuracy of models on train data or even exploring the training dataset.

3.1.3 Data description

We have obtained historical data about the admission process and the following study results of 3595 students who started their first year of studies between 2017 and 2021. We get information like points achieved in admissions, name of high school, year of high school graduation, number of credits and grades average obtained in the first year, the form of study, and study program name. We will inspect these features in more detail in our EDA in 3.1.6. These data were provided anonymized.

3.1.4 Tools selection

A critical step of every task is choosing (or building) the right tools to make it easier to complete. We have chosen Python and its libraries like pandas for data manipulation, preprocessing and descriptive statistics, matplotlib and seaborn for data visualization, scipy for statistics calculation, and scikit-learn for models training. Other libraries will be eventually mentioned along with their usage. Code (hopefully self-commented) is available as an attachment in the Jupyter Notebook format or HTML, which makes it easy to present the code with its intermediate results (even with images embedded) and its explanation. We will use this benefit and often skip less important details referencing the source code for more interested readers.

3.1.5 Data preprocessing

The dataset contains different names for the same or substitutive programs through academic years and free text input for a high school name. Therefore, we map high school names into eight categories based on the type of school, like "grammar school" or "business academy." We merged the same university programs with different names and then mapped all programs to eight categories like "languages" or "natural sciences." The explainability and richness of categories were compromised towards the smaller feature vector size, as each category adds one to it (when using one-hot encoding and not applying any dimensionality reduction techniques). Fewer features mean a less rich hypothesis class which generally means lower estimation error (discussed in 1.2.4). It is even more important with a limited number of observations.

We significantly changed the column with points achieved in the admission process. We divided all values by the maximum points attained in a given year and program (and multiplied by 100), respectively. This change makes them more comparable in terms of values. However, it is the only thing we can do about different admission conditions through the years and programs. It is necessary to mention that any other value normalization or standardization than the one mentioned was not done. This process's parameters should be obtained only from the training dataset to avoid test data leakage to the training stage. We also feature-engineered the rest of the variables to simplify our work as we simultaneously did this necessary process for the whole dataset. Nevertheless, these were just ordinary things like mapping binary features to $\{0, 1\}$.

In the end, we split the dataset to train and test parts. It is a different order than Cassie Kozyrkov proposes, but we needed the whole dataset to create practical and consistent classes of described features. The primary reason for Kozyrkov's order is that test data leakage can be introduced by exploring the testing set, as a person can direct training based on the obtained information. It means we could introduce bias by preprocessing data to such an extent without splitting it first.

The last completed academic year, 2021/2022, was chosen as the testing set (instead of random sampling). The main reason is that the task itself is about predicting the study success of applicants. The model can only be trained on the previous years. However, we are not interested in predicting the past, so we want to test against new data as possible, hence choosing the latest year available as the test set. This novelty is also the reason for using only the five last years available in the whole dataset instead of all available data (about twenty years of data are technically accessible). Consequently, we have dedicated about 20 % of our data for testing purposes, the usual amount to choose from. We did not create a validation set as we will apply K-fold cross-validation for hyper-parameters selection (more about in 3.2).

More details of preprocessing are in the source file *select_features_split* in the attachment.

3.1.6 Data exploration

As mentioned above, we will only explore the training dataset (academic years started in 2017-2020) to avoid test data leakage to the training process. Data are prepared in the format processible by our selected algorithms. We will start with basic descriptive statistics in figure 3.1.

	ROKY_OD_MATURITY	BODY_VAZENE	PREZENCNI	KRED_1_ROC	35_KREDITU_1_ROC	MISTNI
mean	4.34	76.28	0.63	34.59	0.58	0.46
std	7.64	17.85	0.48	26.1	0.49	0.5
min	0.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	63.0	0.0	0.0	0.0	0.0
50%	1.0	75.0	1.0	46.0	1.0	0.0
75%	5.0	93.0	1.0	59.0	1.0	1.0
max	39.0	100.0	1.0	100.0	1.0	1.0

Figure 3.1: Summary of our numerical and binary variables.

Before jumping to model training, we must ensure that features are uncorrelated. High correlation can produce models with poor generalization capacity because the model can learn that specific changes in the response variable are due to the first correlated variable when they are due to the second. We do not have this issue with our numerical variables, as shown in figure 3.2.

For quantifying an association between categorical (nominal) variables in our dataset, we used Cramér's V, which is based on Pearson's Ξ^2 statistic.

$$V = \sqrt{\frac{\chi^2/N}{min(C-1, R-1)}} \tag{3.1}$$

We computed χ^2 statistic from a contingency table where N is the total number of observations, C is the number of columns, and R is the number of rows. Figure 3.3 shows an overview of values for every categorical variable in the dataset (except the response variable). All contingency tables are in source file *visualize_explore* with other computation details like p-values.

Another problem could be a relatively low number of specific values in a categorical feature. We can conclude something about the effect of such a value on the response variable only with enough observation concerning other predictors. Moreover, a model is probably useless in this part of the

