A SIMILARITY BASED OVERSAMPLING METHOD FOR MULTI-LABEL IMBALANCED TEXT DATA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İSMAIL HAKKI KARAMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2022

Approval of the thesis:

A SIMILARITY BASED OVERSAMPLING METHOD FOR MULTI-LABEL IMBALANCED TEXT DATA

submitted by **İSMAIL HAKKI KARAMAN** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Esra Karasakal
Head of Department, **Industrial Engineering** _____

Prof. Dr. Gülser Köksal
Supervisor, **Industrial Engineering, METU** _____

Assoc. Prof. Dr. Levent Erişkin
Co-supervisor, **Industrial Engineering, NDU** _____

Examining Committee Members:

Prof. Dr. Serhan Duran
Industrial Engineering, METU _____

Prof. Dr. Gülser Köksal
Industrial Engineering, METU _____

Assoc. Prof. Dr. Seçil Savaşaneril
Industrial Engineering, METU _____

Assoc. Prof. Dr. Levent Erişkin
Industrial Engineering, NDU _____

Prof. Dr. İnci Batmaz
Statistics, METU _____

Date: 01.09.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: İsmail Hakkı Karaman

Signature :

ABSTRACT

A SIMILARITY BASED OVERSAMPLING METHOD FOR MULTI-LABEL IMBALANCED TEXT DATA

Karaman, İsmail Hakkı
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Gülser Köksal
Co-Supervisor: Assoc. Prof. Dr. Levent Erişkin

September 2022, 111 pages

In the real world, while the amount of data increases, it is not easy to find labeled data for Machine Learning projects, because of the compelling cost and effort requirements for labeling data. Also, most Machine Learning projects, especially multi-label classification problems, struggle with the data imbalance problem. In these problems, some classes, even, do not have enough data to train a classifier. In this study, an oversampling method for multi-label text classification problems is developed and studied to solve performance problems arising from the data imbalance. The proposed method finds new samples from unlabeled data by utilizing similarities between instances. It finds similar instances for a class from the unlabeled set and checks for improvement to see the effect on the performance of these instances. The unlabeled set is searched iteratively and the instances that assist the performance improvement are added to the labeled set. The experiments show that our method works well and the performance of the classifier is improved after oversampling.

Keywords: oversampling, text classification, multi-label classification, imbalanced classification, text similarity

ÖZ

ÇOK ETİKETLİ DENGESİZ METİN VERİ KÜMELERİ İÇİN BENZERLİĞE DAYALI BİR AŞKIN ÖRNEKLEME YÖNTEMİ

Karaman, İsmail Hakkı
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Gülser Köksal
Ortak Tez Yöneticisi: Doç. Dr. Levent Erişkin

Eylül 2022 , 111 sayfa

Dünyamızda veri miktarı artmasına rağmen veri etiketlemenin maliyeti ve zorluğu nedeniyle etiketlenmiş veri bulmak kolay değildir. Ayrıca makine öğrenmesi projelerinde, özellikle çok etiketli sınıflandırma problemlerinde, veri dengesizliği nedeniyle başarım sorunlarıyla karşılaşılmaktadır. Çok etiketli sınıflandırma problemlerinde bazı sınıflar için bir sınıflandırıcı eğitmek için bile yeterli veri olmayabilir. Bu çalışmada çok etiketli sınıflandırma problemlerindeki başarım problemlerini çözmek için bir aşkın örnekleme yöntemi geliştirilmiştir. Önerilen yöntem etiketsiz veri kümesinden benzerlik yardımıyla yeni örnekler bulur ve bu örnekler, başarımı iyileştirmesi halinde etiketli sınıfa dahil edilir. Etiketsiz küme tekrarlı olarak taranır ve başarımı iyileştiren örnekler etiketli kümeye eklenerek bu sınıf genişletilir. Yapılan denemelerde algoritma başarımının iyileştiği gözlemlenmiştir. Önerilen yöntem ile insan çabası gerekmeden veri etiketlemenin mümkün olacağı düşünülmektedir.

Anahtar Kelimeler: aşkın örnekleme, metin sınıflandırma, çok etiketli sınıflandırma, dengesiz veri sınıflandırma, metin benzerliği

to my venerable father...

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ML	Machine Learning
NLP	Natural Language Processing
SVM	Support Vector Machine
DL	Deep Learning
SMOTE	Synthetic Minority Oversampling Technique

LIST OF SYMBOLS

ρ	Performance factor
d	Distance score
f_i	Similarity factor for class *i*
m	Number of classes
n	Total number of instances
n_i	Number of instances for class *i*
p_0	Initial performance measure
p_1	Final performance measure
pc_i	Probability of belonging to class c_i
r	Balance ratio
s	Similarity score
s_i	Similarity score for class *i*
t_i	Output class of instance *i*

CHAPTER 1

INTRODUCTION

Text classification is a technique to assign text pieces such as sentences, paragraphs, and documents into pre-defined categories with the help of Machine Learning (ML). It provides great convenience by reducing human effort and so eliminating human errors in the classification tasks. Classifying documents into topics, classifying emails for the related departments, and analyzing positive and negative comments for a product are some examples of application areas of text classification. While it has many different applications and the number of studies and projects increases day by day, there are also some challenges in text classification.

First, text data has a complex structure because it is composed of text pieces that have semantics, unlike the simple structure of numerical data. In order to make text data understandable by computers, there are some complicated techniques that should be applied. Second, the problem of finding labeled data, especially for some domain-specific fields is not easy. Labeling text data requires great cognitive effort and in some fields like law, and medicine, it also requires domain expertise. So, there are some challenges to labeling text data and a solution that helps to reduce labeling effort will provide great convenience.

In some classification tasks, the data has more than one label at the same time. For example, news can belong to different categories like the economy and foreign relations. Or in a picture labeling project with animals, there can be more than one animal in one picture. When the instances belong to more than one class, the problem is called a multi-label classification problem. In this case, the label is not a class

or a category, instead, it is a vector that has more than one dimension. Multi-label classification problems are more complex problems compared to the cases when the output is a single class like binary classification. Also, in the real world, for most cases, while the number of instances is high for some classes, for some classes it is not even sufficient to train a classifier. This is the so-called data imbalance problem and it is one of the most challenging problems in classification tasks.

When the data has an imbalanced distribution, the number of instances for some classes is not enough to train a classifier. At this point, the first solution that comes to mind is labeling more data but sometimes it is not feasible or too costly to restart the labeling project. Or, labeling data requires domain expertise, and finding experts to label data is not possible. In order to solve this problem in multi-label imbalanced text data classification tasks, we propose an auto-labeling algorithm that labels the unlabeled instances by utilizing the similarity between the instances. The proposed algorithm finds similar instances to the current ones in the data space and considers them as candidate instances. If the candidate instances help to improve overall performance, they are added to the labeled set. The algorithm searches the unlabeled data space iteratively and finds the possible instances to extend the labeled set. In this study, the details of the algorithm are explained and the experimental analysis is presented to test our algorithm.

The organization of this study is as follows. In Chapter 2, the literature review and background of the proposed algorithm are presented. NLP, text classification, multi-label classification, the imbalance data problem, and similarity functions that are used for text data are surveyed and the background for these topics is introduced. In Chapter 3, the proposed method is explained with its components. Experimental results and analysis is presented in Chapter 4. We explain the method with a toy example to demonstrate how our algorithm works, the design of the experiments and analysis of the results are also given in Chapter 4. Lastly, we finalize the study with the conclusion part to discuss the performance and future research directions.

CHAPTER 2

LITERATURE REVIEW AND BACKGROUND

In this section, the related subjects to the proposed method are explained. A detailed literature review is presented to exhibit the background of the study. In order to explain the problem clearly, a top-down strategy is employed.

2.1 NLP and Text Classification

Natural Language Processing (NLP), a subfield of Artificial Intelligence, is a field of study to process natural language data with computers. The main aim is to program computers in a way that they are able to understand human language. NLP has various applications and text classification is one of them. Classifying text data into categories with the help of ML is the main logic of text classification.

2.1.1 Natural Language Processing

According to a report prepared by IDC, by 2025, 80% of the data will be formed by unstructured data (Rydning, 2021). Images, texts, videos, sensor readings, tweets, logs, audios, and emails are the data types that have no structure. Unstructured data doesn't follow a defined schema or structure and can be formed in any kind of way. It can contain text, numbers, dates, or bytes to represent an information piece. Text data is the most common type of unstructured data because humans are the supreme data source in the world and humans create text data. Communications, interactions, and information transfer are the main human activities and they take place through the creation of text data. Every day, people and also machines programmed by people, produce a lot of text data for outrageous purposes. Websites, news, searches, reports, tweets, emails, transactions, social media posts, and comments are just a few exam-

ples of text data types. According to another report by DOMO, on average, every person on earth creates 1.7 Mb of data in one second (DOMO, 2018).

The world produces a great amount of data and this considerable amount of data does not mean they are worth reading, or they carry valuable information. Firstly, a huge amount of noise arises from the data sources, or even a data source may be a noise itself in most cases. Articles that do not have any useful meaning, statements that do not match the reader's purpose, and millions of tweets that are off-topic for a Twitter user are the data types that can be counted as noise. As human beings, we have limited time and interest, so all kinds of data types that do not cross with our interests are just noise for us. Secondly, data loses its validity as time passes. Today, yesterday's weather or news data does not mean anything to us. Alternatively, tons of suggested solutions or comments for a problem lose their validity after finding the exact solution. If we encounter that problem, we probably want to reach the exact solution, not the useless suggestions or comments proposed in the past. Furthermore, accuracy and consistency are other properties of the data. The deficiency of these properties makes it impossible to infer or learn something from the data. Always, we are in search of reliable data sources to obtain information but, in spite of that, the increase in the number of data sources decreases the reliability of the data. Today, we can reach a piece of information by just searching on the internet and clicking on the first or second link in a search engine. The main problem is how we can trust the information presented in that link. The mostly used encyclopedia on the internet, Wikipedia, says that the reliability of the information in Wikipedia should be questioned in an own article (Wikipedia, 2022). So, the accuracy of the data is another problem that we solve while we acquire data. That is to say, having a lot of data does not mean anything unless it is in our interest, accurate, up-to-date, and presents the information that we need.

In this context, NLP tools help people to increase the utilization of text data. NLP refers to understanding human language by analyzing text data with the components of syntax, grammatical structure, and semantics. In other words, NLP tries to mimic human communication by utilizing some advanced ML techniques. It is a combina-

tion of computational linguistics and models which can be a simple rule-based model or a very complex DL model. Its main applications are text summarization, text classification, named entity recognition, spell checking, machine translation, chatbots, etc. These applications can help us with different tasks and problems and make our lives easier. For example, text summarization tools summarize a very long text or an article and abbreviate it to a few paragraphs. Instead of reading tons of pages, it is much more efficient to read only some paragraphs to obtain the desired information. Also, the statistics for the NLP market can depict the recent advancements and the future of the NLP. In 2020, the market size for the NLP hit $16.5 billion and is projected to reach $127 billion by 2028 (Fortune, 2021) at a CAGR (compound annual growth rate) of nearly 30%. Another application of NLP, translation tools, is one of the most widely used tools on the internet. The market size for the machine translation industry is valued at $800 million in 2021 and projected to be $7.7 billion by 2030 as indicated in Figure 2.1(GMI, 2022). Chatbots are another application that entered our daily life recently. When we have a problem with our bank account or if we want to return a product that we bought from an e-shopping website, we generally communicate with a chatbot to solve our problem. According to a report, 80% of the global users have interacted with a chatbot and its market revenue has already reached $100 million (Leah, 2021, Thormundsson, 2022).

