TU WIEN Informatics

Binäre Textklassifizierung unter Verwendung positiver und unbeschrifteter Daten in der Anwendung auf Twitter

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

David Hinterndorfer, BSc
Matrikelnummer 01526409

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung:
Univ.Prof. Dr. Allan Hanbury,
Univ.Prof. Dipl.Inform.Univ. Dr. Claudia Plant

Wien, 22. Juni 2020

David Hinterndorfer Allan Hanbury

Informatics

Binary Text Classification Using Positive and Unlabeled Data in Application to Twitter

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

David Hinterndorfer, BSc
Registration Number 01526409

to the Faculty of Informatics

at the TU Wien

Advisor:
Univ.Prof. Dr. Allan Hanbury,
Univ.Prof. Dipl.Inform.Univ. Dr. Claudia Plant

Vienna, 22nd June, 2020

David Hinterndorfer   Allan Hanbury

Erklärung zur Verfassung der Arbeit

David Hinterndorfer, BSc

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, 22. Juni 2020

David Hinterndorfer

Acknowledgements

I would like to thank Markus Tretzmüller, Michael Trimmel and Alexander Steiner for their great support in the course of the master thesis. Especially Markus, who helped me to stay on track and gave his continued feedback. I would also like to thank my advisors Prof. Allan Hanbury and Prof. Claudia Plant for their feedback and their expertise.

Last but not least, I would like to thank my parents, friends and my girlfriend Theresa who supported me and kept me focused on the work.

Kurzfassung

Binäre Textklassifizierung unter Verwendung positiver und unbeschrifteter Daten, auch oft unter dem Namen »Positive-Unlabeled (PU) Learning« zu finden, ist ein gut erforschtes Gebiet im Bereich »Machine Learning (ML)«. Das Problem ist, dass die meisten vorgestellten Ansätze nur in einer sehr generischen, experimentellen Umgebung getestet wurden und deshalb deren Eignung in spezifischen Domänen, wie Twitter, unklar ist.

Aus diesem Grund ist das Ziel dieser Arbeit die Evaluierung existierender Ansätze und die anschließende Entwicklung eines »PU Learning« Ansatzes, in Bezug auf die Domäne Twitter.

Um dies zu erreichen, verwenden wir eine explorative Strategie, um einen neuen, auf Twitter optimierten, »PU Learning« Ansatz zu entwickeln. Die Optimierungen bestehen zum einen aus der Ermittlung, Analyse, Einpflegung und Evaluierung von domänen-spezifischen Eigenschaften. Zum anderen, beinhalten diese die Kombination dieser Eigenschaften gemeinsam mit den bestgeeignetsten Algorithmen innerhalb eines neuen »PU Learning« Ansatzes. Das Resultat ist ein 2-stufiger Ansatz, mit dem Namen TWLR-SVM welcher »Weighted Logistic Regression« in der ersten Stufe und »SVM« in der zweiten Stufe verwendet. Die Evaluierung und Verifizierung der verschiedenen Entwicklungsstufen, wurde durch wiederholte, quantitative Experimente durchgeführt. Die Experimente betrachteten dabei immer die zwei Anwendungsfälle »Topic Detection« und »Spam Detection« in der Domäne Twitter.

Die Ergebnisse zeigen das unser Ansatz, in verschiedensten Szenarien und Anwendungsfällen in der Domäne Twitter, sehr genaue Vorhersagen liefert und alle bisher in der Literatur vorgestellten Ansätze übertrifft. Um genau zu sein übertrifft unser Ansatz, TWLR-SVM, alle in der Literatur vorgestellten Ansätze in Hinblick auf durchschnittlich erzielten F1-Score um 6.37 Prozentpunkte bei »Topic Detection« und 4.38 Prozentpunkte bei »Spam Detection«.

Abstract

Binary text classification using only positive and unlabeled examples often found under the term positive-unlabeled (PU) learning, is a well-studied area in machine learning (ML). The problem is that most of the proposed techniques were only evaluated in a very generic experimental setup. Therefore, the suitability of most of these approaches for specific domains, such as Twitter, is unclear.

For that reason, this work looks at PU learning problems related to the domain of Twitter. The aim is to evaluate existing approaches in the Twitter domain and consequently investigate and develop a PU learning approach, which is optimized to the Twitter domain.

Based on an explorative strategy, we investigated and developed a new domain-optimized PU approach. The optimizations include, on the one hand, feature engineering in the form of investigation, analysis, incorporation and evaluation of domain-specific features. On the other hand, they contain the combination of those features alongside the best-suited algorithms in a new PU approach. The result is a new 2-step approach, called TWLR-SVM, which uses a domain-optimized version of Weighted Logistic Regression in the first step and Support Vector Machine (SVM) in the second step. To evaluate and verify the different optimizations, we conducted recurring quantitative experiments and analyzed their results. The conducted experiments always considered two use cases, namely topic detection and spam detection, in the domain of Twitter.

The evaluation results indicate that our approach performs very well, delivers very precise predictions and outperforms all existing PU learning approaches, represented in the literature, on different scenarios and use cases in the Twitter domain. To be more precise, it is shown that our approach, TWLR-SVM, outperforms the existing PU approaches in the Twitter domain in terms of average F1-Score by 6.37 percentage points on topic detection and 4.38 percentage points on spam detection.

Contents

CHAPTER 1

Introduction

Text classification also known under the terms document classification or text categorization describes the automated process of assigning texts to predefined classes or categories. Binary text classification is a sub-area of text classification, where the classifiers only distinguish between two classes, therefore it assigns a text to one of two classes. In the literature, those two classes are often called "positive" and "negative" class. For example, the detection of spam in emails can be interpreted as a binary text classification task, where the positive class should be assigned to the mails which are spam and the negative class should be assigned to all other mails or vice versa. Normally such classifiers are built by passing a high amount of labeled examples from each class to a learning algorithm, which subsequently is able to classify future examples based on the words and features they contain (supervised learning). This means in binary text classification a high amount of examples of the positive and of the negative class are needed.

In practice, this is a problem very often because most of the time the available training examples are either incompletely labeled (semi-supervised learning), where labeled examples are not available for each class, or not labeled at all (unsupervised learning). In one special case of this problem, which can be found in practice very often (e.g. in medical diagnosis, knowledge base completion, advertisement), we only have an incomplete set of examples of the positive class, and another set of unlabeled examples, which can contain positive examples as well as negative examples. The building of a binary classifier in such a scenario, where only labeled positive and unlabeled examples are available can be found in the literature under the term positive-unlabeled (PU) learning. This term first appeared in the early 2000s and since then a variety of techniques, approaches and algorithms with their advantages and disadvantages were proposed. [49, 28, 9, 24, 29, 21, 11, 27]

This introduces the major problem, which has to be dealt with in this work. Namely, most of the proposed techniques were only evaluated in a very generic experiment setup, therefore the suitability of most of these approaches for specific domains is unclear.

Additionally, the absence of publicly available implementations of the algorithms to evaluate the techniques in specific domains, is a problem.

This work looks at PU learning in the Twitter domain. Twitter is by definition a social network, where users (persons of public interest, journalists, news agencies, private persons, etc.) can connect with each other and post and share content in the form of a maximum of 280 characters long text documents, called tweets. Twitter provides access to its public conversational data for academic research. Through the vast amount of data made available, this domain is a very popular area of research. The data also offers a lot of room for investigation because of its domain-specific characteristics and features. Among others, these domain-specific characteristics and features include the restricted length of the tweets, the very considerable difference in the quality of tweets (slang, dialects, spam, etc.) and the network features. There exists a lot of research about modifying existing approaches [46, 36] or applying feature engineering to optimize existing approaches [38, 6, 17, 45] in this domain. The absence of research dealing with PU learning in the Twitter domain as well as the popularity of Twitter as a research domain is the reason we look at PU learning in the Twitter domain in this work.

One problem or use case in the Twitter domain that we look in this work is strongly related to a general problem of Boolean information retrieval and arises when searching for tweets related to a specific topic (e.g. cryptocurrencies) over the Twitter API. Since the Twitter API only allows keyword-based filtering, when filtering by keywords that are not unique to a specific topic (e.g. the keyword "tron" can be related to the cryptocurrency or the movie), tweets of multiple topics are found. This means that an unlabeled set containing tweets of the desired topic (positive class) as well as tweets of other topics (negative class), is returned. However, an incomplete set containing only tweets of the desired topic (positive class) can be constructed, by searching for keywords that are unique to the specific topic (e.g. bitcoin). Through the possibility of constructing a labeled set containing only positive examples and the availability of unlabeled examples, everything is given to consider this as a valid PU learning problem.

Another problem or use case in the Twitter domain that we look in this work is the detection of spam tweets. This can be interpreted as a PU learning problem by considering low-quality tweets as the positive class and a set of quality tweets (negative class) combined with low-quality tweets as an unlabeled set. By using a labeled spam detection set from the literature [6], which contains examples for low-quality tweets as well as quality tweets, we can construct a positive labeled set and an unlabeled set and therefore this also can be considered as valid PU learning problem.

1.1 Aim of the Work

On the one hand, the goal of this research project is to evaluate existing PU learning approaches in the domain of Twitter since no research and no evaluation results in the domain-specific setting are available. On the other hand, the aim is to investigate and develop a PU learning algorithm, which is suitable and optimized for different scenarios and use cases in the Twitter domain. Thus, the work also deals with the challenges which arise through the domain-specific formulation of the problem, such as the constrained length of the tweets (short text documents), the very considerable difference in the quality of tweets (slang, dialects, spam, etc.) and the incorporation of network features.

1.2 Contributions of the Work

As already mentioned, PU learning is a well-studied area and a variety of techniques, approaches and algorithms were proposed. [49, 28, 9, 24, 29, 21, 11, 27]
However, most of the proposed approaches were only evaluated in a very generic experiment setup, therefore the suitability of most of these approaches for specific domains is unclear. For that reason, we evaluate the suitability and performance of existing PU approaches in different scenarios and use cases in the domain of Twitter in this work. Based on the gathered knowledge, we furthermore investigate and develop a new PU approach including domain-specific optimizations in the form of algorithm selection and feature engineering. Through that, this work accomplishes the following contributions:

- **Evaluation Results and Baseline Metrics**
  As already mentioned there does not exist any research about PU learning in the Twitter domain. For that reason, we re-implement a variety of PU approaches and evaluate them on different use cases (topic detection, spam detection) in the Twitter domain. Through that, we explore baseline metrics and see how well PU learning performs in the Twitter domain. In other words, this work contributes evaluation results and baseline metrics as well as insights about the performance of PU learning in domain-specific settings.

- **Extensive Feature Engineering**
  Besides the evaluation of the existing approaches, we develop a domain-optimized PU approach. To optimize a PU approach or rather to optimize our PU approach we conduct extensive feature engineering. This includes the investigation, analysis, evaluation, selection and incorporation of domain-specific features. Through that, this work contributes insights about the performance of different domain-specific features, which best work in the Twitter domain, or more precisely in the use cases topic detection and spam detection in the Twitter domain.

- **Domain-Specific PU Learning Approach - TWLR-SVM**
  As briefly mentioned, we investigate and develop a domain-optimized PU approach in this work. This PU approach is called TWLR-SVM and combines the best-suited methods and features for different use cases (topic detection, spam detection) in the Twitter domain. This contributes a flexible PU approach which can be used, modified and extended for different use cases in the Twitter domain.

1.3 Synopsis

In Chapter 2, we present the domain Twitter, including the terminology as well as literature. Furthermore, we describe the theoretical foundations and basic concepts of (binary) text classification and positive-unlabeled (PU) learning and present existing approaches and literature in the field of (binary) text classification and PU learning. First, we show the relevance of the domain Twitter in research. Then, we define a taxonomy, to understand where to place the field of PU learning. Afterward, the concepts and existing algorithms of text classification and PU learning are presented and described in further detail.

Afterward, in Chapter 3, we explain the research methods and the methodology used for this study in detail. Through that, the study should be made more comprehensible and reproducible. Also, it shows and justifies why we chose the respective research methodology for this research. As a result, the reliability and validity of this study are strengthened.

In Chapter 4, we present our approach, called Twitter Weighted Logistic Regression-Support Vector Machine or short TWLR-SVM. It is a modular and hybrid PU learning approach optimized to the Twitter domain. The optimizations include, on the one hand, feature engineering in the form of investigation, analysis, incorporation and evaluation of domain-specific features. On the other hand, they contain the combination of those features alongside the best-suited algorithms in a new PU approach. To be more clear, it is a 2-step approach using an optimized version of the biased learning approach, Weighted Logistic Regression, in the first stage and a SVM classifier in the second stage. Besides the approach, we introduce assumptions that have to hold for our approach to work. Furthermore, we describe the features we extracted and how we incorporated them to optimize our approach for the Twitter domain.

We describe, in Chapter 5, the design of the experiments we conducted. Firstly, this includes how the datasets we used for training and evaluation are created. Secondly, the prototypes we implemented and how we verified their correctness. Finally, the evaluation metrics used to analyze and compare the performance of the approaches as well as the experiment framework we built to conduct the experiments and collect the results.

Then, in Chapter 6, we present, analyze and discuss the results of the experiments we conducted. This includes the verification of the prototypes as well as the subsequent exploration of baseline metrics. But also, the results of the experiments analyzing the different optimization steps of our approach, including the analysis of TWLR and TWLR-SVM. It is shown that from the re-implemented existing PU approaches, Weighted Logistic Regression performs the best on the Twitter domain, achieving average F1-scores of 90.26% on topic detection and 80.56% on spam detection. Our approach, TWLR-SVM, which combines domain-specific features with the best-suited algorithms, on the other hand, achieves average F1-scores of 96.63% on topic detection and 84.94% on spam detection. This means it is shown that our approach, TWLR-SVM, outperforms the existing PU approaches in the Twitter domain in terms of average F1-Score by 6.37 percentage points on topic detection and 4.38 percentage points on spam detection.

Afterward, in Chapter 7 we summarize and reflect on the research. This contains the explicit answering of the research questions as well as the new knowledge contributed by this thesis.

Finally, in Chapter 8 we present possible directions for future research, like further investigating and optimizing our approach (e.g. by applying ensemble learning to the approach or investigating other features) or looking at other assumptions (e.g. selected at random) and use cases.

1.4 Notation

Table 1.1 shows the notation used in this work.

Symbol	Description
D	Entire document set
P	Labeled positive set
U	Unlabeled set
N	Labeled negative set
RN	Reliable negative set
RP	Reliable positive set
X	Matrix of input values
x	Input data
y	Output value (label)
$\vec{y}$	Vector of output values (labels)
s	Indicator for an example to be labeled
α	Class prior $\alpha = P(y = 1)$
c	Label frequency $c = P(s = 1\|y = 1)$
n	Number of documents in the document set $n = \|D\|$
t	Term variable
d	Document variable
$freq(t, d)$	Raw count of term t in document d
$tf(t, d)$	Term frequency (TF) of term t in document d
$idf(t, D)$	Inverse document frequency (IDF) of term t in document set D
$df(t, D)$	Document Frequency of term t in document set D
$tfidf(t, d, D)$	TF-IDF of term t in document d and document set D

Table 1.1: Notation used in this work

CHAPTER 2

Theory and Related Work

In this chapter, we present the domain Twitter, including the terminology as well as literature. Furthermore, we describe the theoretical foundations and basic concepts of (binary) text classification and positive-unlabeled (PU) learning and present existing approaches and literature in the field of (binary) text classification and PU learning.

First we present the domain Twitter and show its relevance in research, in Section 2.1. Since text classification, or more precisely PU learning touches a multitude of fields, we then define a taxonomy, which shows where to place the topic of (binary) text classification and PU learning, in Section 2.2. As PU learning is a sub-area, or rather a special case, of text classification and the algorithms used in PU learning are often modified or extended versions of the text classification algorithms, we than describe the basic concepts of text classification in more detail and show existing text classification algorithms and their corresponding literature, in Section 2.3. After that, in Section 2.4, we also discuss the theoretical foundations and basic concepts of PU learning in detail and show proposed PU learning approaches and their related literature. Since there exists a large variety of approaches, we categorize the approaches and indicate just the primal and often cited ones. At the end, we describe in Section 2.5 and Section 2.6 the necessary steps to prepare the data and extract features from it, because this is required for text classification and consequently PU learning. This contains preprocessing methods like tokenization, stop word removal and lemmatization as well as feature extraction methods like TF-IDF.

2.1 Twitter

By definition Twitter is a social network, where users can connect with each other and post and share content in the form of maximum 280 characters long text documents, called tweets. Twitter users can be for example people of public interest or private persons, but also companies, groups and many more. Sharing a tweet of another Twitter user is called retweeting. Connections on Twitter are made by following another user, mentioning another user in a tweet or retweeting the tweet of another user. Twitter provides tools to annotate content with so called hashtags (#) and symbols, often called cashtags ($). While hashtags are designed for overall categorization, cashtags are meant to denote specific assets and financial instruments.

Twitter provides access to its public conversational data for academic research. Through the vast amount of data made available, this domain is a very popular area of research. The data also offers a lot of room for investigation because of its domain-specific characteristics and features. Among others, these domain-specific characteristics and features include the restricted length of the tweets, the very considerable difference in the quality of tweets (slang, dialects, spam, etc.) and the network features.

There exists a lot of research about modifying existing approaches [46, 36] or applying feature engineering to optimize existing approaches [38, 6, 17, 45] in the Twitter domain.

2.2 Taxonomy

Text classification, or more precisely PU learning is touching a variety of fields including artifical intelligence (AI), machine learning (ML), deep learning (DL) and natural language processing (NLP). Figure 2.1 shows the relation between these fields and where the topic text classification is placed. In the upcoming sections, we describe the fields and their relation to each other.

2.2.1 Artificial Intelligence

Artificial intelligence (AI) is a wide-ranging topic in computer science and looks at the building of systems, often called intelligent agents. These agents are built to execute tasks, where usually human or natural intelligence is required. This is done by using algorithms that can draw predictions and make decisions by analyzing and learning from data. One characteristic of such systems is that it learns and improves over time. This is also the major difference to rule based programs because in rule based programs all scenarios have to be defined and the program only operates in these defined scenarios.
In the last decade, artificial intelligence was applied to a variety of fields like Image Recognition, Speech Recognition, Intelligent Robots, Gaming or Natural Language Processing. [1, 40, 41]

2.2.2 Machine Learning

Machine learning (ML) is a sub-field of artifical intelligence. In ML a computer learns from historical data (past experience) to draw predictions and make decisions on future data based on the historical data without being explicitly programmed. This is achieved by using statistical models and algorithms which can recognize and detect patterns in data as well as infer further knowledge from the data. The more data is known to the model, the more patterns can be recognized and knowledge can be inferred, which

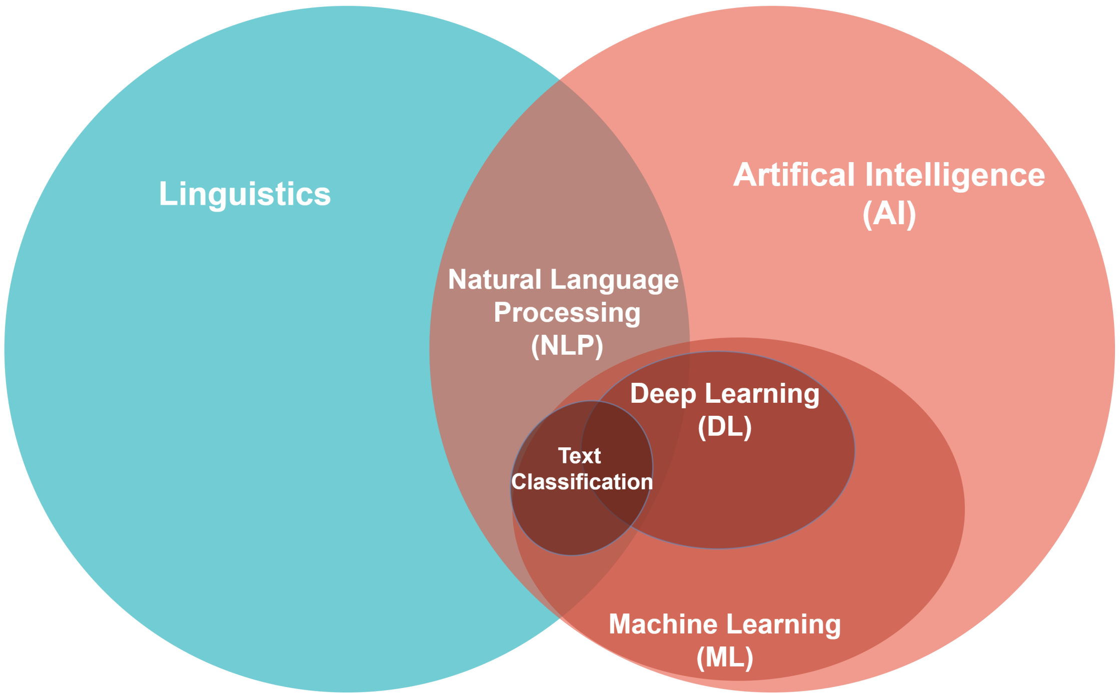


Figure 2.1: Venn-Diagram showing the relationship between fields of linguistics, artifical intelligence (AI), machine learning (ML), deep learning (DL), natural language processing (NLP) and text classification.

consequently leads to better accuracy on predictions and decisions on future data. In the literature, ML is often divided into the four categories: supervised machine learning, unsupervised machine learning, semi-supervised machine learning and reinforcement learning. [1, 40, 41]

In the upcoming sections, we just describe supervised machine learning, unsupervised machine learning and semi-supervised machine learning, since these are essential to understand PU learning and especially our work. Reinforcement learning was just mentioned for the sake of completeness, but will not be described any further since it is not essential for this work.

Supervised Learning

In supervised learning, labeled data is available to train a model. This means we have a set of historical examples, where we know the desired output (often called class or label). In other words, we have a training set of input-output pairs, which is used to train a model. The aim is to make decisions about the output value (label) for future input data. To allow an algorithm to predict the output to a given input, a function is inferred by analyzing a training set of labeled examples. Equation 2.1 shows the mathematical definition of this function, where X is a matrix of input values and y is a vector of output values.

$$f(X) = \vec{y} \tag{2.1}$$

In supervised learning, the training set has to contain all possible, desired labels. Labels that are not contained in the training set are unknown to the algorithm and the inferred function. Supervised learning is often used in classification and anomaly detection tasks.