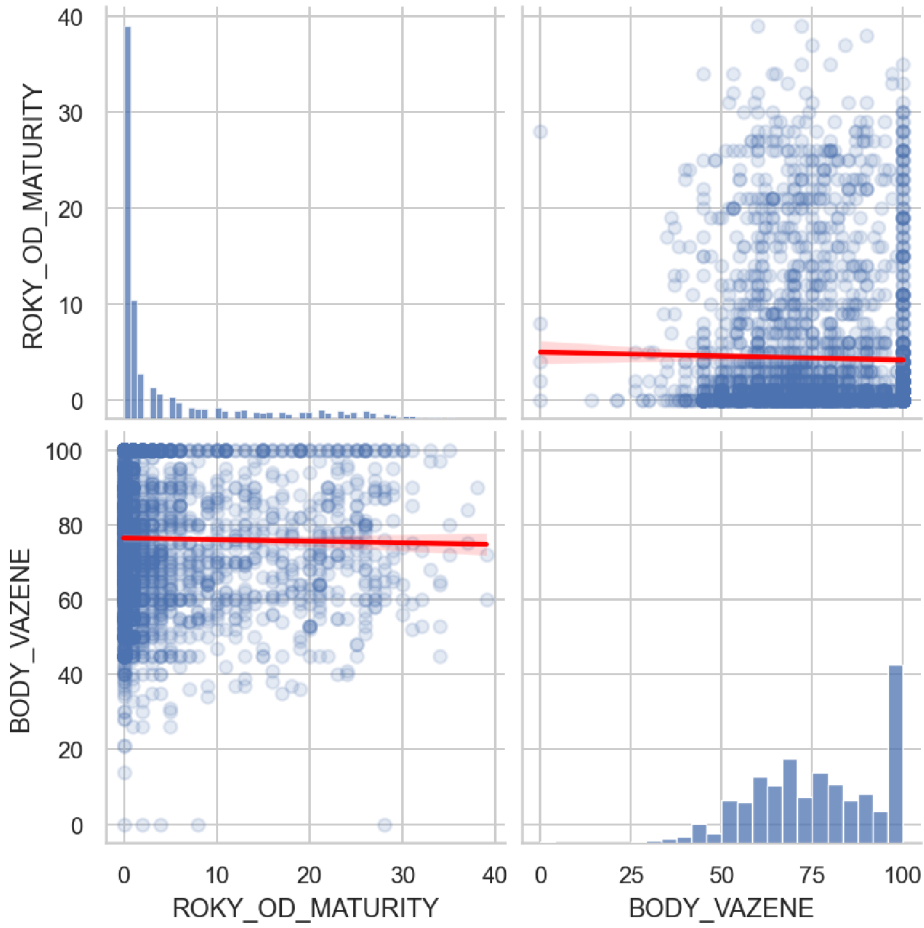


Figure 3.2: Pair plot of numerical features in our dataset. On the main diagonal are histograms, and on the anti-diagonal are scatter plots with linear regression lines. These lines' slope hints that the correlation of displayed variables is close to zero. Indeed, Pearson's correlation coefficient of years from high school (ROKY_OD_MATURITY) and weighted points (BODY_VAZENE) is about -0.0186.

feature space if the number of features is higher than the number of observations with this value. The lowest count through all categorical features has the high school of type gastronomy, hotel industry, and tourism (gastronomie, hotelnictví, ruch) with 52 occurrences. We solved the problem in advance in the preprocessing phase.

These were the main concerns, but we can continue. Exploring relationships between numerical variables in the context of a categorical variable or between numerical and categorical can also be valuable. In the following figures 3.4, 3.5, 3.6, 3.7, we present selected plots focused on categorical variables college field of study (VS_OBOR) and high school type (SS_OBOR). In the legend of these figures are the actual values of these categories. Furthermore, there is shown difficulty when one observation can acquire more

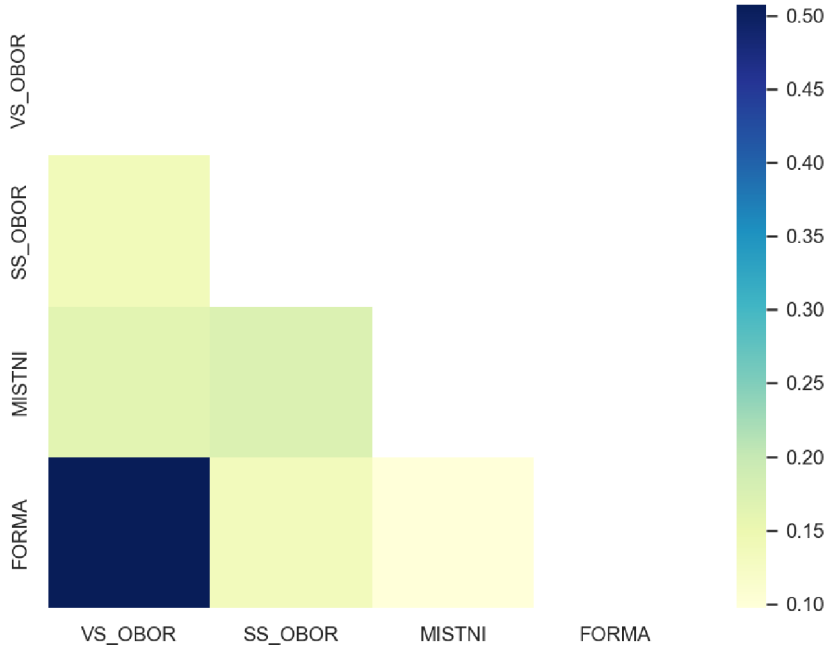


Figure 3.3: The plot shows a heatmap of Cramér's V values for our categorical features. We can see that college field of study (VS_OBOR) is correlated with a form of study (FORMA). This collinearity is not an issue as we are doing a predictive task (see 1.1).

than one value of a category variable (VS_OBOR). We dealt with it so that when a pedagogy student has two appropriations different in its field, it is something like a new category value. We even modified one-hot encoding so that it is not "one" anymore but "two" (same with approbations from the same field). Variable (SS_OBOR) poses another difficulty as there were 604 unique text values in the source dataset, many of which are not easily classifiable or represent a minor type of school, so there is a rest category (ostatní) which has about 19 % occurrence.

The last two categorical variables mean whether a student is from the region (MISTNI) and their form of study (FORMA). We define the region as Liberecký kraj except for most of the former okres Česká Lípa (as it is close to the university in Ústí nad Labem as well) with the addition of Mnichovo Hradiště and Mladá Boleslav with nearby villages. Types of study are two, full-time (P) or part-time (K). We visualize these two categorical variables as context for our numerical variables in figures 3.8 and 3.9.

We end our exploration here as it is not the main topic of our work. More details and visualizations are in the source file *visualize_explore*.

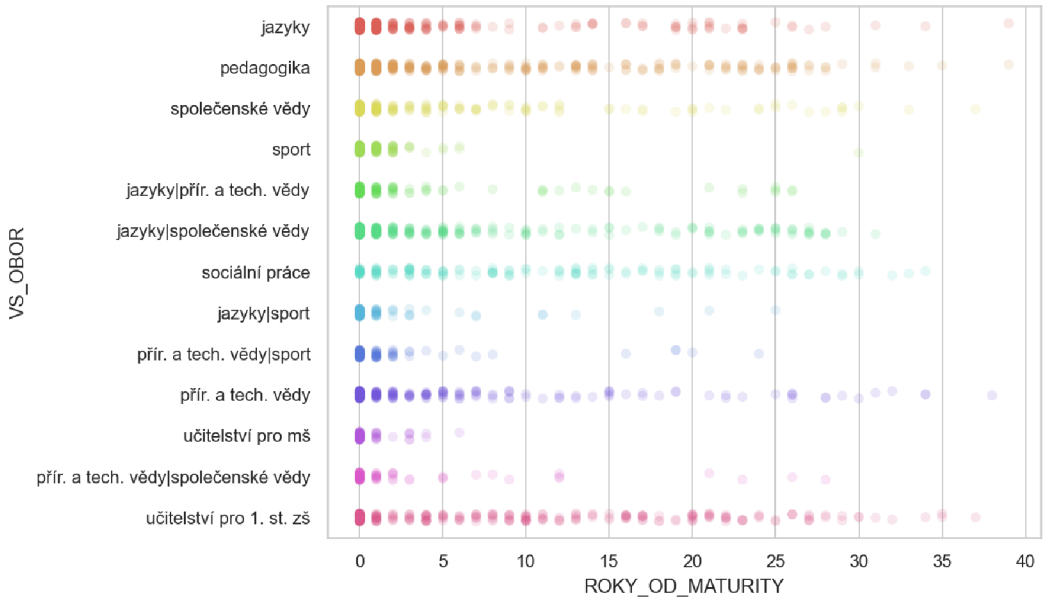


Figure 3.4: Strip plot of college field of study vs. years from high school. With the help of transparency, we can see that most students in the first year have less than five years from high school. Also, certain college fields have (almost) no older students, probably because of only full-time form.