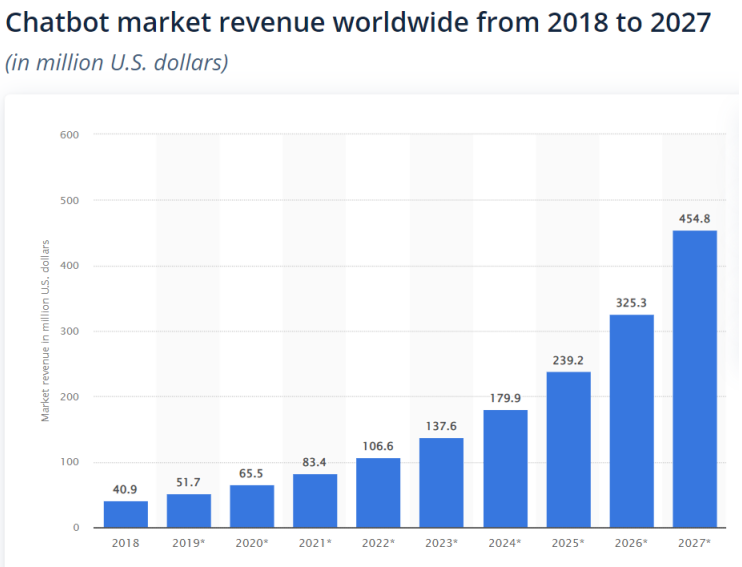

Figure 2.1: Chatbot market revenue by years (Reprinted from Thormundsson (2022)).

With the great increase in the market size, the number of researchers and studies on the NLP tools soared in recent years as well. According to Mohammad (2020), in the last 10 years, the studies increased nearly more than double and it shows the accelerating interest in NLP. Figure 2.2 shows the number of papers and the number of researchers who published in NLP.

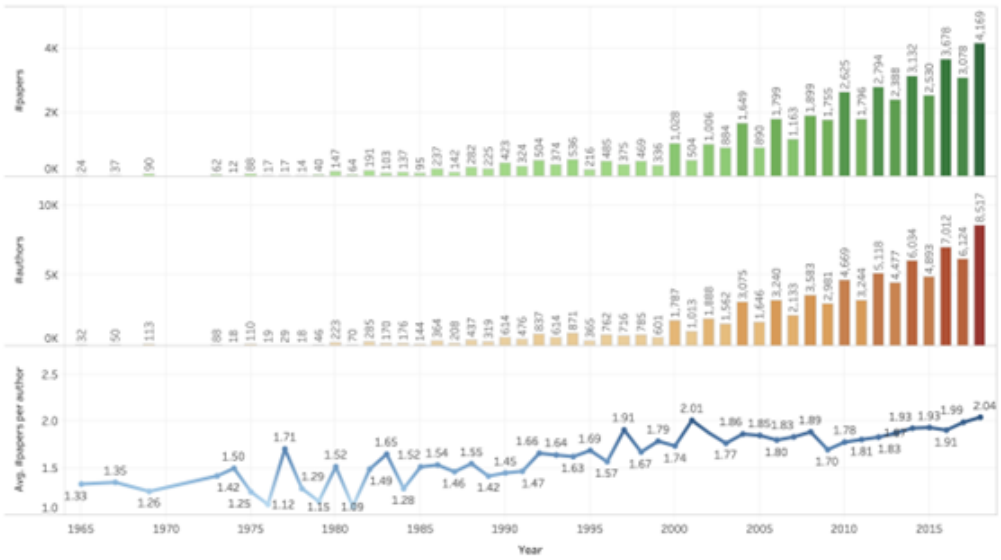

Figure 2.2: Number of papers, and the number of authors published in NLP, presented at ACL. (Reprinted from Mohammad (2020)).

NLP can be used in any kind of field that produces or comprises text data. Human is the main actor in the NLP systems because human is the main data source of text. Chatbots and machine translation systems are some examples to help to develop people's communication with the help of NLP. Also, insurance, management, medicine and healthcare, automotive, finance, manufacturing, retail, and education are the various industries in that NLP is widely used (Ly et al., 2020, Kalyanathaya et al., 2019, Kang et al., 2020, Spyns, 1996).

2.1.2 Text Classification

Text classification is one of the most widely used tasks in NLP. Aggarwal and Zhai (1970) remark that assigning a given text to predefined categories with the help of ML has countless advantages in different domains. ML models can help users to understand the text without any requirement to read the context of a text for a specific topic in a fast and reliable way. Providing amenities for different kinds of tasks increases the interest in text classification applications. For example, for a news website, classifying news for users based on their specified field of interest attracts more people to read more articles and also helps users to find what they want to reach easily. Another example is classifying the positive or negative comments to rate a product on an e-shopping website to inform users what other people think about the product. Also, an email filtering tool classifies emails as spam or not spam by utilizing a text classification is another example of it. Spam filtering is an important application to provide security for email users.

As the examples account for, text classification is a process of automation of classifying text pieces into predefined categories. Instead of reading and understanding a text piece and categorizing it manually, a text classification tool can help us to do the task in an efficient, consistent, and scalable way. Thus, the effort of humans is removed, and so are the errors that come from nature of humans. Automating text classification tasks provides companies or anyone who is interested in cost-efficiency, automation, and scalability.

Text data can be categorized in different levels, as text lengths can vary from very short to very long. In Kowsari et al. (2019), text classification is grouped into 4 levels. Sentence level and sub-sentence level are the smallest pieces of text that express a statement. The paragraph level is the middle level between a sentence and a document. Paragraphs are generally formed by a number of sentences and are shorter than a document. The document level is the highest level. Longer texts than a document should break into pieces to make a meaningful classification.

ML algorithms work with numeric data. In most cases, datasets contain non-numerical features like categories, text, ranks, etc. In order to convert non-numerical features to the numerical format, a variety of preprocessing techniques are available. Converting text data to numeric is an important problem in the NLP world as well. There are some techniques developed to represent text data in numerical format. These techniques vary from very simple methods like one-hot encoding to very complex models like transformers. **Bag of words**, **skip-gram**, **word2vec**, and **tf-idf** vectorizer are some main techniques used in text vectorization (A. K. Singh and Shashi, 2019).

Thanks to the surprising increase in the ability of the Deep Learning (DL) models and computation power, very complex models have emerged that solve vectorization problems successfully. With the help of these models, the representative power of the created numeric forms gets better and so do the results of classification tasks. For example, one-hot encoding, one of the simplest ways of vectorization, only considers the existence of a word, and meaning is not included. In word embeddings, they are much more complex models compared to one-hot encoding, not only the semantic, but also the context of the text is included, so the most successful models are created to represent text data. The architecture of the word embeddings is generally based on complex DL models. A word can have different meanings and its meaning can change in a text in accordance with the context. Word embeddings are designed to capture the context and semantics of the text. S. Wang et al. (2020) presents the recent advances in word embeddings. According to his study, Word2vec, GloVE, fastText, Elmo, and BERT are some examples of widely-used word embeddings developed recently.

In order to apply embedding techniques, a series of preprocessing steps should be applied. In Anandarajan et al. (1970), the text preprocessing step refers to the process of converting text into a standardized format. Its aim is to clean, format, and standardize the text before vectorization. Simply, cleaning text from words that do not carry any meaning, like numbers, emails, stop words, and finding its etymon are the main steps that should be followed. The operations can be determined according to the content and type of the data. Punctuations, URLs, emails, and so on are the components in a

text that does not carry any meaning, and removing them would benefit the model's performance. Most programming languages are case-sensitive and upper-case and lower-case of the same word refer to different objects. In order to prevent this disorder, lowering operation is applied to fit the same standard. For example, if the data that is obtained by web scraping operations probably contains some noisy components like HTML tags, numbers, chars, etc. Also, text data contains stop words like 'is', 'that', 'with', and so on that do not carry any meaning by themselves.

Kowsari et al. (2019) mentions that after obtaining a standardized format of the text, tokenization is applied to get smaller units like sentences, words, etc. from the text. The tokenization step generally takes the piece of text and returns a list of sentences or words depending on the chosen tokenization method. The next step in the text classification project is finding the root words (Forman et al., 2003). The first way to do this is stemming, which refers to the standardization of words to their root form. In other words, different forms of a word can exist in a text and carry the same meaning. For example, 'working', 'worker', and 'works' come from the same root and will be stemmed to 'work'. Stemming has one main disadvantage; words lose their meanings. For instance, working and worker have different meanings even though they come from the same root. Another way to obtain the root form of words is lemmatization. Lemmatization is a technique of converting each word to its base form to provide a standard form across the text (Balakrishnan and Lloyd-Yemoh, 2014). One of the techniques can be chosen to standardize words before vectorization.

After the text preprocessing step, a more clean and standard format of the text is obtained. Still, feeding them to an ML model is not possible, since ML models work with only numerical features. A feature extraction step should be applied to convert text to vectors. In Y. Li and Yang (1970), Bag of Words, Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, WordNet, and Global Vectors for Word Representation (GloVe) are some of the techniques that are utilized to convert text to vectors (Pennington et al., 2014, Kowsari et al., 2019). After applying these techniques, text data is ready to be fed into ML algorithms.

2.1.3 Steps in Text Classification

Unlike structured data, text data is not in a standard format that is directly feedable to an ML algorithm. Because humans create text data with errors and also the challenges faced during data collection processes cause non-standard formats and non-standard contexts (Kowsari et al., 2019). It may contain misspellings or unnecessary components like numbers, e-mails, and symbols that are considered noise. Fixing the misspelling, removing the noises, and putting in a standard format are the main steps that are followed in a text classification task under the text cleaning step. Tokenization, stop word removal, and stemming/lemmatization is the other processes applied to put text in a more standard format before the vectorization step. The vectorization step converts texts to long numerical vectors and the size of this vector is generally quite large. In order to scale down the size of the vectors, dimensionality reduction methods are applied. Lastly, a learning algorithm is trained to finalize the classification task.

2.1.3.1 Preprocessing

A series of operations is applied to preprocess text before passing to the vectorization step (Vijayarani et al., 2015). In the first step, all texts are lowered to ensure all the words are in the same case format. Lowercasing is applied because lowercased and uppercased strings correspond to different objects in programming languages. The next step is correcting misspellings in the text. Misspellings occur usually and there are some ready-to-use libraries to fix them. In the text cleaning step, the text should be cleaned from noises which means any kind of text piece that does not add any meaning to the text's semantics. Noise can be present in the text because of the collection and the creation process. For instance, in a job posting, the email or the phone of the company, or in financial news the currency rate carries information but does not add any meaning to text semantics can be considered noise. Types of noises can be listed as symbols, emojis, HTML tags, URLs, emails, non-word chars, words that contain numeric characters, etc. All kinds of noises should be removed to obtain a clean formatted text.

After the text cleaning step, long text pieces are crumbled into tokens, meaning smaller text pieces, generally words or n-grams. Tokenization is necessary to obtain the list of words in a text that will be used for removing stop words and standardization operations. An example of tokenization is shown in Figure 2.3.

