BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MACHINE LEARNING OPTIMIZATION OF KPI PREDICTION

OPTIMALIZACE STROJOVÉHO UČENÍ PRO PREDIKCI KPI

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. DANIEL HARIS

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2018

Zadání diplomové práce/21142/2017/xharis00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Haris Daniel, Bc.**
Obor: Informační systémy
Téma: **Optimalizace strojového učení pro predikci KPI**
Machine Learning Optimization of KPI Prediction
Kategorie: Data mining

Pokyny:

1. Seznamte se s problematikou predikce KPI (klíčové ukazatele výkonosti projektů). Pro tento účel nastudujte vhodné modely.
2. Zvolte a implementujte vhodný postup pro výběr vhodných dat k učení zvolených modelů. Vyhodnoťte a využijte nejvhodnější existující metriky, které budou vyvážené tak, aby měl kandidátní model dobrou predikční schopnost a byl schopen generalizovat své předpovědi nad novými daty (cross-validation, k-fold a další).
3. Porovnejte vhodné kandidátní modely mezi sebou a vyberte nejlepší pro daný problém.
4. Zvolený nejlepší model poté implementujte v jazyce Python v rámci existující aplikace firmy.
5. Zhodnoťte dosažené výsledky a navrhněte další metriky, které by mohly zvýšit predikční sílu modelů v raných a pozdních částech vývoje projektů.

Literatura:

- Abbott, D.: Applied Predictive Analytics. Wiley, 2014. ISBN: 978-1118727966
- Parmenter, D.: Key Performace Indicators. Wiley, 2010. ISBN: 978-0470545157

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

[illegible]

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstract

This thesis aims to optimize the machine learning algorithms for predicting KPI metrics for an organization. The organization is predicting whether projects meet planned deadlines of the last phase of development process using machine learning. The work focuses on the analysis of prediction models and sets the goal of selecting new candidate models for the prediction system. We have implemented a system that automatically selects the best feature variables for learning. Trained models were evaluated by several performance metrics and the best candidates were chosen for the prediction. Candidate models achieved higher accuracy, which means, that the prediction system provides more reliable responses. We suggested other improvements that could increase the accuracy of the forecast.

Abstrakt

Cieľom tejto práce je optimalizácia strojového učenia pre predikciu KPI metrík pre jednu organizáciu. Organizácia predpovedá oneskorenie termínov ukončenia poslednej fázy projektov v procese vývoja pomocou strojového učenia. Práca sa zameriava na analýzu predikčných modelov a stanoví si za cieľ vybrať nové kandidátne modely na predikciu. V rámci práce sme implementovali systém, ktorý automaticky vyberie najlepšie rysy pre učenie. Naučené modely sme vyhodnotili pomocou rôznych výkonnostných metrík a vybrali najlepšie kandidátne modely. Kandidátne modely majú vyššiu presnosť predpovede, čo pre organizáciu znamená, že sa zvýšila dôveryschopnosť predpovede oneskorenia. V závere práce sme navrhli ďalšie vylepšenia, ktoré by mohli zvýšiť presnosť predpovede.

Keywords

software development, KPI, NPI, machine learning, preprocessing, cross-validation, classification, prediction, boosting, neural networks, ensemble, boosting, support vector machines, decision tree, random forest, naive bayes, performance metrics, ROC

Klíčová slova

vývoj softvéru, KPI, NPI, strojové učenie, predspracovanie, krížová validácia, klasifikácia, predikcia, boosting, support vector machines, neurónové siete, rozhodovací strom, náhodný les, naive bayes, výkonnostné metriky, ROC

Reference

HARIS, Daniel. *Machine Learning Optimization of KPI Prediction*. Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vladimír Bartík, Ph.D.

Rozšířený abstrakt

Úvod

Cieľom tejto práce je optimalizácia strojového učenia pre predikciu KPI metrík pre jednu organizáciu. Organizácia používa systém na predpoveď oneskorenia termínov ukončenia poslednej fázy procesu vývoja projektov pomocou strojového učenia.

Na začiatku práce sme sa venovali výkonnostným metrikám, ktoré sa používajú v rámci organizácií na sledovanie naplánovaných cieľov. Popísali sme 4 typy výkonnostných metrík. Organizácia používa metriku, ktorá sa nazýva *Phase Gate Compliance* (voľne preložené ako *súlad s fázovou bránou*). Táto metrika odráža aktuálny stav projektov, či sú v súlade s naplánovanými termínmi jednotlivých fáz vývoja. Organizácia predpovedá, s akou pravdepodobnosťou spĺňajú aktívne projekty stanovené termíny posledných fáz vývoja. Existujúci predikčný systém sa používa, avšak bol implementovaný rapídnym vývojom. Táto práca si stanovuje za cieľ rozšíriť daný systém o ďalšie predikčné modely a taktiež optimalizovať celkový proces spracovania dát.

Zvýšenie dôveryhodnosti predpovede danej metriky je kľúčové pre organizáciu – odhalením potenciálnych problémov v skorších fázach vývoja projektu dokáže organizácia minimalizovať oneskorenie projektov a tým aj vstup produktov na trh.

Problém predpovede oneskorenia poslednej fázy vývoja projektu je definovaná ako binárna klasifikačná úloha – projekty klasifikujeme do dvoch tried podľa toho, či ich fáza testovania skončila s oneskorením alebo nie. Z tohoto dôvodu sme sa v tejto práci zamerali na klasifikačné algoritmy strojového učenia. Každý klasifikačný model sme dôkladne popísali a vyhodnotili ich výhody a nevýhody. Práca sa venuje taktiež aj teoretickému rozboru nástrojov na vyhodnotenie klasifikačných modelov.

Popis riešenia

V druhej časti práce sme popísali proces získania dát na učenie. V organizácii sa používajú viaceré systémy na projektový manažment, pričom každý systém uchováva inú doménu informácii o daných projektoch. Na základe úplnosti a kvality dát sme vybrali 3 systémy, z ktorých budeme exportovať dáta slúžiace na učenie predikčných modelov. Exportované dáta sme museli rozdeliť podľa biznisu a veľkosti projektov (projekty týkajúce sa *termostatov* a *bezpečnostných panelov*), pretože proces vývoja týchto skupín projektov sa môže líšiť. Získané dáta boli v surovom stave, čo nie je vhodné pre klasifikačné modely – z tohto dôvodu bolo potrebné dáta predspracovať a normalizovať, aby s nimi modely dokázali pracovať. Programovacím jazykom práce je Python.

Exportované dáta obsahovali veľký počet dimenzií, ktoré sme museli zredukovať a vybrať vhodné rysy (features). V rámci práce sme navrhli a implementovali systém, ktorý automaticky vyberie najlepšie rysy na predikciu. Jadro práce tvorí implementácia klasifikačných modelov a ich učenie. Na vyhodnotenie modelov používame vizualizačné nástroje, ktoré generujú grafy na vizualizáciu výkonu a úspešnosti daných modelov.

Vytvorili sme dve sady trénovacích množín:

- prvá sada obsahovala projekty z rokov 2014 a 2015 – táto sada sa použila na učenie modelov, ktoré sú momentálne nasadené v predikčnom systéme;
- druhá sada obsahovala projekty z rokov 2014 až 2016 – sada použitá na učenie nových modelov.

Následne sme vytvorili testovaciu sadu tvorenú z projektov z roku 2017 (vieme, či tieto projekty skončili s oneskorením). Na tejto sade sme otestovali predikčnú silu existujúcich a nových modelov.

Výsledky

Na konci práce sme porovnali nové modely s existujúcimi modelmi. Modely, ktoré mali najvyššiu presnosť predpovede, sme navrhli ako kandidátne modely do predikčného systému organizácie. Všetky kandidátne modely dosiahli vyššiu presnosť predpovedí ako momentálne nasadené modely. Táto skutočnosť znamená, že všetky kandidátne modely úspešne vytvorili generalizačné pravidlá, pomocou ktorých dokázali úspešne klasifikovať projekty podľa oneskorenia poslednej fázy vývoja.

Na ďalšie vylepšenie predikčného systému sme navrhli možnosti, ktoré by mohli zvýšiť predikčnú silu modelov. Jedno z možných vylepšení spočíva v dôkladnejšej analýze vybraných rysov na učenie a zistenie, prečo nadobúdajú dané hodnoty. Ďalšie vylepšenie sa týka špecifikácií požiadaviek. Organizácia si uvedomuje, že vágne, nepresné a často meniace sa požiadavky znamenajú problémy počas procesu vývoja. Je potrebné nastaviť také metriky, pomocou ktorých by organizácia sledovala kvalitu požiadaviek a taktiež by pomohli znížiť počet žiadostí o zmenu požiadaviek.

Machine Learning Optimization of KPI Prediction

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Vladimír Bartík, PhD. The supplementary information was provided by Ing. et Ing. Petr Boháček. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

....
Daniel Haris
May 22, 2018

Acknowledgements

Firstly, I would like to express my sincere gratitude to my manager in the organization Ing. et Ing. Petr Boháček for the continuous support, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Besides my manager, I would like to thank my thesis supervisor Ing. Vladimír Bartík, PhD. for his valuable advice, guidance, and support he provided me. Furthermore, I would like to thank my loved ones for supporting me spiritually throughout writing this thesis.

Contents

Chapter 1

Introduction

The assignment of this thesis comes from an IT company operating in the area of smart home and IoT. Nowadays, in the field of smart home technologies is a significant competition going on and to succeed in the market, one has to minimize the cycle time of products [29][3][50]. This company specializes on design and development of various smart home technologies.

Internally, every product is part of a project that is process-driven. The organization implements the New Product Introduction (NPI) process which consists of six phases of development, five of which are marketing and research and development (R&D) related and the last phase is the manufacturing of final products. Any delay in development will delay the delivery of the product to the target customers and thus jeopardize the success of the product on the market.

The process of developing these devices (generally an IT product) is a complex task, and in reality, it is often the case that due to problems, the project is delayed. Typically, IT companies are having problems with incomplete, inaccurate, vague requirements or inadequate testing [41]. If the development expands to a more extended period, the company may lose potential customers, and the project may end up in financial loss.

Management had set several performance metrics that track the development of projects. One of these performance metrics is called Phase Gate Compliance which defines whether the project is compliant with deadlines set during the project planning. The company had to implement a program based on machine learning to reveal potential problems in earlier development phases. This program is predicting the compliance status of projects in earlier phases. Fortunately, this idea arose when the author of this thesis joined this company and had the opportunity to participate in the development of this prediction system.

The goal of this thesis is to optimize this prediction system. We will focus on the analysis of various classification algorithms and tools for their evaluation. After training new models, we will summarize their achieved performance. According to these results, we will suggest best candidate models for the prediction system.

In chapter 2 we will focus on measuring performance within an organization. Next, in chapter 3 we will describe the organization where the assignment of this thesis comes from. We analyze the NPI process to have a better insight of projects in this organization. Then, we will introduce and define the Phase Gate Compliance metric and give more detail of the existing prediction system. In chapter 4 we will dive into machine learning and describe the main approaches for solving classification problems. Evaluation of model's performance and candidate selection will be covered also in this chapter. Chapter 5 contains data preprocessing and feature selection. The implementation of prediction models are described in chapter 6. Then, all the results of classification models are summarized in chapter 7. We will evaluate the existing and newly created models and compare their results. According to the result, we will choose candidate models which will be deployed in the organization. Finally, in chapter 8 we will discuss possible improvements to the prediction system.

Chapter 2

Measuring performance

Measuring performance is a vital part of monitoring an organization's progress. It contains measuring the actual performance outcomes or results of an organization against its settled goals [6]. The strategic plan provides performance targets for the organization, and it sets the direction of the organization. Unfortunately, the strategic plan usually doesn't set performance measurement target for all levels of the organization. As a result, performance improvement opportunities are overlooked to support the strategic plan, and the organization's progress is therefore limited. The performance measurement adopted might have no linkage to the critical factors of the organizations. Unfortunately, many companies are working with the wrong measures, many of which are incorrectly termed key performance indicators.

To be clear, we define terms measure and metric:

- **Measure** – a number derived from taking a measurement or the observed value of a number at a point in time. Measures are raw numbers and data points found in data or reports — on their own, measures deliver little value.
- **Metric** – a calculated number derived from measures. Typically, expressed as a ratio, average, percentage etc.

There are 4 types of performance measures:

- **Key result indicator (KRI)** – tells how have been done in a perspective or critical success factor,
- **Result indicator (RI)** – tells what have been done,
- **Performance indicator (PI)** – tells what to do,
- **Key performance indicator (KPI)** – tells what to do to increase the performance dramatically.

2.1 Key result indicator (KRI)

They are often mistaken for KPI. They include measurements such as customer satisfaction, net profit before tax, employee satisfaction or profitability of customers. A common characteristic of these measures is that they are the result of many actions. Despite showing the organization's right direction, they do not tell what to do to improve these results. They typically cover a more extended period than KPIs (reviewed on monthly/quarterly cycles, not daily/weekly basis as KPIs). Between KRIs and KPIs are various performance and result indicators. Although there can be many result indicators, only key result indicators tell how one has done in a part of their business that is critical to meeting corporate goals [12].

2.2 Result indicator (RI)

RIs typically summarize activities, all financial performance measures are RIs for instance. They only tell what happened, but not why. Typical examples of RIs could be:

- daily or weekly sales analysis,
- customer complaints from key customers,
- net profit on key product lines,
- hospital bed utilization in week.

To fully understand what to increase or decrease, we need to look at the activities that created the result, e.g., the sales.

2.3 Performance indicator (PI)

Whereas result indicators measure the results of many business actions as an aggregate, performance indicators track specific actions or activities. They are metrics that inform how a business is doing and what to do and what actions to take.

They help teams to align themselves with their organization's strategy, but they are not key to the business. PIs are non-financial and complement to the KPIs. Typical examples of PIs could be:

- percentage increase in sales with top 10% customers,
- late deliveries to customers,
- number of employees' suggestions implemented in last 30 days.

2.4 Key performance indicator (KPI)

Since there are many PIs produced in the organization, KPIs should only be the ones that are necessary for business performance measurement. They should provide vital feedback and actionable insight which directly drive business goals. They track essential items that are considered critical to the success of the business [12].

KPIs represent a set of measures focusing on those aspects of organizational performance that are the most critical for the current and future success of an organization. The reason why lots of organizations fail to increase their performance could be the inability to recognize KPIs.

According to [46] there are 7 characteristics of KPIs. KPIs:

- are non financial measures,

- are measured frequently,
- are acted on by the CEO and senior management team,
- clearly indicate what action is required by staff,
- are measures that tie responsibility down to a team,
- have a significant impact,
- encourage appropriate action.

KPIs should be monitored 24/7, daily or weekly for some. Monthly, quarterly or annual measure cannot be a KPI, as it might be hard for business to react with a proper decision promptly. KPIs are current or future-oriented measures. Most organizational measures are very much past indicators measuring events of the last month or quarter. These indicators cannot be and never was KPIs.

2.5 Difference between KPIs and KRIs

KPIs measure precise actions to take to obtain specific results, whereas KRIs inform about the results of many activities. KRIs are backward looking and measure the effect of business activities, but ignore the cause. KRIs measure business goals, KPIs do not — it is essential to keep track of business goals, but it is not clear what is the cause of results. Actions and activities that align with these goals have to be tracked − with the KPIs.

Chapter 3

Organization

This work relates to an organization, which operates in the IT sector. It mainly focuses on research, development, and manufacturing of various smart home devices, e.g., thermostats. First, we describe the process-driven development called NPI process, which is implemented in this organization. Second, we explain, how does the management set business plans, how do they track active projects and how do they manage the progress of the goals. At the end of the chapter, we introduce a performance metric called *Phase Gate Compliance*, through which management has an overview of the success how well its projects meet the set deadlines. This metric is the subject of the prediction analysis of this thesis.

3.1 NPI process

A New Product Introduction (NPI) program encompasses all the activities within an organization to define, develop and launch a new or improved product. The product in our case could be an intelligent thermostat with cloud connectivity and user comfort prediction.

The acronyms NPI (New Product Introduction) and NPD (New Product Development) are used interchangeably by many organizations. In some cases, NPI activities begin after design and development and merely deal with the product production launch and marketing. The NPI process can vary from organization to organization. In some cases, it can vary within different divisions of the same company. For an NPI process to succeed it must have the full active support of upper management in all divisions and departments [10].

Successful organizations realize the importance of NPI process. Today is the era of a highly competitive market, and companies must develop the right product at right time and right cost. Advantages of NPI process:

- **Define** – in this phase the product's requirements are defined with the help of marketing and product management divisions. These requirements are converted into design specifications and they are integrated into one or more concepts to be reviewed by the project team. During this phase an initial business case is created -— it should identify the market, customers etc,

- **Feasibility** – the purpose of this phase is to allow management an opportunity to evaluate the project's potential for success. During this phase, the project team reviews the product design concepts and selects the one which fulfills previously defined requirements. The business case is reviewed as well and redefined. The output of this phase is a determination if the project and proposed product design should move forward to the next phase, be redesigned or dropped,

- **Develop** – this phase is focused on advancing the product design into a more defined form. A validation plan is also developed to evaluate the robustness of the design and its ability to meet the requirements. The project team presents updated information about the design, project timeline, risks, an updated business plan and financial status. The management will determine if the project should move to the next phase or change the design or test plan,
- **Validate** – during this phase product analysis and testing are performed. Design changes are possible due to the results of validation testing but are very costly. Besides, the manufacturing process is developing, and process risk is being analyzed; significant steps of the manufacturing process are being developed and reviewed. At the end of this phase, the team should review the project with the owner and stakeholders. The purpose of this review is to gain approval to move the project to the next phase,
- **Implement** – this phase of the NPI process refines and validates the manufacturing processes through pilot builds and capability studies. In addition, process documentation and quality control are being developed and implemented,
- **Evaluate** – during this phase the product is being produced or the service provided. At this point in the NPI process, the team has an opportunity to tie up any remaining documentation tasks, review process performance and collect customer feedback data.

3.2 Setting business goals

The organization consists of several businesses. Goals are set for each business which can vary due to market differences and specifics. Management sets KPI metrics for all businesses, tracks the performance of projects and monitors the success of meeting goals. If a particular business does not meet the settled goals, managers ask for causes lower in the hierarchy. The lower management is finding, analyzing and proposing solutions for reactive and systemic (where possible) solutions for these causes.

For example, let's assume that the global management wants to increase the effectivity of development of their products and therefore to decrease the cycle time of projects to withstand the competition on the market. They create a metric, which tracks which projects are within a cycle time limit since cycle time represents the duration of a process. Figure 3.1 represents a box plot with the distribution of cycle time for each type of projects in the given business. According to this statistics, they know what projects to target to increase the overall development efficiency.

3.3 Phase gate compliance (PGC)

This metric was created in the organization to capture the actual state of compliance with planned deadlines. PGC on a global level can be viewed as a result indicator, due to reflecting the current state of compliance with all projects in the organization. However, in a lower level of organization in a given business can be considered as a key result indicator, because the main goal of a R&D unit is the on-time delivery of products.

During the development process, the management is setting the *baseline dates* of phase gates. A baseline date is a fixed date, unlike a *planned date*. When a phase gate's baseline

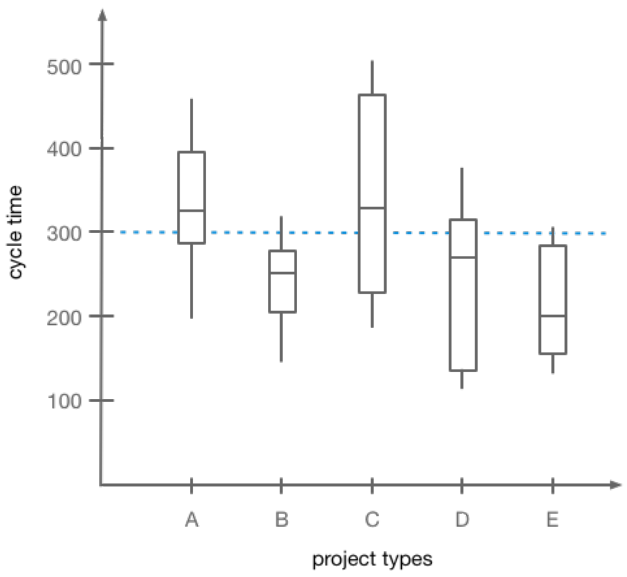


Figure 3.1: This box plot shows the cycle time distribution of projects in given business. Projects are grouped by their type. We can observe that project types B, D, and E meet the metric – they are within the metric limit of 300 cycle time. The management has to investigate, why do projects A and C have higher cycle time than the others.

is set, the planned date is also set to the same date. If the project must be delayed, the planned date is postponed. The PGC is defined as:

$$PGC = \begin{cases} \text{compliant} & (Planned - Baseline) \leq 30 days \\ \text{non-compliant} & \text{otherwise} \end{cases} \tag{3.1}$$

The problem is that if a project is labeled as *non-compliant*, it will stay *non-complaint* until the end of the project. This is the point where the prediction system comes in.

If the management knows how to predict which project would be non-compliant, they could deal with the project's problems and thus shorten the eventual delay of the project at later phases. Therefore, the management has instructed to implement such a prediction system that helps to keep the set business goal, i.e., on-time delivery.

3.4 Existing prediction system

The organization has already implemented a prediction system that predicts PGC status of phase 5 of active projects. The 5th phase is essential for the organization because testing ends and so the R&D phase ends, too. The project is then sent to the manufacturing for which another part of the organization is responsible.

The prediction consists of 2 phases:

early - prediction before the end of phase 3,

late - prediction between phases 3 and 5.

In general, the sooner the system predicts, the better the forecast will be. However, our goal is the opposite – we need to get the best forecast as soon as possible.

Despite the fact that this system is used by management to reveal and investigate potential problems in earlier phases of projects, it was developed rapidly with no broader analysis. Unfortunately, we do not know enough about the quality of the results the system provides us.

In this work, we will focus on detailed data analysis, analysis of feature selection and prediction models. At the end of this work, we compare current performance of the system with the performance of the optimized prediction system.

Chapter 4

Predictive models

The prediction of PGC status of projects is a binary classification problem − we classify projects into those that meet defined deadlines (compliant) and those that do not meet them (non-compliant). We have a four year (2013-2017) collection of project development data in the organization. Due to a large number of attributes (~400) in the data file, the problem is too complicated to use a rule-based system to solve this problem (due to the curse of dimensionality which we describe later in this chapter). To come up with a solution, we have to use machine learning that provides us the tools to handle a task of such complexity.

In this chapter, we describe what machine learning is and what problems can it solve. Since our problem is a classification problem, we will present some of the classification methods that are used in machine learning. Later in this chapter, we will look at measuring the performance of these models and describe how to select candidate models.

4.1 Introduction to machine learning

„*Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed*“ [48]. It explores the study and creation of such algorithms and techniques which has the ability to learn from data and make predictions on data. These algorithms overcome following strictly static program instructions by making data-driven predictions or decisions through building a model from sample inputs.[15]. Machine learning allows us to solve a wide range of computing tasks which are too complex or infeasible to solve with explicit algorithms or rule-based approaches.

Next few sections will discuss the types of learning in machine learning. The figure machine-learning-diagram shows the classification of machine learning algorithms by type of learning with a couple of examples. Later in this chapter, we will focus on supervised learning especially on classification algorithms.

4.1.1 Unsupervised learning

In this type of learning, we only have input data − there is no target variable. The aim of these methods is to explore the structure of data and find patterns in them. There is a structure to the input space such that certain patterns occur more often than others, and we want to investigate what generally does generally happen and what does not. In statistics, this is called *density estimation*.