Unsupervised Learning

In unsupervised learning, unlabeled data is available and used to train a model. This means in comparison to supervised learning, in unsupervised learning a training set containing historical data without any desired output is used to learn. Since no information about the outputs is available, unsupervised learning models search for hidden patterns and similarities in the input data. Through that, the model can build clusters and make decisions about the class of future data. Unsupervised learning is often used for clustering tasks.

Semi-Supervised Learning

As well supervised learning as unsupervised learning have their disadvantages and problems. The major problem in supervised learning is that the labeling of data is most of the time a costly process, and often unfeasible, especially when dealing with large volumes of data. In unsupervised learning, on the other hand, a problem is that the result is hard to control and therefore it is not suitable for many use cases, also the models often suffer from a low accuracy. For that reason, another research direction, called semi-supervised learning, emerged. Semi-supervised learning is a combination of supervised learning and unsupervised learning, where both labeled examples and unlabeled examples are used to learn. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabeled data.

2.2.3 Deep Learning

Deep learning (DL) is, again a sub-field of machine learning and artifical intelligence and is loosely inspired by the human thinking and the structure of the human brain. The field of deep learning has received a lot of attention over the past decade and many different approaches and structures to solve a variety of use cases were proposed. We briefly present the basic intuition and the structures used in deep learning in Section 2.3.2. However, deep learning is not essential in this work and we just mentioned it for the sake of completeness.

2.2.4 Natural Language Processing

Linguistics is the study of language containing the analysis and understanding of language. Natural language processing (NLP) is the automatic processing and handling of the human language (speech or text) and therefore lies in between the fields artifical intelligence and linguistics. The problem for the machine in processing and understanding the human language is that it is ambiguous, imprecise and unstructured. NLP is applied to many different fields like sentiment analysis, text classification, speech recognition or language translation.

2.2.5 (Binary) Text Classification and PU learning

Text classification is part of the field of machine learning and natural language processing, since mathematical models are trained from data, which find patterns in a text to assign the text to a specific class. As shown in Figure 2.2, binary text classification is a sub-area of text classification and PU learning is again a sub-area, or rather a special case of binary text classification. Since text classification, or rather PU learning is the main topic of this thesis we describe these fields in Section 2.3 and Section 2.4 in more detail.

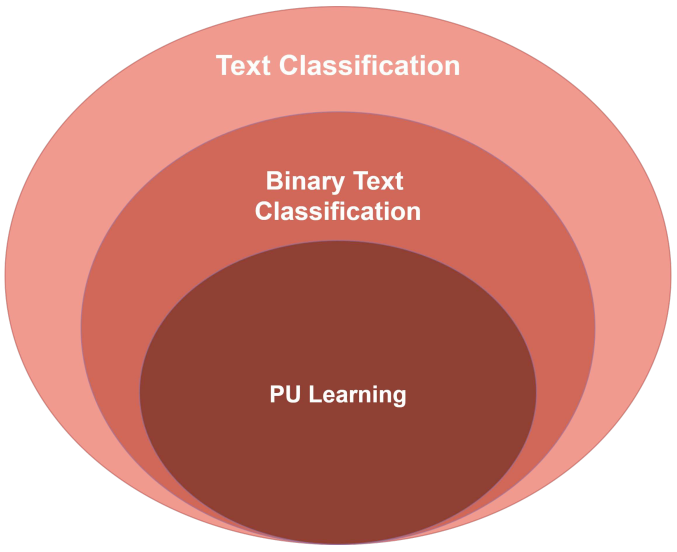


Figure 2.2: Venn-Diagram showing the relationship between fields of text classification, binary text classification and PU learning.

2.3 Text Classification

Text classification describes the automated process of assigning texts to predefined classes or categories. As shown in Figure 2.3, this is usually separated into a training phase and a prediction phase. In the training phase, a set of text documents and their respective labels (labeled training set) is used to train a text classification algorithm and build a trained model (supervised learning). After that, in the prediction phase, the model can predict labels for new text documents. Text documents are unstructured data and therefore can not simply be interpreted by the computer. Consequently, we have to preprocess the data and extract features, which can be used and interpreted by the machine. In Section 2.5 and Section 2.6 we go into detail about the preprocessing and the feature extraction to prepare the data for the text classification algorithm.

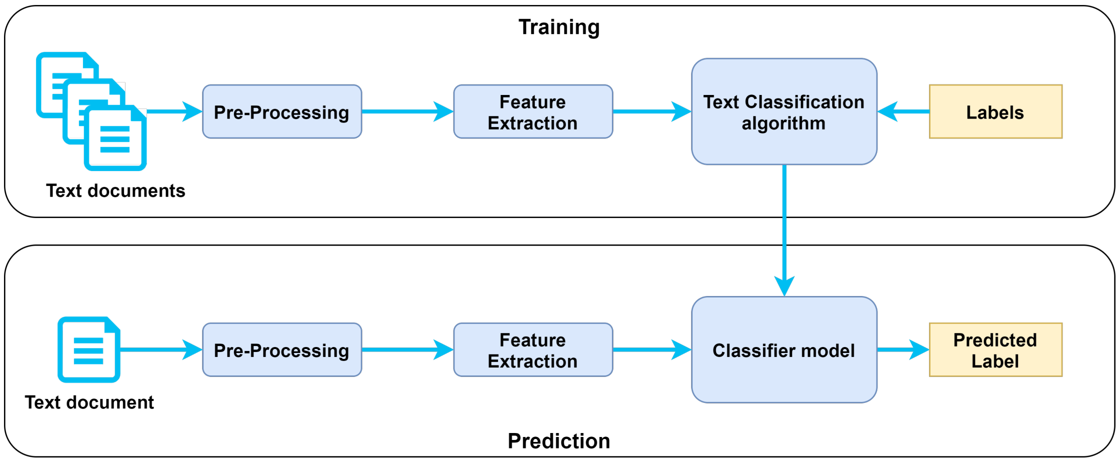


Figure 2.3: Schematic illustration showing the process of text classification, split into a training phase, where an algorithm is trained on preprocessed text documents and their corresponding label, and a prediction phase, where the resulting classifier model is used to predict the labels for new text documents.

2.3.1 Binary Text Classification

As already mentioned, binary text classification is a sub-area of text classification, where text documents get assigned exactly one of two classes. The process in binary text classification is the same as the process in text classification shown in Figure 2.3. At first, a classifier model is built by training a text classification algorithm using labeled data. Afterward, the resulting model is able to make predictions on future input data.
The difference to text classification is that instead of using multiple labels, in binary text classification just two labels, often called positive ($y = 1$) and negative ($y = 0$) class, are used to train the model. Through that, the model just knows these two classes and therefore distinguishes only between these two classes. An example of a dataset in binary text classification is shown in Table 2.1, with x being the input features and y being the corresponding class or label. For simplicity, the examples in the table represent the input features as text. Normally this text is transformed into a numeric vector using a feature extraction technique, which we describe in Section 2.6.

2.3.2 Algorithms

Text classification is a well-studied field in machine learning and a high number of approaches and techniques were proposed in the past. Caruana et al. [4] give a good overview of existing machine learning algorithms, which are used in text classification and empirically compares them. In the following paragraphs, we introduce established

x	y
Free Bitcoins!!! Here -> bit.ly/eePon3	1
You Won 1.000.000 euro: bit.ly/ccPnn	1
Good morning #twitter.	0
I bought a Tesla today!	0

Table 2.1: Snippet of a labeled dataset for spam detection in binary text classification, containing input data (x) and its corresponding binary label (y). For simplicity, the input data x is represented as text. Normally this text is transformed into a numeric vector before it is handed over to the machine learning algorithm.

algorithms used for the classification of text. As we describe later, most of the algorithms introduced in the following sections, can be found and are used in PU learning approaches in some modified or extended way.

We only describe the algorithms Logistic Regression and Support Vector Machine (SVM) in more detail, since we use these two algorithms, or rather a modified version of them, in our approach. For all other algorithms, we just briefly describe their functionality, since a detailed explanation of each would be out of scope.

Naive Bayes (NB)

Naive Bayes (NB) classifiers are probabilistic, generative models and the technique was first proposed for text classification tasks by Maron et al. [34] in 1961. As the name suggests the classifier makes use of Bayes' Theorem, which addresses conditional probability. In other words, the likeliness of an event given that another event already happened. The Theorem is presented in Equation 2.2, where the result $P(A|B)$ describes the probability of A being true, given that B is true. It is calculated using $P(B|A)$, $P(A)$ and $P(B)$. The first describes the probability of B being true, given that A is true. The second describes the probability of A being true and the third describes the probability of B being true.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{2.2}$$

Even though the algorithm is one of the earlier ones, it is due to its performance, simplicity and fast computation still a very popular method, often used as a baseline method in text classification.

Rocchio (Roc)

The Rocchio (Roc) algorithm was proposed by J.J. Rocchio [43] and is based on a method for relevance feedback in information retrieval systems. In this algorithm, the features of the text documents are represented as a vector in a vector space. To classify text documents, first, the algorithm builds so-called prototype vectors for each class. This is done by calculating average vectors using a training set of documents. Afterward, the algorithm can classify documents by comparing the similarity of the respective document vector to the different prototype vectors.

k-Nearest Neighbor (kNN)

The k-Nearest Neighbor (kNN) algorithm is a very fundamental and intuitive classification method. Like most of the algorithms in this list, it was applied to a multitude of domains in the past and many different modifications and improvements to the algorithm exist. [2, 16, 44]. Like in the Rocchio algorithm, in kNN the features of a text document are represented as a vector in a vector space. To classify a document the algorithm searches for the k nearest documents of the training set in the vector space, using some distance function. After that, the algorithm checks, which class occurs the most in the found documents and assigns the respective class to the document. Since the algorithm is non-parametric it is a good choice where no prior knowledge about the data distribution exists.

Decision Tree

Creating and using decision trees to classify text was introduced by Magerman et al. [33]. A decision tree is a tree which contains nodes representing conditions (root node and decision nodes) and nodes representing decisions over the class (terminal nodes) The algorithm uses the features of the documents of the training set to form conditions and build a decision tree, which is then used to decide about the class. When classifying a text document the features are used to check, which conditions in the decision tree are met. Consequently, a path is traversed through the decision tree, which ends at a terminal node, which represents the decision over the class. The advantages of decision trees are that they are easy to understand, interpret and visualize as well as the fast execution speed.

Bagging

Bagging, also called bootstrap aggregating, is an ensemble learning meta-algorithm, which is applied to machine learning algorithms to improve their robustness and stability. In Bagging the training set is separated into multiple parts (bags) using sampling. Afterward, multiple models are trained. When classifying a new text document, the trained models vote about the class. The algorithm is often applied to the Decision Tree algorithm, but it can be used with any type of method, as we will show later.

Random Forest

Because of the intuitiveness of decision trees, the algorithm is often used in bagging algorithms. Ho et al. [12] proposed an extended version of the decision tree algorithm, or rather the bagging version of the decision tree algorithm. The bootstrapping of the training set into so-called bags using sampling and the building of multiple classifiers using these bags as well as the voting about the classification of a text document is the same as in the traditional bagging approach. The difference is that the decision tree models are built using only a random subset of features in the bags.

Support Vector Machine (SVM)

The Support Vector Machine (SVM) algorithm was proposed by Cortes et al. [8] and was designed for binary classification tasks. Like in many of the algorithms, in this algorithm as well, the input or rather the features of the input are represented as a vector in an n-dimensional vector space, where n represents the number of features. Given a set of training data, the SVM algorithm, than searches for an optimal hyperplane that differentiates the training data into their respective classes in the best possible manner.

A hyperplane can be defined as shown in Equation 2.3, with w being a weight vector, x being the input vector and b being a bias term.

$$w^T x + b = 0 \tag{2.3}$$

This definition (Equation 2.3) is dimension invariant and therefore can be equally applied to low-dimensional spaces as well as high-dimensional spaces. Thus, using this definition and considering only two features ($x = \{x_1, x_2\}$), the hyperplane is represented and can be visualized as a line, as shown in Figure 2.4.

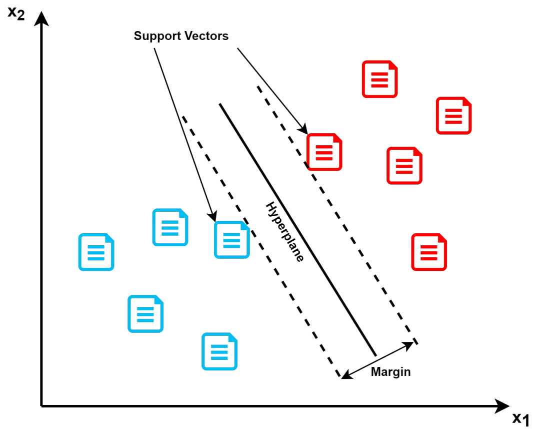


Figure 2.4: Schematic illustration of a 2-dimensional vector space in Support Vector Machine (SVM) considering the features x_1 (x-axis) and x_2 (y-axis). SVM searches the hyperplane, which best separates the text documents and maximizes the margin by using the support vectors.

Figure 2.4 also shows the intuition about how the optimal hyperplane is found. This is done using so-called support vectors, which are the critical input vectors, which are nearest to the hyperplane. In fact, the algorithm maximizes the so-called margin, which represents the distance between the hyperplane and the support vectors, by transforming and shifting the hyperplane by modifying the weight vector.

After finding the optimal hyperplane the classifier is able to predict the label for a given input using a hypothesis function defined as shown in Equation 2.4.

$$h(x) = \begin{cases} 1, & \text{if } w * x + b \geq 0. \\ 0, & \text{if } w * x + b < 0. \end{cases} \tag{2.4}$$

As already mentioned, SVM was designed especially for binary classification and therefore it performs very well on binary text classification tasks. Consequently, it is also, as we show later in Section 2.4, very often used in PU learning approaches.

Logistic Regression

Contrary to its name, Logistic Regression is a probabilistic, linear classification algorithm, which can be used for binary classification tasks. [15, 10]

Logistic Regression is very similar to Linear Regression since both use a linear expression as a basis. This linear expression can be defined as shown in Equation 2.5, with w being a weight vector, x being the input vector and b being a bias term.

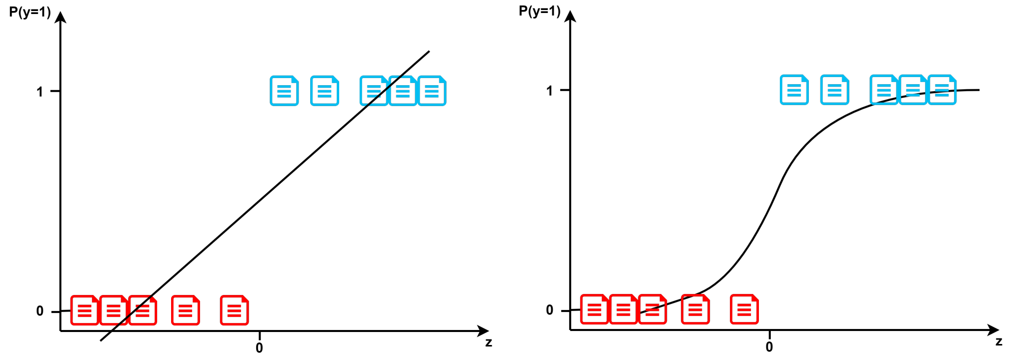
$$z = w^T x + b \tag{2.5}$$

As illustrated in Figure 2.5a, in Linear Regression, the linear expression (Equation 2.5) is directly used to fit a line to the data. In Logistic Regression, on the other hand, the linear expression (Equation 2.5) is passed to a logistic function, called sigmoid function σ. This sigmoid function σ can be defined as shown in Equation 2.6.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

In other words, an S-shaped logistic function, called sigmoid function σ, is fit to the data. This S-shaped curve can be illustrated in the vector space as shown in Figure 2.5b. Given some input vector x, the x-axes of the two illustrations in Figure 2.5 depict the output z of the linear expression (Equation 2.5) and the y-axes depict the probability of being of some class (e.g. spam).

One big advantage of Logistic Regression compared to Linear Regression is that the output of the sigmoid function σ can directly be used as the probability of the input being of some class (e.g. spam), as shown in Equation 2.7.



(a) Linear Regression (b) Logistic Regression

Figure 2.5: Schematic illustration of the functions used in Linear Regression (Equation 2.5) and Logistic Regression (Equation 2.6).

$$\begin{aligned} P(y=1|x) &= \sigma(w^T x + b) \\ P(y=0|x) &= 1 - P(y=1|x) \end{aligned} \tag{2.7}$$

This is the case since the sigmoid function σ maps the input x to a value between 0 and 1, other than in Linear Regression where higher and lower values are possible. This also can be seen by comparing the two illustrations in Figure 2.5.

As described, the curve in Logistic Regression is fit to the data. This is done by using some kind of optimization algorithm, to modify the weights, often called coefficients. Through that, the curve is transformed and shifted to fit the data. An often used optimization algorithm, also used in our approach is Gradient Descent. We do not go into further detail about Gradient Descent since it is not relevant for our approach directly and therefore it would go beyond the scope of this work.

Artificial Neural Network (ANN)

Artificial Neural Networks (ANN) are structures oriented to the functionality of the human brain and are part of the field of deep learning. Like the human brain, ANNs are networks of neurons. These neurons are grouped into different layers (input layer, hidden layer, output layer) and the layers are connected by weighted channels. To learn, a vast amount of data is feed through the network and the weights are adjusted by back-propagation. Figure 2.6 shows a schematic presentation of such a neural network. [1, 40, 41]

The use of ANNs in text classification has gained a lot of interest in the past few years and multiple Artificial Neural Network structures were proposed. Since these are neither used much in PU learning nor in our approach, it would go beyond the scope of this work to present all of the structures and go into details of deep learning. For that reason, we just mentioned Artificial Neural Networks for the sake completeness.

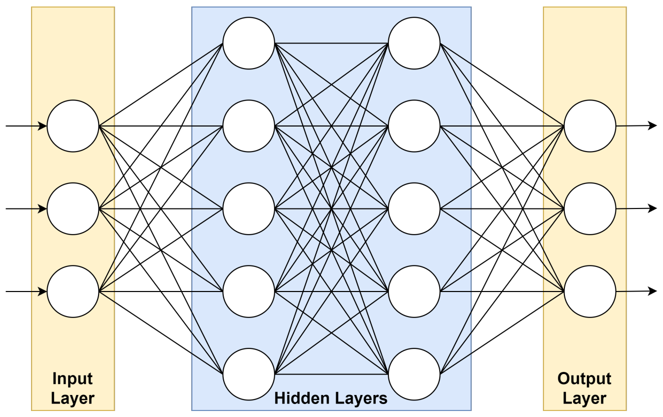


Figure 2.6: Schematic illustration of a Artificial Neural Network (ANN). Like the human brain, ANNs are networks of neurons (circles). These neurons are grouped into different layers (input layer, hidden layer, output layer) and the layers are connected by weighted channels (lines).

2.4 PU Learning

All of the text classification algorithms introduced in Section 2.3.2 were designed for a supervised setting, where fully labeled examples for each class are required and used to train an algorithm. Unfortunately, this is in practice very often not the case and the available examples are either incompletely labeled or not labeled at all (unsupervised learning) and the proposed algorithms do not perform very well in such scenarios. Through that, other research directions, which look at these special scenarios, opened.
One of them is positive-unlabeled (PU) learning, where a binary classifier is learned using only a labeled set P containing only positive examples and an unlabeled set U containing both positive and negative examples.

Through that, the definition of a PU dataset can be seen as a set of triplets (x, y, s), where x is a vector containing the features of an example, y is the class or label of the example and s defines if the example is labeled and therefore part of the positive set P. In other words, in PU learning, if an example is labeled $(s = 1)$ then it belongs to the positive class. This can formally be defined as shown in Equation 2.8.

$$P(y = 1|s = 1) = 1 \tag{2.8}$$

Otherwise, if the example is unlabeled $(s = 0)$ then it can belong to either class (positive or negative). Table 2.2 presents an example of such a dataset, with x being the input features and y being the corresponding class or label. For simplicity, the examples in the table represent the input features as text. Normally this text is transformed into a numeric vector using a feature extraction technique, which we describe in Section 2.6.

x	y	s
Free Bitcoins!!! Here -> bit.ly/eePon3	1	1
You Won 1.000.000 euro: bit.ly/ccPnn	?	0
Good morning #twitter.	?	0
I bought a Tesla today!	?	0

Table 2.2: Snippet of a PU dataset for spam detection in PU learning, containing input data (x) and its corresponding binary label (y) as well as an indicator if the example is labeled (s). For simplicity, the input data x is represented as text. Normally this text is transformed into a numeric vector before it is handed over to the machine learning algorithm.

Because in practice, such a setting is very common (e.g. in medical diagnosis, knowledge base completion, advertisement), this area is researched very well and many different approaches were proposed. [9, 24]
In the literature the approaches are often separated into two [24, 21] or three [49] categories. For simplicity, we use the two category point of view, which separates the approaches into the following two categories.

- 2-step approaches
- Biased approaches

The categories differ in the way how the unlabeled data (U) is handled. In the upcoming sections, we briefly describe the two categories and provide a list of proposed approaches and literature for each of them.

2.4.1 2-Step Approaches

2-step approaches were the first approaches to deal with the PU learning problem. As the name suggests, 2-step approaches can be seen as an iterative method, splitting the process of building a classifier into two major steps. The process is illustrated in Figure 2.7 and works as follows.

- **Step 1** - In the first step of the 2-step approaches a set of negative examples, so-called reliable negative set RN is identified in the unlabeled set U. This is done by treating the unlabeled set U as a negative set N and using it in combination with the positive set P to train a text classification model. The resulting classifier is then used to classify the unlabeled set U. The documents classified as negative by the classifier form the reliable negative examples RN.