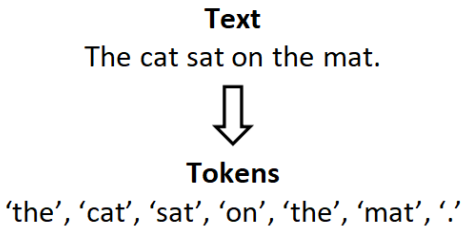

Figure 2.3: Tokenization process of a sentence.

The next step is removing stop words from the text. Stop words represent the word group that does not carry any specific meaning for the text like 'is', 'at', 'through' etc. Stop words are the word set that is commonly used and removing them provides convenience to focus on the words that are important. The stop words list can be obtained from libraries like NLTK (Loper and Bird, 2002). The text is purged from the noise and now is ready for standardization. Lemmatization or stemming is used to get the root of a word, in other words, removing morphological variations from the word to put them in a more standard format. Stemming is a heuristic process that removes the derivational affixes from a word and thus aims to obtain the root of a word. Lemmatization is a process of analyzing words morphologically and also, considering the context to convert them into their meaningful base forms. According to Balakrishnan and Lloyd-Yemoh (2014), lemmatization outperforms stemming overall.

Finally, a length checking function is applied to eliminate very long and short words that are not present in the English corpus. The summary of the steps is presented below, in Table 2.1.

Table 2.1: Preprocessing steps for text data.

Explanation	**Processed Text**
raw text	' 9. Questions and feedback we welcome your questions, comments, and concerns about this site. Please send any and all feedback pertaining to this site to: Pgforme@post-gazette.com or by calling 1-855-pg-for-me. Feedback should be directed to the feedback tab on the site or by E-mailing: Contact@post-gazette or Comments@post-gazette.com '
removing nonword characters	' questions and feedback we welcome your questions comments and concerns about this site please send any and all feedback pertaining to this site to or by calling pg for me feedback should be directed to the feedback tab on the site or by e mailing or '
tokenization	['questions', 'and', 'feedback', 'we', 'welcome', 'your', 'questions', 'comments', 'and', 'concerns', 'about', 'this', 'site', 'please', 'send', 'any', 'and', 'all', 'feedback', 'pertaining', 'to', 'this', 'site', 'to', 'or', 'by', 'calling', 'pg', 'for', 'me', 'feedback', 'should', 'be', 'directed', 'to', 'the', 'feedback', 'tab', 'on', 'the', 'site', 'or', 'by', 'e', 'mailing', 'or']
removing stop words	'questions feedback questions comments concerns site send feedback pertaining site calling pg feedback directed feedback tab site mailing'
lemmatizing or stemming	'question feedback question comment concern site send feedback pertain site call pg feedback direct feedback tab site mailing'

2.1.3.2 Vectorization

Text is cleaned, standardized, and is now ready for vectorization. Vectorization is the process of converting text into numerical features to feed an ML model. ML models only work with numerical data and there are various methods that are used to convert text data into numeric. The simplest approach is converting text into a one-hot encoding or bag of words model. The bag of words model counts the number of occurrences of each word in a text and represents it as the count vectors. It simply converts all the words in the corpus to columns and counts the words' frequency for each text instance. Here is an example of the bag of words model presented in Table 2.2.

Table 2.2: Bag of words model representation (Reprinted from Consultant (2022)).

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Term frequency-inverse document frequency, i.e. tf-idf, is a technique that evaluates the importance of a word for a document in a set of documents (Fautsch and Savoy, 2010). It has two terms, term frequency, and inverse document frequency. Term frequency calculates the frequency of the terms in a document while inverse document frequency calculates the frequency of the word among the corpus. By multiplying two terms, tf-idf score is calculated and is used to score the relativity of the terms to that document.

In these kinds of approaches, the main problem is the size of the vectors is too big and most of the values in these vectors are zero which causes the sparsity problem. For long texts, the vocabulary size is generally too big and so the size of the vectors is too

long as well. If the number of unique words in a text corpus is 10,000, then the size of the vectors for each text instance will be 10,000. Also, most of the columns will be zero for the words that do not exist in different text instances. This is the so-called sparsity problem in ML. Sparsity is inefficient for computation causing a great increase in computation time and complexity, and it should be fixed to regain efficiency (Nasiri et al., 2017). On the other hand, these methods do not consider the semantics of the text. Only counting the words does not offer promising results because of two reasons. First, different meanings can be presented with the same word groups. Second, the same meanings can be obtained by gathering different word groups together. In short, using count-based methods can cause problems if the semantics of the text is important.

In order to solve the sparsity and pay attention to semantics, word embeddings are proposed. Word embeddings are the methods that convert words into dense vectors by training complex neural networks. A word embedding is generally a fixed-size vector and it contains the characteristics of the words such as the semantic relationship of the words, definitions, context, etc. With recent studies, the number of embedding methods is increasing and their success in NLP tasks is proven. Word2Vec, BERT, GloVe, and Elmo are some state-of-the-art embedding methods used widely. A representation of word embeddings can be seen in Figure 2.4.

Word2Vec, proposed in Mikolov et al. (2013), is an efficient two-layer neural network to produce word embeddings. Its architecture is a shallow neural network and it produces a vector for each word to demonstrate proximity between words. In Pennington et al. (2014), **GloVe** is an unsupervised technique that utilizes local statistics (context information of words) and global statistics (word co-occurrence). It tries to derive semantic relationships by using global statistics. **Elmo**, proposed in Ilić et al. (2018), uses a two-layer bidirectional language model to train with characters, not words. Unlike other embeddings, **Elmo** is capable of capturing different meanings of words in different contexts. A vector representation for the same word can be different if it has different meanings in different texts. **BERT**, Bidirectional Encoder Representations from Transformers, has an architecture that is based on transformers

(Devlin et al., 2018). Transformers are complex DL models in which all output elements are connected to all input elements. Instead of processing words one by one in a sequence, it uses parallelism to process words to speed up the training.

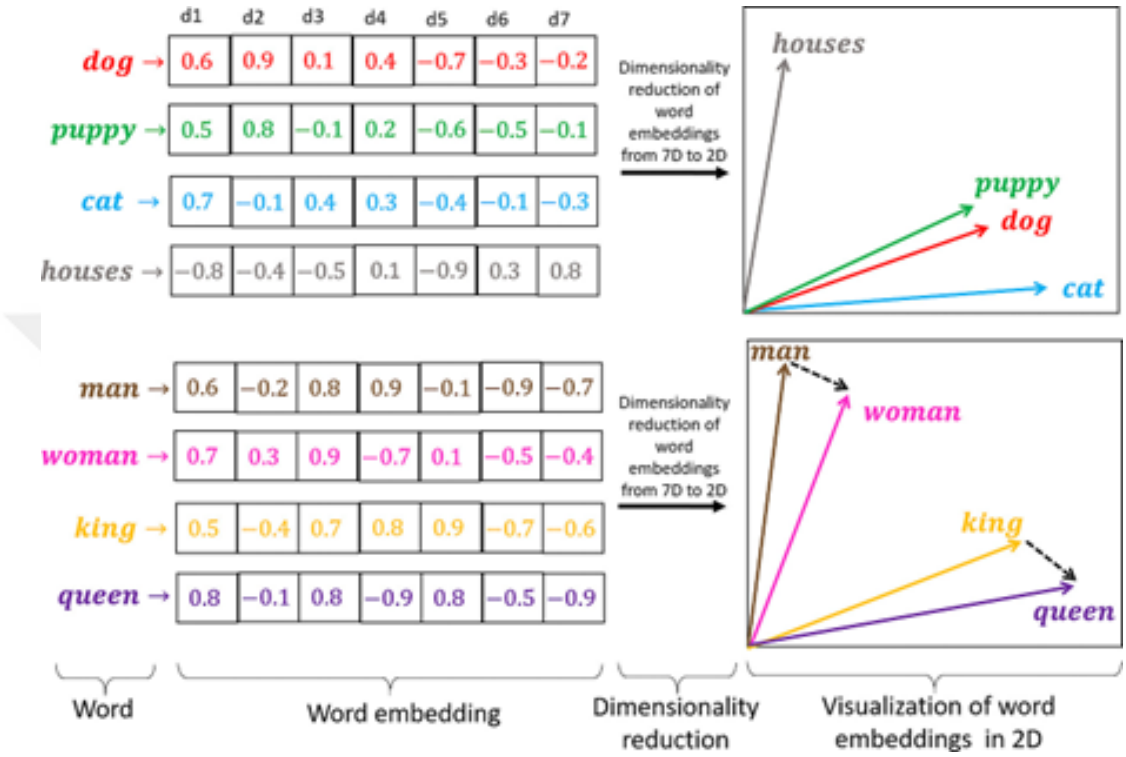

Figure 2.4: Embedding representation for some words.

Embeddings are a powerful representation method for texts (Ruzzetti et al., 2021). Embeddings capture the semantics of the words and the relation between them. For example, king and queen are words that are similar words semantically. The only difference between them is gender. In Figure 2.5, the relation between king and queen can be seen and the difference vector between them is very similar to the difference between man and woman vectors. The vector for 'queen' can be obtained by subtracting the gender vector from 'king'. So, it can be seen that the power of the word embeddings in representing the relation between words.

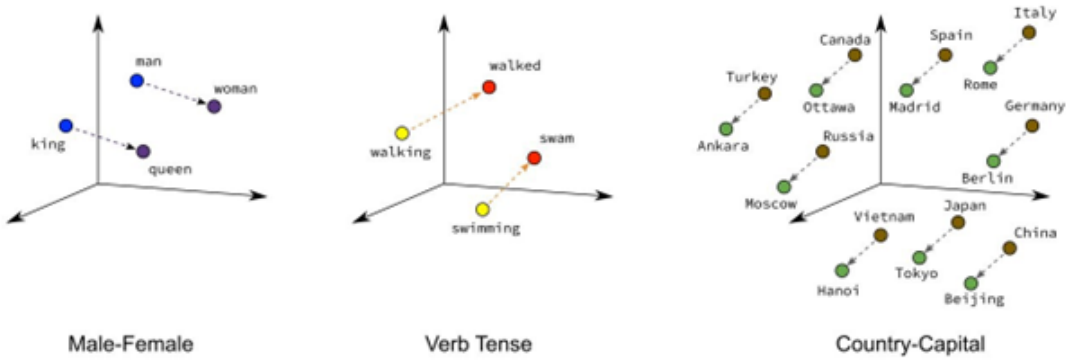

Figure 2.5: Embeddings can capture the semantic similarity between words.

2.1.3.3 Dimensionality Reduction

After getting the word embeddings, the input vectors are still too long like 512, 768, or 1024 depending on the preferred embedding technique. In order to decrease that dimension, some dimensionality reduction techniques are used. Dimensionality reduction techniques are statistical methods that reduce a vector to a lower dimension while trying to keep its characteristics. Sorzano et al. (2014) presents a comprehensive survey of dimensionality reduction techniques.