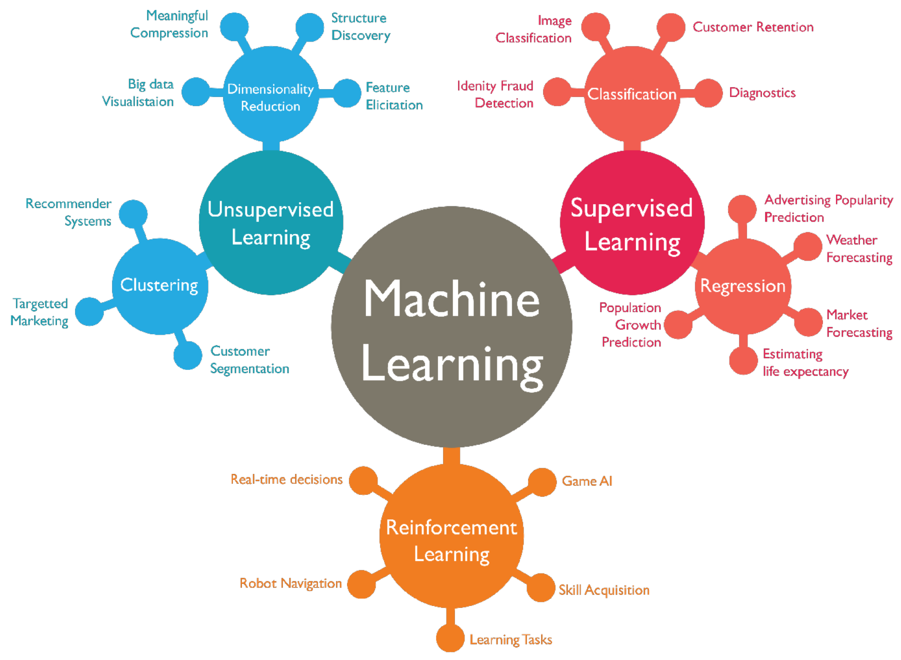


Figure 4.1: This diagram shows the classification of machine learning methods by type of learning they use. This work is related to classification algorithms which belongs to supervised learning. All the types of learning are discussed in section 4.1. Image taken from [5].

One method for density estimation is *clustering*. Clustering is the task of grouping a set of objects in such a way that objects in the same group (cluster) are more similar (in some sense) to each other than to those in other groups (clusters) [2]. This technique is used also to recognize outlying objects in the input data. Possible applications:

- customer segmentation,
- summarization of news,
- creation of fylogenetic trees in molecular biology,
- image compression [15].

4.1.2 Supervised learning

In this case, the computer is presented with example inputs and their desired outputs, given by a „teacher“, and the goal is to learn a general rule that maps inputs to desired outputs. Desired output is also called target variable and it is chosen to represent the answer to a question the organization would like to answer or a value unknown at the time the model is used that would help in decisions. Sometimes supervised learning is also called predictive modeling.

A *training dataset* is a dataset of examples used for training, that is to fit the parameters (or weights) of a model. A *testing dataset* is a dataset independent of the training dataset, but follows the same probability distribution as the training dataset. Supervised learning involves:

Classification - solves the problem of identifying to which of a set of classes (categories) a new observation belongs. In this case, target variables are discrete variables of categorical type. There are several types of classification according to the number of classes:

- problem with 2 classes is often called *two-class* or *binary* classification problem e.g. spam filtering,
- problem with more than 2 classes is called *multi-class* classification problem e.g. classification of images of fruits which may be oranges, apples or pears,
- problem where an input is assigned multiple classes is called *multi-label* classification problem e.g. recognizing topics in documents − text might be about any of religion, politics, finance or education at the same time or none of these.

Regression - this type of models work with continuous target variables. Regression is an approach for modeling the relationship between a dependent variable y (output/target variable) and 1 or more independent variable X (explanatory variable). The relationship between dependent variable and independent variables are described using predictor functions. According to the number of independent variables, we recognize these types of regression:

- *simple linear regression* - 1 independent variable e.g. predict the employee's salary according to his years of experience. In this example salary is the dependent variable (y) and years of experience represents the independent variable (X).
- *multiple linear regression* - in the case of more than 1 independent variables. For example predicting the price of a car. Inputs are the car attributes − brand, year, engine capacity and mileage. The output is the price of the car. In this example we have 4 independent variables.
- *polynomial regression* - previous regression types used linear functions to describe the relationship between variables. In cases where linear models are too restrictive, one can use polynomial regression. As an example we could mention the previous example of a car price prediction. If there is no linear relationship between attributes of a car and its price, we could try polynomial regression instead.

Machine learning algorithm optimizes the parameters of the predictor function in a way that the approximation error is minimized, that is, our prediction is as close as possible to the correct values given in the training dataset.

4.1.3 Reinforcement learning

It is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. It differs from supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

The output of the system is a sequence of *actions*. In such case, single action is not so important than the sequence of correct action leading to the goal. There is no best action in any intermediate state − an action is considered as good if it is part of a good *policy*. A good application of reinforcement learning is *game playing* where a single move by itself is not important; it is the *sequence of right moves* that is considered good. A move is good if it is part of a good game policy. An example of a game, where reinforcement learning was successfully applied is chess. It is a game with a small number of rules but it is very complex because of the large number of possible moves at each state and the large number of moves that a game might contains. Once we have good algorithms that can learn to play games well, we can also apply them to applications with more evident economic utility [15].

4.2 Classification algorithms

In this section we introduce several types of classification methods from which we choose some candidate models which we use later in the prediction system to forecast PGC status of projects in the organization.

4.2.1 Decision trees

A decision tree consists of internal decision nodes and terminal leaves. Each decision node implements a test function with discrete output values labeling the branches. Each node takes an input and applies the test and one of the branches is taken depending on the result. This process starts at the root node and is repeated recursively until a leaf node is hit, what represents the class where the input was classified. Figure 4.2 shows an example of a decision tree.

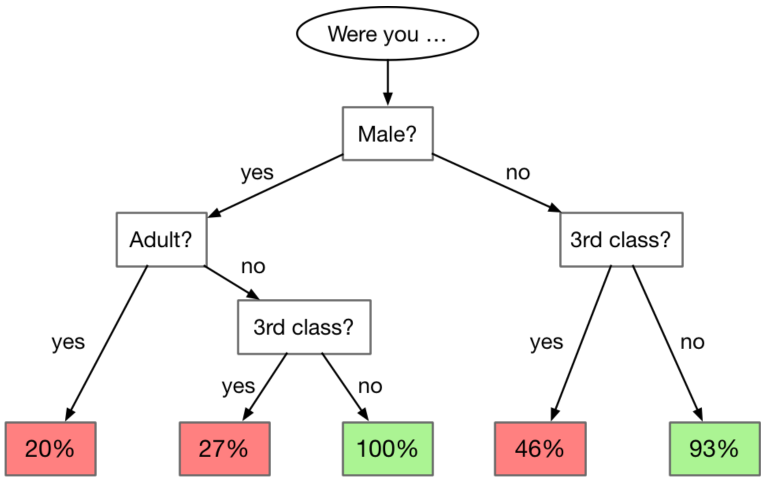


Figure 4.2: This figure shows an example of a decision tree classifying/predicting the survival likelyhood of a person on Titanic. From the results, it seems that one would have quite a chance of being rescued from the ship if one is a female from 1st/2nd class cabin or a male child from 1st/2nd class cabin. Image was taken from [44].

Decision trees belong to a class of *recursive partitioning algorithms* which are simple to describe and implement. Each variant of these algorithms share the following steps [14]:

1. For each candidate input variable, assess the *best* way to split the data into 2 or more partitions, select the best split and divide the data into groups defined by the split.

2. For all groups repeat step 1 (recursively).

3. Continue splitting until all records after a split belong to the same class or until another stop condition is applied (statistical significance test or minimum record count).

The key principle which differentiate various algorithms in this group of classifiers is the way they do the split. They all provide a measure of purity of the class distribution [14]. Let's describe the variants of decision tree algorithms.

ID3

This algorithm uses an *information gain* to decide which attribute goes into a decision node. Gain measures how well a given attribute separates training samples into classes. Attribute with the highest information is selected. Gain is a measure based on *entropy* which measures the amount of information in an attribute [13].

Let S represent the training samples. Let $A = \{a_1, a_2, \ldots, a_v\}$ represents a set of individual values of attribute A. Then let S_j be a subset of training samples for which attribute A contains only values $a_j(j = 1, \ldots, v)$ and s_{ij} represents training samples of set S_j, which belong to class $C_i(i = 1, \ldots, m; j = 1, \ldots, v)$ [24]. We define the entropy for S_j

$$Entropy(S_j) = -\sum_{i=1}^{m} \frac{s_{ij}}{|S_j|} \log_2(\frac{s_{ij}}{|S_j|}) \tag{4.1}$$

Then we can express the requested information for classification based on splitting by attribute A

$$Entropy_A(S) = \sum_{j=1}^{v} \frac{|S_j|}{|S|} Entropy(S_j) \tag{4.2}$$

After expressing the entropy for attribute A, we can compute the information gain after splitting by attribute A

$$Gain(A) = Entropy(S) - Entropy_A(S) \tag{4.3}$$

The ID3 algorithm chooses attribute A with the *highest* value of $Gain(A)$. Alternatively, one can choose attribute A with the lowest value of $Entropy_A(S)$. This approach states that value of attribute A with smaller probability of occurrence carries more information than value of attribute with higher probability of occurrence [14].

Disadvantage of this method is that the $Gain(S)$ depends on the number of unique values in attributes. The more values an attribute contains the lower the value of $Entropy_A(S)$ is and the higher the value of $Gain(S)$ is [24].

C4.5 and C5.0

Both variants improves the ID3 algorithm with a new way of computing the information gain. The *C4.5* algorithm uses *Gain Ratio* as the splitting criterion. It is a normalized information gain and is defined as

$$SplitEntropy(A) = -\sum_{j=1}^{v} \frac{|S_j|}{|S|} \log_2(\frac{|S_j|}{|S|}) \tag{4.4}$$

$$GainRatio(A) = \frac{Gain(A)}{SplitEntropy(A)} \tag{4.5}$$

The algorithm chooses the attribute with the highest value of *GainRatio(A)* [24].

This algorithm was improved by improvements in misclassification costs, cross-validation and ensembling and it is called *C5.0* or just *C5*. It also significantly improved the speed of building the trees and built them with fewer splits while maintaining the same accuracy [14].

Summary

According to [14] decision trees are among most popular classification methods. Since they are considered easy to understand – they are made up of sequence of if-then-else rules, which are more transparent for regular people than mathematical formulas. Decision trees are easy to build and are scalable in contrast to other types of classifiers. They have the ability to work with both numerical and categorical variables (some classification models request only numerical (neural networks) or only categorical (naive bayes)).

Sometimes, decision trees are likely to create over-complex trees that do not generalize the data well – this is called *overfitting*. In such cases, we could use techniques as pruning or setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree to avoid overfitting [47].

4.2.2 Ensemble methods

They use multiple learning algorithms to obtain better predictive performance that could be obtained from any of the constituent learning algorithms alone. A machine learning ensemble consists of only a finite set of alternative models, but typically allows for much more flexible structures to exist among those alternatives [4]. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve the ability to generalize over a single estimator. We distinguish two families of ensemble methods [47]:

Averaging (bagging) methods - their key principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator due to reduced variance. Examples: bagging, forests of randomized trees, extremely randomized trees.

Boosting methods - in contrast to previous methods, base estimators are built sequentially and one tries to reduce the bias of combined estimator. Key principle is to combine several weak estimators to create a powerful ensemble. Examples: AdaBoost, Gradient Boosting.

Random Forests

Despite decision trees are very popular classifiers, generally, with quite good results, they might overfit to their training dataset. Random forests correct this problem of decision trees. Random forests are ensemble learning method for classification (also for regression) and other tasks, that operate by constructing multiple decision trees at training time and outputting the class that is the mode of the classes [47].

Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training dataset, with the goal of reducing the variance [35]. This comes with the cost of small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Random forests applies bagging to decision trees. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of training samples with corresponding outputs $Y = \{y_1, y_2, \dots, y_n\}$, bagging repeatedly (M times) select a random sample with replacement of X and fits trees to these samples; for $m = (1, \dots, M)$

1. Sample, with replacement, n training samples from X, Y; call these X_m, Y_m.
2. Train a decision tree f_m on X_m, Y_m.

After training, predictions for unseen samples X' are determined by taking the majority of votes of all trees.

While the outputs of a single tree are highly sensitive to noise in training samples, the average of many trees is not if the trees are not correlated. Simply training many trees on a single training set might give strongly correlated trees; bagging is a way of de-correlating the trees by showing them different training sets. This leads to better performance because it decreases the variance of the model without increasing the bias.

The approach above describes the original bagging algorithm for trees. Random forests [18] differ in only one way: they use a modified tree algorithm that selects, at each candidate split in the learning process, a *random subset of the features*. This process is sometimes called *feature bagging*. Typically, for classification problem with N features, $\sqrt{N}$ features are used in each split [35].

Extremely Randomized Trees

Sometimes called *ExtraTrees*. They add a further step of randomization to random forests. They are trained like an ordinary random forest, but additionally the top-down splitting in the tree learner is randomized also. Instead of computing the locally optimal feature/split combination (based on entropy or Gini index[1]), for each feature a random value is selected for the split. This value is selected from the feature's empirical range (in the tree's training set i.e. the bootstrap sample) [32].

AdaBoost

Boosting is a general method for improving the performance of any given learning algorithm [51]. AdaBoost was first introduced by Freund and Schapire [43] as first algorithm solving many of the practical difficulties of early boosting algorithms. It is able to create a strong classifier from several weak classifiers. Decision trees are the most commonly used machine learning algorithms used with AdaBoost.

AdaBoost takes as input a training set $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. Basic principle of this algorithm is to maintain a set of weights over this training set [51]. Initially, all weights are set equally, but after each round the weights of misclassified samples are increased so that the weak learner is forced to „focus" on these samples in the training set. We take each sample in the training set and assign it a weight:

$$w(x_i) = \frac{1}{N} \tag{4.6}$$

where x_i is the ith training sample and N is the number of training samples. Only binary classification problems are supported so each weak classifier make a decision and outputs

[1] *Gini index* - eliminates the logarithm in the ID3's gain index and uses a method based on probability.

one of the two classes. Then, we can compute the misclassification rate:

$$E = \frac{(N - C)}{N} \tag{4.7}$$

where E stands for error (misclassification rate), N is the total number of training samples and C is the number of samples classified correctly by the model. We modify this equation to use weighting of training samples:

$$E = \frac{\sum w(i) * E_t(i)}{\sum w(i)} \tag{4.8}$$

where E is a weighted average of misclassification rates, $w(i)$ is the weight of i$^{\text{th}}$ training sample and E_t is the i$^{\text{th}}$ classification error which is defined as:

$$E_t(i) = \begin{cases} 0 & \text{correctly classified sample} \\ 1 & \text{misclassified sample} \end{cases} \tag{4.9}$$

Then, a stage value is calculated for the model to provide weighting for any prediction the model makes. The stage value is calculated as follows

$$s = ln(\frac{1 - E}{E}) \tag{4.10}$$

where s is the stage value used to weight predictions of the model and E is the misclassification error of the model. The stage value adjusts the weights so that more accurate models have more contribution to the final prediction.

The training weights are updated in such a way that incorrectly classified instances gain more weight than correctly classified samples

$$w = w * e^{s * E_t} \tag{4.11}$$

where w is the weight of the training sample, s is the misclassification rate for the weak classifier and E_t is the error the weak classifier made predicting the output variable for the training sample. This has the effect of increasing the weight if the weak classifier incorrectly classified the training sample. In the case of correct classification the weight will not be changed.

Weak models are added sequentially, trained using the weighted training data. The process continues until a pre-defined number of weak learners have been created or no further improvement can be made on the training samples. Once completed, we get a pool of weak learners each with a stage value. Predictions are made by calculating the weighted average of the weak classifiers.

For a new input instance, each weak learner assigns a prediction value. Predicted values are weighted by each weak learners stage value. The prediction for the ensemble model is taken as a the sum of the weighted predictions. If the sum is positive, then the first class is predicted, if negative the second class is predicted.

Stochastic gradient boosting

AdaBoost and other related methods were recast in a statistical framework first by Breiman calling them ARCing algorithms (Adaptive Reweighting and Combining) [17]. This framework was further developed by Friedman and called Gradient Boosting Machines [**?**] and later become just gradient boosting. Gradient boosting involves three elements:

1. a loss function to be optimized,
2. a weak learner to make predictions,
3. an additive model to add weak learners to minimize the loss function.

A loss function must be differentiable and it heavily depends on the type of problem being solved which function to use. For example, regression may use squared error and for classification there is logarithmic loss function. Gradient boosting is a generic framework which can use any differentiable loss function.

Such as in the case of AdaBoost, gradient boosting also uses decision trees as weak learners. Trees as constructed in a greedy manner, choosing the best split points based on scores such as Gini index [20]. At the beginning, very short decision trees were used that only had a single split point called *decision stump*. Larger trees can be used generally with 4-8 levels. In order to ensure that weak learners remain weak enough, we have to constrain them in specific ways − maximum number of layers, nodes, splits or leaf nodes.

Trees are added one at a time. Existing trees in the model are not changed. The goal of this algorithm is to minimize the loss while adding trees. Neural networks use gradient descent to minimize the weights of the network. In this case, we have weak learners − decision trees. After calculating the loss (error), we add another tree to the model to reduce the loss to perform the gradient descent procedure. This action is done by parametrizing the tree, then modify the parameters of the tree and move in the right direction by reducing the residual loss. The output of the new tree is added to the output of the existing sequence of trees to improve the final output of the model.

Despite their very promising performance in variety of problems [45], gradient boosting is a greedy algorithm and is likely to overfit and dataset quickly. Due to this problem, several improvements were suggested. It can benefit from regularization methods that penalize parts of the algorithm and improve the performance of the algorithm by reducing overfitting [20]. Most significant improvements are:

Tree constraints - there is number of ways to constrain the trees − learners should have skill but remain weak. There is a trade-off between number of trees and how much they are constrained. The more constrained the tree creation is, the more trees the model need. If less constrained tree creation, the fewer trees we need.

Shrinkage - the predictions of trees are added together sequentially. The contributions of trees to this sum can be weighted to slow down the learning process − this weighting is called *shrinkage*. As well as learning rate in stochastic optimization, shrinkage reduces the contribution of each tree and leaves space for future trees to improve the model [31].

Random sampling - we can reduce the correlation between the trees in the sequence by allowing trees to be greedily created from subsamples of the training set. This variation of boosting is called stochastic gradient boosting [20].

Gradient Boosting is a highly effective method both for regression and classification [26, 47]. At Kaggle competitions[2], gradient boosting is one of best performing model across competitions [33]. They are used in a variety of areas including web search ranking and

[2] *Kaggle* is a platform for predictive modelling and analytics competitions in which data miners compete to produce the best models for predicting and describing the datasets uploaded by companies and users.

ecology [47]. These methods can work with mixed type data. The are quite robust to outlier values via robust loss functions. Disadvantage of gradient boosting methods is the inability to parallelize them due to their sequential nature [47].

4.2.3 Support Vector Machines (SVMs)

SVMs are a set of supervised learning methods used both for classification and regression of linear and non-linear data. Linear SVM is used for binary classification and the classes should be linearly separable and non-overlapping.

Assume we have a training set containing n-dimensional input vectors. Each of these data points belongs to one of two classes. Our goal is to separate them with a *n-1* dimensional *hyperplane* − this is called a *linear classifier*. There are, of course, plenty of such classifiers that may satisfy this property. However, we want to find a hyperplane with *maximum margin/separation* between two classes − the distance between the hyperplane and the nearest data point is maximized. If such a hyperplane exists, it is called the *maximum-margin hyperplane* and this linear classifier is known as a *maximum margin classifier* [11, 16]. Term *support vectors* are those vectors (data points) which lie nearest to this hyperplane.

Later a modification of the maximum margin called *soft margin* was introduced [15]. In this case, some misclassifications are allowed but they penalize the function to minimize with a factor that is proportional to a parameter C and the distance of the mistakes to the margin. With other words, SVM maximizes the margin between classes while minimizing the penalization term weighted by parameter C which is a boundary for the number of misclassifications.

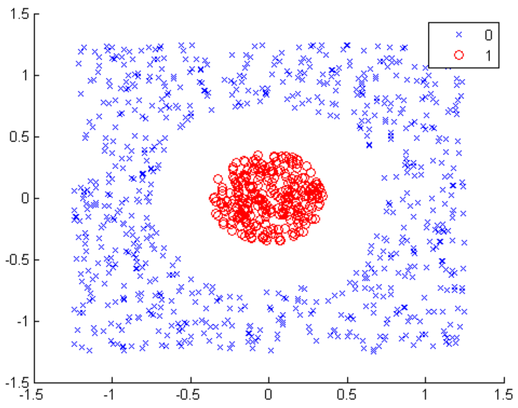


Figure 4.3: In this case, linear SVM is unable to find a hyperplane separating the two classes even with soft margin. We must perform a so called *kernel trick* − we need to solve the linear separation problem in a higher dimensional feature space. Image taken from [38].

Kernel trick

Consider data shown in figure 4.3. We can clearly see that data can be separated by a circle. This is where we use an approach called *kernel trick* which says that a data set is linearly inseparable in $\mathbb{R}^N$ might be linearly separable in a higher dimension $\mathbb{R}^M$. We need a mapping function ϕ that lifts the data set X to a higher-dimensional data set X' such that X' is linearly separable. In that case, we are able to train a linear SVM in X' to find a hyperplane $\vec{w}$ separating the data points in X'. Then, if one projects the hyperplane $\vec{w}$ from $\mathbb{R}^M$ back into the original input space $\mathbb{R}^N$ one will get a non-linear hyperplane [37]. Figure 4.4 shows an application of a kernel trick on the example 4.3.

However, the only problem with this approach is the stepwise increase of dimensions from $\mathbb{R}^N$ to $\mathbb{R}^M$ – with the increasing M we are getting into serious computational and memory problems [37]. Fortunately, it turned out that there is no need to explicitly work in higher dimensions at training or testing time. In [36] is shown that during training, the process of finding the maximum margin only uses the training inputs to compute *pair-wise* dot products[3] $\vec{x_i} \bullet \vec{x_j}$ where $\vec{x_i}, \vec{x_j} \in \mathbb{R}^N$. This knowledge is the key to perform the kernel trick.

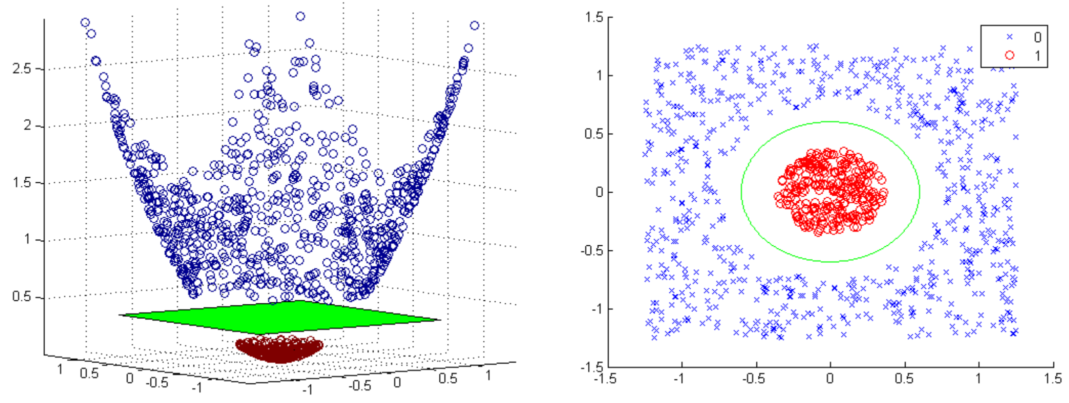


Figure 4.4: These images are capturing the solution of the linear separation problem we faced with data in 4.3. We had to separate the data points in higher-dimension $\mathbb{R}^3$ (a linear boundary now became a plane instead of a line). This is shown on the left picture. All we need to do is perform an inverse mapping of the plane back into the original input space – getting a non-linear hyperplane (shown in green circle) separating the data points in the original input space. Image taken from [38].