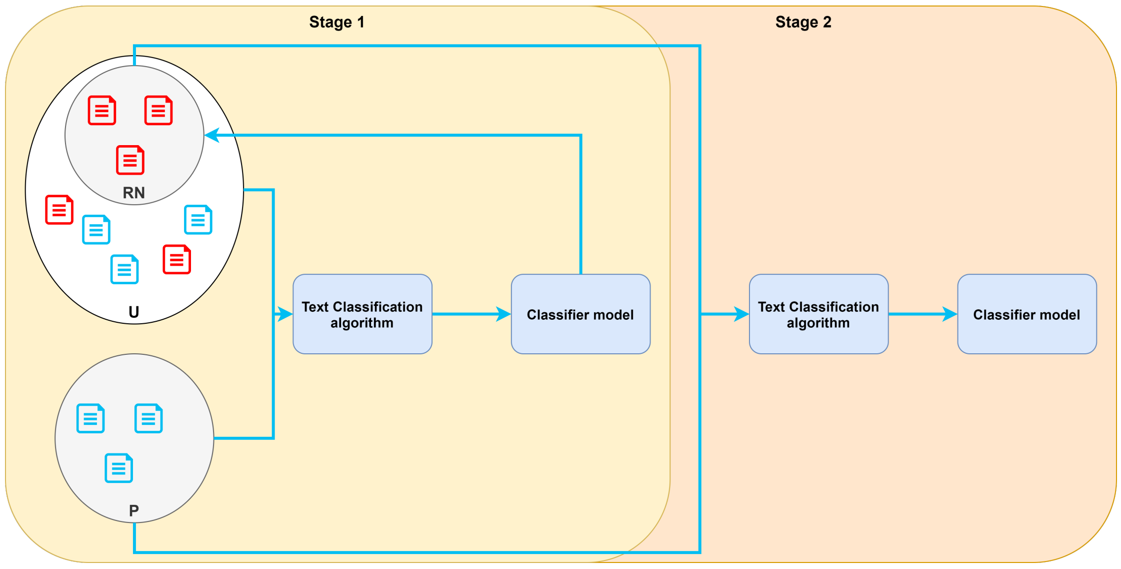


Figure 2.7: Schematic illustration showing the process of 2-step approaches in PU learning. First, a text classification model is trained on the positive set P and the unlabeled set U to identify reliable negatives RN in the unlabeled set. Afterward, the reliable negatives RN and the positive examples P are used to train the final classifier.

As we can see in Table 2.3, Rocchio is a very popular algorithm for step 1. Other algorithms used in the literature quite often are the Spy technique [29], 1-DNF [47] or NB but also kNN and many others can be found.

- **Step 2** - In the second step a traditional supervised learning text classification algorithm is used to train a classifier using the positive examples P and the reliable negatives RN from step 1.

  In step 2 almost every algorithm used for text classification can be used. As shown in Table 2.3, SVM is used in the literature in many different approaches, but also NB or some iterative version of SVM (Iterative SVM) and NB (Expectation-Maximization NB) can be found very often.

- **Step 3** *(Optional)* - The second step is often performed iteratively and multiple classifiers are built to get better results. For that reason, in the literature, often a third step is introduced which looks at the selection of the best performing classifier from step 2. As shown in Table 2.3, there also exist multiple techniques to select a classifier. Most of the time the last model, where some metric has improved is used (ΔE, ΔF). Another technique often used is to check what percent of the positive set P is classified as negative by a classifier. (FNR $> 5\%$)

The performance of this kind of approach is dependent on the chosen classification algorithm and the quality and number of identified reliable negatives RN in the first step. Since it is very easy to replace an algorithm used in one of the steps with another, many different approaches were proposed and published in the past. Liu et al. [28] was the first paper summarizing some 2-step approaches. Table 2.3 summarizes some of the approaches and their respective literature. The table is a modified version from Bekker et al. [3].

Method	Step 1	Step 2	Step 3
S-EM [29]	Spy	EM NB	ΔE
CR-SVM [26]	Rocchio	SVM	-
Roc-SVM [27]	Rocchio	Iterative SVM	FNR $> 5\%$
Roc-Clu-SVM [27]	Rocchio	Iterative SVM	FNR $> 5\%$
PEBL [47]	1-DNF	Iterative SVM	Last

Table 2.3: Proposed 2-step approaches in the literature of PU learning, showing the algorithms and methods used in the different stages of the approach. The third stage is necessary to choose a final classifier when the second stage is executed iteratively. Often some metric is used to measure if the model has improved (ΔE, FNR $> 5\%$) [3]

There are a variety of further approaches proposed and published, which have just modified or replaced the algorithms in one of the steps. [5, 14, 25, 30, 48]

2.4.2 Biased Approaches

Biased approaches consider the PU data as fully labeled data. In other words, it treats the unlabeled set U as a negative set N with class label noise. The goal of this type of approach is to deal with the noise in the negative set in the best way possible. In the upcoming sections, we describe the most important and influential approaches and their respective literature corresponding to this category.

Biased SVM

Many approaches in this category deal with the class label noise by penalizing misclassified examples differently. One of the first approaches applying different penalties to misclassified examples was proposed by Liu et al. [28] in 2003 and is called BiasedSVM. Similar to the 2-step approaches and many other approaches in this category, the base of this approach is the Support Vector Machine (SVM) algorithm. BiasedSVM reformulates the SVM algorithm, applying different penalties to misclassified positive and negative examples. Liu et al. [28] also evaluated the method and observed that it outperforms the original 2-step approaches. There exist a variety of approaches modifying and extending this approach, which we do not discuss in detail here since it would be out of scope. [31, 20]

Bagging PU

Another set of approaches uses the bagging algorithm or a modified version of it. This is done to reduce the effect of noise when considering the unlabeled set as negative set. With other words, the effect of misclassified or rather positive examples in the negative set is reduced.

BaggingSVM was proposed by Mordelet et al. [39] and was one of the first algorithms applying a modified version of bagging to SVM. Different from the traditional bagging algorithm, it just bootstraps the unlabeled set, or rather, in this case, the negative set. Therefore, multiple classifiers are trained using the full set of positive examples and the respective bags of the unlabeled/negative set.

RESVM was proposed by Cleasen et al. [7] and applies the standard bagging algorithm to the SVM algorithm.

Here as well, a variety of other approaches exist but are not discussed in detail. [18, 19]

Weighted Logistic Regression

The last approach of this category that we present is Weighted Logistic Regression, proposed by Lee et al. [22]. This algorithm is one of the few approaches of this category using Logistic Regression instead of SVM. In fact, it uses weights on the input data when training the Logistic Regression model. To be more precise it favors correct positive classification over correct negative classification by assigning higher weights to the examples of the positive set P.

There exist different methods to assign the weights. For example, the weights can be assigned by parameter-tuning using cross-validation or by estimating them in some way. A very common way is to define the weights through the class prior $\alpha = P(y = 1)$, where the positive examples get the negative class prior as weight and the negative examples get the positive class prior as weight. The problem is that the class prior is not known in the PU setting. For that reason, the class prior and therefore the weight has to be estimated in some way. In our approach, we use the label frequency to assign the class weights. The label frequency is closely related to the class prior. Since Weighted Logistic Regression is an essential part of our approach, we describe the estimation of the label frequency, the relation between label frequency and class prior and the functionality of the Weighted Logistic Regression in Chapter 4 in more detail.

2.5 Preprocessing and Text Cleaning

As shown in Figure 2.3 and already mentioned in Section 2.3, text documents are unstructured data and cannot be interpreted by text classification algorithms. For that reason, the data has to be prepared, which includes the preprocessing and the extraction of features. In the following sections, we present the different preprocessing steps to prepare the text data for the extraction of features.

The preprocessing phase can be seen as a chain of transformations applied to the text data. The applied transformations strongly depend on the specific use case and therefore differ from use case to use case. In the following list, we describe the most common and essential preprocessing steps, in more detail.

- **Tokenization**
  The most essential and also mandatory transformation is called tokenization. In tokenization, the text documents are split into smaller pieces according to some configurable rules. These pieces are called tokens. Depending on the rules used to split the text, the resulting tokens can be words, phrases, symbols or other meaningful structures. The rules define which characters are indicators to split the text. One very common character used to split text is the whitespace.

- **Stop Word Removal**
  Text documents often contain words, which carry no important meaning. For example words like "the", "a", "on", "is", "all" are some of the stop words of the English language and are often removed when applying this kind of transformation. Stop word removal is not mandatory and is often left out.

- **Stemming and Lemmatization**
  Stemming and lemmatization describe the reduction of the tokens, or rather the words, to a common base form. The first describes the reduction of the tokens by chopping off word inflections, according to various rules. The latter describes the reduction of the tokens by converting them to their base forms using lexical knowledge. Therefore the difference between stemming and lemmatization is that in lemmatization lexical knowledge is used to get the correct base form of a word, instead of simply chopping off parts of the word.

- **Other Transformations**
  As mentioned, there can be done many different transformations, which strongly depend on the respective use case and text classification task. For example, also often applied are transformations like converting the text to lowercase, removing punctuation, removing numbers, removing other symbols or even spell correction.

2.6 Feature Extraction

After preprocessing, the data is prepared to extract features. This is done by encoding the text documents into numerical vectors. These vectors, often called feature vector, contain, depending on the technique used, information about the syntax and semantics of the text and the words it contains. There are a variety of techniques proposed in the past, often separated into word weighting and word embedding techniques.

Word weighting techniques are due to their simplicity very common in the literature and normally used in text classification, or more precise in PU learning. For that reason, we describe the word weighting techniques in the upcoming section in more detail. For the sake of completeness, we also briefly describe the intuition and basic concept of word embedding techniques.

2.6.1 Word Weighting

Word weighting techniques are normally simple techniques, which make use of word counts and term frequencies. These techniques are, due to their simplicity and the less amount of data needed, commonly used in the literature of text classification. Another advantage is that this kind of technique can be applied to almost every use case. In the upcoming sections, we describe some of the word weighting techniques commonly used in the literature as well as used in our approach.

Term Frequency (TF)

One of the simplest techniques is to use term frequency (TF). The term frequency of a token in a document is calculated by counting the occurrences of the token in the document. The vector for a document is built by calculating the term frequency for each token in a document.
There exist some modifications to this method. One commonly used modification is to logarithmically scale the term frequency. Equation 2.9 shows the mathematical definition of the logarithmically scaled term frequency $tf(t,d)$, where t is the term and d is the document.

$$tf(t,d) = log(1 + freq(t,d)) \tag{2.9}$$

Term Frequency - Inverse Document Frequency (TF-IDF)

Term frequency - inverse document frequency (TF-IDF) is a extended version of the technique using term frequency. In this technique, besides considering the number of occurrences of a term using the term frequency, also the rarity of a term over the entire document set is considered. This is done by simply multiplying the term frequency with another measure called inverse document frequency. Like the term frequency, inverse document frequency is commonly logarithmically scaled, resulting in the mathematical definition shown in Equation 2.10, where D is the entire set of documents, N is the number of documents ($N = |D|$) and $df(t,D)$ is the number of documents in which term t occurs.

$$idf(t,D) = log(\frac{N}{df(t,D)}) \tag{2.10}$$

The TF-IDF for a word in a document is therefore defined as shown in Equation 2.11.

$$tfidf(t,d,D) = tf(t,d) * idf(t,d) \tag{2.11}$$

2.6.2 Word Embedding

Word embedding techniques are more complex techniques, which not only consider the word occurrences but also consider other things like the context and semantics of the text. These techniques often use neural networks and a vast amount of data to learn so-called word embeddings. A multitude of techniques were proposed in the past. The most prominent ones are Word2Vec, Doc2Vec, GloVe, FastText, ELmo and Bert.

Most of these techniques build on the idea of Word2Vec, proposed by Mikolov et al. [37]. In this technique, a large corpus of text is used to produce an n-dimensional vector space. In this vector space, each word can be represented as a vector. Vectors of words with similar contexts are positioned near to each other in the vector space.
As shown in Figure 2.8, Mikolov et al. [37] introduced two shallow neural network architectures to learn and produce the word embeddings, namely Continuous Bag-of-Words (CBOW) and Skip-gram. CBOW takes a window of context words as input and tries to predict the word which suits these surrounding context words. In Skip-gram, on the other hand, the surrounding context words are predicted given the current word as input.

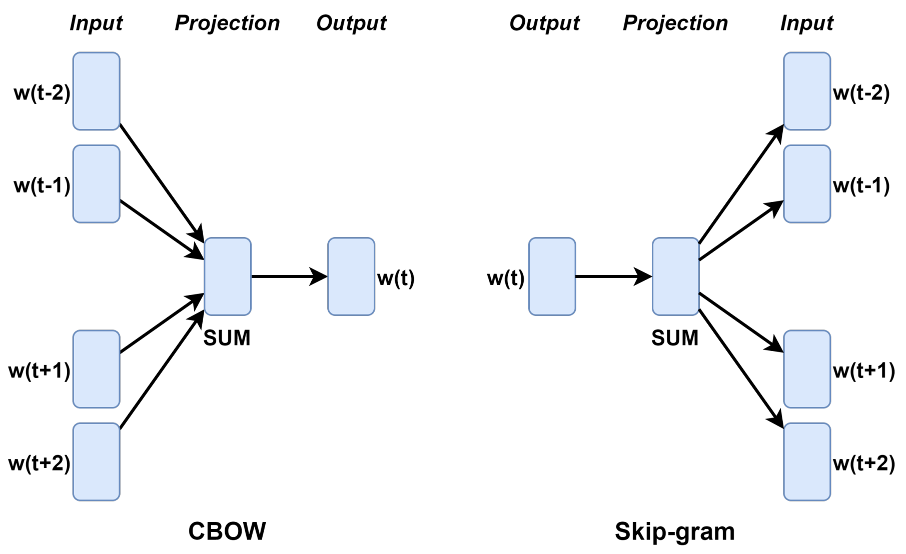


Figure 2.8: Illustration of the two proposed neural network architectures of Word2Vec, namely Continuous Bag-of-Words (CBOW) and Skip-gram. [37]

We do not go into further detail about the different word embedding techniques since it would go beyond the scope of this work.

2.7 Summary

In this chapter we introduced the domain Twitter and showed its relevance in research. Also, we defined a taxonomy and presented existing approaches and literature in the field of text classification and PU learning. Through that it is shown that PU learning is a well-studied sub-field of text classification or more precisely of binary text classification. Furthermore, we described the theoretical foundations and basic concepts of (binary) text classification and PU learning which are essential for this work.

CHAPTER 3

Methodology

This chapter explains in detail the research methods and the methodology used for this study. Through that, the study should be made more comprehensible and reproducible. Also, it shows and justifies why we chose the respective research methodology for this research. As a result, the reliability and validity of this study are strengthened.

First, in Section 3.1, we describe the general research approach. Afterward, we describe the design of the research, in Section 3.2.

3.1 Research Approach

This research made use of an exploratory, quantitative, experimental research strategy. Exploratory research looks at the investigation of unexplored fields, through the discovery of new variables, models and measures. Quantitative research looks at the observation of phenomena using systematic empirical investigation. The data in quantitative research is represented as numbers and is analyzed using statistical, mathematical models. An often-used research strategy in quantitative research is experimental research, where empirical experiments are conducted to observe phenomena.

An exploratory, quantitative, experimental research strategy was appropriate in this study, because of several reasons. Firstly, it is true that PU learning is a well-studied field, but PU learning in practical and domain-specific settings or rather in the Twitter domain was completely unexplored. For that reason, we had to conduct experiments analyzing existing PU learning approaches in the domain-specific setting. The results of these experiments built the baseline in our research. Secondly, we used the gained knowledge, alongside stepwise improvements, backed by recurring experiments, to develop our new approach. Finally, we compared the existing approaches with our new approach using the experiment results. Through that, we drew conclusions and answered the research question, or rather achieved the research goal.

3.2 Research Design

As we already mentioned the research used an exploratory, quantitative, experimental research strategy. In other words, we had an unexplored field, which we investigated by using empirical methods in the form of experiments. The research procedure can be separated into four major stages, namely experiment design, baselining, approach development and evaluation. The research procedure is illustrated in Figure 3.1. The different stages are briefly described in the upcoming sections.

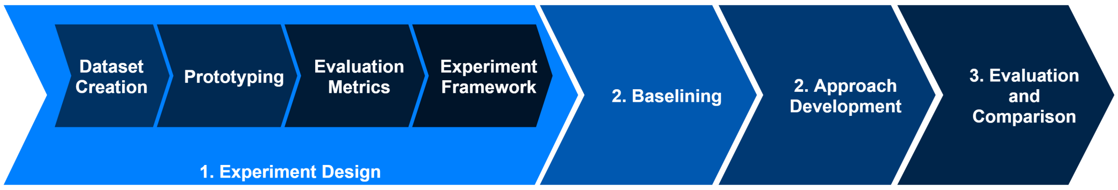


Figure 3.1: Illustration of the chronological ordered research process.

Experiment Design Since most of the proposed PU learning approaches were only evaluated in a generic experiment setup, we evaluated the performance of existing approaches in the domain of Twitter in the earlier stages of the research. Through that, we gathered insights about the performance of PU learning in the domain Twitter and were able to derive baseline metrics, which we used to compare the performance of our approach with existing PU learning approaches.

For this reason, in the first stage of the research, we setup the experiments containing the creation of datasets, the re-implementation of existing PU learning approaches and the definition of evaluation metrics as well as putting it all together in an experiment framework to conduct the experiments and collect the results. The experiment setup is described in detail in Chapter 5. In the following, we briefly summarize the steps of the experiment design.

- The dataset creation was the first step of the experiment design. In this step, we created a variety of PU datasets from three data sources, namely Reuters-21578, Twitter Topic (TTopic), Twitter Spam (TSpam). The PU datasets were used to simulate different PU scenarios. Besides the use of different data sources to simulate different use cases (topic detection, spam detection), this was achieved by constructing multiple PU datasets from each data source, simulating different noise settings. This was done by tweaking the number of labeled examples and the number of positive examples in the unlabeled set using a configurable parameter γ. Since these PU datasets are used in the experiments to train and evaluate the PU approaches and thereby used to simulate a variety of PU scenarios, they are essential for the validity of the results of this study and therefore for the validity of the whole research. In Section 5.1, we describe the single data sources and the creation of the different PU datasets using these data sources in more detail.

- In the second step of the experiment design, we re-implemented a selection of existing PU learning approaches and verified their correctness using Reuters-21578 datasets. These prototypes were later used to build a baseline by evaluating their performance in the domain-specific setting using the Twitter datasets. The re-implemented approaches are discussed in detail, in Section 5.2.

- In the third step, we defined the evaluation metrics used to analyze and compare the experiment results. The PU datasets are constructed in a way that the real label of each example is known and therefore we were able to use evaluation metrics that are commonly used in supervised learning. In Section 5.3, we describe the used evaluation metrics and justify why we can use them in our study.

- In the last step of the experiment design, we put all the parts together in an experiment framework. The framework allows the execution of different experiments, evaluating different PU learning scenarios and approaches as well as collecting and saving the results and artifacts (e.g. trained models, plots, evaluation results) of the experiments. The experiment framework is described in Section 5.4.

Baselining In the second stage of the research, we conducted experiments to evaluate the performance of the prototypes using Twitter datasets. To be more precise, we evaluated the performance of PU approaches for the use cases topic detection and spam detection in the Twitter domain. The evaluation results of these experiments built the performance baseline and gave us insights into the performance of PU learning, or rather existing PU learning approaches in the domain-specific setting. The baseline metrics are presented together with the evaluation results of our approach in Chapter 6.

Approach Development In the third stage of the research, we used the gathered knowledge about the prototypes as well as the domain knowledge to derive and develop a new approach optimized to the Twitter domain. First, we selected one of the re-implemented approaches based on its performance, potential and optimization margin. Afterward, we used stepwise improvements backed by recurring experiments, to create an approach optimized to the Twitter domain. The new approach and all the intermediate stages are described in Chapter 4. The evaluation results of our approach are presented together with all other evaluation results of this research in Chapter 6.

Evaluation and Comparison As briefly mentioned in the previous paragraphs, in the last stage, we collected and summarized all the experiments and evaluation results. This includes all the evaluations made during the research. This is done to make a comparison possible as well as show that the newly created approach outperforms all the existing approaches in the domain-specific setting. All the conducted experiments along with the evaluation results are shown in Chapter 6.

3.3 Summary

In this chapter we explained the research methods and the methodology used. Through that, the research is made more comprehensible and reproducible. Also, it shows and justifies why we chose the respective research methodology for this research. As a result, the reliability and validity of this research are strengthened.

CHAPTER 4

Approach

In this chapter, we present each part of our approach as well as its intermediate optimization and improvement steps. Our approach, called Twitter Weighted Logistic Regression-Support Vector Machine, or short TWLR-SVM, is a hybrid and modular model, more precisely, it is an approach of the 2-step category using an optimized version of the biased learning approach, Weighted Logistic Regression, in the first stage and a domain-optimized SVM classifier in the second stage.

We first introduce some assumptions, which have to hold to enable our approach, in Section 4.1. Afterward, in Section 4.2, we describe Twitter Weighted Logistic Regression, which optimizes the Weighted Logistic Regression approach for the Twitter domain. There we will go in more detail about the optimizations, such as the label frequency estimation (LFE) and the incorporation of it as well as the domain-specific optimizations in the form of feature engineering, including investigating, analysing and incorporating domain-specific statistics and metrics using domain knowledge. At the end, in Section 4.3, we present TWLR-SVM, which uses TWLR in combination with a traditional SVM classifier inside of a 2-step approach.

4.1 Assumptions and Preliminaries

There are different assumptions, which had to be made and hold to enable our approach to work or PU learning in general. In the upcoming section, we define these assumptions and concepts needed to understand those assumptions.