Principal Component Analysis, simply PCA proposed by Pearson (1901), is the most popular technique in dimensionality reduction. It is a linear and unsupervised technique that transforms the input vector into a smaller uncorrelated vector called principal components. Its aim is to retain the variation in the original vector space as much as possible. While PCA is a linear technique, Kernel-PCA is another method to capture nonlinear relations in the input space. The kernel function projects the data into a higher dimensional space in which the data can be linearly separable. Then the standard PCA method is applied to degrade the data to a lower-dimensional space back. Singular Value Decomposition, SVD, is a matrix decomposition method. It composes the input matrix X into the product of three matrices, $X = WDU$. The size for W is m by m, D is a diagonal matrix of size m by n, and lastly, U is of size n by n (Abdi and Williams, 2010). SVD also can work with sparse data as well. Mul-

tidimensional Scaling or MDS in short is a non-linear method that tries to maintain the distances between samples while reducing the dimensionality of non-linear data (Kruskal, 1964). t-distributed Stochastic Neighbor Embedding (t-SNE) is another non-linear dimensionality reduction method (Van der Maaten and Hinton, 2008). The aim of t-SNE is to project high-dimensional objects to low dimensions by considering their similarities with a probability distribution. It constructs a probability distribution to map the similar objects close to each other in the low-dimensional space by minimizing Kullback–Leibler divergence. It is used generally for visualizing image and text data. Isomeasure mapping (isomap) reduces dimension by connecting instances by calculating the geodesic distances to their nearest neighbors (Yang et al., 2006). Max pooling is a method that is widely used in Computer Vision and NLP applications in case of the existence of high-dimensional vectors (Paul and Chaki, 2019). It gets the maximum values for each window that slides over a vector or matrix. While it is reducing the dimension it also brings out the most important features and gets rid of the insignificant ones.

2.1.3.4 Classification Algorithms

After applying preprocessing steps, vectorization, and dimensionality reduction techniques, the input set is now ready to be fed into a classification algorithm. It is still a major research topic and in the literature, almost any kind of classifier is tried to examine results to find the best algorithm. Naive Bayes is one of the most popular algorithms chosen in text classification tasks (Kim et al., 2002). Due to its mechanism that captures the probabilistic relations, it tries to find the pattern between word occurrences and output. However, while the complexity of embeddings increases Naive Bayes loses its ability to capture that pattern. Support Vector Machines work well with high-dimensional datasets and so it is widely preferred for text classification tasks (Chau and Chen, 2008; Goudjil et al., 2018; Joachims, 1998; Z.-Q. Wang et al., 2006). SVM tries to find a decision boundary in the data space if it exists. Its main drawback is that it requires computation power and is slow for training compared to other methods (Yuan et al., 2008). On the other hand, the linear version of SVM performs well and it is very efficient to train compared to standard SVM (Du-

mais et al., 1998). Decision Tree is another classification algorithm widely used in different areas and also in text classification tasks (Lewis and Ringuette, 1994). Its tree-based structure helps to create decision rules to decompose the data space. The decision tree is a very fast and easy-to-interpret algorithm. On the other hand, it tends to overfit the data easily. Random Forest is an ensemble algorithm which composes of many decision trees in parallel (Ho, 1995). Predictions from parallel decision trees are combined to create a final prediction via the voting scheme. Random Forest performs well in most of the tasks, as well as text classification. Since it is an ensemble algorithm, its computation time is high compared to other algorithms. Logistic Regression is the modified version of linear regression by adding a sigmoid function to the output layer. The predicted real values are converted to classes by using the sigmoid function. Logistic regression is also widely used in text classification problems (Pranckevičius and Marcinkevičius, 2016).

Today, for most tasks, DL reaches state-of-the-art results. Also in NLP, for various kinds of problems, the best results are obtained with DL algorithms. With the help of development in computation power, it is possible to train more complex DL models. Moreover, the increase in the amount of data helped to train these complex models. Complex models necessitate a great amount of data and today's state-of-the-art models are trained with a huge amount of datasets. For example in 2020, GPT-3, one of the most complex models in the NLP world which has 175 billion parameters, is trained with 45 TB of text data (Cooper, 2022). Since it is an emerging study field, the number of studies in the literature that studies different architectures to improve the performance on different NLP tasks. Recurrent Neural Networks, Long-Short Term Memory Networks, Convolutional Neural Networks, and Transformers are the popular architectures in NLP (Gupta et al., 2020). Most of the state-of-the-art models are shaped around Transformers architecture (S. Singh and Mahmood, 2021). However, as it is mentioned, DL models require a huge amount of data for training. So, when the main problem is data insufficiency, it is not a good choice to use a DL model at all.

Linear Support Vector Machines (SVM) is simply the linear version of SVM that uses a linear kernel function (Chauhan et al., 2019). SVM is a mathematical model

that tries to find the maximum margin hyperplane to separate classes (Boser et al., 1992, Cortes and Vapnik, 1995). The mathematical model of the SVM is given in Equation 2.1 and the graphical representation of the algorithm can be seen in Figure 2.6. SVM is an effective algorithm in high-dimensional space which makes it so popular for text classification algorithms. After applying vectorization methods the data become high-dimensional and it is required to use an algorithm that works well in high-dimensional space. SVM handles high-dimensional datasets well even if the size of the features is greater than the size of the samples which is called the curse of dimensionality. Another advantage of SVM is it uses only a subset of samples during the training of the decision function which makes it memory efficient. Again, after applying the vectorization the data becomes high dimensional and generally requires a lot of memory for calculations. The main drawback of the SVM is that it is not suitable for large datasets. Thanks to the nature of our problem, we aim to solve the imbalanced and insufficient data problem. So, the datasets that our algorithm works with will not be affected by this problem.

$$y = wx' + \gamma \tag{2.1}$$

SVM uses kernel functions to transform data that is not separable in the current data space. Kernels are simply mathematical functions to transform data from the current space to another high-dimensional space. In that way, the data is mapped to another space where data is linearly separable. The equation for the general kernel function is given in equation 2.2. Kernel functions take two inputs and after applying them to a mapping function, it returns their dot product.

$$K(x_1, x_2) = \phi(x_1)\phi(x_2) \tag{2.2}$$

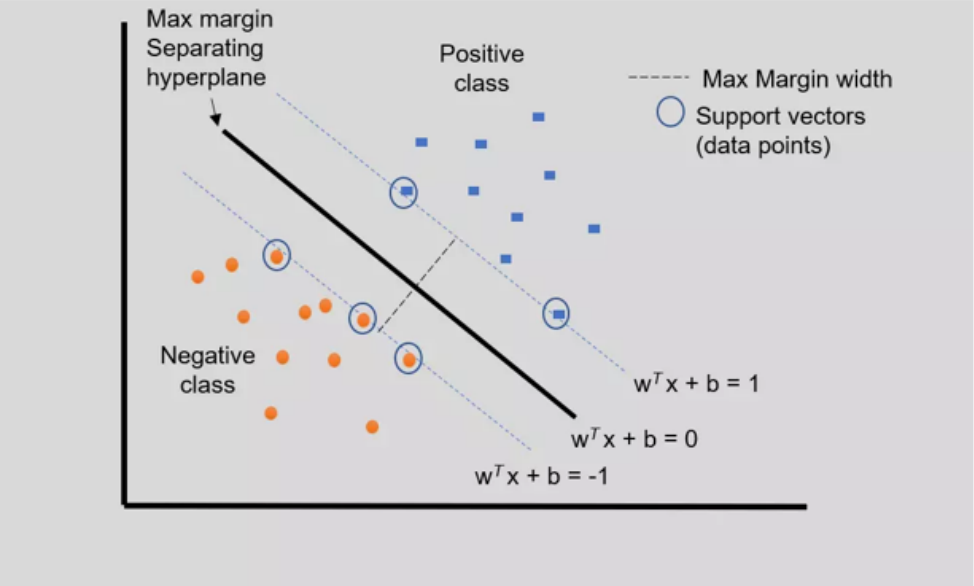

Figure 2.6: How SVM classify instances (Reprinted from Bedre (2021)).

In Linear SVM, the kernel function is the input itself. It is because the feature set is rich enough to not require any mapping and simply the input itself is sufficient. It is a very efficient technique, even compared to SVM, because of its computational efficiency. It reaches nearly the same performance for some tasks like document classification and etc. (Sun et al., 2010). Not only providing good performance tasks but also its efficiency make the Linear SVM a favored alternative for text classification. Sulaiman et al. (2020), Kandhro et al. (2019), Lilleberg et al. (2015), and Raghavan et al. (2019) use Linear SVM as their learning algorithm, and Joshi and Abdelfattah (2021), Yıldırım et al. (2019), and Omar et al. (2021) made a comparative study between classification algorithms and Linear SVM performs best in their experiments. So, after surveying the literature we decided to move on with Linear SVM as our main learning algorithm. The algorithm is coded in Python and we use the well-known Python library for ML applications, the scikit-learn library, to apply Linear SVM. SVC object is used from scikit-learn and the $class\_weight$ parameter is set to balance in order to give proportional weights to classes. It adjusts the C parameter as $class\_weight\_i * C$ by calculating the class weights inversely proportional to class frequencies with equation 2.3.

$$class\_weight_i = n/(m * n_i) \tag{2.3}$$

In Vapnik (1998), Crammer and Singer (2002), and Weston, Watkins, et al. (1999), SVM is extended to multi-class problems, or in the same way, to multi-label classification problems using the one-versus-rest technique. One-versus-rest classification trains, the number of classes, and t classifiers in parallel and assemble the results for final prediction (J. Xu, 2011). One-versus-rest classifier considers each class label independently and creates a label set by combining predicted labels obtained by t classifiers.

2.2 Multi-Label Classification and Evaluation measures

Multi-label classification is a type of classification problem when the instances belong to more than one class. Its structure and used metrics are different than the binary output classification problems and are explained in this section.

2.2.1 Multi-Label Classification

Classification problems can be categorized into single-label classification and multi-label classification (Er et al., 2016). In single-label classification, an instance can only take one label. If the outcome is the existence of a label or not, it is called binary classification. If the number of possible labels is greater than 2, the problem is called a multi-class classification problem. On the other hand, in multi-label classification, instances may belong to more than one class. In other words, in a multi-label classification problem, each instance has a label vector instead of a certain class value (Herrera et al., 2016). The representation of the labels for binary, multiclass, and multi-label classification problems is shown in Table 2.3.

Table 2.3: Label representation in binary, multiclass, and multi-label classification.

	Binary classification		Multi-class classification		Multi-label classification	
Input	**Label value**	**Label vector**	**Label value**	**Label vector**	**Label value**	**Label vector**
$\mathbf{X_1}$	t_1	0	t_2	[0,1,0,0]	[t_2, t_3]	[0,1,1]
$\mathbf{X_2}$	t_1	0	t_4	[0,0,0,1]	[t_1,t_3]	[1,0,1]
$\mathbf{X_3}$	t_2	1	t_1	[1,0,0,0]	[t_2]	[0,1,0]
$\mathbf{X_4}$	t_1	0	t_4	[0,0,0,1]	[t_1,t_2,t_3]	[1,1,1]
possible classes	{t_1, t_2}	[0], [1]	{t_1, t_2, t_3, t_4}	[0], [1], [2], [3]	{{t_1},{t_2}, {t_3},{t_1,t_2}, {t_1,t_3}, {t_2,t_3}, {t_1,t_2,t_3}}	[0,0,0], [0,0,1], [0,1,0] ...