Kernel functions $K(\vec{v}, \vec{u})$ are functions, which given two vectors $\vec{v}, \vec{u} \in \mathbb{R}^N$, implicitly compute the dot product of $\vec{v}$ and $\vec{u}$ in a higher dimension $\mathbb{R}^M$ *without explicitly transforming $\vec{v}$ and $\vec{u}$ to $\mathbb{R}^M$*. Kernel functions solves the computational bounds and memory problems we would have faced by explicit calculations in higher dimensions. Therefore, we are able to efficiently learn non-linear decision boundaries for SVMs. There are several kernel functions available [37], we mention two popular kernels:

- **Polynomial**
$$K(\vec{v}, \vec{u}) = (\gamma \vec{v} \bullet \vec{u})^{dimension} \tag{4.12}$$
  where parameter γ controls the characteristic distance of the influence of a single data point [38]. Figure 4.5 shows tuning of parameters γ and C.

- **Sigmoid**
$$K(\vec{v}, \vec{u}) = tanh(\kappa \vec{v} \bullet \vec{u} + c) \tag{4.13}$$
  where $\kappa > 0$ and $c < 0$.

[3]The dot product of 2 vectors $\vec{u} = (u_1, u_2, \ldots, u_n)$ and $\vec{v} = (v_1, v_2, \ldots, v_n)$ is defined as $\vec{x_i} \bullet \vec{x_j} = \sum_{i=1}^{n} u_i v_i$

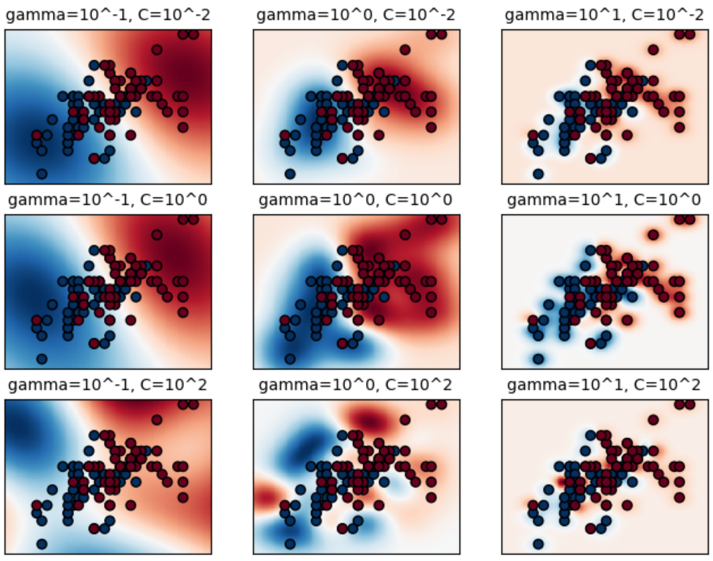


Figure 4.5: This figure captures the hyper-parameter tuning of SVM classifier.

Summary

SVMs are used for text classification tasks such as spam detection or sentiment analysis. They are also commonly used for image recognition challenges and they are also used in many areas of handwritten digit recognition such as postal automation services [16]. SVMs are very effective in high dimensional spaces thanks to the kernel trick described above. They use a subset of training samples in the decision function (support vectors), so are they memory efficient.

In order to use SVMs one have to experimentally choose what kernel function and what combination of hyper-parameters (γ, C) best fit to his problem. Unfortunately, kernel functions might be quite sensitive to over-fitting the model selection criterion [25]. Additionally, SVMs do not directly provide probability estimates – calculated using an expensive 5-fold cross-validation [47].

4.2.4 K-Nearest Neighbors (k-NN)

Described firstly in 1951 in this paper [30] (a work that was republished later in 1989). This algorithm is a so-called „lazy learner“ since the training data itself is considered for the model [14]. The „K“ in the name of this algorithm stands for the number of surrounding data points needed for prediction. The key to this method is the computation of the distance between the data point and its K nearest neighbors.

Since training inputs are represented with vectors of size n of numeric attributes input vectors are point in a n-dimensional space. For each testing input the classifier projects the testing data point into the same n-dimensional space and finds K nearest data points from the training set and according to the majority of „votes“ of the nearest data points, classifies the testing data point into the majority class. The bigger the number K is, smoother predictions we get. For binary classification, it is common to use odd count of nearest neighbors to avoid ties in voting [14]. The number of neighbors are recommended to search iteratively [14] since its hard to tell which number will suit best for our task. The proximity of data points is defined by distance metrics. The primary and most commonly used distance metric for k-NN is Euclidean distance [34]. Given two vectors $U = (u_1, u_2, \ldots, u_n)$ and $V = (v_1, v_2, \ldots, v_n)$, the Euclidean distance is defined as:

$$d(U,V) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2} \tag{4.14}$$

There are some other distance metrics which are sometimes found in software libraries – Manhattan, Hamming, Minkowski and Mahalanobis distance [14].

Data preprocessing

Since k-NN computes distances between vectors of numeric attributes of size n, it is sensitive to features scaling. We have to normalize the values of all attributes in the training set (then also the attributes of the testing set on equal ranges). We can use *min-max scaling* [57] which is defined as follows:

$$v^{new} = \frac{v^{old} - min_A}{max_A - min_A}(max_A^{new} - min_A^{new}) + min_A^{new} \tag{4.15}$$

where v^{old} is the old and v^{new} is the new value of numeric attribute A; max_A and min_A is the maximum and minimum values of attribute A; max_A^{new} and min_A^{new} are new maximum and minimum values of attribute A. The advantage of min-max scaling is that we will have guaranteed to have bounded maximum and minimum values. Another option is *z-score* normalization:

$$v^{new} = \frac{v^{old} - \mu_A}{\sigma_A} \tag{4.16}$$

where v^{old} is the old and v^{new} is the new value of numeric attribute A; μ_A is the average and σ_A is the standard deviation of attribute A. This method is most appropriate if the data is normally distributed (because the mean and standard deviation used in the equation assume normal distribution). Its interpretation can be as follows: each unit refers to 1 unit of standard deviation from the mean [14]. Before performing normalization, we must be aware of heavily skewed data and outlier values.

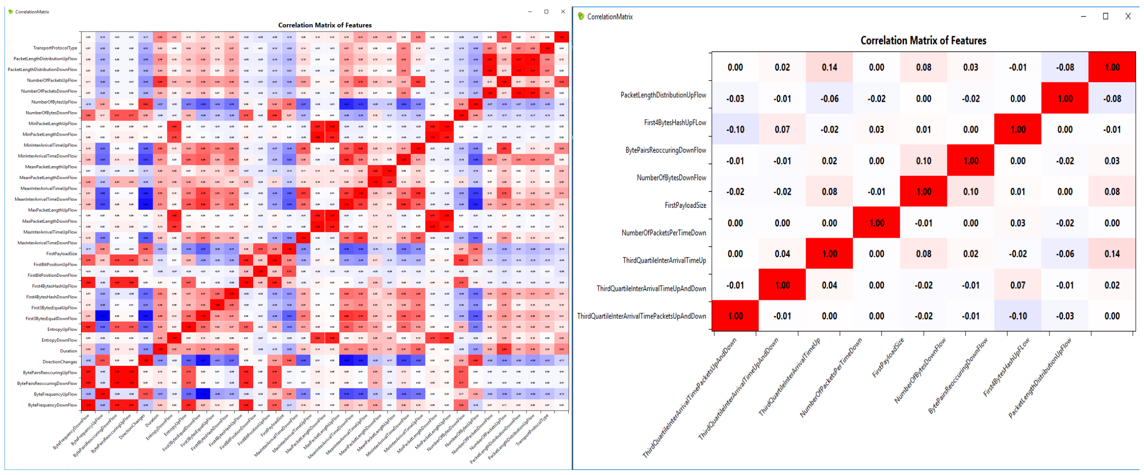
k-NN requires all inputs to be numeric. Categorical variables must therefore be transformed into numeric format. According to [23], we have 2 options:

label/integer encoding - used for ordinal variables. Each unique category value is assigned an integer value. Algorithms like decision trees or random forests can work with this encoding. This process is also easily reversible.

one-hot encoding (dummy variables) - for non-ordinal categorical variables the label encoding is not enough. This approach adds a new binary variable for each unique numeric value. This method has the advantage that the result is binary rather than ordinal and everything sits in an orthogonal vector space. However, the disadvantage is that for high cardinality, the number of features can exponentially increase.

One of the challenges, generally, with distance-based algorithms is the number of inputs used in building a model. With the increasing number of inputs the number of records needed to populate the data space increases exponentially. If the size of the space increases without increasing the number of records to populate the space, records could be closer to the edge of the data space than they are to each other, rendering many, if not all, records as outliers. This is the *curse of dimensionality* [14, 57].

One solution to this problem is to keep dimensionality low – include only a few tens of inputs in k-NN models. If too many irrelevant features are included, the distance will be dominated by noise, thus reducing the contrast in distances between the target variable values. One could perform correlation analysis[4] and exclude inputs that are highly correlated with other inputs in the data – in a case of high correlation one could exclude one of the two correlated variables, not both. However, reducing dimensionality too much will result in poorer predictive accuracy in models.



	0.00	0.02	0.14	0.00	0.08	0.03	-0.01	-0.08	1.00
PacketLengthDistributionUpFlow	-0.03	-0.01	-0.06	-0.02	0.00	-0.02	0.00	1.00	-0.08
First4BytesHashUpFLow	-0.10	0.07	-0.02	0.03	0.01	0.00	1.00	0.00	-0.01
BytePairsReoccuringDownFlow	-0.01	-0.01	0.02	0.00	0.10	1.00	0.00	-0.02	0.03
NumberOfBytesDownFlow	-0.02	-0.02	0.08	-0.01	1.00	0.10	0.01	0.00	0.08
FirstPayloadSize	0.00	0.00	0.00	1.00	-0.01	0.00	0.03	-0.02	0.00
NumberOfPacketsPerTimeDown	0.00	0.04	1.00	0.00	0.08	0.02	-0.02	-0.06	0.14
ThirdQuartileInterArrivalTimeUp	-0.01	1.00	0.04	0.00	-0.02	-0.01	0.07	-0.01	0.02
ThirdQuartileInterArrivalTimeUpAndDown / ThirdQuartileInterArrivalTimePacketsUpAndDown	1.00	-0.01	0.00	0.00	-0.02	-0.01	-0.10	-0.03	0.00
	ThirdQuartileInterArrivalTimePacketsUpAndDown	ThirdQuartileInterArrivalTimeUpAndDown	ThirdQuartileInterArrivalTimeUp	NumberOfPacketsPerTimeDown	FirstPayloadSize	NumberOfBytesDownFlow	BytePairsReoccuringDownFlow	First4BytesHashUpFLow	PacketLengthDistributionUpFlow

Figure 4.6: This figure shows an example of correlation analysis performed in a school project. We were identifying application protocols in network flows using machine learning. The left image shows the correlation matrix of original features of the network flow data. Red and blue colored points in the correlation matrix represent positive and negative correlation of two feature variables. The saturation of the color indicates the strength of the relationship. Our goal is to iteratively remove those features which have strong relationship with other features. Right image shows the leftover independent features. In such a way we were able to reduce the dimensionality therefore we were able to build simpler classifiers with preserved accuracy.

Summary

Since k-NN algorithm is a distance-based algorithm assigning each attribute an equal weight, this algorithm might have low performance for noisy data or irrelevant attributes, however it is insensitive to outliers. It is quite simple algorithm to explain and interpret the results. k-NN can be used both for classification and regression.

[4] *Correlation analysis* is a method of statistical evaluation used to study the strength of a relationship between two (or more), numerically measured, continuous variables [57]. Correlation can be used to help better understand associations in data, make prediction from data [28] and also to help to eliminate highly correlated features used for predictions – severe multicollinearity is a problem because it can increase the variance of the predictions and make them very sensitive to minor changes in the model i.e. standard errors are likely to be high [7]. Figure 4.6 shows an example of feature variable reduction by correlation analysis performed in a school project.

Despite giving quite good result in different fields, it has some disadvantages. k-NN has high memory requirements since it stores the training data. Predictions stage might be slow especially for bigger K values. This algorithm requires only numerical data, but we have the necessary tools to transform categorical values into numerical. Last, k-NN is also sensitive to unscaled data – one have to consider normalizing the data using techniques described in this subsection.

4.2.5 Naive Bayes

These methods are a set of supervised learning methods based on *Bayes' theorem*. Due to the assumption of independence between every pair of feature variables they are called „naive". Bayes' theorem states the following:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{4.17}$$

where

- $P(A)$[5] is the probability that A is true,
- $P(B)$ is the probability that B is true,
- $P(A \mid B)$[6] is the conditional probability that A is true given that B is true,
- $P(B \mid A)$ is the conditional probability that B is true given that A is true.

Assume a data sample $X = (x_1, x_2, \dots x_n)$ which has to be classified into 1 of classes $C_1, C_2, \dots C_n$. We classify X into C_i if $P(C_i \mid X)$ returns the biggest likelihood. Because

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)} \tag{4.18}$$

where $P(X)$ is a constant, we are looking for the maximal $P(X \mid C_i)P(C_i)$ [24].

First, we compute the likelihood of class C_i

$$P(C_i) = \frac{|s_i|}{|S|} \tag{4.19}$$

where s_i is the set of samples if class C_i and S is the set of training samples.

Second, we compute the product of likelihoods of each variable in the sample X

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) \tag{4.20}$$

Finally, we classify the sample X to that class C_i for which $P(X \mid C_i)P(C_i)$ has maximal value.

[5] This is the form usually presented in the litarature, but variables A and B are just placeholders for actual conditions one find in the data.

[6] Note that $P(A \mid B)$ is, in general, not equal to $P(B \mid A)$ because these conditional probabilities expect different subsets of data.

Summary

Naive Bayes models are easy to implement. These models require categorical data – we must perform binning on numerical data to obtain categories. We might get poor classification accuracy in case of high correlation between feature variables. Naive Bayes models assume feature independence [14] hence they are called naive. In spite of their over-simplified assumptions, they have very promising results in many real-world problem especially in document classification or spam filtering [47]. In comparison with other machine learning models, they can be extremely fast.

The difference between Naive Bayes models and other machine learning models is that Naive Bayes models consider only one class at a time. For each class they compute a likelihood and the class with the largest probability is assigned to the sample.

4.2.6 Neural Networks (NN)

They are computer systems inspired by biological NNs that represent animal brains. These systems are able to learn (improve their performance) tasks by considering examples, generally without task-specific programming [42]. They are universal tools of machine learning – broad spectrum of data-intensive applications such as speech recognition, sales forecasting, stock market prediction, fraud detection etc. [8].

A neural network is based on a collection of connected units called *neurons*. Each connection between neurons called *synapse* can transmit signals from one to another. A neuron that receives the signal can process it and then send signal to another one which is connected to it. The signal is a real number and it is called *weight* of the synapse. At the beginning, weights are randomly initialized and they are adapted during the learning process. Each neuron has a *transfer function* (also called *threshold functions* [15]) and an *activation function* (also called *squashing function* [14]). The transfer function computes the net input of a neuron:

$$net = \sum_{i=1}^{n} w_i x_i + w_0 x_0 \tag{4.21}$$

where w_0 is the intercept value to make the model more general; it is generally modeled as the weight coming from an extra *bias unit* x_0, which is always $+1$. The activation function defines the output of the neuron. An example of an activation function might be the following function:

$$y = \begin{cases} 1 & net > 0 \\ -1 & \text{otherwise} \end{cases} \tag{4.22}$$

This function defines the output of a neuron with respect to the net input computed by the transfer function.

Perceptron

A single neuron unit is also called *perceptron*. Perceptrons adapts as follows: *change the weights by an amount proportional to the difference between the desired output nad the actual output*:

$$\vec{w}(0) = random \tag{4.23}$$

$$\vec{w}(i) = \vec{w}(i-1) + \mu(\vec{d}(i) - \vec{y}(i))\vec{x}(i) \tag{4.24}$$

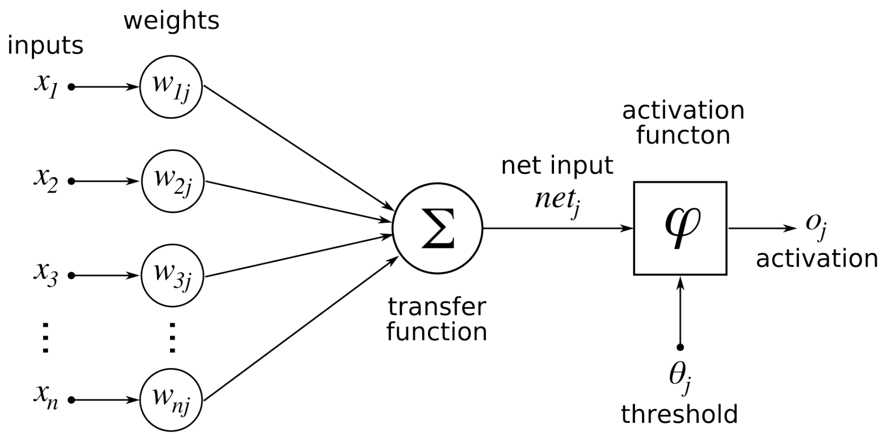


Figure 4.7: This figure represents a single neuron. A neuron is composed of an input vector and a corresponding weight vector. The net input is calculated using the transfer function 4.21 (there are also other transfer functions [42]). Then, according to the net input and the given threshold the activation function computes the output of the neuron. o_j (we use y further in the text) represents the output of the neuron which is transmitted to the successor neuron in a form of signal. Each neuron is simply an equation $y = \varphi(w_0x_0 + w_1x_1 + \ldots + w_nx_n) = \varphi(\sum_{i=0}^{N} w_ix_i)$ [14].

where μ is the learning rate – it is a real number in range $0 < \mu < 1$ [56] and it determines how quickly or how slowly to update the weights. Usually, one can start with a large learning rate, and gradually decrease the learning rate as the training progresses.

Perceptrons are able to adapt only problems with linearly separable input vectors. Otherwise these networks do not converge [56] – they are not able to learn the XOR problem as its data points are linearly inseparable. We have to add an additional layer (called *hidden* layer) to the network. This helps to create more complex non-linear hyperplane, which can separate these data points – this is shown in figure 4.8.

Multilayer perceptron (MLP)

In an MLP, neurons are organized in layers in a fully connected, feedforward network. A key concept in MLP is the usage of continuous activation functions e.g. sigmoid function:

$$y = \frac{1}{1 + e^{-net_j}} \tag{4.25}$$

Continuous functions are differentiable which means that we can compute derivatives, an essential property of neurons so that the learning algorithms used to train NNs can be applied [14]. For classification modeling, the output layer neurons usually have the same sigmoid activation function already described, one that transforms the neuron to a value between 0 and 1. For continuous-valued prediction, however, implementations of NNs most often use "linear" activation function, which means that no transformation is applied to the linearly weighted sum transfer function.

After the network calculates the output for a given input, the error is calculated by a *loss or cost function*. For classification, lots of implementations use cross-entropy as the cost function instead of squared error due to more appropriate measure of error for a binary classification problem [14].

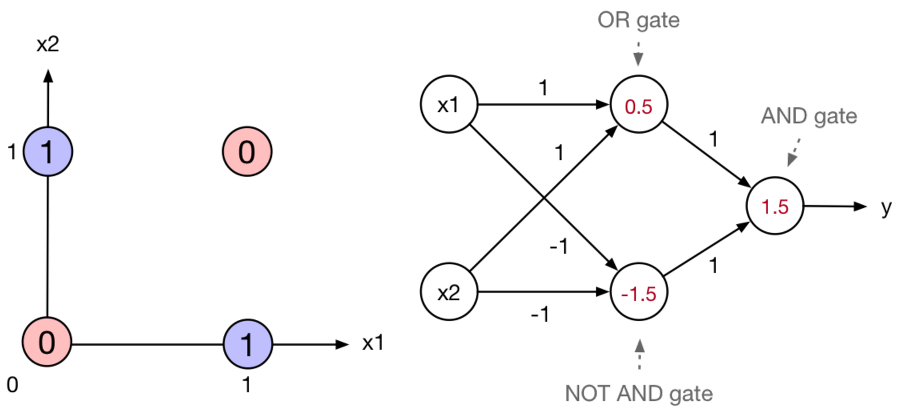


Figure 4.8: This is an example of a neural network learning the XOR problem. You can see that the neuron labeled as *OR gate* ensures that at least one of the inputs is on (numbers inside neurons labeled as red are threshold values – the net input must be greater that this value in order to activate the output), while the *NOT AND gate* ensures that both are not on at the same time. The output from these neurons pass through a final AND gate that makes sure that both conditions are met. This problem was solved by adding a hidden layer of neurons, otherwise simpler NNs did not converge to the result. In other words, they could not separate the data points in the XOR problem as they are linearly inseparable. Neural network was taken from [9].

MLP uses an algorithm for learning called *gradient descent* [53]. It is an optimization algorithm for finding the minimum of the loss function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. In order to use this optimization algorithm, activation function must be differentiable - this is the reason why MLP uses continuous activation functions. Sigmoid satisfies this requirement. Formula for weight adaptation can be written as:

$$\Delta\vec{w} = -\mu\nabla E \tag{4.26}$$

There are several variants of gradient descent algorithm [39] which differ in how much data we use to compute the gradient of the loss function:

- *batch* - weights are updated after processing all inputs of the training set.
- *mini-batch* - weights are updated after a given number of inputs of the training set.
- *stochastic* - weights are updated after every processed input of the training set

All gradient descent approaches share the same workflow [56]:

1. Pick a sample from the training set
2. Input vector is set as input of neurons of input layer.
3. Layer by layer are calculated the outputs of neurons (feed forward); output of last layer is the output of network.
4. Calculate the error E of the output.

5. Calculate gradient of the error function ∇E.

6. Update all weights of the network.

An *epoch* occurs when every record in the training data has passed through the neural network and weights have been updated. Training a neural network can take even thousands of training epochs [14] – hence the reputation for taking a long time to train. The training time on commonly available computer hardware is acceptable for most problems.

Summary

NNs require numerical data and they can be very sensitive to feature scaling – we need to transform all features into numeric and perform normalization. NNs have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy – we might consider using an *autoencoder*[7]. They also require tuning a number of hyper-parameters such as the count of hidden neurons, layers and iterations. NNs have the reputation of being "black boxes" – it is not transparent why did NN predict such values. However, there are several ways of determining the most important features of the input variables [14].

Generally, they are models with very promising performance in various fields, despite disadvantages. NNs are capable to learn non-linear models and have the ability to learn models in real-time (on-line learning). They can be used both for classification and regression. In some applications, NNs might outperform other statistical models [49].

4.3 Measuring model's performance

To determine what is a good model depends on the interests of the organization and is specified as the business success criterion. These criterions needs to be transformed to a predictive modeling criterion in order to use it for selecting candidate models [14]. We use measures of accuracy if we need highly accurate predictions. However, one may prefer to have more transparent model in order to better understand the predictions. In such cases, we use subjective measures instead which provides maximum insight. Some projects may use a combination of both so that the most accurate model is not selected if a less accurate but more transparent model with nearly same accuracy is available [14].