4.1.1 Labeling Mechanism

In PU learning, it is assumed that the labeled positive examples are selected by a probabilistic labeling mechanism, from the complete positive distribution. Thus, this means that negative examples are certainly not labeled. This can be defined formally as shown in Equation 4.1, with s being the indicator that the example is labeled, x being the input and y being the true label.

$$P(s = 1|x, y = 0) = P(s = 1|y = 0) = 0 \tag{4.1}$$

In words, this means that the probability of a negative example to appear in the labeled set is zero. Furthermore, this means that there exists some distribution over the PU dataset defining which examples are labeled. In other words, there exists a certain probability, as defined in Equation 4.2, of a positive example being labeled.

$$P(s = 1|x, y = 1) \tag{4.2}$$

4.1.2 Selected Completely at Random (SCAR)

From the definition of the labeling mechanism, the selected completely at random (SCAR) assumption can be derived. [9] The SCAR assumption is made very commonly in the literature of PU learning. In this assumption, it is assumed that the labeled positive examples are selected completely at random, independent of their attributes. In other words, positive examples that are labeled are a random sample from the positive distribution, completely independent of the attributes of the respective example and therefore the probability of being labeled is equal for each positive example. As we describe later, this is especially important in PU approaches of the biased learning category and methods incorporating the class prior or in our case the label frequency.

The assumption can be formally defined as shown in Equation 4.3, with $c = P(s = 1|y = 1)$ being the constant probability that a arbitrary positive example is labeled. This probability c is called label frequency.

$$P(s = 1|x, y = 1) = P(s = 1|y = 1) = c \tag{4.3}$$

4.1.3 Label Frequency and Class Prior

As already mentioned in Section 2.4.2, the class prior α and the label frequency c are an essential part of many biased learning approaches and also with our approach. The class prior, on the one hand, defines the distribution of the classes in the overall dataset and can be defined by $\alpha = P(y = 1)$. The label frequency, on the other hand, defines the distribution of the labeled examples given that the class is positive and can be defined by $c = P(s = 1|y = 1)$.

The class prior and the label frequency are closely related to each other. The relation is shown formally in Equation 4.6, where the definition of the label frequency (Equation 4.3), alongside with the definition of the label mechanism (Equation 4.5) and the theorem of conditional probability (Equation 4.4) is used to get to the class prior α. Of course, this can also be done the other way around.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{4.4}$$

$$\begin{aligned} P(s=1) &= P(s=1|y=1) + P(s=1|y=0) \quad //\text{Equation 4.1} \\ &= P(s=1|y=1) + 0 \\ &= P(s=1|y=1) \end{aligned} \tag{4.5}$$

$$\begin{aligned} c &= P(s=1|y=1) && //\text{Equation 4.4} \\ &= \frac{P(s=1, y=1)}{P(y=1)} && //\text{Equation 4.5} \\ &= \frac{P(s=1)}{P(y=1)} \\ &= \frac{P(s=1)}{\alpha} \qquad \Rightarrow \alpha = \frac{P(s=1)}{c} \end{aligned} \tag{4.6}$$

$P(s=1)$ can simply be determined by calculating the fraction of labeled examples in the PU dataset. Through that, given either the label frequency or the class prior, the respective other value can be calculated.

4.2 Twitter Weighted Logistic Regression (TWLR)

In this section, we introduce Twitter Weighted Logistic Regression (TWLR), which is a domain-optimized version of the algorithm Weighted Logistic Regression. First, we give a short overview and present the architecture of TWLR. Afterward, we describe the different parts of the architecture in more detail, including the algorithm Weighted Logistic Regression as well as the domain-specific modifications we built-in to optimize the algorithm for Twitter. In the end, we discuss some limitations of the algorithm and the modifications.

4.2.1 Overview

As mentioned in Section 2.4.2, in biased learning, we treat the unlabeled set as a negative set with class label noise and algorithms of this kind try to handle the noise by placing penalties or weights, to reduce the effect of misclassified examples. Weighted Logistic Regression [22] is one of the algorithms from the biased learning approaches and uses larger weights on positive examples to aid correct positive classification over correct negative classification. There are different approaches to set the weights, like treating them as hyperparameters or estimating and using the class prior or the label frequency.

In our approach, we define the weights based on the label frequency. Since the label frequency is normally not known we estimate it using a calibrated probabilistic classifier. The weights in Weighted Logistic Regression normally just use class level weights, in our approach we additionally use the domain knowledge to put weights on sample level to further improve the handling of the class label noise. This is done by calculating different domain-specific statistics and metrics based on the data, and then placing additional weights on negative samples accordingly. Furthermore, we improve the overall performance of the algorithm by extracting custom domain-specific and linguistic features and incorporating them into the learning process of the algorithm using feature engineering techniques.

Figure 4.1 shows the general architecture of the approach. In the upcoming sections, we describe the single parts of it in more detail.

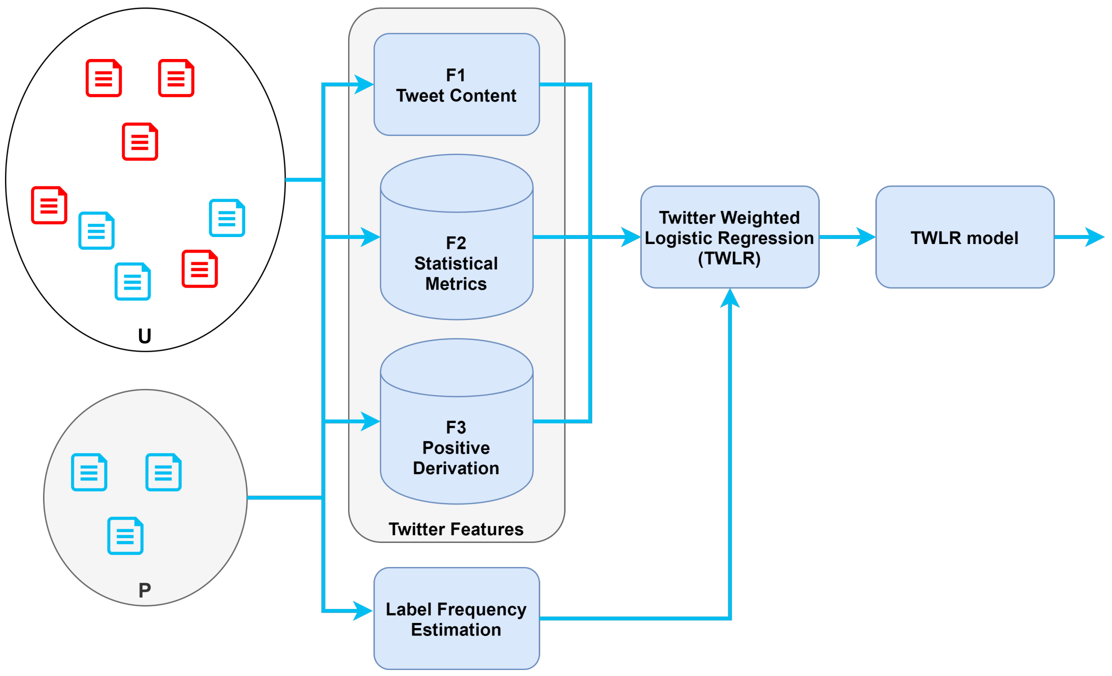


Figure 4.1: Schematic illustration showing the process/architecture of TWLR, where a label frequency estimation (LFE) and Twitter Features are incorporated into a Weighted Logsitic Regression algorithm.

4.2.2 Weighted Logistic Regression

As briefly mentioned, Weighted Logistic Regression [22] is a biased learning approach and uses different class-level weights to aid correct positive classification over correct negative classification. The weights can be defined using different approaches. In our approach, we define the weights based on the label frequency. Since the label frequency is normally not known we estimate it using a calibrated probabilistic classifier.

First, we describe the concept and intuition behind the weighting and why we can use the label frequency to define the weights. Afterward, we describe the estimation of the label frequency.

Class Weighting

Logistic Regression and consequently Weighted Logistic Regression are probabilistic classifiers. A probabilistic classifier is a classifier, which predicts the probability of an example being of a specific class, rather than only predicting the most likely class. In other words, a probability distribution over a set of classes is predicted.

A traditional probabilistic classifier is trained using a fully labeled dataset, containing labeled examples for each class. Therefore in traditional supervised binary classification a probabilistic classifier can formally be defined as linear function $f(x)$, as shown in Equation 4.7, predicting the probability of an given input x being in the positive class $(y = 1)$. [9]

$$f(x) = P(y = 1|x) \tag{4.7}$$

The problem is that we don't have a fully labeled dataset in PU learning and thus such a classifier can not simply be trained. But, using the PU dataset to learn an non-traditional probabilistic classifier, treating the unlabeled set as a negative set with class label noise, we can get a classifier $g(x)$ predicting the probability of an example to be labeled, as presented in Equation 4.8.

$$g(x) = P(s = 1|x) \tag{4.8}$$

Through that, we can get the desired linear function or rather probabilistic classifier $f(x)$ by transforming Equation 4.6, as shown in Equation 4.9.

$$\begin{aligned} P(s = 1|x) &= c * P(y = 1|x) \\ \Rightarrow P(y = 1|x) &= \frac{P(s = 1|x)}{c} \\ \Rightarrow f(x) &= \frac{g(x)}{c} \end{aligned} \tag{4.9}$$

This means, if the probabilities are of no importance, we could simply shift the target probability threshold τ using the label frequency c, as shown formally in Equation 4.10.

$$\tau_{pu} = c * \tau \tag{4.10}$$

Instead of shifting the threshold, it is also possible to employ weights on the training data. Since our target threshold τ is 0.5, we employ the weights using the label frequency c as described in Hsieh et al. [13] and Bekker et al. [3]. The weights we employ are shown in Equation 4.11, with w_+ being the weight for the positive examples and w_- being the weight for the negative or rather unlabeled examples.

$$\begin{aligned} w_+ &= 1 - c * \tau \qquad // \text{ Since } \tau = 0.5 \\ &= 1 - c * 0.5 \\ &= 1 - \frac{c}{2} \\ w_- &= (1 - \tau) * c \qquad // \text{ Since } \tau = 0.5 \\ &= (1 - 0.5) * c \\ &= \frac{c}{2} \end{aligned} \tag{4.11}$$

Since the label frequency c is not known in a PU learning setting, we describe in the upcoming section how we estimate the label frequency c using a well-calibrated probabilistic classifier.

4.2.3 Label Frequency Estimation

To estimate the label frequency c, we use a well-calibrated probabilistic classifier, trained on the PU dataset. In our approach, we use a simple Logistic Regression classifier since it is well-calibrated and probabilistic by nature.

As described in the previous section, by learning a probabilistic classifier on a PU dataset, we get a linear function $g(x) = P(s = 1|x)$ (Equation 4.8). To estimate the label frequency c, we then simply retrieve a subset of labeled examples P from the PU dataset and use the classifier $g(x)$ to predict each example $x \in P$. Afterward, we calculate the average of the predicted probabilities, which is the estimate for the label frequency c. This estimator for $p(s = 1|y = 1)$ can formally be defined as in Equation 4.12, with n being the cardinality of P.

$$E = \frac{1}{n} * \sum_{x \in P} g(x) \tag{4.12}$$

In Equation 4.13, we show the formal proof that the estimator provides an estimation for the label frequency c.

$$\begin{aligned} g(x) &= P(s = 1|x) \\ &= P(s = 1|x, y = 1) * P(y = 1|x) + P(s = 1|x, y = 0) * P(y = 0|x) \\ &= P(s = 1|x, y = 1) * 1 + 0 * 0 \ //\text{since } x \in P \\ &= P(s = 1|x, y = 1) = c \end{aligned} \tag{4.13}$$

Thus, in theory, and through that definition a single example $x \in P$ should be sufficient to determine the label frequency c. In practice, it is advisable to use the average of the predictions over all examples in P. [9]

4.2.4 Twitter Features

To further improve the Weighted Logistic Regression algorithm, we extract and incorporate a multitude of domain-specific features. As shown in Figure 4.2, the features used in our approach can be separated into three main categories. In the upcoming section, we describe the three categories and the features in them, in more detail.

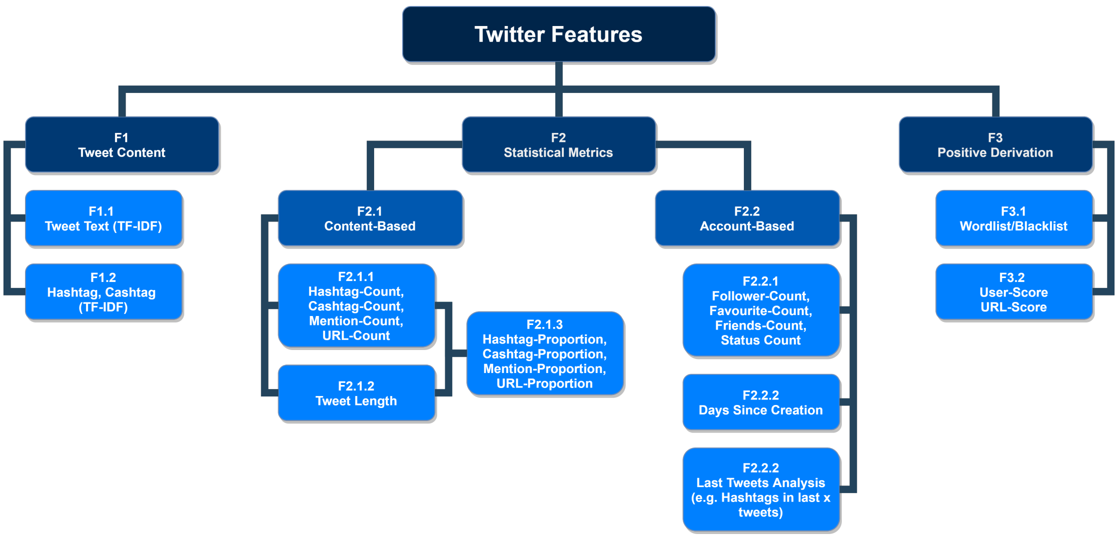

Twitter Features
F1 Tweet Content
F1.1 Tweet Text (TF-IDF)
F1.2 Hashtag, Cashtag (TF-IDF)
F2 Statistical Metrics
F2.1 Content-Based
F2.1.1 Hashtag-Count, Cashtag-Count, Mention-Count, URL-Count
F2.1.2 Tweet Length
F2.1.3 Hashtag-Proportion, Cashtag-Proportion, Mention-Proportion, URL-Proportion
F2.2 Account-Based
F2.2.1 Follower-Count, Favourite-Count, Friends-Count, Status Count
F2.2.2 Days Since Creation
F2.2.2 Last Tweets Analysis (e.g. Hashtags in last x tweets)
F3 Positive Derivation
F3.1 Wordlist/Blacklist
F3.2 User-Score URL-Score

Figure 4.2: Overview of Twitter features we identified and incorporated in our approach.

Tweet Content Analysis (F1)

Features of the first category (F1) look at analyzing the tweet content, or more precisely the syntax of the tweets. In our approach, this is done by preprocessing the text of the tweet first and then converting the preprocessed text into a feature vector using TF-IDF. This is the traditional procedure in text classification very common in the literature, as described in Section 2.3.

To preprocess the text, we transform the text to lowercase, strip the accents, remove stopwords, replace URLs and numbers with a placeholder and of course, tokenize the text. We did the preprocessing and the conversion into a feature vector, using the TfidfVectorizer class from scikit-learn [42] in combination with a custom preprocessing function.

We used this kind of feature, applied to the text of the tweet (F1.1) and optionally to specific fields like hashtags or cashtags (F1.2), in the topic detection use case as well as in the spam detection use case. As we will show later, in Chapter 6, the syntactic analysis of the text works very well for topic detection since the topic is very dependent on the terms used in a tweet. For this reason, this type of feature is the main source of features in topic detection.

The detection of spam on the other side is very ambiguous using only syntactic features derived from the text since spam is most of the time masked and tries to imitate normal tweets. For that reason, the combination with features of other categories is very essential for the accuracy of our approach for spam detection tasks.

Statistical Metrics (F2)

Features of the second category (F2) are statistical metrics calculated from the tweets and the accounts related to the tweets. Thus, we can further separate the second category into content-based metrics (F2.1) and account-based metrics (F2.2). Which metrics are derived from which sources is shown in Figure 4.3.

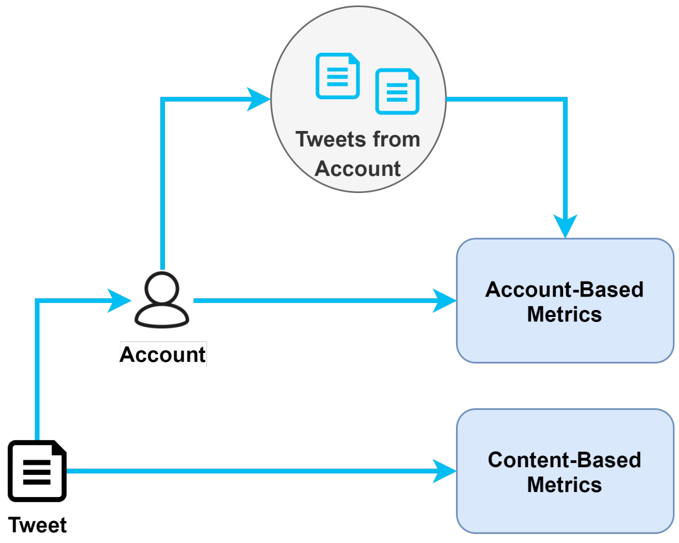


Figure 4.3: Schematic illustration of sources to calculate domain-specific statistical metrics (F2). The statistical metrics (F2) are separated into account-based metrics (F2.1) and content-based metrics (F2.2).

The content-based metrics are directly derived from the tweet like the number of hashtags, cashtags, mentions or URLs used in the tweet (F2.1.1), but also the length of the tweet (F2.1.2). Through the combination of the count-based tweets, with the length of the tweet, proportion metrics are calculated. The account-based metrics are calculated by retrieving extra information about the author of the tweet, like the number of followers, favorites, friends or statuses (F2.2.1), but also the days since the account was created. Through the account information, it is also possible to retrieve and analyze other tweets of the account (F2.2.2). The information needed can be calculated or directly retrieved using the Twitter API and therefore we used the python library tweepy [1] to get the information to calculate the metrics.

[1] `http://www.tweepy.org/` (Accessed on 21.03.2020)

This kind of feature is mainly used in spam detection since the topic is uninfluenced by most of these metrics and therefore unessential for topic detection. Spam tweets, on the other hand, tend to follow specific patterns, which are correlated to these metrics. [17] In the following list we give an overview of the features and how they are relevant for detection of spam tweets:

- **F2.1.1 - Hashtag-Count, Cashtag-Count, Mention-Count, URL-Count**
  Hashtags and Cashtags are tools of twitter to tag, group and categorize tweets. Through that people can easily find tweets of topics they are interested in. User mentions are another tool of twitter, where people can be directly referred and notified. For that reason, spam tweets tend to contain a lot of hashtags, cashtags and user mentions to attract as many people as possible. Also, spam tweets often try to lure people on malicious websites. Thus, spam tweets contain most of the time at least one URL.

- **F2.1.2 - Tweet-Length & F2.1.3 - Hashtag-Proportion, Cashtag-Proportion, Mention-Proportion, URL-Proportion**
  The maximum length of a tweet is 280 characters and is itself therefore not that significant to detect spam. However, through the division of other metrics (e.g. Hashtag-Count, Cashtag-Count, Mention-Count, URL-Count) by the length of a tweet metrics like the proportion of hashtags of a tweet or the proportion of user mentions of a tweet, can be calculated.

- **F2.2.1 - Follower-Count, Favourite-Count, Friends-Count, Status-Count**
  Besides the content of the tweet, the author of a tweet is a great source of information to derive features and metrics. There are different attributes of accounts and users provided by Twitter, which can be used to calculate metrics relevant to spam detection. The Friends-Count and the Follower-Count give insights about the following of a specific account. Spam accounts tend to follow many accounts to attract a lot of people to its tweets but are not followed itself that much. Thus a spam account tends to have a high number of Friends, but a low number of Followers, compared to a normal account. The Status-Count shows how many tweets the specific account wrote. Spam accounts tend to write a lot of statuses in a short amount of time and therefore the number of statuses tends to be high.

- **F2.2.2 - Days Since Creation**
  Another simple information, which we retrieved from the account is the lifetime of the user or rather the days since the user was created since spam accounts tend to be short-lived.

- **F2.2.3 - Last Tweet Analysis**
  The information that we can derive from the account can further be expanded by retrieving other tweets written by the account. Through that, we can add further analysis of the behavior of a user as a feature, ranging from simple statistical

metrics to complex analysis of the tweet content. For example, a spam account tends to post very similar tweets in a short amount of time.

Since we only have the Standard Twitter API the retrieval of the tweets of every account would be very time-consuming. For that reason, we did not incorporate it in this work. The incorporation of last tweet analysis would be a potential topic for future work.

Positive Derivation (F3)

Features of the third category (F3) are derived from the labeled positive set.

One of the features in this category is called wordlist, often found in the literature of spam detection under the name blacklist (F3.1). A wordlist or blacklist identifies common words that are often used in the data of the respective task. For example, words that are common in spam tweets.

To create such a wordlist we apply TF-IDF on the labeled positive set. Afterward, we use the top x weighted terms as wordlist, with x being a configurable variable. In the end, we integrate the wordlist as a feature in the algorithm, as described in Section 4.2.5, using again TF-IDF with a restricted vocabulary. The process is illustrated in Figure 4.4

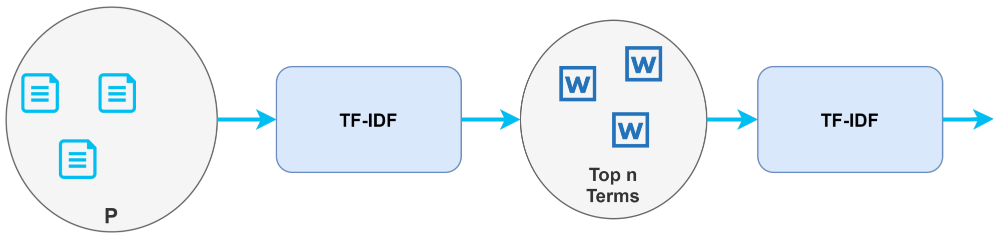


Figure 4.4: Schematic illustration showing the process to create a wordlist/blacklist (F3.1) from the positive set P.

