FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

# Klassifizierung von Anforderungen aus Ausschreibungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Wirtschaftsinformatik**

eingereicht von

**Alexander Schörghuber, BSc**
Matrikelnummer 01225890

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Mag. Dr.techn. Getrude Kappel
Mitwirkung: Prof. Dr. Walid Maalej
Christoph Stanik, MSc

Wien, 28. Jänner 2019

Alexander Schörghuber    Getrude Kappel

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

# Classification of Requirements in the Tender Process

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Business Informatics**

by

**Alexander Schörghuber, BSc**
Registration Number 01225890

to the Faculty of Informatics

at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Mag. Dr.techn. Getrude Kappel
Assistance: Prof. Dr. Walid Maalej
Christoph Stanik, MSc

Vienna, 28th January, 2019

Alexander Schörghuber Getrude Kappel

# Erklärung zur Verfassung der Arbeit

Alexander Schörghuber, BSc
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. Jänner 2019

Alexander Schörghuber

# Acknowledgements

I want to thank everyone who has contributed in any way to my study and the making of this thesis. First, I want to thank my advisors for their great feedback and inspiration. Thank-you to Prof. Dr. Kappel, for making this thesis possible and your great support. Thank-you to Prof. Dr. Maalej, for having me in this great environment and your motivational words. A special thanks goes to Christoph Stanik, who tirelessly guided me to the completion of this thesis. I am really grateful for your efforts.

I am truly grateful to Andreas Falkner, Gottfried Schenner and Herwig Schreiner for giving me the opportunity and the necessary freedom to write this thesis. I am thankful for all the inspiring discussions. Thanks to the requirements manager for your participation, especially in the interviews – I hope this thesis' outcomes serve you well. This thank-you also goes to the systems manager for providing background information.

A very big thank-you goes to my family, for supporting me throughout my whole life, especially as a student, mentally and monetarily. I can call myself truly blessed to see work as fun and room for development instead of a life-sustaining activity.

Finally, I want to thank the love of my life. Thank you for your support throughout this thesis. Thank you for your love, your support and all the joy you bring into my life. Thank you for you.

The work presented in this thesis has been conducted within the scope of the Horizon 2020 project OpenReq, which is supported by the European Union under the Grant Nr. 732463.

# Kurzfassung

Im Ausschreibungsprozess publiziert der Kunde eine Ausschreibung mit einer langen Liste von Anforderungen. Der Zulieferer überprüft jede Anforderung, ob und wie er dieser erfüllen kann. Diese Arbeit ist in Zusammenarbeit mit einem Zulieferer geschrieben.

Da nicht eine einzige Person alle Anforderungen beantworten kann, werde diese weiter an Experten zugeteilt. Diese Aufteilung basiert anhand deren Rollen innerhalb des Projekts. So eine Rolle kann beispielsweise Projektleitung oder ein Experte für ein Produkt des Zulieferers sein. In dieser Arbeit wird eine Rolle abstrakt Subsystem genannt. Diese Zuweisung wird aktuell manuell von einer einzigen Person erledigt, was einerseits lange dauert und andererseits für diese Person umständlich ist.

Um den Partner dahingehend zu unterstützen, wird eine auf Machine Learning basierte Lösung entwickelt, um solche Anforderung automatisch den jeweiligen Subsystemen zuzuweisen. In einer Literaturerecherche werden geeignete Ansätze identifiziert, welche anschließend in einem Benchmark verglichen werden, um die beste Konfiguration für vier Subsysteme zu finden. Diese Konfigurationen werden anschließend auf allgemeine Anwendbarkeit überprüft, indem sie auf fünf weiteren Subsystemen evaluiert werden. Die Gründe für Klassifizierungfehler werden in einem Interview mit der aktuell für die Zuteilung verantwortlichen Person aufgedeckt.

# Abstract

In the tender process, the customer publishes a request for tender (RFT) document containing a large list of contractly binding requirements. Suppliers need to process all of them and come up with solutions for each requirement. This thesis is written in cooperation with an industry partner on the supplier side.

Since not a single person can answer all requirements, these are further assigned to responsible experts. This split is performed based on roles within the project, such as project management or technical experts for some of the company's products. Within this thesis, such a role is abstractly called subsystem. This assignment is done manually by a single person, making this task tedious and time-consuming.

To support the partner, a machine learning approach is developed to automatically assign requirements to subsystems. In a literature review, suitable machine learning methods are identified, which are then compared in a benchmark to find the best configuration for each of four selected subsystems. These configurations are then checked upon generalization by evaluating them on five additional subsystems. The reasons for false classification are then identified in an interview with the person, who is currently in charge with the assignment.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation and problem statement

The market of safety equipment for railways is regulated by law when public sectors are the customers. To ensure open, fair, and bribery free competition, the customer, who is a national railway provider, is required to publish a Request For Tender (RFT). This document contains natural language documents of hundreds of requirements. For a certain time period, suppliers have the chance to answer with a tender by evaluating all requirements in regard to their ability to fulfill them, coming up with solutions and setting a final price. The customer then can choose from all submitted tenders the most suitable supplier with regard to requirements fulfillment and price.

This thesis is written in cooperation with Siemens AG Austria, our industry partner on the supplier side, that operates in this market. The approach this thesis introduces is generic and can be applied to an arbitrary use-case. The following paragraphs describe the partner's situation.

Once the sales team of the industry partner acquires an RFT document, the bid management team processes its requirements. The team consists of various roles, including a requirements manager (RM) and technical experts. All requirements need to be evaluated by experts from different roles like project management or technical experts for different domains, such as signaling hardware or interlocking software. In this thesis, such a role is called subsystem. Each subsystem has exactly one responsible technical expert. These experts have detailed knowledge about the company's provided products within their subsystem.

One fictive, but illustrative requirement is "The signal lamp must be compatible to the replaced lamp (24V/40W, 1500lm) and its case needs to be water resistant according to IP67". This requirement belongs to the subsystems "signaling hardware" (specifications about case) and "signaling lamps" (specifications about lamp). In this example, the expert of the subsystem "signaling lamp" can answer if the company offers a lamp that is compliant to the example requirement (to 24V, 40W and 1500lm), and the "signaling hardware" expert can confirm the company offers a suitable case that is water resistant according to IP67.

The requirements manager is responsible for assigning the requirements to subsystems and ensures that all subsystems' experts give a statement within a deadline. Currently, the industry partner manages this process as follows:

1. The sales department of a company delivers the RFT to the bid management team, containing Microsoft Word and PDF documents.
2. The requirements manager prepares these documents and imports them into the requirements management tool. Each paragraph is imported as a requirement, but is referred to as requirement candidate, as described in the next step.
3. The requirements manager categorizes each requirement candidate if it is an actual requirement or information.
   This extra classification is needed since non-requirements (such as headings from the document or notes introducing abbreviations) are also imported. The purpose of keeping this information is to provide better a understanding for the evaluation by the subsystems' experts but do not require a compliance statement from them.
4. The requirements manager assigns at least one subsystem to each actual requirement.
   This assignment requires reading every requirement and selecting the responsible subsystems from a multi-selection combo box, making this task time-consuming.
5. All experts of the various subsystems evaluate their assigned requirements.

The problem this thesis solves is the facilitation of step 4: The assignment of requirements to subsystems. This task requires the requirements manager reading every actual requirement. Depending on the project, an RFT document contains about 800 to 8000 requirements. Furthermore, the assignment requires selecting the responsible subsystems from a multi-selection combo box. The quantity of assignments and the user interface makes this task time-consuming and tedious.

## 1.2 Aim of the work

The aim of this thesis is to automatically assign requirements to subsystems by means of machine learning. Machine learning is used because the assignment can also be seen as a classification problem where requirements are classified into subsystems. Concrete, the task of classifying natural language text is referred to as text categorization. Since a computer or classification algorithm is not able to understand text natively, text categorization requires some preprocessing steps to represent the text, or in this case requirements, in a way so that a classification algorithm can handle it.

To address this goal, a benchmark is performed using four data sets of the industry partner, containing one subsystem per data set from ten real projects from the past. Common methods of text categorization are compared along with different classifiers from the literature. These methods and classifiers are optimized for each data set to find the best models' configuration, according to selected evaluation measurements. As stated later in section 6.1.1, these configurations are optimized upon $F_2$ metric. This may reveal configurations that work for all data sets.

These contributions will answer the following research questions which have been identified in first place:

- **RQ 1:** How accurate is an automated assignment of requirements to subsystems according to predefined measures?
- **RQ 2:** To what extent can the assignment of requirements to subsystems be generalized?
- **RQ 3:** What are the reasons of false classification?

## 1.3 Methodological approach

The methodological approach consists of the following steps:

1. **Literature review** with focus on related work:
   - In the area of railway (safety equipment) requirements
   - Requirements categorization in other domains, such as software engineering
2. **Literature review** with topics of the solution domain:
   - Text Categorization
   - Particularities of imbalanced data
   - Feature selection for requirements
   - Evaluation criteria for machine learning models
3. **Literature review** with the topics regarding the methodology.
4. **Quantitative analysis** of data sets
   The data is quantitatively analyzed by means of using descriptive statistics to exploit the characteristics of the data sets.

5. **Implementation** of a benchmark framework.
   Since many configurations are compared with different classifiers, a framework is needed that executes all those benchmarks semi-automatically and manages the evaluation results.

6. **Benchmark** of different models' configurations.

7. **Quantitative analysis** of benchmark results
   Based on the results from the benchmarks, the best configurations for each subsystem are found. These are then checked for exchangeability amongst the other subsystems and may lead to configurations that can be used in general for this use-case.

8. **Interview** with the requirements manager with the focus on requirements elicitation for the integration of such a classifier in this person's work flow. It also investigates what criteria are used when assigning a subsystem to a requirement.

9. **Qualitative analysis**
   To better understand how the proposed approach can be improved in a future iteration, a qualitative analysis of misclassified requirements is performed. In an interview, misclassified requirements are shown to the requirements manager. This person is asked, why he/she has (not) assigned those requirements to the subsystems in the past.

## 1.4 Structure of the work

The remainder of this thesis is structured as follows: In chapter 2, an interview with the requirements manager was held to get to know the use-case in more detail. This includes questions about details of the problem the industry partner's use-case is facing, as well as those that clarify the requirements of the thesis' intended solution.

Chapter 3 provides an overview of machine learning topics needed to develop the thesis' intended solution. This especially focuses on text categorization, evaluation criteria, and imbalanced data mechanisms. It also summarizes the related work on the topics railway requirements from RFT documents and requirements classification in general.

Subsequently, Chapter 4 describes the methodology followed. In detail it presents, the setup for the classification benchmarks to answer RQ1 and RQ2, as well as the interview and assessment with the requirements manager which addresses RQ3.

Chapter 5 shows how the data is extracted from the requirements management system and describes which attributes and information is contained in the given data sets along with some statistics.

Chapter 6 is an extensive documentation of the optimization benchmarks. First, the benchmark setup is explained in detail. The implementation of the benchmark framework is described. Afterwards, the benchmark results are presented and discussed. Those optimized configurations are then further compared in terms of generalization. This chapter answers research question one and two.

In chapter 7, the requirements manager assesses requirements that are incorrectly classified by the models. Findings, why those errors are made, are presented to answer RQ3. These reveal possible improvements to the classifier in future work.

The work then progresses with a discussion that proves practical usefulness but also critically reflects this work in chapter 8. It shows the impact of the thesis, compare it to related work and show its limitations and short comings.

This thesis finishes with a summary and an outline of future work in chapter 9.

# CHAPTER 2

# Motivating interview

At the beginning of this work, an interview with the requirements manager (RM) was held. The aim is to understand the problem and how it is currently solved. This arises some important information and risks for the suggested solution by this thesis.

This chapter lists the asked questions, the answers and discusses their impact on the intended solution. The answers are taken from the transscript of a scribe and are transposed into third person.

## 2.1 Question 1

**Question** Imagine, you have one requirement in the requirements management tool in front of you and want to make a subsystem assignment. Which criteria do you consider/what are you looking for in the requirement when assigning subsystems?

**Answer** The requirements manager reads the headings and the text of a requirement but does not need to understand the text completely. S/he tries to understand intuitively by reading some keywords. Some indicators (such as "GEN"/"RBC") at the beginning of a requirement help to categorize them.

**Impact** Keywords are used to classify requirements. Some indicator words help directly classify a requirement. For instance, "RBC" indicates a requirement for the subsystem RBC (which is short for "Radio Block Centre" and can be compared to an air traffic control tower at an airport). Since other keywords are not mentioned for the subsystems investigated in this thesis, this information is not helpful for us.

## 2.2 Question 2

**Question** Which difficulties are you currently facing in your workflow?

**Answer** Two main problems are listed: Translation and tedious labor.
The requirements originate from foregin languages and are translated into English or German. Sometimes, this leads to funny translations that reduce understandability and cost efford to understand.
The assignment of requirements is a tedious task. Assigning 8000 requirements in a few days is boring and the motivation drops after a few hours. S/he sets himself challenges (e.g. 4000 requirements assignments per day).

**Impact** The first problem is out of scope. The second answer reveals the pain this thesis tries to solve: To facilitate the tedious labor of manual assignment. This thesis aims to solve this pain by means of machine learning to automatically assign the requirements to subsystems. The answer also shows a possible integration of gamification aspects in the finished tool but is not of interest for this thesis.

## 2.3 Question 3

**Question** (a) What is the process when a subsystem is assigned wrongly and how often does this happen? (b) How do you know if a requirement belongs to multiple subsystems, but you assigned them to just a few (missed some subsystems)?

**Answer** (a) There are no incorrectly assigned requirements. If a subsystem thinks the requirement does not belong to it, it deselects itself via a proposal function implemented as plugin in the requirement management system. Optionally it proposes another subsystem. The requirements manager needs to guarantee that at least one subsystem is assigned per requirement and therefore can reject this proposal. Often such proposals are not accepted since the project demands from the subsystem to find a solution for the requirement. Unclear situations are sorted out via a telephone call or internal meetings.
(b) Overlooked subsystems (those that are not assigned but should be) are proposed by other subsystems that are assigned to this requirement. For instance, a requirement deals with an interface between two subsystems. If one subsystem notices, that the other one is missing, it can propose the missing subsystem via the same plugin. If possible, the requirements manager uses the four-eyes principle and asks the system manager to check for possible missed assignments.

**Impact** Answer (a) shows what happens with false positives[1] (FP), in case the requirements manager does not correct them. This scenario requires the RM to read the requirement, evaluate if the proposal of the subsystem is valid and accept or reject it.

[1]A requirement that is incorrectly assigned to a subsystem, see section 3.4.1

Two persons are involved, the subsystem's expert and the RM.
Answer (b) informs about the work flow to detect false negatives[2] (FN). Since at least one subsystem is assigned, the correct subsystem is guaranteed to be found eventually. For the RM it is easier to handle FP, since in this case the deselection of a subsystem from a requirement needs to be accepted. In case of FN, the correct subsystem needs to be selected, which requires to think about the appropriateness of the proposed subsystem, which may be more time consuming.

## 2.4 Question 4

**Question** How would you like to be supported in your daily work regarding subsystem assignments?

**Answer** Via a tool that preselects subsystems for the requirements that are then corrected by the requirements manager in the requirements management tool.

**Impact** The requirements manager wants to have a tool, that assigns the requirements to subsystems. In case this tool makes some errors, s/he can correct them within the requirements management tool. The impact is also related to question 5, where it goes more into detail.

## 2.5 Question 5

**Question** Imagine an automated system can help you assigning subsystems, how should it be integrated into your daily work? (Preselection, Recommendations, Subsystem reordering, highlighting, Export/Import functions?)

**Answer** The automated system does not necessarily need to be integrated into requirements management tool. The schema of the requirements can change over time, meaning the maintenance of such a 100% integrated tool is difficult for the requirements manager. This person wishes to get a preselection from the tool (externally) and to have this new information back into the requirements management tool as quick as possible. After the import, the preselections are evaluated and corrected. Preferably, a script is desired with no extra user interface.

**Impact** This answer is important for the integration of the assignment tool. No matter how it is integrated into the requirements management tool, the interface must be configurable so that it is robust against changes of the schema.
This answer also shows that the requirements management tool is accepted for the daily work. We therefore think, that the external classification tool should provide as less as possible user interaction as needed.

[2]A requirement that is incorrectly not assigned to a subsystem, see section 3.4.1

## 2.6 Question 6

**Question** If such an automated system is integrated, (a) which problems could occur in your opinion? (b) How critical and impactful do you see each of those problems?

**Answer** (a) The requirements manager is concerned about not being able to use the system. This could be caused by an update of the requirements schema, a system crash caused by software updates, tools are declared as malware by security settings, because they do not fulfill the newest security guidelines. It takes time until the next version is implemented and certified as allowed software. Such problems are time costly to solve for the requirements manager since they lay outside this person's sphere of influence.
(b) Worst case, all assignments are incorrect and the process needs to be completely manual as it is currently done. The one thing that could be really critical is that no deletions may happen in the database!

**Impact** The first part of the answer deals with worries about the availability of such an external tool. The availability seems restricted by the IT department. This is out of scope and therefore left out in this thesis.
The second part shows, that there is high risk when interacting with the database directly. We do not have direct access to database, only indirectly through the user interface using scripts. Alternations by such scripts are handled within the requirements management tool the same way as user input. We see low risk in using such a classification tool since the requirements management tool provides a versioning function so that everything can be undone.

## 2.7 Question 7

**Question** In Machine Learning, there is a trade-off between precision (how sure am I?) and recall (not to miss one). (a) What is more important for you? To not miss one subsystem assignment or be very sure about the assignment? In case recall is preferred, more subsystem assignments need to be deselected, and if precision is preferred, more subsystem assignments need to be selected. (b) Why is this more important for you?

**Answer** (a) The answer was clearly towards recall.
(b) Deselecting a subsystem is much easier for this person than selecting since s/he has to think if this requirement does not deal with those preselected requirements (deselection of a subset of preselected subsystems vs. selection from all subsystems). It is simpler to correct the preselections than selecting the subsystems.

**Impact** The first answer shows the direction the classifiers of the subsystem should be optimized.
The RM states, that deselection from a smaller preselected subset is easier for this person than selecting from the whole set of subsystems. This may lead to dangerous

assumptions since the classification also have a FN proportion. If only the preselected list of subsystems for a requirement (which contains true positives[3] (TP) and FP) is taken into consideration, FN may be ignored and not corrected. This is a potential risk.

[3]A requirement that is correctly assigned to a subsystem, see section 3.4.1

CHAPTER 3

# State of the art

The objective of this chapter is to given an overview on the topics addressed by this thesis. This thesis uses machine learning to categorize natural language requirements into subsystems. Therefore, we start by introducing text categorization and showing its different machine learning stages. Afterwards, methods of each stage are explained and suitable classifiers are introduced. Particularities of imbalanced data are then presented and methods for evaluating the performance of classification models are shown. Finally, this chapter ends by presenting related work in the context of requirements classification.

## 3.1 Text categorization

Text categorization is the is one activity in the area of information retrieval and is the task of labeling natural language documents with predefined categories. These categories, or also classes, are identified by their labels. For example in E-Mail classification, one label might be "Spam", the other "Ham". The training data (see section 3.3) provides pre-labeled E-Mails with examples belonging to both categories. Therefore, text categorization is part of the supervised learning, which assumes to have an initial corpus $\Omega$ (see section 3.1.1) with all instances labeled. [Agg18, KZP07, Seb02].

In general, there might be $k$ different categories, which leads to a categorization of supervised learning into [Agg18, SL09]:

- Binary: The input is classified into one of two non-overlapping classes.
- Multi-class: The input is classified into one of more than two non-overlapping classes.
- Multi-labeled: The input is classified into several of more than two overlapping classes.

- Hierarchical: The input is classified into one class which can be grouped into superclasses or divided into subclasses.

Multi-label classification can be split into multiple binary classification problems. To do so, the problem need to be transformed so that each label has its own binary classifier. This requires the categories to be stochastically independent [Seb02].

Sebastini [Seb02] provides a formal definition of what text categorization is and defines it as follows: It is the task of assigning Boolean values to each pair $(d_j, ci) \in D \times C$, where $D$ is a set of documents (see 3.1.1) and $C = \left\{c_1, ..., c_{|C|}\right\}$ a set of predefined categories (or classes). Furthermore, the author defines an unknown target function $\breve{\Phi} : D \times C \rightarrow \{T, F\}$ that describes how the documents should be classified, and a function $\Phi : D \times C \rightarrow \{T, F\}$, called the classifier (see section 3.1.7). The target values $T$ and $F$ stand for true and false, indicating if $d_j$ belongs to the category or not. The goal is to approximate $\breve{\Phi}$ by $\Phi$ so that $\breve{\Phi}$ and $\Phi$ "coincide as much as possible" [Seb02]. Measurements to define this coincidence can be found in section 3.4.

Since text cannot be directly interpreted by a classifier, an indexing procedure needs to be performed where a document is usually represented as a vector of term weights (see section 3.1.5) $\vec{d_j} = \left(w_{1,j}, ..., w_{|T|,j}\right)$ where $T$ is the set of terms (see section 3.1.2). This results in a dimension of the term-document matrix of $D \times T$. The weight $w_{i,j}$ is a value that, informally spoken, shows how important the term $t_i$ is for the document [Seb02]. This representation is known as the vector space model. It is fundamental since it allows to perform information retrieval operations such as document classification [KM12, SMR08].

The stages of the machine learning process are also shown in figure 3.1, which is based on [KM12] and are discussed in the following sections. Section 3.1.1 explain the term document, which is an important unit for the data collection stage. Preprocessing consists of tokenization (section 3.1.2), using n-grams (section 3.1.2), stop word removal (section 3.1.3) and reduction of word inflections (section 3.1.4)). The feature engineering stage comprise of calculating feature weights (section 3.1.5) and selecting relevant features (section 3.1.6) [APA$^+$17].

### 3.1.1 Documents

Documents are the input to an indexing process. In the above given formal description, a document $d_j \in D$ is a positive instance of a class $c_i$ if $\breve{\Phi}(d_j, c_i) = true$ and a negative instance if $\breve{\Phi}(d_j, c_i) = false$. To have a more simple wording, this thesis will refer to $d_j$ as a document or an instance. The collection of documents is also referred to as a corpus $\Omega$ [Agg18, Seb02].

An important step is to choose the correct document unit. If the units are too small, it is more likely that important passages are missed because the information is now spread across multiple documents. If the units are too large, too much information is included in one document resulting in irrelevant findings [SMR08].

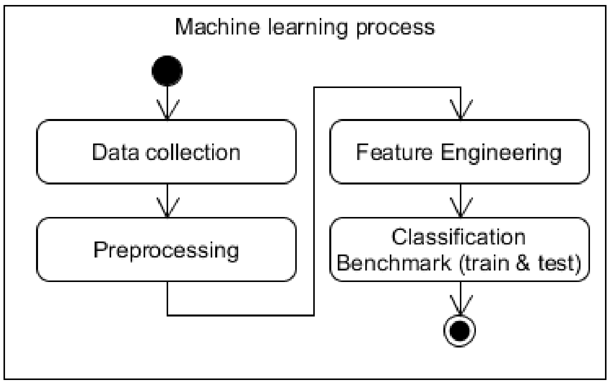


Figure 3.1: Machine learning process, based on [KM12]

In this thesis, each requirement represents a document, since we are interested in assigning subsystems to requirements. As stated in section 1.1, each paragraph is imported as requirement. Therefore, the document unit in this thesis is therefore one paragraph.

It would also be possible to group all requirements of one tender document and use this ensemble as document unit. Such a scenario would be news article classification, where the whole article belongs to a category. Contrary, it is also possible to select a word as document unit, resulting in multiple documents per given requirement. A possible use-case for this granularity could be spell-checking.

### 3.1.2 Tokenization and n-grams

In literature, a common step when dealing with text mining is tokenization—a process that splits sentences into a list of words, also called tokens [WIZ15]— as the first step during text preparation phase [HF06].

While humans do not face difficulties in tokenizing a text, a computer may have problems. Besides trivially splitting text at every white space (space, tab and new line), more challenges exist. For instance, some characters such as dots may be misinterpreted as they can either represent the end of a sentence or a decimal number. Another example would be an apostrophe ('), which can be a part of a token (e.g. hasn't or O'Neil) or opening or closing a quote [WIZ15].

These tokens are then used as columns in the vector space model. Therefore, a token is used as feature [Seb02]. The word "token" or "term" is used in this thesis when referring to word(s) and feature, when referring to a column in the vector space model.

To overcome this issue, it is of relevance to use the exact same tokenization for preprocessing the query [SMR08]. This is said for the context of information retrieval, applied to text categorization this means to use the same tokenizer for the building of the model as well as for making predictions on the built model.

When using single words as token in the vector space model, the ordering as well as their correlations between each other is lost. For instance, the sentence "A is faster than B" is represented equally to "B is faster than A" [SMR08].

To overcome this issue, n-grams can be used. N-grams use multiple sequential words as one token. $n$ is a parameter that sets the amount of sequential words to be used as token. For $n = 3$, the above sentence sentence "A is faster than B" would be tokenized to "A is faster", "is faster than", "faster than B". Using one word as token ($n = 1$) is also called uni-gram, using two words ($n = 2$) is also called bi-gram, and so on [Agg18, Seb02, CMS01, SMR08, WIZ15].

### 3.1.3 Stop words removal

Besides reducing complexity in dimension by applying methods shown in 3.1.6, a obvious thing to do is to remove words that almost never have any predictive value [SMR08, WIZ15]. Examples would be "a", "the", "it" or "they". These so called stop words can be removed from the documents before the vector generation process [WIZ15].

One possible strategy is to compile a list of stop words is to calculate the collection frequency (the number of occurrences of a token in all documents), and select the most common ones, often even by hand [SMR08].

It is also possible to use precompiled stop word lists, for instance in the Natural Language Toolkit[1] (NLTK), that contain the most common words of the language they are compiled for [BKL09].

### 3.1.4 Word inflection removal

The natural language contains words of different forms for grammatically correctness (e.g. "have", "has", "had"), the so called word inflection. Additionally, word families consist of derivated words with similar meanings (such as "functionality" and "functions"). The goal of this stage is to reduce these forms to a common base form. Two possible approaches are introduced and compared in the following paragraphs [SMR08].

**Stemming** is a crude heuristic process of chopping off the ends of a word. This achieves the goal most of the time but sometimes cut off the derivational affixes [SMR08]. This type of algorithm operates on a word level, not taking any context into consideration [Jiv11]. Lovins stemmer was the first popular stemmer introduced 1968. By now, Porter's stemmer is commonly used, although already introduced in 1980 [Jiv11].

**Lemmatization** achieves this goal by using morphological analysis of words and a proper vocabulary in the background. This process retrieves the correct base form, also known as "lemma" [SMR08]. A lemmatizer works on a sentence level, first determining the part of speech of a word in a sentence and then using this information to find the corresponding lemma [Jiv11].

[1] `http://www.nltk.org/`

The difference of both approaches can be seen by the example "saw" (depends also on the concrete algorithm, but should give a clarification):

- Stemming: returns "s"
- Lemmatizing: returns "see" or "saw", depending on how it is used in the sentence. Either you ...
  - ... saw a movie. ⇒ "see"
  - ... saw wood. ⇒ "saw"
  - ... buy a saw. ⇒ "saw"

Stemming mostly collapses derivationally related words and lemmatization commonly only collapses the different inflectional forms of a lemma. An example is shown in table 3.1.

Table 3.1: Application of different stemmers and lemmatizer

| original | Porter Stemmer | Lovin Stemmer | StanfordNLP Lemmatizer |
|---|---|---|---|
| operational research | oper research | oper research | operational research |
| operating system | oper system | oper system | operate system |
| operative dentistry | oper dentistri | oper dentister | operative dentistry |

### 3.1.5 Feature Weights

A row within the vector space model represents a vector consisting of feature weights [Seb02]. This section summarizes different feature weights $w_{i,j}$ from various papers [OK09, SMR08, Seb02, WIZ15].

#### Set of words and Bag of words

The most basic form is the set of words (SOW). If the word is present in the document, the weight is set to 1 where is in absence, the value is set to 0 [Seb02, WIZ15]. It is also referred to as feature presence [OK09].

The also very trivial, hence a little more advanced form, bag of words (BOW) simply uses the term frequency $tf\,(t,d)$. It counts the occurrences of a token in a document and uses it as value [SMR08, Seb02]. For some learning methods, this give a slightly better result [WIZ15]. It is also referred to as feature frequency [OK09].

There exists also a compromise where additional to the set of words, also the value 2 exists, indicating that this document has more than occurrence of the token [WIZ15].

These methods face a critical problem in case of tokens are present in many documents as they are then shown as relevant but in reality not providing any extra information [SMR08].

### Inverse document frequency

Its definition is shown in equation 3.1, were *df* ($t$) is the count of documents in which the term $t$ occurs. When $t$ is a rare term, *idf* ($t$) is high, where in contrast if $t$ is a common term, the *idf* ($t$) is low[Seb02, SMR08, JHS02].

$$idf\ (t) = \log \frac{|D|}{df\ (t)} \tag{3.1}$$

### Term frequency - inverse document frequency

Term frequency inverse document frequency (TF-IDF, see equation 3.2) is a commonly used weight in text categorization [OK09]. It discriminates the term frequency by multiplying the beforehand introduced weights with IDF. This results in a weight, that is highest, when t occurs many times within a low number of documents and lowest, when t occurs in all documents [SMR08].

$$tfidf\ (t, d) = tf\ (t, d) * idf\ (t) \tag{3.2}$$

### Term frequency - inverse gravity moment

Chen et al. [CZLZ16] introduce term frequency inverse gravity moment (TF-IGM) as feature weight. With the similar thought as TF-IDF, the authors use IGM (see equation 3.3) as discriminating function. The idea is to apply the gravity moment (GM) from physics cross domain on feature weighing. The more concentrated a term inter-class distribution is, the shorter the distance of the gravity center to the origin.

$$igm(t) = \frac{f_{k1}}{\sum_{r=1}^{m} f_{kr} * r} \tag{3.3}$$

To construct the IGM (see equation 3.3) for a term $t$, the term frequency for class $r$ ($tf_r(t)$) out of all $m$ classes needs to be calculated. Then this list is then sorted descendingly, where the first element of the list is referred to as $f_{k1}$, so that $f_{k1} \geq f_{k2} \geq ... \geq f_{km}$ holds.

$$tfigm(t, d) = tf(t, d) * (1 + \lambda * igm(t)) \tag{3.4}$$

$$rtfigm(t, d) = \sqrt{tf(t, d)} * (1 + \lambda * igm(t)) \tag{3.5}$$

Besides TF-IGM (see equation 3.4), also a rooted version RTF-IGM (see equation 3.5) exists. These two equations also introduce a coefficient $\lambda$. Its value is set empirically in the interval $[5; 9]$.

#### Related work

Benchmarks from Chen et al. [CZLZ16] with Support Vector Machines (see section 3.1.7) and k-Nearest Neighbors (k=15, see section 3.1.7) classifiers on the Reuters-21578 corpus show that RTF-IGM is slightly better than TF-IGM and both are superior to TF-IDF and TF.

O'Keefe and Koprinska [OK09] compared various feature weight methods and feature selection methods, as well as the use of Naïve Bayes (see section 3.1.7) and Support Vector Machines, in the context of sentiment analysis on a data set of 2000 movie reviews. Their result was that the set of words approach worked best.

### 3.1.6 Feature selection

A major difficulty in text categorization is the dimensionality of the feature space. This is usually high since it results from using all tokens as features. Feature selection is the task of removal of non-informative features [YP97]. The idea is to reduce the vector space from $T$ to $T'$, where $|T| < |T'|$, which is also called the reduced term set. This is beneficial since it tends to reduce over-fitting [Seb02].

Filters are commonly used technique, where the importance of a feature is measured by a metric and ones with the best scores are kept [GE03, JBB15]. These metrics are presented in the next section.