4.3.1 Confusion matrix and performance metrics

A confusion matrix provides a breakdown of classification errors through a table of actual values and predicted values. Through this table we can reveal the types of errors the classifier makes as well as the two ways the model is correct. Confusion matrix defines four basic values:

True positives (TP) - cases when the model predicted *True* and the actual value is *True*,

True negatives (TN) - cases when the model predicted *False* and the actual value is *False*,

[7] *Autoencoder NN* is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs i.e. $y = x$ [1]

False positives (FP) - cases when the model predicted *True*, but the actual value is *False* (also known as *Type I Error*),

False negatives (FN) - cases when the model predicted *False*, but the actual value is *True* (also known as *Type II Error*).

There are various performance metrics, which are calculated from the confusion matrix. Here is a list of performance metrics which are often computed for a binary classifier:

Accuracy tells in overall, how ofter is the classifier correct. Defined as $\frac{TP+TN}{Total}$.

Misclassification Rate (Error Rate) tells in overall, how often is the classifier wrong. Defined as $\frac{FP+FN}{Total}$ or also $1 - Accuracy$.

True Positive Rate (Sensitivity/Recall) tells how often does the model predict $True$ if the actual value is $True$. Defined as $\frac{TP}{Actual\ True} = \frac{FP}{TP+FP}$.

False Positive Rate tells how often does the model predict $True$ if the actual value is $False$. Defined as $\frac{FP}{Actual\ False} = \frac{FP}{FN+TN}$.

Specificity tells how often does the model predict $False$ if the actual value is $False$. Defined as $\frac{TN}{Actual\ False}$ or also $1 - False\ Positive\ Rate$.

Precision tells how often is the model correct if it predicts $True$. Defined as $\frac{TP}{Predicted\ True}$.

Prevalence tells how often does the $True$ condition actually occur in data. Defined as $\frac{Actual\ True}{Total}$.

Positive Predictive Rate is very similar to precision, except that it takes prevalence is account. In cases where classes are perfectly balanced (when *prevalence* = 50%), this metric is equivalent to precision.

F Score (F-measure) is a weighted average of the true positive rate (recall) and precision. Defined as

$$F_1 = 2 * \frac{1}{\frac{1}{Recall} + \frac{1}{Precision}} = 2 * \frac{recall * precision}{recall + precision} \tag{4.27}$$

4.3.2 ROC curve

A ROC (receiver operating characteristic) curve is created by plotting the true positive rate (sensitivity) against the false positive rate at various threshold settings. It is the most commonly used way to visualize the performance of a binary classifier [40]. There are several things a ROC curve demonstrates [54]:

- closer the curve is to the diagonal of the ROC space, the less accurate the model is,
- closer the curve is to the left-hand border and the top border of the ROC space, the more accurate model is,
- it shows the trade-off between sensitivity and specificity (any increase of sensitivity will follow by a decrease in specificity).

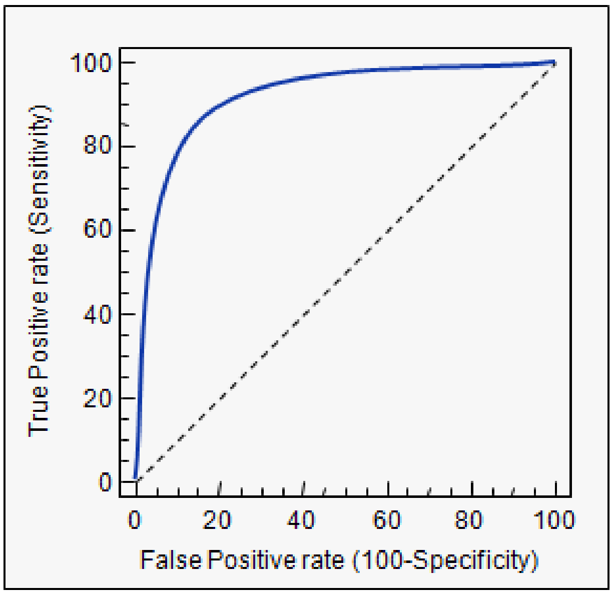


Figure 4.9: This figure shows a ROC curve; taken from [54].

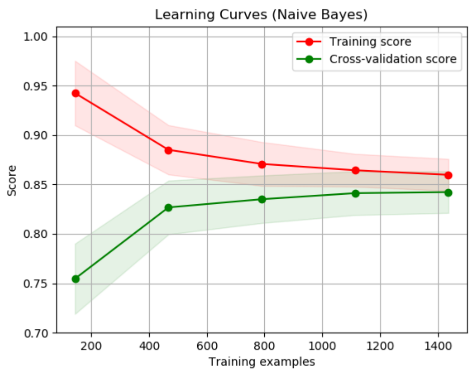
4.3.3 Learning curve

A learning curve captures the training and testing score of a classifier for varying number of samples. It is a handy tool to find out how much can we benefit by adding more training data and if the classifier suffers more from a variance error or a bias error [47]. Left picture of figure 4.10 captures a situation, when both the training and testing score converge to a value that is too low on increasing number of training samples − we will probably not benefit much from adding more training samples. However, if the training score is much greater that the testing score for the maximum number of training samples, adding more samples to the training will most likely increase the generalization power. On the right picture of figure 4.10 there is a case, when we could benefit from adding more training samples.

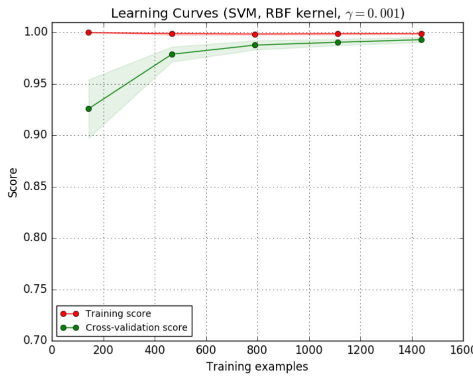
4.3.4 Model evaluation and hyper-parameter optimization

During candidate model selection process we would like to know how would a given model perform on unforeseen data. An approach called *cross validation* does not uses the entire data set when training. It removes some data from the training set before learning (validation data). Then, when training is over, we will test the performance of the model on validation data. This is the basic idea of cross validation [52]. There is few variants of cross validation:

Holdout method - this is the simplest cross validation. The data is divided into two parts called training and testing set. The model is trained only on the training set. The prediction model predicts the output of the testing set (it has never seen these data before). To evaluate the model the mean absolute test error is calculated. The disadvantage of this approach is that it might have high variance [52]. The evaluation might depend heavily on which data end up in the training set and which in the testing set, thus the evaluation may be significantly different depending on how the data split was performed.



(a) Converging to a low score



(b) Might increase generalization power by adding more samples

Figure 4.10: Images taken from [47].

k-fold cross validation - the data set is split into k subsets and the holdout method is repeated k times. Each iteration, one of the k subsets is used as the test set and the other *k-1* subsets are merged into the training set. The average error across all k iterations is calculated. The advantage of this approach in contrast with the holdout method is that it matters less how the data is split. Each data point gets to be in the test set exactly once and gets to be in the training set *k-1* times. The larger k is set the lower the variance of the resulting estimate will be. However, the algorithm has to be rerun k times from scratch, which means that it might be computationally expensive depending what model we are evaluating and the data set size. A variant of this method is to randomly split the data set into a training and test set k times. The advantage of this variant is that one can independently choose how large each test set is and how many iterations one average over.

leave-one-out cross validation - this approach is a special case of the previous one when K equals to the number of data points in the set. Therefore, the model is trained on all data except for one point and a prediction is made for that data point. As before the average error is calculated and used to evaluate the model.

Majority of models require definition of hyper-parameters[8]. Since we could not know what parameters yield the best resulting estimate for each model, we have to perform hyper-parameter optimization (or hyper-parameter tuning) [27]. We will use an algorithm called *grid search* (or *parameter sweep*). Grid search is simply an exhaustive searching through a manually specified subset of the hyper-parameter space of a model. This algorithm has to be guided by some performance metric, typically measured by cross validation.

As an example assume we want to use soft-margin SVM equipped with polynomial kernel function. This kernel function has at least 2 hyper-parameters that need to be tuned for good performance on unseen data: the regularization constant C and kernel parameter γ. For both hyper-parameters we define a set of values: $C \in \{10, 100, 1000\}$ and $\gamma \in \{0.01, 0.1, 0.2, 0.5, 1.0\}$. Grid search then trains an SVM with each pair $(C_i, \gamma_i) \in C \times \gamma$

[8] *Hyper-parameter* is a parameter of a model whose value is set before the learning process begins. For instance, consider the number of hidden layers in neural networks.

and evaluates their performance. Finally, the grid search algorithm returns the settings that achieved the highest score in the validation process.

The disadvantage of this method is the exhaustive search itself − if the parameter space is too large, grid search is not effective. In such cases we can use randomized search, which picks random combinations of parameters from the grid for a given maximum number of times.

4.4 Candidate model selection

In order to choose among classification models, we will first analyze the data. There are classification models requiring only categorical variables (Bayes) or only numerical variables (SVM or NN). We have to also analyze the hyper-parameters of models. After selecting a set of models which can be fit on our data, we will start a two phase process:

1. **Evaluation of models** in this phase we evaluate chosen models using cross validation. Since majority of models require setting of hyper-parameters, we use the grid search algorithm described above. After model evaluation, we collect the best evaluation scores of all models and then pick a subset of models with the highest scores.

2. **Candidate selection** - after the evaluation we simply test the best models on real data and analyze their results. High score at evaluation does not necessarily mean that the model will have such accuracy on unforeseen real data. Some models may overfit during the training which result in poor performance on real data. In this process we pick those models, which yield best results in the prediction of PGC status of projects.

Chapter 5

Data preparation

The organization uses various cloud-based systems to keep track of active projects. Each system collects a different type of information — one tracks all the requirements, another collects issues and defects. There is also a platform for storing information about hours each employee worked at given day. Since we wanted to collect all information we had available about the projects, we had to create data collector scripts for all of these systems. These scripts are storing information into a database.

In this thesis we have been working with Python language. It has very powerful modules for data related tasks such as *numpy*[1] and *pandas*[2].

5.1 Data sources

We have decided to use only 3 data sources on the basis of completeness and quality of data in each system to cover as many projects as possible. Unfortunately, there are no strict rules which system to use in the whole organization. We had to exclude systems, which provided incomplete information or which were not used uniformly among businesses. At the end we left off with 3 data sources from which 1 is used only for joining the other 2 systems:

System for project management - this system contains the majority of information we have about projects. it contains:

- basic information such as *project size*, *business unit*, *country*, *city* etc.,
- information about people responsible for given project such as *owner*, *team leader*, *employees* etc.,
- information about project planning such as *baseline dates* and *planned dates*,
- financial information
- ...

System for reporting hours - this system records employee hours worked on projects.

System for additional project management information - this system is used only for linking the previous systems together. Otherwise it does not contain any valuable information we could use for prediction.

[1] *Numpy* Python module available at http://www.numpy.org.
[2] *Pandas* Python module available at https://pandas.pydata.org.

5.2 Data separation

At a very high level we can recognize 2 businesses in the organization – *thermostats* and *security panels*. Projects of these businesses can be divided into 2 groups by their size – *major* and *minor* projects. Due to this fact, we had to separate the projects by business unit and size in the prediction analysis since their data can differ sufficiently. The development process of projects within these groups are different so differs their data. This separation should ensure the preservation of projects' typical characteristics.

Since we want to use supervised learning to make predictions, we distinguish training and testing data. For training and model evaluation we take all projects ended between 2014 and 2017. Then, we will test our models on projects which has not ended yet.

5.3 Data preprocessing

Real world data in raw form cannot be used in prediction. Data have to be cleaned and transformed into a form, which meets the requirements of prediction models. Raw data we collected contain missing data, irrelevant columns and there are also significant differences between ranges of numerical columns (e.g., number of hours worked vs. total project investment). We will describe the process of data preprocessing, which we use for all groups of data. This process starts right after we exported the raw data from systems and inserted them into a database. Figure 5.1 shows the preprocessing workflow we used. Preprocessing steps will be described in more detail in this section.

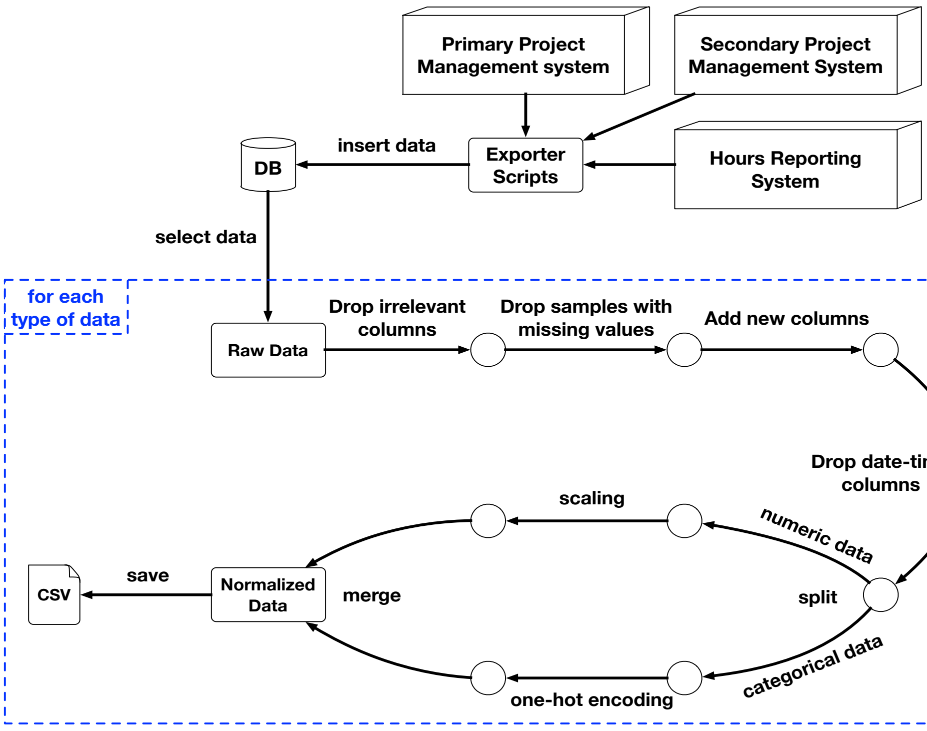


Figure 5.1: This figure captures the process of data preprocessing. We implemented several Python scripts that export data from selected systems and then store them in a database. We consider these data raw. According to the steps described, we gradually transform the data into a form that suits to prediction models.

5.3.1 Cleaning and column construction

First, we retrieve the raw data from the database via SQL select scripts. Second, we have to remove some columns from the dataset containing project specific information – ID, name, business unit, size. We also need to remove information related directly to the prediction target, i.e., baseline planned and actual date of phase gate 5. This information is not removed entirely, we keep them until the end of preprocessing process.

Third, we have to deal with missing data. The raw dataset contains some columns with lots of missing data. Unfortunately, the best thing we can do is set a threshold value (we set 33%) according to which we decide whether we drop that given column. If a given column has more than 33% of its rows empty, we drop that column. These columns are irrelevant for the prediction because models cannot handle empty values effectively.

Aside from columns with missing data, raw dataset contains several irrelevant columns for the prediction. These columns contain information such as:

- project IDs in systems,
- project description,
- project owners and creators,
- custom fields containing user inputs (majority of these columns contain sparse data),
- information available after phase gate 5.

We removed these columns from the dataset because they do not contain any valuable information for the prediction. In the case of phase gate 5 related columns, we have to remove them either, because we want to predict whether projects will miss or meet the phase gate 5 deadline. We cannot provide this information to our models as they will be used before the phase gate 5 will end.

Then, we create new columns which will aggregate information from few columns in the dataset. We create these new columns:

- **Project Length** - this information is easily calculated from project start and end date.
- **Phase {1-4} Length** - like the previous column, lengths of project phases are calculated as difference of start and end dates of each project phase.
- **Phase {1-4} Miss** - this column contain boolean value and is calculated according to PGC metric specification see 3.1.

After creating new columns, we can remove all columns containing date-time information. We will not lose any information, because we have already aggregated them into columns described above.

5.3.2 Transformations

At this point, we have to deal with *different ranges of numerical columns* and appropriate *representation of categorical columns*. The problem with different ranges of numerical columns is that they are not suitable for distance-based prediction models. A model could prioritize columns with more extensive ranges and neglect ones with small spreads. This is the reason, why we need to scale and center (having mean $= 0$ and variance > 1) all numerical columns. There is a long list of normalization methods [14]. We have already described min-max and z-score in 4.2.4. In our implementation, we used min-max scaling because our data are not normally distributed.

To handle categorical data, we transform them into numerical data. There are two ways to perform this transformation, which were already described in 4.2.4. We decided to prefer *one-hot encoding (dummy variables)* over the simple *integer encoding*, due to its better characteristics. After applying the one-hot encoding to categorical columns, the feature space was extended by lots of binary variables carrying information of the original categorical data.

In the end, we merge the scaled numerical data and the one-hot encoded categorical data. We have cleaned and normalized the raw data. Data are saved into CSV[3] files. The scaling objects used at preprocessing are serialized to be able to use them later at prediction.

5.4 Dimensionality reduction

Although we have preprocessed normalized data, we cannot use them for prediction yet. Data retrieved from the systems contained about 400 attributes. After preprocessing, the original data was extended by another 800 variables due to the encoding of categorical variables. This is a problem we have to solve with dimensionality reduction techniques. There are two basic approaches to reduce the number of dimensions of our data − *feature selection* and *feature extraction*. By reducing the high dimensionality, we can create more accurate predictive models.

5.4.1 Feature selection

Feature selection helps us to choose features that will give as good or better model accuracy while requiring fewer data. These methods can be used to identify and remove irrelevant or redundant attributes from data that do not contribute to the accuracy of a predictive model or may in fact even decrease the overall accuracy of the model. It is more desirable to have fewer attributes because it reduces the complexity of models and simple models are easier to understand and explain.

The main goals and benefits of feature selection are [22, 21]:

- improve the performance of predictors,
- provide faster and more cost-effective predictors − less data means that algorithms train faster,
- provide a better understanding of the underlying process of prediction,
- reduces overfitting − less redundant data means less opportunity to make decisions based on noise,

In this thesis we use 4 feature selector methods which implement 3 distinct approaches of feature selection:

[3]**CSV (comma-separated values)** is a delimited text file using a comma (or other separator) to separate values. It stores tabular data (numbers and text) in plain text.

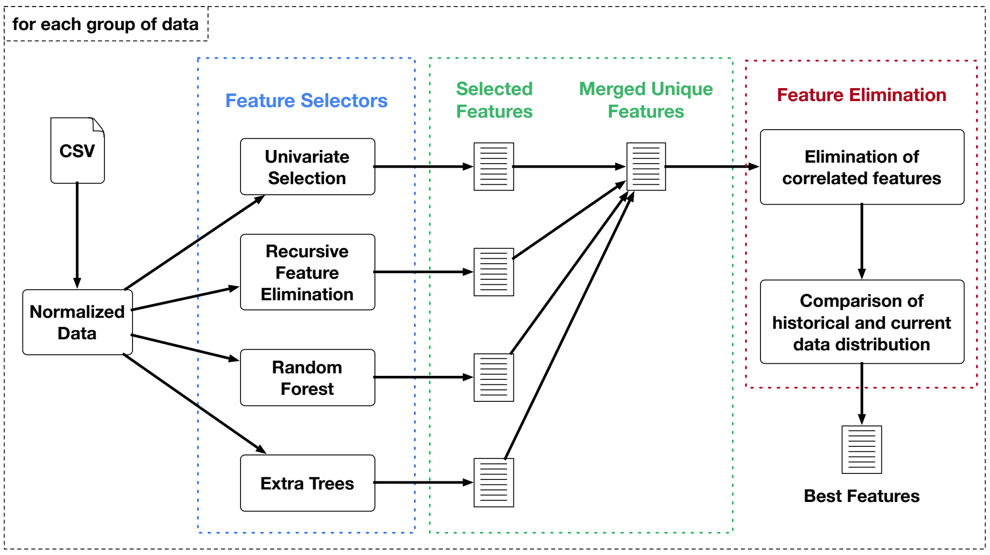


Figure 5.2: This figure captures the process of selecting the best features from given data file. Resulting features are used for training and evaluation of prediction models.

Univariate selection - these methods work by selecting the best features based on univariate statistical tests. These tests are used to select those features which have the strongest relationship with the target variable. The *sci-kit learn* library in Python provides *SelectKBest* class that can be used with a suite of statistical tests to select a given number of features.

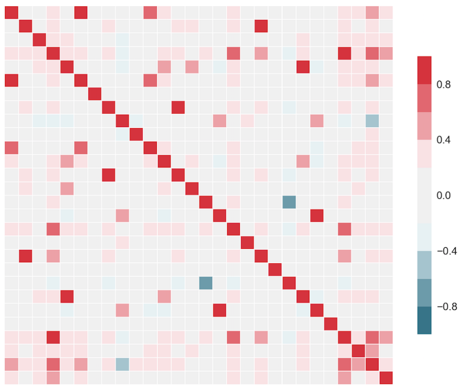
Recursive feature elimination - this approach is recursively removing attributes and builds a model on remaining attributes. It uses the model accuracy to identify which attributes contribute the most to predicting the target.

Feature importance - ensemble methods like *Random forests* or *Extremely randomized trees* can be used to estimate the importance of features. These methods use a metric called *gini importance* or *mean decrease impurity.* It is defined as the total decrease in node impurity (weighted by the probability of reaching that node which is approximated by the proportion of samples reaching that node) averaged over all trees of the ensemble [19].

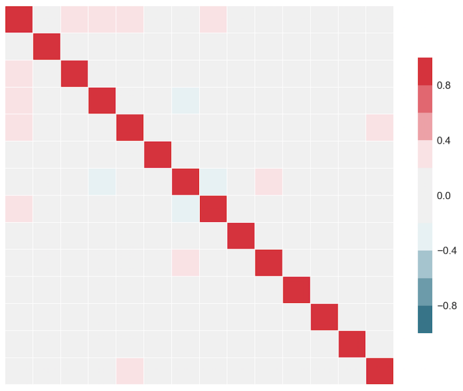
We implemented the following process to retrieve k best features from preprocessed data. This process is shown on figure 5.2 and it is made of these steps:

1. Retrieve normalized data created at the phase of data preprocessing.
2. Fit feature selector models with data and retrieve m features with largest contribution to model accuracy.
3. Merge selected features into one list and remove duplicate features.
4. Perform correlation analysis and remove highly correlated features. An example can be seen on figure 5.3.
5. Visualize and compare data distributions of selected features on ended projects and projects currently in progress. Keep features with equal or similar distribution.

6. Use resulting k features for training and evaluation of models.

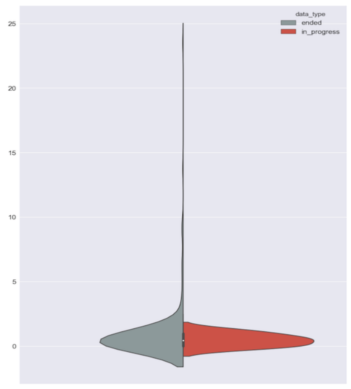


(a) Selected merged unique features

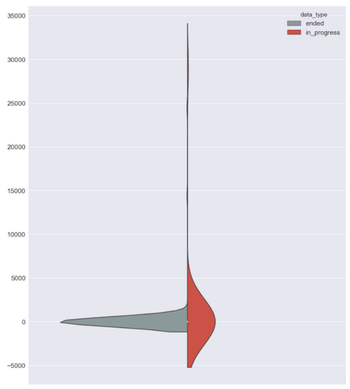


(b) Eliminated highly correlated features

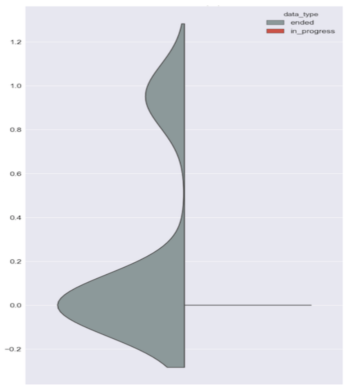
Figure 5.3: We performed correlation analysis on selected features. On the left image we can see that there are some features highly correlated to each other. We recursively remove features which has higher correlation coefficients in absolute value than a given threshold value ϕ (we use $\phi = 0.3$) and correlates to the most features. The elimination process stops once there are no such feature with higher correlation coefficient than ϕ. The resulting set of features are shown on the right image.