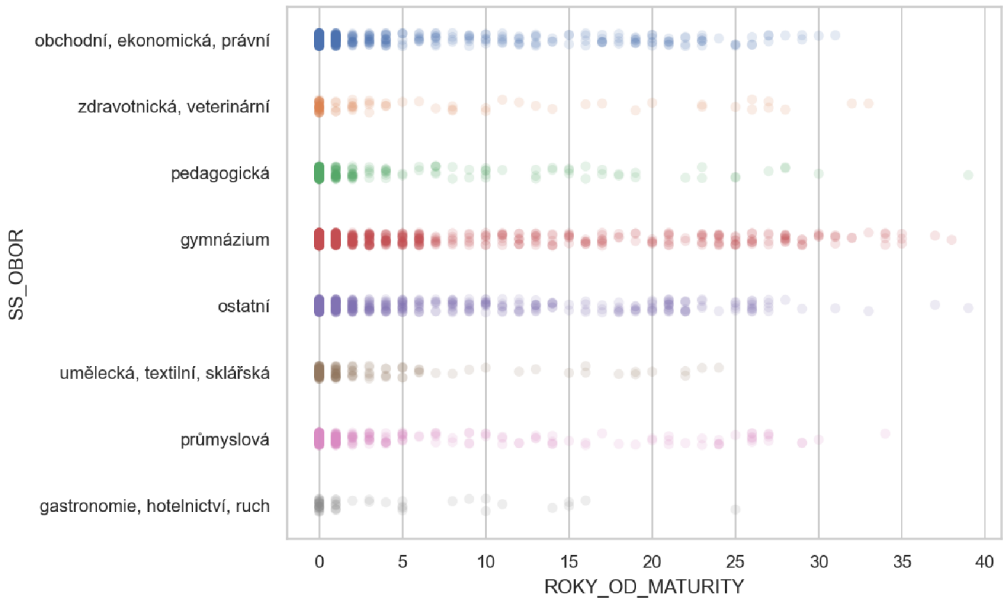


Figure 3.5: This strip plot shows that older students can have any school. The dominance of certain high school types is primarily given by their frequency, like grammar school (gymnázium) with 33 % occurrence.

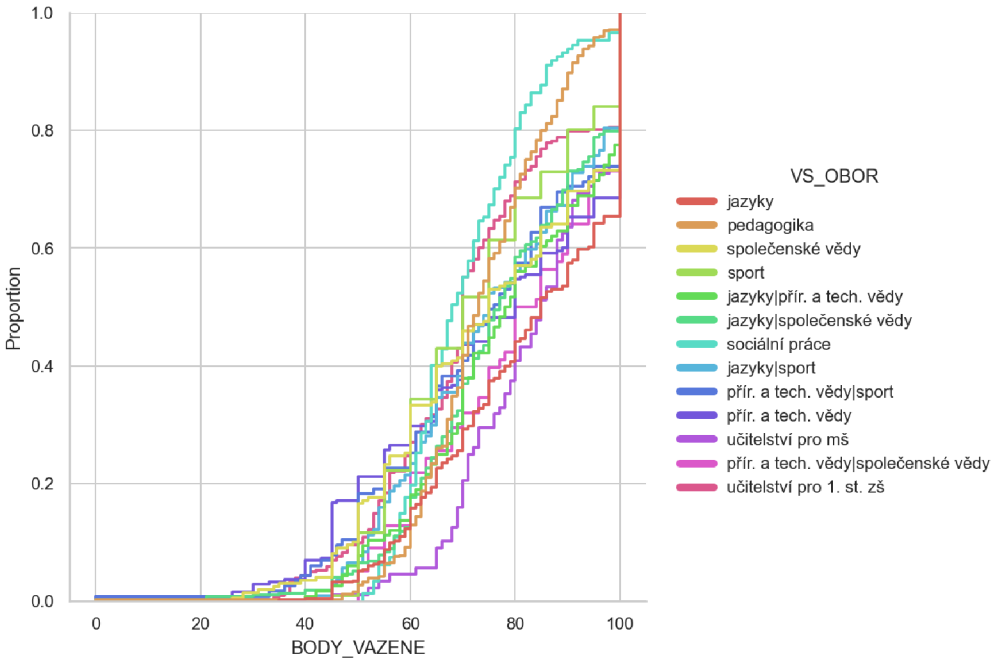


Figure 3.6: This empirical Cumulative Distribution Function (eCDF) shows that 80 % of applicants to field/program social work (sociální práce) has less than 80 admission points or that about 35 % of applicants to language (jazyky) programs have maximum points.

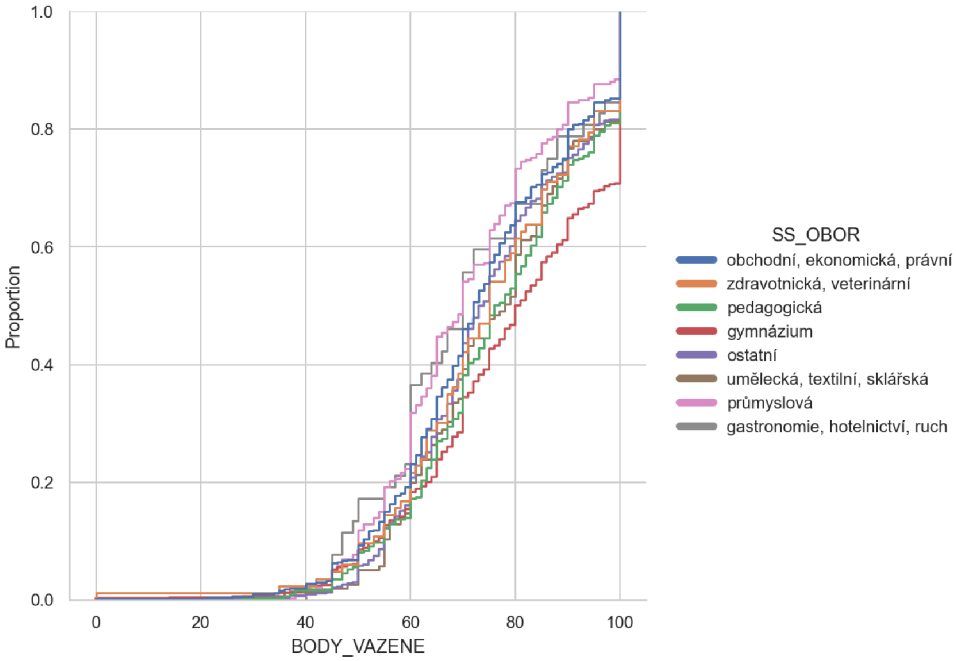


Figure 3.7: With the help of this eCDF, it is most visible that the best applicants in terms of distribution of admission points are from grammar school (gymnázium).