In the table above, the labeling schema for the problem types is represented to exhibit the difference between problem types. In binary classification problems, a label set has two outcomes one or zero, meaning the existence of a label or not, represented as t_1, t_2. An example of binary classification is whether a bank intersection is a fraud or not. In multiclass classification, there are more than two classes and an instance has to belong to only one of them. In this case, there are 4 classes, represented by t_1, t_2, t_3, t_4. An example of it, a bird can belong to one of the species. On the other hand, in some cases we cannot limit the label to one class, for example, an article may belong to more than one category, or in an image, there may be more than one object that can be detected. In other words, an instance can belong to more than one class at the same time. In the given table, the number of classes is 3 and the number of all possible label combinations for the multi-label case is 23. If we formalize it mathematically, the total probability of belonging to different classes sums to one in binary and multiclass classification but not in multi-label classification.

$$\sum_{i=0}^{m} p\{c_i\} = 1 \qquad (m\text{: \# of classes, } m = 1 \text{ for binary classification}) \tag{2.4}$$

$$\sum_{i=0}^{m} p\{c_i\} = 1 \qquad (m\text{: \# of classes, } m > 1 \text{ for multiclass classification}) \tag{2.5}$$

$$\sum_{i=0}^{m} p\{c_i\} \leq m \qquad (m\text{: \# of class, } m > 1 \text{ for multi-label multiclass classification}) \tag{2.6}$$

Where m is the number of all possible classes and $p\{c_i\}$ is the probability of belonging to class c_i for an instance. m is defined as 1 for binary classification, equal to the number of classes in multiclass classification and multi-label classification. The main difference for probabilities is that it is summed to exactly 1 for binary and multiclass classification but it can be more or less than 1 for multi-label classification.

A multi-label classification example is given to demonstrate the multi-label classification setting. X is the input vector and if it is labeled with a class, it takes 1 and otherwise takes 0 in the table.

Table 2.4: A multi-label classification example.

Input / Classes	1	2	3	4	5	6	7	8	9
X_1	0	0	1	0	1	0	0	1	0
X_2	1	0	0	1	0	0	0	0	0
X_3	1	0	1	0	0	0	0	0	0
...									
X_n	0	0	0	0	0	0	0	0	1
# of instances	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9

In the given example above, different instances take different labels for different classes. For example instance 1 belongs to classes 3, 5, and 8, instance 2 belongs

to classes 1 and 4, while the nth instance belongs to only class 9. The total number of instances belonging to all classes, $\sum n_i$, can exceed the number of instances in multi-label classification while there is equality for multiclass classification.

In order to solve the multi-label classification problem, there are some solution methods proposed. Nareshpalsingh and Modi (2017) presents a comparative study on the solutions to the multi-label classification problem. Binary relevance, label power-set, classifier chains, and calibrated label ranking methods are some of the solutions for multi-label classification problems. Binary relevance learns independent binary classifiers for each class and to make a prediction it combines all the predictions of all independent binary classifiers. It considers all the classes mutually exclusive and as indicated in M.-L. Zhang and Zhou (2013), it ignores the correlation between classes.

2.2.2 Evaluation Measures for Multi-Label Classification

Compared to binary classification and multi-class classification in which the output is a single class, in multi-label classification the label is a label set or simply a vector. So, the traditional evaluation measures do not work for multi-label settings. They should be modified or new measures should be introduced to evaluate the performance. In the literature, modified versions of existing measures and also, new measures designed for multi-label classification are used.

Tsoumakas and Katakis (2007), Ganda and Buch (2018), and M.-L. Zhang and Zhou (2013) surveyed the measures used in multi-label classification problems. They, all present modified versions of traditional measures, and new measures specially designed for multi-label settings are introduced. Overall, measures can be categorized under classification measures and ranking measures. Classification measures are interested in the performance of the predictions. Ranking-based measures are designed to measure the performance of the prediction probabilities.

We explain the performance measures one by one. Before starting, it would be convenient to give some definitions for the sake of clarity. TP (True Positive) represents

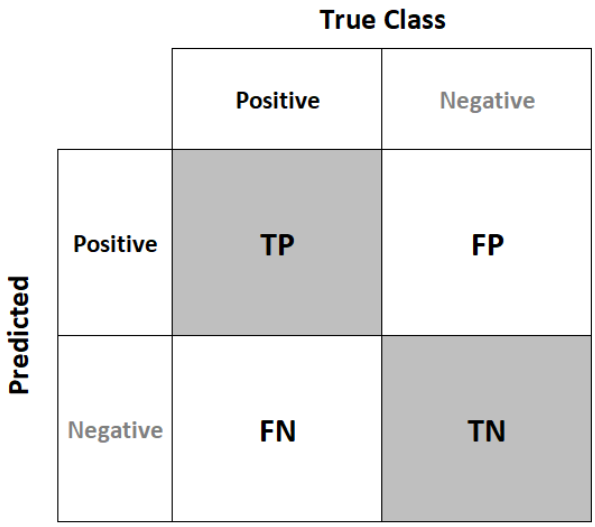

Figure 2.7: Confusion Matrix

the correctly predicted positive class. Likewise, TN (True Negative) stands for the correctly predicted negative class. If the model predicts an instance incorrectly, it is called FP (False Positive) for the positive class and called FN (False Negative) for the negative class. In Figure 2.7, a confusion matrix is given to demonstrate the terms TP, FP, TN, and FN. Also, in the below equations, n stands for the number of instances, Y stands for true labels, Z stands for predicted labels and L represents the length of the label vector.

Exact Match Ratio

The exact match ratio (EMR), also known as subset accuracy, is the proportion of correctly predicted label set, i.e. label vector, to the total number of label vectors (Equation 2.7). It considers the label set as a whole and counts only if all the labels of an instance are predicted correctly (Ganda and Buch, 2018). Because it does not care about the label vector components, it is a very strict measure.

$$EMR = \frac{1}{n}\sum_{i=1}^{n} I(Y_i = Z_i) \tag{2.7}$$

Accuracy

Accuracy measures the ratio of correctly predicted individual labels to the total number of labels as given in Equation 2.8 (Nazmi et al., 2020). Compared to subset accuracy it considers all the components of the label vector and a more indulgent version of it.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

Hamming Loss

Hamming loss is the proportion of individual labels that are incorrectly predicted (Ganda and Buch, 2018). It simply equals one minus accuracy score. The formula for hamming loss is given in Equation 2.9.

$$Hamming\ Loss = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|L|} \tag{2.9}$$

Precision and Recall

Precision is found by dividing the number of correctly predicted outputs by the total number of predicted outputs. The proportion of correctly predicted outputs to the total number of true outputs is called recall (Gibaja and Ventura, 2015). There is a trade-off between precision and recall. If the focus is on precision, an increase in precision will cause a decrease in the recall and vice versa. If we want our model performs well in detecting a class, we should focus on recall. The Equations 2.10 and 2.11 show how precision and recall are calculated.

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.11}$$

F1-Score

On the other hand, if the desire is to obtain accurate predictions, then the precision should be maximized. F1-score solves this trade-off and focuses on precision and recall at the same time. As it can be seen from Equation 2.12, it is the harmonic mean of precision and recall. If there is no special focus on precision or recall, f1-score is generally a better measure in classification tasks.

$$F1\text{-}score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.12}$$

One Error

One error is the ratio of examples whose top-ranked label is not in the true label set (Tarekegn et al., 2021). It only cares about the top-ranked label. It is a loss measure and should be minimized.

Coverage

Coverage computes how far it is needed to go through the ranked scores to cover all true labels. The rank of a label is calculated by ranking the prediction probabilities and finding the number of labels whose probability is greater than or equal to its score. The best value is equal to the average number of labels in the true label set per sample (Tsoumakas et al., 2009).

Ranking Loss

Ranking loss is the average number of label pairs that are incorrectly ordered given that the prediction probabilities are weighted by the size of the label set and the number of labels, not in the label set (Gao and Zhou, 2011). It is similar to error set size with a difference in weighting with relevant and irrelevant labels. Since it measures the loss, the best value is zero.

Average Precision

Average precision is the weighted mean of precisions achieved at each threshold level on a precision-recall curve. The weight is the increase in recall from the previous threshold used as the weight (Liu and Chen, 2015). The formula for average precision is given in Equation 2.13. In the equation, R represents the recall score and P represents the precision score at each threshold level j.

$$AvgPrec = \sum_{j} (R_j - R_{j-1}) P_j \tag{2.13}$$

Another aspect of our problem is the multi-label settings. In binary classification, using one of the mentioned can solve the problem but in the multi-label settings, the number of classes is more than two. Binary classification measures can present the results for a single class but not overall performance. In order to overcome this problem, different weighting methods are used to determine the importance of the classes(M.-L. Zhang and Zhou, 2013). Micro averaging calculates the measures globally, without separating classes. In other words, it weighs all instances equally. On the other hand, macro averaging calculates the measure for each class and averages them as they have equal weights. Macro averaging is good when all the classes have equal weights, or if the aim is predicting all classes with the same performance. Also, weighted averaging is a method that calculates measures for each class and takes the weighted average of the class measures. Weights are calculated proportionally to the number of instances in the test set. Lastly, sample averaging calculates the measures for each instance and finds their average. It does not discriminate against classes and considers all the instances equally important.

Quantifying the performance of models correctly has vital importance in case of data imbalance. It is crucial to know and choose the correct evaluation measure to evaluate models. The measure should meet the requirements of the problem. Choosing the wrong measure will mislead the evaluations and result in choosing a poor model. For example, accuracy measures the correctly classified labels without considering the class ratios. For the highly-imbalanced datasets which means the ratio of classes is

very high using accuracy as the primary measure may cause problems. For instance, in one case, the ratio of positives to negatives, let's say 1 to 1,000. If the trained classifier predicts all the instances as negative, it will reach an accuracy of 99.9% which looks perfect but it will mislead the modeler to very poor results for the positive class. 0% of accuracy is obtained for the positive class, in which the actual aim is to detect the positive class. In order to prevent this problem, precision, recall, or f1-score are preferred over accuracy for imbalanced classification problems (Feng et al., 2021).

2.3 Imbalanced Data Classification

Imbalanced data simply means the inequality of the number of instances among label classes. ML algorithms perform poorly when there is considerable inequality between classes. If one class has fewer instances while others have many, the algorithm tends to learn majority classes better and yields poor performance in the minority class. Unfortunately, in most real-world cases the datasets are imbalanced and the number of instances for different classes is distributed unequally. An example of the imbalanced dataset in the real world is depicted in Figure 2.8. A dataset for the purpose of visits to a branch office of a bank is collected and it can be seen that most of the people visit for debt collection. Other classes like credit reporting and mortgage are also at acceptable levels but classes like money transfers and virtual currency have a very low number of visits. The number of instances for these classes is very low. If that bank wants to train a classifier that predicts the customer's purpose of visit, the data imbalance problem will arise and the algorithm cannot learn the classes that have few instances.

The class imbalance problem has received great attention because of the problems it causes in the studies of ML and Pattern Recognition (Mollineda et al., 2007). It is more important in such real-world applications where misclassifying a minority example costs a lot. Spam mail classification, diagnosis of rare diseases, and authentic document classification are the main examples where the cost of classifying rare instances is very high. Showing a customer a spam email as safe may cost a lot to the email provider if the customer is hacked because of that spam email. Similarly, clas-

sifying a cancer patient as a non-cancer according to test results may lead to wrong treatment and costs to the health of a person. In order to prevent the effect of imbalanced data in this kind of critical problem, different solutions have been proposed in the literature. The trend for the published papers in the literature is presented in Figure 2.9. We can see that the number of papers is increasing greatly.