(a) Numeric feature 1



(b) Numeric feature 2



(c) Numeric feature 3

Figure 5.4: After removing highly correlated features, we have to compare the distribution of data of ended projects and projects in-progress by each feature. We used *violin plots*[4]to compare the distribution of numeric features and *histograms* for categorical features comparison. The first violin plot captures a feature which can be used safely for prediction due to a similar distribution of values of ended and current projects. We can also use the second feature, but the similarity of distributions are not so much the same. The last feature is irrelevant because it contains information after PG5. We forgot to add it to the list of irrelevant columns during preprocessing.

[4]**Violin plot** is a combination of box plot and a density plot that is rotated and placed on each side, to show the distribution of data.

5.4.2 Feature extraction

In this thesis, we used feature selection approaches to deal with a large number of dimensions. However, there is another approach called feature extraction. A well-known algorithm of this approach is *Principal component analysis* (*PCA*). Generally, this is a data reduction technique. A property of PCA is that we can choose the number of dimensions in the transformed result. PCA tries to find correlated variables and describe them as Principle Components (PCs). When a PCA model is built, we can use 5 PCs instead of original 20 variables − we know that most of the information in the 20 variables are still present in the 5 PCs. These PCs can be viewed as features of the original data, which provide a different representation of the data for the models to use.

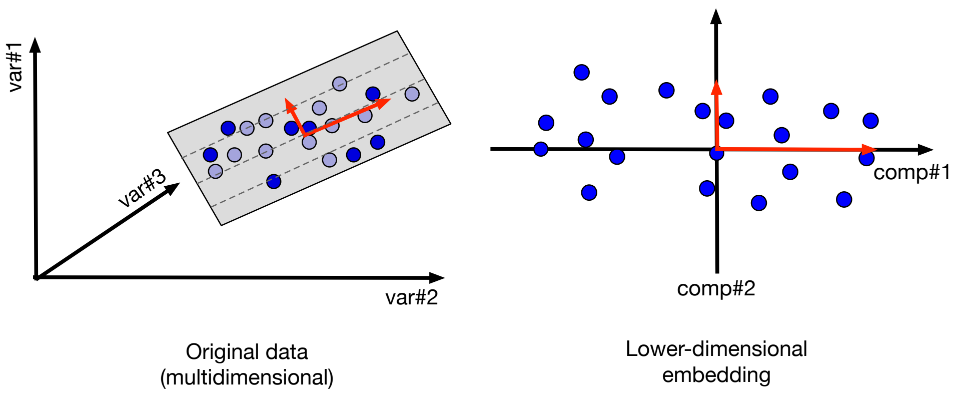


Figure 5.5: This figure captures PCA transformation. Blue points represent samples in data. In this schematic, PCA reduces the dimensionality from 3 to 2. In particular, it finds a pair of orthogonal vectors (red arrows) that define a lower-dimensional space (grey plane) which captures as much variance as possible from the original dataset. The example is taken from [55].

Despite being very useful as a feature extraction algorithm, PCA has several drawbacks we have to be aware. First, PCA provides a linear projection of the data. If patterns in the data are non-linear, PCA could even destroy them, producing worse models than the original data would create [14]. In case we have fewer samples than variables PCA is an inconsistent estimator − we need to regularize the given problem. PCA is not suitable to use if data contain missing values. Then, PCA is provably an NP-hard problem[55].

Another drawback of PCA is the lack of interpretability it yields. Principal components aggregate correlated variables in the data, and for this reason, we lose interpretability of the prediction models since we do not know what features they are using to predict the target variable. For this reason, we could not use PCA to extract features from the data. We preferred feature selection methods due to better interpretability of the features and prediction models.

Chapter 6

Implementation

In this chapter, we will describe the implementation of the prediction system. Preprocessing and feature selection was already described in the previous chapter. This chapter will cover the implementation of:

- classification models and their training,
- hyper-parameter tuning of models,
- cross-validation of models,
- performance metrics and evaluation methods used to select candidate models,
- prediction.

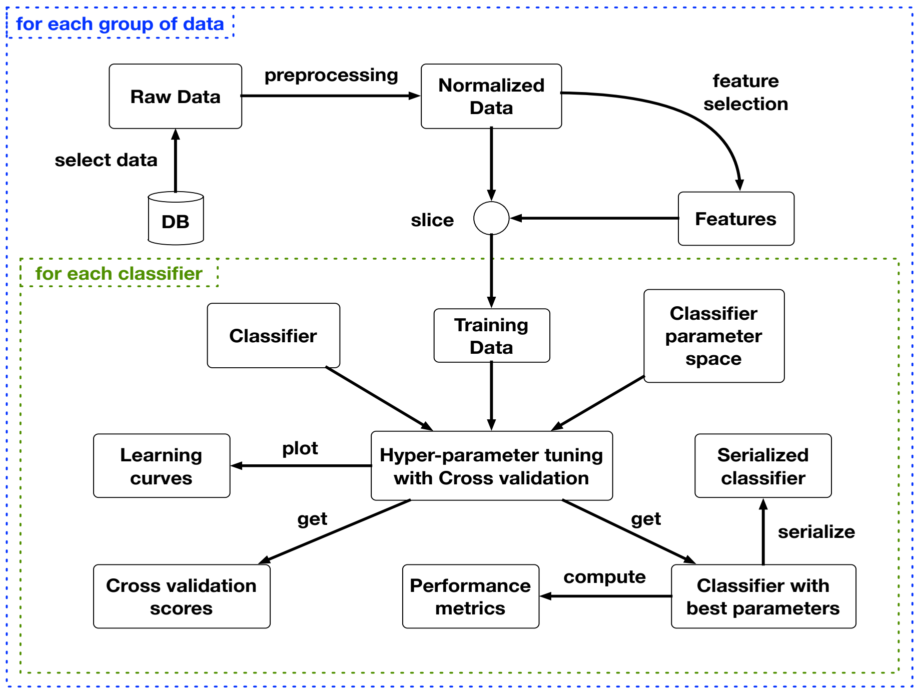


Figure 6.1: This figure captures the workflow of the model training and evaluation process.

6.1 Technology stack

The main programming language of this thesis was Python. We chose Python because it has lots of robust and efficient libraries for data processing e.g. *Numpy*, *Pandas* or *Scipy*. Python also has popular libraries for machine learning such as *Sci-kit learn* or *Keras*, which natively supports previously mentioned data processing libraries. There are also public visualization libraries written in Python. We used *Matplotlib* and *Seaborn* libraries. The data science community of Python is quite large, which we appreciated during the development.

The organization uses MySQL database for persistent storage. Python has a wide range of libraries for database support. We used *SqlAlchemy*'s connection object called *engine*, which can be combined with the *Pandas* library – we retrieved data from the database in a pandas object (data-frame).

Trained models were deployed on an intern server as a small web application. This web application is showing current projects and their phase gate compliance prediction in the form of a dashboard. We used Python's *Flask* web application library.

6.2 Classifiers

We have described various algorithms in section 4.2. In this section, we will focus on their implementation. We used *Keras* library to implement neural networks. This library is a wrapper above the popular *Tensorflow* deep learning library. *Keras* simplifies the implementation of neural networks and it is very helpful in quick prototyping. Other models were implemented with the *sci-kit learn* library.

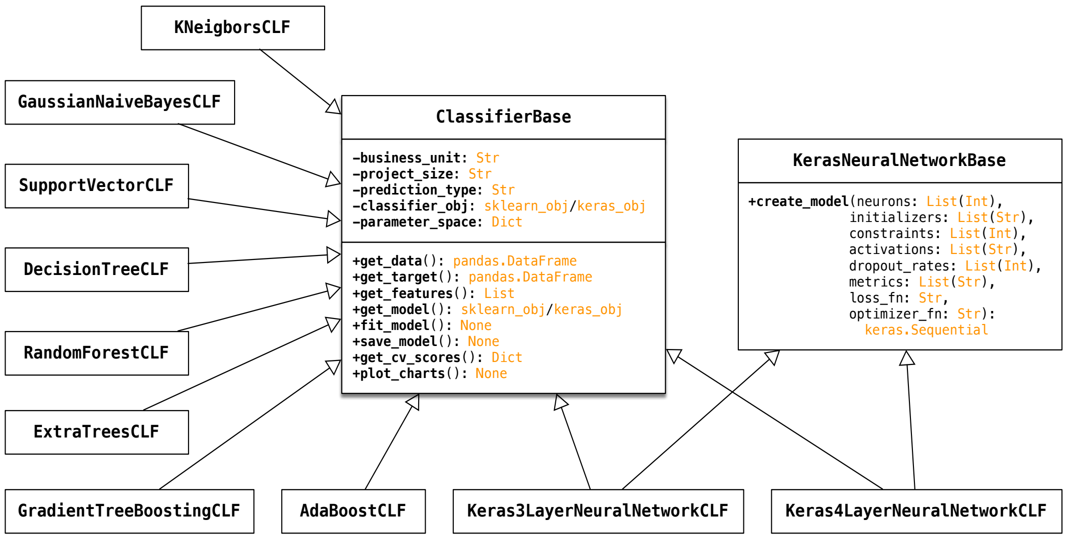

KNeigborsCLF
GaussianNaiveBayesCLF
SupportVectorCLF
DecisionTreeCLF
RandomForestCLF
ExtraTreesCLF
GradientTreeBoostingCLF
AdaBoostCLF
Keras3LayerNeuralNetworkCLF
Keras4LayerNeuralNetworkCLF
ClassifierBase
-business_unit: Str
-project_size: Str
-prediction_type: Str
-classifier_obj: sklearn_obj/keras_obj
-parameter_space: Dict
+get_data(): pandas.DataFrame
+get_target(): pandas.DataFrame
+get_features(): List
+get_model(): sklearn_obj/keras_obj
+fit_model(): None
+save_model(): None
+get_cv_scores(): Dict
+plot_charts(): None
KerasNeuralNetworkBase
+create_model(neurons: List(Int), initializers: List(Str), constraints: List(Int), activations: List(Str), dropout_rates: List(Int), metrics: List(Str), loss_fn: Str, optimizer_fn: Str): keras.Sequential

Figure 6.2: Figure shows the implementation of classification models in form of a class diagram. All operations shared by all classification models are encapsulated in the `ClassifierBase` class. Each classifier is instantiated with its object and parameter space. In the case of neural networks they share a factory method.

Figure 6.2 shows the implemented hierarchy of classes. We can see that there is a `ClassifierBase` class encapsulating shared operations within the classifiers. Specialized classifier classes are initialized by these parameters:

- `business_unit`, `project_size` and `prediction_type` – we need them to properly load data and features.
- `classifier_obj` and `parameter_space` – each classifier class instantiates appropriate classifier method and defines a dictionary of hyper-parameter values for their tuning.

The following code snippet from the implementation shows an example of classifier class specifically `KNeighborsCLF` class.

```python
from sklearn.neighbors import KNeighborsClassifier

class KNeighborsCLF(ClassifierBase):
   def __init__(self, data_group):
       clf = KNeighborsClassifier()
       clf_params = {
           "n_neighbors": [5, 7, 11],           # number of neighbors
           "weights": ["uniform", "distance"],  # weight function
           "p": [1, 2]                          # minkowski (1), euclidean (2) distances
       }
       super().__init__(data_group, clf, clf_params)
```

First, we instantiate a `KNeighborsClassifier` class with given parameters. Then, we define the parameter space for the classifier, which will be used for the tuning. To construct the parameter space for each classifier method, we had to properly understand what does a given parameter do and what impact may it have on the performance of the model.

Other classifier classes are very similar except for neural network. They inherit additionally from `KerasNeuralNetworkBase` class, which contain `create_model` method for creating the neural network according to parameters of the method.

6.3 Cross validation and hyper-parameter tuning

As described in the previous section, we have created a classifier object and its parameter space. These values are passed into the constructor of base class, which calls the following method:

```python
def fit_model(self) -> None:
   grid_search = GridSearchCV(
       self.classifier_obj,   # classifier object
       self.parameter_space,  # classifier's parameters space
       cv=30,       # number of cross-validation folds
       refit=True,  # refit the classifier with best hyper-parameters on whole dataset
       n_jobs=-2,   # number of parallel jobs (-1 = max)
       return_train_score=True)
   # fit the grid search with data
   grid_search.fit(self.samples, self.target)
   # set classifier with best hyper-parameters
   self.model = grid_search.best_estimator_
   # retrieve cross validation scores and best hyper-parameters
   self.cv_results = self.get_scores(self.model.cv_results_, self.model.best_params_)
   self.best_params = self.model.best_params_
```

This method uses *sci-kit learn*'s `GridSearchCV` class, which combines cross-validation with hyper-parameter tuning. It uses `StratifiedKFold`, which is a variation of the k-fold method that returns stratified folds – folds made by preserving the percentage of samples for each class [47]. `GridSearchCV` takes all n-tuples gradually from the cartesian product of hyper-parameters and performs cross-validation for each n-tuple of hyper-parameters. It collects all the cross-validation results during its run. With the parameter `refit=True` set, it refits the classifier with best-found hyper-parameters on the whole dataset and returns it. The best hyper-parameters are those that maximize the score of the testing data. We have retrieved the average score and the standard deviation of scores of the classifier. An example of the cross-validation scores and best parameters of the KNN model:

```
best_params = {"n_neighbors": 5, "p": 1, "weights": "uniform"}
cv_scores = {"mean_train_score": 0.8836, "std_train_score": 0.0016,
             "mean_test_score": 0.8801,  "std_test_score": 0.0462}
```

After the classifier with best parameters is already trained, we need to evaluate the model to see how it performs with comparison to other models. In the next section, we will cover the performance metrics and visualization tools used in our implementation.

6.4 Performance metrics and visualization

We have implemented several performance measures and visuals to compare prediction models. We encapsulated the implementation in the `PerformanceMetrics` class. Class diagram is shown on figure 6.4.

We used functions from *Sci-kit learn*'s `metrics` module which contain implementations of performance measures and more. `PerformanceMetrics` class implements:

- most significant classification performance measures – *precision*, *recall* and *F-measure* using `metrics.classification_report` function. Given function calculates model's performance and returns the following output. An example of classification report of `KNeighborsClassifier`:

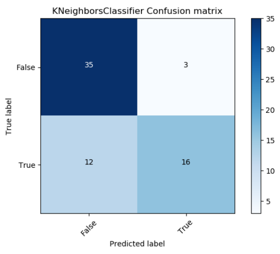
```
  classes    precision    recall    F1_score   support
    False         0.74      0.92        0.82        38
     True         0.84      0.57        0.68        28
avg/total         0.79      0.77        0.76        66
```

- plotting of *confusion matrix*. It is calculated using `metrics.confusion_matrix` function. Plotted confusion matrix is visualized on figure 6.3.

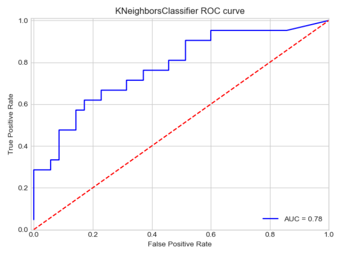
- plotting of *ROC* curve. This important performance descriptor is calculated by `metrics.roc_curve` and `metrics.auc` functions. Plotted ROC curve is shown on figure 6.3.

- plotting of *learning curves*. In order to plot the learning curve of a classifier, we used previously collected cross validation scores of given classifier and passed them into *Sci-kit learn's* `model_selection.learning_curve` function. This function return value can be passed right into *Matplotlib*'s plotting functions. An example can be viewed on figure 6.3.

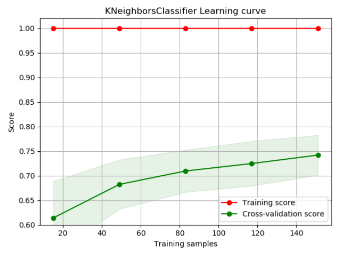
- plotting of *neural network's topology*. *Keras* library's function `utils.vis_utils-.plot_model` makes easy for us to accomplish that.



(a) Confusion matrix

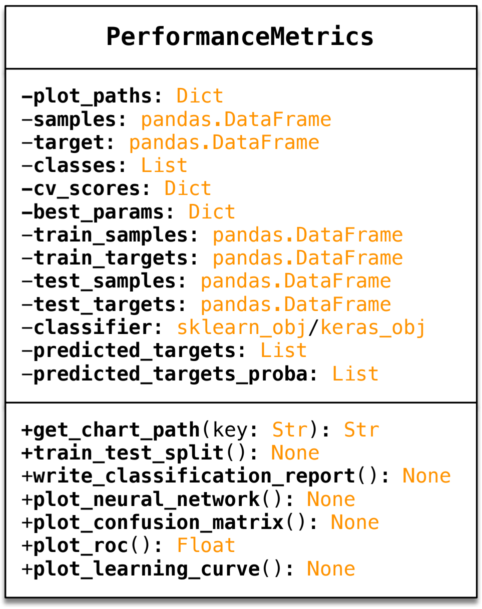


(b) ROC curve



(c) Learning curve

Figure 6.3: On left picture you can see the confusion matrix of validation performance of `KNeighborsClassifier` on a given dataset. Middle picture shows its ROC curve and AUC score. The right one captures the learning curve of given classifier.

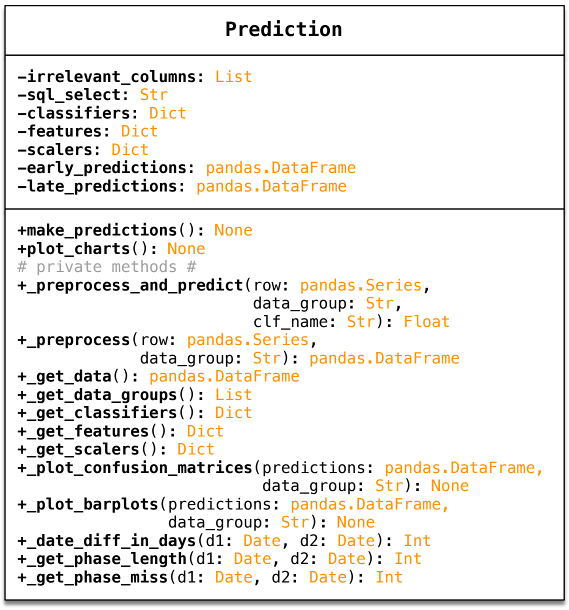


```
PerformanceMetrics

-plot_paths: Dict
-samples: pandas.DataFrame
-target: pandas.DataFrame
-classes: List
-cv_scores: Dict
-best_params: Dict
-train_samples: pandas.DataFrame
-train_targets: pandas.DataFrame
-test_samples: pandas.DataFrame
-test_targets: pandas.DataFrame
-classifier: sklearn_obj/keras_obj
-predicted_targets: List
-predicted_targets_proba: List

+get_chart_path(key: Str): Str
+train_test_split(): None
+write_classification_report(): None
+plot_neural_network(): None
+plot_confusion_matrix(): None
+plot_roc(): Float
+plot_learning_curve(): None
```

(a) Performance metrics class



```
Prediction

-irrelevant_columns: List
-sql_select: Str
-classifiers: Dict
-features: Dict
-scalers: Dict
-early_predictions: pandas.DataFrame
-late_predictions: pandas.DataFrame

+make_predictions(): None
+plot_charts(): None
# private methods #
+_preprocess_and_predict(row: pandas.Series,
                         data_group: Str,
                         clf_name: Str): Float
+_preprocess(row: pandas.Series,
             data_group: Str): pandas.DataFrame
+_get_data(): pandas.DataFrame
+_get_data_groups(): List
+_get_classifiers(): Dict
+_get_features(): Dict
+_get_scalers(): Dict
+_plot_confusion_matrices(predictions: pandas.DataFrame,
                          data_group: Str): None
+_plot_barplots(predictions: pandas.DataFrame,
                data_group: Str): None
+_date_diff_in_days(d1: Date, d2: Date): Int
+_get_phase_length(d1: Date, d2: Date): Int
+_get_phase_miss(d1: Date, d2: Date): Int
```

(b) Prediction class

Figure 6.4: This figure shows the class diagram of class `PerformanceMetrics`.

6.5 Prediction

After we have trained our models, tuned their parameters and visualized their performance, we describe the implementation of the most exciting part of this thesis – the prediction. The workflow of the prediction process is shown in figure 6.5.

We encapsulated the whole implementation of prediction into a class called `Prediction`. Class diagram is shown on figure 6.4. We selected all projects in the organization which are in progress at the time of writing this thesis. We will describe these projects in the next chapter.

The process of predicting the PG5 compliance of currently active projects are made of these steps:

1. First, for each data group we retrieve active projects in the organization.

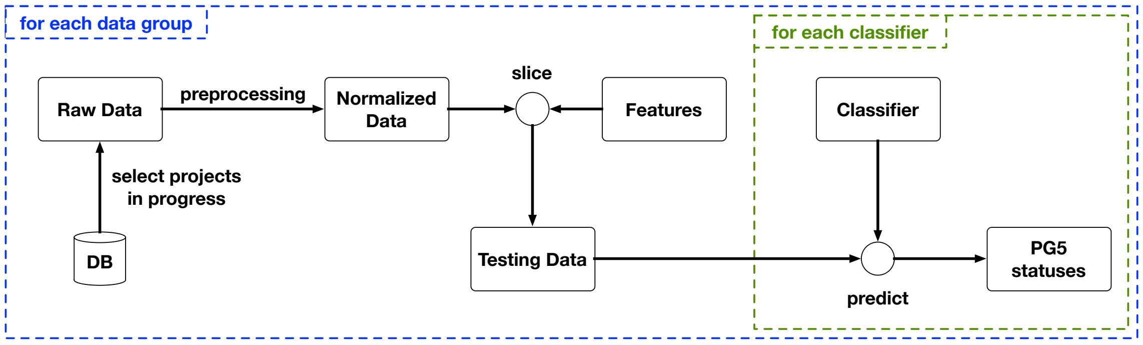


Figure 6.5: Figure showing the workflow of the prediction process. First, we retrieve the data of currently active projects from our database. We apply the same preprocessing steps we used previously and select features. Then, we pass normalized data into classification models and collect their predictions.

2. Then, we preprocess these samples using the same preprocessing steps as before.
3. Select features from data.
4. Predict PG5 compliance with each classification models and collect their predictions.
5. Save predictions into CSV file.

The following code is our implementation of the prediction process.

```python
def make_predictions(self) -> None:
   early_predictions, late_predictions = list(), list()
   for data_group in self._get_data_groups():         # for each data group
       raw_data = self._get_data(data_group)          # load data from DB
       for clf_name in self.classifiers[data_group]:  # for each classification model
           # preprocess each row and predict PG5 status
           predict_data[clf_name] = raw_data.apply(
              lambda row: self.preprocess_and_predict(row, clf_name, data_group),axis=1)
       if "early" in data_group:  # collect predictions
           early_predictions.append(predict_data)
       else:
           late_predictions.append(predict_data)
   # concatenate and set all predictions
   self.early_predictions = pd.concat(early_predictions)
   self.late_predictions = pd.concat(late_predictions)

def preprocess_and_predict(self, row, clf_name, data_group):
   normalized_sample = self._preprocess(row, data_group)                 # preprocessing
   testing_sample = normalized_sample.loc[:, self.features[data_group]]  # get features
   return (self.classifiers[data_group][clf_name]                        # prediction
           .predict_proba(testing_sample).item(0) * 100)
```

It is very important to use the same preprocessing steps during the prediction as we used during the training. This is the reason why did we serialize the scaler objects before training – to use the same scaling technique and ranges. All predictions are shown in appendix B.