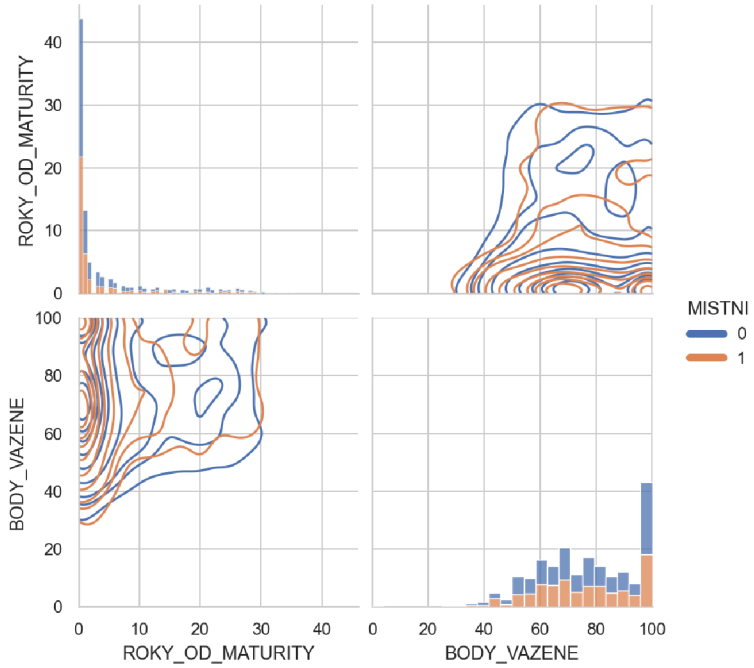


Figure 3.8: Pair plot shows that years from high school and admission points do not depend on student's region by both stacked histograms on diagonal that visualize univariate distributions and KDE plots on anti-diagonal that visualize bivariate distributions.

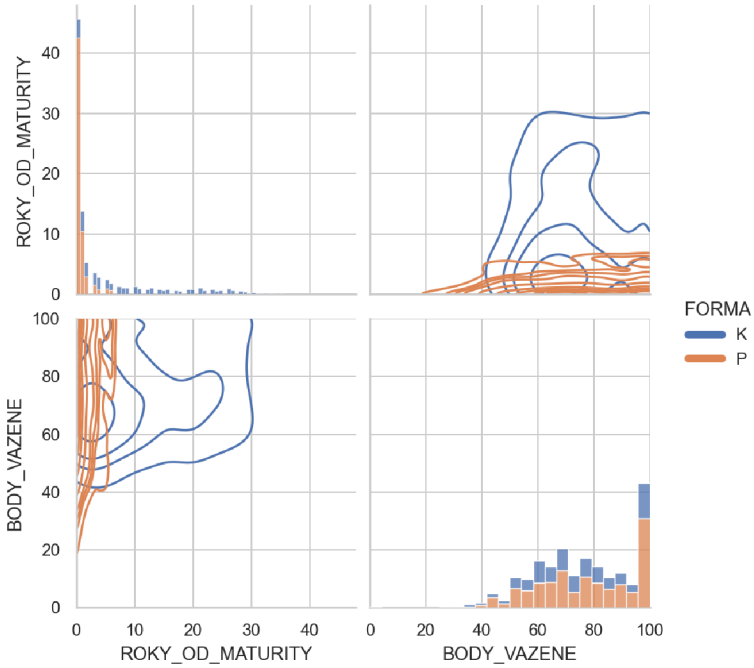


Figure 3.9: This pair plot shows that full-time students (FORMA=P) generally have fewer years from high school than part-time students (FORMA=K), but the distribution of admission points looks similar.

3.2 MODEL CREATION

We selected logistic regression, decision tree, Random Forest, K-Nearest Neighbors, Support Vector Machines, and Multi-Layer Perceptron methods to choose from as the final model. We described these in 2.1. We followed the usual workflow when obtaining the best model. It means splitting the dataset into training and testing and then splitting further into training and validation. To avoid bias towards particular samples in the validation dataset, we split with the help of the cross-validation technique (briefly explained in the following subsection). For several methods (K-Nearest Neighbors, Support Vector Machines, and Multi-Layer Perceptron), we applied standardization. After the training process, we picked the best model of each method and compared them together with the help of chosen statistical test.

3.2.1 Model training and validation

Model training is the process of finding (sub)optimal parameters (coefficients, weights) of the given model by optimizing the loss function (we described the idea behind it in 1.2). Model validation is the process of testing on the unseen data, usually done while doing hyperparameters tuning, i.e., the process of selecting the best values for (inverse of) regularization coefficient and solver in logistic regression, the minimal proportion of samples in a leaf or maximum depth of the tree in the decision tree, number of neighbors and type of metric in KNN, number of trees and the maximum depth of a tree in the RF, (inverse of) regularization coefficient and type of kernel in SVM, the learning rate and type of activation function in MLP and others we did not use. These hyperparameters' specific intervals and categories are in the source file *train_test*, and we selected them somewhat broader but still tried to make them sensible. In some cases, the optimization algorithm hits the limit of the interval. The reasons could be that the minimum exists outside the interval or there is no absolute minimum with only the saddle present. The figures like 3.12 show that second is probably the case. Moreover, even when the first would be the case, there is a question if increasing (or decreasing) interval limits are sensible, i.e., if we get hyperparameters values that make sense for the given method.

We use model validation results only for comparison models with different hyperparameters settings, not for the final evaluation of model performance because the model is undoubtedly biased toward the validation dataset.

We did the training on academic years started in 2017-2020 of our dataset (described in 3.1.3). We did 10-fold cross-validation for hyperparameters tuning. The K-fold cross-validation is a simple process of splitting the dataset into K distinct (not overlapping) sets (folds), which union is the entire dataset. The current fold is chosen for validation, and the rest is used for training. K validation results are averaged and then used to compare the different hyperparameters settings more robustly.

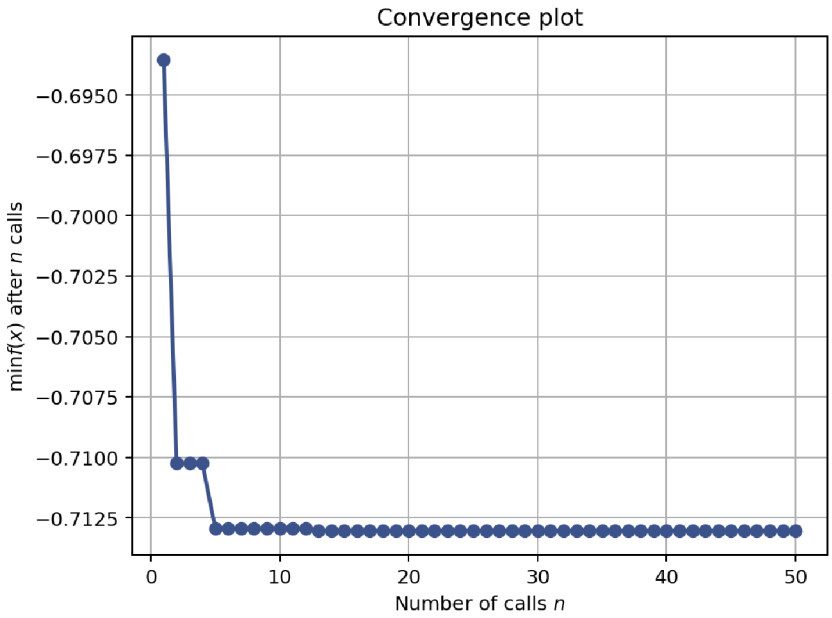


Figure 3.10: This convergence plot shows value of negated AUC metric depending on the iteration of Bayesian optimization.