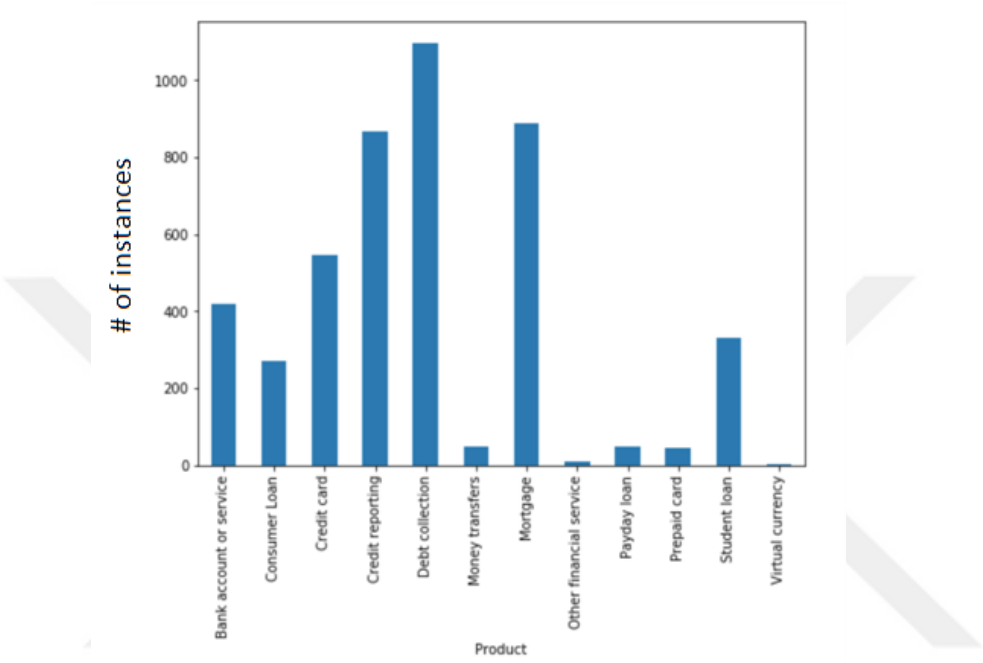

Figure 2.8: Distribution of the visits to a branch bank. (KDnuggets, 2018)

In Kaur et al. (2019), a comprehensive review is done to organize the proposed solutions for the imbalanced data classification problem. They analyzed 152 papers and they presented the studies in the categories: of application domains, approach types, learning algorithms, and performance measures. The main approach types of imbalanced learning are preprocessing methods, cost-sensitive learning methods, algorithm-centered approaches, and hybrid methods. Also, Haixiang et al. (2017) analyzes 527 papers and categorizes the solutions for the imbalanced data problem into 4 categories. According to the papers they found, nearly 30% of the papers are published in resampling techniques and we can say that the resampling technique is the most popular one among the different techniques. Resampling techniques are for generating new instances by utilizing the existing ones. While undersampling re-

moves the instances from the majority class, oversampling generates new synthetic instances with the help of existing instances. Hybrid methods employ both strategies at the same time with the aim of yielding better results.

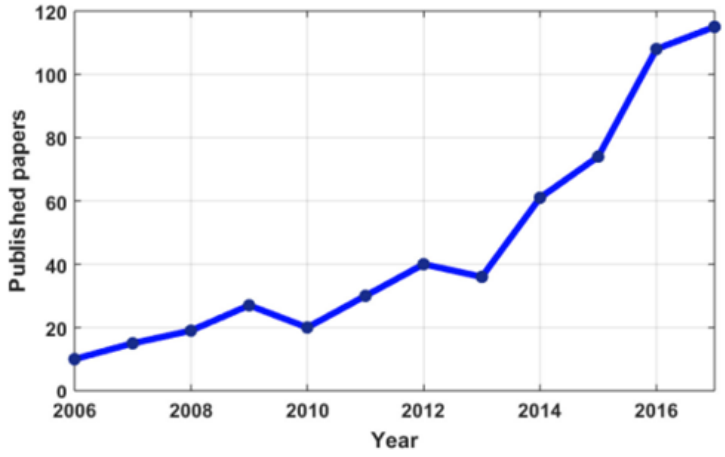

Figure 2.9: The number of published papers in imbalanced classification (Reprinted from Kaur et al. (2019)).

A well-known example of an oversampling method is Synthetic Minority Oversampling Technique (SMOTE) (Chawla et al., 2002). SMOTE creates new synthetic instances by the linear combination of current instances in the same neighborhood. Also, different versions of SMOTE are proposed like Borderline-SMOTE, DBSMOTE, MWMOTE, etc. (Fernández et al., 2018). SMOTE can be considered as a baseline approach and many other approaches are built upon it. Unfortunately, there are some drawbacks to SMOTE. It chooses instances randomly and results in oversampling of uninformative instances or noisy instances (Jiang et al., 2021). When there is overlapping between classes, it is hard or even impossible to find a distinction between different classes. So, it can be said that SMOTE is not a good solution for multi-label classification. Also in the cases where there is a curse of dimensionality in the data and the existence of real-time processing, SMOTE performs poorly (Fernández et al., 2018).

2.4 Imbalanced Multi-label Text Classification

In the real world, it is very difficult to find a balanced dataset. Especially in multi-label settings, this difficulty goes to extremes. Depending on the case, in most of the scenarios, the distribution of the class labels varies diversely. It is because there is great diversity in the real world. For example, for a classification task of animals, in the data collection process, the number of pictures taken will be different for each animal species. This is due to the nonuniformity between the population of the animals we can come across. In our living environment, the pictures would generally be of cats and dogs. Another example is the number of news reports. The number of news for each category differs from each other because of the interest of people or the agenda for the world. For example, most people are interested in entertainment, but very few are interested in fashion. In Figure 2.10, the number of news reported by different sources belonging to given topics is presented. It can be seen that the distribution of the topics varies for topics for each news source and also for total distribution.

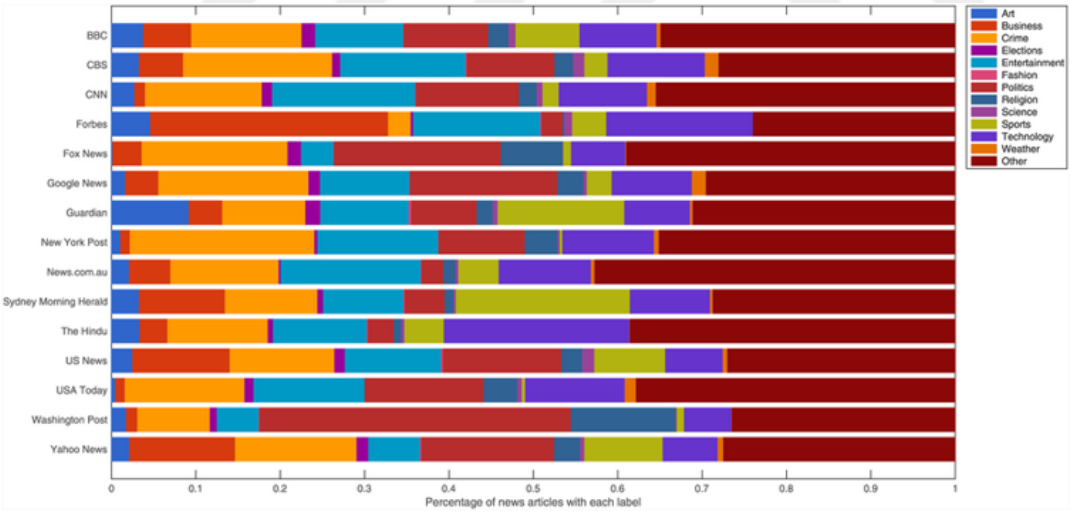

Figure 2.10: The distribution of news from different sources (Reprinted from Jia et al. (2016)).

Since most of the datasets in multi-label settings are imbalanced, there is a substantial need for a comprehensive solution. For the multi-label classification case, the

imbalance problem gets harder because of the nature of the multi-label classification, multiple labels can be assigned to an instance. In other words, there is no certain separation between classes. The lack of separation between classes makes classical oversampling approaches fail in multi-label classification cases. Classical approaches focus on classes one by one to oversample them by using the instances in the classes. When there is no divergence between classes, focusing on them one by one is not possible. If an instance belongs to different classes at the same time and if it is used as a reference point during an oversampling process, it is not possible to be sure that the newly created instances belong to other classes as well. To overcome this problem, there are some proposed methods for multi-label cases in the literature. In Tarekegn et al. (2021), a categorization approach is employed to group approaches for imbalanced multi-label classification. The methods are gathered in adaptation, resampling, ensembles, and cost-sensitive categories.

As indicated in the imbalanced data classification section, the most commonly preferred methods are resampling methods, for multi-label cases as well. Two different LP transformation-based resampling approaches are proposed in Charte et al. (2013), LP-RUS for undersampling and LP-ROS for oversampling. LP-ROS is the proposed oversampling method that clones the instances in minority label sets. These methods do not work well when the label combinations are distinct. Charte et al. (2015) proposes one undersampling and one oversampling method that focuses on the frequency of individual labels instead of label sets are proposed. ML-RUS deletes the instances in the majority class and ML-ROS clones samples in the minority class. On the other hand, these methods do not work well when the joint occurrence of minority and majority labels exists. In Charte et al. (2019), REMEDIAL is proposed to overcome this problem by decoupling majority and minority classes. Also, there are some heuristic approaches like MLeNN which is proposed in Charte et al. (2014). It utilizes the Nearest Neighbor algorithm, MLTL (Pereira et al., 2020) which utilizes the Tomek Link algorithm, and so on. The problem with oversampling algorithms, they generally copy the existing instances and copied instances don't add any predictive power to the algorithm and so the lack of data problem is still unsolved.

Cost-sensitive methods modify the cost function to put emphasis on the minority classes. Daniels and Metaxas (2017), Cao et al. (2016), and G. Wu et al. (2018) propose cost-sensitive approaches for multi-label cases. Besides, ensemble methods combine different base classifiers to optimize the final result. Tahir, Kittler, and Yan (2012), Tahir, Kittler, and Bouridane (2012), and Wan et al. (2017) are some examples of ensemble methods to solve imbalance problems in multi-label settings.