Chapter 7

Model evaluation

In this chapter, we will introduce existing prediction models which are used within the company. We will describe their training performance and their prediction power. Existing models are trained on projects ended between 2014 and 2015. These models were trained using previously selected feature variables. We will test their prediction power on testing set.

Next, we will evaluate new models trained on projects ended between 2014 and 2016. We will test new models on the same testing set to be able to compare their performance with previous models. New models were trained using new feature variables retrieved by our feature selectors described in 5.4. According to training and testing results we will choose best candidate models among new predictors.

At the end of this chapter, we will compare the performances of existing and new models. Depending on the results, we will choose models with best prediction power. Abbreviation used in this chapter are enlisted in table 7.1.

Abbreviation	**Classification Model**
3LNN	3-layer Neural Network
4LNN	4-layer Neural Network
ADA	AdaBoost
DTC	Decision Tree
ETC	Extremely Randomized Tree
GNB	Gaussian Naive Bayes
KNN	K-Nearest Neighbors
RFC	Random Forest
SVM	Support Vector Machine
XGB	Extreme Gradient Boosting

Table 7.1: List of abbreviation used in this section

7.1 Training and testing sets

Training sets consisted of only NPI projects, which were not stopped or canceled during the development. As already mentioned in section 5.2, training samples were divided by business unit and project size. Existing models were trained on projects ended between 2014 and 2015. Then, we extended these training sets by projects ended in 2016. New models were trained on this extended training set. The number of samples in training sets are following:

- major thermostats – 176,
- minor thermostats – 292,
- major security panels – 221.

7.2 Current models

In this section, we will evaluate the prediction performance of existing models in the organization. We have trained all models on projects ended between 2014 and 2015, but only 3 models were originally implemented during the rapid development – `SVM`, `RFC` and `ETC` (we marked them with ↑ in the following charts for better orientation). At each project group, we will evaluate its models by their performance metric scores measured during cross-validation and by their prediction performance.

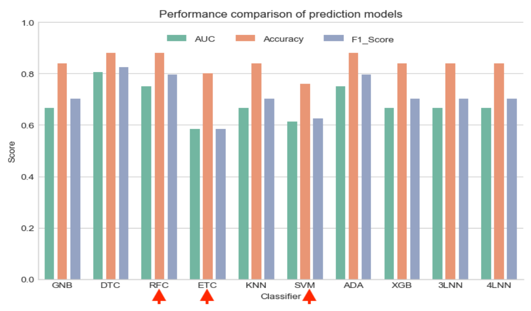
We measured three performance metrics – *AUC* score, which is the area under ROC curve of given model, *F1 Score*, which represents the relationship between recall and precision of given model and the third metric is *Accuracy* reflecting the number of correctly predicted values.

7.2.1 Major thermostat projects

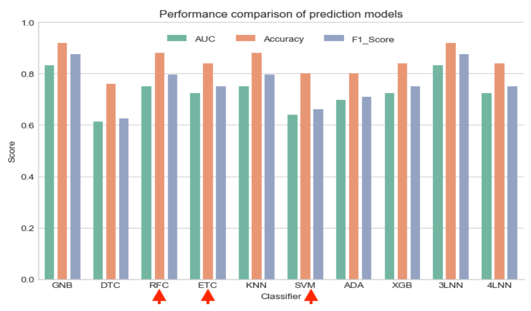
Figure 7.1 shows major thermostat prediction models' performance scores. We can see that late models have higher scores than early ones. Late models were trained on features, which contain information about phase 4. Bar charts show that some models increased their scores, while some of them did not.

Figure 7.1: **Current major thermostat models' performance metrics**

(a) Early models' performance



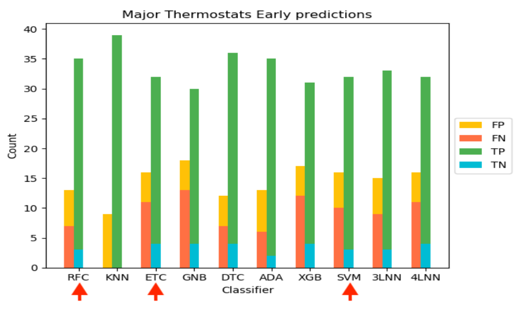
(b) Late models' performance



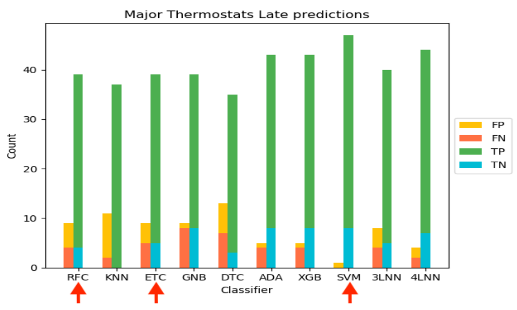
Although performance metrics are an important part of the model evaluation and help us to the get better insight to models' behavior, the prediction accuracy on real projects is the final test, whether we choose given model for candidate model or not. Stacked bar charts on figure 7.2 show prediction results on projects ended in 2017. The left bar contains mistakes the model made during the prediction, and the right bar shows all correctly predicted samples. Colors represent different values of the confusion matrix.

Figure 7.2: **Current major thermostat models' predictions**

(a) Early predictions



(b) Late predictions



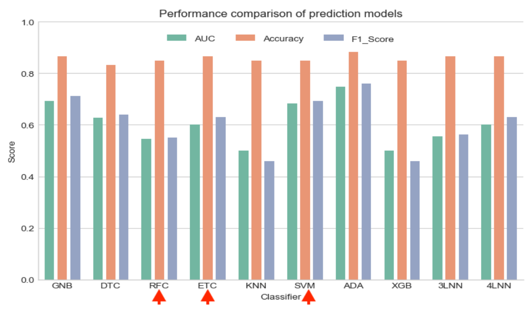
Currently deployed models have quite good predictive power. We prefer models with higher false negative rate than false positive rate. False negative errors serve as warnings that something might not be okay with given projects.

7.2.2 Minor thermostat projects

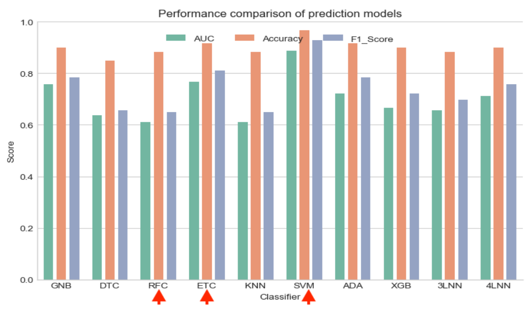
Bar charts on 7.3 captures the performance of minor thermostat predictors measured by cross-validation. We can see that late models have quite increased their performance with the inclusion of phase 4 features. Late `SVM` model has achieved nearly 100% accuracy score. Let's analyze their prediction accuracy on projects ended in 2017.

Figure 7.3: **Current minor thermostat models' performance metrics**

(a) Early models' performance



(b) Late models' performance



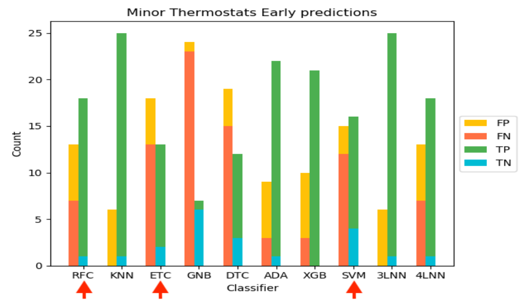
We can see that currently used early models in the organization has made quite a lot of mistakes. However, some of the newly implemented early predictors have predicted with higher accuracy – early models with the least mistakes are `KNN` and `3LNN`. In contrast, late models' accuracy looks roughly the same except `3LNN` – this network was unable to generalize the given problem properly. Predictions are shown on stacked bar charts in figure 7.4.

7.2.3 Major security panel projects

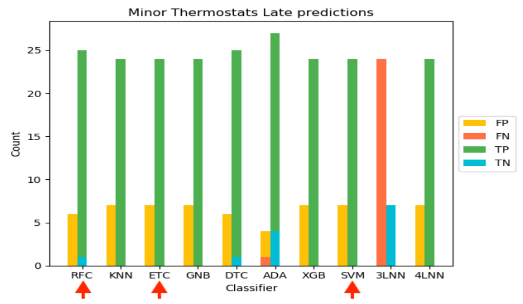
Bar charts in 7.5 captures the performance scores of models calculated during cross-validation. We can see that late models have increased their performance with information provided

Figure 7.4: **Current minor thermostat models' predictions**

(a) Early predictions



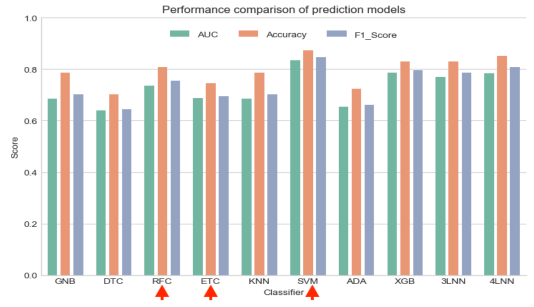
(b) Late predictions



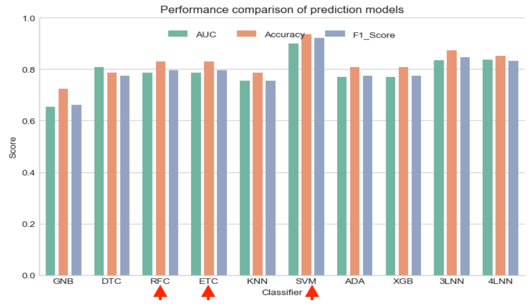
in phase 4 features. As it was in minor thermostat predictors, SVM has the highest cross-validation scores. We will test its predictive power on 2017 projects.

Figure 7.5: **Current major security panel models' performance metrics**

(a) Early models' performance



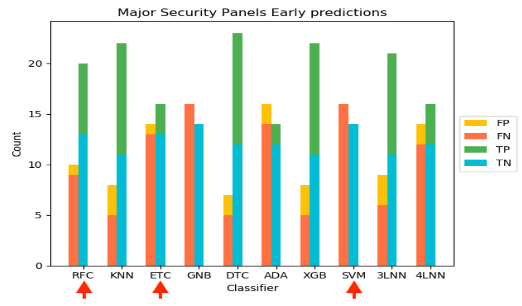
(b) Late models' performance



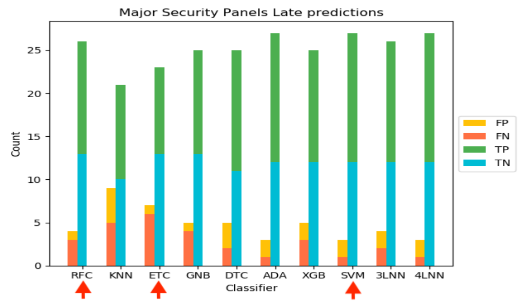
If we take a look at prediction results visualized in stacked bar charts in 7.6, we see that early models are not as accurate as the late models. It seems that in the case of major security panel prediction, the knowledge of phase 4 feature is essential. We can see that early SVM and GNB models are overfitted as they were unable to predict projects of 2017 – it seems that these models always predicted negatively.

Figure 7.6: **Current major security panel models' predictions**

(a) Early predictions



(b) Late predictions



Among early models, `DTC`, `KNN` and `XGB` models performed very well. However, by adding phase 4 features into the training set, the prediction accuracy has improved significantly.

7.3 New models

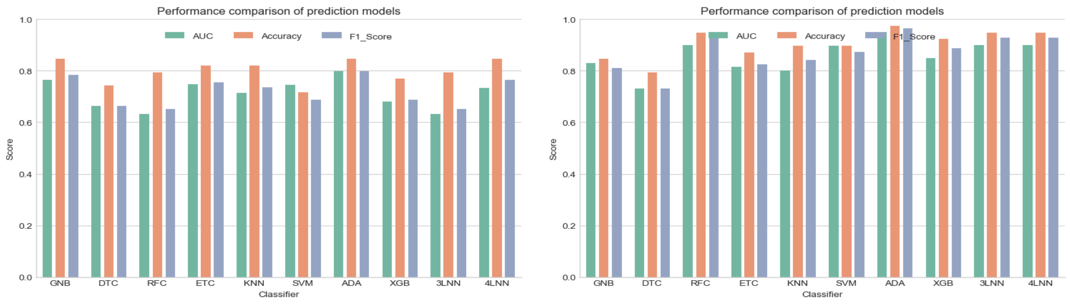
In this section, we will evaluate the prediction performance of newly trained models. We have trained all models on training set consisted of projects ended between 2014 and 2016. We will see whether models increased their prediction accuracy by including projects ended in 2016. At each subsection, we will suggest the best candidate models for the prediction system. Feature variables selected by feature selectors and their description are located in appendix C.

7.3.1 Major thermostat projects

According to charts in 7.7, there is an increase of performance metric scores both on early and late predictions. `ADA` has in both cases the highest score. To properly choose best candidates for this project group, we need to evaluate the predictions on projects ended in 2017. `RFC`, `ADA` and `3LNN` are models with the biggest improvement.

Figure 7.7: **New major thermostat models' performance metrics**

(a) Early models' performance (b) Late models' performance



If we look at figure 7.8, we see that all models performed roughly the same except `DTC` model. Models have higher false positive rate than false negative rate. However, the accuracy has improved in comparison with current models. Late predictors have slightly improved their prediction accuracy and reduced the false positive rate – this is true especially for `RFC`, `ADA`, `XGB`, `SVM`, `3LNN` and `4LNN`.

We have carefully analyzed the performance metric scores and prediction accuracy of new models. For the early models, we suggest `ADA` for the candidate model due to having best performance scores of cross-validation and having the highest accuracy and lowest false positive rate on projects of 2017. We could suggest `ADA` or neural networks for late candidates – they have highest performance metric scores, and they also performed well on 2017 projects.

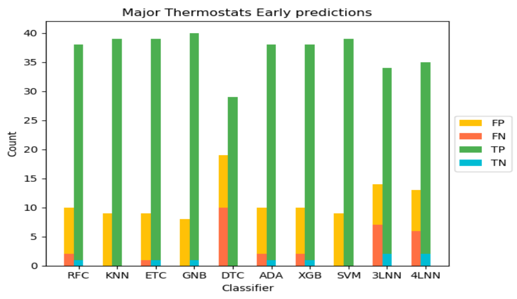
7.3.2 Minor thermostat projects

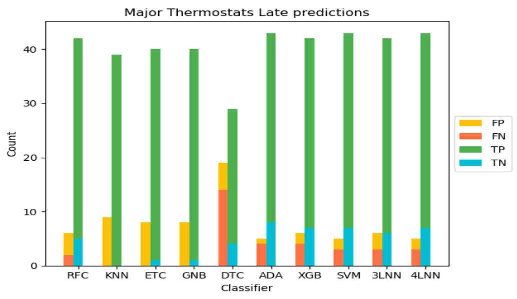
In this subsection, we will analyze newly trained prediction models of minor thermostat project group. According to performance metrics visualized on 7.9, models with best scores

Figure 7.8: **New major thermostat models' predictions**

(a) Early predictions

(b) Late predictions



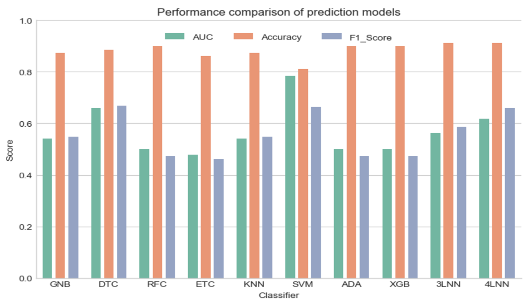


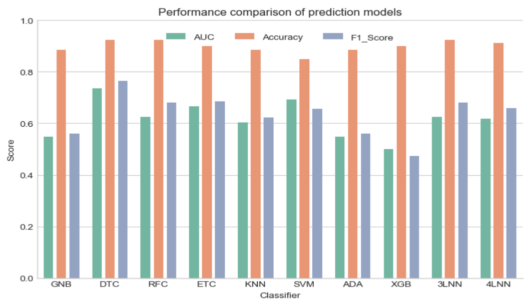
are DTC and SVM. Interestingly, late SVM has decreased in scores, late DTC in contrast, has increased overall performance metric scores. Neural networks have quite high scores, too. We will see how they predict 2017 projects and according to results, we will choose candidates for the prediction system.

Figure 7.9: **New minor thermostat models' performance metrics**

(a) Early models' performance

(b) Late models' performance





Prediction of *PG5 Miss* by both early and late models are summarized in stacked bar charts on figure 7.10. The early SVM model did not perform well on new data with predictions. Also early models ETC and GNB have low accuracy – it seems that they predicted all labels negatively. Unlike early models, the results of late models are very good. DTC, 3LNN, and 4LNN could be chosen for candidates because they have the most accurate prediction and their false positive rate is meager.

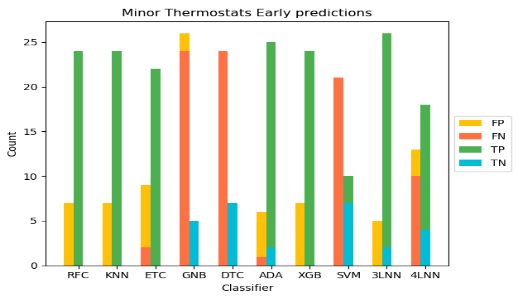
Among early models, we suggest one model of the following: ADA, 3LNN, XGB. Despite having average performance metric scores compared to other models, they have performed best on the testing data. All late models look quite the same according to prediction accuracy. We suggest DTC or 3LNN for candidate, because they performed slightly better than other models in this project group.

7.3.3 Major security panel projects

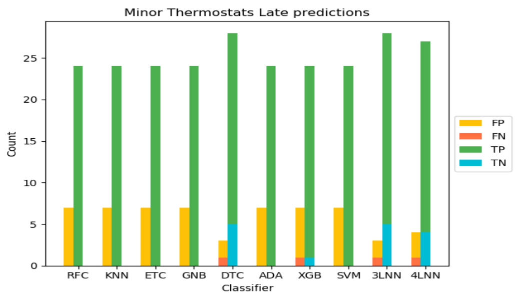
The testing set of this project group contain more missed projects than the other groups. We will see, how well did our models learn to generalize this project group. First, we will analyze the performance metric scores of prediction models, which are shown on figure 7.11.

Figure 7.10: **New minor thermostat models' predictions**

(a) Early predictions



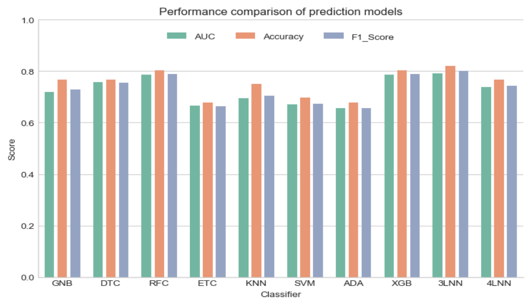
(b) Late predictions



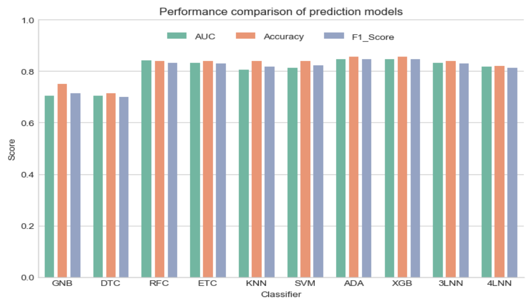
Bar charts show, that models have quite the same performance scores. This is especially visible on the right picture.

Figure 7.11: **New major security panel models' performance metrics**

(a) Early models' performance



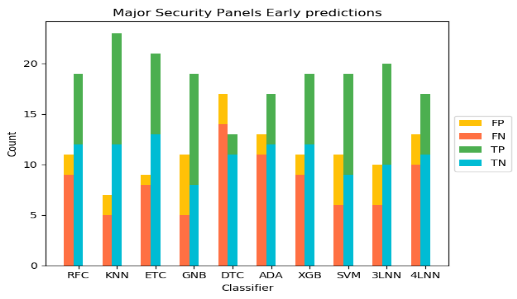
(b) Late models' performance



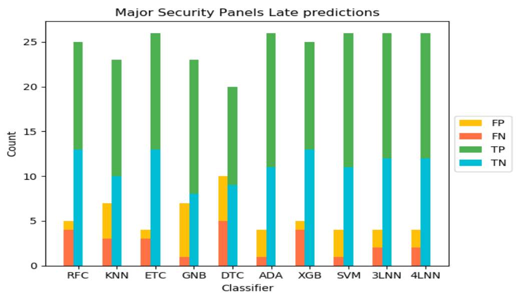
In order to evaluate the prediction results we will evaluate how well did our models perform on 2017 projects. Figure 7.12 contains two stacked bar charts, where we summarized the prediction results of models. We can see, that `KNN` had the best accuracy on the testing set among early models. Late models, in the other hand, performed a lot better that the early models.

Figure 7.12: **New major security panel models' predictions**

(a) Early predictions



(b) Late predictions



As we already mentioned above, we suggest KNN for the candidate model for the early prediction. There are several late models, which performed equally well. We suggest ETC, RFC or XGB for candidate models of major security panel prediction.

7.4 Comparison of current and new predictors

In previous sections, we have evaluated currently used and newly trained prediction models. In this section, we will compare them and analyze whether they improved or not.

Although the organization implemented only three models (SVM, RFC, ETC), we have trained the rest of the predictors on the same training set and feature set. In this way, we could objectively compare currently used models with the new ones. We found out that in some cases some other models had more accurate predictions than the currently deployed models. For example, the company uses ETC model to predict *Phase Gate 5 Miss* of late thermostats. ADA has made fewer mistakes and at the same time had a lower false positive rate.

In table 7.2 we enlisted and compared selected candidate models with current models of the prediction system. We have achieved higher prediction accuracy by:

- extending the original training set with projects ended in 2016,
- using new feature variables
- and considering wider range of prediction models.

Candidate models were able to create generalization rules successfully. Models have been able to predict projects ended in 2017 with high accuracy using these generalization rules. Cross-validation results of candidate models are shown on figures in D.

			Current model			**Candidate model**		
			Name	**Accuracy (%)**	**FP rate (%)**	**Name**	**Accuracy (%)**	**FP rate (%)**
Thermostats	**Major**	**Early**	ETC	66.67	55.56	ADA	↑ 79.17	↑ 88.89
		Late	ETC	83.33	44.44	ADA	↑ 89.58	↓ 11.11
	Minor	**Early**	ETC	41.94	71.43	3LNN	↑ 83.87	71.43
		Late	ETC	77.42	100.00	3LNN	↑ 90.32	↓ 28.57
Security Panels	**Major**	**Early**	ETC	53.33	7.14	KNN	↑ 76.67	↑ 14.28
		Late	ETC	76.67	7.14	ETC	↑ 86.67	7.14

Table 7.2: This table contains the comparison of current models and selected candidate models. We compared the prediction accuracy of candidate models with the currently deployed model in the company – ETC predictor. In each category, new candidate models achieved higher accuracy than the current model.

Chapter 8

Further improvements

We have successfully trained and evaluated new prediction models and chosen new candidates for the prediction. In this chapter, we will think about the possibilities, how could we obtain more accurate predictions. What could we do to improve the predictive power of models? We have tried various models with different approaches to solving our classification problem. We identified and chose the best candidates. There are two ways we could improve our prediction accuracy.