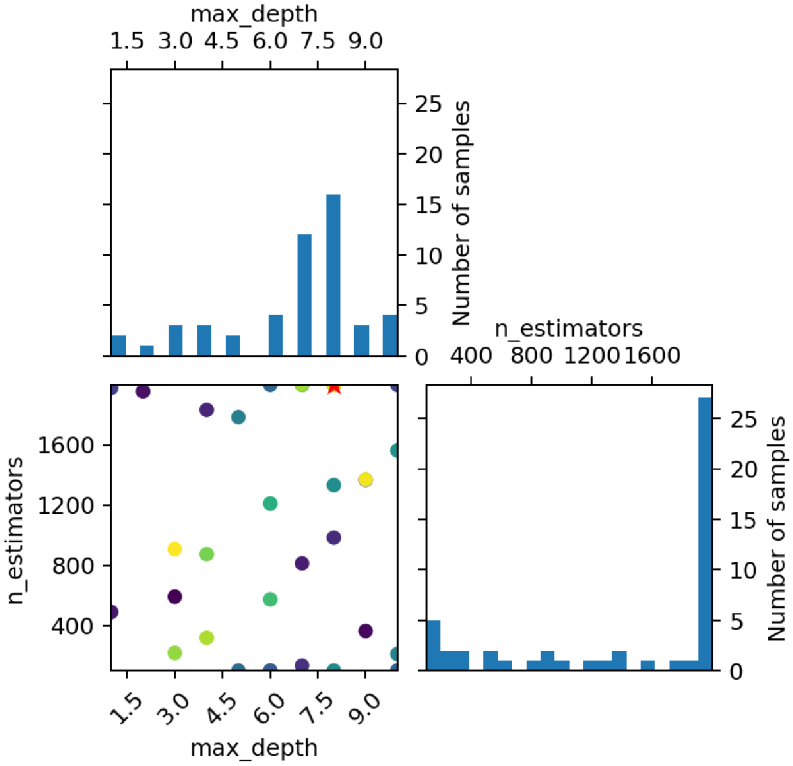


Figure 3.11: This pair plot shows our search space for our RF classifier. Hyperparameter *n_estimators* means the number of trees. The star denotes the location of the minimum found in the process.

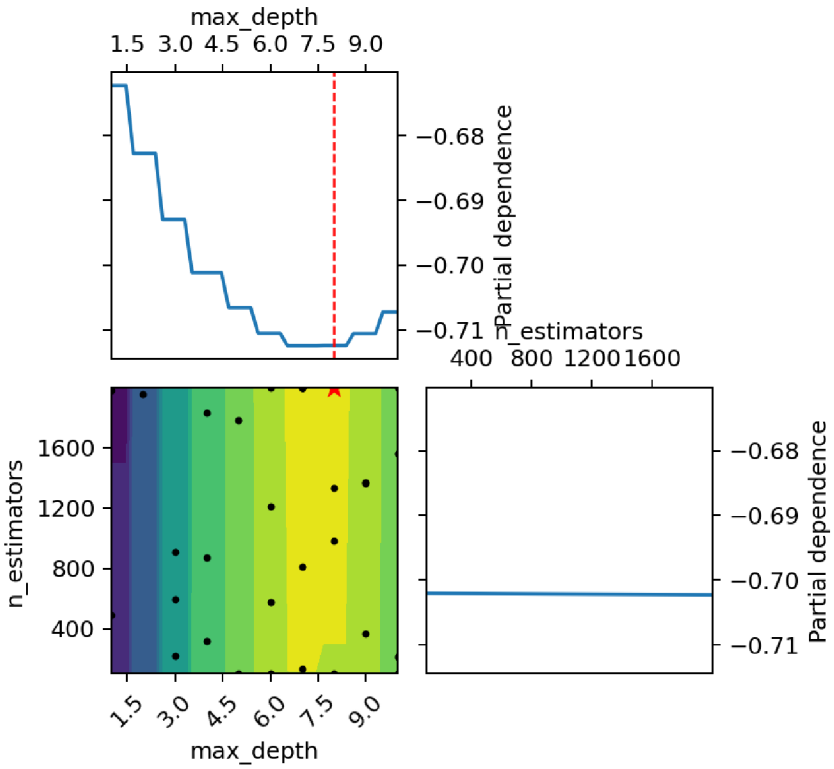


Figure 3.12: On the diagonal are graphs showing how values of given hyper-parameter influence the function we minimize.

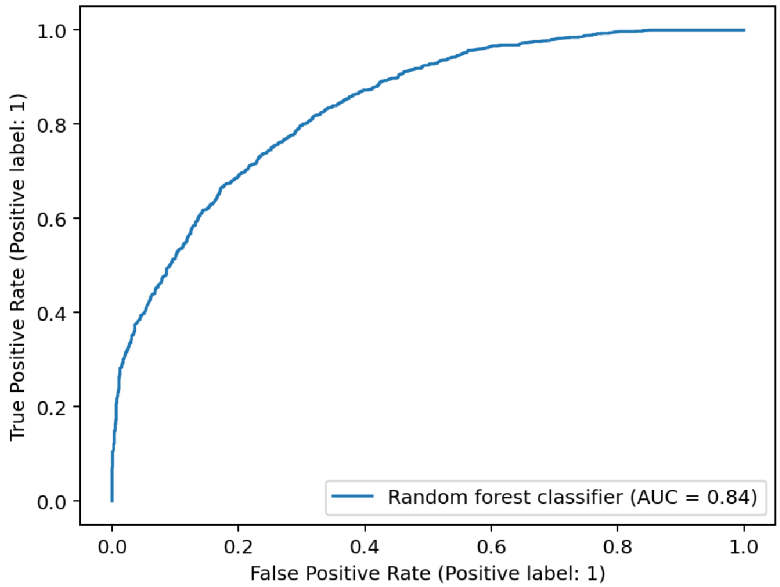


Figure 3.13: This graph shows ROC with AUC score of our classifier on training samples. The reason why The recall score is 0.866.

To semi-automatize the hyperparameters tuning, we used Bayesian optimization, also called sequential search, explained in this paper by [41]. We used implementation in the scikit-optimize library. Roughly speaking, this algorithm randomly selects values of chosen hyperparameters in a given search space, evaluates model performance on objective function (AUC in our case), and selects the values of the following hyperparameters based on it. It treats our objective function as random and places prior over our function, and after each evaluation, updates posterior distribution over it. After a few iterations (we used 100), it should converge to the suboptimum without computing any gradient.