In Y. Luo et al. (2019), an oversampling method for text data is proposed using sequential generative adversarial networks (seqGAN). A GAN model contains a generator and a discriminator, and the mechanism functioning by the competition of generator and discriminator. The main goal of the generator is to create data that is close to the original dataset as much as possible in order to confuse the discriminator, while the aim of the discriminator is to separate the data coming from the generator and the original dataset. The discriminator returns feedback of generated data, so the generator can improve its performance in generating data. According to experiments, the proposed method outperforms other oversampling methods such as random oversampling, SMOTE, borderline SMOTE, and ADASYN. Shaikh et al. (2021) introduces a text generation algorithm built based on two powerful DL models; GPT-2 and LSTM. The generated text is completely new and retains the proper grammatical structure, context, and semantic information in relation to the original text. According to experiments, LSTM works great with short texts like sentences and GPT-2 works well with long sentences like documents. The main drawback of these studies, they use complex DL models like LSTM and GAN which require a great amount of initial data. In Chen et al. (2014), at first, using word2vec a continuous skip-gram model is trained using a large corpus to obtain word/POS pairs. The reason why part-of-speech (POS) is also used is that the same words with different POS lead to different lexical functions. Sentence vectors are obtained by summing all the word vectors in a sentence. Summing two-sentence vectors which belong to the same class also refers to a new vector that pertains to the same class as well. For all classes that are the minority, randomly selected two sentences are added together to create a new one until all classes have the same number of instances. The main problem in this study is that the assumption of summing two vectors may not result in a vector in the same class.

Moreo et al. (2016) suggests a version of latent space oversampling; distributional random oversampling is proposed. A generative function is created based on the distributional properties of the documents to return a vectorial representation of words. By calling this function, the desired number of new samples is created to balance the dataset. Mohasseb et al. (2018) uses SMOTE to oversample the question data to balance the data ratio. And NB classification is used to classify instances into categories.

Most of the proposed methods work with numerical data. They can be applied if the text data is converted to numerical form. The mentioned studies so far convert text data to numerical form by different methods. There are also some methods that use text data in raw format to oversample it. Their general approach is changing with synonyms or adding/removing random words from the text. The proposed method in Morris et al. (2020) is called TextAttack which is a framework for adversarial attacks, data augmentation, and adversarial training for text data. In the data augmentation tool, TextAttack's transformations generate perturbations of the input text and apply constraints to verify their validity. These tools can be reused to dramatically expand the training dataset by introducing perturbed versions of existing samples. Comparably, in Qiu et al. (2020), the developed tool, EasyAug is designed to help users to compare different text data augmentation methods and find the best one. Random oversampling, word-level transformation and variational encoding-based models are the main augmentation methods utilized in the tool. Y. Li et al. (2018) proposes a method that over-samples the minority class texts by two strategies: inversion and imitation. In the inversion process, artificial texts are generated by extracting samples from the majority class and reversing the sentimental words in the majority-class texts into their counterparts. In the imitation process, creating new samples from the original minority-class samples. In particular, given a minority-class text, a proportion of words are randomly selected and replaced by words that share a similar semantic meaning. Note that although the generated synthetic texts may not be grammatically correct, they should be semantically interpretable to some extent.

Other than mentioned strategies, there are also some studies that offer different solutions. In Jang et al. (2021), an adaptive distribution selection strategy is proposed. Instead of undersampling or oversampling methods which destructs the original representation of the data, a continual learning approach is proposed which refers to partitioning the training dataset into mutually exclusive subsets. Learning on these subsets considered distinct tasks is done sequentially by the order of similarity to the target distribution. During training on all subsets sequentially, the learner does not forget about the previous task, and parameters are optimized at the end. Shi et al. (2020) proposes a new general Multiple Distribution Selection instead of single softmax distribution which is inadequate for modeling complex and imbalanced data. MDS employs a mixture distribution that is composed of a single softmax distribution and a set of degenerate distributions to model imbalanced data. Firstly, words are converted to pre-trained word embeddings. To construct sentence embeddings, a max-pooling method which is a step in CNN is used. Using these embeddings, the feature extractor is trained and parameters are saved for the second stage. Using the parameters obtained from the first stage, distribution weights are obtained in the second stage. The proposed method automatically learns the weights of distributions, avoiding the artificial weighting of each category, and achieving better results than the cost-sensitive method. Tian et al. (2020) utilizes graph models to solve imbalance problems. In this study, the unknown distribution of the data is converted to a graph model. In the proposed method, first, the dataset is converted to a graph model, overlapping and disjunct degrees are calculated, and utilizing this graph-based imbalance index is calculated. Using the graph-based imbalance index, intrinsic characteristics of the data are considered and analyzed on the performance of imbalanced classification.

2.5 The Problem of Labeling the Data

While the data imbalance arises as a vital problem for ML, occasionally, the number of instances in some classes may not even be sufficient for the training process. When a classification algorithm is trained on imbalanced and insufficient data, it is not possible to obtain acceptable results for the classes that have insufficient data. Especially complex ML models always starve for more data (Severyn and Moschitti,

2015). For text classification tasks, generally, complex models are used because of the complicated nature of the problem. So, the need for enough data is certain to have a utilizable model for text classification tasks.

In the first stage, collecting and labeling new data is suggested to solve this problem but as researched in Fredriksson et al. (2020) collecting more data is sometimes not possible or too costly. Almost 80% of the effort in Data Science projects is related to collecting, labeling, and preparation of the data. This great effort is inevitable because most of the time data is unlabeled, poorly labeled, or, as we discussed, insufficient to train a working model. With the help of software systems, scraping tools, and scanners, collecting data is not a big deal anymore but labeling is. Labeling the text data, if there is no past data or an automatic labeling system at hand, requires an effort by human annotators. There are two aspects that increase the complexity of labeling text data. First of all, text data is one of the most complex data types to label. Unlike images, videos, or quantitative data, text data has semantics and requires reading and understanding effort to label. It requires more time compared to other data types. For image data, just seeing an animal in it or recognizing the written digit is just enough to label it but text data requires a great cognitive effort. Secondly, because of the reliability needed for the labels, the annotators should be experts in the related field. The annotators should know the business and context very well. For example, labeling legal text documents require effort from lawyers.

Because of the increasing demand for data labeling efforts, there are crowdsourcing alternatives to meet that demand but most businesses are skeptical about them because of several limitations. Privacy reasons, lack of experienced annotators, the problem of training the annotators, the need for cross-labeling to reach a consensus, and the required effort for ensuring the quality of the labels are the main reasons why companies do not prefer crowdsource labeling. In some cases, labeling the data requires a more complex process, so great effort and domain expertise are required in the fields such as law, medicine, technical documents, etc. For example, for most people, it is not easy to understand a legal document and so labeling it is. Martınez et al. (2022) investigates the reasons behind the difficulty of understanding legal texts. The study

shows that long sentences with clauses, unnecessary words that don't add any meaning, and rare words increase the complexity of the legal texts and so they decrease the clarity.

In order to emphasize the complexity of labeling domain expertise required texts, let us summarize the problems that increase the complexity and effort of labeling. In such texts, first, complex and long sentences are used very commonly. Secondly, juxtaposition, parallelism, and subordination are the techniques that are used for the sake of completeness, meticulousness, and clarity in these kinds of texts (Z. Li, 2019). These syntactic features make the texts hard to understand and interpret for the readers. Also, there are a lot of domain-specific terms, professional jargon, and stereotypes that occur in the text and it increases the required endeavor to understand the context. Another aspect is using the passive voice to express objectivity and impartiality. In Femmer et al. (2014), an experiment is conducted to measure the effect on the understanding of using passive sentences in a text by asking the students to frame a domain model. The results show that using passive sentences complicates the understanding and the participants missed the associations almost twice as the normal case.

The mentioned kind of texts require domain experts to label them and that need increases the cost and effort of the labeling process. For instance, labeling data in medicine requires deep medical expertise, so the doctors as annotators or law data require experts like lawyers, judges, or prosecutors. Especially, the importance and necessity of legal documents there are some research groups and projects that focus on legal documents to develop tools. Blackstone project, The Usable Privacy Policy Project, and Liquid Legal Institute are some examples of that projects and groups. The Usable Privacy Policy Project works on the website privacy policies. With the data policy regulations set by the governments, websites are obliged to share their legal notice on how they act with the user's data under the privacy policy title. Because of the legal obligations, these privacy policies contain a lot of legal terms and complex sentences to inform users. Unfortunately, they are complex plain texts that require a great amount of effort for users to understand and they are unlikely to read them. Because of the finding of those experts and cost limitations, labeling more data

looks unpromising and sometimes impossible as the first solution. Furthermore, if labeling more data is very costly or the data collection process is finalized because the project has ended. In most cases, the data collection and labeling phase is planned at the beginning of the project and the problems regarding the lack of labeled data are realized during the modeling phase. Unfortunately, it is too late to go back to the collection process and label more data. For cases like that, insisting on collecting more data will not help and block the progress of the project. In addition to that, even if it is decided to collect more data, the distribution of the data in the real world will be the same as the collected dataset, so labeling new data will not help to solve the insufficient and imbalanced data problems. Think of a scenario where only five of the one thousand labels belong to one class, should we label thousands of more instances to increase the number of instances that belong to the minority class? In that scenario, even spending a great amount of cost and effort, the number of instances will not be at the desired level and the distribution of the labels will remain the same which does not solve the insufficient and imbalanced label problem.

In cases where labeling more data is not possible, the so-popular self-supervised methods are proposed to utilize unlabeled data at hand. In self-supervised methods, a classifier is trained with the little labeled data at hand and labels the new instances by making predictions of unlabeled instances using the trained classifier. Thus, the labeled dataset is extended by using the trained model. In the literature, there are some studies that focused on this specific area and proposed several methods.

Pavlinek and Podgorelec (2017) proposed a self-training method based on LDA is proposed for document classification. ST LDA algorithm is a semi-supervised method to classify texts that have very few examples. It takes a few labeled documents and many more unlabeled documents as the initial set and produces a classifier that is trained with the final labeled set, as the output. Its aim is to extend the labeled set by adding new instances from the unlabeled document set by utilizing the similarity between unlabeled instances and centroids of the labeled set. It has two major steps, first, a self-training algorithm to enlarge the dataset, and another step for tuning the algorithm's parameters.

In D. Wu et al. (2017), a differential-evolution semi-supervised classification algorithm is proposed. Firstly, all the parameters of the model are initialized, and using the labeled dataset a set of initial classifiers are trained. From the unlabeled set, high-confident instances are selected and predicted with the trained classifier. Using the predicted set and originally labeled set, a differential-evolution-based selection procedure is applied to optimize the predicted set and added to the labeled set to extend the original labeled set. In Meng et al. (2018), a weakly-supervised method is proposed. It, firstly, generates new documents using the labeled ones and trains a model to share its parameters with a self-training classifier to label unlabeled documents. Guzmán-Cabrera et al. (2009) proposes a new self-training method for text classification that utilizes automatic extraction of unlabeled text from the web. In order to acquire new data instances from the web, a set of queries is constructed with the help of labeled examples for related classes. Using these queries, web scraping is performed to obtain unlabeled examples. Finally, the semi-supervised method is utilized to assign new instances to classes. Also, B. Zhang et al. (2007), Karisani and Karisani (2021), Lanquillon (2000), and Z. Xu and Iwaihara (2021) are the studies that employ self-training for text classification.

Also, there are some alternative approaches proposed in automated data labeling. Desmond et al. (2021) uses the active learning approach to label unlabeled data. Firstly, K-Means clustering is used to partition data into clusters. Data is iteratively served to the labeler and label spreading is applied to calculate probabilities for the rest of the data. H. Zhang et al. (2021) and Hajmohammadi et al. (2015) also employ active learning to automate the labeling process with the help of an annotator. M. Luo et al. (2019) and H. Li et al. (2021) are the methods that combine self-learning with DL. In M. Luo et al. (2019), a multi-labeled medical patent classification approach is studied. After preprocessing text, using Glove, embeddings are obtained and architecture is developed containing a bi-directional LSTM and a fully connected layer. Another method that is used in this context is zero-shot learning (Meng et al., 2020, Ye et al., 2020). In zero-shot learning, a model is trained on a dataset with specific labels, and this method is used to predict instances that belong to unseen classes. Fur-

thermore, there are some studies that combine different approaches with self-training such as K-NN, evolutionary algorithms, and few-shot learning (J. Li and Zhu, 2020, Donyavi and Asadi, 2020), and Mukherjee and Awadallah, 2020).