First, we should analyze our feature variables in more detail to get better insight − why did we get such feature variables; why are they more relevant than others. For example, let's analyze *Phase 4 Length* feature. It reflects the length of phase 4 of the development process, however, it is only a result metric. Typically, the longer the length of phase 4, the higher the probability of *Phase Gate 5 Miss*. This can be seen in figure 8.1. However, what are the causes of phase gate length extension?

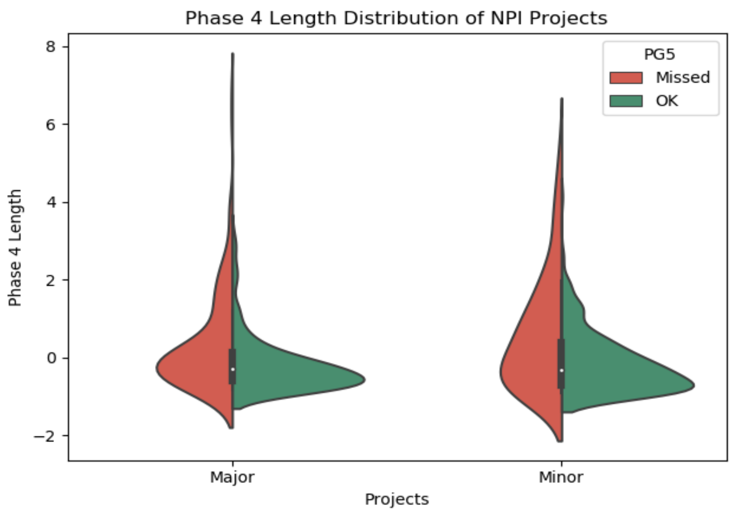


Figure 8.1: Figure captures the *Phase Gate 4 Length* distribution for major and minor NPI projects in the organization. We can see that projects which met Phase Gate 5 schedule have lower *Phase Gate 4 Length*. Values were scaled to not reveal confidential information.

It can be the case, that major projects are more challenging to manage. If this is true, we could suggest changing the development process of major projects − we could divide these projects into two smaller projects. For example, let's consider we want to break down the development of a thermostat with lots of features into two parts. The first project will have the responsibility to develop a basic version of the thermostat, while the second will update the basic version with the rest of the features. However, the question is whether the basic version would have success on the market − if yes, it would worth to adopt this development process change.

Let's consider *Phase Gate 4 Miss* feature. It says whether a project meets planned phase gate 4 schedules, however, we have no information about the causes of delays. We have to undertake a more in-depth analysis, whether there are problems with suppliers, for instance, − we should carefully analyze the risks that suppliers do not deliver components on time.

The second improvement is related to requirements. At the beginning of the Phase Gate Compliance prediction, we have chosen three systems, from which we gather data and use them for the prediction. We have decided to choose these systems by completeness and quality of data covering as many projects in the organization as possible. However, some information is missing from these data. For example, we did not include any information about requirements of projects − project managers use several systems to keep track of requirements, what causes inconsistency in data processing.

We suggest to focus on requirements analysis and to create several performance metrics, which would help to keep track of their quality. In this way, we could also set metrics focusing on the reduction of requirements change requests during the project development process.

Chapter 9

Conclusion

This work was created in cooperation with an IT company operating in the area of smart homes and IoT devices. The organization is predicting whether projects meet planned deadlines of the last phase of development process using machine learning in earlier phases of development. The already implemented prediction system works, but it was created under rapid development. The goal of this thesis was to optimize this prediction system since no broader analysis was made and the company does not know enough about the quality of the predictions.

At the beginning of this work, we focused on performance metrics in an organization. Then, we outlined how the company sets its business goals and how it uses performance metrics to track the success of meeting business goals. We introduced a metric called Phase Gate Compliance that determines whether a given project meets planned schedules. Next, we dived into machine learning and analyzed the existing classification algorithms. Then, we defined how we will measure the performance of classification models and how we will select candidate models.

In the second part of the thesis, we characterized the data sources we used during the training of predictive models. We have described in detail the way of data pre-processing and the selection of the best feature variables. We have also described the implementation of predictive models, implementation of training and performance evaluation. We also specified how did we implement the prediction of Phase Gate Compliance. In the last part of the thesis, we have described and compared the existing and newly trained prediction models and suggested further improvements how could we increase the prediction power of models.

The original training set included NPI projects ending between 2014 and 2015. By extending the training set with projects ended in 2016, we have achieved a slight increase in prediction accuracy. It means that NPI projects ended between 2014 and 2016 have some characteristics in common. The prediction models were able to create generalization rules successfully. An essential part of the optimization is an automatic selection of the best features.

At the end of the work, we outlined how the organization could increase the prediction power of the models. One of the options is to focus on a more profound analysis of selected features, according to which models predict project delays. Another option is to design some performance metrics to help the organization monitor the quality of project requirements, which would reveal possible problems at earlier stages of development and thus reduce the risk of delaying the project. Suggested metrics could help reduce the number of change requests of requirements.

Bibliography

[1] *Autoencoders*. Stanford Deep Learning. [Online; visited 2018-01-16].
Retrieved from:
http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

[2] *Cluster analysis*. Wikipedia. [Online; visited 2018-01-13].
Retrieved from: https://en.wikipedia.org/wiki/Cluster_analysis

[3] *Cycle Time Reduction - Benefits*. Reference for Business. [Online; visited 2018-01-13].
Retrieved from:
http://www.referenceforbusiness.com/encyclopedia/Cos-Des/Cycle-Time.html

[4] Ensemble learning. Wikipedia. [Online; visited 2018-01-15].
Retrieved from: https://en.wikipedia.org/wiki/Ensemble_learning

[5] *Machine learning diagram overview*. Isazi Consulting. [Online; visited 2018-01-08].
Retrieved from: http://www.isaziconsulting.co.za/machinelearning.html

[6] *Measuring Your Organization's Performance*. Business Improvement Architects.
[Online; visited 2017-12-27].
Retrieved from: https://bia.ca/measuring-your-organizations-performance/

[7] *Multicollinearity*. Wikipedia. [Online; visited 2018-01-18].
Retrieved from: https://en.wikipedia.org/wiki/Multicollinearity

[8] *Neural Networks Applications*. Alyuda Products & Solutions - Intelligent Decision Making. [Online; visited 2018-01-16].
Retrieved from:
http://alyuda.com/products/forecaster/neural-network-applications.htm

[9] *Neural networks with genetic algorithms*. Abhranil blog. [Online; visited 2018-01-17].
Retrieved from: https://blog.abhranil.net/2015/03/03/training-neural-networks-with-genetic-algorithms/

[10] *New Product Introduction*. Q-1 Quality-One International. [Online; visited 2018-01-05].
Retrieved from: https://quality-one.com/npi/

[11] *Support Vector Machine*. StateMaster. [Online; visited 2018-01-18].
Retrieved from:
http://www.statemaster.com/encyclopedia/Support-vector-machine

[12] *The Difference between KPIs and KRIs and Why It Matters*. KPI Karta Inc.. [Online; visited 2018-01-02].
Retrieved from: http://www.kpikarta.com/blog-1/2015/4/30/the-difference-between-kpis-and-kris-and-why-it-matters

[13] *The ID3 Algorithm*. University of Florida. [Online; visited 2018-01-14].
Retrieved from:
https://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm

[14] Abbott, D.: *Applied predictive analytics*. Wiley. 2014. ISBN 978-1118727966.

[15] Alpaydin, E.: *Introduction to Machine Learning*. MIT Press. 2010. ISBN 978-0-262-01243-0.

[16] Bambrick, N.: *Support Vector Machines: A Simple Explanation*. AYLIEN. [Online; visited 2018-01-18].
Retrieved from: https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html

[17] Breiman, L.: *Prediction Games and Arcing Algorithms. Neural Computation*. vol. 11, no. 7. 1999: pp. 1493–1517. ISSN 0899-7667.

[18] Breiman, L.: *Random Forests. Machine Learning*. vol. 45, no. 1. 2001: pp. 5–32. ISSN 0885-6125. [Online; cited 2018-01-15].

[19] Breiman, L.; Friedman, J. H.; Olshen, R. A.; et al.: *Classification and Regression Trees*. Taylor & Francis. 1984. ISBN 0412048418.

[20] Brownlee, J.: *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. Machine Learning Mastery. [Online; visited 2018-04-02].
Retrieved from: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

[21] Brownlee, J.: *Feature Selection For Machine Learning in Python*. Machine Learning Mastery. [online; visited 2018-04-25].
Retrieved from: https://machinelearningmastery.com/feature-selection-machine-learning-python/

[22] Brownlee, J.: *Feature Selection to Improve Accuracy and Decrease Training Time*. Machine Learning Mastery. [online; visited 2018-04-25].
Retrieved from: https://machinelearningmastery.com/feature-selection-to-improve-accuracy-and-decrease-training-time/

[23] Brownlee, J.: *Why One-Hot Encode Data in Machine Learning?* Machine Learning Mastery. [Online; visited 2018-01-17].
Retrieved from: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[24] Burgetová, I.; et al.: Classification and prediction. Intern document, VUT FIT.
Retrieved from: https://www.fit.vutbr.cz/study/courses/ZZN/private/prednasky/06_klasifikace_predikce.pdf

[25] Cawley, G. C.; Talbot, N. L. C.: *Over-fitting in model selection and subsequent selection bias in performance evaluation. Journal of Machine Learning Research.* vol. 11. 7 2010: pp. 2079–2107.

[26] Chen, T.; Geustrin, C.: *XGBoost: A scalable Tree Boosting System.* 2016. [online; visited 2018-04-04].
Retrieved from: https://arxiv.org/abs/1603.02754

[27] Claesen, M.; Moor, B. D.: Hyperparameter Search in Machine Learning. *CoRR.* vol. abs/1502.02127. 2015.
Retrieved from: http://arxiv.org/abs/1502.02127

[28] Dalinina, R.: *Introduction to correlation.* Data Science. [Online; visited 2018-01-18].
Retrieved from: https://www.datascience.com/blog/introduction-to-correlation-learn-data-science-tutorials

[29] Demeester, L.; Tang, C. S.: *Reducing Cycle Time at an IBM Wafer Fabrication Facility. Interfaces.* vol. 26, no. 2. 3 1996: pp. 34–49. ISSN 0092-2102. [Online; cited 2018-01-13].

[30] Fix, E.; Hodges, J. L.: *Discriminatory Analysis - Nonparametric Discrimination: Consistency Properties.* California University Berkeley. 1951. [Online; visited 201].
Retrieved from: http://handle.dtic.mil/100.2/ADA800276

[31] Friedman, J. H.: *Stochastic Gradient Boosting. Computational Statistics and Data Analysis.* vol. 38, no. 4. 2002: pp. 367–378. ISSN 0167-9473.

[32] Geurts, P.; Ernst, D.; Wehenkel, L.: *Extremely Randomized Trees. Machine Learning.* vol. 63, no. 1. 2006: pp. 3–42. ISSN 0885-6125. [Online; cited 2018-01-15].

[33] Gorman, B.: *A Kaggle Master Explains Gradient Boosting.* [online; visited 2018-04-04].
Retrieved from: http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

[34] Han, J.: *Data mining concepts and Techniques.* Morgan Kaufmann. 2006. ISBN 1-55860-901-6.

[35] Hastie, T.; Tibshirani, R.; Friedman, J.: *Elements of statistical learning (2nd ed.).* Springer. 2008. ISBN 0-387-95284-5.

[36] Jordan, M.; Thibaux, R.: *The Kernel Trick.* California University Berkeley: Lecture Notes. 2004. [Online; visited 2018-01-18].
Retrieved from: http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf

[37] Kim, E.: *Everything You Wanted to Know about the Kernel Trick.* 2015. [Online; visited 2018-01-18].
Retrieved from: http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick_blog_ekim_12_20_2017.pdf

[38] Lopez, P.: *SVM vs a monkey.* Quantdare. [Online; visited 2018-01-18].
Retrieved from: https://quantdare.com/svm-versus-a-monkey/

[39] Mahanta, J.: *Simple understanding gradient descent algorithm*. KDnuggets. [Online; visited 2018-01-16].
Retrieved from: https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html

[40] Markham, K.: *ROC curves and AUC explained*. Data School. 2014. [Online; visited 2018-03-11].
Retrieved from: http://www.dataschool.io/roc-curves-and-auc-explained/

[41] Martin, J.: *An Information Systems Manifesto*. Prentice Hall. 1984. ISBN 978-0134647692.

[42] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of Artificial Neural Networks*. Combridge Mass: MIT Press. 1997. ISBN 978-0-262-13328-9.

[43] Michal Hradiš, t. .: Master's Thesis.

[44] Ng, A.: *Decision Trees: A Disastrous Tutorial*. KDnuggets. [Online; visited 2018-01-13].
Retrieved from: https://www.kdnuggets.com/2016/09/decision-trees-disastrous-overview.html

[45] Olson, R. S.; Cava, W. L.; Mustahsan, Z.; et al.: *Data-driven Advice for Applying Machine Learning to Bioinformatics Problems*. 2017. [online; visited 2018-04-04].
Retrieved from: https://arxiv.org/abs/1708.05070

[46] Parmenter, D.: *Key Performance Indicators: Developing, Implementing, and Using Winning KPIs*. Wiley. 2010. ISBN 978-0470545157.

[47] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; et al.: *Scikit-learn: Machine Learning in Python*. *Journal of Machine Learning Research*. vol. 12. 2011: pp. 2825–2830.

[48] Samuel, A. L.: *Some Studies in Machine Learning Using the Game of Checkers*. *IBM Journal of Research and Development*. vol. 3, no. 3. 7 1959: pp. 210–229. ISSN 0018-8646.

[49] Sargent, D. J.: *Comparison of artificial neural networks with other statistical approaches*. *Cancer*. vol. 91, no. S8. 2001: pp. 1636–1642. ISSN 0008-543X.

[50] Saunders, R.: Book reviews: Evaluation of R&D processes. *Effectiveness Through Measurements*. *International Journal of Project Management*. vol. 17, no. 6. 12 1999: pp. 401–403. ISSN 02637863. [Online; cited 2018-01-13].

[51] Schapire, R. E.: *A Brief Introduction to Boosting*. 1999.

[52] Schneider, J.: *Cross Validation*. Carnegie Mellon University: Lecture notes. 1997. [Online; visited 2018-01-18].
Retrieved from: https://www.cs.cmu.edu/~schneide/tut5/node42.html

[53] Smith, L.: *An Introduction to Neural Networks*. University of Stirling. [Online; visited 2018-01-17].
Retrieved from: http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html

[54] Tape, T. G.: *Introduction to ROC Curves*. University of Nebraska Medical Center. [Online; visited 2018-03-11].
Retrieved from: http://gim.unmc.edu/dxtests/Default.htm

[55] Williams, A.: *Everything you did and didn't know about PCA*. Its Neuronal (math, neuroscience blog). [online; visited 2018-04-25].
Retrieved from: http://alexhwilliams.info/itsneuronalblog/2016/03/27/pca/

[56] Zbořil, F.: *Neural networks Madaline and Backpropagation*. Intern document; VUT FIT. [Online; visited 2018-01-16].
Retrieved from:
https://www.fit.vutbr.cz/study/courses/SFC/private/17sfc_2.pdf

[57] Zendulka, J.: *Data preprocessing*. Intern document, VUT FIT. [Online; visited 2018-01-16].
Retrieved from: https://www.fit.vutbr.cz/study/courses/ZZN/private/prednasky/02_03_predzpracovani.pdf

Appendix A

DVD content

All the source code was created within the organization. Since Python scripts are gathering data from databases of the organization, scripts are working only within their intern network. Scripts were modified to not reveal confidential information. The source codes on the DVD were implemented by the author of this thesis. DVD contains all the source codes written in Python, charts of current and current prediction models, source codes of the thesis text and the thesis in PDF format.

```
/
├── src/ ...  source codes in Python language
│   ├── classifier.py
│   ├── feature_selection.py
│   ├── main.py
│   ├── prediction.py
│   └── preprocessing.py
├── charts/
│   ├── current_models/ ...  charts of current prediction models
│   └── new_models/ ...  charts of new prediction models
├── doc/ ...  tex source codes of the thesis text
└── thesis.pdf
```

Appendix B

Predictions

In this chapter we attached two tables with predictions – early and late predictions. Rows in these tables represent projects ended in 2017. These projects have already ended and we know their final PG5 compliance. *PG5 Miss* column represents the true labels and we are going to compare our predictions against them. `PG5 Miss` column was calculated according to definition 3.1. Prediction models return a probability of the predicted class – we calculated the predicted label as follows

$$\text{prediction (PG5 Miss)} = \begin{cases} False & x \leq 50\% \\ True & \text{otherwise} \end{cases} \tag{B.1}$$

Table B.1: **Early major thermostat predictions**

#	PG5 Miss	GNB	DTC	RFC	ETC	KNN	ADA	XGB	SVM	3LNN	4LNN
1	True	100%	100%	100%	80%	80%	71%	88%	76%	82%	85%
2	True	100%	0%	80%	65%	80%	53%	57%	72%	53%	55%
3	True	100%	100%	80%	79%	80%	57%	79%	87%	84%	95%
4	True	100%	100%	100%	92%	100%	57%	90%	73%	71%	76%
5	True	100%	100%	80%	85%	80%	56%	89%	76%	88%	90%
6	True	100%	100%	90%	73%	80%	62%	77%	75%	78%	78%
7	True	100%	100%	90%	79%	100%	63%	76%	85%	93%	98%
8	True	100%	100%	90%	88%	100%	76%	87%	83%	94%	96%
9	True	100%	100%	90%	74%	80%	72%	88%	78%	85%	91%
10	True	100%	100%	90%	86%	100%	63%	74%	83%	88%	92%
11	True	100%	0%	80%	71%	80%	70%	86%	77%	83%	89%
12	True	100%	100%	90%	90%	100%	56%	86%	69%	60%	51%
13	True	100%	100%	90%	79%	80%	72%	88%	80%	88%	93%
14	True	100%	100%	100%	75%	80%	57%	89%	75%	85%	86%
15	True	100%	100%	90%	89%	100%	76%	87%	73%	70%	69%
16	True	100%	100%	100%	82%	100%	63%	89%	81%	92%	94%
17	True	100%	100%	100%	89%	100%	77%	86%	83%	95%	97%
18	True	100%	100%	100%	91%	100%	78%	86%	70%	63%	49%
19	True	100%	100%	90%	91%	100%	64%	89%	82%	91%	94%
20	True	100%	100%	100%	85%	60%	71%	87%	77%	82%	84%
21	True	100%	100%	40%	60%	80%	48%	54%	68%	39%	30%
22	True	100%	100%	100%	81%	80%	72%	87%	77%	83%	86%
23	True	100%	0%	60%	49%	60%	51%	57%	77%	58%	83%
24	True	100%	100%	100%	88%	100%	65%	82%	89%	91%	96%
25	True	100%	0%	80%	72%	100%	54%	68%	75%	62%	70%
26	True	100%	100%	90%	72%	60%	64%	85%	84%	91%	98%
27	True	100%	0%	70%	54%	60%	51%	75%	60%	29%	13%
28	True	100%	0%	90%	68%	60%	51%	66%	70%	47%	52%
29	True	100%	100%	90%	75%	80%	57%	81%	73%	74%	74%
30	True	100%	100%	80%	84%	100%	63%	86%	84%	86%	96%
31	True	100%	0%	80%	52%	80%	53%	40%	63%	23%	22%
32	True	100%	0%	80%	63%	80%	53%	59%	71%	45%	44%
33	True	100%	100%	100%	91%	100%	77%	88%	85%	95%	98%
34	True	100%	0%	40%	80%	80%	44%	67%	57%	24%	18%
35	True	100%	100%	60%	60%	60%	54%	63%	68%	53%	57%
36	True	100%	100%	80%	76%	60%	54%	85%	81%	88%	96%
37	True	100%	100%	60%	79%	80%	51%	78%	69%	59%	67%
38	True	100%	0%	70%	63%	60%	67%	42%	84%	55%	94%
39	True	100%	100%	70%	65%	80%	57%	56%	77%	50%	72%
40	False	100%	100%	100%	70%	100%	62%	87%	72%	77%	65%
41	False	100%	100%	40%	40%	60%	50%	46%	75%	52%	78%
42	False	100%	100%	80%	77%	100%	55%	74%	81%	72%	77%
43	False	100%	100%	90%	88%	100%	64%	89%	81%	90%	94%
44	False	100%	100%	60%	56%	80%	53%	53%	74%	55%	77%
45	False	100%	100%	90%	83%	100%	56%	75%	77%	74%	82%
46	False	100%	100%	90%	89%	60%	76%	87%	70%	74%	60%
47	False	100%	100%	70%	82%	100%	66%	59%	72%	35%	50%
48	False	0%	100%	80%	60%	100%	53%	74%	73%	19%	5%

Table B.2: **Early minor thermostat predictions**

#	PG5 Miss	GNB	DTC	RFC	ETC	KNN	ADA	XGB	SVM	3LNN	4LNN
1	True	0%	0%	94%	79%	100%	53%	51%	50%	67%	61%
2	True	0%	0%	74%	68%	100%	52%	51%	51%	71%	62%
3	True	0%	0%	84%	68%	100%	51%	51%	50%	71%	62%
4	True	0%	0%	86%	60%	100%	53%	51%	50%	62%	54%
5	True	0%	0%	68%	73%	100%	52%	51%	42%	60%	54%
6	True	0%	0%	84%	70%	100%	53%	51%	49%	65%	57%
7	True	0%	0%	74%	48%	82%	53%	51%	50%	58%	50%
8	True	0%	0%	74%	67%	100%	54%	51%	50%	62%	54%
9	True	0%	0%	64%	60%	82%	52%	51%	49%	60%	49%
10	True	0%	0%	80%	73%	81%	52%	51%	45%	61%	52%
11	True	0%	0%	64%	59%	100%	48%	51%	49%	59%	50%
12	True	0%	0%	78%	68%	100%	52%	51%	50%	67%	60%
13	True	0%	0%	92%	72%	100%	53%	51%	51%	71%	62%
14	True	0%	0%	94%	80%	100%	53%	51%	50%	73%	70%
15	True	0%	0%	74%	68%	75%	51%	51%	47%	57%	45%
16	True	0%	0%	90%	73%	100%	52%	51%	50%	71%	63%
17	True	0%	0%	72%	68%	77%	51%	51%	48%	61%	50%
18	True	0%	0%	86%	75%	100%	51%	51%	46%	68%	59%
19	True	0%	0%	62%	43%	100%	51%	51%	50%	54%	43%
20	True	0%	0%	86%	71%	100%	54%	51%	51%	68%	58%
21	True	0%	0%	58%	53%	100%	52%	51%	50%	55%	42%
22	True	0%	0%	66%	54%	79%	51%	51%	50%	59%	50%
23	True	0%	0%	68%	62%	81%	52%	51%	49%	60%	50%
24	True	0%	0%	78%	66%	80%	52%	51%	49%	54%	42%
25	False	0%	0%	56%	55%	82%	51%	51%	49%	58%	50%
26	False	0%	0%	86%	71%	100%	53%	51%	50%	64%	55%
27	False	0%	0%	74%	57%	83%	51%	51%	49%	57%	45%
28	False	0%	0%	78%	68%	80%	50%	51%	31%	49%	41%
29	False	0%	0%	88%	63%	86%	53%	51%	50%	61%	53%
30	False	100%	0%	58%	56%	100%	48%	51%	46%	48%	47%
31	False	100%	0%	80%	71%	100%	52%	51%	48%	78%	84%