The other feature in this category is a simple URL-score and a user-score which is calculated by counting the occurrences of each URL, and user (author and user mention) in the labeled positive set (F3.2). The user-score and URL-score are mainly used as sample weights in topic detection.

4.2.5 Feature Engineering

We use two different approaches to incorporate the features into the Weighted Logistic Regression algorithm. The first approach is to use the features to identify probably positives in the unlabeled set and put beside the class-level weights additional sample-level weights on them. The other approach is to combine the features and incorporate them in the algorithm when training the model. In the upcoming sections, we describe the two approaches as well as their advantages and limitations.

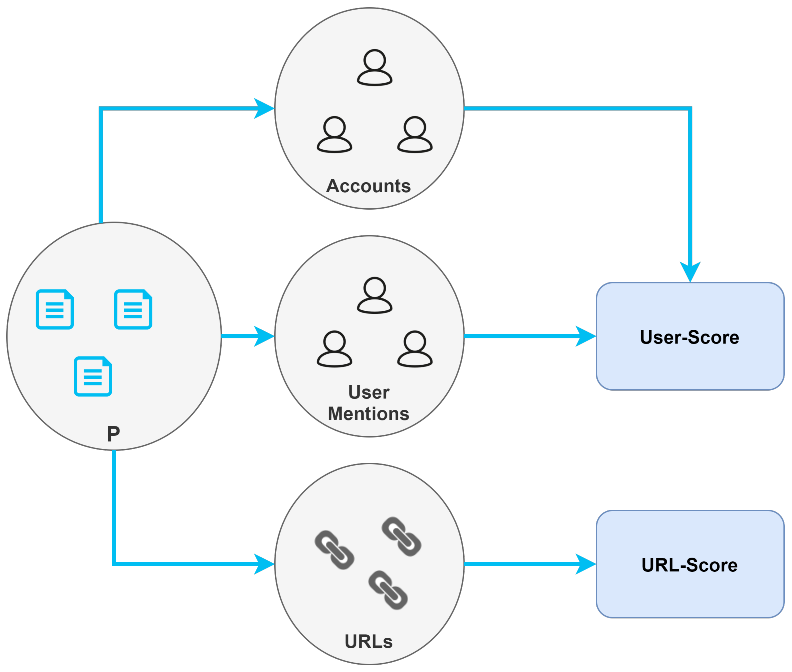


Figure 4.5: Schematic illustration of sources to calculate user-score and URL-score (F3.2).

Sample-Level Weighting

As previously described, we put class-level weights based on the label frequency on the samples when learning. This reduces the impact of noise, or rather the impact of misclassified positive examples in the unlabeled set when learning, considering the unlabeled set as a negative set. The problem is that the label frequency can not be calculated from the PU dataset directly and we have to estimate it, which leads to room for improvements. For that reason, we used some of the previously discussed features to first identify examples in the unlabeled set that are positive with a high probability. Then we put in addition to the class-weight, sample-level weights based on the domain-specific features, or rather the probability of being positive, on the examples, to further reduce the impact of them when training a model. As shown in Equation 4.14, the sample weight w_x for an example x is simply multiplied with the class-weight w_-. Since only the unlabeled set, or rather a negative set, contains noise, we only use sample-weights on examples of the unlabeled set.

$$\begin{aligned} x \in P &\rightarrow x * w_+ \\ x \in U &\rightarrow x * w_- * w_x \end{aligned} \tag{4.14}$$

As we briefly mentioned, the sample-weighting approach further reduces the impact of misclassified examples, or rather of positive examples, when considering the unlabeled set as negative set. As we discuss in Chapter 6, through this the trained model is even more robust and performs more constantly on different levels of noise. The problem is that the identification of positive examples is not a trivial task and has to be configured very well.

Feature Incorporation

The second approach we apply is to combine the different heterogeneous features and incorporate them when training the algorithm. In other words, the algorithm should learn patterns from a variety of different features. Through this, the algorithm should be able to detect and learn further patterns and therefore predict labels more accurately. We did this by putting the different features into pipelines and combining them using the FeatureUnion class provided by scikit-learn [42]. The combination of the heterogeneous features is illustrated in Figure 4.6.

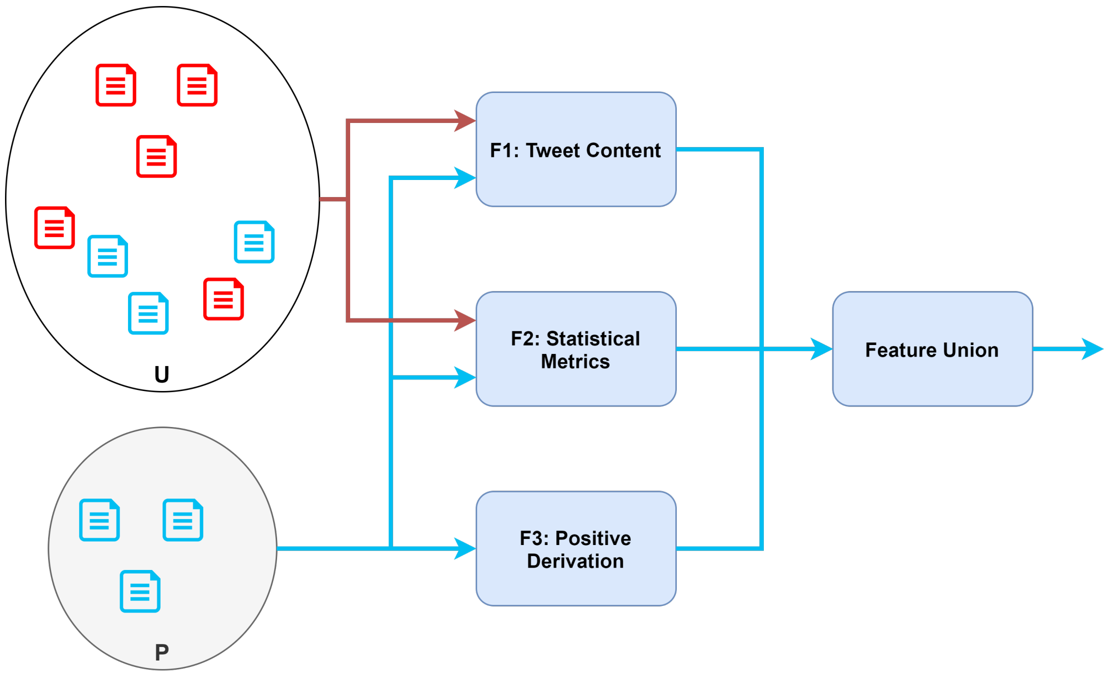


Figure 4.6: Schematic illustration of Twitter Features combination using Feature Union.

The combined features allow the trained model to detect more patterns and therefore the predictions are more accurate. As we will discuss in Chapter 6, this approach can improve the performance significantly, but it is hard to find a good subset of features. Firstly, when choosing too many features or too many low-quality features we face the problem of overfitting the model. This means the model performs well on the training data but fails to generalize to new examples. Secondly, all the chosen features have to be calculated for each example, as well as training examples as new examples we want to predict. This can be very costly and decrease the performance of the algorithm significantly.

4.3 TWLR-SVM

In this section we introduce TWLR-SVM, which combines the domain-optimized algorithm TWLR, presented in the previous section, with the SVM algorithm in a 2-step approach. First, we give a short overview and present the architecture of TWLR-SVM. Then, we describe the 2-step algorithm to combine TWLR with SVM. In the end, we discuss further modifications and optimizations as well as their advantages and disadvantages.

4.3.1 Overview

As mentioned in Section 2.4.1, the performance of 2-step approaches strongly depends on the performance of the first stage and how many reliable negatives RN can be defined in the unlabeled set U. Since the domain-optimized version of the Weighted Logistic Regression algorithm, TWLR, performs standalone already very well and is able to return probabilities of examples being in a class, we used it in the first stage of the 2-step approach. Besides the normal procedure of identifying reliable negatives RN, we also identify additional reliable positives RP and add them to the labeled positive set P ($P = P \cup RP$). The identification is done, as shown in Equation 4.15, using probability thresholds τ_{rn} and τ_{rp}, which can be configured.

$$\begin{aligned} RN &\leftarrow \{x | x \in U, P(y=0|x) \geq \tau_{RN}\} \\ RP &\leftarrow \{x | x \in U, P(y=1|x) \geq \tau_{RP}\} \end{aligned} \tag{4.15}$$

Through the threshold, the quantity and the quality of the identified set can be controlled. The architecture of the 2-step approach TWLR-SVM is illustrated in Figure 4.7.

4.3.2 Algorithm

The algorithm for TWLR-SVM, is a modified version of the 2-step algorithm and is presented in Algorithm 4.1. Since we developed the approach to be modular and the TWLR could be used as a standalone classifier as well, we hand a pre-trained TWLR model over to the modified 2-step algorithm. Using the TWLR model, we then predict the probabilities for each example x in the unlabeled set U of being in the positive or the negative class. Afterward, we identify a set of reliable negatives RN and a set of reliable positives RP, based on configurable probability thresholds τ_{rn} and τ_{rp}. In the end, we combine the labeled positive set P with the reliable positive set RP and use it together with the reliable negative set RN to train a SVM model.

4.3.3 Twitter Features

As in TWLR, we are also incorporate different domain-specific features into TWLR-SVM. The features are the same as described in Section 4.2.4. In TWLR-SVM it is possible to incorporate different features in each of the two steps and there are different strategies of choosing which features to use in which stage of the algorithm.

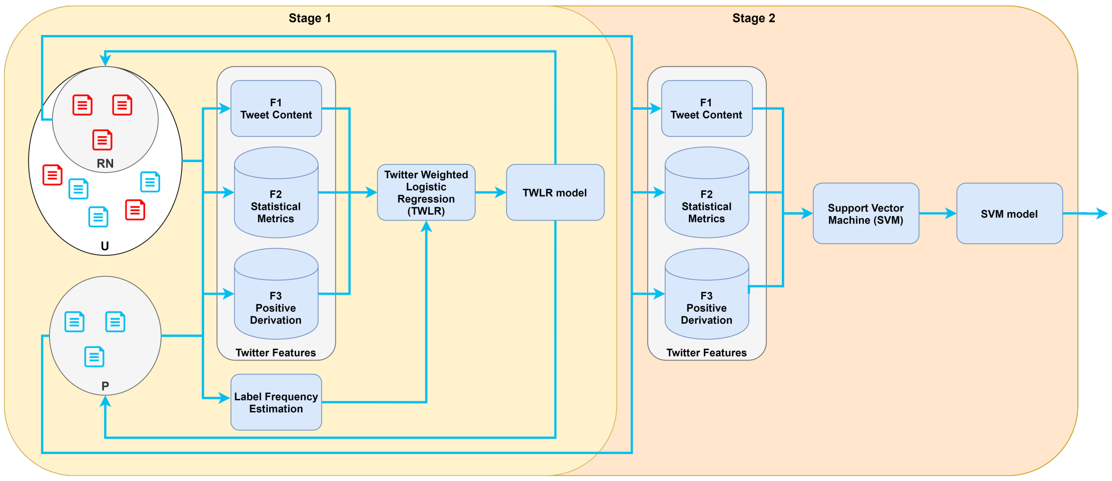


Figure 4.7: Schematic illustration showing the process/architecture of our 2-step approach TWLR-SVM. First, a TWLR model is trained on the positive set P and the unlabeled set U to identify reliable negatives RN in the unlabeled set. Afterward, the reliable negatives RN and the positive examples P are used to train a SVM classifier.

Algorithm 4.1: TWLR-SVM

```
  Input: positive examples $\mathbf{P}$ with class label 1,
  unlabeled examples $\mathbf{U}$,
  a trained TWLR classifier $\mathbf{c_{twlr}}$,
  a untrained SVM classifier $\mathbf{c_{svm}}$,
  two thresholds $\tau_{\mathbf{RN}}, \tau_{\mathbf{RP}}$ to identify RN and RP
  Output: trained final classifier $\mathbf{c_{final}}$
1 $\mathbf{RN} \leftarrow [\,]$;
2 Use $\mathbf{c_{twlr}}$ to predict probabilities $P(y|x)$ of examples $x \in \mathbf{U}$;
3 $\mathbf{RN} \leftarrow \{x | x \in \mathbf{U}, P(y=0|x) \geq \tau_{\mathbf{RN}}\}$;
4 $\mathbf{RP} \leftarrow \{x | x \in \mathbf{U}, P(y=1|x) \geq \tau_{\mathbf{RP}}\}$;
5 $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{RP}$;
6 Train $\mathbf{c_{svm}}$ using $\mathbf{P}$ and $\mathbf{RN}$;
7 $\mathbf{c_{final}} \leftarrow \mathbf{c_{step2}}$;
8 return $\mathbf{c_{final}}$
```

To understand the strategies, we differentiate between complex features and lightweight features. Complex features, on the one hand, are the features, which are costly to compute or features where a separate API[2] call, to retrieve additional data from Twitter, is required. Lightweight features, on the other hand, are features that are simple to

[2]An API is an interface, which allows developers accessing application-related data using programs and tools. In the case of Twitter the API allows access to Twitter-related data (e.g. tweets, accounts).

compute and can simply be retrieved directly from the tweet.

In the first stage or in the TWLR part of the approach, we normally use the best performing set of features. Therefore, we use a variety of features containing lightweight features as well as complex features. This is done since the classifier of the first stage is just used to identify reliable negatives and reliable positives in the training set. Through that, we only need to compute the costly features for the training set.

In the second stage or in the SVM part of the approach, there are different strategies with their own advantages and disadvantages.

- The first and simplest strategy is to incorporate no domain-specific features in the SVM classifier. Through that, the resulting trained classifier is very simple since no features have to be computed for prediction. Furthermore, the risk of overfitting is reduced and therefore all problems of TWLR are addressed. However, the accuracy of the classifier suffers as a result.

- The second strategy is to incorporate the same features as in the first stage, containing lightweight features as well as complex features. Through that, the final classifier delivers the best accuracy. However, the risk of overfitting and poor performance on prediction is present.

- A trade-off between the two strategies above is to use only lightweight features in the SVM classifier. Through that, the approach still delivers very good results on tasks where features are essential, such as spam detection. Through this strategy, we also only have to calculate the costly features for the training set and don't need them for the prediction of new examples, since the SVM model was not trained using them. Furthermore, leaving out the complex features and using only lightweight features reduces the risk of overfitting since lightweight features are not as prone to overfitting as complex features.

In Chapter 6, we evaluate all three strategies and discuss the advantages and disadvantages in further detail.

4.4 Summary

In this chapter, we presented our new approach, called Twitter Weighted Logistic Regression-Support Vector Machine or short TWLR-SVM. It is a modular and hybrid PU learning approach optimized to the Twitter domain.

The optimizations include, on the one hand, feature engineering in the form of investigation, analysis, incorporation and evaluation of domain-specific features. On the other hand, they contain the combination of those features alongside the best-suited algorithms in a PU approach. To be more clear, it is a 2-step approach using an optimized version of the biased learning approach, Weighted Logistic Regression, in the first stage and a SVM classifier in the second stage.

Besides the approach, we introduced assumptions that have to hold for our approach to work. Furthermore, we described the features we extracted and how we incorporated them to optimize our approach for the Twitter domain.

CHAPTER 5

Experiment Design

In this chapter, we describe how the experiments we conducted are designed. This includes, on the one hand, how the datasets we used for training and evaluation are created, on the other hand, the prototypes we implemented and how we verified their correctness. Furthermore, the evaluation metrics used to analyze and compare the performance of the approaches as well as the experiment framework we built to conduct the experiments and collect the results are presented.

First, in Section 5.1, we describe the dataset creation, where we go into the details of how we retrieved and labeled the data to get valid PU learning scenarios. Afterward, we present the approaches we re-implemented and evaluated to derive baseline metrics, which we used to compare the performance of algorithms, in Section 5.2. In Section 5.3 we describe the evaluation metrics used in the experiments to measure and analyze the performance of the different approaches. In the end, in Section 5.4, we describe the experiment framework to conduct the experiments and collect the evaluation results.

5.1 Dataset Creation

We created a variety of different PU datasets from three different sources, which were used to simulate different PU learning scenarios.

One data source used to create PU datasets was the popular text collection Reuters-21578 from Reuters newswire. The PU datasets generated from this data source were used to verify the correctness of our re-implemented PU approaches since this data source is often used in the literature and many evaluation results of PU learning approaches are published using this data source.

The other two data sources were used to simulate PU learning scenarios in the domain-specific setting, or rather in the Twitter domain and therefore contain tweets. One of the data sources was directly retrieved over the Twitter API and was used to simulate the use case of topic detection. The other was a fully labeled dataset from the literature used to simulate the use case of spam detection. On the one hand, the datasets to simulate different PU learning scenarios generated from these two data sources were used to build a baseline from the re-implemented approaches, on the other hand, they were used to evaluate and compare the existing approaches (baseline), with our new approach and its intermediate optimization stages. As mentioned in Section 3.2, a variety of PU scenarios were covered. Besides the use of different data sources, this was achieved by constructing different PU datasets from the data sources. In the upcoming sections, we describe the data sources as well as the respective creation of PU datasets in more detail.

5.1.1 Reuters-21578

As mentioned in the previous section, one data source used for creation of PU datasets was the popular text collection Reuters-21578 [23], which contains 21.578 documents divided into 135 categories from the news agency Reuters newswire. For the creation of PU datasets, we only used the top 10 categories based on the number of documents in each category since this is very common in the literature. The top 10 categories and the document counts in the respective category are listed in Table 5.1.

earn	acq	money	grain	crude	trade	interest	ship	wheat	corn
3964	2369	717	582	578	486	478	286	283	238

Table 5.1: Top 10 categories in terms of document counts in the Reuters-21578 corpus.

PU Dataset Creation

A PU dataset was created by first defining one of the categories as the positive class ($y = 1$) and the rest of the categories as the negative class ($y = 0$). Since we normally don't have a negative set N in a PU setting, we then randomly selected a fraction γ from the documents of the positive class, which was used as labeled positive set P ($s = 1$). The rest of the documents of the positive class ($1 - \gamma$) alongside the documents of the other categories, or rather the negative class, were defined as the unlabeled set U ($s = 0$). Through that, a valid PU dataset was created and since we know the real labels, we were able to use evaluation metrics that are commonly used in supervised learning to determine the performance of PU learning approaches. Through different configurations of γ and choosing other categories as the positive class, we were able to generate a variety of PU scenarios, which differ in the available information about the positive class and the number of positives in the unlabeled set. Similar setups are very common in the literature and therefore many evaluation results of existing PU approaches are available using this setting. [9, 11, 27, 28] Since we were mainly interested in the domain-specific setting, we only used this setup to verify the re-implemented approaches, discussed in Section 5.2.

Implementation and Retrieval

To retrieve the Reuters-21578 corpus we used the python library Natural Language Toolkit (NLTK) [32], which provides an interface to load the documents in a preprocessed form. We then organized and labeled the documents as described above. We did this by combining the documents of the top 10 categories, together with their respective real label and their label in the PU setting, inside of a dataframe using the python library pandas [35]. A example snippet of such a dataframe/dataset, is shown in Table 5.2.

document	label	training_label
AMATIL PROPOSES TWO-FOR-FIVE...	1	1
BOWATER 1986 PRETAX PROFITS...	1	1
CITIBANK NORWAY UNIT LOSES...	1	0
SUMITOMO BANK AIMS AT QUICK...	0	0

Table 5.2: Snippet of a PU dataset/dataframe generated from the data source Reuters-21578.

5.1.2 Twitter Topic (TTopic)

The second data source used was one of the main data sources in this work simulating the domain-specific setting. More precisely, this data source was used to simulate the use case topic detection. Datasets created using this data source consisted of tweets, which were gathered directly from Twitter using the Twitter API.

The Twitter API provides different interfaces to retrieve tweets. We used mainly the API to retrieve random samples and the Streaming API, which returns recent tweets. With keyword-based queries, we filtered the tweets returned by the interfaces.

PU Dataset Creation

Like in the previous setup using Reuters-21578, we first defined the positive and the negative class. For that reason, we chose an arbitrary topic as the positive class ($y = 1$). In our experiments, we used the topic cryptocurrency as the positive class ($y = 1$). Afterward, we derived keywords, which are unique to the chosen topic. In our case, keywords like "cryptocurrency", "bitcoin", "$BTC" and "$ETH" are topic-unique keywords. In the next step, we gathered tweets filtered by the topic-unique keywords, as well as completely random tweets, over the Twitter API. Then, again a fraction γ of the documents of the positive class was randomly selected and used as labeled positive set P ($s = 1$). The randomly gathered tweets were then combined with the rest of the positive documents ($1 - \gamma$) to build the unlabeled set U ($s = 0$). Through that, again a valid PU dataset was artificially created, where we know the real labels and therefore can use evaluation metrics that are commonly used in supervised learning to measure and compare the performance of different PU learning approaches. Also, we were able to generate and simulate a variety of PU scenarios by using different settings for γ and choosing different topics as the positive class. Besides the text of the tweets, we also extracted information, like hashtags, cashtags, URLs and the user from the tweets, which we used in our approach, described in Chapter 4.

Implementation and Retrieval

To retrieve the tweets over the Twitter API we used the python library tweepy [1], which provides both an interface to retrieve random samples, and streaming recent tweets. We wrote a data loader, where we can set the keyword after which should be filtered by and how many samples should be gathered. The data loader gathers the tweets and saves them as JSON into a file. Afterward, we were able to read the file into a pandas [35] dataframe and add the labels as described above. An example snippet of such a dataframe is shown in Table 5.3

document	user_id	...	hashtags	label	training_label
I love $BTC	2274004166	...	[]	1	1
...market plunges #bitcoin ...	2574004356	...	[bitcoin]	1	0
good morning #twitter	453629852	...	[twitter]	0	0
I bought a Tesla today!	8075913251	...	[]	0	0

Table 5.3: Snippet of a PU dataset/dataframe for topic detection generated from the data source TTopic.

In our experiments, we retrieved 10.000 topic-filtered tweets representing the positive class and 10.000 random tweets representing the negative class.

This data source has one limitation. Since the negative set was gathered by retrieving random tweets, it already can contain topic-specific tweets. Through that, the dataset can contain a very small fraction of mislabeled examples.