#### Metrics for feature importance

Various metrics have arisen over the past years. A good overview is given by Jović et al. [JBB15] for different application domains, of which three are described [JBB15, YP97, Seb02]:

- Information Gain (IG): The IG measures the number of bits of information obtained for category prediction by taking the presence or absence of a feature in a document into consideration. For each token, the IG is computed and all features where its values are below a certain threshold are removed.
- Chi-square $\chi^2$: The $\chi^2$ measures the dependence between a token and the class. We are therefore interested in those features where $\chi^2$ is highest.
- Term frequency (TF): Also the term frequency introduced earlier (see section 3.1.5) can be used to select features and is a simple but effective metric. Tokens where its $tf(t)$ are below a certain threshold are removed. Therefore, this metric selects the most common words.

#### Related work

Sebastini [Seb02] states that IG or $\chi^2$ can reduce the feature set by factor of 100 with no loss but possibly with a small increase of effectiveness.

Yang et al [YP97] compare IG, $\chi^2$, amongst others, by training a k-Nearest Neighbors classifier. They found IG and $\chi^2$ have similar effects on the performance and are most aggressive without loosing accuracy.

### 3.1.7 Classifiers for text categorization

This section present some classifiers that have reported in the literature [Agg18, PG12, Seb02, YL99] for text categorization.

#### k-Nearest-Neighbor Classifier

Those classifiers follow the simple principle of "Similar instances have similar labels". k-Nearest-Neighbor (kNN) classifiers identify the $k$ nearest neighbors of an instance according to a similarity function. The class reported with its most representants amongst the neighbors is chosen as prediction class. To compute the similarity, a function such as the cosine similarity is used. This classifier can be used for binary and multi-class classification since the class with the highest vote is predicted as well as for regression where the average of the neighbors is chosen [Agg18].

Nearest neighbor classifiers are also referred to as lazy learners, since they postpone most of the classification until the very end [Agg18], contrary to other classifiers that need to build a model before making predictions.

The parameter $k$ denotes the number of Nearest Neighbors to use without the instance to be tested itself. $k$ can be set by empirically finding the best value [Agg18].

1-nn ($k = 1$) classifiers are a special case. They are not very robust because they are sensitive to very specific data. As the size of the training set increases, the accuracy increases as well. An infinite large training set, 1-nn classifiers are able to approximate arbitrary class boundaries very well, only misclassifying noise or mislabeled instances. The main problem is that the amount of required instances grows exponentially with the dimension of the data set. Therefore, a smooth boundary should be achieved by selecting $k > 1$ [Agg18].

#### Naïve Bayes

The Naïve Bayes classifier assumes that the corpus is generated from a mixture of different classes. The observed data is assumed to be the outcome of this generative process. This classifier tries to estimate the parameters used for this generation so that the log likelihood of the data being created is maximized [Agg18].

The classifier got its name "naïve" from two assumptions made [JL95]:

- The predictive attributes are conditionally independent given the class.
- No hidden or latent attributes influence the prediction process.

Although those assumptions never hold in practical settings, the actual predictions are robust [Agg18].

The most common choices are the Bernoulli and multinomial distributions. One of their main differences is that the Bernoulli model uses the presence or absence of a term (SOW, see 3.1.5) where the multinomial model uses the term frequency (BOW, see 3.1.5). Without going into more details about the statistical background of this two approaches, only criteria which to choose is reported. The main factors are the length of each document and the size of the lexicon. If the document length is short and the lexicon small, the Bernoulli model is preferred, in the other cases the multinomial distribution should be used [Agg18].

### Support Vector Machines

A Support Vector Machine (SVM) chooses, among all surfaces $\sigma_1, \sigma_2, ...$ in the $|T|$-dimensional space, the surface that separates positive and negative instances the best. This is with the widest possible margin. Figure 3.2 shows the two dimensional case. Negative instances are shown as circles whereas the positive instances are indicated as plus. Many decision surfaces can be drawn but only one ($\sigma_1$ in this case) has the widest margin. The best surface is found by only taking few training instances into consideration, the so called "support vectors" [Seb02].

SVMs offer two important advantages for text categorization [Seb02]:

- Feature selection is often not needed since SVMs tend to be robust against over-fitting and scale up to high dimensions.
- Hyper-parameters[2] often do not need to be tuned since the default choice of parameters are very effective.

### Decision Trees

A decision tree is a tree whose nodes are labeled as features, branches separating from them are tests on the weight that the feature has in the vector space matrix and the leafs are labeled by the class names [Seb02]. The idea is to partition the data space into attribute regions that are biased towards a particular label [Agg18].

Decision trees define their split conditions of each node into branches so that a child node is dominated by a class. Evaluation criteria are similar to those in feature selection, where e.g. the word frequency can be taken into consideration. A split based on one attribute is called an univariate split, whereas a split based on more than one attribute is called a multivariate split. Those splits are applied top down until in each leaf contains only one class [Agg18].

[2] A parameter of the classification algorithm.

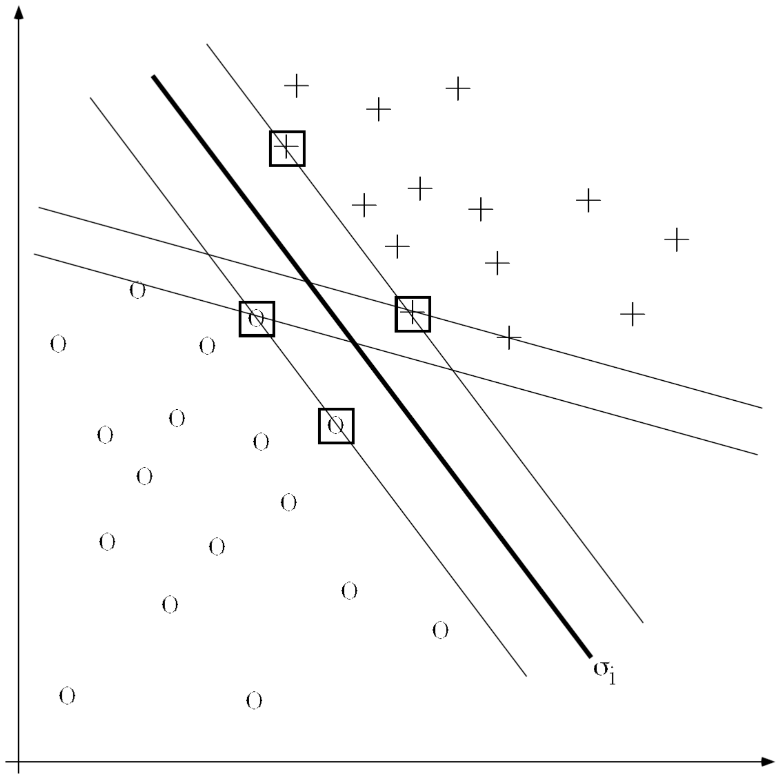


Figure 3.2: SVM decision surfaces (from [Seb02])

A prediction is made by traversing this tree top down until a leaf node is reached. The class assigned to this leaf is than the predicted output [Agg18].

When growing a tree to full height, meaning that all the training data (see section 3.3) is used to create a tree, the accuracy on the training data will always be 100%. To prevent this over-fitting, pruning can by applied where at the beginning a part of the training data is held out (as kind of prune test set) for tree construction. For each node, it is tested whether its presence increases or decreases accuracy. In the second case, the node is converted into a leaf [Agg18].

It is recommended to use univariate splits for short text and multivariate splits for longer documents [Agg18].

**Random Forest**

Breiman [Bre01] defines a Random Forest as "a classifier consisting of a collection of tree-structured classifiers $\{h(x, \Theta_k), k = 1, ...\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x." He refers to $h(...)$ as a tree classification function and to $\Theta$ as feature set.

The idea is to grow multiple trees but each of them only gets a randomly chosen feature sub set. This results in a forest. When a prediction needs to be made, all trees vote and the class with the most votes is the output. This leads to more robust predictions. The trees are grown to full height and are not pruned since over-fitting is prevented by feature selection [Agg18].

## 3.2 Imbalanced data

Imbalanced data is a dataset within which one or more classes have a much higher number of instances than the others. The rarest class is called the minority class. Contrary, the class with the most instances is called the majority class [YHX$^+$16]. In text categorization, the classes are mostly unevenly distributed. This means that there are fewer instances assigned to the positive class than to the negative class [Seb02]. Therefore, text categorization is affected by imbalanced data problems.

Imbalanced data emerges new challenges [HYS$^+$17], some of them are:

- Some classifiers, as decision tree or Support Vector Machine, are suitable for balanced data. When applied to imbalanced data, their results may be suboptimal, i.e. they provide a good coverage of the majority classes, but minority instances are distorted [LFG$^+$13].
- Minority instances may be treated as noise by the algorithms but also vice versa: Noise may be mistakenly identified as minority examples [BF15].
- Global performance metrics, such as accuracy, guiding the learning process induces bias towards the majority class, while rare sequences may remain unknown even if the overall model's precision is high [LGMTCOGB16].

There are three basic approaches to overcome these difficulties [LGMTCOGB16]:

- **Data level**: The goal is to rebalance the distribution of the classes by resampling which can be divided into three groups [HYS$^+$17]:
  - Over-sampling methods create new minority class's instances. Two commonly used methods are [HYS$^+$17]:
    - Randomly duplicating minority instances.
    - Synthetic Minority Oversampling TEchnique (or short SMOTE) as proposed by Chawla [CBHK02].
  - Under sampling methods discard instances of the majority class. One commonly used method is random under sampling (RUS), which removes random majority class instances.
  - Hybrid methods apply a combination of over sampling and under sampling methods.

- **Algorithmic level**: This approach modifies an supervised classifier to strengthen the accuracy towards the minority class. This results in a very specific classifier for solving one specific problem.
- **Cost-sensitive**: This approach assumes a higher cost for misclassifying a minority class and a lower cost for misclassifying a majority class. Cost-sensitive learning can be used in both, data and algorithmic, levels. Compared to resampling strategies, this approach is computationally more efficient. However it is much less popular than resampling: First, it is difficult to set the cost values since the true costs are unknown and cannot be given by experts. Second, resampling is much easier to implement and just needed to be configured and executed, since no modification of algorithms is needed [HYS+17].

Class imbalance can be measured in *Skew* and shows how many times negative instances occur more often than positive instances in a data set. *Skew* is defined as [JCDLT13]:

$$Skew = \frac{numberOfNegativeExamples}{numberOfPositiveExamples} \tag{3.6}$$

## 3.3 Data sets for evaluation

Text categorization relies on the presence of the initial corpus $\Omega = \left\{d_1, ..., d_{|\Omega|}\right\} \subset D$. To evaluate the classifier, the corpus needs to be split according to various approaches. Two approaches exist to evaluate a classifier [Seb02, WW17]:

- Train-test approach: The set of all documents is split in one train set $TV$ and one test set $Te$. The classifier is trained on the train set and under no circumstances may be trained on the test set, since it would tune the model upon the data we want to evaluate on. The train and test set are therefore mutual exclusive, meaning $TV \cap Te = \{\}$.
- k-fold cross-validation: This approach iteratively applies the train-test approach k times on the initial set. A model $\Phi_i$ is build for each pair $(TV_i = \Omega - Te_i, Te_i)$ where $i$ ranges from 1 to k. Each model is evaluated the same way as in the train-test approach but since there are now k models, their performance is averaged.

## 3.4 Evaluation criteria for classifiers

To be able to compare the performance of different classifiers, various performance measures exist. The following section shows an overview of some of the measurements, but does not claim to be complete. Various papers [Seb02, BTR16, LFG+13, HYS+17, SL09, BDA13, JCDLT13, Pow11] are summarized. The focus lies on binary evaluation measures since this thesis splits the multi-label classification problem into binary as further described in section 4.3.

Table 3.2: Confusion matrix in case of binary classification

| | | Classified as | |
|---|---|---|---|
| | | Pos | Neg |
| Actual Class | Pos | TP | FN |
| | Neg | FP | TN |

Bekkar et al. [BDA13] distinguishes between fundamental and combined performance measures, as well as graphical performance evaluation. Jeni et al. [GZC09] substantiate these divisions.

Many measurements are derived from the confusion matrix. Commonly used ones are introduced in the next sections [HYS$^+$17]. If not stated differently in the text, the range of a metric is zero to one, where one means best.

### 3.4.1 Confusion matrix

A confusion matrix shows the classification performance of a classifier with respect to test data. It is a square matrix where its dimensions are equal to the class count and therefore is not limited to binary classification. The rows represent the actual classes against the classified instances as columns. Table 3.2 shows the special case of binary classification. One class is designated the positive class (shown as "Pos"), the other the negative class (shown as "Neg"). In this context, the cells are called true positives (TP), false negatives (FN), false positives (FP) and true negatives (TN) [Tin10].

They are defined as follows [BDA13]:

- **TP**: Number of instances of positive class correctly identified as positive.
- **FN**: Number of instances of positive class incorrectly identified as negative.
- **FP**: Number of instances of negative class incorrectly identified as positive.
- **TN**: Number of instances of negative class correctly identified as negative.

Bekkar et al. [BDA13] state that in case of imbalanced data, the minority class is labeled as positive class.

### 3.4.2 Fundamental evaluation measures

Fundamental evaluation measures are derived from the confusion matrix. Their disadvantage is that it is not possible from looking at one measure to evaluate the whole classifier's performance since they only evaluate a very specific part [LFG$^+$13].

### Accuracy and error rate

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{3.7}$$

$$Error\ rate = \frac{FP + FN}{TP + FN + FP + TN} = 1 - Accuracy \tag{3.8}$$

Accuracy (see equation 3.7) the most used metric for evaluating classifiers [BDA13, BTR16, HYS$^+$17, LFG$^+$13]. It measures the overall effectiveness of a classifier [SL09]. The error rate (see equation 3.8) is the inverse to accuracy and measures the misclassification rate [BDA13].

Both are not widely used in text classification scenarios. Yang states that in this application domain the large denominator makes them insensitive to different ratios of TP and TN [Seb02].

This is also why these measures are not usable in imbalanced scenarios because of its bias against the majority class [BTR16, HYS$^+$17, GZC09, LFG$^+$13].

### Precision and Recall

$$Recall = \frac{TP}{TP + FN} \tag{3.9}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.10}$$

The recall (see equation 3.9 [BTR16, LFG$^+$13, SL09, Seb02]), or also true positive rate ($TP_{Rate}$) or sensitivity [BTR16], is the percentage of positive instances classified correctly [LFG$^+$13] or in other words the ability of a classifier to detect positive labels [SL09]. It is a measure of completeness [BDA13].

A high recall shows that many positive instances have been detected. It is desirable to increase recall without reducing accuracy [BDA13]. A downside of this measure is that it totally neglects the amount of TN and FP.

The precision (see equation 3.10 [BTR16, LFG$^+$13, SL09, Seb02]) is also called positive predictive value (PPV). It is a measure of correctness and shows out of all positive classified instances, how many are really actually positive [BDA13].

Precision and recall are often used together to evaluate a classifier since they complement another because of their inverse relationship. Unlike Accuracy and error rate, they do not sum up to 1. It is desirable to have both measures equal to 1 [BDA13]. It is well-known that high recall means low precision [Seb02].

They only focus on the positive class, although if interpreted both, they include some information about the erros made. Even if combined, they completely ignore TN [Pow11].

Both are applicable to imbalanced data since they do not mask bad performance on the minority class, such as accuracy does [BDA13].

**Specificity**

$$Specificity = \frac{TN}{TN + FP} \tag{3.11}$$

Specificity (see equation 3.11[BDA13, SL09, BTR16]), or also the true negative rate ($TN_{Rate}$), measures how efficient the classifier predicts the negative class [SL09]. It can be seen as recall but on the negative class.

### 3.4.3 Combined evaluation measures

Combined evaluation measures combine fundamental measures in order to overcome their disadvantage of being not meaningful when considered isolated [LFG+13, BTR16].

**F-measure**

$$F_\beta = \frac{(\beta^2 + 1) * precision * recall}{(\beta^2 * precision + recall)} = \frac{(\beta^2 + 1) * tp}{(\beta^2 + 1) * tp + \beta^2 * fn + fp} \tag{3.12}$$

The F-measure (see equation 3.12 [SL09, Seb02, LFG+13, BDA13]) is the harmonic mean of recall and precision [GZC09, BDA13]. The coefficient $\beta$ influences how much recall and precision are weighted [BDA13, BTR16, Seb02]:

- $\beta = 0$: F-measure equals precision.
- $\beta < 1$: precision is more important.
- $\beta = 1$: precision and recall are weighted equally
- $\beta > 1$: recall is more important.
- $\beta = \text{inf}$: F-measure equals recall.

This measure is commonly used and more informative about predicting the cases correctly that matters for the user by selecting a suitable coefficient. The value is high when recall and precision is high [BTR16], therefore a high value of F-measure is desired.

The F-measure shows more insight of the classifier's performance than accuracy and is suitable for imbalanced data [HYS+17, BDA13].

**G-mean**

$$GM = \sqrt{recall * specificity} = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}} \tag{3.13}$$

The G-Mean (see equation 3.13 [BDA13, BTR16, LFG+13]) is the geometric mean of recall and specificity. It attempts to maximize the accuracy of both classes with a good balance [BDA13, BTR16, LFG+13]. Due to the nature of the geometric mean, it is hard to contrast models according to their precision on each class [LFG+13].

It is suited for imbalanced data sets, since a poor performance on the minority class will lead to a low GM value. This measure is important to avoid over-fitting towards the negative class [BDA13].

### Balanced accuracy

$$Balanced\ Accuracy = \frac{recall + specificity}{2} \tag{3.14}$$

The balanced accuracy (see equation 3.14 [BDA13]) is the average of recall and specificity and therefore also the average accuracy of both classes [BDA13].

If the classifier performs equally well on both classes, the balanced accuracy is equal to accuracy (see section 3.4.2). If the accuracy is only high because the majority class outperforms the minority class, the balanced accuracy will drop [BOSB10].

### Matthews correlation coefficient

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{3.15}$$

Matthews correlation coefficient (or MCC, see equation 3.15 [BDA13]) considers mutually accuracies and error rates on both classes. All values of the confusion matrix are involved [BDA13].

Different from the other measures, this metric ranges from -1 to 1. 1 indicates a perfect model, 0 means the model is performing equal to a random classifier and -1 is the worst possible prediction [BBC+00].

It is declared as suitable for imbalanced data scenarios [BJEA17].

## 3.4.4 Graphical performance evaluation

Graphical performance evaluation methods are useful if the misclassification costs or the class distribution after deployment is unknown. While a numerical measure only represent the average performance over a set of operating points (combinations of costs and class distribution), these methods present a classifier's actual performance for a wide variety of different operating points [GZC09].

### Receiver operating characteristics curves

$$FP_{Rate} = \frac{FP}{FP + TN} \tag{3.16}$$

Receiver operating characteristics (ROC) curve is a two dimensional graph where the false positive rate ($FP_{Rate}$, see equation 3.16 [BTR16, LFG+13]) is plotted on the x-axis and the recall on the y-axis [GZC09, LFG+13, BTR16] for all possible threshold values

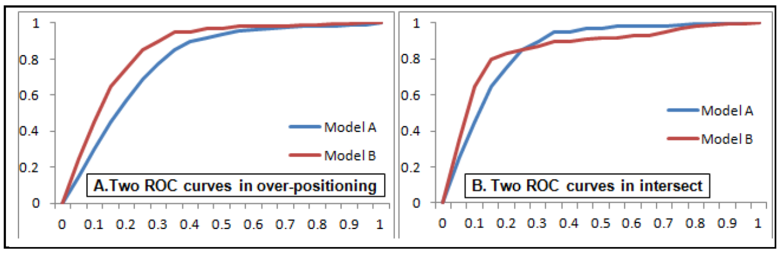


Figure 3.3: Comparison of different ROC curves, from [BDA13]

where the resulting points are joined to a curve [BDA13]. Sometimes the x-axis is also defined as $1 - Specificity$ [BDA13] which is mathematically equal.

This graph allows to visualize the trade-off between the $TP_{Rate}$ and the $FP_{Rate}$, evidencing that it is impossible to increase the $TP_{Rate}$ without also increasing the $FP_{Rate}$ [LFG+13].

The lower left point (0, 0) represents a classifier, that classifiers all instances as negative, where the upper right point (1, 1) indicates that all instances are classified as the positive class. A perfect classifier would be on the upper left (0, 1), indicating that all instances are classified correctly with no FP. A curve equally to the first median indicates that the classifier performs randomly [GZC09]. The more the curve is towards the upper left corner, the better is the classifier to distinguish between positive and negative class [BDA13].

However, it is not always simple to compare to classifier's ROC curves. The trivial case is where one curve dominates over the other. It is hard to select the best model if the two curves intersect. In the left graph of figure 3.3 Model B clearly dominates over Model A, where in the right graph the evaluation is more difficult [BTR16].

ROC curves are introduced by Provost and Fawcett to overcome accuracy's problem in imbalanced data sets [PF+97] and have been widely used in this context [HYS+17].

#### Area under ROC Curve

To overcome the difficulties of comparing different models by just looking at the ROC curve, the area under the ROC curve (AUROC) summarizes the ROC curve's performance into a singe numerical value. This value corresponds to the probability of correctly identifying the class of instance [Agg18]. A classifier will typically have an AUROC between 0.5 (random classifier) and 1 (perfect classifier) [BDA13].

AUROC is robust to imbalanced data [BJEA17].

### 3.4.5 Related Work

Sokolova et al. [SL09] give an overview of 24 different performance measures for binary, multi-class and multi-label for classification tasks.
They are then compared in terms of invariance properties. The authors investigate which

Table 3.3: Performances measures compared upon invariance, from [SL09]

| Confusion matrix manipulation | Invariant measures |
|---|---|
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} tn & fn \\ fp & tp \end{bmatrix}$ | Accuracy |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} tp & fn \\ fp & tn' \end{bmatrix}, tn \neq tn'$ | Precision, recall, F-score |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} tp' & fn \\ fp & tn \end{bmatrix}, tp \neq tp'$ | Specificity |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} tp & fn' \\ fp & tn \end{bmatrix}, fn \neq fn'$ | Precision, specificity |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} tp & fn \\ fp' & tn \end{bmatrix}, fp \neq fp'$ | Recall |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} k_1 * tp & k_1 * fn \\ k_1 * fp & k_1 * tn \end{bmatrix}$ | Accuracy, precision, recall, F-score, specificity, AUROC |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} k_1 * tp & k_2 * fn \\ k_1 * fp & k_2 * tn \end{bmatrix}, k_1 \neq k_2$ | Precision |
| $\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} k_1 * tp & k_1 * fn \\ k_2 * fp & k_2 * tn \end{bmatrix}, k_1 \neq k_2$ | Recall, specificity, AUROC |

measures are invariant in eight different confusion matrix manipulations as shown in table 3.3. This comparison is especially relevant in this thesis when deciding which measures to use for the evaluation of the classifiers in section 6.1.1.

Jeni et al. [JCDLT13] investigated how *Skew* may spoil performance metrics. They looked at the behavior, amongst others, of accuracy, F1-score, as well as the area under the ROC in skewed data.
The authors use a SVM Classifier and evaluated different data sets with *Skew* 3.52 to up to over 2000 and compared their results against the respective normalized data sets using random under-sampling with averaging.
Accuracy is completely unaffected by *Skew*, but especially vulnerable to bias from *Skew*. F1-score increased dramatically when applied on normalized data (from 0.45 to 0.77, from 0.23 to 0.68 and from 0.17 to 0.65, depending on the data set). Alpha's and kappa's improvement is not that high, meaning that these measurements are more tolerant. ROC was unaffected by the *Skew*.
They suggested to report both obtained performance measurements of the skewed scores and the *Skew*-normalized scores. If this is not possible at least to report the obtained scores and the *Skew*.
Applied to this thesis, this means that when measures are applied to imbalanced data, it is not possible to directly compare them to a balanced data application since the scores will be lower due to *Skew*.

## 3.5 Related work

This section introduces work that has been done in the domains of railway tender requirements categorization and some work that has been performed on the general case of requirements categorization. At the time this thesis was written, no one seems to have reported any requirement categorization on railway equipment tender documents or railway tender documents using machine learning yet. Although, a lot of research has been done on the topics of functional and non-functional requirements classifications (NFR) as well as NFR's into finer subcategories.

### 3.5.1 Requirements categorization

#### Railway industry

By the time of writing this thesis, no results in SpringerLink or IEEE found any results on the categorization of railway or railway equipment requirements using machine learning.

Ferrari et al. [FGR$^+$18] detect defects from requirements of the railway domain into various classes (such as anaphoric ambiguity, missing condition, etc.) using linguistic patterns.
This paper deals with requirements belonging to the same subsystems of railway safety equipment, such as IXL, that are also included in the data set of the thesis. Although it is not stated explicitly in the paper, we assume that the requirements taken are similar to the ones in our use-case.
They use a adapted version of the GATE tool [Cun02] to define those patterns. They use patterns to detect defects and also discard patterns to declassify false positives. Finally they compare their rules against the SREE tool from Tjong and Berry [TB13].
They have two data sets named pilot study, with 241 requirements including subsystems as automatic train protection (ATP) and an interlocking system (IXL), and a large data set, with 1866 requirements containing subsystems ATP, centralised traffic control (CTC) and Axle Counter. On the pilot study they reached an overall recall of 0.8833 and a precision of 0.6424. The large data set's recall is 0.8539 and precision 0.8316. The SREE tool increased recall (+0.1124) with the cost of lower precision (-0.0079).
This is a different application of classification as for the thesis and therefore not comparable. The authors do not report any special characteristics of the requirements.

Løkberg and Skogstad [LS05] present an approach for assigning safety requirements to railway signaling systems.
National Railway authorities publish metrics as the tolerable loss of life (TLL) that need to be fulfilled by the railway providers with the help of railway safety equipment. The authors show how this TLL can be transformed into the tolerable hazard rate (THR) which then gets assigned safety critical functions (SCF). Those SCF are specifications of what the equipment should be capable of.
The authors assign SCFs to the equipment using a mapping between two pyramids, one for the SCFs (which can be divided into sub SCFs) and one for the equipment and its

(sub-)components.
This work is out of scope since we try to classify requirements in subsystem classes. This work maps regulatory specifications on equipment and its components.

### Automotive industry

Ott et al. [Ott13] presents a similar use-case to the thesis' use-case at Mercedes-Benz. They identify relevant topics for a requirement in order to summarize all requirements to audit one topic at a time in a review session. To do so, an user creates a list of possible topics. Each requirement is then assigned to one or more topics. The inspector then chooses one topic and reviews it.
They present a tool-supported approach to automatically classify (German) requirements of product specification documents such of a doors closure module, extract related requirements and visualize the resulting classes. The categorization performance of Multinominal Naive Bayes (MNB) and Support Vector Machines is compared.
Existing studies showed that stemming or lemmatization of German texts does not improve prediction quality. The use of a stop word list (they used the one from `snowball.tartarus.org`[3]) only improved classification speed but not the results. The authors use 4-gram tokens. Furthermore Ott et al. perform two post-processing steps. The first is that he assumes that if a heading is assigned to a topic, all underlying requirements and headings can be assigned to this topic. The second is the self introduced, but unproven, "Recall+" approach. In this approach, unclassified requirements, which are structural near (based on distance) to an already classified requirement, are also assigned to the same class.
For evaluation, they used 10-fold cross validation, as well as precision and recall as evaluation measures on two different data sets. The best pre- and post-processing methods are 4-gram indexing with topic generalization. SVM reaches a recall of 0.83 and 0.8, with a precision of 0.66 and 0.64. MNB has a recall of 0.94 and 0.93 with a precision of 0.16 and 0.17.
Compared to this thesis, this work presents classifiers and preprocessing methods that work well for this data set. Since this is German text about specifications of a door module, we expect the data set to be different than for railway safety equipment. For future work, the two post-processing steps are of interest to see if they improve classification results also on this thesis' data set.

Winkler et al. [WV16] propose a convolutional neural network approach to automatically classify requirements whether they are requirements or information. This use-case is similar to this thesis' use-case step 3 (see section 1.1).
The data set consists of 10000 requirements extracted from 89 requirements specifications from the automotive industry in German language. Only the text body is used as feature. They define criteria that the origin documents must fulfill, such as the document must only describe a single component, the language must be German and the requirements

[3]Accessed on 13.09.2018. The domain has now possibly a new owner, since its logo says "24 hours in America" and no relations to stop words or machine learning in general can be seen.

must be pre-labeled. Those where randomly split, using 90% as training data and the rest as test data.
They use words to lowercase and stop word removal as preprocessing steps. To overcome the class imbalance of $Skew \approx 5$, the decide for an under sampling approach with a clustering algorithm.
Overall, they reached a precision of 0.733 and recall of 0.885 for the requirement class and a precision of 0.896 and recall of 0.754 for the information class.
This thesis uses no neural network but machine learning classifiers instead. The aim of this thesis is to optimize the models' configurations upon specific data sets. This paper uses "standard preprocessing steps", as stated by the authors. Since we are not investigating the classification of a requirement being information or an actual requirement, the final results of this paper are not comparable.

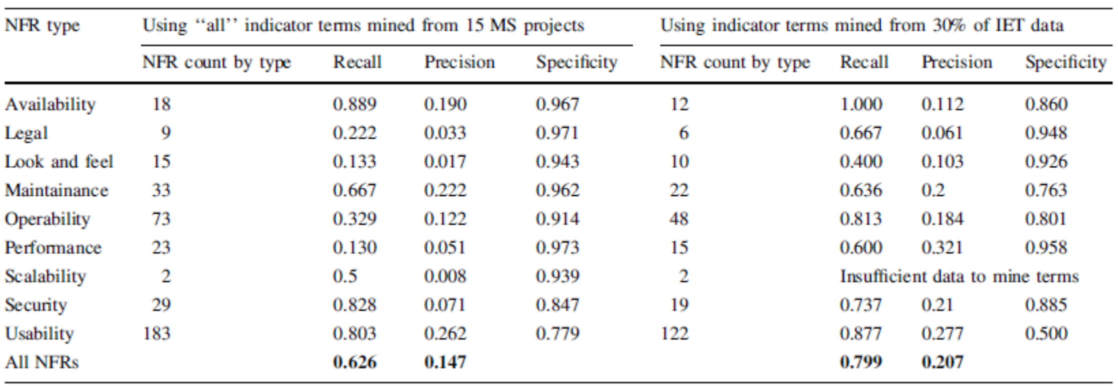
### Software engineering

Kurtanović and Maalej [KM17] classify requirements into FR and NFR of the RE17 data challenge "quality attributes (NFR)" data set using SVM.
They use random under sampling to achieve class balance, the removal of stop words as well as a lemmatizater from the Stanford Parser[4]. As features, they use bag of ngrams ($n \in \{1, 2, 3\}$), part of speech (POS) tags, n-grams ($n \in \{1, 2, 3\}$), clause and phrase uni-grams, Fraction of noun, verbs, adjectives, adverbs and modal words, text length, subtree count of a sentence and tree height of a sentence.
They achieve precision and recall of 0.92 to 0.93 on precision and recall on both classes when using 10-fold cross-validation.

Cleland-Huang et at. [CHSZS07] propose a new iterative, keyword-based NFR-classifier to classify requirements into NFR's categories such as "Availability", "Legal", "Look-and-feel", etc.
The authors first use a data set originating from students' projects using the Volere template. A second data set from an industry partner is used. The results can be seen in figure 3.4.
This thesis investigates the performance of different models' configurations and compares them. Since this paper introduces a new classifier that is not publicly available, we cannot compare its performance. The classes differ from that of the thesis, which tries to categorizes requirements into subsystems.

Hussain et al. [HKO08] classify requirements into FR and NFR using requirements from software requirements specification documents.
By using the decision tree showed in figure 3.5, the authors reach an accuracy of 99.35% using 10-fold cross validation. Recall is 1, meaning no FN, and out of 270 negative instances, the tree misclassifies 11. The used features are solely POS groups.
This indicates that POS tags are valuable features. This thesis uses only a keyword based approach leaving this set of features could be used for future work.