The figures 3.10, 3.11, 3.12 and 3.13 are examples from the hyperparameters tuning process and show results on Random Forest classifier. More information about the first three figures can be found in scikit-optimize documentation. The AUC metric in the last one is explained in 2.2.4). Figures for the rest of the trained models and sub-models are in the source file *train_test* in the attachment.

3.2.2 Model testing

After achieving enough performance in the training stage, we can proceed to test to ensure our model generalizes well enough. Model testing is, in practice tricky thing if only limited data is available. We can do a point estimate of our business metric by giving all test data to the model and obtaining one number to compare with our goal or other models. However, to do a statistical test, we need more points. We can bootstrap our dataset to obtain several datasets of the same size, but we will break the assumed independence of individual observations. Without this assumption, we should not use Student's t-test and ANOVA (or their non-parametric alternatives). We chose as our approach to evaluate our classifiers and use Cochran's test [42], a generalization of McNemar's test. It tests if they give similar predictions because it compares the proportions of situations when models give the same or different output. We applied it with the help of the mlxtend library.

Here are the results running our best models on the test dataset.

Model	AUC	Recall
Logistic regression	0.63	0.787
Decision tree	0.57	0.691
Random forest	0.63	0.825
K-nearest neighbors	0.64	0.748
Support vector machines	0.61	0.760
Multi-layer perceptron	0.64	0.803

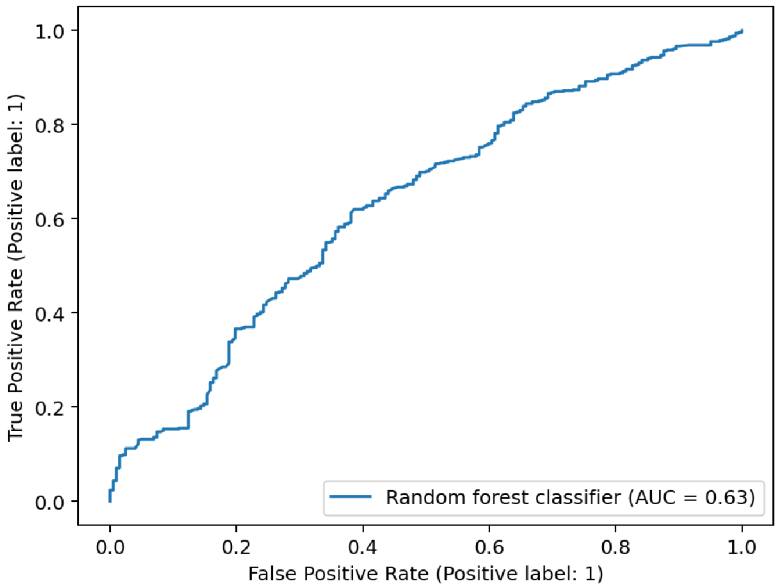


Figure 3.14: This graph shows ROC with AUC score of our classifier on testing samples. When comparing with 3.13, we can see that our classifier poorly generalize. The recall score is 0.825.

Here is the confusion matrix of our best (RF) classifier.

	pred 1	pred 0
true 1	419	89
true 0	130	72

We did three Cochran tests in total. We rejected the null hypothesis ($p < 2e-6$) that the best models of each method are the same. We did not reject the null hypothesis ($p = 0.374$) that tree best models - RF, MLP, and logistic regression are the same. We rejected the null hypothesis ($p < 1e{-}15$) that the best model based on Random Forest is the same as an artificial classifier with a 0.9 recall score (as we specified in the Goal definition 3.1.2). Because the RF classifier's score is lower than the artificial one, we can conclude that the RF classifier is not good enough. We show Receiver Operating Characteristic curve for the final best model in figure 3.14. The attachment shows the test's details at the end of the source file *train_test*.

3.2.3 Model interpretation

Model interpretation is not the main task of predictive modeling. Implications are that we are not limiting ourselves to highly interpretable models, and we are only sometimes conscientiously checking all assumptions as we are motivated by generalization on unseen data, not relationships. However, looking at what parameters the model learns is beneficial to check if

it makes sense and if we did everything correctly. Alternatively, we want to be inspired for our next try.

We looked at our logistic regression coefficients, which are easily interpretable as the slope of lines in subspace given by selected independent and dependent variables. We also printed out feature importances in our RF model, which proportionally measures how often a feature was used in trees. Because the algorithm chooses split in a tree based on criteria like entropy, we can roughly conclude that features with higher importance also have higher predictive power. We present these findings at the very end of the source file *train_test*, as we think it is in a more readable format.

3.3 FURTHER STEPS

Based on the results, there are two possible steps to take. If we have not achieved enough performance, we can return to the training model, data preprocessing, or data collection.

Suppose we have achieved enough performance on our business metric. Then we can present the model to a stakeholder or deploy it to a production environment depending on our project objectives.

3.3.1 Another iteration of training process

Another iteration is our next possible step, as we have yet to achieve the targeted performance. We need a new test dataset, as the current was already used. We are increasing the chance of Type I error otherwise by reusing it and introducing bias towards samples in the test dataset in our model. On our try, we trained submodels (mainly RF models) or used a more complex dataset with more granular variables SS_OBOR and VS_OBOR, but nothing led to better results. It hints that our data might not be of sufficient quality (we have to, for example, vaguely categorize an applicant's high school), or we may not have used some significant predictor that influences the applicant's success.

3.3.2 Deployment and continuous evaluation

We would like to use our model for our project objectives if we would have achieved enough performance. In the case of our task to predict if an applicant can finish the first year of college studies successfully, this usage could involve predicting the applicants' success and then deciding on a threshold in the admission process. However, a question is whether the variables we would use for such prediction can be part of the decision process.

Part of deployment to the real world is continuously evaluating our model and retraining if needed. Because the world constantly changes, and our data should go with it.

CONCLUSIONS

We have empirically compared six methods often used in ML on a real dataset. Before that, we have explained these methods (2.1) alongside metrics (2.2) used for comparison. Furthermore, even before that, we started this work with a brief introduction about why is such empirical comparison needed for every specific task and why we cannot do better (1.2.3). In comparison, we have not rejected that our three best models (based on Random Forest, Multi-Layer Perceptron, and logistic regression methods) are the same. We can be sure about such a conclusion because we fed our dataset into the algorithms in an identical format and used the same optimization for hyperparameter tuning. Of course, methods like logistic regression could be improved by adding interactions, but it is an extra process that makes comparison with, for example, neural networks less straightforward. One can argue that such a thing makes comparison fairer because neural networks and (other used algorithms to some extent) can implicitly exploit a non-linear relationship between dependent and independent variables.

To conclude our classification task to predict the applicant's success in the first year of study, we failed to provide a good classifier compared to our target performance. We did our best as we applied optimization techniques to tune hyperparameters, did cross-validation in the training process to be more certain about selecting a model with good generalization capability, and even tried submodels or a slightly modified training dataset. This result leads to the question if the task is solvable with targeted performance because, as we can see in theory (we specifically mean agnostic PAC discussed in 1.2.2), we usually assume that minimal possible error is non-zero and theoretically comes from noise in our data or the absence of important predictors.