5.1.3 Twitter Spam (TSpam)

The last data source was also a data source simulating the domain-specific setting. More precisely, this data source was used to simulate the use case of spam detection. This data source was gathered using a labeled dataset proposed by Chen et al. [6]. The dataset contains 100.000 labeled examples, in the form of IDs of tweets together with a label indicating if the respective tweet is spam or not. We gathered the data of the tweets using the Twitter API.

[1] http://www.tweepy.org/ (Accessed on 21.03.2020)

PU Dataset Creation

The creation of datasets was then again very similar to the previous data source, besides the fact that the positive set and the negative set are already given and therefore we just had to construct the PU setting. Once again, a fraction γ of the documents of the positive class was randomly selected and used as labeled positive set P $(s = 1)$. The examples of the negative class were then combined with the rest of the positive documents $(1 - \gamma)$ to build the unlabeled set U $(s = 0)$. Through that, we had once again artificially created a valid PU dataset, where we know the real labels and therefore were able to use evaluation metrics that are commonly used in supervised learning to measure and compare the performance of different PU learning approaches. Also, we were able to generate and simulate a variety of PU scenarios by using different settings for γ. Besides the text of the tweets, we also extracted information, like hashtags, cashtags, URLs and the user from the tweets, which we used in our approach, described in Chapter 4.

Implementation and Retrieval

To gather the tweets over the ID we also used the python library tweepy [2], which provides an interface to gather the tweet for a specific ID. We wrote a data loader, which loads the tweets for each ID in the dataset and saves them as JSON into a file. Afterward, we can read the file into a pandas [35] dataframe and add the labels as described above. An example snippet of such a dataframe is shown in Table 5.4

document	user_id	...	hashtags	label	training_label
Click Here -> https://...	1379010116	...	[free, ipad]	1	1
You Won 1.000.000 euro ...	2134356	...	[won]	1	0
othellomor Knw ur busy ...	3343359453	...	[]	0	0
I just picked up a book, ...	607592343	...	[]	0	0

Table 5.4: Snippet of a PU dataset/dataframe for spam detection generated from the data source TSpam.

Policies of Twitter restrict publishing tweet content in the form of datasets. For that reason, labeled tweet datasets often just contain the tweet ID alongside the corresponding label. To use the dataset the tweets have to be gathered using the Twitter API. The problem is that in case of spam tweets, many are detected by Twitter's spam detection algorithm and therefore are often removed and not available anymore. Through that, spam datasets are incomplete after a short amount of time.

This means that in the case of the dataset proposed by Chen et al. [6], we were only able to retrieve 54.000 labeled examples (containing only 4.000 spam examples), instead of the 100.000 labeled examples originally in the dataset.

[2] `http://www.tweepy.org/` (Accessed on 21.03.2020)

5.1.4 Train/Test Split

Each dataset, independent of the data source and settings used to create it, was split into a training set and a test set. We used a 70/30 split, which is very common in the literature. This means we randomly selected 70% of the examples in a dataset and used them as training data. The remaining 30% of the dataset are used as test data. The documents and labels in the training data were used to learn models. The test set was used to test the trained model, by first making predictions on the documents and then comparing the predictions with the real label.

5.2 Prototyping and Baselining

As described in Section 2.4, there exist a variety of PU learning approaches. The problem is that there is almost no research about these approaches in domain-specific settings, like in our case in the domain of Twitter. For that reason, we had to re-implement a selection of existing PU learning approaches, in one of the earlier stages of this research. Afterward, we verified their correctness and evaluated their performance on the domain-specific setting by conducting different experiments. The evaluation results in the domain-setting served us as baseline metrics. Also, we used the gained knowledge about which approaches perform well in which conditions and settings for our new approach, described in Chapter 4.

Since there exists a variety of different approaches and it would be out of scope to implement all of them, we first had to select which approaches to re-implement and evaluate. We therefore mainly used and re-implemented original approaches, since these approaches are commonly referenced in the literature and most of the time used as the basis in newer approaches. Of course, we re-implemented both types of approaches, 2-step approaches and biased learning approaches. In the upcoming sections we describe the re-implemented 2-step approaches, as well as the biased learning approaches in more detail.

5.2.1 Two-step approaches

As described in Section 2.4.1, in two 2-step approaches it is very easy to replace an algorithm, used in one stage, by another. For that reason, we implemented a parametric 2-step classifier, where the classifiers used can be passed as a parameter. Through that, it was possible to evaluate a multitude of different approaches. The algorithm of the 2-step classifier is presented in Algorithm 5.1.

The algorithm can be separated into three areas. Line 1- 4 represents the logic for step 1 of the 2-step approach. Line 5-27 represents the logic for step 2 of the 2-step approach, where Line 6-24 alone describes the iterative version of step 2.

In step 1 of the 2-step classifier (Line 1- 4) a passed untrained classifier (c_{step1}) is trained on the full PU training set. Afterwards, the resulting trained classifier predicts the labels

Algorithm 5.1: 2-step classifier

```
Input: positive examples $\mathbf{P}$ with class label 1,
unlabeled examples $\mathbf{U}$ labeled with class label 0,
two untrained classifiers for the respective step $\mathbf{c_{step1}}$, $\mathbf{c_{step2}}$,
flag indicating iterative second step iter
Output: trained final classifier $\mathbf{c_{final}}$
1  $\mathbf{RN} \leftarrow []$;
2  Train $\mathbf{c_{step1}}$ using $\mathbf{P}$ and $\mathbf{U}$;
3  Use $\mathbf{c_{step1}}$ to classify $\mathbf{U}$;
4  $\mathbf{RN} \leftarrow$ examples classified as negative;
5  Train $\mathbf{c_{step2}}$ using $\mathbf{P}$ and $\mathbf{RN}$;
6  if $iter == True$ then
7      $\mathbf{c_{last}} \leftarrow \mathbf{c_{step2}}$;
8      $\mathbf{Q} \leftarrow \mathbf{U} - \mathbf{RN}$;
9      Use $\mathbf{c_{last}}$ to classify $\mathbf{Q}$;
10     $\mathbf{NN} \leftarrow$ examples classified as negative;
11     while NN *not empty* do
12         $\mathbf{RN} \leftarrow \mathbf{RN} \cup \mathbf{NN}$;
13         $\mathbf{c_{last}} \leftarrow$ new classifier with same type like $\mathbf{c_{last}}$;
14         Train $\mathbf{c_{last}}$ using $\mathbf{P}$ and $\mathbf{RN}$;
15         $\mathbf{Q} \leftarrow \mathbf{U} - \mathbf{RN}$;
16         Use $\mathbf{c_{last}}$ to classify $\mathbf{Q}$;
17         $\mathbf{NN} \leftarrow$ examples classified as negative;
18     end
19     Use $\mathbf{c_{last}}$ to classify $\mathbf{P}$;
20     if $> 5\%$ *are classified as negative* then
21         $\mathbf{c_{final}} \leftarrow \mathbf{c_{step2}}$;
22     else
23         $\mathbf{c_{final}} \leftarrow \mathbf{c_{last}}$;
24     end
25 else
26     $\mathbf{c_{final}} \leftarrow \mathbf{c_{step2}}$;
27 end
28 return $\mathbf{c_{final}}$
```

for the unlabeled documents U and defines the negative predicted documents as reliable negatives RN used in step 2.

In step 2 of the 2-step classifier (5-27), we differentiate between the normal version and the iterative version. In the normal version (Line 5), just another passed untrained classifier is trained on the positive documents P and the reliable negatives RN of step 1. Afterwards, the resulting classifier is used as final classifier.

In the iterative version (Line 6-24), we iteratively train new classifiers (c_{last}), which are used to further enrich the reliable negatives RN by predicting the labels for the rest of the unlabeled set ($Q = U - RN$). The iteration stops when no new reliable negatives are identified by a classifier. In the end, the algorithm checks if the last classifier predicts the majority of the positive documents as positive. In our experiment, we configured, that 95% of the documents have to be classified as positive. If the check fails the first classifier is used as the final classifier, otherwise the last classifier is used as the final classifier.

5.2.2 Implementation

We implemented the 2-step classifier in python. The classifiers passed to the algorithm were used from the machine learning library scikit-learn [42]. In our experiments, we used the algorithms Naive Bayes, Rocchio and k-Nearest Neighbor for the first stage and Support Vector Machine, including the normal and the iterative version, for the second stage. The classifiers representing these algorithms were used from scikit-learn and are shown in Table 5.5.

Step	Algorithm	Classifier (scikit-learn)
1	Naive Bayes	MultinomialNB
1	Rocchio	NearestCentroid
1	k-Nearest Neighbor	KNeighborsClassifier
2	SVM	LinearSVC
2	SVM-I	LinearSVC (incl. iterative flag)

Table 5.5: Text classification algorithms and corresponding scikit-learn class used in the different stages of the 2-step prototypes. When the iterative flag is enabled, new classifiers to enrich the reliable negative set are iteratively trained.

Each technique for step 1 can be combined with each technique of step 2. Through that, we were able to evaluate up to 6 different combinations, which easily could be expanded by further. The combinations include the probably most common PU algorithms Roc-SVM [27] and CR-SVM [26].

5.2.3 Biased Learning

In Section 2.4.2, we presented three of the original and most influential approaches. From the three approaches presented we re-implemented two approaches, namely Bagging PU and Weighted Logistic Regression. We left out Biased SVM since it is outperformed by Weighted Logistic Regression and Bagging PU in the literature. In the upcoming sections, we describe the prototypes in more detail.

Bagging PU

Bagging PU uses a modified version of the Bagging algorithm. The difference is that, instead of bootstrapping the positive and the negative set, in Bagging PU only the unlabeled set is bootstrapped. We implemented two versions of the Bagging PU algorithm. One using SVM and one using Decision Trees. The version using SVM was proposed by Liu et al. [28] and was the first version of this approach using bagging.

Algorithm 5.2 presents the algorithm implemented to train a Bagging PU algorithm. The algorithm for the prediction of a single example is presented in Algorithm 5.3

Algorithm 5.2: Bagging PU - Training

```
   Input: positive examples $\mathbf{P}$ with class label 1,
   unlabeled examples $\mathbf{U}$ labeled with class label 0,
   a untrained classifiers which should be used classifier,
   number of estimators $\mathbf{est_{count}}$
   Output: list of classifiers $\mathbf{c_{list}}$
1  for $i$ in $\mathbf{est_{count}}$ do
2      $\mathbf{B} \leftarrow$ random sample of size $|P|$ of $U$ with replacement;
3      Train SVM/DT classifier using $\mathbf{P}$ and $\mathbf{B}$;
4      Add classifier to $\mathbf{c_{list}}$
5  end
6  return $\mathbf{c_{list}}$
```

Weighted Logistic Regression

Weighted Logistic Regression [22] uses larger weights on positive examples to aid correct positive classification over correct negative classification. There are different approaches to set the weights, like treating them as hyperparameter or estimating and using the class prior or the label frequency.

In the prototype of this algorithm, we used a simple weighting strategy using different static weights and choosing the best performing algorithm for our evaluation results. To be more precise, we used four different weight settings for the unlabeled set (0.3, 0.6, 0.9, 1.0) and one weight setting for the positive set (1.0) and chose the evaluation results of the best performing algorithm based on its F1-score.

Algorithm 5.3: Bagging PU - Prediction

```
   Input: a example to predict doc,
   the trained classifiers $\mathbf{c_{list}}$
   Output: prediction for sample
1  for $i$ in $|\mathbf{c_{list}}|$ do
2      Use $\mathbf{c_{list}}[i]$ to classify doc ;
3      $\mathbf{sum_{pred}} \leftarrow \mathbf{sum_{pred}} +$ prediction of $\mathbf{c_{list}}[i]$ ;
4  end
5  $\mathbf{avg_{pred}} \leftarrow \mathbf{sum_{pred}} / |\mathbf{c_{list}}|$ ;
6  if $\mathbf{avg_{pred}} > 0.5$ then
7      return $1$ ;
8  else
9      return $0$ ;
10 end
11 return $\mathbf{c_{list}}$
```

As we describe later in Chapter 6, even using this simple weighting strategy the algorithm outperformed most of the other algorithms. That's also why we used this algorithm in our approach. For that reason, this algorithm, as well as a more complex weighting schema, is described in Chapter 4, in more detail.

5.2.4 Preprocessing and Feature Extraction

The text examples in the PU datasets are unstructured data. Unstructured data is difficult to analyze and making sense of by algorithms. For that reason, we had to preprocess and extract features from the data.

We applied just very basic preprocessing methods on the prototypes since this is often the case in the literature. The preprocessing methods we applied were lowercasing, accent stripping, stopword removal and tokenization, but neither stemming, nor feature selection, nor other methods. Like with the preprocessing, we also chose the feature extraction techniques based on the literature and therefore we used term frequency (TF) on Naive Bayes and term frequency-inverse document frequency (TF-IDF) on all other algorithms. [28]

The feature extraction was implemented using the CountVectorizer and the TfidfVectorizer of the python library scikit-learn [42]. To preprocess the text, the built-in functionality of CountVectorizer and TfidfVectorizer was used.

5.3 Evaluation Metrics

Evaluation metrics are used to evaluate and compare machine learning models. In our case we used the evaluation metrics to verify the re-implemented approaches, explore baseline metrics, and compare all the prototypes (baseline) with our approach as well as all intermediate steps of our approach.

Traditional evaluation metrics for machine learning models are calculated using a labeled evaluation set in combination with predictions made on the evaluation set by a model which should be evaluated. By comparing the labels with the predictions we can retrieve some values, which are used to calculate the evaluation metrics. These metrics are described in the following.

- **True positive** (TP)
  Number of examples, where the model correctly predicted the positive class.
- **True negative** (TN)
  Number of examples, where the model correctly predicted the negative class.
- **False positive** (FP)
  Number of examples, where the model predicted the positive class, but the example is of the negative class.
- **False negative** (FN)
  Number of examples, where the model predicted the negative class, but the example is of the positive class.

Using these values the evaluation metrics used in this work can be calculated. The following list describes the evaluation metrics we used in this work.

- **Accuracy**
  The accuracy is the percentage of examples for which where the correct label was predicted. (Equation 5.1)

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

- **Precision**
  The precision is the percentage of examples, which the model got right out of the total number of examples that the model predicted for a given label. (Equation 5.2)

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall**
  The recall is the percentage of examples, which the model predicted for a given tag out of the total number of examples it should have predicted for that given tag. (Equation 5.3)

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- **F1-score**
  The F1-score is the harmonic mean of precision and recall. (Equation 5.4)

$$F1 = \frac{2 * recall * precision}{recall + precision} \tag{5.4}$$

In a PU setting, we normally don't have a fully labeled evaluation dataset, since the information about the negative class is missing. But, as mentioned in Section 5.1, we created the PU datasets in a way so that we knew the real labels. Through that, we were able to use the evaluation metrics presented above.

5.4 Experiment Framework

We conducted a multitude of experiments during our research, where we evaluated different PU approaches in a variety of PU scenarios. For that reason, we developed an experiment framework, which automated the process of setting up and conducting experiments. Figure 5.1 schematically shows the process of setting up and conducting experiments with the experiment framework.

First, the researcher configures the datasource, the γ parameter and the prototypes of the experiment. Afterward, the experiment framework creates the PU dataset (split into training set and evaluation set) and selects and initializes the prototypes based on the configurations of the researcher. In the next step, each prototype is trained using the training set of the previously created PU dataset. Then, each trained model makes predictions on the evaluation set. Based on the predictions subsequently, evaluation results are calculated. In the end, the trained models, the evaluation results and different plots for further analysis are exported.

5.5 Summary

In this chapter we described the design of the experiments we conducted. Firstly, this includes how the PU datasets we used for training and evaluation were created. Secondly, the prototypes we implemented and how we verified their correctness. Finally, the evaluation metrics used to analyze and compare the performance of the approaches as well as the experiment framework we built to conduct the experiments and collect the results.

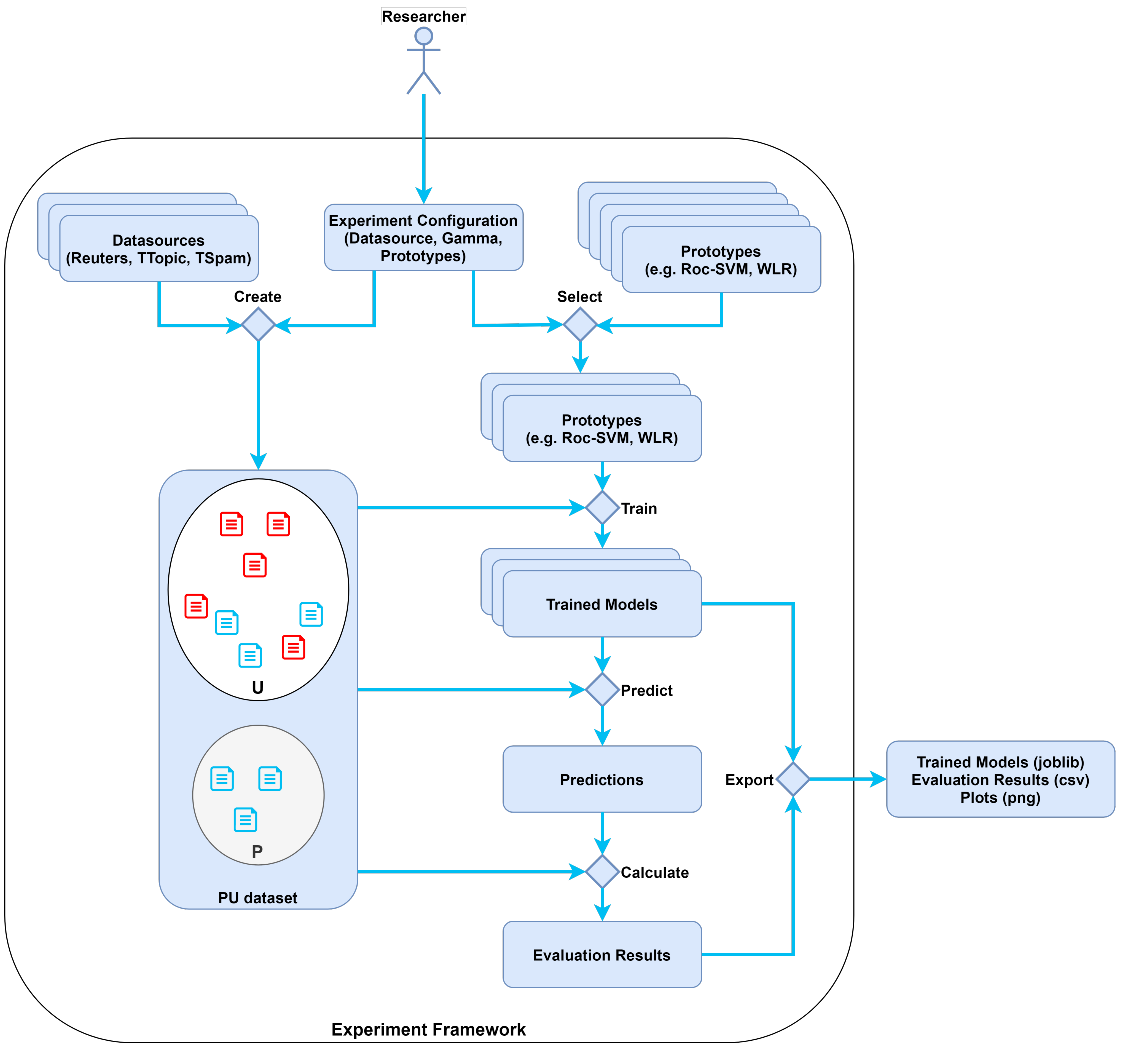


Figure 5.1: Schematic illustration showing the process/architecture of the experiment framework.

CHAPTER 6

Evaluation and Results

We present the evaluation results of our approach. First, we show the results of the prototype verification and the subsequent exploration of baseline metrics, in Section 6.1 and Section 6.2. In Section 6.3, we analyze the modifications and optimizations that we made to Weighted Logistic Regression. In the end, in Section 6.4, we evaluate our approach using the domain-optimized Weighted Logistic Regression algorithm, TWLR in combination with SVM.

6.1 Verification

In the first stages of the research and the experiments, we re-implemented and verified a variety of PU learning approaches. In the upcoming sections, we describe the conducted experiments to verify the prototypes in more detail. This includes the experiment setup, the experiment results as well as a concluding discussion.

6.1.1 Experiment Setup

We verified the correctness with different datasets generated from the Reuters-21578 data source using different noise settings (γ) as described in Section 5.1. More precisely, we used the "earn" category as the positive class and created two PU datasets from it. The two datasets differ in the noise setting, or rather the number of labeled examples controlled by γ. One was created setting γ to 0.15 (15%) and the other setting γ to 0.45 (45%).

In the experiments, we verified the different re-implemented approaches. For that reason, all the prototypes described in Section 5.2 were part of the experiments. This includes the pure NB classifier, the 6 variations of 2-step approaches (NB-SVM, Roc-SVM, NB-SVM-I, Roc-SVM-I, KNN-SVM, KNN-SVM-I) as well as the 3 biased approaches (Bagging DT, Bagging SVM, Weighted Logistic Regression). In the classifiers used in the re-implemented prototypes, we almost only used default parameter (provided by the scikit-learn library) since this is often the case in the literature. The Bagging PU approaches each used 10 estimators.

6.1.2 Experiment Results

Table 6.1 shows the evaluation results using the two PU datasets. The table contains the accuracy and the F1-scores achieved by each of the prototypes in the respective PU setting.

	$\gamma = 15\%$		$\gamma = 45\%$	
Classifier	**Accuracy**	**F1-score**	**Accuracy**	**F1-score**
NB	**0.9134**	**0.8854**	**0.9490**	**0.9212**
NB-SVM	0.8653	0.8020	**0.9554**	**0.9422**
Roc-SVM	**0.9050**	**0.8388**	0.9212	0.8930
NB-SVM-I	0.8552	0.7840	0.9534	0.9395
Roc-SVM-I	0.8737	0.8165	0.9202	0.8915
KNN-SVM	0.5982	0.0276	0.7611	0.5867
KNN-SVM-I	0.5982	0.0276	0.7574	0.5771
Bagging DT	0.8928	0.8570	0.8975	0.8667
Bagging SVM	**0.9373**	**0.9174**	**0.9534**	**0.9403**
WLR	**0.9420**	**0.9235**	**0.9735**	**0.9667**

Table 6.1: Evaluation results of experiments to verify the re-implemented approaches in form of Accuracy and F1-scores achieved by the different prototypes using different PU datasets generated from the Reuters-21578 (earn) data source.