[4]http://nlp.stanford.edu/software/lex-parser.shtml, accessed 27.10.2018



| NFR type | Using "all" indicator terms mined from 15 MS projects | | | | Using indicator terms mined from 30% of IET data | | | |
|---|---|---|---|---|---|---|---|---|
| | NFR count by type | Recall | Precision | Specificity | NFR count by type | Recall | Precision | Specificity |
| Availability | 18 | 0.889 | 0.190 | 0.967 | 12 | 1.000 | 0.112 | 0.860 |
| Legal | 9 | 0.222 | 0.033 | 0.971 | 6 | 0.667 | 0.061 | 0.948 |
| Look and feel | 15 | 0.133 | 0.017 | 0.943 | 10 | 0.400 | 0.103 | 0.926 |
| Maintainance | 33 | 0.667 | 0.222 | 0.962 | 22 | 0.636 | 0.2 | 0.763 |
| Operability | 73 | 0.329 | 0.122 | 0.914 | 48 | 0.813 | 0.184 | 0.801 |
| Performance | 23 | 0.130 | 0.051 | 0.973 | 15 | 0.600 | 0.321 | 0.958 |
| Scalability | 2 | 0.5 | 0.008 | 0.939 | 2 | Insufficient data to mine terms | | |
| Security | 29 | 0.828 | 0.071 | 0.847 | 19 | 0.737 | 0.21 | 0.885 |
| Usability | 183 | 0.803 | 0.262 | 0.779 | 122 | 0.877 | 0.277 | 0.500 |
| All NFRs | | **0.626** | **0.147** | | | **0.799** | **0.207** | |

Figure 3.4: Overall results compared for each NFR category, from [CHSZS07]

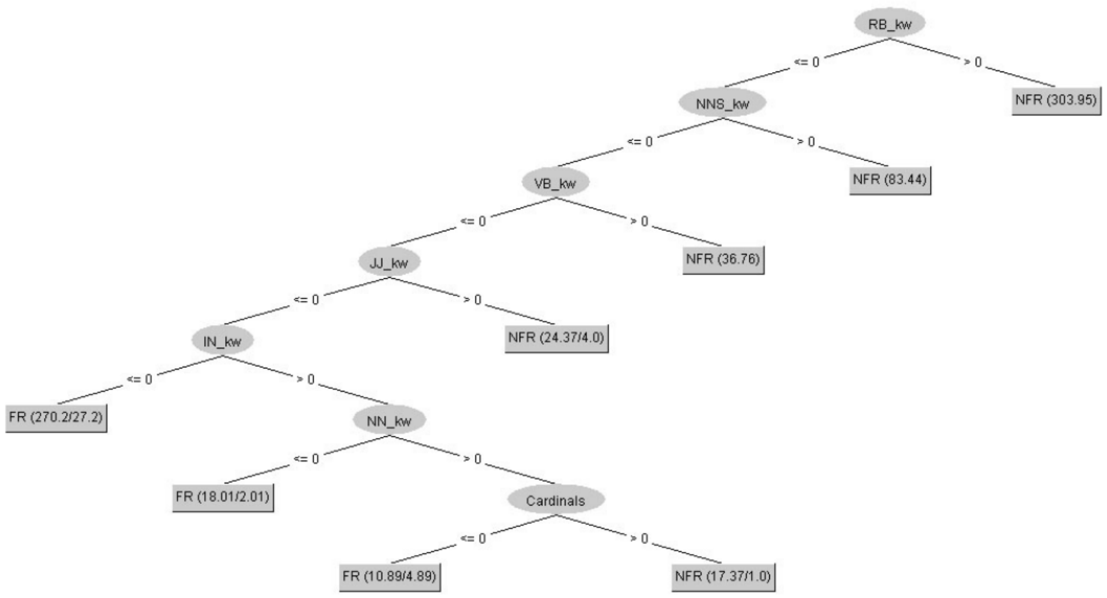


Figure 3.5: Overall results compared for each NFR category (from [HKO08])

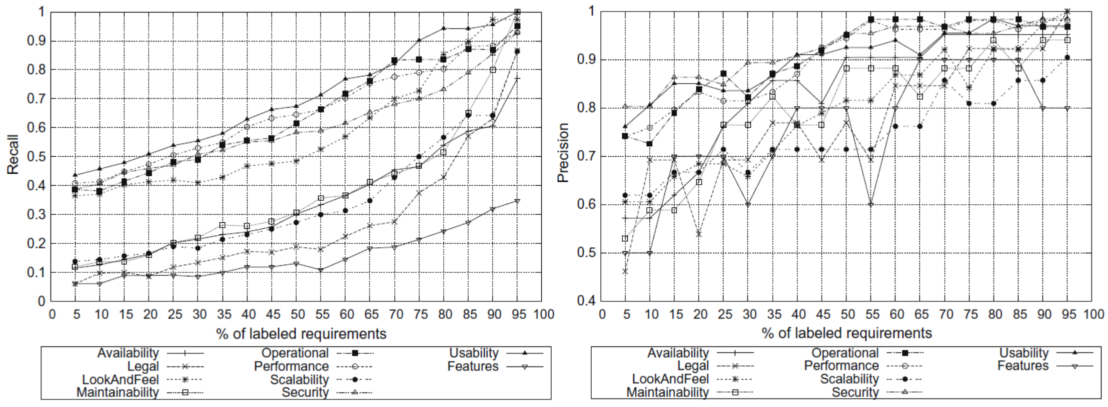


Figure 3.6: Overall accuracy compared across classifiers, from [CGC10]

Abad et al.[AKG+17] investigate how grammatical, temporal and sentimental characteristics of a sentence affect classification accuracy when classifying requirements into FR und NFR. The authors also compare different classification method's performance on classifying NFRs into sub-categories. A publicly available data set[5] is used.
To answer the first question they use the approach described in [HKO08] and achieve an increase in accuracy of 4.48%. Precision and recall gained 0.03 and 0.05 on the NFR class. This also improved performance when classifying into sub-categories. Precision and recall doubles for Multinominal Naïve Bayes and Latent Dirichlet Allocation (LDA). The comparison of different classification methods shows that the performance depends on the selected method. Multinominal Naïve Bayes achieves recall and precision of 0.91 and 0.9 where as LDA achieves 0.62 and 0.62. Other approaches, such as Biterm Topic Model and clustering approaches (Hierarchical, K-means and Hybrid) have precision and recall below 0.25.
This supports the thesis' need for investigating the preprocessing methods and comparing different classifiers to find the best working method. This thesis tries to evaluate a set of preprocessing methods and different classifiers to report first performance results on a data set from the industry. The preprocessing shown in this paper shows approaches for further improvements in future work.

Casamayor et al. [CGC10] propose a semi-supervised text categorization approach to classify FRs and NFRs, as well as NFRs into finer categories such as availability, look-and-feel, performance etc. They compare the Expecation Maximization (EM) suited for semi-supervised machine learning with Multi Nominal Bayes, kNN and Rocchio classifiers. For preprocessing, they remove stop words[6], use porter algorithm for stemming and for feature weight they use TF-IDF.
As data set they use 15 online available projects containing 255 functional and 370

[5]`https://terapromise.csc.ncsu.edu/!/#repo/view/head/requirements/nfr/nfr.arff`, Accessed 14.11.2018

[6]From `http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words`, the file is not available any more as accessed on 27.10.2018

non-functional requirements.
The results are presented in figure 3.6 and show that the EM classifier is superior to the other classifiers, but for the use-case of this thesis, the recall is too low.

CHAPTER 4

# Methodology

This chapter describes the methods used to answer the research questions presented in chapter 1. First we explain the initial motivating interview conducted with the requirements manager. We developed a benchmarking framework which allows to run reproducable multiple benchmarks semi-automatically. Afterwards, a benchmark is executed to quantitatively analyze and optimize the configuration of the machine learning model for each subsystems. We then check for generalization of these selected methods. Finally, the models are qualitatively evaluated in an interview with the requirements manager.

## 4.1 Motivating interview

In an initial motivating interview (presented in section 2), we aim to understand the process, identify risks and elicitate requirements how a tool that automatically assigns requirements to subsystems should be integrated in the daily workflow.

We decided to perform an interview as the research method because in the context of this work, there is only one requirements manager responsible for the whole process. Further, interviews help to get a deep understanding of the topic and the thinking process of the person being interviewed [PEM03].

Interviews can be distinguished in open or closed. In open interviews, a conversation is started with the stakeholder on the topic of the system of which the requirements need to be elicitated. In a closed interview, the stakeholder strictly answers the requirements engineer's predefined questions. It is also possible to start an open discussion based on the previously defined questions [PEM03]. We use this approach and define open questions on which a discussion is started.

## 4.2 Development of a benchmarking framework

To find the best performing machine learning model configuration for each data set (subsystem), multiple benchmarkings are necessary. To reduce the complexity of manually configuring and executing each run, a semi-automated benchmarking framework is implemented. This framework uses an existing machine learning toolkit. Besides many options, as for instance NLTK[1] for Phyton, we decided to use WEKA[2] in version 3.8.2 for Java. This tool is open source and contains many machine learning algorithms, as well as tools for preprocessing and classification.

Furthermore, WEKA provides a Java API (application program interface), which means that it can be called integrated into an arbitrary Java application. We decide to use this feature to integrate this toolkit in a benchmarking framework. The technical details are reported in section 6.2.

## 4.3 Quantitative analysis based on benchmarking

As later in section 5.1 the data analysis revealed, this is a multi-labelling problem. Therefore, we take the decision to transform it into a binary classification problem (see section 3.1) by splitting the collected data into multiple data sets, each for one subsystem, in section 5.4.

To find the best ml model configuration for each subsystem, various ml methods and classifiers need to be evaluated. We design an iterative benchmark for that purpose.

For this, we define the parameters of the configuration that can be optimized. We see this as a six dimensional optimization, where each dimension represents a step that modifies the data used to train a model. According to state of the art (see chapter 3), these are: sampling, removal of word inflection, use of n-grams, feature weight, removal of stop words, and feature selection. Each dimension provides multiple possible methods or algorithms, where each method is a possible parameter for this dimension. We refer to a vector, that contains one method for each dimension, as benchmark configuration/setup. The goal is to find the optimal configuration of methods for each classifier and subsystem.

The benchmark is further illustrated in figure 4.1 and consists of three phases:

- **Phase 1 - Benchmark configuration**: This phase creates a list of all possible methods for all six dimensions. Furthermore, the classifiers that are compared in the benchmark are selected as well as a set of subsystems on which those configurations and classifiers are evaluated on. Also performance measures, on which those evaluations are compared, are selected.

[1]`https://www.nltk.org/`, Accessed 10.11.2018
[2]`https://www.cs.waikato.ac.nz/ml/weka/`, Accessed 30.10.2018

- **Phase 2 - Baseline creation**: A benchmark configuration is chosen as baseline. The same is used for every classifier to benchmark every selected subsystem. These initial evaluation results are then be incrementally improved in the next phase.
- **Phase 3 - Iterative optimization**: For each subsystem and classifier, the best working benchmark setup is identified. To do so, the setup of the baseline is iteratively improved: Based on the order given in the next paragraph, methods of each dimension (e.g. sampling) are benchmarked and compared. The technique with the best result is selected and from now on used in this optimization. The next dimension is optimized until all six machine learning steps are optimized. Having done so results in one benchmark configuration that works best for one subsystem and classifier.

  This process can also be seen as a gradient optimization which finds a local maximum but not necessarily the global maximum. This could only be found by trying out all possible combinations. But because of limited computational resources, we decided for this approach. The authors of [PR07] define this as the usual approach.

Figure 4.1 also shows the belonging steps in the machine learning process (see section 3.1). The following order for optimizing each dimension of the benchmark configuration is defined:

- **Optimization step 1**: Sampling
- **Optimization step 2**: Removal of word inflections
- **Optimization step 3**: Tokenization (n-grams)
- **Optimization step 4**: Feature Weight
- **Optimization step 5**: Removal of stop words
- **Optimization step 6**: Feature selection

We selected sampling as the first optimization since this is the only stage that impacts the number of instances. Once the sampling strategy is selected, we investigate the tokens (Removal of word inflection/use of n-grams), look at their vector representation (feature weight) and see if the removal of stop words improves the classification model performance. Finally, the vector dimension reducing techniques are compared using some feature selection methods.

During a pilot of the benchmark, we found that, due to long run times, optimizing all stages for all classifiers and all subsystems is not feasible. We therefore decided to selected only one classifier for all subsystems. The classifier was chosen by comparing its performance on the first subsystem.

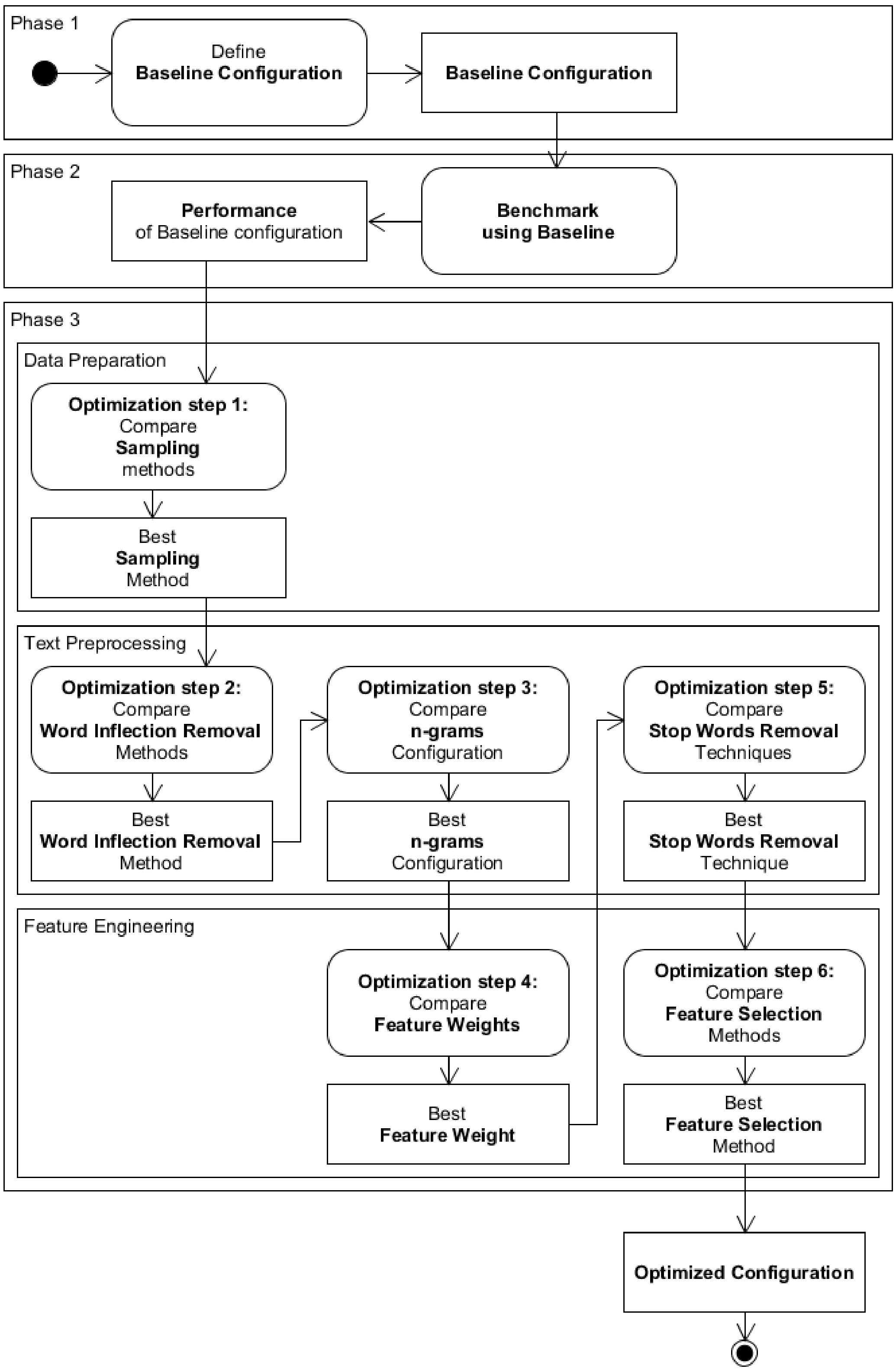

Phase 1
Define Baseline Configuration
Baseline Configuration
Phase 2
Performance of Baseline configuration
Benchmark using Baseline
Phase 3
Data Preparation
Optimization step 1: Compare Sampling methods
Best Sampling Method
Text Preprocessing
Optimization step 2: Compare Word Inflection Removal Methods
Best Word Inflection Removal Method
Optimization step 3: Compare n-grams Configuration
Best n-grams Configuration
Optimization step 5: Compare Stop Words Removal Techniques
Best Stop Words Removal Technique
Feature Engineering
Optimization step 4: Compare Feature Weights
Best Feature Weight
Optimization step 6: Compare Feature Selection Methods
Best Feature Selection Method
Optimized Configuration

Figure 4.1: Optimization process

## 4.4 Generalization

After having found the configuration that works best for each selected subsystems, the general applicability is tested:

- **Between subsystem analysis**: The configurations of each selected subsystems are used on the other selected subsystems using the same classification algorithm it was optimized on. This shows generalization of the configurations between those subsystems.
- **Analysis on other subsystems**: The optimized configurations of the selected subsystems are then used on five other subsystems, measuring their performance. This indicates if the configurations are practically useful just for the initially optimized subsystems or also on the other data sets.

## 4.5 Evaluation

The goal in this part is to get an understanding on the performance of results from the classification benchmark. These insights help to design future improvements of the classification process and show interesting characteristics of the requirements and subsystems in this domain.

To get this knowledge, a qualitative study (interview) was performed with the requirements manager. This person is asked, why s/he assigns this subsystem to requirements that the classifier not assigns to and vice versa.

For each investigated subsystem, ten false positive and ten false negative requirements were randomly chosen from the test set. The requirements manager assesses each of those requirements, knowing that s/he already (not) assigned them to the subsystem. For each picked requirement, a verbal explanation is given as well as marks of some phrases and keywords why this requirement would (not) belong to this subsystem.

CHAPTER 5

# Data sets

This chapter describes the data this work is based on. For that, we first describe how the data source is currently managed and how it was retrieved. Next, it explains how the data was cleaned and transformed into a machine readable format. Eventually, we use descriptive statistics to give insights about the content of the data.

## 5.1 Data storage

The industry partner uses IBM DOORS[1] to manage the requirements and the data is stored in its proprietary database. DOORS is short for "Dynamic Object-Oriented Requirements System" and treats every requirement as objects. These requirements are collected in so called modules, which can be seen as documents. This software also allows to create views, where users, in this case technical experts, only see requirements that are part of the subsystem they are responsible for. Furthermore, this tool allows for extensions in a proprietary script language: DXL[2].

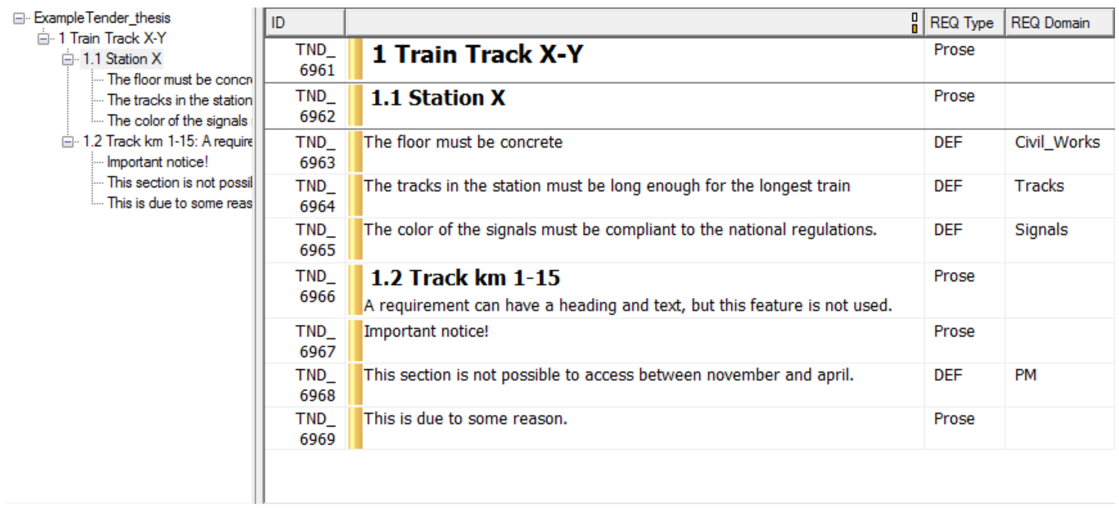
Figure 5.1 shows an exemplary screenshot of the tool, but with the toolbox cut out. The tool shows two columns. The left column shows the whole document in a tree view. The right one contains the main view, where the requirements are presented as rows with all its selected attributes as column. The text in bold represents a heading.

Figure 5.1 shows four attributes in the right view column:

- **ID**: This is the object id of each requirement with the prefix "TND" (short for "Tender"), which can be customized. This id is unique within one module, of which one project can have many of.

[1] `https://www.ibm.com/support/knowledgecenter/de/SSYQBZ_9.6.1/com.ibm.doors.requirements.doc/topics/c_welcome.html`, Accessed 03.01.2019

[2] `https://www.ibm.com/support/knowledgecenter/de/SSYQBZ_9.6.1/com.ibm.doors.configuring.doc/topics/c_dxl.html`, Accessed 03.01.2019



- ExampleTender_thesis
  - 1 Train Track X-Y
    - 1.1 Station X
      - The floor must be concr
      - The tracks in the station
      - The color of the signals
    - 1.2 Track km 1-15: A require
      - Important notice!
      - This section is not possil
      - This is due to some reas

| ID | | REQ Type | REQ Domain |
|---|---|---|---|
| TND_6961 | **1 Train Track X-Y** | Prose | |
| TND_6962 | **1.1 Station X** | Prose | |
| TND_6963 | The floor must be concrete | DEF | Civil_Works |
| TND_6964 | The tracks in the station must be long enough for the longest train | DEF | Tracks |
| TND_6965 | The color of the signals must be compliant to the national regulations. | DEF | Signals |
| TND_6966 | **1.2 Track km 1-15** A requirement can have a heading and text, but this feature is not used. | Prose | |
| TND_6967 | Important notice! | Prose | |
| TND_6968 | This section is not possible to access between november and april. | DEF | PM |
| TND_6969 | This is due to some reason. | Prose | |

Figure 5.1: Screenshot of IBM DOORS showing exemplary module

- **Column 2**: Includes the text and heading of a requirement. A requirement can have both, requirement TND_6966 shows this feature exemplary. Since the requirements are the result of an import function, only either of those attributes is set and therefore they are mutual exclusive in these data sets.

- **REQ Type**: This attribute distinguishes whether a requirement needs a compliance statement from a technical expert ("DEF") or not ("Prose"). It is the outcome of step three of the of bid process as described in section 1.1. For instance, requirement TND_6967 only indicates that an important notice follows but there is no actual requirement that can be fulfilled. Furthermore, headings are "Prose". The type of this attribute is a single selection enumeration with values "DEF" and "Prose". The default value is empty.

- **REQ Domain**: As stated in section 1.1, a domain is called subsystem in this thesis. This attribute contains a list of subsystems, whose experts need to give a compliance statement for the requirement. The type is also an enumeration with every subsystem needed for a project as possible value. Since every requirement can have multiple subsystems assigned, this attribute allows multi selection.

## 5.2 Data extraction

Besides many different output formats, including Excel, CSV and many more, the software also allows to run scripts in a proprietary script language called DXL. In the context of another project, an export script was already created in DXL, that exports all requirements of the current module into a JSON file.

Listing 5.1: Example JSON output from DXL Script

```
{
  "reqs": [
      ...
    {
      "toolId":"TND_6963",
      "text":"",
      "heading":"The floor must be concrete",
      "level":3,
      "reqType":"DEF",
      "reqDomains":"Civil_Works\nOther_Subsystem1"
    },
    ...
  ],
  "pds": [
    {
      "domainName":"Civil_Works",
      "number":0,
      "color":-1
    },
    ...
  ]
}
```

Each project is exported using this script, resulting in a JSON format as shown in listing 5.1 for the example module of figure 5.1. It contains one list of requirements and one list of all subsystems contained in this project. A subsystem has 3 attributes of which only the domainName is relevant and represents the name of a subsystem. Number and color are used to color code the subsystems inside IBM DOORS and the number is the internal representation of the enumeration. A requirement has the following attributes:

- **toolId**: The IBM DOORS internal identifier from the attribute "id".
- **text**: The text of a requirement.
- **heading**: The heading of a requirement. Text and heading attributes are mutual exclusive.
- **level**: The depth of a requirement within the tree representation in the left column (see figure 5.1). In IBM DOORS, the level of the first requirement is 1. If a sub-heading is added, its level and all its containing requirements is 2, a sub-sub-heading would be 3. In the example from figure 5.1, requirement with (tool)id TND_6961 has level 1, TND_6963 level 3 and TND_6966 level 2.

Table 5.1: Number of requirements of each projects

| Project | Number of requirements | Number of subsystems |
|---|---|---|
| Project 1 | 1525 | 36 |
| Project 2 | 8115 | 27 |
| Project 3 | 1000 | 5 |
| Project 4 | 6509 | 36 |
| Project 5 | 1854 | 24 |
| Project 6 | 1382 | 30 |
| Project 7 | 840 | 3 |
| Project 8 | 853 | 3 |
| Project 9 | 3819 | 16 |
| Project 10 | 2300 | 24 |
| Average | 2819.7 | 20.4 |

- **reqType**: This contains the value of the attribute "REQ Type".
- **reqDomains**: A list of all assigned subsystems from the attribute "REQ Domain" concatenated with "\n".

The requirements manager ran this script for each provided project. The export resulted in 18 projects in different languages. Eight of them purely English, two mixed English with German and Hungarian. The rest is German or German mixed with Hungarian. This thesis is only interested in English projects, so a first selection task was performed. To include the mixed projects, the others languages' requirements where simply deleted by hand. This results in ten usable projects with their number of requirements and subsystems shown in table 5.1.

## 5.3 Data cleaning

Data cleaning affects issues regarding language ambiguity in subsystem names where two terms exist for the same subsystem, e.g. "Axle Counter" and "Axle_Counter". This also leads to inconsistencies in some requirements texts, that include the subsystem name, for instance, "Subsystem1, Subsystem2, [...] please also look at [external documents] for additional figures".

To clean up the ambiguity in the subsystem names, an alphabetically sorted list is created with all subsystem names used in all projects. This list is then manually reviewed and similar subsystem names are grouped. This grouping is shown to the requirements manager and named correctly after the current subsystem name. For the above mentioned subsystem "Axle Counter", the term selected by this person is "Axle_Counter".

For applying these renamings to all requirements, a dedicated tool was created. It parses every given JSON file and updates the subsystem names in the "reqDomain" attributes of every requirement and every subsystem name in the "pds" array. This results in 59 unique subsystems.

To keep those requirements' texts consistent that mention an outdated subsystem name, another dedicated tool is implemented. The tool searches the requirements' text and heading attributes and generates a Markdown file where the subsystem name occurrences are marked bold for easy readability. There were only four requirements found with the above mentioned message, linking to an external document. Consulting the requirements manager confirmed this is bad practice. Therefore, these four requirements are manually removed in the JSON files.

## 5.4 Data transformation

The last step was to merge and transform these ten project files into usable relations for WEKA. ARFF is WEKA's file format for saving data sets. This step also performs the translation of a multi-label categorization problem into many binary categorization problems (as decided in section 4.3).

This step was also done by a self implemented tool that creates for each subsystem a separate ARFF file containing all requirements. In case of the attribute reqDomain contains the subsystem name, the requirement is labeled as "subsystem_name", indicating a positive instance. If the name is missing in this attribute, the instance is labeled "Non_subsystem_name".

Also the attribute text and heading are transformed into two other attributes: The text of heading or text is now contained in the new attribute text. A new attribute textIsHeading now indicates if the text origins from the heading or the text. Also an attribute fake_id was assigned to each requirement which is a trivial incremental number starting from one at each document. This results in the following attributes:

Each data set contains the following attributes:

- **fake_id***, numerical, contains a incremental number and helps for referencing instances.
- **toolid***, textual, contains the object identifier used inside IBM DOORS.
- **text**, textual, contains the text or heading of the requirement.
- **textIsHeading**, numerical, 1 indicates that the text is a heading, 0 that it is a regular text.
- **hierarchical_level***, numerical, ranges from 1-9 and shows the structural depth within the origin document.
- **req_type**, ordinal, with the possible values "DEF" or "Prose". This indicates if a requirement is declared as actual requirement ("DEF") that needs a compliance statement.

- **req__class**, ordinal, with the possible values "SubsystemName" or "Non__SubsystemName". This attributes sets the class wether it is a positive instance ("SubsystemName") or a negative one. The placeholder "SubsystemName" is actually replaced with the correct subsystem name.

The attributes marked with an asterisk (*) are removed later (see section 6.2.1).

## 5.5 Data analysis

This section analyzes the generated 59 data sets by means of descriptive statistics. It is split into three sections. First, a short introduction of the subsystems is given. It then continues by analyzing the data sets based on the number of instances and mainly focuses on class imbalance. The last part investigates the data sets based on their attributes.

As this thesis is scoped to nine of these data sets this section will report on those only[3]. The statistics, on the other hand, consider all data sets.

### 5.5.1 Meanings of subsystems

This section shortly introduces each of the nine investigated subsystems and gives an understanding which requirements are thematically assigned to them.

**PM** PM is short for project management. All requirements that are important to know for the project manager are assigned to this subsystem.

**IXL__HW** IXL is short for interlocking and this subsystem deals with requirements talking about hardware for interlocking systems.

**MMI__IXL** This subsystem deals with interfaces a human can have with an interlocking system. "MMI" is the German abbreviation of "Mensch-Maschinen-Interface", which is translated to human machine interface.

**Bid__Manager** This person is in charge of the bid process and therefore for the final price. Requirements that need a special solution, and therefore have an impact on the price, are assigned to this person.

**LED__Signals** This subsystem deals with requirements regarding signal lamps.

**Commissioning** This subsystem talks about requirements regarding the commissioning of the project.

[3]for more information see section 6.1 and section 6.5.3

Table 5.2: Number of positive instances, percentage and class imbalance per subsystem

| Position | Subsystem | #Pos | %Pos | *Skew* |
|---|---|---|---|---|
| 1 | PM | 6556 | 20.61% | 3.3 |
| 2 | MMI_IXL | 2885 | 9.07% | 8.77 |
| 3 | IXL_HW | 2440 | 7.67% | 10.56 |
| 8 | Bid_Manager | 1198 | 3.77% | 22.54 |
| 12 | LED_Signals | 870 | 2.74% | 31.41 |
| 20 | Commissioning | 362 | 1.14% | 76.89 |
| 30 | Build_Civ | 182 | 0.57% | 153.93 |
| 39 | OBU | 96 | 0.3% | 292.72 |
| 47 | IT_Network | 64 | 0.02% | 439.58 |
| | Total (all) | 31806 | | |

**Build_Civ** Requirements dealing with civil works and building are assigned to this subsystem.

**OBU** OBU is short for "on board unit". This unit is part of trains to wirelessly communicate with track-side equipment.

**IT_Network** Requirements assigned to this subsystem deal with IT networks in general.

### 5.5.2 Based on instances

In total there are 59 data sets, one for each subsystems. Every data set consists of 28197 instances. This is because the requirements are collected and merged from all projects. One data set is created for each subsystem. Requirements that belong to the subsystem are labeled as positive instances,

Table 5.2 gives a summary of the nine data sets. It shows the number of positive instances as well as their class imbalance by $Skew = \frac{\#Pos}{\#Neg}$. The number of negative instances is omitted, because it is always $28917 - \#Pos$. The table shows also an ordering in the first column. This represents the position when ordered all subsystems ascending according their *Skew*, meaning that a subsystem with a low class imbalance has a higher position number.

PM is the subsystem with the lowest *Skew* of 3.3. As stated above, only the investigated nine subsystems are presented in the table. When looking at all subsystems, the last three ones according to the *Skew* order have only four to five positive instances. The subsystems with the position 39 to 56 have a between 12 and 96 positive instances. The first ten subsystems ($position \leq 10$) have more than 1000 positive instances.