Talking about noise, having clear, recognizable categories of high schools (or even better, high school programs of an applicant, as one high school can have multiple different study programs) could improve the performance of our models. When talking about other potential predictors in our task, we can take a look at case study *Early Prediction of student's Performance in Higher Education* by Martins et al. [43] and specifically at dataset they used. There are 37 variables included in this dataset about Portuguese higher education students, such as marital status, qualification and occupation of parents, if a student is a holder of a scholarship, or what is current inflation or unemployment rate in the country is. It gives more power, as we could use infor-

mation about student's financial and family background or current economic situation in a country that could motivate them to obtain higher education degrees.

Another way how results could be improved is to try different algorithms. We have yet to try every promising method available, as it was out of our resources. Nevertheless, different neural network architectures may lead to a better result. Using meta-models, when we average or mode output of different models to get the final prediction, is also a possible way to achieve improved performance. However, the data-centric approach (when we improve our models by improving our dataset) is emerging trend in academia, and it is the first thing to try to get the better model.

BIBLIOGRAPHY

1. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. *Efficient Estimation of Word Representations in Vector Space*. arXiv, 2013. Available from DOI: 10.48550/ARXIV.1301.3781.
2. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need*. arXiv, 2017. Available from DOI: 10.48550/ARXIV.1706.03762.
3. BROWN, Tom B.; MANN, Benjamin; RYDER, Nick; SUBBIAH, Melanie; KAPLAN, Jared; DHARIWAL, Prafulla; NEELAKANTAN, Arvind; SHYAM, Pranav; SASTRY, Girish; ASKELL, Amanda; AGARWAL, Sandhini; HERBERT-VOSS, Ariel; KRUEGER, Gretchen; HENIGHAN, Tom; CHILD, Rewon; RAMESH, Aditya; ZIEGLER, Daniel M.; WU, Jeffrey; WINTER, Clemens; HESSE, Christopher; CHEN, Mark; SIGLER, Eric; LITWIN, Mateusz; GRAY, Scott; CHESS, Benjamin; CLARK, Jack; BERNER, Christopher; MCCANDLISH, Sam; RADFORD, Alec; SUTSKEVER, Ilya; AMODEI, Dario. *Language Models are Few-Shot Learners*. arXiv, 2020. Available from DOI: 10.48550/ARXIV.2005.14165.
4. GOODFELLOW, Ian J.; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. *Generative Adversarial Networks*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1406.2661.
5. KARRAS, Tero; LAINE, Samuli; AILA, Timo. *A Style-Based Generator Architecture for Generative Adversarial Networks*. arXiv, 2018. Available from DOI: 10.48550/ARXIV.1812.04948.
6. REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. *You Only Look Once: Unified, Real-Time Object Detection*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1506.02640.
7. COVINGTON, Paul; ADAMS, Jay; SARGIN, Emre. Deep Neural Networks for YouTube Recommendations. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA, 2016.

8. SILVER, David; HUANG, Aja; MADDISON, Chris J.; GUEZ, Arthur; SIFRE, Laurent; DRIESSCHE, George van den; SCHRITTWIESER, Julian; ANTONOGLOU, Ioannis; PANNEERSHELVAM, Veda; LANCTOT, Marc; DIELEMAN, Sander; GREWE, Dominik; NHAM, John; KALCHBRENNER, Nal; SUTSKEVER, Ilya; LILLICRAP, Timothy; LEACH, Madeleine; KAVUKCUOGLU, Koray; GRAEPEL, Thore; HASSABIS, Demis. Mastering the game of Go with deep neural networks and tree search. *Nature*. 2016, vol. 529, no. 7587, pp. 484–489. ISSN 1476-4687. Available from DOI: 10.1038/nature16961.
9. OPENAI; : BERNER, Christopher; BROCKMAN, Greg; CHAN, Brooke; CHEUNG, Vicki; DĘBIAK, Przemysław; DENNISON, Christy; FARHI, David; FISCHER, Quirin; HASHME, Shariq; HESSE, Chris; JÓZEFOWICZ, Rafal; GRAY, Scott; OLSSON, Catherine; PACHOCKI, Jakub; PETROV, Michael; PINTO, Henrique P. d. O.; RAIMAN, Jonathan; SALIMANS, Tim; SCHLATTER, Jeremy; SCHNEIDER, Jonas; SIDOR, Szymon; SUTSKEVER, Ilya; TANG, Jie; WOLSKI, Filip; ZHANG, Susan. *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1912.06680.
10. JUMPER, John; EVANS, Richard; PRITZEL, Alexander; GREEN, Tim; FIGURNOV, Michael; RONNEBERGER, Olaf; TUNYASUVUNAKOOL, Kathryn; BATES, Russ; ŽÍDEK, Augustin; POTAPENKO, Anna; BRIDGLAND, Alex; MEYER, Clemens; KOHL, Simon A. A.; BALLARD, Andrew J.; COWIE, Andrew; ROMERA-PAREDES, Bernardino; NIKOLOV, Stanislav; JAIN, Rishub; ADLER, Jonas; BACK, Trevor; PETERSEN, Stig; REIMAN, David; CLANCY, Ellen; ZIELINSKI, Michal; STEINEGGER, Martin; PACHOLSKA, Michalina; BERGHAMMER, Tamas; BODENSTEIN, Sebastian; SILVER, David; VINYALS, Oriol; SENIOR, Andrew W.; KAVUKCUOGLU, Koray; KOHLI, Pushmeet; HASSABIS, Demis. Highly accurate protein structure prediction with AlphaFold. *Nature*. 2021, vol. 596, no. 7873, pp. 583–589. ISSN 1476-4687. Available from DOI: 10.1038/s41586-021-03819-2.
11. DAVIES, Alex; JUHÁSZ, András; LACKENBY, Marc; TOMASEV, Nenad. *The signature and cusp geometry of hyperbolic knots*. arXiv, 2021. Available from DOI: 10.48550/ARXIV.2111.15323.
12. FAWZI, Alhussein; BALOG, Matej; HUANG, Aja; HUBERT, Thomas; ROMERA-PAREDES, Bernardino; BAREKATAIN, Mohammadamin; NOVIKOV, Alexander; R. RUIZ, Francisco J.; SCHRITTWIESER, Julian; SWIRSZCZ, Grzegorz; SILVER, David; HASSABIS, Demis; KOHLI, Pushmeet. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*. 2022, vol. 610, no. 7930, pp. 47–53. ISSN 1476-4687. Available from DOI: 10.1038/s41586-022-05172-4.
13. ŠPETLÍK, Martin; BŘEZINA, Jan. Groundwater Contaminant Transport Solved by Monte Carlo Methods Accelerated by Deep Learning Meta-Model. *Applied Sciences*. 2022, vol. 12, no. 15, p. 7382. ISSN 2076-3417. Available from DOI: 10.3390/app12157382.