6.1.3 Findings

Li et al. [27] used the same settings and two similar algorithms in their experiments, namely the pure NB algorithm and the 2-step approach Roc-SVM. In the setting of γ set to 15%, they achieved F1-scores of 0.910 and 0.858 for pure NB and Roc-SVM. In the 45% setting they achieved F1-scores of 0.924 and 0.886. In the setting of γ set to 15% their results only differ by 0.025 percentage points for pure NB and 0.019 percentage points for Roc-SVM compared to our results. In the 45% setting the difference is only 0.003 percentage points for pure NB and 0.007 percentage points for Roc-SVM. The results differ since the exact experiment setup, preprocessing steps and parameter-settings are not fully comprehensible from the paper. Since similar results are achieved, we assume the correctness of our prototypes for NB and Roc-SVM. Thus, we assume that all other 2-step approaches are correct, since we used the same 2-step classifier developed by us and just replaced the algorithms used in it.

There are no trustworthy evaluation results using Reuters-21578 for biased learning approaches. For that reason, we furthermore assumed that the biased learning algorithms (Bagging PU and Weighted Logistic Regression) are correct since they perform well and outperform the 2-step approaches, which agrees with the literature. [9, 28]

6.2 Baselining

After we verified the correctness of the re-implemented approaches, we explored baseline metrics for PU learning in the domain-specific setting, or more precise for topic detection and spam detection in the domain of Twitter. In the upcoming sections, we describe the conducted experiments to explore baseline metrics in more detail. This includes the experiment setup, the experiment results as well as a concluding discussion.

6.2.1 Experiment Setup

To explore baseline metrics, we conducted a variety of experiments simulating a multitude of PU settings using the two Twitter data sources, TTopic and TSpam described in Section 5.1.2 and Section 5.1.3. From each data source, we generated five different PU datasets, simulating different noise settings, controlled by the configurable γ. To be more precise, we conducted experiments using γ set to 0.15 (15%), 0.25 (25%), 0.5 (50%), 0.75 (75%) and 1 (100%).

Like in the experiments conducted to verify the re-implemented approaches (Section 6.1), here as well all the prototypes described in Section 5.2 were part of the experiments. This includes the pure NB classifier, the 6 variations of 2-step approaches (NB-SVM, Roc-SVM, NB-SVM-I, Roc-SVM-I, KNN-SVM, KNN-SVM-I) as well as the 3 biased approaches (Bagging DT, Bagging SVM, Weighted Logistic Regression). In the classifiers used in the re-implemented prototypes, we used default parameters (provided by the scikit-learn library) since this is often the case in the literature. The Bagging PU approaches used each 10 estimators.

6.2.2 Experiment Results

The experiment results are presented in Table 6.2. The table is split into two parts containing the results for the use cases topic detection and spam detection. Each part contains the F1-scores achieved by each of the prototypes in the specific PU setting as well as the average F1-score achieved by each prototype.

To better interpret and analyse the results, we visualized them in Figure 6.1 and 6.2. Figure 6.1 visualizes the average F1-scores reached by the algorithms. The x-axis shows the algorithms in the experiment and the y-axis depicts the average F1-score the algorithms reached. The different tasks are represented as separated bars in the chart.

Figure 6.2 visualizes the development curves of the F1-scores achieved by the different prototypes in line charts. Figure 6.2a visualizes the evaluation results for topic detection

Task	Classifier	15%	25%	50%	75%	100%	Avg
Topic	**NB**	0.0495	0.1651	0.7148	0.8605	0.9269	0.5434
	NB-SVM	0.0337	0.1768	0.7616	0.9223	0.9572	0.5703
	Roc-SVM	0.6785	0.7800	0.8985	0.9336	0.9571	**0.8495**
	NB-SVM-I	0.0337	0.1768	0.7590	0.9217	0.9576	0.5698
	Roc-SVM-I	0.6126	0.7616	0.8972	0.9332	0.9570	0.8323
	KNN-SVM	0.0331	0.1513	0.6164	0.8547	0.9567	0.5224
	KNN-SVM-I	0.0331	0.1513	0.6153	0.8547	0.9567	0.5222
	Bagging DT	0.8457	0.8544	0.8943	0.9146	0.9547	**0.8927**
	Bagging SVM	0.7919	0.8191	0.8718	0.8991	0.9569	0.8678
	WLR	0.8197	0.8833	0.9190	0.9391	0.9521	**0.9026**
Spam	**NB**	0.0756	0.1460	0.7547	0.8234	0.8528	0.5305
	NB-SVM	0.0300	0.0733	0.7295	0.8256	0.8590	0.5035
	Roc-SVM	0.5524	0.6091	0.7497	0.8138	0.8599	**0.7170**
	NB-SVM-I	0.0300	0.0733	0.7288	0.8260	0.8590	0.5034
	Roc-SVM-I	0.5524	0.6050	0.7501	0.8148	0.8598	0.7164
	KNN-SVM	0.0317	0.0686	0.6722	0.8098	0.8594	0.4883
	KNN-SVM-I	0.0317	0.0686	0.6696	0.8098	0.8594	0.4878
	Bagging DT	0.6872	0.7368	0.7534	0.7902	0.8219	0.7579
	Bagging SVM	0.7135	0.7528	0.8065	0.8238	0.8414	**0.7876**
	WLR	0.7568	0.7965	0.8234	0.8258	0.8256	**0.8056**

Table 6.2: Evaluation results of experiments to explore baseline metrics in the form of F1-scores achieved by the different prototypes using different PU datasets generated from the TTopic and TSpam data sources.

and Figure 6.2b visualizes the evaluation results for spam detection. The x-axes depict the γ-settings used to create the PU dataset. This means the lower the γ value is, the higher is the noise in the unlabeled set. The y-axes show the F1-score. Each prototype, or rather the evaluation results in the form of the F1-scores achieved by the prototype are represented as a line in the chart.

We only presented the evaluation results in form of F1-scores since this metric is very common in the literature of PU learning and presenting all other evaluation metrics presented in the previous chapter would be out of scope.

6.2.3 Findings

In the following list, we describe the observations we made and the conclusions we draw, by analyzing the experiment results.

- As expected, the results indicate that the overall performance of the algorithms is

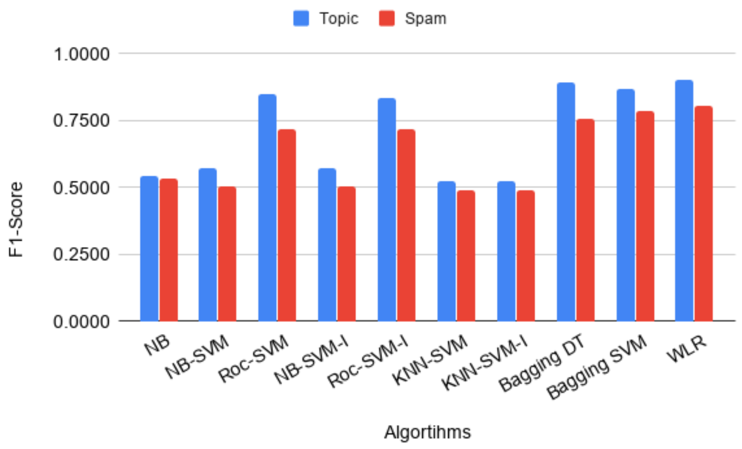
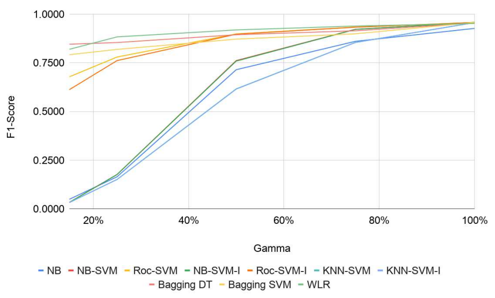
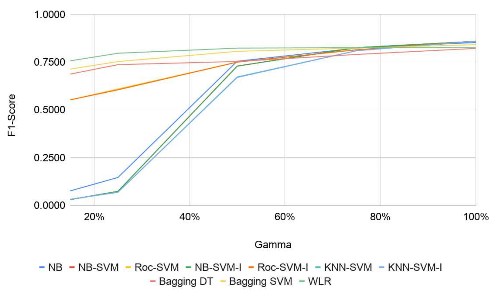


Figure 6.1: Bar chart showing the average F1-scores (y-axis) achieved by the different prototypes (x-axis) in the experiments to explore baseline metrics.



(a) Topic detection



(b) Spam detection

Figure 6.2: Line charts showing the development curves of the F1-score (y-axes) achieved by the different prototypes on different noise-settings (x-axes) in the experiments to explore baseline metrics.

worse in the domain-specific setting, compared to the experiments using Reuters-21578 (Section 6.1). This is probably the case because the examples in the Reuters-21578 datasets contain more and high-quality text and therefore the algorithms can better detect and learn the patterns of the texts. Twitter examples, on the other hand, are short text documents containing text with many spelling-errors and slang, which makes the learning process harder.

- The results also indicate that the algorithms perform better in topic detection than in spam detection. In other words, the topic can be better determined or rather detected through the text and the words it contains, compared to detecting spam tweets. This was to be expected since spam detection is an ambiguous task and it is not always clear if a tweet is spam or not. Also, spam accounts try to mask spam tweets as normal tweets to make detection harder.

- The results suggest that approaches which are represented in the literature, such as Roc-SVM, Roc-SVM-I, Bagging PU or Weighted Logistic Regression, outperform the additional prototypes we implemented.
- We can see that the performance of 2-step approaches is very dependent on the first stage and how robust the algorithm chosen for the first stage is against noise. Thus, the results show that biased learning approaches are more robust against noise and outperform 2-step approaches, except when no noise is present Roc-SVM performs slightly better.
- The best and most constant result is delivered by Weighted Logistic Regression, even though we used a very simple weighting approach. That is also why we used an optimized version of Weighted Logistic Regression, which we evaluate in Section 6.3, in our approach.
- From these results, we chose the evaluation metrics of the best performing 2-step approach as well as the best performing biased learning approach as baseline metrics in the experiments evaluating our approach. Thus, we used the evaluation results of Roc-SVM and Weighted Logistic Regression as baseline metrics.
- The experiments provide new insights into the performance of PU learning in domain-specific settings, or more precisely in the Twitter domain.

6.3 Twitter Weighted Logistic Regression (TWLR)

We present the evaluation results of the Twitter Weighted Logistic Regression (TWLR) algorithm, which is a Weighted Logistic Regression algorithm optimized for the domain of Twitter. We evaluate it since it is an essential part of our approach. In the upcoming sections, we describe the conducted experiments to evaluate TWLR in more detail. This includes the experiment setup, the experiment results as well as a concluding discussion.

6.3.1 Experiment Setup

To compare the evaluation results with the baseline metrics, we used the same PU datasets as in the baseline experiments described in the previous section. This means, we conducted experiments using PU datasets, generated from the data sources TTopic and TSpam described in Section 5.1.2 and Section 5.1.3. For each data source, we used five different PU datasets, simulating different noise settings. To be more precise, the PU dataset were generated using γ set to 0.15 (15%), 0.25 (25%), 0.5 (50%), 0.75 (75%) and 1 (100%).

In the experiments, we looked at different variations and modifications of TWLR, including label frequency estimation (LFE), sample weighting (SW) and feature incorporation

(FI). As already mentioned in Section 4.2.4, the features useful in the respective use cases of topic detection and spam detection differ from each other. For that reason, the approaches used in the experiments in the respective use cases, differ from each other. In Table 6.3 we present the approaches, which were part of the conducted experiments. This includes the features used as well as the method to incorporate them for each of the approaches. In the Logistic Regression classifier used in TWLR, we only used default parameters (provided by the scikit-learn library) since this is often the case in the literature.

Task	Classifier	SW	FI
Topic	**TWLR-LFE**	-	F1.1
	TWLR-LFE-SW	F3.2	F1.1
	TWLR-LFE-FI	-	F1.1, F3.1
	TWLR-LFE-SW-FI	F3.2	F1.1, F3.1
Spam	**TWLR-LFE**	-	F1.1
	TWLR-LFE-LWF	-	F1.1, F2.1.1, F2.1.3, F2.2.1, F2.2.2
	TWLR-LFE-FI	-	F1.1, F2.1.1, F2.1.3, F2.2.1, F2.2.2, F3.1

Table 6.3: List of TWLR variations evaluated in the experiments, including the corresponding Twitter features incorporated for each variation. The Twitter features are described in Section 4.2.4.

6.3.2 Experiment Results

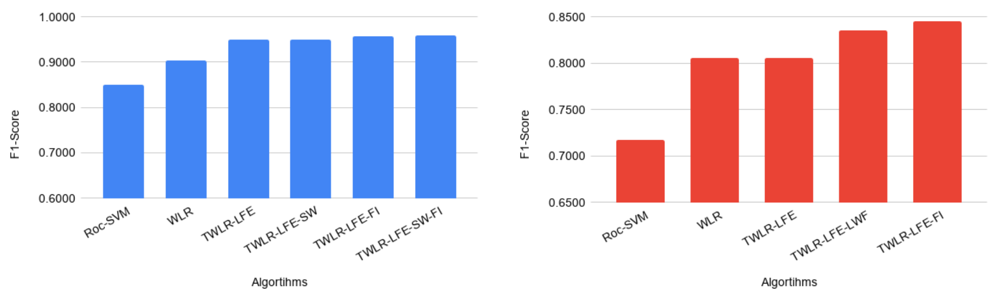
The experiment results are presented in Table 6.4. The table is again split into two parts containing the results for the respective use cases topic detection and spam detection. Each part contains the F1-scores achieved by the different modifications of the TWLR algorithm as well as the baseline algorithms in the specific PU setting as well as the average F1-score achieved by each algorithm.

Like before, to better interpret and analyse the results, we visualized them in Figure 6.3 and 6.4. Figure 6.3 visualizes the average F1-scores reached by the algorithms. Figure 6.3a shows the F1-scores achieved by the TWLR variations together with the baseline metrics on topic detection. Figure 6.3b again compares the F1-scores achieved by the TWLR variations together with the baseline metrics but on spam detection. The x-axes show the algorithms in the experiment and the y-axes depict the average F1-score the algorithms reached.

Figure 6.4 visualizes the development curves of the F1-scores achieved by the different prototypes in line charts. Figure 6.4a visualizes the evaluation results for topic detection and Figure 6.4b visualizes the evaluation results for spam detection. The x-axes depict the γ-settings used to create the PU dataset. This means the lower the γ value is, the

Task	Classifier	15%	25%	50%	75%	100%	Avg
Topic	**Roc-SVM**	0.6785	0.7800	0.8985	0.9336	0.9571	0.8495
	WLR	0.8197	0.8833	0.9190	0.9391	0.9521	0.9026
	TWLR-LFE	0.9320	0.9454	0.9558	0.9563	0.9576	0.9494
	TWLR-LFE-SW	0.9341	0.9462	0.9544	0.9569	0.9557	0.9495
	TWLR-LFE-FI	0.9422	0.9515	0.9583	0.9637	0.9662	0.9564
	TWLR-LFE-SW-FI	0.9500	0.9524	0.9599	0.9642	0.9660	**0.9585**
Spam	**Roc-SVM**	0.5524	0.6091	0.7497	0.8138	0.8599	0.7170
	WLR	0.7568	0.7965	0.8234	0.8258	0.8256	0.8056
	TWLR-LFE	0.7881	0.7875	0.8045	0.8178	0.8290	0.8054
	TWLR-LFE-LWF	0.8268	0.8354	0.8384	0.8484	0.8247	0.8348
	TWLR-LFE-FI	0.8373	0.8407	0.8451	0.8544	0.8468	**0.8449**

Table 6.4: Evaluation results of experiments to evaluate TWLR in the form of F1-scores achieved by the different TWLR variations using different PU datasets generated from the TTopic and TSpam data sources.



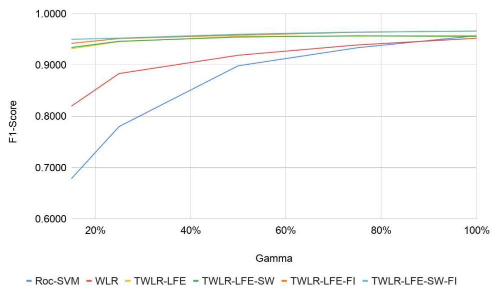
(a) Topic detection w/ baseline

(b) Spam detection w/ baseline

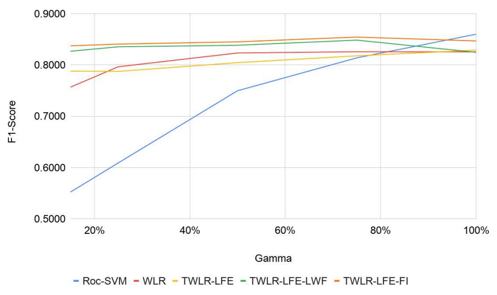
Figure 6.3: Bar charts showing the average F1-scores (y-axes) achieved by the different TWLR variations (x-axes) in the experiments to evaluate TWLR.

higher is the noise in the unlabeled set. The y-axes show the F1-score. Each prototype, or rather the evaluation results in the form of F1-scores achieved by the prototype are represented as a line in the chart.

We only presented the evaluation results in form of F1-scores since this metric is very common in the literature of PU learning and presenting all other evaluation metrics presented in the previous chapter would be out of scope.



(a) Topic detection w/ baseline



(b) Spam detection w/ baseline

Figure 6.4: Line charts showing the development curves of the F1-score (y-axes) achieved by the different TWLR variations on different noise-settings (x-axes) in the experiments to evaluate TWLR.

6.3.3 Findings

In the following list, we describe the observations we made and the conclusions we draw, by analyzing the experiment results.

- The results indicate that through the incorporation of the label frequency estimation the algorithm gets very robust and constant over all noise settings. By additionally incorporating features the approach gets even further robust and the overall performance increases. As a result, all the TWLR variations evaluated outperform the baseline algorithms and consequently all other prototypes.
- Our experiments have shown that incorporating sample weights in the form of user-scores and URL-score, which are simple occurrence counts derived from the positive set, only increases the performance in topic detection. The problem in spam detection is that the identification of probable positives is more ambiguous than in topic detection and simple features like the user-score and URL-score are not suitable for spam detection. For that reason, the best performing approach in topic detection uses a combination of sample weighting (SW) and feature incorporation (FI) and the best performing approach in spam detection only uses feature incorporation (FI).
- The results also indicate that the statistical metrics (F2) are essential in spam detection and increase the performance of the approach significantly. In topic detection, the statistical metrics are not significant. However, the usage of a wordlist/blacklist (F3.1) increases the performance in both use cases. Nevertheless, we suggest dispensing with the wordlist/blacklist feature when using the algorithm standalone since wordlists/blacklists carry the risk of overfitting.
- In our approach we use TWLR in the first stage of a 2-step approach and therefore only to identify reliable negatives and reliable positives in the training set, but

not as the final classifier. Through that, the problem of overfitting, as well as fast outdating, is minimized. Thus, we can use the best performing version of the algorithm in our approach. This means, in the case of topic detection we used TWLR-LFE-SW-FI in our approach and in the case of spam detection we used TWLR-LFE-FI in our approach.

6.4 TWLR-SVM

In this section, we present the evaluation results of our approach TWLR-SVM. In the upcoming sections, we describe the conducted experiments to evaluate TWLR-SVM in more detail. This includes the experiment setup, the experiment results as well as a concluding discussion.

6.4.1 Experiment Setup

Like before, to compare the evaluation results with the baseline metrics as well as with the results of the TWLR algorithm, we used the same datasets as in the baseline experiments described in the previous sections. This means, we conducted experiments using PU datasets, generated from the data sources TTopic and TSpam described in Section 5.1.2 and Section 5.1.3. For each data source, we used five different PU datasets, simulating different noise settings. To be more precise, the PU dataset were generated using γ set to 0.15 (15%), 0.25 (25%), 0.5 (50%), 0.75 (75%) and 1 (100%).

In the experiment, we looked at the approach TWLR-SVM, including the incorporation of different features. As already mentioned in Section 4.2.4, the features useful in the respective use cases of topic detection and spam detection differ from each other. For that reason, the approaches used in the experiments in the respective use cases differ from each other. In Table 6.5 we present the approaches, which were part of the conducted experiments. This includes the variation of TWLR used in the first stage of the approach as well as features incorporated in the second stage or rather in the SVM classifier. In the Logistic Regression and SVM classifier used in TWLR-SVM, we only used default parameters (provided by the scikit-learn library) since this is often the case in the literature.

6.4.2 Experiment Results

The experiment results are presented in Table 6.6. The table is again split into two parts containing the results for the respective use cases topic detection and spam detection. Each part contains the F1-scores achieved by the different TWLR-SVM variations as well as the baseline algorithms in the specific PU setting as well as the average F1-score achieved by each algorithm.

Like before, to better interpret and analyse the results, we visualize them in Figure 6.5 and 6.6. Figure 6.5 visualizes the average F1-scores reached by the algorithms. Figure 6.5a shows the F1-scores achieved by the TWLR-SVM variations together with the

Task	Classifier	Step 1 (TWLR)	Step 2 (SVM)
Topic	**TWLR-SVM**	TWLR-LFE-SW-FI	F1.1
	TWLR-SVM-AF	TWLR-LFE-SW-FI	F1.1, F3.1
Spam	**TWLR-SVM**	TWLR-LFE-FI	F1.1
	TWLR-SVM-LWF	TWLR-LFE-FI	F1.1, F2.1.1, F2.1.3, F2.2.1, F2.2.2
	TWLR-SVM-AF	TWLR-LFE-FI	F1.1, F2.1.1, F2.1.3, F2.2.1, F2.2.2, F3.1

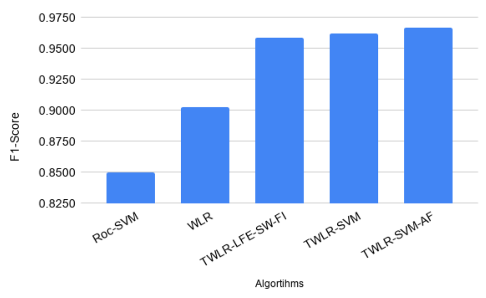
Table 6.5: List of TWLR-SVM variations evaluated in the experiments, including the respective TWLR variation used in the first stage and the respective Twitter features incorporated in the second stage for each variation. The Twitter features are described in Section 4.2.4.