The number of positive instances are equal to the assignments of requirements to subsystems. These are done by the requirements manager by selecting the subsystem

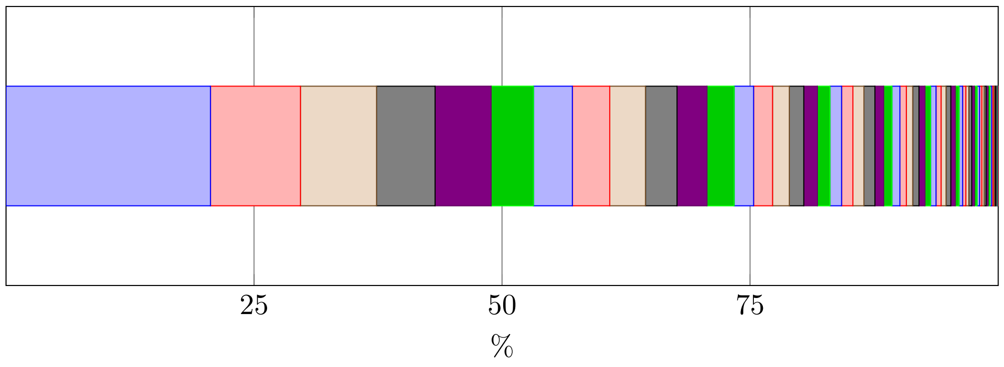


Figure 5.2: Bar chart showing each subsystem's percentage of total assignments as box

in a drop down field in IBM DOORS. Therefore, positive instances can also be seen as performed user interactions by this person. As table 5.2 shows, the total number of all assignments is 31806.

To get a better understanding, how the actual assignment work is distributed amongst the subsystems, the percentage of positive instances (see %Pos from table 5.2) is now closer investigated. This is graphically represented in figure 5.2, where every box represents a data set, when ordered by the number of positive instances descendingly. This shows that the first two subsystems (PM and MMI_IXL) cover more than 25% (29.68%) of all assignments. Summing the coverage of the six most common subsystems results in 53.17%. To further increase to above 75%, seven additional subsystems are needed, which results in a coverage of 75.35%. This means, that the last 46 subsystems only take up 24.65% of the requirements manager's work.

The *Skew* is further shown in a box plot in figure 5.3. This reveals a heavily right-handed distribution, meaning the mean of 598.74 is higher than the median of 153.93. 50% percent of the data sets (between quartile one and three) have $Skew \in [53.43; 396.37]$. The heavy tail on the right results from subsystems having 70 or less positive instances.

### 5.5.3 Based on Attributes

This section analyzes the data sets based on their attributes. It shows the frequencies of some attribute values and combinations of those.

Every data set contains the same instances only differentiating in the class label and fake_id. Therefore the absolute numbers of the following attributes are equal among all data sets:

- textIsHeading: There are 3154 headings in total.
- req_type: 22051 requirements are marked as "DEF" and 6146 are selected as "Prose".

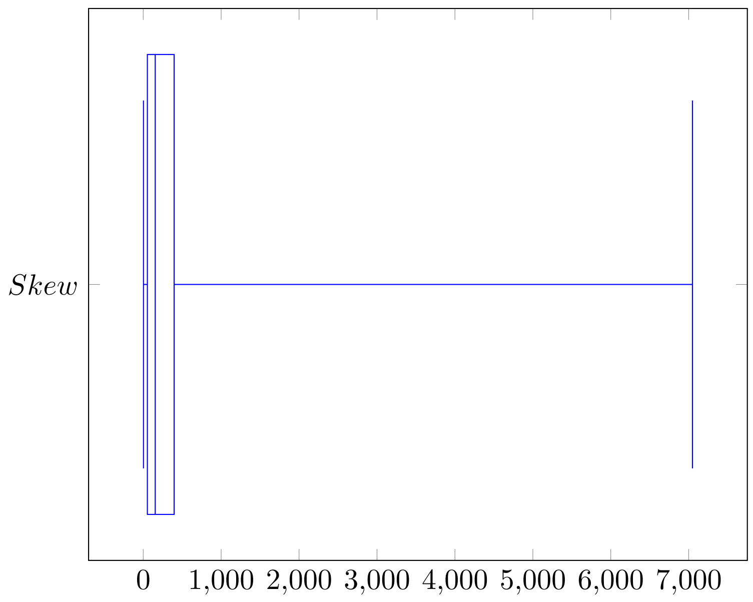


Figure 5.3: Box plot of *Skew*

Table 5.3: Number of requirements having subsystem assigned and are "Prose"

| Subsystem | "Prose" & Positive instance | Percent of positive instances |
|---|---|---|
| PM | 31 | 0.47% |
| MMI_IXL | 83 | 2.88% |
| IXL_HW | 161 | 6.6% |
| Bid_Manager | 111 | 9.27% |
| LED_Signals | 34 | 3.91% |
| Commissioning | 3 | 0.83% |
| Build_Civ | 8 | 4.4% |
| OBU | 0 | 0% |
| IT_Network | 0 | 0% |
| Total | 985 | 3.1% |

- textIsHeading & req_type: 3149 instances are headings and marked as "Prose", 5 are headings marked as DEF. 2997 requirements are "Prose" and not a heading, leaving 22046 instances as marked "DEF" and not a heading.

Table 5.3 shows the number of requirements that are marked as "Prose" but belonging to the class. In total there are 985 assignments in all data sets. The definition of "Prose" means that a technical expert of the subsystem does not need to give a compliance statement. Having subsystems assigned to requirements contradicts this definition. Since a total of 3.1% of all assignments belongs to this case, we decided to include these cases. Later, during the qualitative evaluation of the classifiers (see section 7.2.1), it turns out that these instances are actual false positives and should have been removed.

CHAPTER 6

# Quantitative analysis - Classification benchmark

This chapter explains the performed benchmark in detail. The goal of the benchmark is to find the best performing configuration for the machine learning model for each selected data set. To achieve this goal, we follow several steps: The first step being phase 1, where the possible parameters of the configuration are defined, the data sets and classification algorithms are selected, as well as the evaluation criteria on which to compare them. It then continues to illustrate the implementation of the benchmark framework. Phase 2 creates a baseline for each subsystem end each selected classifier. Based on those results, the configurations are optimized for each subsystem and each classifier in phase 3, which answers RQ1. This chapter concludes by answering RQ2—the investigation of generalizability.

## 6.1 Phase 1 - Benchmark configuration

This section deals with the selection of the possible benchmark configurations, which includes the selection of data sets, configuration parameters and classifiers.

### 6.1.1 Performance measuring

To evaluate the benchmark results, we have to select performance measures and define how the performance is measured. The success of the outcome mainly depends if the configuration of the model is tuned upon the right direction.

#### Performance measures

In the literature review (see section 3.4), we identified several ways to compare machine learning models: accuracy, error rate, precision, recall, specificity, F-measure, G-mean, balanced accuracy, matthews correlation coefficient, receiver operating characteristics (ROC) curves, and area under the ROC (AUROC). Upon these, we selected $F_2$ as main metric to consider, because of the during the interview identified importance of recall. Furthermore, we compare recall, AUROC, precision and specificity.

For the industry partner, recall is one of the most important metrics in the process of automatically assigning requirements to subsystems. The worst case is if a subsystem is not assigned to a requirement, and therefore the critical subsystem's evaluation is missing. This could lead to expensive costs due to the fact that the company contracts to the requirements' fulfillment. Penalties or external solutions must be expensively acquired.

A perfect classifier that supports this low risk behavior is a trivial acceptor: Every subsystem is assigned to every requirement. But this is not practically useful since there are only true positives and false positives. Therefore, also the precision needs to be taken into account to measure the FP.

We decided to use the $F_2$ metric as main measurement since it combines recall and precision into one value with weighting recall twice as much. We also look at the AUROC and specificity, because those measures, besides recall, evaluate the performance regardless to data imbalance. This is presented by Sokolova et al. [SL09] (see last row of table 3.3).

**Evaluation data sets**

In the literature review, two possible approaches are identified to evaluate a model: cross-validation and train-test (see section 3.3).

We look at the three-fold cross-validation results with the exception of sampling. In sampling we change the class distribution and therefore make some metrics as precision and $F_2$ not comparable. Therefore we compare the train-test results at this optimization step.

### 6.1.2 Selection of subsystems

As for the subsystems, we choose to investigate the two most common: PM (short for "Project Management") and MMI_IXL (short for "Men-Machine Interface to Interlocking"). This is because we want to help the requirements manager as much as possible with the contribution of this thesis. By supporting those two subsystems, 29.68 percent of his work (see section 5.5.2) is substituted by a classifier, meaning less tedious work.

To further compare, if similarities between a technical (MMI_IXL) and non-technical (PM) subsystem exists, two additional data sets are chosen: Bid_Manager as non-technical representative and IXL_HW (short for "Interlocking hardware") is selected as technical comparison subsystem. This means we will optimize the models' configurations for all four subsystems individually. This increases the coverage to 41.12%.

### 6.1.3 Parameters of a benchmark configuration

In this section, the parameters of a benchmark configuration are selected. It covers one paragraph for each optimization step in the same order as the benchmark iterations are executed.

**Optimization step 1: Sampling** For sampling methods, we decide to compare using no sampling strategy (using the raw imbalanced data), random under sampling (RUS), and SMOTE. For sampling in general, we choose the class distribution to be equal. For RUS, we randomly pick the same amount of the majority instances as there are for the minorty. To avoid selecting instances that lead to over or under-fitting, we repeat this process $max\left(\lceil Skew \rceil, 10\right)$ times with different random seeds. All metrics are then calculated individually and macro averaged. SMOTE has a parameter, which sets the amount of nearest neighbors to take into consideration when creating new synthetic instances. A fixed scaling factor of $\frac{numberNegativeInstances}{numberPositiveInstances} - 1$ is used and a variation of the nearest neighbors parameter from 2 to 5.

**Optimization step 2: Removal of word inflections** For the optimization step two, the stemming and lemmatization approach is compared. For this, the Porter stemmer and StanfordNLP[1] lemmatizer are selected. A shortcome is that the selected platform WEKA does only support stemming on word base. This means that the lemmatizer is only aware of the presented token and cannot see its context (e.g. the sentence).

**Optimization step 3: Tokenization** For this optimization step, we compare different n-grams, ranging $n$ from 1 to 5, meaning that we investigate five variations: $n \in \{1\}$, $n \in \{1, 2\}$, $n \in \{1, 2, 3\}$, $n \in \{1, 2, 3, 4\}$, $n \in \{1, 2, 3, 4, 5\}$.

**Optimization step 4: Feature weights** Besides the most commonly used feature weights, binary (0/1), term frequency (TF), and TF-IDF, we also decide to compare them to TF-IGM and RTF-IGM. These weights have a parameter that needs to be empirically set by trying $\lambda \in \{5, 6, 7, 8, 9\}$.

**Optimization step 5: Removal of stop words** As stop word removal can either positively or negatively affect the performance of a model (see section 3.1.3) we decided to include this step as part of the optimization. We use the stop word list provided by the NLTK framework[2] for the english language.

**Optimization step 6: Feature selection** For feature selection, we compare the results of IG and $\chi^2$ together with the term frequency (TF). The number of features (words) are in the range of 100 to 1500 in increments of 100.

[1]`https://stanfordnlp.github.io/CoreNLP/lemma.html`, Accessed 30.10.2018

[2]`https://www.nltk.org/`, Accessed 30.10.2018

### 6.1.4 Selection of classifiers

For classification algorithms, we compare Naïve Bayes (NB), Support Vector Machine (SVM), k-nearest-neighbors (kNN), and Random Forest (RF). Since RF is more robust than a single decision tree, we decided to not include decision tree. We choose the Bernulli NB classifier, which performs well on short text (see section 3.1.7). All four classifiers are natively supported by WEKA.

For all classifiers, we choose a naïve approach by using the default parameters of the WEKA implementations. Since $k$ of kNN is default 1, we decide to compare different values from 1 to 7 and select the best $k$ based on the results at the sampling step.

### 6.1.5 Selection of baseline

We define the following configuration for the baseline:

- Sampling: No sampling
- Removal of word inflections: Porter stemmer
- Tokenization: uni-gram
- Feature Weight: term frequency
- Removal of stop words: yes
- Feature Selection: term frequency with max words set to 1000

## 6.2 Implementation of a benchmark framework

This section describes the implementation of the framework used to execute the benchmark. The above described benchmark requires 70 runs per classifier and per data set (total of 1120), of which each run consists of preprocessing, feature engineering, model creation, and evaluation. Those steps can be performed by using the graphical user interface (GUI) provided by WEKA, but has some disadvantages:

- Using a graphical interface requires human operation. This would be a tedious task, because of the quantity of tasks.
- In WEKA, only one CPU core per invoked GUI instance is used[3]. To reduce the runtime, parallel data processing and model creation is desired.
- Not all methods, such as the feature weight TF-IGM, are supported by WEKA. This can be either solved by extending WEKA by implementing a plugin or provide the functionality as an external data processing application.

[3]Some classifiers, such as Random Forest, support multi-threading

To overcome these disadvantages, we decide to implement a semi-automated framework. Semi-automated because the comparison of the models' performance and setup of the configuration is done manually. This framework supports the above mentioned missing features and uses the WEKA Java API. Java version 8 is used.

We decided to split the framework into a data processing component and a classification component. The following sections explain the implementation in detail and focus on which functionality is already provided by the WEKA API and how we overcome some limitations for reproducability.

### 6.2.1 Data processing component

This component processes the data from the previously generated ARFF files from section 5.4. The configuration of which methods used for each dimension is specified via an input file. The resulting processed instances are then saved as ARFF file which are then used in the classification component (see section 6.2.2).

The following sections explain this component in more detail. The supported input arguments are presented, followed by the data schema of the benchmark configuration. Then the data processing is explained in detail which methods are used from the WEKA API or supplemented by own implementations. Finally, the directory structure is shown in which the processed data is managed.

#### Input arguments

This component accepts multiple arguments on startup:

- **baseDir**: The base directory which is used to store all processed files and to read the benchmark configurations (see section 6.2.1). Also the ARFF files resulting from the data transformation (see section 5.4) are expected to be in a subfolder called "RAW".
- **numberOfCores**: This is the number of threads that are maximal simultaniously used to process all given benchmark configurations.

#### Benchmark Configuration

To provide better configurability, we decided to specify the algorithms/methods to be used to process the data for each optimization step in a JSON file. The used schema is shown in the class diagram in figure 6.1.

The class ExperimentSetup represents the benchmark configuration explained in section 4.3 where each optimization step has one Setting in the ExperimentSetup class. For better readability, the composition between Setting and ExperimentSetup is shown only once but resolved into six attributes in ExperimentSetup.

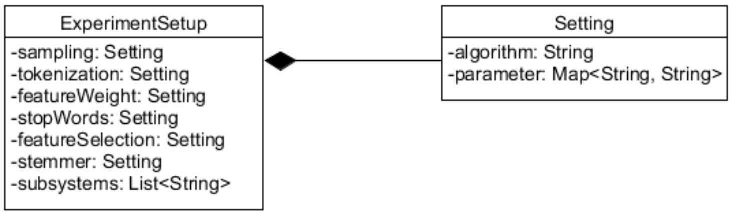


Figure 6.1: Class diagram of ExperimentSetup and Setting

A setting consists of an attribute algorithm which contains the name as abbreviation of the used algorithm (e.g. "RUS") and a map that takes the parameter name and its value as string. An example is shown in listing 6.1, where term frequency (TF) is used as feature selection method with a parameter $maxWords = 1000$. This representation allows a very flexible configuration.

Listing 6.1: JSON represenation of example Setting instance for a feature selection method

```json
{
    "algorithm": "TF",
    "parameter": {
      "maxWords": "1000"
    }
}
```

## Data processing

Those benchmark configuration files are then parsed via Gson[4] from the benchmark configuration directory which creates one ExperimentSetup instance per file. All files without the prefix "_" are parsed and then renamed with the prefix "_". This helps to prevent reprocessing the same data and is just for convenience. For each ExperimentSetup instance, a new thread is started within a ThreadPool with a fixed number of maximum threads that may be executed simultaneously as defined by the parameter numberOfCores. Each thread performs data processing steps sequentially as shown in figure 6.2 are explained in the next section.

## Data processing implementations

This section deals with the implementations of the different data processing steps including the six optimization steps. It shows which WEKA implementations are available and if missing or insufficient, how those missing functions are implemented.

[4]`https://github.com/google/gson`, Accessed 3.11.2018

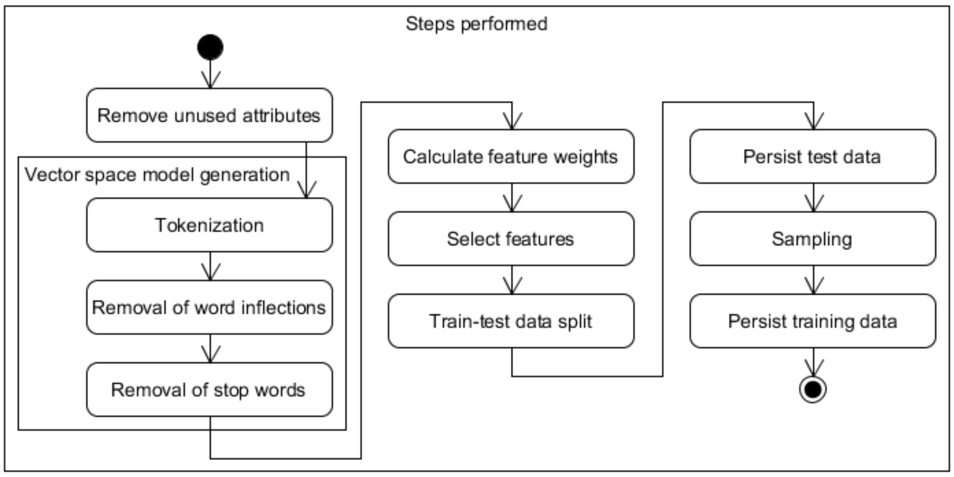


Figure 6.2: Data processing steps executed sequentially per thread using WEKA toolkit

The WEKA interface for most of the methods is a filter which can be applied on instances via the static function Filter.useFilter(Instances, Filter) which returns the processed instances as Instances. An example can be seen in listing 6.2 that shows the application of random under sampling and is discussed in the next paragraphs. Generally spoken, the execution order consists of instantiating a new filter, setting its parameters, setting the input format and then applying the filter by calling the above mentioned function.

Listing 6.2: Random under sampling using WEKA in Java

```java
    private static Instances subsample(Instances data, int
       seed) throws Exception {
        SpreadSubsample subsample = new SpreadSubsample();
        subsample.setDistributionSpread(1.0);
        subsample.setRandomSeed(seed);
        subsample.setInputFormat(data);

        return Filter.useFilter(data, subsample);
      }
```

**Removing attributes** The WEKA toolkit provides a filter called Remove. This implementation expects an array of indizes of the attributes which are to be removed. Concretely, we remove the attributes fake_id, tool_id and hierarchical_level since we do not expect them to provide valuable information.

**Vector space model generation** WEKA offers the StringToWordVector class to split a string attribute into a vector space model. This class offers extensive functionality with 17 parameters. It applies tokenization and for each token removal of word inflection and stop words (in this order). It also allows to use binary, TF and TF-IDF as feature weight and can save the dictionary. In the following paragraphs (optimization step 2, 3 and 5), each step is explained in more detail.

**Optimization step 1: Sampling** RUS can be performed by using the SpreadSubsample class. To keep the class distribution, the distribution spread needs to be set to 1 as shown in listing 6.2. SMOTE is also implemented for WEKA but as additional package that is available via the package manager, or when using Java code, via Maven[5]. We set the parameter $percentage = \frac{numberOfNegativeInstances}{numberOfPositiveInstances} - 1$ to gain a 1:1 ration of positive and negative classes. The parameter nearest neighbors is varied.

**Optimization step 2: Removal of word inflection** As stated earlier, we use the Porter stemmer. WEKA makes it possible to use this stemmer when having the snowball package in the classpath by using the weka.core.stemmers.SnowballStemmer class [Jav18]. However, this approach turned out to be unstable. Sometimes the class was not found and therefore the default NullStemmer was used. A PorterStemmer class was created that implements the weka.core.stemmers.Stemmer interface and calls the porterStemmer[6] class directly.
Lemmatization is not supported in WEKA which only offers a Stemmer interface with one method that accepts a single word. Lemmatizers would favorably need the whole sentence. We compromise and lemmatize on word level. As library, we use StanfordNLP which is available on Maven[7].
Since the tokenizer can also produce tokens that contain more than one word, the porterStemmer is limited since it only looks at the last word to stem. We overcome this limitation by the implementation of a MultiWordStemmer which acts as a proxy. This class splits each token at every white space, calls for each single word the underlying stemmer and concatenates the stemmed words by a white space.
We therefore set the MutliWordStemmer with the given implementation of the stemmer or lemmatizer as stemmer in the StringToWordVector.

**Optimization step 3: Tokenization** WEKA provides a word as well as a n-gram tokenizer. Since the output of the n-gram tokenizer is the same when setting $nGramMinSize = nGramMaxSize = 1$, we decided to use this class. A tokenizer also requires a parameter that sets the delimiters. Our approach is to add each unwanted character that was found in the preprocessed data to this delimiter string until the data

[5]`https://mvnrepository.com/artifact/nz.ac.waikato.cms.weka/SMOTE/1.0.2`, Accessed 3.11.2018

[6]Downloaded from `http://snowballstem.org/download.html`, Accessed 4.11.2018

[7]`https://mvnrepository.com/artifact/edu.stanford.nlp/stanford-corenlp/3.9.1`, Accessed 04.11.2018

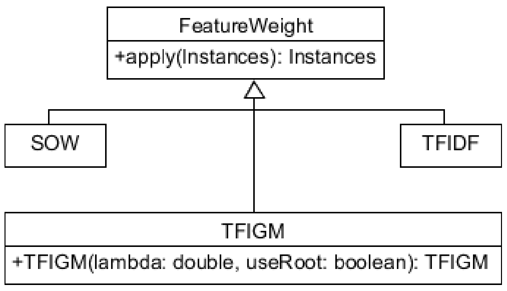

FeatureWeight
+apply(Instances): Instances
SOW
TFIDF
TFIGM
+TFIGM(lambda: double, useRoot: boolean): TFIGM

Figure 6.3: ExperimentSetup class diagram

was clean. This contains a lot of different variations of quote and bullet point characters. As described earlier in section 6.1.3, we fix parameter *nGramMinSize* to one and vary *nGramMaxSize*. In case multiple words are contained in a token, those words are concatinated by a white space. The NGramTokenizer is set in StringToWordVector as tokenizer.

**Optimization step 4: Feature weight** StringToWordVector provides implementations for 0/1, TF and TF-IDF weights. For this thesis, we are also interested in (R)TF-IGM. To extend this functionality, StringToWordVector is limited. We therefore decide to set StringToWordVector to output the TF as feature weight and manually calculate all other weights from these values and introduce a new interface FeatureWeight and its implementing classes as shown in figure 6.3. Class SOW is short for set of words (see section 3.1.5) and represents the binary feature weight. This also includes a FeatureWeightApplicator that selects the needed implementation based on the benchmark configuration. This applicator is simply called from the Java code after the vector generation.
The binary case is very simple as it sets all weights greater than one to one. TF is trivial since it only passed through the instances and this case is handled in the FeatureWeightApplicator. For TF-IDF, the DF needs to be calculated first which is a simple for loop that counts each word in all instances and then calculates the weight according to the formula presented in section 3.1.5. (R)TF-IGM is implemented very similar but instead each word is counted for each class and then calculated according to the formula from section 3.1.5.

**Optimization step 5: Removal of stop words** For the stop words removal we decided for two handlers: A regex based and a word list based handler. The first one is to get rid of symbols and numbers, where as the second one removes common words.
As for the first one, WEKA provides a RegExpFromFile implementation, which loads regular expressions from a file where each line is read as one expression. This functionality is used to remove generated features such as (roman) numerals and symbols. The used expressions are shown in listing 6.3.

As stated in section 6.1.3, we use the included stop words of the NLTK toolkit. Such a word list can be imported using the WordsFromFile class.
Those two handlers are then combined using the MultiStopwords class from WEKA. These implementations are limited in terms of n-grams because these compare a whole token and a stop word. Only if they match on token level, the token is removed. To solve this issue, we implement a MultiWordSplitterStopword that acts similar to the earlier introduced MultiWordStemmer and passes each word to the underlying MultiStopwords handler. If one of the containing words is a stop word, we return the whole token marked as stop word.

Listing 6.3: Regular expressions to remove (roman) numerals and symbols

```
^[+/<=>\§±-]+
^.*[0-9]+.*$
^(xc|xl|l?x{0,3})(ix|iv|v?i{0,3})$
^$
```

**Optimization step 6: Feature selection** StringToWordVector provides a basic TF feature selection implementation that can be set with the parameter wordsToKeep. This selection works on class basis, so setting this parameter to e.g. 1000, a maximum of 1000 words is kept for each class. Also, WEKA offers implementations for $\chi^2$ and IG. Those interfaces are a bit different since these methods calculate a metric to rank each feature. A Ranker is then used to select the best ranked features. To make both approaches comparable, we decide to implement the TF feature selection so that its selections is not class but instance based.
To implement the TF feature selection method, we create a new class that extends the weka.attributeSelection.ASEvaluation class and implements the weka.attributeSelection.AttributeEvaluator interface. This implementation, along with the $\chi^2$ and IG, is passed to the weka.attributeSelection.AttributeSelection class where the weka.attributeSelection.Ranker is set as search method and its numToSelect parameter is set to the desired count of maximum words.
Since it takes long to build the evaluator with IG and $\chi^2$, but we want to compare different values for the maximum words that would use the same evaluator, we decide to cache the evaluator. The standard Java serialization is used and the evaluator is written to a file named with the prefix "IG_", "CHI_" or "TF_" and the MD5 checksum of the instances it should select the features upon. Before building an evaluator, it is checked if a cached version exists and is then loaded.

**Writing data** WEKA offers a DataSink class, that writes all given instances to a file. Prior to that, we use the NonSparseToSparse filter to create a sparse representation of the vector space model.

### Directory Structure

All processed data is saved in ARFF files. Those are managed inside a directory structure as shown in listing 6.4. The names of each optimization step is given by concatinating the algorthim, and its parameter names and its values. An example for the baseline is shown in listing 6.5. Some parameters are not included such as the delimiters for tokenization.

Listing 6.4: Directory structure of processed files

```
/Preprocessed/sampling/removalOfWordInflections/tokenization/
    featureWeight/stopWords/featureSelection/
```

Listing 6.5: Example path for processed files of baseline

```
/Preprocessed/No/porter/n-gram_max_1_min_1/TF/
    NLTK_fileName_nltk.txt/TF_maxWords_1000/
```

Each directory contains two sub directories:

- Train: All training data in ARFF format is stored here named "subsystemName_suffix.arff". This suffix is needed because in case of random under sampling, multiple training files are created. In case of random under sampling, this suffix is an incrementing number, for SMOTE it is "SMOTE" and for no sampling, "Imbalanced".
- Test: All test data in ARFF format is stored here named "subsystemName.arff"

### 6.2.2 Classification component

This component reads the processed ARFF files and builds a model for every given training file. This model is then evaluated and the performance measures are then saved into a CSV file.

The following sections explain this component in detail. The supported arguments are described, along with the model building procedure and the calculation of the evaluation measurements. This section ends with a brief overview how the evaluation results are managed.

#### Input arguments

This component expects four parameters:

- pathToLearningFile: This file contains a directory per line, which is expected to contain a train and test sub directory as shown before (see listing 6.4) and the subsystems concatenated by a comma for which the models should be created. The last parameter is a filter function in case the given directory contain more subsystems and only a few are of interest. One example line is shown in listing 6.6 in which the base line is chosen as directory containing the processes files and the models should be created for PM and MMI_IXL.

- pathToResultCSV: This path directs to the CSV file in which the results are written to. If the file already exists, the results are appended to the given file, otherwise the file is created and the header is written.
- threadsForParallelLearning: This parameter sets the number of threads used for building models in parallel.
- threadsForClassifier: This parameter defines how many threads a classifier can use to build his model. For the classifiers used in this thesis, only RF supports multi threading which can be enabled by the numExecutionSlots parameter greater than one.

It is favorable to set the threadsForParallelLearning and threadsForClassifier parameter so that $threadsForParallelLearning * threadsForClassifier \leq numberOfCpuCores$ holds.

Listing 6.6: Example line of the input learning file. In this case for the baseline

```
.../Preprocessed/No/porter/n-gram_max_1_min_1/TF/
   NLTK_fileName_nltk.txt/TF_maxWords_1000/ PM,MMI_IXL
```

To specify the classifiers, we choose a trivial approach and create a ClassifierAssembler class, whose method getAllClassifiersToRun returns a List of weka.classifiers.AbstractClassifiers. If not all classifiers should be used then those lines are commented out.

### Model building procedure

This section describes how the many models are build. Figure 6.4 shows the main classes of this component. Some other helper classes exists but are not of interest.

The class Main parses the learning file and creates for every training file in the given directories and for every specified classifier one TestExperiment instance. Therefore the cardinality (in figure 6.4 shown as $a$) between those classes equals $numberOfClassifiers * numberOfTrainingFiles$. Main also instantiates a CsvWriterOnFinishedListener, that writes the defined evaluation measures into a given CSV file.

The TestExperiment class creates an ExperimentTrainTestThread and an ExperimentCrossValThread. As the names imply, the first one builds a model and evaluates it using the train-test approach, where as the second one uses 3-fold cross validation for evaluation. Once a thread has finished, the results are passed to the CSV writer. In case an exception is thrown, such as out of memory, the exception is also reported to the listener and question marks are written into the file instead of the results.

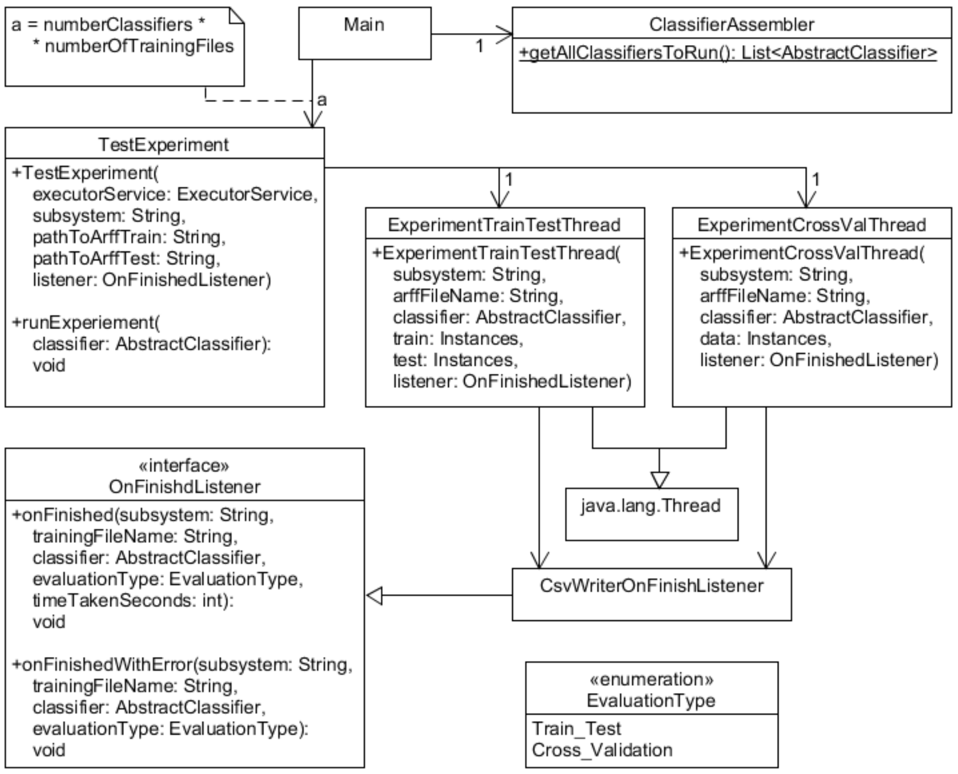

a = numberClassifiers * * numberOfTrainingFiles
Main
1
ClassifierAssembler
+getAllClassifiersToRun(): List<AbstractClassifier>
a
TestExperiment
+TestExperiment( executorService: ExecutorService, subsystem: String, pathToArffTrain: String, pathToArffTest: String, listener: OnFinishedListener)
+runExperiment( classifier: AbstractClassifier): void
1
1
ExperimentTrainTestThread
+ExperimentTrainTestThread( subsystem: String, arffFileName: String, classifier: AbstractClassifier, train: Instances, test: Instances, listener: OnFinishedListener)
ExperimentCrossValThread
+ExperimentCrossValThread( subsystem: String, arffFileName: String, classifier: AbstractClassifier, data: Instances, listener: OnFinishedListener)
«interface» OnFinishdListener
+onFinished(subsystem: String, trainingFileName: String, classifier: AbstractClassifier, evaluationType: EvaluationType, timeTakenSeconds: int): void
+onFinishedWithError(subsystem: String, trainingFileName: String, classifier: AbstractClassifier, evaluationType: EvaluationType): void
java.lang.Thread
CsvWriterOnFinishListener
«enumeration» EvaluationType
Train_Test
Cross_Validation

Figure 6.4: Class diagram of main classes of the learning component

### Evaluation measurements

Most of the used evaluation measures are included in WEKA out of the box, only the $F_2$ measure is not supported. We decided to implement the calculation in the CsvWriterOnFinishedListener just before writing the value into the file. The used code is presented in listing 6.7.