Table B.3: **Early major security panel predictions**

#	PG5 Miss	GNB	DTC	RFC	ETC	KNN	ADA	XGB	SVM	3LNN	4LNN
1	True	94%	0%	61%	62%	74%	50%	61%	55%	65%	49%
2	True	4%	0%	64%	64%	83%	53%	58%	73%	95%	93%
3	True	3%	0%	16%	20%	28%	49%	17%	42%	29%	21%
4	True	77%	0%	47%	20%	52%	51%	49%	59%	69%	64%
5	True	0%	0%	17%	18%	18%	49%	13%	49%	43%	43%
6	True	90%	100%	60%	66%	65%	50%	47%	52%	58%	39%
7	True	79%	0%	30%	28%	56%	49%	35%	49%	43%	27%
8	True	90%	0%	50%	58%	64%	50%	57%	54%	63%	47%
9	True	86%	0%	26%	36%	54%	49%	27%	47%	44%	26%
10	True	0%	0%	31%	18%	28%	48%	23%	48%	5%	2%
11	True	0%	0%	61%	52%	75%	51%	50%	78%	96%	98%
12	True	92%	100%	67%	68%	81%	51%	70%	52%	59%	41%
13	True	57%	0%	40%	36%	34%	49%	54%	54%	62%	57%
14	True	84%	0%	56%	62%	74%	50%	58%	63%	83%	75%
15	True	76%	0%	53%	62%	82%	51%	60%	62%	83%	75%
16	True	70%	0%	21%	26%	44%	49%	13%	48%	42%	27%
17	False	60%	0%	23%	18%	43%	49%	19%	53%	52%	41%
18	False	6%	0%	21%	18%	35%	49%	17%	33%	10%	3%
19	False	100%	0%	64%	46%	65%	53%	72%	79%	71%	84%
20	False	63%	0%	29%	36%	46%	50%	28%	31%	12%	2%
21	False	27%	0%	27%	34%	45%	49%	15%	29%	9%	1%
22	False	6%	0%	29%	22%	23%	49%	48%	44%	40%	28%
23	False	0%	100%	37%	32%	31%	50%	30%	44%	7%	4%
24	False	0%	0%	29%	24%	31%	48%	27%	50%	29%	24%
25	False	100%	0%	36%	30%	37%	50%	19%	57%	5%	4%
26	False	54%	0%	10%	16%	26%	48%	30%	39%	24%	11%
27	False	0%	0%	30%	38%	37%	49%	13%	54%	76%	71%
28	False	93%	100%	63%	66%	83%	51%	78%	59%	75%	64%
29	False	1%	100%	27%	16%	33%	49%	27%	43%	21%	12%
30	False	0%	0%	19%	10%	22%	48%	22%	36%	1%	0%

Table B.4: **Late major thermostat predictions**

#	PG5 Miss	GNB	DTC	RFC	ETC	KNN	ADA	XGB	SVM	3LNN	4LNN
1	True	100%	100%	89%	84%	100%	83%	97%	97%	98%	100%
2	True	100%	0%	73%	82%	80%	65%	73%	88%	85%	98%
3	True	100%	100%	79%	86%	100%	80%	87%	94%	96%	100%
4	True	100%	100%	84%	80%	80%	67%	95%	82%	95%	100%
5	True	100%	100%	86%	80%	100%	80%	96%	97%	98%	100%
6	True	100%	100%	84%	84%	100%	69%	83%	75%	87%	87%
7	True	100%	100%	92%	90%	100%	74%	95%	98%	99%	100%
8	True	100%	100%	93%	88%	100%	80%	98%	97%	99%	100%
9	True	100%	100%	87%	82%	100%	83%	93%	94%	94%	100%
10	True	100%	100%	91%	90%	100%	74%	94%	93%	98%	100%
11	True	100%	0%	87%	72%	100%	80%	97%	96%	96%	100%
12	True	100%	100%	84%	84%	80%	69%	94%	52%	87%	87%
13	True	100%	100%	88%	80%	100%	80%	96%	98%	99%	100%
14	True	100%	100%	85%	78%	100%	74%	90%	92%	97%	100%
15	True	100%	100%	90%	78%	80%	85%	97%	58%	84%	90%
16	True	100%	100%	91%	86%	100%	74%	97%	97%	99%	100%
17	True	100%	100%	91%	88%	100%	76%	97%	93%	97%	100%
18	True	100%	100%	91%	88%	80%	75%	96%	43%	87%	75%
19	True	100%	100%	91%	88%	100%	79%	94%	91%	97%	100%
20	True	100%	100%	88%	84%	100%	80%	97%	97%	98%	100%
21	True	100%	0%	60%	76%	80%	75%	53%	71%	76%	68%
22	True	100%	100%	68%	72%	100%	31%	54%	71%	80%	70%
23	True	100%	0%	51%	64%	80%	65%	23%	74%	32%	56%
24	True	100%	100%	91%	90%	100%	74%	96%	94%	98%	100%
25	True	100%	0%	79%	76%	100%	76%	94%	96%	95%	100%
26	True	100%	100%	71%	82%	80%	64%	77%	91%	78%	99%
27	True	100%	0%	74%	72%	100%	86%	89%	85%	87%	98%
28	True	100%	0%	34%	60%	80%	24%	14%	83%	79%	91%
29	True	100%	0%	80%	76%	100%	78%	81%	87%	93%	98%
30	True	100%	100%	87%	84%	100%	70%	96%	96%	99%	100%
31	True	100%	0%	71%	62%	80%	74%	72%	54%	70%	81%
32	True	100%	0%	77%	80%	100%	71%	86%	76%	74%	82%
33	True	100%	100%	93%	88%	100%	80%	98%	98%	99%	100%
34	True	100%	0%	61%	70%	80%	78%	80%	48%	51%	48%
35	True	100%	0%	29%	54%	80%	25%	10%	59%	56%	25%
36	True	100%	100%	87%	82%	100%	78%	95%	98%	95%	100%
37	True	100%	0%	73%	76%	100%	80%	94%	88%	93%	100%
38	True	100%	0%	65%	62%	80%	50%	74%	89%	39%	99%
39	True	100%	100%	69%	68%	80%	71%	26%	41%	15%	3%
40	False	100%	100%	60%	68%	100%	19%	18%	20%	53%	1%
41	False	100%	0%	27%	52%	80%	25%	7%	93%	74%	99%
42	False	100%	100%	49%	68%	100%	20%	5%	19%	27%	0%
43	False	100%	100%	75%	78%	100%	28%	60%	77%	90%	90%
44	False	100%	0%	67%	56%	60%	68%	82%	47%	3%	1%
45	False	100%	100%	45%	60%	100%	18%	28%	18%	29%	0%
46	False	100%	100%	59%	68%	80%	29%	40%	5%	46%	0%
47	False	71%	0%	28%	46%	80%	25%	12%	7%	12%	0%
48	False	0%	0%	32%	60%	80%	24%	6%	2%	23%	0%

Table B.5: **Late minor thermostat predictions**

#	PG5 Miss	GNB	DTC	RFC	ETC	KNN	ADA	XGB	SVM	3LNN	4LNN
1	True	100%	100%	97%	92%	100%	76%	92%	97%	99%	100%
2	True	100%	100%	97%	90%	100%	76%	97%	100%	100%	100%
3	True	100%	100%	90%	76%	100%	74%	84%	94%	96%	99%
4	True	100%	0%	79%	80%	100%	54%	92%	73%	41%	4%
5	True	100%	100%	95%	96%	100%	74%	72%	100%	100%	100%
6	True	100%	100%	98%	100%	100%	75%	91%	100%	100%	100%
7	True	100%	100%	87%	86%	100%	74%	95%	98%	99%	100%
8	True	100%	100%	97%	84%	100%	74%	96%	92%	95%	99%
9	True	100%	100%	98%	88%	100%	76%	96%	98%	99%	100%
10	True	100%	100%	93%	92%	100%	74%	84%	98%	98%	100%
11	True	100%	100%	86%	80%	100%	68%	74%	98%	83%	99%
12	True	100%	100%	96%	84%	100%	74%	73%	98%	97%	100%
13	True	100%	100%	98%	86%	100%	75%	96%	94%	98%	99%
14	True	100%	100%	89%	98%	100%	74%	94%	100%	100%	100%
15	True	100%	100%	89%	88%	100%	71%	50%	94%	95%	96%
16	True	100%	100%	94%	92%	100%	76%	93%	99%	100%	100%
17	True	100%	100%	97%	92%	100%	74%	87%	97%	98%	100%
18	True	100%	100%	93%	92%	100%	59%	90%	92%	94%	97%
19	True	100%	100%	84%	94%	100%	73%	91%	100%	100%	100%
20	True	100%	100%	95%	88%	100%	76%	96%	91%	91%	94%
21	True	100%	100%	91%	88%	100%	57%	86%	87%	94%	93%
22	True	100%	100%	96%	88%	100%	73%	94%	96%	98%	100%
23	True	100%	100%	91%	84%	100%	72%	92%	98%	99%	100%
24	True	100%	100%	90%	96%	100%	75%	90%	98%	99%	100%
25	False	100%	0%	70%	72%	80%	55%	79%	81%	31%	2%
26	False	100%	100%	84%	80%	80%	67%	68%	86%	43%	88%
27	False	100%	0%	79%	86%	100%	57%	87%	86%	43%	14%
28	False	100%	0%	78%	84%	100%	54%	74%	82%	14%	1%
29	False	100%	0%	76%	82%	100%	55%	91%	81%	45%	10%
30	False	100%	0%	72%	94%	100%	51%	60%	100%	100%	100%
31	False	100%	100%	88%	86%	81%	69%	48%	100%	98%	100%

Table B.6: **Late major security panel predictions**

#	PG5 Miss	GNB	DTC	RFC	ETC	KNN	ADA	XGB	SVM	3LNN	4LNN
1	True	100%	100%	100%	77%	100%	57%	95%	81%	90%	96%
2	True	100%	100%	80%	73%	100%	63%	97%	92%	98%	100%
3	True	100%	0%	10%	34%	30%	49%	22%	54%	45%	50%
4	True	100%	100%	30%	69%	87%	51%	23%	67%	51%	74%
5	True	99%	100%	40%	45%	41%	72%	84%	97%	99%	100%
6	True	100%	100%	90%	84%	85%	57%	86%	73%	84%	90%
7	True	100%	0%	30%	47%	68%	51%	7%	72%	77%	91%
8	True	100%	100%	80%	85%	89%	54%	91%	81%	90%	96%
9	True	100%	100%	70%	83%	87%	61%	99%	84%	92%	98%
10	True	0%	100%	90%	58%	38%	53%	67%	0%	1%	1%
11	True	100%	100%	90%	81%	100%	54%	98%	92%	96%	100%
12	True	100%	100%	100%	96%	100%	60%	100%	83%	92%	97%
13	True	100%	0%	70%	67%	86%	56%	97%	98%	98%	100%
14	True	100%	100%	70%	80%	85%	63%	98%	92%	97%	100%
15	True	100%	0%	100%	81%	100%	70%	51%	79%	91%	96%
16	True	100%	0%	60%	55%	58%	51%	28%	79%	84%	95%
17	False	100%	100%	40%	35%	58%	51%	11%	15%	7%	0%
18	False	0%	0%	20%	13%	0%	42%	5%	2%	0%	0%
19	False	100%	100%	100%	91%	85%	54%	96%	59%	45%	40%
20	False	100%	0%	10%	15%	14%	43%	3%	1%	0%	0%
21	False	0%	0%	10%	11%	0%	36%	0%	0%	0%	0%
22	False	100%	0%	40%	35%	14%	44%	2%	33%	27%	10%
23	False	0%	100%	0%	11%	14%	39%	12%	6%	1%	0%
24	False	0%	100%	0%	19%	0%	34%	8%	10%	1%	0%
25	False	0%	0%	10%	11%	0%	35%	8%	0%	0%	0%
26	False	100%	0%	20%	33%	59%	43%	0%	61%	77%	88%
27	False	0%	0%	10%	31%	29%	47%	2%	62%	81%	99%
28	False	100%	0%	50%	24%	84%	54%	19%	29%	22%	2%
29	False	46%	0%	10%	25%	12%	50%	12%	31%	19%	13%
30	False	0%	100%	20%	17%	10%	42%	3%	0%	0%	0%

Appendix C

Features

This chapter contains tables with features – all features with their description and the other tables enlists features used during the training for each project group.

Feature Name	**Description**
5 Year NPV Current	Net Present Value after 5 years
ACT Association	whether the project receives a tax advantage by being developed by SW developers in Europe; indirectly, it is a SW project that has a tax advantage
Design Center	place, where the project is being developed
Design Group	whether HW or SW is the main part of the project
First Pass Yield Goal (%)	how many % of pieces may be defective in the production of testing pieces, the higher the target, the harder to meet the goal and then accomplish PG5
Greater China Project	project is serving China market
Gross Margin Current	gross margin
High Growth Regions	whether it is a product for high growth regions such as Africa or Russia; these products may have reduced quality or functionality for reduced price
Idea Type	whether it is a *customization* - development of new feature on existing product, *value engineering* - price reduction or *new product*
IRR Phase 3	Internal Return Rate at phase 3 - early quality issues
ISC Region	ISC (Integrated Supply Chain) - place, where the project will be manufactured
Market Attractiveness	how attractive is the product for given market
Micro Suppliers Count	count of micro suppliers
On Spec Flag	whether the project has a requirement change
Phase {1-4} Length	lengths of phases of project
Phase {1-4} Miss	whether there was a delay on phase gates
Planned Cycle Time Gate {1-4}	planned cycle time on gates {1-4}
Project Classification	whether it is a brand new, customization or incremental product
Project Cost {1-4}	project cost during phases 1-4
Project Cost Say-Do	whether the guaranteed maximum project costs are met; if a supplier cannot deliver some components on time, *Project Health* may change, but project costs stay the same
Project Health	captures the project managers subjective view on project's condition
Project Length	total length of the project
Program	name of program the project is part of
Tech-forward Content	whether the product has features which are new to the industry
Total Cannibalized Revenue	revenue gained by reducing the sales volume or market share of one product as a result of introduction of a new product
Total Project Investment	total project investment

Table C.1: Table showing selected features by feature selectors. In this table we merged all features in order to describe them, but each data group has a different subset of these features.

#	**Early Features**	**Late Features**
1	10 Year NPV Phase 3	ACT Association (Yes)
2	ACT Association (Yes)	Design Center (Golden Valley)
3	First Pass Yield (%) Goal	First Pass Yield (%) Goal
4	Greater China Project (Yes)	Greater China Project (No)
5	Gross Margin Incremental Phase 3	Idea Type (New Product)
6	HGR Mid Segment (No)	Planned Cycle Time Gate 2
7	Idea Type (New Product)	Planned Cycle Time Gate 4
8	Planned Cycle Time Gate 2	Product Cost Say-Do (R)
9	Product Cost Say-Do (G)	Program (Connected Home)
10	Program (Connected Home)	Project Classification (Incremental)
11	Project Cost - Gate 3	Project Cost - Gate 3
12	Project Health (R)	Project Health (G)
13	Total Project Investment	Total Project Investment
14	Phase 1 Length	Phase 1 Length
15	Phase 2 Miss	Phase 3 Length
16	Phase 3 Length	Phase 4 Length
17	Project Length	Phase 4 Miss
18		Project Length

Table C.2: Features of major thermostat projects selected during training.

#	**Early Features**	**Late Features**
1	ACT Association (No)	Design Center (Emmen)
2	Cycle Time Gate 3	Design Center (Golden Valley)
3	Design Center (Emmen)	Gross Margin % Incremental Current
4	IRR Phase 3	IRR Phase 3
5	Planned Cycle Time Gate 3	Idea Type (New Product)
6	Product Cost Say-Do ()	Product Cost
7	Project Classification (Customization)	Project Classification (Incremental)
8	Project Health (G)	Project Health (G)
9	Total Incremental Revenue Phase 3	Total Project Investment
10	Phase 3 Length	Phase 3 Length
11	Project Length	Phase 4 Length
12		Phase 4 Miss

Table C.3: Features of minor thermostat projects selected during training.

#	**Early Features**	**Late Features**
1	Cycle Time Goal	ACT Association (Yes)
2	Design Center (Shanghai)	Cycle Time Gate 4
3	Design Group (Hardware)	Design Group (Hardware)
4	First Pass Yield (%) Goal	First Pass Yield (%) Goal
5	Greater China Project (Yes)	Idea Type (New Product)
6	ISC Region (Americas)	Planned Cycle Time Gate 2
7	Planned Cycle Time Gate 3	Planned Cycle Time Gate 3
8	Program	Planned Cycle Time Gate 4
9	Project Classification (Incremental)	Project Category (Electrical/Mechanical)
10	Project Cost - Gate 3	Project Cost Say-Do (G)
11	Project Health (R)	Project Health (R)
12	Total Cannibalized Revenue Phase 3	Total Project Investment
13	Phase 3 Length	Phase 1 Length
14	Project Length	Phase 3 Length
15		Phase 3 Miss
16		Phase 4 Length
17		Phase 4 Miss
18		Project Length

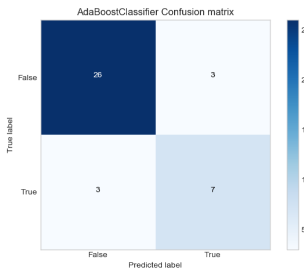
Table C.4: Features of major security panel projects selected during training.

Appendix D

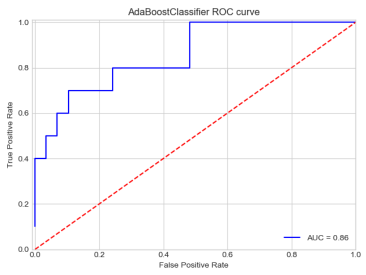
Candidate models

Figure D.1: **Early major thermostat candidate model (`ADA`) CV results**

(a) Confusion matrix



(b) ROC curve



(c) Learning curve

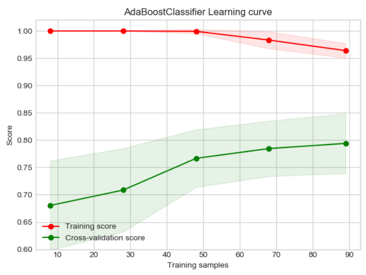
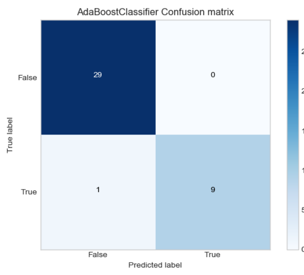
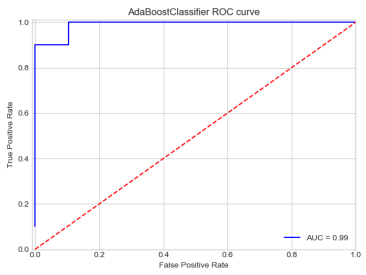


Figure D.2: **Late major thermostat candidate model (`ADA`) CV results**

(a) Confusion matrix



(b) ROC curve



(c) Learning curve

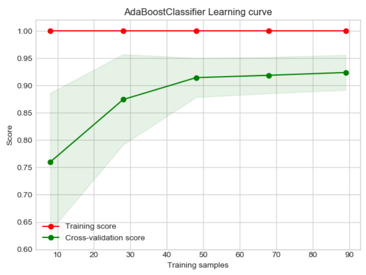
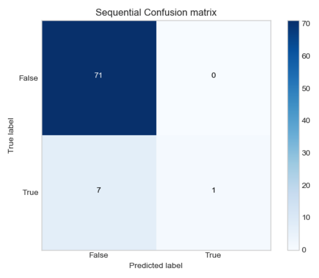
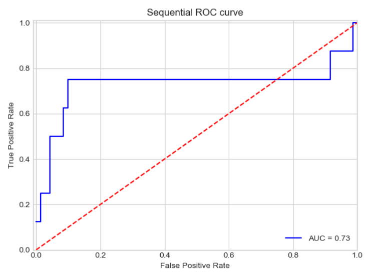


Figure D.3: **Early minor thermostat candidate model (`3LNN`) CV results**

(a) Confusion matrix



(b) ROC curve



(c) Loss

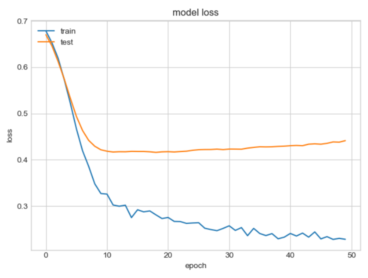
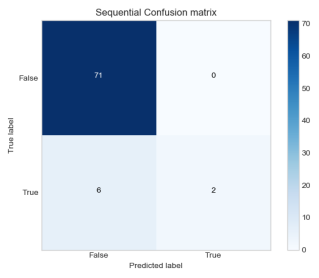
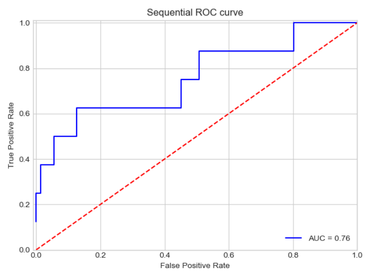


Figure D.4: **Late minor thermostat candidate model (`3LNN`) CV results**

(a) Confusion matrix



(b) ROC curve



(c) Loss

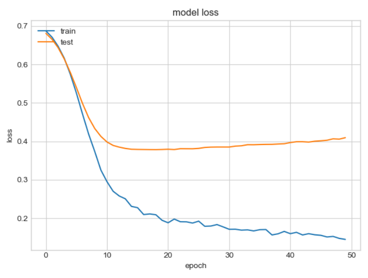
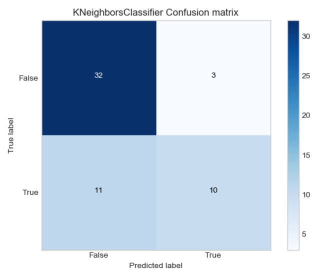
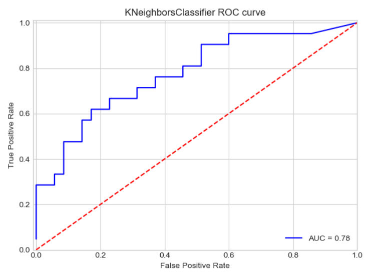


Figure D.5: **Early major security panel candidate model (`KNN`) CV results**

(a) Confusion matrix



(b) ROC curve



(c) Learning curve

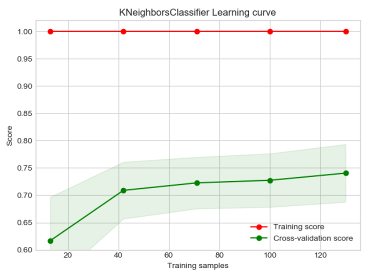
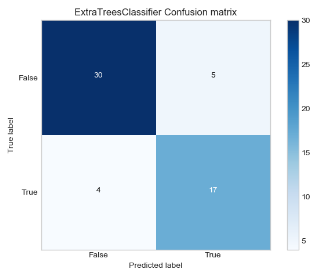
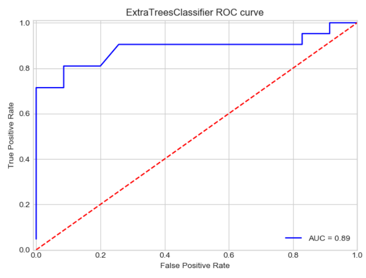


Figure D.6: **Late major security panel candidate model (`ETC`) CV results**

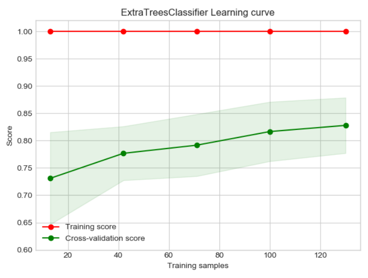
(a) Confusion matrix



(b) ROC curve



(c) Learning curve


ExtraTreesClassifier Learning curve
Score
Training samples
Training score
Cross-validation score