Task	Classifier	15%	25%	50%	75%	100%	Avg
Topic	**Roc-SVM**	0.6785	0.7800	0.8985	0.9336	0.9571	0.8495
	WLR	0.8197	0.8833	0.9190	0.9391	0.9521	0.9026
	TWLR-LFE-SW-FI	0.9500	0.9524	0.9599	0.9642	0.9660	**0.9585**
	TWLR-SVM	0.9464	0.9541	0.9624	0.9715	0.9737	**0.9616**
	TWLR-SVM-AF	0.9506	0.9608	0.9680	0.9750	0.9769	**0.9663**
Spam	**Roc-SVM**	0.5524	0.6091	0.7497	0.8138	0.8599	0.7170
	WLR	0.7568	0.7965	0.8234	0.8258	0.8256	0.8056
	TWLR-LFE-FI	0.8373	0.8407	0.8451	0.8544	0.8468	0.8449
	TWLR-SVM	0.8085	0.8287	0.8388	0.8465	0.8584	0.8362
	TWLR-SVM-LWF	0.8334	0.8437	0.8496	0.8545	0.8554	**0.8473**
	TWLR-SVM-AF	0.8372	0.8482	0.8496	0.8548	0.8573	**0.8494**

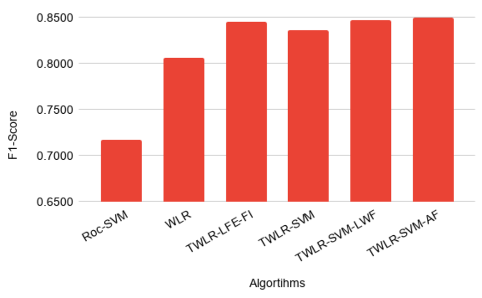
Table 6.6: Evaluation results of experiments to evaluate TWLR-SVM in the form of F1-scores achieved by the different TWLR-SVM variations using different PU datasets generated from the TTopic and TSpam data sources.

baseline metrics on topic detection. Figure 6.5b again compares the F1-scores achieved by the TWLR-SVM variations together with the baseline metrics but on spam detection. The x-axes show the algorithms in the experiment and the y-axes depict the average F1-score the algorithms reached.

Figure 6.6 visualizes the development curves of the F1-scores achieved by the different prototypes in line charts. Figure 6.6a visualizes the evaluation results for topic detection and Figure 6.6b visualizes the evaluation results for spam detection. The x-axes depict the γ-settings used to create the PU dataset. This means the lower the γ value is, the higher is the noise in the unlabeled set. The y-axes show the F1-score. Each prototype,



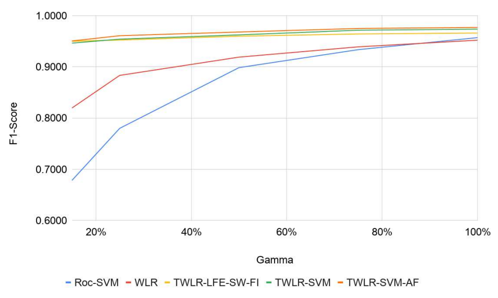
(a) Topic detection w/ baseline



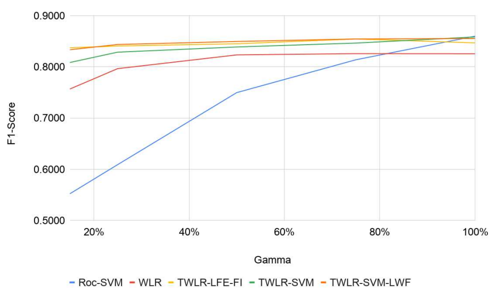
(b) Spam detection w/ baseline

Figure 6.5: Bar charts showing the average F1-scores (y-axes) achieved by the different TWLR-SVM variations (x-axes) in the experiments to evaluate TWLR-SVM.

or rather the evaluation results in the form of the F1-score achieved by the prototype are represented as a line in the chart.



(a) Topic detection w/ baseline



(b) Spam detection w/ baseline

Figure 6.6: Line charts showing the development curves of the F1-score (y-axes) achieved by the different TWLR-SVM variations on different noise-settings (x-axes) in the experiments to evaluate TWLR-SVM.

We only presented the evaluation results in form of F1-scores since this metric is very common in the literature of PU learning and presenting all other evaluation metrics presented in the previous chapter would be out of scope.

6.4.3 Findings

In the following list, we describe the observations we made and the conclusions we draw, by analyzing the experiment results.

- The results indicate that combining TWLR with SVM in a 2-step approach increases the performance compared to the standalone version of TWLR. In other words, our approach outperforms TWLR and consequently the baseline metrics as well as all other prototypes. Furthermore, through the combination in a 2-step approach, the final classifier is unbiased and the risk of overfitting is reduced.

- The results also indicate that additionally incorporating features into the SVM classifier further improves the performance. Like in TWLR the statistical metrics (F2) are essential in spam detection and increase the performance of the approach significantly. In topic detection, the statistical metrics are not significant. However, the usage of a wordlist/blacklist (F3.1) increases the performance in both use cases. Thus, the best performing approach in the case of topic detection is TWLR-SVM-AF and in the case of spam detection TWLR-SVM-AF.

- Nevertheless we suggest dispensing the wordlist/blacklist feature and other complex features from the second stage of the algorithm since, on the one hand, the impact on the performance is averagely only 0.0047 percentage points on topic detection and 0.0021 on spam detection. On the other hand, using only lightweight features reduces the computation costs and the risk of overfitting. This means, in the case of topic detection we suggest TWLR-SVM and in the case of spam detection we suggest TWLR-SVM-LWF.

6.5 Summary

In this chapter we presented, analyzed and discussed the results of the experiments we conducted. This includes the verification of the prototypes as well as the subsequent exploration of baseline metrics. But also, the results of the experiments analyzing the different optimization steps of our approach, including the analysis of TWLR and TWLR-SVM.

It is shown that from the re-implemented existing PU approaches, Weighted Logistic Regression performs the best on the Twitter domain, achieving average F1-scores of 90.26% on topic detection and 80.56% on spam detection. Our approach, TWLR-SVM, which combines domain-specific features with the best-suited algorithms, on the other hand, achieves average F1-scores of 96.63% on topic detection and 84.94% on spam detection.

This means it is shown that our approach, TWLR-SVM, outperforms the existing PU approaches in the Twitter domain in terms of average F1-Score by 6.37 percentage points on topic detection and 4.38 percentage points on spam detection.

CHAPTER 7

Conclusion

This research aimed to identify and develop a PU learning approach, or rather a binary text classification algorithm trained using only positive and unlabeled data, which is suitable and optimized for the Twitter domain.

Based on an explorative strategy, we investigated and developed a new domain-optimized PU approach. The optimizations include, on the one hand, feature engineering in the form of investigation, analysis, incorporation and evaluation of domain-specific features. On the other hand, they contain the combination of those features alongside with the best-suited algorithms in a new PU approach. The result is a new 2-step approach, called TWLR-SVM, which uses a domain-optimized version of Weighted Logistic Regression in the first step and Support Vector Machine (SVM) in the second step. To evaluate and verify the different optimization steps, we conducted quantitative experiments and analyzed their results.

The evaluation results indicate that our approach outperforms all existing PU approaches, represented in the literature, in the Twitter domain. To be more precise, the key findings can be summarized as follows:

- From the existing approaches from the literature, Roc-SVM was the best performing 2-step approach and Weighted Logistic Regression was the best performing biased approach in the Twitter domain. Roc-SVM achieved average F1-scores of 84.95% on topic detection and 71.70% on spam detection. Weighted Logistic Regression achieved average F1-scores of 90.26% on topic detection and 80.56% on spam detection.

- Through the incorporation of label frequency estimation (LFE) and domain-specific features identified and evaluated using feature engineering, the performance of Weighted Logistic Regression was optimized. As a result, the algorithm, called

TWLR, achieved average F1-scores of 95.85% on topic detection and 84.49% on spam detection. Thus, the optimized algorithm outperformed the existing approaches or rather the Weighted Logistic Regression prototype by 5.59 percentage points on topic detection and 3.93 percentage points on spam detection in terms of average F1-score.

- Furthermore, the work shows that domain-specific statistical metrics are essential in spam detection and increase the performance of the approach significantly. In topic detection, the statistical metrics are not significant. However, the usage of a wordlist/blacklist increases the performance in both use cases. To be more precise, in topic detection, we incorporated tweet content analysis using TF-IDF, simple features like a user-score and a URL-score, as well as more complex features like the wordlist/blacklist, to increase the performance of our approach. In spam detection, on the other side, we incorporated, besides the tweet content analysis using TF-IDF and the wordlist/blacklist, content-based metrics such as hashtag count, user-mention count or URL count, as well as account-based metrics such as follower count, status count or the lifetime of a user, to increase the performance of our approach.
- By combining the best performing TWLR classifier with a domain-optimized SVM classifier in a 2-step approach, the performance was further increased. Our final approach, TWLR-SVM, achieved average F1-scores of 95.85% on topic detection and 84.49% on spam detection. This means it is shown that our approach, TWLR-SVM, outperforms the existing PU approaches in the Twitter domain in terms of average F1-Score by 6.37 percentage points on topic detection and 4.38 percentage points on spam detection.

The research gives new insights into the performance of PU approaches in the Twitter domain. It shows that there is great potential for optimizing PU approaches in domain-specific settings.

CHAPTER 8

Future Work

During this research, we touched a variety of areas, which were out of scope in this thesis, but would be interesting to pursue in future research. Future studies could investigate other assumptions and use cases in the Twitter domain. Also, future research could be devoted to further optimize our approach, through development, incorporation and evaluation of further features or modification and extension of our approach. Our approach could be, for example, further modified by using modifications common in the literature of PU learning like an iterative second stage or by applying ensemble learning.

Ensemble TWLR-SVM

Since the first stage of TWLR-SVM, already performs very well standalone, the combination of TWLR with TWLR-SVM by applying ensemble learning would be an interesting topic for future research. The architecture of this approach is shown in Figure 8.1.

A possible implementation would be to convert the SVM model to a calibrated probabilistic model. Through that, it would be able to predict probabilities. Afterward, the TWLR model could be combined with the calibrated SVM model using a simple voting algorithm, which just calculates and returns the average over the predicted probabilities for an example.

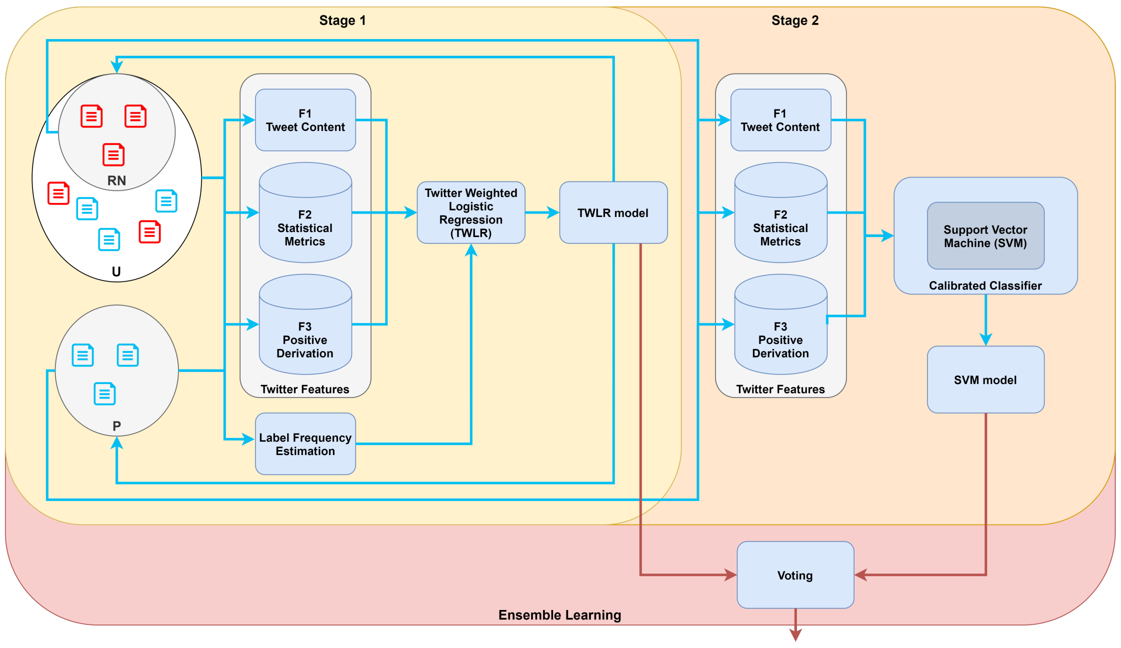


Figure 8.1: Schematic illustration showing the process/architecture of Ensemble TWLR-SVM. First, a TWLR model is trained on the positive (P) and the unlabeled set (U) to identify reliable negatives RN in the unlabeled set. Afterward, the reliable negatives RN and the positive examples P are used to train a SVM classifier. The TWLR model and the SVM model are then combined in an ensemble using a simple voting algorithm.

List of Figures

List of Tables

List of Algorithms

Acronyms

AI artifical intelligence. 8–10, 79

ANN Artificial Neural Network. 17, 18, 79

API application programming interface. 2, 38, 40, 44, 47, 49–51

CBOW Continuous Bag-of-Words. 25

DL deep learning. 8–10, 17, 79

FI feature incorporation. 66, 67, 69

IDF inverse document frequency. 5, 24

kNN k-Nearest Neighbor. 14, 20, 54

LFE label frequency estimation. 31, 34, 66, 69, 75

ML machine learning. ix, xi, 8–12, 14, 79

NB Naive Bayes. 13, 20, 54, 61–63

NLP natural language processing. 8–11, 79

NLTK Natural Language Toolkit. 49

PU positive-unlabeled. ix, xi, xiii, 1–5, 7–9, 11, 13, 16–23, 26–29, 31–33, 35, 36, 41, 45, 47–52, 54–56, 58, 61–64, 66–68, 70–73, 75–77, 79, 81

RESVM Robust Ensemble SVM. 22

Roc Rocchio. 13, 14, 54, 62, 66, 75

SCAR selected completely at random. 32

SVM Support Vector Machine. ix, xi, xiii, xiv, 3, 4, 13, 15, 16, 20–22, 31, 43–46, 54, 61, 62, 66, 70–73, 75–81, 83

SW sample weighting. 66, 67, 69

TF term frequency. 5, 24, 56

TF-IDF term frequency-inverse document frequency. 7, 24, 37, 40, 56, 76

TWLR Twitter Weighted Logistic Regression. ix, xi, xiii, xiv, 3, 4, 31, 33–35, 37, 39, 41, 43–45, 61, 66–73, 75–81, 83

Bibliography

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[2] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[3] Jessa Bekker and Jesse Davis. Learning from positive and unlabeled data: A survey. *arXiv preprint arXiv:1811.04820*, 2018.

[4] Rich Caruana and Alexandru Niculescu-mizil. An empirical comparison of supervised learning algorithms. In *In Proc. 23 rd Intl. Conf. Machine learning (ICML'06*, pages 161–168, 2006.

[5] Sneha Chaudhari and Shirish Shevade. Learning from positive and unlabelled examples using maximum margin clustering. In *Proceedings of the 19th International Conference on Neural Information Processing - Volume Part III*, ICONIP'12, page 465–473, Berlin, Heidelberg, 2012. Springer-Verlag.

[6] Weiling Chen, Chai Kiat Yeo, Chiew Tong Lau, and Bu Sung Lee. A study on real-time low-quality content detection on twitter from the users' perspective. *PLOS ONE*, 12(8):1–22, 08 2017.

[7] Marc Claesen, Frank De Smet, Johan A.K. Suykens, and Bart De Moor. A robust ensemble approach to learn from positive and unlabeled data using svm base models. *Neurocomputing*, 160:73–84, Jul 2015.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[9] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 213–220, New York, NY, USA, 2008. Association for Computing Machinery.

[10] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[11] Gabriel Pui Cheong Fung, J. X. Yu, Hongjun Lu, and P. S. Yu. Text classification without negative examples revisit. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):6–20, Jan 2006.

[12] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, page 278, USA, 1995. IEEE Computer Society.

[13] Cho-Jui Hsieh, Nagarajan Natarajan, and Inderjit S. Dhillon. PU learning for matrix completion. *CoRR*, abs/1411.6081, 2014.

[14] Dino Ienco and Ruggero G. Pensa. Positive and unlabeled learning in categorical data. *Neurocomputing*, 196:113 – 124, 2016.

[15] Georgiana Ifrim, Gökhan Bakir, and Gerhard Weikum. Fast logistic regression for text categorization with variable-length n-grams. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 354–362, 2008.

[16] Shengyi Jiang, Guansong Pang, Meiling Wu, and Limin Kuang. An improved k-nearest-neighbor algorithm for text categorization. *Expert Systems with Applications*, 39(1):1503 – 1509, 2012.

[17] Abdullah Talha Kabakus and Resul Kara. A survey of spam detection methods on twitter. *International Journal of Advanced Computer Science and Applications*, 8, 03 2017.

[18] Ting Ke, Ling Jing, Hui Lv, Lidong Zhang, and Yaping Hu. Global and local learning from positive and unlabeled examples. *Applied Intelligence*, 48:1–20, 11 2017.

[19] Ting Ke, Hui Lv, Mingjing Sun, and Lidong Zhang. A biased least squares support vector machine based on mahalanobis distance for pu learning. *Physica A: Statistical Mechanics and its Applications*, 509:422 – 438, 2018.

[20] Ting Ke, Bing Yang, Ling Zhen, Junyan Tan, Yi Li, and Ling Jing. Building high-performance classifiers using positive and unlabeled examples for text classification. In Jun Wang, Gary G. Yen, and Marios M. Polycarpou, editors, *Advances in Neural Networks – ISNN 2012*, pages 187–195, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[21] Ryuichi Kiryo, Gang Niu, Marthinus C du Plessis, and Masashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1675–1685. Curran Associates, Inc., 2017.

[22] Wee Sun Lee and Bing Liu. Learning with positive and unlabeled examples using weighted logistic regression. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 448–455. AAAI Press, 2003.

[23] David D. Lewis. Reuters-21578 text categorization test collection, distribution 1.0. 1997.

[24] Tianyu Li, Chien-Chih Wang, Yukun Ma, Patricia Ortal, Qifang Zhao, Björn Stenger, and Yu Hirate. Learning classifiers on positive and unlabeled data with policy gradient. *2019 IEEE International Conference on Data Mining (ICDM)*, pages 399–408, 2019.

[25] Xiao-Li Li and Bing Liu. Learning from positive and unlabeled examples with different data distributions. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 218–229, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[26] Xiao-Li Li, Bing Liu, and See-Kiong Ng. Negative training data can be harmful to text classification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 218–228, Cambridge, MA, October 2010. Association for Computational Linguistics.

[27] Xiaoli Li and Bing Liu. Learning to classify texts using positive and unlabeled data. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 587–592, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[28] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu. Building text classifiers using positive and unlabeled examples. In *Third IEEE International Conference on Data Mining*, pages 179–186, Nov 2003.

[29] Bing Liu, Wee Sun Lee, Philip S. Yu, and Xiaoli Li. Partially supervised classification of text documents. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, page 387–394, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[30] Lu Liu and Tao Peng. Clustering-based method for positive and unlabeled text categorization enhanced by improved tfidf. *Journal of Information Science and Engineering*, 30:1463–1481, 09 2014.

[31] Zhigang Liu, Wenzhong Shi, Deren Li, and Qianqing Qin. Partially supervised classification – based on weighted unlabeled samples support vector machine. In Xue Li, Shuliang Wang, and Zhao Yang Dong, editors, *Advanced Data Mining and Applications*, pages 118–129, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[32] Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[33] David M. Magerman. Statistical decision-tree models for parsing. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, Massachusetts, USA, June 1995. Association for Computational Linguistics.

[34] M. E. Maron. Automatic indexing: An experimental inquiry. *J. ACM*, 8(3):404–417, July 1961.

[35] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[36] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. Improving lda topic models for microblogs via tweet pooling and automatic labeling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, page 889–892, New York, NY, USA, 2013. Association for Computing Machinery.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[38] Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. Twitter spammer detection using data stream clustering. *Information Sciences*, 260:64–73, 2014.

[39] Fantine Mordelet and Jean-Philippe Vert. A bagging svm to learn from positive and unlabeled examples, 2010.

[40] Nils J Nilsson and Nils Johan Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[41] P Russel Norvig and S Artificial Intelligence. *A modern approach*. Prentice Hall, 2002.

[42] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[43] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The Smart retrieval system - experiments in automatic document processing*, pages 313–323. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[44] Songbo Tan. Neighbor-weighted k-nearest neighbor for unbalanced text orpus. *Expert Systems with Applications*, 28:667–671, 05 2005.

[45] Alex Hai Wang. Don't follow me: Spam detection in twitter. In *2010 international conference on security and cryptography (SECRYPT)*, pages 1–10. IEEE, 2010.

[46] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. pages 1445–1456, 05 2013.

[47] Hwanjo Yu, Jiawei Han, and Kevin Chang. Pebl: Positive example based learning for web page classification using svm. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2002.

[48] Shuang Yu and Chunping Li. Pe-puc: A graph based pu-learning approach for text classification. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 574–584. Springer, 2007.

[49] B. Zhang and W. Zuo. Learning from positive and unlabeled examples: A survey. In *2008 International Symposiums on Information Processing*, pages 650–654, May 2008.