Listing 6.7: Calculating $F_2$ measure in Java from the WEKA Evaluation class

```java
public double calculateFMeasure(Evaluation evaluation, double
    beta) {
     double tp = evaluation.confusionMatrix()[0][0];
     double fn = evaluation.confusionMatrix()[0][1];
     double fp = evaluation.confusionMatrix()[1][0];
     double upper = (Math.pow(beta, 2) + 1) * tp;
     double under = upper + Math.pow(beta, 2) * fn + fp;
     return upper / under;
}
```

The other measures can be directly retrieved from the Evaluation class. The parameter in those methods define on which class to calculate those measures. Zero indicates to calculated them on the positive class. We use the following methods:

- AUROC: `evaluation.areaUnderROC(0);`
- Recall: `evaluation.recall(0);`
- Specificity: `evaluation.trueNegativeRate(0);`
- Precision: `evaluation.precision(0);`

**Result management**

The overall results are managed with a spreadsheet. If new results are stored in a CSV file, these values are copied manually into the table. When RUS is used, the results need to be macro averaged. This is done using a pivot table in an this spreadsheet. For each subsystem, one sheet is used which contains one table for each classification algorithm.

## 6.3 Phase 2 - Baseline creation

This section reports initial benchmark results. Those origin from evaluating the selected classifiers on the selected processed data sets according to the configuration defined as baseline in phase 1 (see 6.1). The data is leaved imbalanced, each word is used as token (uni-gram) and is stemmed using the Porter stemmer. As feature weight, the term frequency (TF) is used. For reducing the dimension, stop words are removed and the top 1000 features are selected based on their overall occurrence (using TF).

The results are shown in table 6.1, where the two best results according to $F_2$ of each classification algorithm are marked bold. In general, these results reveal that the classifiers are trained to classify negative instances better than positives (recall is smaller than specificity). This is because of the use of imbalanced training data. For this thesis' use-case, recall is of importance. Therefore these results are optimized upon $F_2$, which aggregates recall and precision giving recall twice the importance, in phase 3 (see section 6.4).

The ranking accroding to $F_2$ score is the same for PM, MMI_IXL and IXL_HW: kNN classifier with $k = 2$ results in highest $F_2$ score. The second best classification algorithm is Random Forest. Comparing these two algorithms, it becomes apparent that using kNN leads to higher recall, but having lower specificity compared to Random Forest.

The results on Bid_Manager show that Naïve Bayes leads to highest $F_2$ score and kNN algorithm with $k = 2$ results in second highest $F_2$. Naïve Bayes has a higher recall than kNN, but also lower specificity.

Table 6.1: Results using the benchmark configuration from the baseline

| | | | Train-Test | | |
|---|---|---|---|---|---|
| Classifier | $F_2$ | AUROC | recall | specificity | precision |
| | | | PM | | |
| Naïve Bayes | 0.6012 | 0.8237 | 0.6108 | 0.858 | 0.5659 |
| kNN (k=1) | 0.7475 | 0.8762 | 0.7581 | 0.9051 | 0.7078 |
| **kNN (k=2)** | **0.7532** | **0.8882** | **0.7896** | **0.863** | **0.6358** |
| kNN (k=3) | 0.7287 | 0.895 | 0.7266 | 0.9215 | 0.7371 |
| kNN (k=4) | 0.7249 | 0.8978 | 0.7327 | 0.9027 | 0.6953 |
| kNN (k=5) | 0.7029 | 0.8963 | 0.6982 | 0.9187 | 0.7224 |
| kNN (k=6) | 0.7018 | 0.8958 | 0.7022 | 0.9088 | 0.7001 |
| kNN (k=7) | 0.6739 | 0.895 | 0.6636 | 0.9212 | 0.7184 |
| SVM | 0.6638 | 0.7975 | 0.6301 | 0.9649 | 0.8447 |
| **Random Forest** | **0.7397** | **0.9383** | **0.7104** | **0.9723** | **0.8859** |
| | | | MMI_IXL | | |
| Naïve Bayes | 0.474 | 0.829 | 0.5266 | 0.8828 | 0.3388 |
| kNN (k=1) | 0.5958 | 0.8324 | 0.5958 | 0.9539 | 0.5958 |
| **kNN (k=2)** | **0.6231** | **0.8645** | **0.6536** | **0.9326** | **0.525** |
| kNN (k=3) | 0.5038 | 0.8697 | 0.485 | 0.9626 | 0.5966 |
| kNN (k=4) | 0.5118 | 0.8734 | 0.5012 | 0.955 | 0.5593 |
| kNN (k=5) | 0.4175 | 0.876 | 0.3857 | 0.9734 | 0.6231 |
| kNN (k=6) | 0.4474 | 0.8773 | 0.418 | 0.971 | 0.622 |
| kNN (k=7) | 0.3762 | 0.8756 | 0.3395 | 0.9802 | 0.6622 |
| SVM | 0.4021 | 0.674 | 0.3557 | 0.9924 | 0.8415 |
| **Random Forest** | **0.5673** | **0.938** | **0.5312** | **0.9829** | **0.7797** |
| | | | IXL_HW | | |
| Naïve Bayes | 0.5262 | 0.831 | 0.5874 | 0.9058 | 0.3713 |
| kNN (k=1) | 0.5673 | 0.8115 | 0.5574 | 0.9664 | 0.6108 |
| **kNN (k=2)** | **0.5838** | **0.8325** | **0.6093** | **0.9423** | **0.5** |
| kNN (k=3) | 0.5046 | 0.8468 | 0.4781 | 0.9754 | 0.6481 |
| kNN (k=4) | 0.5212 | 0.8484 | 0.5027 | 0.9697 | 0.6113 |
| kNN (k=5) | 0.4287 | 0.8521 | 0.3907 | 0.9842 | 0.701 |
| kNN (k=6) | 0.4411 | 0.8504 | 0.4071 | 0.9803 | 0.6622 |
| kNN (k=7) | 0.3475 | 0.8486 | 0.3087 | 0.9873 | 0.6975 |
| SVM | 0.41 | 0.6785 | 0.3634 | 0.9935 | 0.8418 |
| **Random Forest** | **0.568** | **0.925** | **0.5273** | **0.989**1 | **0.8213** |
| | | | Bid_Manager | | |
| **Naïve Bayes** | **0.2962** | **0.7655** | **0.4278** | **0.8758** | **0.1328** |
| kNN (k=1) | 0.2214 | 0.6667 | 0.2111 | 0.9753 | 0.2754 |
| **kNN (k=2)** | **0.2263** | **0.685** | 0.**2333** | **0.959** | **0.2019** |
| kNN (k=3) | 0.0972 | 0.6956 | 0.0833 | 0.9909 | 0.2885 |
| kNN (k=4) | 0.1255 | 0.7142 | 0.1111 | 0.9859 | 0.2597 |
| kNN (k=5) | 0.0853 | 0.7075 | 0.0722 | 0.9928 | 0.3095 |
| kNN (k=6) | 0.0854 | 0.7128 | 0.0722 | 0.9931 | 0.3171 |
| kNN (k=7) | 0.0474 | 0.7169 | 0.0389 | 0.9973 | 0.3889 |
| SVM | 0.0069 | 0.5028 | 0.0056 | 1 | 1 |
| Random Forest | 0.0603 | 0.8549 | 0.05 | 0.9958 | 0.3462 |

When looking at AUROC reveals that Random Forest has highest score on all four subsystems, indicating that this model provides the best ability of a random requirement to be classified correctly.

## 6.4 Phase 3 - Iterative optimization

This section optimizes the models' configurations for each subsystem and each classification algorithm. As baseline, the results from phase 2 are used. For optimization, a gradient descent like approach is utilized (as described in detail in section 4.3). Each dimension is sequentially optimized by selecting the method, amongst all selected, of this optimization step that leads to the best performance for the classifier. As evaluation metric, the beforehand selected $F_2$ score is mainly used. The evaluation measurements are compared on the cross-validation results with the exception of sampling, where the train-test results are compared.

To provide easier readability, some formats are introduced. In a table presenting results, the first row represents the method selected in the baseline. The selected method for each optimization step is marked bold. If a method has a parameter, we specify the parameter in brackets after the method for easier readability. In case absolute and relative scores are presented in a table (for instance to show the improvements), the absolute measurements are marked with a gray background. Also in the tables comparing removing stop words versus keeping them, the abbreviation SW is short for stop words.

### 6.4.1 Subsystem PM

This section covers the results of each selected classifier for every method of each of the six optimization steps. Its output is one configuration of a model per classification algorithm that lead to highest $F_2$ score. As mentioned earlier, a classifier is chosen based on the results from this subsystem for the further benchmarking of the other three subsystems. This is because of the extensive run time.

#### Random Forest

The results for the Random Forest (RF) classifier are presented in the following paragraphs. Each optimization step is discussed individually.

**Optimization step 1: Sampling** The results are shown in table 6.2. As described earlier, the optimization step sampling is the only one where we compare train-test results.
RUS strengthens the ability of classifying positive instances, which is recognizable in recall. This increases from 0.7104 (imbalanced) to 0.8631, which is also higher than SMOTE's recall of around 0.74 to 0.75. This also improves $F_2$ score to 0.8183, but comes with a lower precision of 0.6778 compared to 0.8859 for using an imbalanced training set or to SMOTE (between 0.8243 and 0.8324). The specificity identifies that the classifier's

Table 6.2: Comparison of sampling methods (for PM using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Imbalanced | 0.7397 | 0.9383 | 0.7104 | 0.9723 | 0.8859 |
| **RUS** | **0.8183** | **0.942** | **0.8631** | **0.8757** | **0.6778** |
| SMOTE (2) | 0.7584 | 0.9379 | 0.7419 | 0.9547 | 0.8324 |
| SMOTE (3) | 0.7637 | 0.9392 | 0.749 | 0.9532 | 0.829 |
| SMOTE (4) | 0.7587 | 0.9383 | 0.7439 | 0.952 | 0.8243 |
| SMOTE (5) | 0.7597 | 0.9378 | 0.7439 | 0.9538 | 0.8299 |
| SMOTE (6) | 0.7576 | 0.9378 | 0.7419 | 0.9532 | 0.8277 |

Table 6.3: Comparison of word inflection removal approaches (for PM using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Stemmer | 0.8428 | 0.9328 | 0.8393 | 0.8604 | 0.8574 |
| **Lemmatizer** | **0.8456** | **0.934** | **0.8427** | **0.8601** | **0.8577** |

Table 6.4: Comparison of n-grams (for PM using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| $n \in \{1\}$ | 0.8456 | 0.934 | 0.8427 | 0.8601 | 0.8577 |
| $n \in \{1, 2\}$ | 0.8464 | 0.9333 | 0.8436 | 0.8598 | 0.8575 |
| $\boldsymbol{n \in \{1, 2, 3\}}$ | **0.8482** | **0.9341** | **0.8462** | **0.858** | **0.8563** |
| $n \in \{1, 2, 3, 4\}$ | 0.8466 | 0.934 | 0.8439 | 0.86 | 0.8577 |
| $n \in \{1, 2, 3, 4, 5\}$ | 0.846 | 0.9335 | 0.8436 | 0.8577 | 0.8557 |

performance on classifying negative instances decreases. It is highest using no sampling (0.9723) and second high, when using SMOTE (from 0.952 to 0.9547).
This shows, that RUS leads to superior results for this use-case compared to the other sampling strategies. Therefore, RUS is selected as the sampling strategy in all further steps.

**Optimization step 2: Removal of word inflection** Table 6.3 shows the comparison between a stemmer and a lemmatizer. Using lemmatization slightly increases the CV $F_2$ score by 0.0028 and is therefore selected.

**Optimization step 3: n-grams** Table 6.4 shows results for different variations of $n$ for using n-grams. As the table shows, the best $F_2$ result is achieved when using $n \in \{1, 2, 3\}$. By comparing the selected features of uni-grams and 3-grams, some new 2-grams appear quite frequently. Some of them are: "power supply", "gsm r", "level cross", "contractor shall", "railway station", "railway line" and "cable network".

Table 6.5: Comparison of feature weights (for PM using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **TF** | **0.8482** | **0.9341** | **0.8462** | **0.858** | **0.8563** |
| 0/1 | 0.847 | 0.9341 | 0.8445 | 0.8591 | 0.8571 |
| TF-IDF | 0.8479 | 0.934 | 0.8461 | 0.8565 | 0.855 |
| RTF-IGM (5) | 0.8477 | 0.9339 | 0.8458 | 0.8568 | 0.8552 |
| RTF-IGM (6) | 0.8474 | 0.934 | 0.8454 | 0.857 | 0.8553 |
| RTF-IGM (7) | 0.8478 | 0.9341 | 0.8459 | 0.8569 | 0.8553 |
| RTF-IGM (8) | 0.8474 | 0.9337 | 0.8458 | 0.855 | 0.8537 |
| RTF-IGM (9) | 0.8478 | 0.9341 | 0.8459 | 0.8571 | 0.8555 |
| TF-IGM (5) | 0.8477 | 0.9337 | 0.8461 | 0.8557 | 0.8543 |
| TF-IGM (6) | 0.8479 | 0.9338 | 0.846 | 0.8569 | 0.8554 |
| TF-IGM (7) | 0.8474 | 0.9339 | 0.8457 | 0.8558 | 0.8543 |
| TF-IGM (8) | 0.8479 | 0.9338 | 0.8461 | 0.8568 | 0.8552 |
| TF-IGM (9) | 0.8472 | 0.9338 | 0.8452 | 0.8568 | 0.8551 |

Table 6.6: Removing stop words or keeping them (for PM using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **SW removal: Yes** | **0.8482** | **0.9341** | **0.8462** | **0.858** | **0.8563** |
| SW removal: No | 0.829 | 0.9293 | 0.8197 | 0.8757 | 0.8683 |

**Optimization step 4: Feature weight** As seen in table 6.5, TF works best in this case because it has the highest $F_2$ score. TF-IGM ($\lambda = 6$) closely follows with a distance of 0.000321 and TF-IDF with a $F_2$ difference to TF of 0.000307. Because of these small differences, we think that the feature weight does not make a lot of difference for this classifier and subsystem, but selected TF as the best working method.

**Optimization step 5: Removal of stop words** When not removing stop words, the $F_2$ decreases by 0.0192 as presented in table 6.6. Therefore, the use of stop words is continued.

**Optimization step 6: Feature selection** Since in this step we compare three different methods with fifteen parameters each, we present the results in figure 6.5. IG has a small peak when the maximum count of features is set to 1300. This selection methods selects more 2-grams and 3-grams, as well as n-grams that are also part of a higher n-gram. For instance, "controller terminal" is ranked with a score of 539.0916 followed by "controller terminal system" with a score of 533.8334. Besides the attribute req_type and textIsHeading, the now new top five best ranked features are "installation", "cable", "work", "equipment" and "cable network" instead of "system", "shall", "must",

Figure 6.5: Comparison of feature selection methods (for PM using RF)

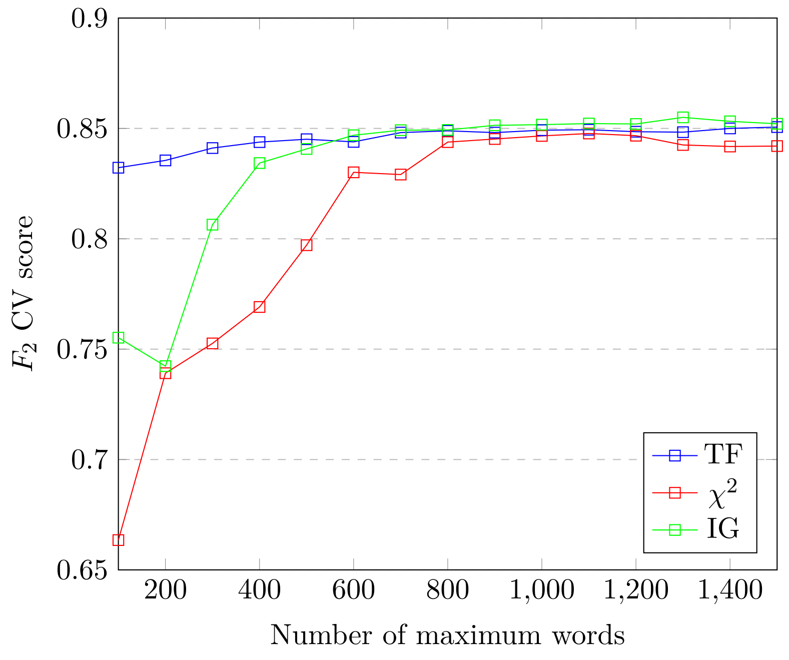


Table 6.7: Improvements of each optimization step (for PM using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.7397 | 0.9383 | 0.7104 | 0.9723 | 0.8859 |
| RUS | +0.0786 | +0.0038 | +0.1527 | -0.0966 | -0.2081 |
| Lemmatizer | +0.0007 | +0.0008 | +0.0018 | -0.0014 | -0.002 |
| $n \in \{1, 2, 3\}$ | -0.0013 | -0.0011 | -0.0008 | -0.0014 | -0.0025 |
| TF | 0 | 0 | 0 | 0 | 0 |
| SW removal: Yes | 0 | 0 | 0 | 0 | 0 |
| IG (1300) | +0.0008 | +0.0007 | +0.0106 | -0.015 | -0.0222 |
| Total Increase | +0.0788 | +0.0041 | +0.1643 | -0.1144 | -0.2348 |
| Total | 0.8185 | 0.9424 | 0.8747 | 0.8579 | 0.6511 |

"installation" and "station" when using TF.

**Final performance** When using the features selection IG with the maximum of 1300 words, a $F_2$ score of 0.855 is achieved using cross validation and 0.8185 using train-test evaluation. The final results including all improvements is presented in table 6.7. Overall, we reached an improvement of +0.1328 for CV and +0.0788 on test set evaluation. The most influence on performance has sampling, which is responsible for over 90% the improvement for both evaluation techniques. The other steps show less improvement.

Table 6.8: Improvements of each optimization step (for PM using NB)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.6006 | 0.8158 | 0.6109 | 0.8563 | 0.5628 |
| SMOTE (3) | +0.1409 | -0.0697 | +0.138 | -0.1568 | +0.1508 |
| Lemmatizer | +0.0133 | +0.0044 | +0.017 | -0.0065 | +0.0002 |
| $n \in \{1, 2, 3, 4\}$ | +0.0004 | -0.0002 | +0.0013 | -0.0047 | -0.0028 |
| 0/1 | +0.025 | -0.0358 | +0.0473 | -0.0928 | -0.0429 |
| SW removal: Yes | 0 | 0 | 0 | 0 | 0 |
| TF (100) | +0.0135 | +0.0897 | +0.0171 | -0.0019 | +0.0036 |
| Total Increase | +0.1932 | -0.0117 | +0.2208 | -0.2627 | +0.1089 |
| Total | 0.7939 | 0.8042 | 0.8317 | 0.5936 | 0.6717 |
| | Train-Test | | | | |
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.6012 | 0.8237 | 0.6108 | 0.858 | 0.5659 |
| SMOTE (3) | +0.07 | -0.0809 | +0.189 | -0.2088 | -0.1573 |
| Lemmatizer | -0.008 | +0.0041 | -0.0213 | +0.0206 | +0.0081 |
| $n \in \{1, 2, 3, 4\}$ | -0.0058 | +0.0057 | -0.0183 | +0.0209 | +0.0102 |
| 0/1 | +0.0217 | -0.0198 | +0.123 | -0.1811 | -0.0738 |
| SW removal: Yes | 0 | 0 | 0 | 0 | 0 |
| TF (100) | -0.0239 | +0.065 | -0.0915 | +0.1118 | +0.0348 |
| Total Increase | +0.054 | -0.0259 | +0.1809 | -0.2365 | -0.1780 |
| Total | 0.6553 | 0.7978 | 0.7917 | 0.6215 | 0.3879 |

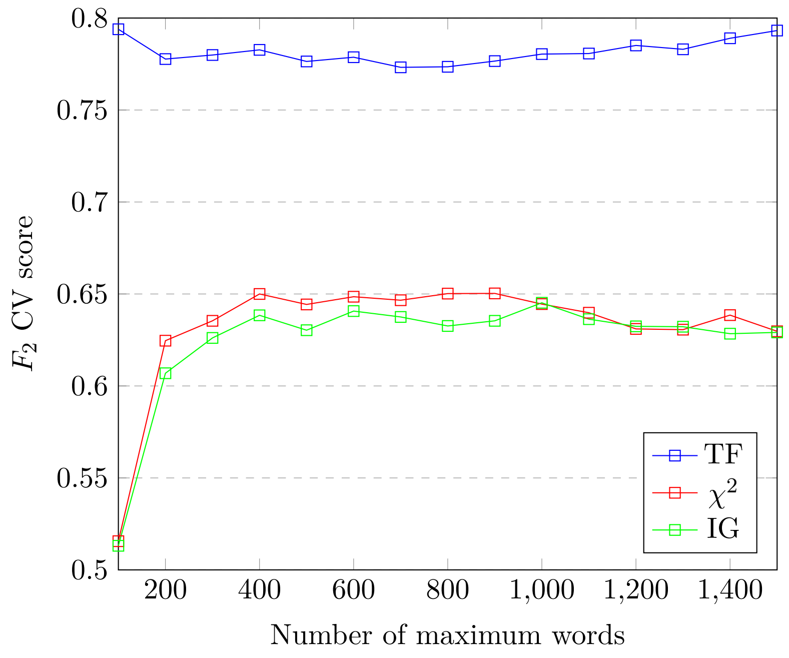
### Naïve Bayes

For NB we decide to report the results of the methods that lead to the best performance as well as the performance increase of each selected methods in the optimization steps in table 6.8.

**Improvements** SMOTE has the biggest impact on performance especially on CV. This is no wonder since the synthetically created instances are also used for evaluation since they are in the training set. But also for train-test evaluation, a improvement of 0.07 is achieved. Using a lemmatizer improves $F_2$ CV scoreby +0.0133, but decreases the performance on the test set. As stated in section 3.1.7, the Bernulli NB classifier expects 0/1 as feature weight, but we decided to start with TF, as for all other classifiers. The $F_2$ score for 0/1 on cross validation is 0.7804 where as the other weights are between 0.7533 (TF-IGM, $\lambda = 7$) and 0.7645 (RTF-IGM, $\lambda = 9$).

The feature selection comparison is shown in figure 6.6. Interestingly the best feature selection method is TF with max words set to 100. When increasing max words, the score could get higher, but since the classifier's performance is much lower than the previously evaluated RF, we decided to stop at this point.

Figure 6.6: Comparison of feature selection methods (for PM using NB)



**Final performance** By optimizing the benchmark configuration, we increase the $F_2$ score by 0.1932 on cross validation and 0.054 when evaluated on test data. The feature selection comparison is shown in figure 6.6. Interestingly the best feature selection method is TF with max words set to 100. When increasing max words, the score could get higher, but since the classifier's performance is much lower than the previously evaluated RF, we decided to stop at this point.

### SVM

The results for the Support Vector Machine (SVM) classification algorithm are presented in the following paragraphs. Each optimization step is discussed individually.

**Optimization step 1: Sampling** The results for different sampling techniques is presented in table 6.9. The $F_2$ Score is highest for SMOTE ($nearestNeighbors = 3$) which is therefore selected as sampling method. By sorting the $F_2$ scores descending, the following order arises: $SMOTE(3) > SMOTE(2) > SMOTE(5) > SMOTE(4) > SMOTE(6) > RUS > imbalanced$. Except for leaving the training data imbalanced, all scores ($F_2$, AUROC, recall, specificity and precision) of the benchmarked methods vary by less than 0.0082 (calculated by subtracting the minimum from the maximum of each score).

Table 6.9: Comparison of sampling methods (for PM using SVM)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.6638 | 0.7975 | 0.6301 | 0.9649 | 0.8447 |
| RUS | 0.7864 | 0.836 | 0.8662 | 0.8058 | 0.5748 |
| SMOTE (2) | 0.7923 | 0.8408 | 0.8709 | 0.8106 | 0.5822 |
| **SMOTE (3)** | **0.7924** | **0.8405** | **0.872** | **0.8091** | **0.5805** |
| SMOTE (4) | 0.7895 | 0.839 | 0.8669 | 0.8112 | 0.5819 |
| SMOTE (5) | 0.7906 | 0.8399 | 0.8679 | 0.8118 | 0.5829 |
| SMOTE (6) | 0.7895 | 0.8384 | 0.8689 | 0.8078 | 0.5781 |

Table 6.10: Comparison of word inflection removal methods (for PM using SVM)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **Stemmer** | **0.8918** | **0.8601** | **0.9095** | **0.8106** | **0.8277** |
| Lemmatizer | 0.8895 | 0.8608 | 0.9054 | 0.8161 | 0.8312 |

Table 6.11: Comparison of n-grams (for PM using SVM)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| $\boldsymbol{n \in \{1\}}$ | **0.8918** | **0.8601** | **0.9095** | **0.8106** | **0.8277** |
| $n \in \{1, 2\}$ | 0.8886 | 0.8598 | 0.9045 | 0.8152 | 0.8303 |
| $n \in \{1, 2, 3\}$ | 0.8888 | 0.8601 | 0.9047 | 0.8154 | 0.8305 |
| $n \in \{1, 2, 3, 4\}$ | 0.8881 | 0.8596 | 0.9038 | 0.8153 | 0.8303 |
| $n \in \{1, 2, 3, 4, 5\}$ | 0.8873 | 0.8588 | 0.9031 | 0.8145 | 0.8296 |

**Optimization step 2: Removal of word inflection** Using the lemmatizer showed a decrease of 0.0023 of the $F_2$ score. Therefore, the stemmer is selected.

**Optimization step 3: n-grams** The results are presented in table 6.11. The use of multiple words as tokens decreases the $F_2$ metric. We therefore continue using uni-grams as tokens.

**Optimization step 4: Feature weight** The results can be seen in in table 6.12. The classifier achieved highest $F_2$ score of 0.8923 when using TF-IGM ($\lambda = 8$). TF-IGM in general outperformed the other methods regarding the $F_2$ metric, followed by TF, TF-IDF and 0/1. Compared to the baseline (TF), using TF-IGM ($\lambda = 8$) increases $F_2$ score by 0,0005, AUROC by 0.0004, recall by 0.0005, specificity by 0.0003 and precision by 0.0003. Because of these improvements, we selected TF-IGM with $\lambda = 8$.

Table 6.12: Comparison of feature weights (for PM using SVM)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| TF | 0.8918 | 0.8601 | 0.9095 | 0.8106 | 0.8277 |
| 0/1 | 0.8895 | 0.8647 | 0.9033 | 0.826 | 0.8385 |
| TF-IDF | 0.8911 | 0.8593 | 0.9087 | 0.8098 | 0.8269 |
| RTF-IGM (5) | 0.8904 | 0.8649 | 0.9044 | 0.8254 | 0.8382 |
| RTF-IGM (6) | 0.8898 | 0.8644 | 0.9038 | 0.8251 | 0.8379 |
| RTF-IGM (7) | 0.8893 | 0.8642 | 0.9032 | 0.8252 | 0.8378 |
| RTF-IGM (8) | 0.8894 | 0.8644 | 0.9032 | 0.8255 | 0.8381 |
| RTF-IGM (9) | 0.8894 | 0.8645 | 0.9032 | 0.8259 | 0.8384 |
| TF-IGM (5) | 0.8922 | 0.8607 | 0.9096 | 0.8118 | 0.8286 |
| TF-IGM (6) | 0.8913 | 0.8602 | 0.9084 | 0.812 | 0.8285 |
| TF-IGM (7) | 0.8915 | 0.8599 | 0.909 | 0.8107 | 0.8276 |
| **TF-IGM (8)** | **0.8923** | **0.8605** | **0.91** | **0.811** | **0.828** |
| TF-IGM (9) | 0.8922 | 0.8603 | 0.91 | 0.8106 | 0.8277 |

Table 6.13: Removing stop words or keeping them (for PM using SVM)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **SW removal: Yes** | **0.8923** | **0.8605** | **0.91** | **0.811** | **0.828** |
| SW removal: No | 0.8896 | 0.8656 | 0.9028 | 0.8285 | 0.8403 |

**Optimization step 5: Removal of stop words** Keeping stop words results in a lower $F_2$ score as can be seen in table 6.13. Therefore we continue the remove stop words.

**Optimization step 6: Feature selection** The results are shown in figure 6.7. Compared within the fixed range of max words from 100 to 1500, $\chi^2$ with max words equals 1300 provides the best $F_2$ score of 0.8998 and is selected. TF with max words set to 1000 selects "system", "shall", "instal", "signal" and "must", where as this feature selection method selects "instal", "cabl", "work", "equip" and "gsm" as top ranked features besides req_type and textIsHeading. IG with max words set to 1300 selects the same top 10 features as $\chi^2$.

**Final performance** The final train-test results are presented in table 6.14. In total, we increased $F_2$ by 0.1596, AUROC by 0.0683 and recall by 0.2642 but also decreased specificity by 0.1275 and precision by 0.2197. This improves the classifiers ability to detect positive instances. The use of SMOTE ($nearestNeighbors = 3$) has the biggest impact on all scores.