14. GUEST, Dan; CRANMER, Kyle; WHITESON, Daniel. Deep Learning and Its Application to LHC Physics. *Annual Review of Nuclear and Particle Science*. 2018, vol. 68, no. 1, pp. 161–181. Available from DOI: 10.1146/annurev-nucl-101917-021019.

15. ABADI, Martín; BARHAM, Paul; CHEN, Jianmin; CHEN, Zhifeng; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; IRVING, Geoffrey; ISARD, Michael; KUDLUR, Manjunath; LEVENBERG, Josh; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek G.; STEINER, Benoit; TUCKER, Paul; VASUDEVAN, Vijay; WARDEN, Pete; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. TensorFlow: A System for Large-Scale Machine Learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 265–283. ISBN 978-1-931971-33-1. Available also from: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

16. JAMES, Gareth; WITTEN, Daniela; HASTIE, Trevor; TIBSHIRANI, Robert. *An introduction to statistical learning : with applications in R*. New York : Springer, [2013] ©2013, [n.d.].

17. MITCHELL, Tom M. *Machine learning*. Vol. 1. McGraw-hill New York, 1997.

18. KOZYRKOV, Cassie. *The simplest explanation of machine learning you'll ever read*. Medium, 2018-05. Available also from: https://kozyrkov.medium.com/the-simplest-explanation-of-machine-learning-youll-ever-read-bebc0700047c.

19. AGGARWAL, Charu C. *Data Classification: Algorithms and Applications*. 1st. Chapman&Hall/CRC, 2014. ISBN 1466586745.

20. SHALEV-SHWARTZ, Shai; BEN-DAVID, Shai. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. Available from DOI: 10.1017/CBO9781107298019.

21. KIVINEN, Jyrki; SLOAN, Robert H. *Computational Learning Theory*. 1st ed. Berlin: Springer Berlin, Heidelberg, 2002. ISBN 978-3-540-45435-9. Available also from: https://doi.org/10.1007/3-540-45435-7.

22. MOHRI, Mehryar; ROSTAMIZADEH, Afshin; TALWALKAR, Ameet. *Foundations of Machine Learning*. 2nd ed. Cambridge, MA: MIT Press, 2018. Adaptive Computation and Machine Learning. ISBN 978-0-262-03940-6.

23. DU, Ke-Lin; SWAMY, M.N.s. *Neural Networks and Statistical Learning*. 2013. ISBN 978-1-4471-5570-6. Available from DOI: 10.1007/978-1-4471-5571-3.

24. KUBAT, Miroslav. *An Introduction to Machine Learning*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN 3319200097.

25. BLUM, Avrim. Machine Learning Theory. 2003, p. 4. Available also from: https://www.cs.cmu.edu/~avrim/Talks/mlt.pdf.

26. MICHIE, Donald; SPIEGELHALTER, D. J.; TAYLOR, C. C.; CAMPBELL, John (eds.). *Machine Learning, Neural and Statistical Classification*. USA: Ellis Horwood, 1995. ISBN 013106360X.

27. BURGER, Scott. *Introduction to Machine Learning with R*. 1st ed. USA: O'Reilly Media, 2018. ISBN 9781491976449.

28. WITTEN, Ian H.; FRANK, Eibe; HALL, Mark A. *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123748569.

29. UDDIN, Shahadat; KHAN, Arif; HOSSAIN, Md Ekramul; MONI, Mohammad Ali. Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making*. 2019, vol. 19, p. 281. ISSN 1472-6947. Available from DOI: 10.1186/s12911-019-1004-8.

30. YUVALI, Meliz; YAMAN, Belma; TOSUN, Özgür. Classification Comparison of Machine Learning Algorithms Using Two Independent CAD Datasets. *Mathematics*. 2022, vol. 10. ISSN 2227-7390. Available from DOI: 10.3390/math10030311.

31. MATTMAN, Chris A. *Machine Learning with TensorFlow, Second Edition*. Manning Publications, 2021. ISBN 9781617297717.

32. TUFFÉRY, Stéphane. *Data Mining and Statistics for Decision Making*. John Wiley & Sons, Ltd, 2011. ISBN 9780470979174. Available from DOI: https://doi.org/10.1002/9780470979174.ch11.

33. GIUDICI, Paolo; FIGINI, Silvia. *Applied Data Mining for Business and Industry*. 2nd Edition. Wiley, 2009. ISBN 978-0-470-74582-3.

34. PEKHIMENKO, Gennady G. Penalizied Logistic Regression for Classification. In: 2006. Available also from: https://www.cs.cmu.edu/~gpekhime/Projects/CSC2515/project.pdf.

35. MINKA, Thomas. A comparison of numerical optimizers for logistic regression. *CMU Technical Report*. 2003, vol. 2003. Available also from: https://tminka.github.io/papers/logreg/minka-logreg.pdf.

36. HAN, Jiawei; KAMBER, Micheline; PEI, Jian. *Data Mining: Concepts and Techniques*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123814790.

37. TING, Kai Ming. *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Boston, MA: Springer US, 2010. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_652.

38. ZHANG, Ethan; ZHANG, Yi. *Encyclopedia of Database Systems*. Ed. by LIU, LING; ÖZSU, M. TAMER. Boston, MA: Springer US, 2009. ISBN 978-0-387-39940-9. Available from DOI: 10.1007/978-0-387-39940-9_483.

39. KOZYRKOV, Cassie. *Making Friends with Machine Learning: The Entire Course*. Google Cloud, 2022. Available also from: https://youtu.be/1vkb7BCMQd0.

40. KOZYRKOV, Cassie. *The 12 Steps to Machine Learning and AI*. Google Cloud, 2022. Available also from: https://youtu.be/C_Q_L0wdPNg.

41. BERGSTRA, James; BARDENET, Rémi; BENGIO, Yoshua; KÉGL, Balázs. Algorithms for Hyper-Parameter Optimization. In: SHAWE-TAYLOR, J.; ZEMEL, R.; BARTLETT, P.; PEREIRA, F.; WEINBERGER, K.Q. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2011, vol. 24. Available also from: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

42. COCHRAN, W. G. THE COMPARISON OF PERCENTAGES IN MATCHED SAMPLES. *Biometrika*. 1950, vol. 37, no. 3-4, pp. 256–266. ISSN 0006-3444. Available from DOI: 10.1093/biomet/37.3-4.256.

43. MARTINS, Mónica V.; TOLLEDO, Daniel; MACHADO, Jorge; BAPTISTA, Luís M. T.; REALINHO, Valentim. Early Prediction of student's Performance in Higher Education: A Case Study. In: ROCHA, Álvaro; ADELI, Hojjat; DZEMYDA, Gintautas; MOREIRA, Fernando; RAMALHO CORREIA, Ana Maria (eds.). *Trends and Applications in Information Systems and Technologies*. Cham: Springer International Publishing, 2021, pp. 166–175. ISBN 978-3-030-72657-7.