Figure 6.7: Comparison of feature selection methods (for PM using SVM)

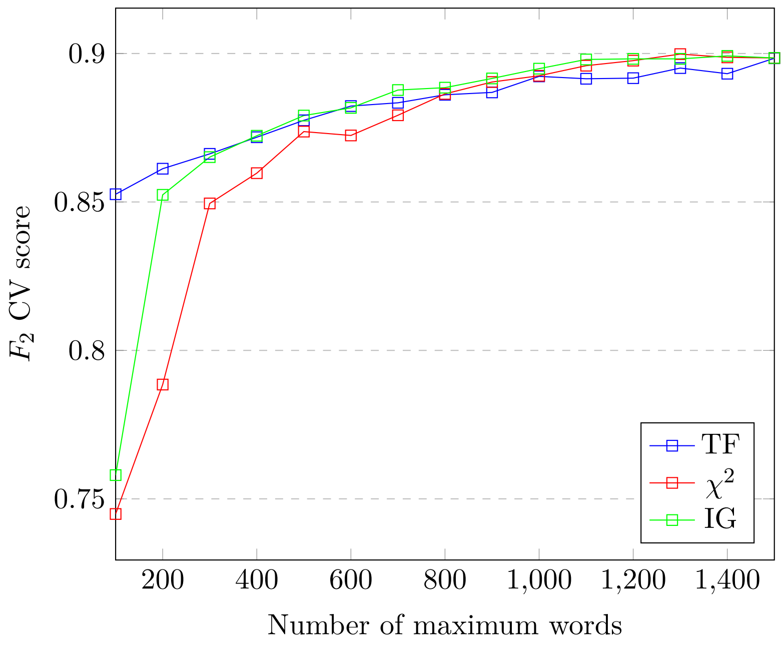


Table 6.14: Improvements of each optimization step (for PM using SVM)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.6638 | 0.7975 | 0.6301 | 0.9649 | 0.8447 |
| SMOTE (3) | +0.1286 | +0.043 | +0.2419 | -0.1558 | -0.2642 |
| Stemmer | 0 | 0 | 0 | 0 | 0 |
| $n \in \{1\}$ | 0 | 0 | 0 | 0 | 0 |
| TF-IGM (8) | +0.001 | +0.0011 | 0 | +0.0021 | +0.0028 |
| SW removal: Yes | 0 | 0 | 0 | 0 | 0 |
| $\chi^2$ (1300) | +0.03 | +0.0242 | +0.0223 | +0.0262 | +0.0417 |
| Total increase | +0.1596 | +0.0683 | +0.2642 | -0.1275 | -0.2197 |
| Total | 0.8234 | 0.8658 | 0.8943 | 0.8374 | 0.625 |

Figure 6.8: Comparison of different values for $k$ using kNN classifier on PM

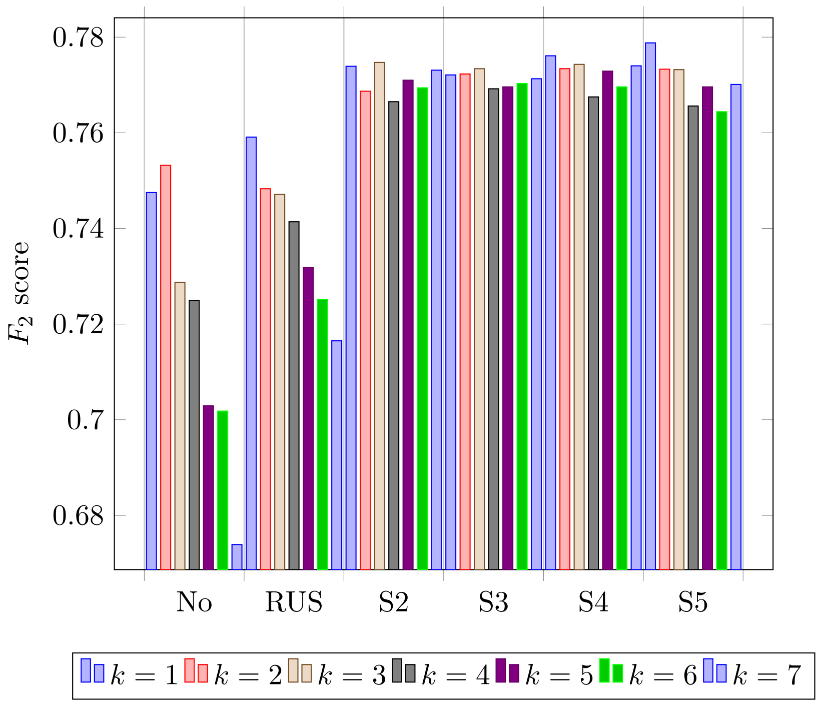


Table 6.15: Comparison of sampling methods (for PM using kNN)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Imbalanced | 0.7475 | 0.8762 | 0.7581 | 0.9051 | 0.7078 |
| RUS | 0.7591 | 0.8725 | 0.8523 | 0.7691 | 0.5281 |
| SMOTE (2) | 0.7739 | 0.8817 | 0.8222 | 0.8516 | 0.6266 |
| SMOTE (3) | 0.7721 | 0.8811 | 0.8222 | 0.8479 | 0.6209 |
| SMOTE (4) | 0.7761 | 0.8828 | 0.8293 | 0.8445 | 0.6177 |
| **SMOTE (5)** | **0.7788** | **0.8842** | **0.8323** | **0.8451** | **0.6195** |
| SMOTE (6) | 0.7787 | 0.8851 | 0.8333 | 0.8432 | 0.617 |

### k-Nearest Neighbors

The results for the k-Nearest Neighbors (kNN) classifier are presented in the following paragraphs. Each optimization step is discussed individually.

**Selecting *k*** To select $k$ for the kNN classifier, the sampling optimization step is executed for $k = \{1, 2, 3, 4, 5, 6, 7\}$. The results for the train-test evaluation is presented in figure 6.8. In this diagram, "No" is short for using no sampling strategy and "S" denotes SMOTE with its parameter set to the number followed. The highest $F_2$ score on the test set is achieved when using $k = 1$ and SMOTE with $nearestNeighbors = 5$. We decided

Table 6.16: Comparison of word inflection removal methods (for PM using kNN)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **Stemmer** | **0.9294** | **0.9348** | **0.9518** | **0.8311** | **0.8493** |
| Lemmatizer | 0.9273 | 0.9338 | 0.9492 | 0.8311 | 0.849 |

Table 6.17: Comparison of n-grams (for PM using kNN)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| $\boldsymbol{n \in \{1\}}$ | **0.9294** | **0.9348** | **0.9518** | **0.8311** | **0.8493** |
| $n \in \{1, 2\}$ | 0.9261 | 0.9345 | 0.9475 | 0.8318 | 0.8493 |
| $n \in \{1, 2, 3\}$ | 0.9275 | 0.9358 | 0.9489 | 0.8333 | 0.8506 |
| $n \in \{1, 2, 3, 4\}$ | 0.9264 | 0.9352 | 0.9474 | 0.8341 | 0.851 |
| $n \in \{1, 2, 3, 4, 5\}$ | 0.9271 | 0.9352 | 0.9485 | 0.833 | 0.8503 |

to use $k = 1$ as our choice, because we stick to the methodology of selecting the best method in the sampling optimization step on the train-test evaluation. An alternative choice could would be $k = 3$ and SMOTE with $nearestNeighbors = 2$. The $F_2$ score difference between those variants is 0.0133.

**Optimization step 1: Sampling** The detailed results for the kNN classifier with selected $k = 1$ is shown in table 6.15. Although RUS improves recall the most, it also has the lowest specificity and precision that only results in an $F_2$ score of 0.7591. Using SMOTE $nearestNeighbors = 5$ leads to highest $F_2$ score and is therefore selected.

**Optimization step 2: Removal of word inflection** As table 6.16 shows, using a lemmatizer reduces the $F_2$ score by 0.0021. Therefore, the stemming method is selected.

**Optimization step 3: n-grams** Table 6.17 presents the results for using different n-grams. Using $n > 1$ decreases the $F_2$ score. Therefore we select uni-grams as best tokens.

**Optimization step 4: Feature weight** The comparison of different feature weights is shown in table 6.18. RTF-IGM seem to have the same $F_2$ score regardless of $\lambda$, but this is due to rounding. $\lambda = 5$ cause the highest $F_2$ score of 0.933632, followed by $\lambda = 6$ with a $F_2$ score of 0.933628 and $\lambda = 8$ of 0.933625. The results expose the following order comparing $F_2$ scores: $RTF\text{-}IGM > 0/1 > TF\text{-}IDF > TF\text{-}IGM > TF$. We selected RTF-IGM ($\lambda = 5$) as feature weight in further steps.

**Optimization step 5: Removal of stop words** Using no stop words removal increases recall by 0.0038, decreases the specificity by 0.0076 and precision by 0.0051 as

Table 6.18: Comparison of feature weights (for PM using kNN)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| TF | 0.9294 | 0.9348 | 0.9518 | 0.8311 | 0.8493 |
| 0/1 | 0.9314 | 0.9381 | 0.961 | 0.8021 | 0.8292 |
| TF-IDF | 0.9311 | 0.9357 | 0.9534 | 0.8334 | 0.8512 |
| **RTF-IGM (5)** | **0.9336** | **0.9368** | **0.9588** | **0.8241** | **0.845** |
| RTF-IGM (6) | 0.9336 | 0.9367 | 0.9587 | 0.8243 | 0.8451 |
| RTF-IGM (7) | 0.9336 | 0.9367 | 0.9587 | 0.824 | 0.8449 |
| RTF-IGM (8) | 0.9336 | 0.9367 | 0.9587 | 0.8245 | 0.8453 |
| RTF-IGM (9) | 0.9336 | 0.9369 | 0.9586 | 0.8247 | 0.8454 |
| TF-IGM (5) | 0.9299 | 0.9357 | 0.9517 | 0.8348 | 0.8521 |
| TF-IGM (6) | 0.9296 | 0.9355 | 0.9513 | 0.8348 | 0.852 |
| TF-IGM (7) | 0.9295 | 0.9353 | 0.9513 | 0.8343 | 0.8517 |
| TF-IGM (8) | 0.9297 | 0.9354 | 0.9514 | 0.8346 | 0.8519 |
| TF-IGM (9) | 0.9296 | 0.9354 | 0.9513 | 0.8345 | 0.8518 |

Table 6.19: Removing stop words or keeping them (for PM using kNN)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| SW removal: Yes | 0.9336 | 0.9368 | 0.9588 | 0.8241 | 0.845 |
| **SW removal: No** | **0.9353** | **0.9346** | **0.9626** | **0.8165** | **0.8399** |

shown in table 6.13. Because of the improvement of the recall, the $F_2$ increases by 0.0016. Therefore we decide to not remove stop words.

**Optimization step 6: Feature selection** The results for TF, $\chi^2$ and IG for different max words are presented in figure 6.9. From max words ranges from 800 to 1500, all three methods perform similar. TF has the highest $F_2$ score of 0.9373, when max words equals 1500 and is therefore selected.

**Final performance** The final train-test scores are summarized in table 6.20. Overall we improved $F_2$ score by 0.0259, AUROC by 0.005 and recall by 0.0813, but decreased specificity by 0.0831 and precision by 0.1195. In total, we achieved a $F_2$ score of 0.7734, AUROC of 0.8812, recall of 0.8394, specificity of 0.822 and precision of 0.5883. Practically this means, this classifier detects positive instances and negative instances nearly equally good (recall and specificity distance is 0.0174), but because of the imbalance in the test set, we get a low precision.

Figure 6.9: Comparison of feature selection methods (for PM using kNN)

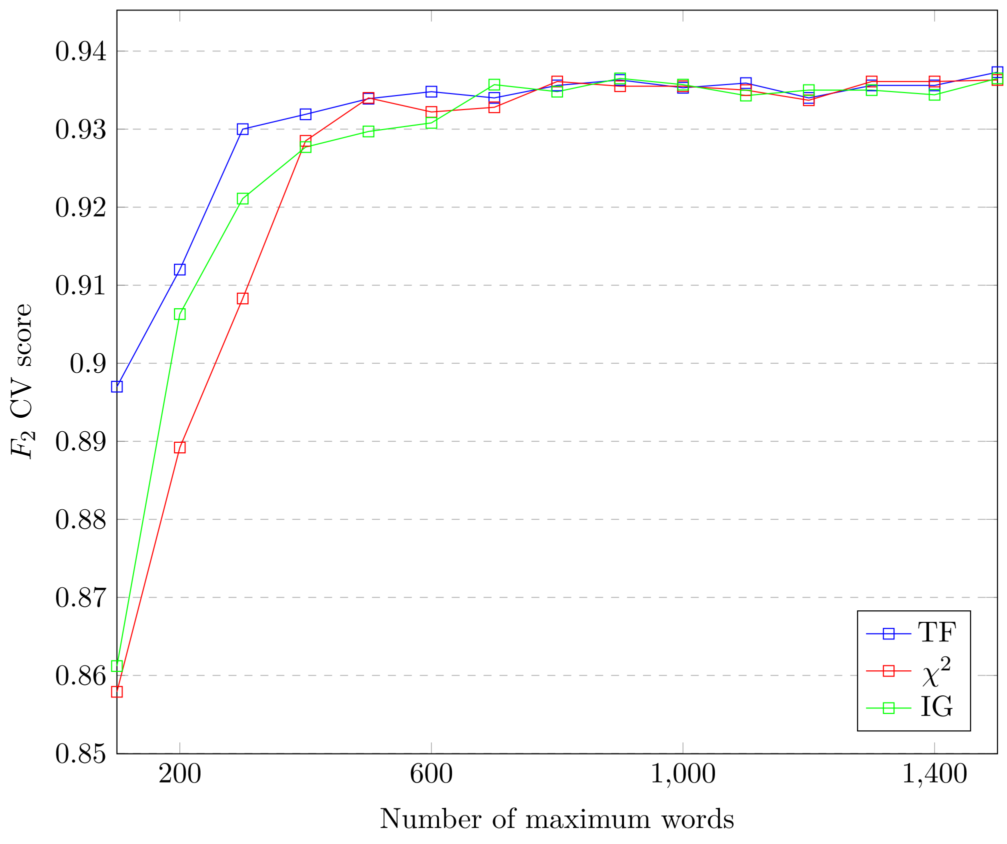


Table 6.20: Improvements of each optimization step (for PM using kNN)

| | | | Train-Test | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.7475 | 0.8762 | 0.7581 | 0.9051 | 0.7078 |
| SMOTE (5) | +0.0313 | +0.008 | +0.0742 | -0.06 | -0.0883 |
| Stemmer | 0 | 0 | 0 | 0 | 0 |
| $n \in \{1\}$ | 0 | 0 | 0 | 0 | 0 |
| RTF-IGM (5) | +0.0032 | +0.0028 | +0.0081 | -0.0068 | -0.0078 |
| SW removal: No | +0.0041 | +0.0036 | +0.0082 | -0.0046 | -0.0044 |
| TF (1500) | -0.0117 | -0.0083 | -0.0091 | -0.0095 | -0.016 |
| Total increase | +0.0269 | +0.0061 | +0.0813 | -0.081 | -0.1165 |
| Total | 0.7744 | 0.8823 | 0.8394 | 0.8241 | 0.5913 |

Table 6.21: Results for PM showing the best results for each classifier

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| NB | 0.6553 | 0.7978 | 0.7917 | 0.6215 | 0.3879 |
| kNN (k=1) | 0.7734 | 0.8812 | 0.8394 | 0.822 | 0.5883 |
| RF | 0.8185 | 0.9424 | 0.8747 | 0.8579 | 0.6511 |
| SVM | 0.8234 | 0.8658 | 0.8943 | 0.8374 | 0.625 |

### Selection of classifier

As stated in the methodology, we decided to select a classifier based on the benchmark results for PM. This is because of the extensive run time and the design of the implementation. We underestimated the manual effort of writing benchmark configuration files individual for each optimization step and classifier. But not only the manual interaction slows down the execution, the algorithms take a long time to finish. For instance, SMOTE takes 35 to 50 minutes for oversampling.

**Comparison of classifiers** Table 6.21 shows the final train-test results for the above benchmarked classifiers ordered by $F_2$ score. SVM performs best (compared by $F_2$), closely followed by RF (-0.0049). kNN's ($k = 1$) and NB's $F_2$ score distance to SVM is 0.05 and 0.1681. We decided to use RF for the other subsystems because of its second best $F_2$ performance, highest AUROC of 0.9424 and its support for multi-threading.

## 6.4.2 Subsystem MMI_IXL

This section covers the results of each selected classifier for every method of each of the six optimization steps for the data set MMI_IXL. It follows the same structure as the previous documentation on PM.

### Random forest

The results for the Random Forest (RF) classifier are presented in the following paragraphs. Each optimization step is discussed individually.

**Optimization step 1: Sampling** The results of table 6.22 reveal using imbalanced training data is superior if the goal is to detect only negative instances because of its high specificity (0.9829), but low recall of 0.5312 indicates a high number of FN. Using SMOTE increases its ability to detect positive instances by around 0.06 with a decrease of around 0.01 in specificity. RUS increases recall to 0.8999 and $F_2$ to 0.7062 but also decreases specificity to 0.8322. This leads to a decreased precision of only 0.3796. The reason is for this low precision but with specificity of 0.8322 is because of the imbalanced test data. We select RUS as best working sampling method for this use-case since recall and $F_2$ is of high importance.

Table 6.22: Comparison of sampling methods (for MMI_IXL using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Imbalanced | 0.5673 | 0.938 | 0.5312 | 0.9829 | 0.7797 |
| **RUS** | **0.7062** | **0.9377** | **0.8999** | **0.8322** | **0.3796** |
| SMOTE (2) | 0.6226 | 0.9371 | 0.5982 | 0.9766 | 0.7443 |
| SMOTE (3) | 0.612 | 0.9376 | 0.5866 | 0.9766 | 0.7405 |
| SMOTE (4) | 0.6253 | 0.9382 | 0.6028 | 0.9752 | 0.7352 |
| SMOTE (5) | 0.6274 | 0.938 | 0.6028 | 0.9771 | 0.75 |
| SMOTE (6) | 0.6256 | 0.9358 | 0.6005 | 0.9774 | 0.7514 |

Table 6.23: Comparison of word inflection removal methods (for MMI_IXL using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Stemmer | 0.8829 | 0.9304 | 0.8988 | 0.8087 | 0.8245 |
| **Lemmatizer** | **0.8832** | **0.9307** | **0.8992** | **0.8089** | **0.8248** |

Table 6.24: Comparison of n-grams (for MMI_IXL using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| $\boldsymbol{n \in \{1\}}$ | **0.8832** | **0.9307** | **0.8992** | **0.8089** | **0.8248** |
| $n \in \{1, 2\}$ | 0.8824 | 0.9305 | 0.8974 | 0.8124 | 0.8271 |
| $n \in \{1, 2, 3\}$ | 0.883 | 0.9303 | 0.8984 | 0.8116 | 0.8267 |
| $n \in \{1, 2, 3, 4\}$ | 0.8822 | 0.9306 | 0.8971 | 0.8126 | 0.8272 |
| $n \in \{1, 2, 3, 4, 5\}$ | 0.879 | 0.9295 | 0.8944 | 0.8071 | 0.8226 |

**Optimization step 2: Removal of word inflection** As table 6.23 reveals that the use of a lemmatizer increases $F_2$ score by 0.0003. Therefore we use the lemmatizer as method of choice for this step.

**Optimization step 3: n-grams** Table 6.24 shows the results for different n-grams. The use of multi words as token does not increase $F_2$ score and therefore we continue only using one word per token.

**Optimization step 4: Feature weight** The results for different feature weights are presented in table 6.25. The best $F_2$ results are achieved when using RTF-IGM ($\lambda = 8$) with score of 0.885. RTF-IGM and TF-IGM when using $\lambda = 8$ or $\lambda = 7$ are the top four weights regarding $F_2$ score. Because of this, we proceed using RTF-IGM with $\lambda = 8$.

Table 6.25: Comparison of feature weights (for MMI_IXL using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| TF | 0.8832 | 0.9307 | 0.8992 | 0.8089 | 0.8248 |
| 0/1 | 0.8839 | 0.9305 | 0.9002 | 0.8078 | 0.8241 |
| TF-IDF | 0.8831 | 0.931 | 0.8991 | 0.8086 | 0.8245 |
| RTF-IGM (5) | 0.8833 | 0.9311 | 0.8989 | 0.8109 | 0.8262 |
| RTF-IGM (6) | 0.8834 | 0.9308 | 0.8994 | 0.809 | 0.8249 |
| RTF-IGM (7) | 0.8842 | 0.9313 | 0.9 | 0.8109 | 0.8264 |
| **RTF-IGM (8)** | **0.885** | **0.9312** | **0.902** | **0.8061** | **0.8231** |
| RTF-IGM (9) | 0.8825 | 0.9308 | 0.8982 | 0.8097 | 0.8252 |
| TF-IGM (5) | 0.8827 | 0.931 | 0.8984 | 0.8099 | 0.8254 |
| TF-IGM (6) | 0.8839 | 0.9307 | 0.8998 | 0.8096 | 0.8254 |
| TF-IGM (7) | 0.8845 | 0.9312 | 0.9005 | 0.8102 | 0.8259 |
| TF-IGM (8) | 0.8841 | 0.931 | 0.9006 | 0.8071 | 0.8236 |
| TF-IGM (9) | 0.8821 | 0.9308 | 0.8977 | 0.8092 | 0.824 |

Table 6.26: Removing stop words or keeping them (for MMI_IXL using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| SW removal: Yes | 0.885 | 0.9312 | 0.902 | 0.8061 | 0.8231 |
| **SW removal: No** | **0.8884** | **0.9315** | **0.9067** | **0.8034** | **0.8219** |

**Optimization step 5: Removal of stop words** Not removing stop words improves $F_2$ score by 0.0034 as shown in table 6.26 . Because of this small improvement, we decide to use keep stop words. This leads to the new top rankend features by TF such as "the", "of", "be", "and" and "in".

**Optimization step 6: Feature selection** Figure 6.10 reveals that TF feature selection is superior to $\chi^2$ and IG as its $F_2$ score is highest for every selected value for max words varying from 100 to 1500. $\chi^2$ selects "operator", "display", "command", "train" and "must" as top five ranked features and IG selects "operator", "display", "command", "must" and "train". Since the subsystem deals with human interaction with a interlocking system, keywords as "operator", "display" and "command" would intuitively make more sense, but the benchmark results show differently.
Since TF (1500) has the highest $F_2$ score in the range 100 to 1500, we extended the interval to 2500. This shows a peak at 1600 with a $F_2$ score of 0.893 and at 2400 with a $F_2$ score of 0.8936. We therefore selected TF (2400) as feature selection method.

**Final performance** The final results on the test set are shown in table 6.27. Overall, we increased recall from 0.5312 to 0.8394. This results in an increase of 0.2061 to a total

Figure 6.10: Comparison of feature selection methods (for MMI_IXL using RF)

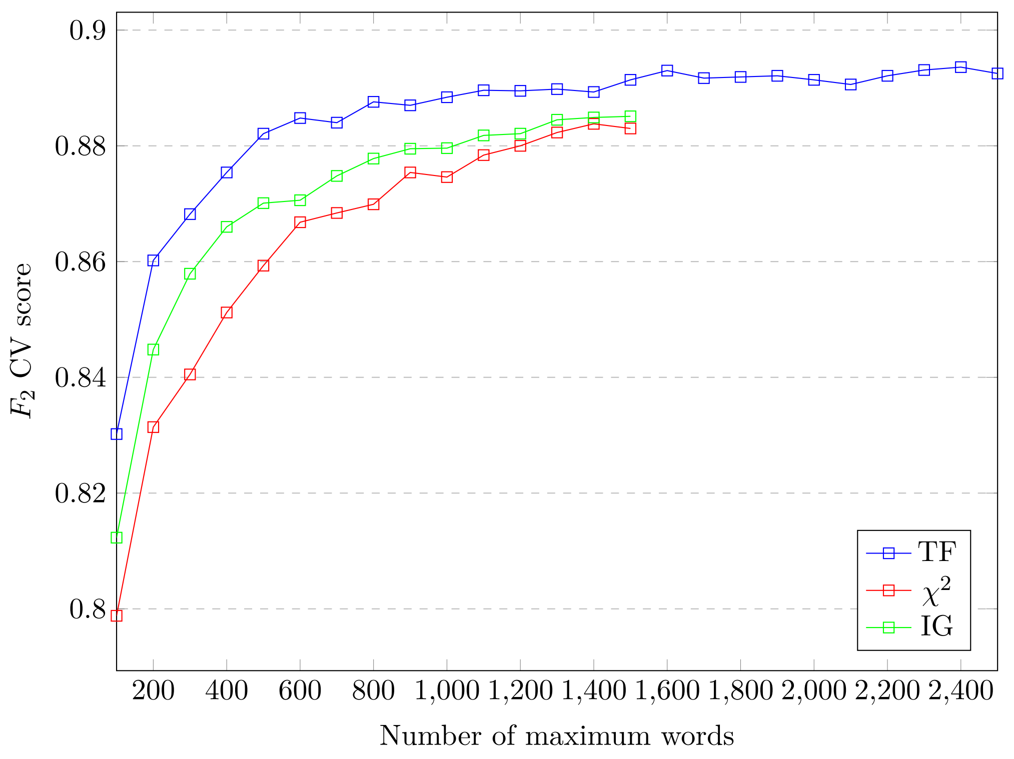


Table 6.27: Improvements of each optimization step (for MMI_IXL using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.5673 | 0.938 | 0.5312 | 0.9829 | 0.7797 |
| RUS | +0.1389 | -0.0003 | +0.3687 | -0.1507 | -0.4001 |
| Lemmatizer | -0.0026 | +0.0004 | -0.0041 | +0.0002 | -0.0008 |
| $n \in \{1\}$ | 0 | 0 | 0 | 0 | 0 |
| RTF-IGM (8) | +0.001 | -0.0001 | +0.0012 | +0.0002 | +0.0006 |
| SW removal: No | +0.0815 | -0.0474 | -0.0484 | +0.0011 | +0.2278 |
| TF (2400) | -0.0127 | -0.0094 | -0.0092 | -0.0117 | -0.019 |
| Total increase | +0.2061 | -0.0568 | +0.3082 | -0.1609 | -0.1914 |
| Total | 0.7734 | 0.8812 | 0.8394 | 0.822 | 0.5883 |

Table 6.28: Comparison of sampling methods (for IXL_HW using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Imbalanced | 0.568 | 0.925 | 0.5273 | 0.9891 | 0.8213 |
| **RUS** | **0.6387** | **0.9184** | **0.8731** | **0.814** | **0.3083** |
| SMOTE (2) | 0.5995 | 0.92 | 0.5792 | 0.9762 | 0.6974 |
| SMOTE (3) | 0.6098 | 0.921 | 0.5902 | 0.9764 | 0.7036 |
| SMOTE (4) | 0.601 | 0.9172 | 0.582 | 0.9754 | 0.6916 |
| SMOTE (5) | 0.6017 | 0.921 | 0.582 | 0.9759 | 0.6961 |
| SMOTE (6) | 0.6056 | 0.9194 | 0.5874 | 0.9752 | 0.6913 |

Table 6.29: Comparison of word inflection removal methods (for IXL_HW using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **Stemmer** | **0.8632** | **0.9195** | **0.8788** | **0.7883** | **0.8059** |
| Lemmatizer | 0.8605 | 0.9184 | 0.8763 | 0.7846 | 0.8028 |

of 0.7734 on $F_2$ score, but also decreases AUROC by 0.0568, specificity by 0.1609 and precision by 0.1914. Since we are optimizing for $F_2$ and recall, we successfully increased the classifiers ability to classify positive instances while adjusting its performance to classify negative instances (specificity) on the same level of about 0.82.

### 6.4.3 Subsystem IXL_HW

This section covers the results of each selected classifier for every method of each of the six optimization steps for the data set IXL_HW. It follows the same structure as the previous documentation on the previous subsystems.

#### Random forest

The results for the Random Forest (RF) classifier are presented in the following paragraphs. Each optimization step is discussed individually.

**Optimization step 1: Sampling** Results of the comparison of different sampling methods are shown in table 6.28. An imbalanced training set and a rebalanced training set using SMOTE has a very high specificity of above 0.975, but with recall below 0.6. SMOTE increases recall, depending on its parameter, by up to 0.0628 when using nearest neighbors set to 3. RUS increases recall to 0.8731 while decreasing specificity to 0.814. The low precision can be explained by the class *Skew* of 10.56. Because of this and its highest $F_2$ score of 0.6387, we select RUS as sampling method.

Table 6.30: Comparison of n-grams (for IXL_HW using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| $\boldsymbol{n \in \{1\}}$ | **0.8632** | **0.9195** | **0.8788** | **0.7883** | **0.8059** |
| $n \in \{1,2\}$ | 0.8592 | 0.9197 | 0.8733 | 0.7912 | 0.8071 |
| $n \in \{1,2,3\}$ | 0.859 | 0.919 | 0.8736 | 0.7889 | 0.8055 |
| $n \in \{1,2,3,4\}$ | 0.8608 | 0.9195 | 0.8751 | 0.792 | 0.808 |
| $n \in \{1,2,3,4,5\}$ | 0.8609 | 0.9188 | 0.8758 | 0.7893 | 0.8061 |

Table 6.31: Comparison of feature weights (for IXL_HW using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| TF | 0.8632 | 0.9195 | 0.8788 | 0.7883 | 0.8059 |
| 0/1 | 0.862 | 0.9194 | 0.8778 | 0.7867 | 0.8045 |
| TF-IDF | 0.8621 | 0.9191 | 0.8775 | 0.788 | 0.8054 |
| RTF-IGM (5) | 0.8622 | 0.9188 | 0.8778 | 0.7875 | 0.8051 |
| RTF-IGM (6) | 0.863 | 0.919 | 0.879 | 0.7865 | 0.8047 |
| RTF-IGM (7) | 0.8624 | 0.9191 | 0.878 | 0.7876 | 0.8053 |
| **RTF-IGM (8)** | **0.8649** | **0.9195** | **0.881** | **0.7882** | **0.8062** |
| RTF-IGM (9) | 0.8627 | 0.919 | 0.8785 | 0.7869 | 0.8048 |
| TF-IGM (5) | 0.8621 | 0.9189 | 0.8781 | 0.7858 | 0.804 |
| TF-IGM (6) | 0.8637 | 0.9191 | 0.8795 | 0.7876 | 0.8056 |
| TF-IGM (7) | 0.8623 | 0.9189 | 0.878 | 0.7869 | 0.8048 |
| TF-IGM (8) | 0.8639 | 0.9193 | 0.8802 | 0.7859 | 0.8044 |
| TF-IGM (9) | 0.8624 | 0.9188 | 0.8784 | 0.7858 | 0.804 |

**Optimization step 2: Removal of word inflection** The use of a lemmatizer showed no improvement as seen in table 6.29. This is the reason to continue using the stemmer.

**Optimization step 3: n-grams** Table 6.30 shows the results for different n-grams. The use of multi words does not increase $F_2$ score and therefore we continue only using uni-grams.

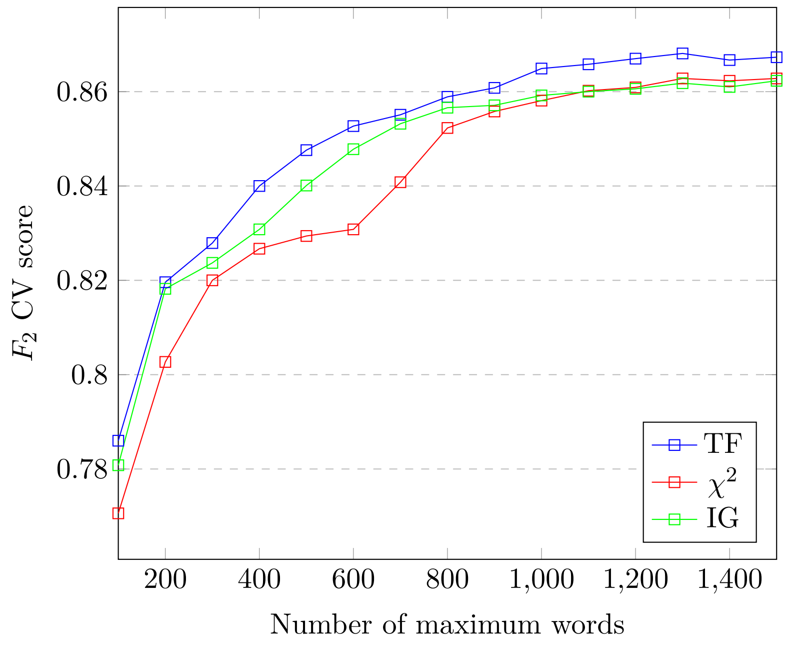
**Optimization step 4: Feature weight** The results for different feature weights are pointed out in table 6.31. The best $F_2$ results are achieved when using RTF-IGM ($\lambda = 8$) with score of 0.8649. RTF-IGM $\lambda = 8$, TF-IGM with $\lambda = 8$ and $\lambda = 6$ and TF are the top four weights regarding $F_2$ score. Because of this, we selected RTF-IGM (8).

**Optimization step 5: Removal of stop words** The use of stop words removal lead to better $F_2$ score by +0.0022. We therefore continue removing stop words.

Table 6.32: Removing stop words or keeping them (for IXL_HW using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| **SW removal: Yes** | **0.8649** | **0.9195** | **0.881** | **0.7882** | **0.8062** |
| SW removal: No | 0.8627 | 0.9193 | 0.8764 | 0.797 | 0.812 |

Figure 6.11: Comparison of feature selection methods (for IXL_HW using RF)



**Optimization step 6: Feature selection** Figure 6.11 shows the comparison between TF, $\chi^2$ and IG. Using TF leads to better $F_2$ performance regardless of the number of maximum words. TF (1300) has a local maximum in this observed range of 0.8681 and is therefore selected.

**Final performance** The final results on the test set are shown in table 6.33. While we manage to improve recall by +0.349, we also decreased precision by 0.5207. This is because of the test set *Skew* of 10.56 and the decreased specificity of 0.8064. Overall, we optimized the model's configuration, so that is more suitable for this use-case.

### 6.4.4 Subsystem Bid_Manager

This section covers the results of each selected classifier for every method of each of the six optimization steps for the data set Bid_Manager. It follows the same structure as the documentation on the previous subsystems.

Table 6.33: Improvements of each optimization step (for IXL_HW using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.568 | 0.925 | 0.5273 | 0.9891 | 0.8213 |
| RUS | +0.0707 | -0.0066 | +0.3458 | -0.1751 | -0.513 |
| Stemmer | 0 | 0 | 0 | 0 | 0 |
| $n \in \{1\}$ | 0 | 0 | 0 | 0 | 0 |
| RTF-IGM (8) | -0.0037 | -0.0002 | -0.0032 | -0.0016 | -0.0027 |
| SW removal: Yes | 0 | 0 | 0 | 0 | 0 |
| TF (1300) | -0.0017 | +0.0011 | +0.0065 | -0.006 | -0.005 |
| Total increase | +0.0653 | -0.0058 | +0.349 | -0.1827 | -0.5207 |
| Total | 0.6333 | 0.9193 | 0.8763 | 0.8064 | 0.3005 |

Table 6.34: Comparison of sampling methods (for Bid_Manager using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Imbalanced | 0.0603 | 0.8549 | 0.05 | 0.9958 | 0.3462 |
| **RUS** | **0.359** | **0.8393** | **0.8099** | **0.7122** | **0.1114** |
| SMOTE (2) | 0.1358 | 0.8516 | 0.1278 | 0.9743 | 0.1811 |
| SMOTE (3) | 0.1131 | 0.8524 | 0.1056 | 0.9751 | 0.1583 |
| SMOTE (4) | 0.1302 | 0.8539 | 0.1222 | 0.9746 | 0.176 |
| SMOTE (5) | 0.1479 | 0.8506 | 0.1389 | 0.9753 | 0.2 |
| SMOTE (6) | 0.1476 | 0.8571 | 0.1389 | 0.9748 | 0.1969 |

### Random forest

The results for the Random Forest (RF) classifier are presented in the following paragraphs. Each optimization step is discussed individually.

**Optimization step 1: Sampling** Table 6.34 shows the results for different sampling methods. Using no sampling strategy, one could say that the classifier is only detects negative instances, being close to a trivial rejector because of the specificity of 0.9958 and recall of 0.05. SMOTE has an increased recall of about 0.10 to 0.14, and a slightly decrease specificity of about 0.02 while precision is approximately halved. RUS increases recall to 0.8099 by decreasing specificity to a moderate score of 0.7122. This results in an $F_2$ score of 0.359 and therefore we decide to select RUS.

**Optimization step 2: Removal of word inflection** The use of a lemmatizer improves $F_2$ score by 0.0081 as seen in table 6.35 and is therefore selected for this optimization step.

Table 6.35: Comparison of word inflection removal methods (for Bid_Manager using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Stemmer | 0.7827 | 0.8219 | 0.7998 | 0.6908 | 0.7215 |
| **Lemmatizer** | **0.7908** | **0.8259** | **0.8108** | **0.6848** | **0.7204** |

Table 6.36: Comparison of n-grams (for Bid_Manager using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| $n \in \{1\}$ | 0.7908 | 0.8259 | 0.8108 | 0.6848 | 0.7204 |
| $\boldsymbol{n \in \{1, 2\}}$ | **0.793** | **0.8251** | **0.8139** | **0.6828** | **0.7197** |
| $n \in \{1, 2, 3\}$ | 0.7926 | 0.8253 | 0.8135 | 0.6822 | 0.7192 |
| $n \in \{1, 2, 3, 4\}$ | 0.7918 | 0.8254 | 0.8125 | 0.6824 | 0.7191 |
| $n \in \{1, 2, 3, 4, 5\}$ | 0.7916 | 0.825 | 0.8121 | 0.6829 | 0.7193 |

Table 6.37: Comparison of feature weights (for Bid_Manager using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| TF | 0.793 | 0.8251 | 0.8139 | 0.6828 | 0.7197 |
| 0/1 | 0.7898 | 0.8242 | 0.8098 | 0.6837 | 0.7193 |
| TF-IDF | 0.7914 | 0.8252 | 0.8117 | 0.6844 | 0.7202 |
| RTF-IGM (5) | 0.7933 | 0.8252 | 0.8141 | 0.6834 | 0.7201 |
| RTF-IGM (6) | 0.7918 | 0.8259 | 0.8125 | 0.6818 | 0.7188 |
| RTF-IGM (7) | 0.7933 | 0.8258 | 0.8141 | 0.6834 | 0.7202 |
| **RTF-IGM (8)** | **0.7937** | **0.8257** | **0.8146** | **0.6829** | **0.72** |
| RTF-IGM (9) | 0.7936 | 0.8257 | 0.8146 | 0.6831 | 0.7201 |
| TF-IGM (5) | 0.7911 | 0.8245 | 0.8119 | 0.6804 | 0.7177 |
| TF-IGM (6) | 0.7924 | 0.8253 | 0.8135 | 0.6809 | 0.7185 |
| TF-IGM (7) | 0.7927 | 0.8251 | 0.8136 | 0.6826 | 0.7195 |
| TF-IGM (8) | 0.7932 | 0.8255 | 0.814 | 0.6833 | 0.7201 |
| TF-IGM (9) | 0.7933 | 0.8257 | 0.8143 | 0.6822 | 0.7195 |

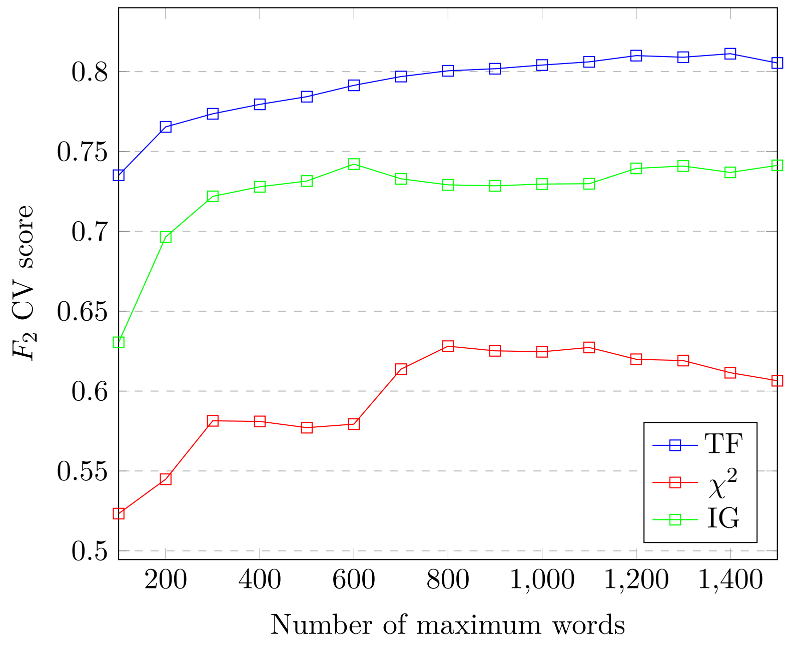
**Optimization step 3: n-grams** Table 6.36 shows the results for different n-grams. Using $n \in \{1, 2\}$ improves $F_2$ score by 0.0022, which is why we select this values for $n$.

**Optimization step 4: Feature weight** The result for different feature weights are shown in table 6.37. The best $F_2$ result is achieved when using RTF-IGM ($\lambda = 8$) with a score of 0.8649. RTF-IGM ($\lambda \in \{8, 9, 5, 7\}$) shows to be the top four best methods for this data set when comparing $F_2$. We therefore selected RTF-IGM ($\lambda = 8$).

Table 6.38: Removing stop words or keeping them (for Bid_Manager using RF)

| | Cross validation | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| SW removal: Yes | 0.7937 | 0.8257 | 0.8146 | 0.6829 | 0.72 |
| **SW removal: No** | **0.8041** | **0.8343** | **0.8272** | **0.6842** | **0.7238** |

Figure 6.12: Comparison of feature selection methods (for Bid_Manager using RF)



**Optimization step 5: Removal of stop words** Using no stop words removal increases $F_2$ score by 0.0104 as seen in table 6.38. Therefore, we decide to keep the stop words.

**Optimization step 6: Feature selection** As figure 6.12 shows, TF is superior to IG and $\chi^2$ regarding $F_2$ score. We therefore decided to use TF with a maximum count of features of 1400, because this has the highest $F_2$ score of 0.8112.

**Final performance** The final results on the test set are shown in table 6.39. Each step except for the feature weight, improves recall, AUROC and $F_2$ score. RUS improves recall most by 0.7599. We increased recall zo 0.8268 by decreasing specificity by 0.2996 to an absolut value of 0.6962. Although the precision reveals that this classifier has nine times more FP than TP. This is because of the class distribution *Skew* of 22.54 and a specificity of 0.6962. Overall when using this optimized configuration, this classifier achieves better performance for this use-case because of its higher $F_2$ score and recall.

Table 6.39: Improvements of each optimization step (for Bid_Manager using RF)

| | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| Baseline | 0.0603 | 0.8549 | 0.05 | 0.9958 | 0.3462 |
| RUS | +0.2987 | -0.0156 | +0.7599 | -0.2836 | -0.2348 |
| Lemmatizer | +0.0015 | +0.0043 | +0.0109 | -0.004 | -0.0001 |
| $n \in \{1, 2\}$ | +0.0065 | +0.0043 | +0.0251 | -0.0053 | +0.0013 |
| RTF-IGM (8) | -0.0009 | +0.0005 | -0.0036 | +0.0009 | -0.0002 |
| SW removal: No | +0.0075 | +0.0108 | +0.0278 | -0.005 | +0.0015 |
| TF (1400) | +0.0008 | +0.0019 | +0.0068 | -0.0026 | -0.0001 |
| Total increase | +0.3141 | +0.0062 | +0.8268 | -0.2996 | -0.2323 |
| Total | 0.3744 | 0.861 | 0.8768 | 0.6962 | 0.1138 |

Table 6.40: Summary of final performance for each subsystem, using RF

| Subsystem | Train-Test | | | | |
|---|---|---|---|---|---|
| | $F_2$ | AUROC | recall | specificity | precision |
| PM | 0.8185 | 0.9424 | 0.8747 | 0.8579 | 0.6511 |
| MMI_IXL | 0.7734 | 0.8812 | 0.8394 | 0.822 | 0.5883 |
| IXL_HW | 0.6333 | 0.9193 | 0.8763 | 0.8064 | 0.3005 |
| Bid_Manager | 0.3744 | 0.861 | 0.8768 | 0.6962 | 0.1138 |

Table 6.41: Summary of optimized configurations for all four subsystem using RF

| Step | PM | MMI_IXL | IXL_HW | Bid_Manager |
|---|---|---|---|---|
| Sampling | RUS | RUS | RUS | RUS |
| Word inflections r. | Lemmatizer | Lemmatizer | Stemmer | Lemmatizer |
| n-grams | $n \in \{1, 2, 3\}$ | $n \in \{1\}$ | $n \in \{1\}$ | $n \in \{1, 2\}$ |
| Feature weight | TF | RTF-IGM (8) | RTF-IGM (8) | RTF-IGM (8) |
| Stop words r. | Yes | No | Yes | No |
| Feature selection | IG (1300) | TF (2400) | TF (1300) | TF (1400) |

### 6.4.5 Summary of final performance

This section shows the final performance of the optimized models' configuration using Random Forest for each subsystem. This section is a copy of the above results, but provides better readability for comparison in table 6.40.

### 6.4.6 Summary of optimized configurations

This section shortly summarizes the previous optimized models' configurations for each subsystem and finally answers research question 1. Since the previous section contains much information, this section gives a summary in table 6.41.

## 6.5 Generalization

This section discusses the previously optimized models' configurations for each subsystem on generalization. The optimized configuration of one subsystem is used for the other subsystems. The beforehand selected Random Forest classifier is trained upon these data sets and the performance is evaluated. It then continues in investigating, why some methods of the optimization steps affect the performance on the other data sets. For this, the methods of the optimized configurations are exchanged with those from the other data sets. Afterwards, five additional subsystems are benchmarked, but not optimized, using the configurations of the four data sets PM, MMI_IXL, IXL_HW and Bid_Manager. Finally, a conclusion on generalization is drawn.

### 6.5.1 Between subsystem analysis

Tables 6.42 show for each data set the comparisons when using the configuration of the other subsystems which are indicated as rows. Each sub-table represents the evaluated data set and the subsystem names in the rows indicate the used configuration. The absolute results are highlighted gray and shown when using the configuration optimized for the subsystem. Relative numbers are given for the configuration of the other subsystems.

As those results reveal, the $F_2$ score differences between the highest and lowest value for PM is 0.006, for MMI_IXL 0.0091, for IXL_HW 0.0246 and for Bid_Manager 0.0515. The highest $F_2$ score is achieved for every subsystem when using the optimized configuration for PM when evaluated on the test set. When using MMI_IXL's configuration, the $F_2$ score of IXL_HW and Bid_Manager increases, but decreases the score of PM. The setup of IXL_HW and Bid_Manager increases only MMI_IXL's $F_2$ score.

When comparing the change of the AUROC score, all values decrease when applying the configuration of another subsystem with only one exception: When using PM's setup on the Bid_Manager data set. This increases AUROC by 0.0039. Overall, the decreases of the AUROC are mostly less than 0.01, with exception to Bid_Manager's configuration on IXL_HW (-0.0105) and vice versa (-0.0178).

Since $F_2$ is an aggregated metric of recall and precision (see section 3.4), we also take a look at those metrics individually. At this level, specificity is also taken into account, since specificity also affects precision.

Using the configuration selected for PM decreases recall by 0.0492 on MMI_IXL, 0.0703 on IXL_HW and 0.1556 on Bid_Manager. In return, specificity is increased by 0.0447, 0.0602 and 0.1247 respectively. This increased negative prediction ability results in an increased precision of 0.039 to 0.064, depending on the subsystem.

When applying MMI_IXL's configuration on PM, recall decreases by 0.0316 and specificity increases by 0.0372. On IXL_HW, recall decreases by 0.0062 and specificity increases by 0.0002. Applied on Bid_Manager, recall decreases by 0.0181 and specificity increases by 0.0127. This increases precision on PM by 0.058, on IXL_HW by 0.0016 and on Bid_Manager by 0.0023.

Table 6.42: Train-test results using RF, using the configurations on all four data sets

| | Train-Test | | | | |
|---|---|---|---|---|---|
| configuration from | $F_2$ | AUROC | recall | specificity | precision |
| | | | PM | | |
| Bid_Manager | -0.003 | -0.0024 | -0.0235 | +0.0305 | +0.047 |
| IXL_HW | -0.0024 | -0.0023 | -0.0112 | +0.0126 | +0.0179 |
| MMI_IXL | -0.0061 | -0.0019 | -0.0316 | +0.0372 | +0.058 |
| PM | 0.8185 | 0.9424 | 0.8747 | 0.8579 | 0.6511 |
| | | | MMI_IXL | | |
| Bid_Manager | +0.0005 | -0.0013 | +0.0042 | -0.0024 | -0.0021 |
| IXL_HW | +0.0054 | ±0 | -0.0021 | +0.0071 | +0.0092 |
| MMI_IXL | 0.7006 | 0.938 | 0.9016 | 0.8252 | 0.3705 |
| PM | +0.0091 | -0.0042 | -0.0513 | +0.0447 | +0.057 |
| | | | IXL_HW | | |
| Bid_Manager | -0.0093 | -0.0105 | -0.0132 | -0.001 | -0.0044 |
| IXL_HW | 0.6333 | 0.9193 | 0.8763 | 0.8064 | 0.3005 |
| MMI_IXL | +0.004 | -0.0019 | +0.0062 | +0.0001 | +0.0016 |
| PM | +0.0153 | -0.0061 | -0.0703 | +0.0602 | +0.0636 |
| | | | Bid_Manager | | |
| Bid_Manager | 0.3744 | 0.861 | 0.8768 | 0.6962 | 0.1138 |
| IXL_HW | -0.0134 | -0.0178 | -0.071 | +0.0212 | -0.0011 |
| MMI_IXL | +0.0021 | -0.0045 | -0.0181 | +0.0127 | +0.0023 |
| PM | +0.038 | +0.0039 | -0.1556 | +0.1247 | +0.039 |

IXL_HW's configuration decreases recall on all three subsystems. On Bid_Manager, recall and precision drops by 0.071 and 0.0011, specificity increases by 0.0212. When applying this configuration on PM, recall decreases by 0.0316, but precision and specificity increases by 0.0179 and 0.0372. This benchmark setup applied on MMI_IXL decreases recall by 0.0021 and increases precision and specificity by 0.0092 and 0.0071.

The use of the configuration of Bid_Manager on PM results in a decrease in recall of 0.0235 and an increase in specificity and precision of 0.0306 and 0.047; On MMI_IXL an increase on recall of 0.0042 and a decrease in specificity and precision of 0.0024 and 0.0021. This setup used on the IXL_HW data set decreases in recall, specificity and precision by 0.0132, 0.001 and 0.0044, respectively.

**Differences between technical or non-technical subsystems** Our initial thought that the subsystems may correlate with technical or non-technical subsystems is not supported. When using PM's configuration on Bid_Manager, the recall and specificity changes by over 0.12. The configuration from a technical subsystem such as MMI_IXL only changes the results by less than 0.02. Vice versa, when using Bid_Manager's setup on PM, recall decreases by more than 0.02 and specificity increases over 0.03. On the other hand, when using IXL_HW's setup on PM, recall only decreases by 0.0112. This means, that some technical subsystems' configurations work better on PM or Bid_Manager than the other non-technical subsystem's configuration.

### 6.5.2 Comparison of configurations' methods

These results presented in table 6.42 show that PM is a special case. When applying the optimized configuration for PM on the PM data set, recall is highest and specificity is lowest compared to the other subsystems' configurations. However, when applying PM's configuration on the other three subsystems, the results are contrary, meaning that they have lowest recall and highest specificity.

To be more precise, we are interested, which methods of the configuration of PM affects the performance of MMI_IXL, IXL_HW and Bid_Manager, so that recall decreases by 0.0513, 0.0703 and 0.1556, specificity increases by 0.0447, 0.0602 and 0.1247, and precision increases by 0.057, 0.0636 and 0.039.

This section first presents the configurations' methods used by MMI_IXL, IXL_HW and Bid_Manager different from PM, and tests the impact of changing one, two or three methods on those three data sets to those from PM. Exchanging four methods would lead to the use of PM's setup. This section finishes with a summary on the findings.

#### Different methods of PM

To further investigate this phenomena, the methods of the other subsystems' optimized configurations different from PM's optimized configuration are identified. Table 6.41 shows the configurations for the four subsystems. It reveals that PM uses $n \in \{1, 2, 3\}$-grams, term frequency as feature weight and information gain as feature selection method. Additionally, compared to PM, MMI_IXL and Bid_Manager does not remove stop words and IXL_HW use stemming instead of lemmatization.

#### Exchanging one method

The effects of those different methods are now compared in table 6.43. The methods indicated as rows are exchanged with the previously optimized configuration for each subsystem. The first row shows the performance when the optimized on the data set; The last row shows the comparison when using PM's configuration.

The exchange of a single method on the MMI_IXL and IXL_HW data sets affects recall, precision and specificity by less than 0.01 (see table table 6.43, row two to five). On the Bid_Manager data set, those metrics can be seen as nearly constant (changes less than 0.01) for the use of tri-grams and PM's feature weight (TF). When using stop words removal, recall and precision decrease by 0.0394 and 0.0008 and specificity increases by 0.0113. However, the use of another feature selection method (information gain) drastically impacts recall, resulting in a decrease of 0.1336 and an increase in specificity and precision of 0.1321 and 0.048.

Table 6.43: Changing one method of optimized configurations

| Methods different from configuration | Train-Test $F_2$ | AUROC | recall | specificity | precision |
|---|---|---|---|---|---|
| | | | MMI_IXL | | |
| MMI_IXL's setup | 0.7006 | 0.938 | 0.9016 | 0.8252 | 0.3705 |
| $n \in \{1,2,3\}$ | +0.0004 | -0.0016 | -0.0035 | +0.0029 | +0.0029 |
| Feature weight: TF | -0.0015 | -0.0004 | +0 | -0.0016 | -0.002 |
| Remove stop words | +0.0059 | +0.0001 | -0.0088 | +0.0122 | +0.0147 |
| IG (1300) | +0.0086 | -0.0008 | -0.0072 | +0.0138 | +0.0177 |
| PM's setup | +0.0091 | -0.0042 | -0.0513 | +0.0447 | +0.057 |
| | | | IXL_HW | | |
| IXL_HW's setup | 0.6333 | 0.9193 | 0.8763 | 0.8064 | 0.3005 |
| $n \in \{1,2,3\}$ | +0.0059 | -0.0009 | -0.0055 | +0.0096 | +0.0094 |
| Feature weight: TF | +0.0021 | -0.0002 | -0.003 | +0.004 | +0.0039 |
| Lemmatizer | -0.0006 | -0.0011 | -0.0042 | +0.0022 | +0.0013 |
| IG (1300) | -0.0063 | -0.0025 | -0.0067 | -0.0021 | -0.004 |
| PM's setup | +0.0153 | -0.0061 | -0.0703 | +0.0602 | +0.0636 |
| | | | Bid_Manager | | |
| Bid_Manager's setup | 0.3744 | 0.861 | 0.8768 | 0.6962 | 0.1138 |
| $n \in \{1,2,3\}$ | +0.0008 | -0.001 | +0.0058 | -0.0021 | 0 |
| Feature weight: TF | -0.0026 | -0.001 | -0.0068 | +0.0002 | -0.0007 |
| Remove stop words | -0.0076 | -0.0113 | -0.0394 | +0.0113 | -0.0008 |
| IG (1300) | +0.0576 | +0.0168 | -0.1336 | +0.1321 | +0.048 |
| PM's setup | +0.038 | +0.0039 | -0.1556 | +0.1247 | +0.039 |

### Exchanging two methods

Since those the exchange of a single method do not completely explain the fluctuation in performance, we decide to also investigate combinations of two. Namely, we compare the performance changes of tri-grams and information gain, removal stop words and information gain (for MMI_IXL and Bid_Manager) and lemmatizer and information gain (for IXL_HW). The results are shown in table 6.44 in row two to four for each data set and discussed in the next paragraphs.

For MMI_IXL, the combination of the use of stop words and information gain decrease recall by 0.0141 and increases specificity and precision by 0.0197 and 0.0248. The use of tri-grams and information gain affects those metrics more, resulting in a decrease on recall of 0.036 and an increase in specificity and precision of 0.0406 and 0.0535. This combination results are close to the increase in specificity and precision when using PM's configuration, but decrease recall less, resulting in a higher $F_2$ score. The combination of TF as feature weight with information gain as feature selection method changes all five metrics by less than 0.003 when compared just using information gain with RTF-IGM ($\lambda = 8$) as feature weight.

Exchanging two methods of the data set IXL_HW with configuration optimized for PM, concrete using tri-grams and information gain and using a lemmatizer in combination with information gain, results in a decrease in recall by 0.0668 and 0.0721, and an increase of specificity by 0.0509 and 0.0681, and precision by 0.0491 and 0.0031. Those results are very close to when using PM's configuration. In contrast, when combining information gain and TF as feature weight, the change in all five metrics is only maximum 0.0025 compared to just using IG.

Besides exchanging the feature selection method on the Bid_Manager data set with information gain, we also try the combination with tri-grams, the removal of stop words and TF as feature weight. The first combination decreases recall by 0.1681 and increases specificity and precision by 0.1469 and 0.0541. The second combination results in a change in those metrics by -0.1616, +0.129 and +0.0411, respectively. All metrics vary by more than using PM's optimized configuration. Therefore, we expect the combination of all three methods to increase recall and decrease specificity and precision. The combination of information gain with TF as feature weight only shows a change of less than 0.002 on all five metrics compared to just using information gain.

### Exchanging three methods

The change of up to two methods of PM's setup revealed, that information gain in combination another method decreases recall and increases precision and specificity on all three data sets, with one exception: the combination of TF as feature weight and IG. Although this combination does not result in any significant changes compared to just using IG, we decided to also investigate the influence of the feature weight. For MMI_IXL and Bid_Manager, we investigate the impact of using tri-grams, stop words removal and information gain, as well as using tri-grams, TF as feature weight and information gain. Since IXL_HW already removes stop words, we look at the impact of using a lemmatizer instead in combination with tri-grams and information gain. Additionally, we evalute this data set using tri-grams, information gain and TF as feature weight. The results are continued in table 6.44 in row five and six.

Using tri-grams, stop words and information gain as feature selection method on the data set MMI_IXL decreases recall by 0.0315 and increases specificity and precision by 0.0226 and 0.0566. Compared to just using tri-grams and information gain, the delta of recall increases by 0.0045, precision's delta decreases by 0.0031 and the difference of specificity decreases 0.018. When using tri-grams, TF as feature weight and information gain, the results of all five performance metrics close to the ones when just using tri-grams and information gain. This means, that the exchange of the feature weight affects performance slightly.

Looking at row five of the results of IXL_HW, where tri-grams, a lemmatizer and information gain are used, recall drops by 0.0646, precision and specificity increase by 0.0608 and 0.0663. Compared to using TF as feature weight with tri-grams and information gain, recall decreases by 0.0658 and specificity and precision increase by

Table 6.44: Changing two to three methods of optimized configurations

| Methods different from optimized configuration | Train-Test $F_2$ | AUROC | recall | specificity | precision |
|---|---|---|---|---|---|
| | MMI_IXL | | | | |
| MMI_IXL's setup | 0.7006 | 0.938 | 0.9016 | 0.8252 | 0.3705 |
| $n \in \{1,2,3\}$ & IG (1300) | +0.0157 | -0.0046 | -0.036 | +0.0406 | +0.0535 |
| r. stop words & IG (1300) | +0.0097 | +0.0003 | -0.0141 | +0.0197 | +0.0248 |
| Feature weight: TF & & IG (1300) | +0.0054 | -0.0017 | -0.0099 | +0.0125 | +0.0148 |
| $n \in \{1,2,3\}$ & r. stop words & & IG (1300) | +0.0074 | -0.0045 | -0.0315 | +0.0226 | +0.0566 |
| $n \in \{1,2,3\}$ & Feature weight: TF & IG (1300) | +0.0166 | -0.004 | -0.0386 | +0.0433 | +0.0576 |
| PM's setup | +0.0091 | -0.0042 | -0.0513 | +0.0447 | +0.057 |
| | IXL_HW | | | | |
| IXL_HW's setup | 0.6333 | 0.9193 | 0.8763 | 0.8064 | 0.3005 |
| $n \in \{1,2,3\}$ & IG (1300) | +0.0075 | -0.0101 | -0.0668 | +0.0509 | +0.0491 |
| Lemmatizer & IG (1300) | -0.0033 | -0.0043 | -0.0721 | +0.0681 | -0.0031 |
| Feature weight: TF & & IG (1300) | -0.004 | -0.0019 | -0.006 | -0.0002 | -0.0016 |
| $n \in \{1,2,3\}$ & Lemmatizer & & IG (1300) | +0.0199 | -0.0049 | -0.0646 | +0.0608 | +0.0663 |
| $n \in \{1,2,3\}$ & Feature weight: TF & IG (1300) | +0.0098 | -0.0105 | -0.0658 | +0.0523 | +0.0517 |
| PM's setup | +0.0153 | -0.0061 | -0.0703 | +0.0602 | +0.0636 |
| | Bid_Manager | | | | |
| Bid_Manager's setup | 0.3744 | 0.861 | 0.8768 | 0.6962 | 0.1138 |
| $n \in \{1,2,3\}$ & IG (1300) | +0.0559 | +0.0107 | -0.1681 | +0.1469 | +0.0541 |
| r. stop words & IG (1300) | +0.0393 | +0.0023 | -0.1616 | +0.129 | +0.0411 |
| Feature weight: TF & & IG (1300) | +0.0571 | +0.0174 | -0.1353 | +0.1325 | +0.0482 |
| $n \in \{1,2,3\}$ & r. stop words & & IG (1300) | +0.0355 | +0.0032 | -0.0557 | +0.0195 | +0.0383 |
| $n \in \{1,2,3\}$ & Feature weight: TF & IG (1300) | +0.0574 | +0.0104 | -0.164 | +0.1461 | +0.0544 |
| PM's setup | +0.038 | +0.0039 | -0.1556 | +0.1247 | +0.039 |

0.0523 and 0.0517. This means, that a lemmatizer increases performance on predicting negative instances. Furthermore, when comparing the results of row two and six of this data set, all five performance measures are close, similar to MMI_IXL's case, as described in the previous paragraph.

Those two combinations of three applied on the Bid_Manager data set reveal an interesting result: When using tri-grams, stop word removal and information gain, recall drops by 0.0557, specificity and precision increase by 0.0195 and 0.0383. This affect the performance significantly lower than using combinations of two of tri-grams and information gain. This means for this data set that the use of tri-grams and information gain lowers the classifier's ability to predict positive instances but increases the performance on negative instances. When using also stop word removal, positive instances are predicted better (recall +0.1124), but the performance on negative instances decreases (specificity -0.1274). Similar to MMI_IXL and IXL_HW, when using tri-grams, TF as feature weight and information gain, the performance metrics are close to MMI_IXL's when using tri-grams and IG: recall drops by 0.0164, specificity and precision increase by 0.01461 and 0.0544.

**Summary**

In this section, we investigated the impact of the methods of PM's optimized configuration on the other three data sets, MMI_IXL, IXL_HW and Bid_Manager.

When exchanging just one method of PM's setup on the configuration of these three data sets, only information gain on the Bid_Manager data set shows a significant change in performance. All other methods alone have an impact of $\pm 0.015$ on every performance measure.

Therefore, we investigated combinations with information gain. Information gain with TF as feature weight shows similar performance than just using information gain, meaning that the exchange of the feature weight affects the performance on all five metrics by less than $\pm 0.003$.

For MMI_IXL, the methods that needs to be exchanged to get similar results to using PM's configuration are the use of tri-grams, TF as feature weight and information gain as feature selection method. When using the feature weight of MMI_IXL's setup, the performance is very similar. We therefore conclude that tri-grams and information gain are responsible for the decrease in recall and increase in specificity and precision.

On the IXL_HW data set, all combinations with information gain have an impact on the performance to be close when using PM's configuration with one exception: IG and TF as feature weight. The use of a lemmatizer together with information gain also seem to exceed the changes when using PM's setup. In conclusion, information gain selects features that increase prediction performance on negative instances by simultaneously decreasing the performance on positive instances when used in combination with $n \in \{1, 2, 3\}$-grams and/or a lemmatizer.

The investigations on the Bid_Manager data set reveal interesting information. The use of information gain together with tri-grams or removal of stop words together with the methods selected for Bid_Manager show a similar performance change. This means that information gain selects features, that decrease prediction performance on positive instances but increases performance on negative instances. However, when a combination of those three is used, the performance is less affected. We therefore draw the conclusion that features that improve prediction of positive instances are common words (because they are selected by TF as feature selection) that are most likely stop words (since the removal decreases recall). Using tri-grams introduces combinations of such stop words that "flood" the possible features, meaning mostly (combinations of) stop words are used as features. By now removing stop words, this effect is smaller, explaining why the combination of those three has a smaller impact.

In conclusion, the recall drop and the increase in specificity on those three data sets are caused by using information gain. The optimized configurations of MMI_IXL, IXL_HW and Bid_Manager use the most common words as features (TF as feature selection method), which seem to be responsible for the prediction quality of positive instances (recall). When changing this method to information gain, other features which introduce more information (see section 3.1.6) are selected. This results in a decrease of words with high occurrence and therefore in a decrease of recall. On the other hand, this introduces the classifier's ability to predict negative instances.

### 6.5.3 Benchmark on other subsystems

After investigating four subsystems in detail, the question arises if their optimized models' configurations generalize to other subsystems. To answer this, we select five more subsystems and use the benchmark configurations of the four subsystems. In this section, only the performance on these five data sets are measured. The methods are not analyzed individually as in the previous section 6.5.2.

Since the performance of the classifier may decrease due to increasing class imbalance, we investigate one subsystem per quartile and also the median of $Skew$. For more details on the analysis of class imbalance, see section 5.5.2. From the remaining 55 subsystems, we selected the following five randomly:

- LED_Signals (Quartile 1, $Skew = 31.41$)
- Commissioning (Quartile 2, $Skew = 76.89$)
- Build_Civ (Median, $Skew = 153.93$)
- OBU (Quartile 3, $Skew = 292.72$)
- IT_Network (Quartile 4, $Skew = 439.58$)

Table 6.45: Train-test results using RF, using optimized configurations on other subsystems

| configuration | Train-Test | | | | |
|---|---|---|---|---|---|
| from | $F_2$ | AUROC | recall | specificity | precision |
| | LED_Signals | | | | |
| Bid_Manager | 0.6677 | 0.9717 | 0.9141 | 0.9367 | 0.3297 |
| IXL_HW | 0.6649 | 0.9689 | 0.9272 | 0.9334 | 0.317 |
| MMI_IXL | 0.6087 | 0.9767 | 0.9411 | 0.9088 | 0.258 |
| PM | 0.7811 | 0.974 | 0.9284 | 0.9675 | 0.4791 |
| | Commissioning | | | | |
| Bid_Manager | 0.2213 | 0.9226 | 0.8611 | 0.8052 | 0.056 |
| IXL_HW | 0.2011 | 0.9231 | 0.889 | 0.7686 | 0.0493 |
| MMI_IXL | 0.1986 | 0.9188 | 0.876 | 0.7669 | 0.0489 |
| PM | 0.4934 | 0.9342 | 0.7028 | 0.9676 | 0.2289 |
| | Build_Civ | | | | |
| Bid_Manager | 0.0865 | 0.851 | 0.8168 | 0.7128 | 0.0189 |
| IXL_HW | 0.0985 | 0.8543 | 0.7712 | 0.7543 | 0.0222 |
| MMI_IXL | 0.1136 | 0.8677 | 0.7606 | 0.7817 | 0.0264 |
| PM | 0.2908 | 0.8965 | 0.4482 | 0.9773 | 0.1256 |
| | OBU | | | | |
| Bid_Manager | 0.1422 | 0.9423 | 0.8725 | 0.9061 | 0.0328 |
| IXL_HW | 0.1887 | 0.9647 | 0.8807 | 0.9334 | 0.0457 |
| MMI_IXL | 0.2094 | 0.9664 | 0.8676 | 0.9426 | 0.0521 |
| PM | 0.3603 | 0.9861 | 0.7223 | 0.981 | 0.1217 |
| | IT_Network | | | | |
| Bid_Manager | 0.0373 | 0.8944 | 0.895 | 0.6957 | 0.0078 |
| IXL_HW | 0.0385 | 0.9098 | 0.9291 | 0.6434 | 0.0081 |
| MMI_IXL | 0.0824 | 0.9183 | 0.8542 | 0.8244 | 0.0186 |
| PM | 0.2508 | 0.8937 | 0.4196 | 0.9905 | 0.0987 |

The results when applying the configuration of PM, MMI_IXL, IXL_HW and Bid_Manager on the additional five subsystems, are shown in table 6.45. Each sub-table represents the evaluated data set and the subsystem names in the rows indicate the used subsystem's configuration.

PM's setup has the highest $F_2$ score on every data set, which is also further analyzed on the other four initial data sets in section 6.5.2. Ordering the subsystems by $F_2$ score using PM's setup, leads to $LED\_Signals > Commissioning > OBU > Build\_Civ > IT\_Network$. This means the $F_2$ performance depends on *Skew* with the exception of OBU, which performs better than Build_Civ.

For the subsystems having $Skew < median = 153.92$ (LED_Signals and Commissioning), the optimized configuration on Bid_Manager results in the second highest $F_2$ score;

MMI_IXL's setup results in lowest $F_2$. In contrast, the data sets with $Skew \geq median$ (Build_Civ, OBU and IT_Network), result in second best $F_2$ score with MMI_IXL's configuration where Bid_Manager's lead to lowest $F_2$ performance.

Using IXL_HW's setup results in third best performance on every of the five data sets. On LED_Signals, this third best result is close to second best $F_2$ score and on IT_Network close to fourth best.

The highest AUROC scores are achieved using PM's configuration for Commissioning, Build_Civ and OBU and MMI_IXL's setup for LED_Signals and IT_Network. Ordering the subsystems by the highest highest AUROC, leads to *OBU > LED_Signals > Commissioning > IT_Network > Build_Civ*. The highest performance on OBU and LED_Signals is 0.9861 and 0.9767. For the other subsystems, this measure ranges from 0.89 to 0.934, which shows a high probability that instances are classified correctly (see section 3.4.4).

The observation made in in section 6.5.2, that PM's setup increases specificity and decreases recall also holds on these five data sets. The only exception is LED_Signals, since IXL_HW's and Bid_Manager's configuration shows lower recall than PM's configuration.

### 6.5.4 Conclusion

To come to a conclusion on the topic of generalization of the optimized configuration for four data sets (PM, MMI_IXL, IXL_HW and Bid_Manager), we first evaluated the configuration of the other three subsystems on each data set. This shows that PM's configuration increases the ability of the classifier on predicting negative instances, but decreases the prediction quality of positive instances. We then further investigated this phenomena, revealing that recall is high when common words are used as features (TF as feature selection method). To increase specificity, information gain can be used. Afterwards, these four configurations are used to evaluate the performance of five additional subsystems, which supports these findings.

Table 6.41 presents that random under sampling is used by every optimized configuration for the four data sets when using Random Forest. We therefore suggest using random under sampling as sampling method in combination with Random Forest.

For all nine benchmarked data sets, using PM's setup increases $F_2$ and specificity significantly, but with the cost of a decreased recall. This is mainly due to the use of information gain as feature selection method. However, the interview with the requirements manager reveals the importance of recall. Therefore, it is not possible to draw a conclusion based on benchmark results, if a higher $F_2$ score with decreased recall or higher recall but lower $F_2$ score suits better for this use-case. This is open for a further empirical evaluation, but is left out in this thesis.

CHAPTER 7

# Qualitative analysis - Interview

This chapter describes the second interview with the requirements manager (RM), which goal was to understand why some requirements were classified incorrectly. For the interview, we randomly selected misclassified requirements and showed them to the RM. The RM then had to look at each of the requirements and state what subsystem s/he thinks is appropriate and then state reasons why this is or is not the case. These findings answer RQ3. Also this interview was a good opportunity to present him the results of the benchmark and see if his answer is still clearly towards recall. Since this is only a short discussion, the chapter starts with the presentation of the benchmark results to the requirements manager and continues with the assessment of misclassified requirements.

The setup was the same as in the interview (see chapter 2). The requirements manager, the author, as well as a scribe attended. The answers are taken from the scribe's transcript as well as from the authors' notes.

## 7.1 Presentation of benchmark results

First, the benchmark results were presented to the requirements manager. In the interview (see section 2.7), the RM stated to prefer recall. After now having the classifiers' configuration optimized, the final results are shown to this person.

To show the practical impact of the optimization, the results of the subsystem Bid_Manager are presented, because it achieved lowest $F_2$ score amongst the four evaluated subsystems. We informed the RM, that according to the test set, which includes around 4200 requirements, 180 of them belong to Bid_Manager. 158 requirements are successfully detected but 22 are overseen (recall of 0.8768). This comes with the downside, that from 4050 requirements not belonging to this subsystem, 1230 requirements are incorrectly assigned to this subsystem (specificity of 0.6962). This means this person needs to deselect the subsystem from 1230 requirements and select the missing 22

requirements.
We also made the requirements manager aware that this or even worse performance could be true for other yet untested subsystems, resulting in more false positives.

**Question** What is your opinion now?

**Answer** The requirements manager is still convinced, that deselecting 1200 assignments is faster.

**Impact** This answer validates, in this context, that the performed optimizations are useful. At this point, only an evaluation of the integrated classifier can validate this statement but is leave out in this thesis (see section 8.2).

## 7.2 Assessment of misclassified requirements

This section details why some requirements are incorrectly classified by the classifiers. To answer this question, randomly selected misclassified requirements are shown to the requirements manager to give an explanation why s/he decided to assign a subsystem to a requirement or not in first place. This process provides information why those requirements should or should not be assigned to a subsystem.

Per subsystem, ten false positive (FP) and ten false negative (FN) requirements are randomly selected from each test set, resulting in a total of 80 requirements. The requirements manager is aware, which subsystem was assigned by the model.

This section summarizes the findings into categories of problems found during the interview. Each category contains examples of the problems, an explanation given by the RM, why the example requirements should be assigned differently, and shows the impact on the classifiers for further improvements. The explanation and the impact are the answer to research question 3.

### 7.2.1 Requirements that are headings or prose

The data set contains some requirements that have subsystems assigned, but also have the attribute textIsHeading set to 1 or a req_type of "Prose".

#### Examples

Table 7.1 shows some exemplary requirements. Ten of the randomly chosen requirements belong to this error class.

#### Explanation

In this interview, the RM states, that those assignments to headings are previously done to provide the assigned subsystems' experts with context of these headings. Now there is

Table 7.1: Misclassified requirements that are a heading and Prose

| Subsystem | text | actual | classification |
|---|---|---|---|
| IXL_HW | GENERAL REMARKS | Yes | No |
| IXL_HW | Presenting system conditions | No | Yes |
| MMI_IXL | Operators room | Yes | No |
| MMI_IXL | Basic logging function | No | Yes |
| PM | 3.11.2 Certifications | Yes | No |
| Bid_Manager | Flexibility and adaptability | Yes | No |
| Bid_Manager | INDUCED | No | Yes |

a feature in the requirements management tool or the plugin, that includes all parent headings in the export and this assignment of headings. Therefore, such requirements now need no assignment of subsystems.

#### Impact

This means, that the data set we used actually contains false positives. Section 5.5.3 reveals, that 985 instances are marked as "Prose" but have a subsystem assigned. The examples show these inconsistencies in the data set, where those requirements are sometimes marked as belonging to the subsystem and sometimes not. Therefore, the classifier has problems in detecting such instances. These need to be removed from the training set and could lead to better classification performance.

### 7.2.2 Requirements from bullet lists

Requirements that belong to the same heading are represented as flat list in the data set. During the assessment, we explore that bullet lists are a special case. Those bullet points are are not related to its introducing line or, as this thesis refers to it, context (such as "The system should provide following functions:"). Only this context reveals about which system these requirements are talking.

#### Examples

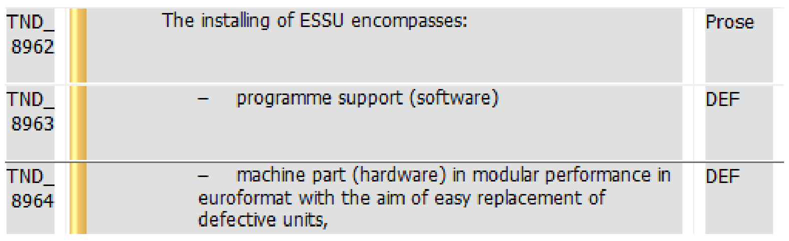
Table 7.2 shows some exemplary requirements where just the text is not enough for a clear understanding. As the text indicates, some of the requirements are taken out of bullet lists, e.g. start with "a)" or end with ";". In total, 18 requirements of randomly picked 80 face a missing context.

#### Explanation

The context is visible in DOORS, since this tool presents the requirements ordered in a table, similar to a spreadsheet, as shown in figure 7.1. The reason for the unrelated flat structure is that those requirements originate from request for tender documents, which are in PDF or Microsoft Word format.

Table 7.2: Misclassified requirements with missing context information

| Subsystem | text | actual | classification |
|---|---|---|---|
| IXL_HW | with N class relays; | Yes | No |
| IXL_HW | isolated section before the announcing signal; | No | Yes |
| MMI_IXL | Actions based on the new schedule defined by the system | Yes | No |
| MMI_IXL | View all the data on the composition of a train | No | Yes |
| PM | Power supply; | Yes | No |
| PM | a) Confirm that the procedure of control tests is in line with the contract documents | No | Yes |
| Bid_Manager | •volt-amp-meter | Yes | No |
| Bid_Manager | Date and time of its occurrence; | No | Yes |



| | | |
|---|---|---|
| TND_ 8962 | The installing of ESSU encompasses: | Prose |
| TND_ 8963 | – programme support (software) | DEF |
| TND_ 8964 | – machine part (hardware) in modular performance in euroformat with the aim of easy replacement of defective units, | DEF |

Figure 7.1: Example view of bullet lists in IBM DOORS

### Impact

To improve the classifier's ability to classify such requirements, this context needs to be taken into consideration and presented to the classifier in a way that he understands this context.

Such bullet lists could be detected by some characters, such as a colon, that ends the first line before the bullets, "•" character or ordinals that indicate bullet points. As the examples above show, those requirements originating from bullet list entries often end with a semicolon.

## 7.2.3 Dependent subsystems

During the assessment of the requirements, it turns out that PM is not an independent subsystem.

### Examples

The examples presented in this section origin from the PM test set. The first requirement (see listing 7.1) is a false negative, meaning that the classifier selected it as not PM although it is labeled PM by the requirements manager. It talks about maintenance, but is assigned to project management. This requirement also misses its context, meaning that we do not know which component needs to be maintained (see section 7.2.2).

The second example (see listing 7.2) is a false positive, meaning that it is not assigned to PM by the requirements manager. The requirement clearly talks about temporal constraints (e.g. "on completion", "checked before being put into service", "Date and time") that are project management specific.

Listing 7.1: Example requirement

```
Preventive maintenance
```

Listing 7.2: Example requirement

```
Contractor shall notify the Employer on completion of the works
   , their checks and tests for commissioning (in service). New
    installations are checked before being put into service and
    the beneficiary committee appointed for this purpose (see
   also Annex no. 1). Date and time teaching new facilities
   will be agreed between the Contractor and the Employer.
```

**Explanation**

The reason for both requirements being incorrectly classified is that PM is assigned in case a more specific subsystem does not exist. In the first example, the project the requirement originates from (project 9) is small enough, so that no extra maintenance subsystem is used. PM takes over the responsibilities from this subsystem.

The second requirement is contrary, meaning that a more specific subsystem exists and is therefore selected by the requirements manager instead of PM. In this case, the more specific subsystem is Bid_Manager because the requirement deals with topics that influence the price.

**Impact**

The impact of this finding is that a dependency between subsystems exist. This means that the initial assumption that it is possible to convert the multi-labelling classification problem into multiple binary classifications may be incorrect. To take this information into consideration, a check if a more specific subsystem is available in a project would be needed. In case of absence, PM could be assigned.

### 7.2.4 Generic information

Some requirements at the beginning of an RFT document are assigned to all subsystems.

### Examples

One example requirement can be seen in listing 7.3. This requirement was not classified belonging to Bid_Manager, therefore a FN.

Listing 7.3: Example requirement for general information

```
NOTE 1  The notes and general requirements in this document are
    marked as follows (I).
    M - Mandatory.
    CL - included in the mandatory Conformity List. The
       requirement shall be complied with until the
       commissioning of the systems. It is not mandatory to
       demonstrate the compliance with the requirement in the
       tender.
    I - Informative. In general, notes are informative. If a
       note states conditions for other subsystems than those
       included in the current document, such conditions shall
       be of the type (CL) or (M) for the related documents.
```

### Explanation

The RM assigns such requirements about general information, such as what abbreviations in this document mean, to all subsystems. This person also marks them as DEF, indicating to expect each subsystem to comply to those requirements as a reading confirmation. But s/he claims, that is his or her own interpretation and other requirements managers may proceed differently. Usually such requirements often start with "NOTE".

### Impact

Other requirements managers may assign those requirements as Prose and the subsystems may know that at the beginning of the document contain general information if needed. It is therefore use-case dependent if these requirements should be assigned to all subsystems or none.

CHAPTER 8

# Discussion

This chapter discusses the implications of the results in terms of practical usability. We want to find out, if our approach reduces or increases the requirements manager's work.

In daily business, the requirements manager uses IBM DOORS to manage the requirements and assign them to subsystems. For each assignment, s/he needs to select the intended subsystem(s) from a multi-selection combo box within the tool. We want to find out, how many selections or deselections (in case of false classification) the RM is required to perform when using our approach compared to the current situation.

To answer this question, an average project data set is created representing a hypothetical new example project of the industry partner. In the following sections, we explain how this average project data set is constructed. Based on the results from section 6.4.5, we identify how many (de)selections are required to have all requirements correctly assigned.

We finish this chapter with the implications and impact of these results and discuss the short comings and limitations.

## 8.1 Implications and impact

This section discusses the final performance results of each subsystem from the benchmark (see section 6.4.5) to show its implications on the requirements manager's work in terms of practicality. To do so, the results are discussed upon an average project data set. To support the readability of the results, we report on rounded numbers.

After having critically discussed the results, the impact and comparison of this work to related work is shown.

Table 8.1: Average project data set

| Subsystem | #positive instances | #negative instances |
|---|---|---|
| PM | 656 | 2164 |
| MMI_IXL | 289 | 2531 |
| IXL_HW | 244 | 2576 |
| Bid_Manager | 120 | 2700 |

Table 8.2: Calculated confusion matrices

| | |
|---|---|
| 574 | 82 |
| 308 | 1856 |

(a) Confusion matrix for PM

| | |
|---|---|
| 243 | 46 |
| 451 | 2080 |

(b) Confusion matrix for MMI_IXL

| | |
|---|---|
| 214 | 30 |
| 499 | 2077 |

(c) Confusion matrix for IXL_HW

| | |
|---|---|
| 105 | 15 |
| 820 | 1880 |

(d) Confusion matrix for Bid_Manager

### 8.1.1 Implications of the results on the daily work of the RM

This section constructs a hypothetical data set for an average project and discusses the results of the final models from the benchmark.

Section 5.5 reveals, that an average project consists of 2820 requirements. Therefore, we assume this project to contain that many requirements and having the same class distribution as the data described in section 5.5. This procedure leads to a data set as shown in table 8.1.

Based on the final results presented in section 6.4.5, confusion matrices are now created for the four subsystems PM, MMI_IXL, IXL_HW and Bid_Manager. To do so, recall and specificity is multiplied with the number of positive or negative instances (see table 8.1), respectively, to calculate the amount of TP and TN. FN can be derived by subtracting TP from the number of positive instances. The same is true for FP but on the number of negative instances. This is done for all four subsystems and presented in table 8.2.

For the total of 656 requirements belonging to PM, 574 requirements are correctly detected (TP) but 82 are overseen (FN). From the 2164 requirements not belonging to PM, 1856 are correctly not selected (TN) but 308 are falsely classified as PM (FP). In other words, the classifier saves 574 (TP) selections, but also introduces 308 (FP) required deselections, that where previously not required. The rest of the 82 (FN) requirements need to be detected manually. We now refer to a selection or deselection as action in the requirement management system, which assumes that no multi-(de)selection is used. With the use of the built model, the RM now needs 390 actions, instead of 656, when assigning subsystems completely manually, resulting in 266 saved actions.

Using this same logic on the other subsystems, reveals that this only holds for PM. For

MMI_IXL it requires 497, for IXL_HW 529 and for Bid_Manager 835 actions to correct the classifier. This clearly exceeds the number of positive instances for each of the three subsystems, which are equal to the actions required when using no classifier.

Tough we achieved a saving on user actions for PM, the required actions for the other subsystems increased. These results are in contrast with the interview with the requirements manager. This person states, that deselecting requirements is easier (see section 2.7). To prove practical usefulness and if "easier" proves to be useful, further evaluation is required. Due to limitations shown in section 8.2.4, this evaluation is out of scope for this thesis.

### 8.1.2 Impact and comparison to related work

To the best of the authors' knowledge, the literature review reveals that no one seems to have reported classification results on railways safety equipment requirements originating from request for tender documents. This thesis optimized the configuration for four different classifiers (Random Forest, Naïve Bayes, Support Vector Machines and k-Nearest Neighbors) for one subsystem (PM). For the other three subsystems, we report results for Random Forest. It creates a foundation upon further improvements can be made and further research can be based on.

Furthermore, the interview and the assessment with the requirements manager give more insights in the subsystem assignment process and reveal criteria among the current assignment is done. It shows that the requirements manager uses more information than we do for classification.

Compared to related work, this thesis contributes on empirically evaluating methods proposed in the literature on real life data from an industry partner. The literature focuses mainly on benchmarking publicly available data sets. However, these public sets may not be representative, especially in such a niche domain as requirements for railway safety equipment from RFT documents.

The outcome of this thesis assists the requirements manager in his daily work. It allows to preselect assignments of four subsystems of an arbitrary unknown project.

## 8.2 Limitations and shortcomings

This section deals with the limitations and shortcomings of this thesis. It critically reflects the selection of data sets, the selection of RF as best working classifier, the optimization strategy as well as the missing validation if correcting a classifier is easier than assigning subsystems from scratch.

### 8.2.1 Selection of data sets

We decide to include all projects in the training set for each subsystem even if the project contains no requirement belonging to this subsystem. This naïve approach increases class imbalance towards the negative class. Possibly, better results may be achieved when selecting training data for each subsystem with precaution.

### 8.2.2 Selection of classifier

We selected a classifier for all subsystems based on the results on one subsystem with the least *Skew*. Other classifiers could perform better on the other data sets.

### 8.2.3 Optimization strategy for preprocessing methods

Our optimization strategy to find the most suited configuration is a gradient search like approach in a self-defined order of the stages. To find the most optimized model, we would have to employ hyper-parameter tuning using, e.g., GridSearch or RandomSearch, but due to the required run time we skipped this step.

### 8.2.4 Validation if preselected requirements is helpful

One shortcoming of the thesis is the unproven statement of the requirements manager, that deselection is easier than selection. Section 8.1 reveals that the use of the classifier introduces many deselections to correct the false positives. We thought about some evaluation strategies, but due to the fact, that only one person available with the required knowledge, those results would not be representative.

One straight forward evaluation would be an A/B test. The test user first assigns subsystems without having any preselections. In a second test, this person gets other, but similar, requirements preselected by the classifier and corrects them. The time could be measured and an interview could be held to compare both approaches. But there are many factors that could interfere, such as different requirement complexity, definition of similarity or also different level of concentration of the test user (as the test user may become tired). We therefore left this evaluation out for future work.

CHAPTER 9

# Summary

The overall goal of this thesis was to reduce the work of the requirements manager, when assigning requirements to subsystems. In an interview with the requirements manager it became apparent that the assignment of 800 to 8000 requirements to subsystems is time-consuming and tedious work.

To solve this problem, a machine learning approach was developed which performs these assignments automatically. The models' configurations were optimized during a benchmark. In this benchmark, we compared four different classification algorithms (Random Forest, Naïve Bayes, Support Vector Machines and k-Nearest Neighbors) on one subsystem (PM) and Random Forest on three subsystems (MMI_IXL, IXL_HW and Bid_Manager). We decided to continue only using Random Forest after benchmarking PM because of its performance. The configurations of the models were optimized upon $F_2$ metric in six optimization steps: Sampling, word inflection removal, n-grams, feature weight, stop words removal and feature selection. The performance of the resulting four optimized configurations answered research question one.

These four configurations were further checked upon generalization. For this, a between subsystem analysis was performed where the configurations of one subsystem was used for the other three subsystems. This analysis revealed that PM's configuration leads to highest $F_2$ score, but decreases recall and therefore the model's ability of predicting positive instances which is of importance for this use-case. This is the first partial answer to research question two.

The performance of these four configurations upon arbitrary subsystems were measured on five randomly picked subsystems, one out of each quartile and the median based on class *Skew*. These measures confirm, that PM's configuration lead to highest $F_2$ score among the evaluated configurations, but decreases recall. Further investigations showed, that this is because of common words that are removed in PM's configuration but important for predicting positive instances. This finding completes the answer of research question two.

The results of the initial benchmark optimization were evaluated in a qualitative study. It came out that some requirements need to be treated differently and were therefore falsely classified. The reasons being that requirements that are heading or prose should not be assigned to subsystems, requirements from bullet lists miss important context, some subsystems are dependent (e.g. PM takes over responsibilities if project does not contain a more specific subsystem) and finally some requirements contain general information that are assigned to all subsystems. These reasons answer research question three.

This evaluation along with the limitations and the short comings show some issues that can be resolved in future work:

Firstly, the shortcoming of the evaluation of the preselection of a new project by the classifier could be remedy in future work. The requirements manager states to prefer deselection but we lack information if this statement holds in practice.

Furthermore, it would be interesting if a strategic selection of requirements for the training sets increases performance. The assessment reveals that requirements that represent a heading should not be classified as belonging to a subsystem but the training set contains some instances. Also it would be interesting if using only training requirements from projects where the subsystem is contained improves classification performance.

Another interesting topic arises when it comes to contextualization. Requirements from bullet lists lack information about their context. Future work could evaluate different methods on how to detect this context.

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography

[Agg18] Charu C Aggarwal. *Machine learning for text.* Springer International Publishing, 2018.

[AKG+17] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider. What Works Better? A Study of Classifying Requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 496–501, Sept 2017.

[APA+17] Mehdi Allahyari, Seyed Amin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *CoRR*, abs/1707.02919, 2017.

[BBC+00] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[BDA13] Mohamed Bekkar, Hassiba Kheliouane Djemaa, and Taklit Akrouf Alitouche. Evaluation measures for models assessment over imbalanced datasets. *Iournal Of Information Engineering and Applications*, 3(10), 2013.

[BF15] Cigdem Beyan and Robert Fisher. Classifying imbalanced data sets using similarity based hierarchical decomposition. *Pattern Recognition*, 48(5):1653–1672, 2015.

[BJEA17] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PLOS ONE*, 12(6):1–17, 06 2017.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[BOSB10] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 3121–3124. IEEE, 2010.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[BTR16] Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):31, 2016.

[CBHK02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[CGC10] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology*, 52(4):436–445, 2010.

[CHSZS07] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, Apr 2007.

[CMS01] Maria Fernanda Caropreso, Stan Matwin, and Fabrizio Sebastiani. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. *Text databases and document management: Theory and practice*, 5478:78–102, 2001.

[Cun02] Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[CZLZ16] Kewen Chen, Zuping Zhang, Jun Long, and Hao Zhang. Turning from tf-idf to tf-igm for term weighting in text classification. *Expert Systems with Applications*, 66:245–260, 2016.

[FGR$^+$18] Alessio Ferrari, Gloria Gori, Benedetta Rosadini, Iacopo Trotta, Stefano Bacherini, Alessandro Fantechi, and Stefania Gnesi. Detecting requirements defects with NLP patterns: an industrial experience in the railway domain. *Empirical Software Engineering*, Feb 2018.

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[GZC09] Qiong Gu, Li Zhu, and Zhihua Cai. Evaluation measures of the classification performance of imbalanced data sets. In Zhihua Cai, Zhenhua Li, Zhuo Kang, and Yong Liu, editors, *Computational Intelligence and*

*Intelligent Systems*, pages 461–471, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[HF06] Marcus Hassler and Günther Fliedl. Text preparation through extended tokenization. *WIT Transactions on Information and Communication Technologies*, 37, 2006.

[HKO08] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. Using linguistic knowledge to classify non-functional requirements in srs documents. In Epaminondas Kapetanios, Vijayan Sugumaran, and Myra Spiliopoulou, editors, *Natural Language and Information Systems*, pages 287–298, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[HYS$^+$17] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220–239, 2017.

[Jav18] SnowballStemmer JavaDoc. `http://weka.sourceforge.net/doc.dev/weka/core/stemmers/SnowballStemmer.html`, 2018. Accessed: 2018-11-04.

[JBB15] Alan Jović, Karla Brkić, and Nikola Bogunović. A review of feature selection methods with applications. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, pages 1200–1205. IEEE, 2015.

[JCDLT13] László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. Facing imbalanced data–recommendations for the use of performance metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 245–251. IEEE, 2013.

[JHS02] Li-Ping Jing, Hou-Kuan Huang, and Hong-Bo Shi. Improved feature selection approach tfidf in text mining. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, pages 944–946. IEEE, 2002.

[Jiv11] Anjali Ganesh et al. Jivani. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[JL95] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[KM12] Vandana Korde and C Namrata Mahender. Text classification and classifiers: A survey. *International Journal of Artificial Intelligence & Applications*, 3(2):85, 2012.

[KM17] Zijad Kurtanović and Walid Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *Requirements Engineering Conference (RE), 2017 IEEE 25th International*, pages 490–495. IEEE, 2017.

[KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[LFG$^+$13] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013.

[LGMTCOGB16] Octavio Loyola-González, José Fco Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa, and Milton García-Borroto. Study of the impact of resampling methods for contrast pattern based classifiers in imbalanced databases. *Neurocomputing*, 175:935–947, 2016.

[LS05] Ola Løkberg and Øystein Skogstad. Functional apportioning of safety requirements on railway signalling systems. In Rune Winther, Bjørn Axel Gran, and Gustav Dahll, editors, *Computer Safety, Reliability, and Security*, pages 289–302, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[OK09] Tim O'Keefe and Irena Koprinska. Feature selection and weighting methods in sentiment analysis. In *Proceedings of the 14th Australasian document computing symposium, Sydney*, pages 67–74. Citeseer, 2009.

[Ott13] Daniel Ott. Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 50–64. Springer, 2013.

[PEM03] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 308–313. IEEE, 2003.

[PF$^+$97] Foster J Provost, Tom Fawcett, et al. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *KDD*, volume 97, pages 43–48, 1997.

[PG12] Pratiksha Y Pawar and SH Gawande. A comparative study on different types of approaches to text categorization. *International Journal of Machine Learning and Computing*, 2(4):423, 2012.

[Pow11] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies, 2(1), 37-63*, 2011.

[PR07] Jan Pomikálek and Radium Rehurek. The influence of preprocessing parameters on text categorization. *International Journal of Applied Science, Engineering and Technology*, 1:430–434, 2007.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[SL09] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[SMR08] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, 2008.

[TB13] Sri Fatimah Tjong and Daniel M Berry. The design of SREE—a prototype potential ambiguity finder for requirements specifications and lessons learned. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 80–95. Springer, 2013.

[Tin10] Kai Ming Ting. *Confusion Matrix*, pages 209–209. Springer US, Boston, MA, 2010.

[WIZ15] Sholom M Weiss, Nitin Indurkhya, and Tong Zhang. *Fundamentals of predictive text mining*. Springer, 2015.

[WV16] Jonas Winkler and Andreas Vogelsang. Automatic classification of requirements based on convolutional neural networks. In *Requirements Engineering Conference Workshops (REW), IEEE International*, pages 39–45. IEEE, 2016.

[WW17] I. H. (Ian H.) Witten and Ian Witten. *Data mining : practical machine learning tools and techniques*. Elsevier, Amsterdam, [Netherlands], fourth edition. edition, 2017.

[YHX$^+$16] Li Yijing, Guo Haixiang, Liu Xiao, Li Yanan, and Li Jinling. Adapted ensemble classification algorithm based on multiple classifier system and feature selection for classifying multi-class imbalanced data. *Knowledge-Based Systems*, 94:88–104, 2016.

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR* *conference on Research and development in information retrieval*, pages 42–49. ACM, 1999.

[YP97] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, pages 412–420, 1997.