There are some drawbacks to self-supervised methods that make them impracticable in some cases. First, if the data is not enough to train an initial classifier with acceptable results, it is not possible to make predictions for labeling new instances. Second, self-labeled techniques follow an iterative procedure, aiming to obtain an enlarged labeled data set, in which they accept their own predictions to be correct. If the trained classifier does not perform well on the initial set then the predicted labels will not be accurate as well. The second case is also related to the first one because when there is not enough data to train a classifier, that classifier performs poorly. As mentioned previously, our starting point is that the labeled data at hand is so insufficient that it is not possible to train an acceptable initial classifier. When it is not possible to train an initial classifier, self-supervised methods become impractical. Especially, the possibility of occurring this problem in the multi-label case is quite high.

2.6 Similarity Functions for Text Data

Similarity measures are functions that are used to measure the matching score between two objects. It calculates a single real value to indicate a score to show how similar two objects are. The more similar a pair of objects, the higher the similarity score produced by the similarity measure. It can be considered as the inverse of the distance function. In NLP, similarity measures are used to measure the similarity between documents, sentences, or words. The similarity of texts can be measured in two ways. Lexical similarity analyzes the resemblance of the words and their sequence. String-based methods are used to measure lexical similarity. For example, ‘apple’ and ‘apply’ are lexically similar and their similarity score will be high. On the other hand, semantic similarity measures the similarity between meanings. Knowledge-based or corpus-based models are used to measure semantic similarity. The semantic similarity between ‘apple’ and ‘apply’ will be low because they represent completely different things.

J. Wang and Dong (2020) presents a detailed summary of text similarity functions. The authors divided the survey into two categories; text distances to measure semantic similarity and text representation to measure lexical similarity. Text distances are classified into length distances, distribution distances, and semantic distances. In length distance, the distance between vectors of texts is calculated. The most popular length distance types are euclidean, cosine, manhattan, and hamming distances (Deza and Deza, 2009; Norouzi et al., 2012). Distribution distances are used to understand whether different text pieces come from the same distribution. JS divergence, KL divergence, and Wasserstein distance are the main examples of distribution distances (Nielsen, 2010). Lastly, semantic distance is to understand the semantic similarity between texts. Word Mover's distance measures the required minimum distance to reach a word from another word in semantic space (Andoni et al., 2008). Gomaa, Fahmy, et al. (2013) is also a study that classifies the similarities in the same way. In the study, they put the similarities under three categories: string-based, corpus-based, and knowledge-based.

In Magara et al. (2018), they studied text similarity measures and conducted a comparative analysis between them. Cosine similarity, Euclidean distance, Jaccard coefficient, and Pearson correlation coefficient are used to measure the similarity between research papers for a recommendation algorithm. According to their experiments, cosine similarity yields the best results.

The Minkowski family is a well-known similarity measure for various problems. Manhattan, Chebyshev, and Euclidean distances are the specialized versions of the Minkowski family (C. Wang et al., 2017). The intersection family, inner product family, fidelity family, and Shannon's entropy family are the other similarity measure families that are used to measure similarity for text data. In C. Wang et al. (2017), they investigated 53 similarity measures and found that, in general, cosine similarity performs well.

The similarity function plays an important role in the design of our algorithm. The new instances are found with the help of similarity and choosing the right similarity function has vital importance. The similarity function should reflect the relations between instances correctly. We mention that the embedding methods represent the text data in a high-dimensional vector space and they capture the semantics of the text. The similarity between these embedding vectors will show the semantic similarity of the text and it is possible to capture the similar instances which belong to the same class.

The widely used measures for text cosine similarity, word mover's distance, Jensen Shannon distance, Minkowski family and its extensions: euclidean distance, manhattan distance, and Chebyshev distance. We prefer to use measures and conduct an experiment to find the best similarity measure that works best for our proposed algorithm. While using similarity measures, we also employ some distance functions as well. The equations for euclidean distance, cosine similarity, and Jensen Shannon distance are given in equation 2.14, 2.15, and 2.16 respectively. To convert distance functions to similarity, we use Equation 2.17. In these equations, p and q represent two vectors and p_i, q_i stand for components of vector p and q. d stands for distance function and lastly, s represents the similarity function. D in Equation 2.16 represents Kullback-Leibler divergence.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_i - q_i)^2 + ... + (p_n - q_n)^2} \quad (2.14)$$

$$s(p,q) \coloneqq cos(\theta) = \frac{p \cdot q}{\|p\|\|q\|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}} \quad (2.15)$$

$$d(p,q) = \sqrt{\frac{D(p\|m) + (D(q\|m)}{2}}, \qquad m = \frac{(p+q)}{2} \quad (2.16)$$

$$s = 1/(1+d) \quad (2.17)$$

CHAPTER 3

THE PROPOSED METHOD

In this chapter, we start by explaining our motivation to develop an algorithm and after that, we demonstrate how our algorithm works. Under the overall algorithm title, the general mechanism of the algorithm is explained. General steps for the text classification tasks: preprocessing, converting text to numerical format, and training a classifier are part of the overall algorithm. The second header is for the oversampling algorithm which is a part of the overall algorithm. We explain the parameters and working mechanism of the oversampling algorithm to conclude this chapter.

3.1 Motivation

As we discuss in the previous chapter, there is a great need for auto-labeling methods to solve imbalanced data problems without making a great effort. Real-world datasets are generally highly-imbalanced and sometimes insufficient to train an ML algorithm and resulting in poor models. Text classification is one of the most affected applications by this lack of data problem in ML. As a solution, there are some methods proposed in the literature. The first solution that becomes prominent is labeling more data. The complexity of labeling the text data, especially in domain-specific fields, the status of the project, and finding experts in the problem-specific fields make this solution inapplicable. As an alternative solution, synthetic data oversampling methods are proposed and these methods do not work well because text data is a kind of unstructured data that has a complex structure. Text data has semantics which makes it more complex to interpret by computers. So, the classical oversampling methods struggle to solve this complex problem.

In contrast to labeling, collecting more data is much easier with the help of a growing number of web scraping tools and the eternal source of data, the internet. A method that helps us or automates labeling efforts will provide great convenience and solve a complex problem that is very prevalent in ML. At this point, we propose a similarity-based oversampling method that finds new instances from unlabeled instances to solve imbalanced and insufficient data problems in multi-label text datasets. Our method uses word embeddings to convert text data to numerical format and fine-tunes embeddings to increase the similarity between in-class instances while decreasing between-class instances. It utilizes a similarity function to find similar instances from the unlabeled set and follows an iterative process to check the newly added instances whether improved the overall performance measure or not. It searches the unlabeled set iteratively and finds the possible instances that can be added to the labeled set.

3.2 The Overall Algorithm

In our proposed method, as in the self-training approaches, we aim to oversample the dataset by using unlabeled instances by utilizing a similarity function between instances. In self-training approaches, an initial classifier is trained to predict unlabeled instance labels to add them to the labeled set. Differently from this, we cannot trust the predicted labels by an initial classifier due to the bad results because of the lack of instances for some classes. So, by utilizing a similarity function that maximizes the within-class similarity while minimizing the between-class similarity, it is possible to find new instances that have high similarity to the labeled set. In order to separate classes clearly from each other, the within-class similarity should be maximized and between-class similarity should be minimized. The main motivation behind using the similarity function is that the instances that belong to the same class are placed close to each other in the vector space and instances that belong to different classes are placed further relatively. The similarity function should be able to capture this relationship and the embedding vectors should reflect the relation by their position in the vector space. So, the similarity function is another parameter of the algorithm and the best performing similarity function is found in the experiments.

Our proposed solution is designed for the case when the below issues arise:

1. When the labeled data at hand is very few to train a classifier and/or the result obtained by training a classifier is not satisfactory for the desired task.
2. Labeling more data is not possible because of cost concerns or project status. Also, if there is a belief for labeling more data will not solve the data imbalance or data insufficiency problem because the real distribution of the data is also highly-imbalanced. We discuss the challenges in Chapter 2.
3. Unlike labeling more data, collecting data is not a big problem. Overall, labeling the data requires great effort, and collecting more data is an easier task thanks to the web scraping tools.
4. The data is in multi-labeled format.
5. The relation between instances can be captured by a similarity function.

In the real world, the mentioned problems arise too often and most ML projects fail because of these challenges. An auto labeling tool that does not require human effort to label more data that sorts out these challenges will have an important place in the literature. We believe our proposed solution will be a baseline idea and it will open doors to new ideas. The flow of the overall structure of the proposed algorithm is given below.

To begin with, some portion of the labeled dataset at hand should be separated for test purposes. In order to validate whether a model works or not, some portion of the dataset, in which the training algorithm has not seen, should be used as the test set. The test set is used for only test purposes and all the preprocessing and fitting operations should be done only with the training set to prevent data leakage. The parameters should be trained with the training set and then by using trained parameters, the test set should be transformed into its preprocessed form. Data leakage is a major problem in ML which falsifies the results and shows the results better than the actual and misguides the modeler (Papadimitriou and Garcia-Molina, 2010). It arises when there is information leakage, like a parameter or range of a feature, from the train set

to the test set. In order to prevent data leakage problems, we split the dataset as a training and test set before starting any kind of preprocessing.

Algorithm 1 The Overall Algorithm

1: Split labeled set as train and test
2: Apply text preprocessing and convert to numeric form (unlabeled, train and test set)
3: Lowering and noise removal
4: Stop words removal
5: Tokenization and lemmatization
6: Vectorization
7: Fine-tune embeddings for training set to optimize in-class and between-class similarities
8: Train an initial classifier as a reference point
9: Parameter optimization and oversampling
10: Design an experiment to optimize parameters
11: Oversampling the dataset using the Algorithm 2 with optimized parameters
12: Train a final classifier to compare results
13: Yield the new labeled set

After splitting the dataset as training and test sets, a preprocessing step is applied to clean, standardize, and arrange the text before converting it to embeddings. The first step of the preprocessing is the lowering operation which is applied to lower all the text to reduce the effect of case sensitivity. Programming languages are case-sensitive and interpret the same words in different cases differently. However, it is not important whether a word is lower-sized or upper-sized from a semantics perspective. So, lowering operations will resolve this confusion. In the second operation, the noisy pieces are removed from the text. It is considered that punctuations, HTML tags, URLs, and emails as the main noise sources, and removing them from the text is necessary for more accurate classification. For example, we can consider that HTML tags are the leftovers from the scrapping process or URLs are the website addresses given for user access but do not have any effect on the meaning which makes the noise. Also, emails, addresses, and words containing numeric can be considered in this way

as well. These words generally represent abbreviations or pieces of an address which means they carry no important information semantically. All these noisy text pieces are removed from the text.

The next step in the preprocessing phase is removing stop words. There are some open-source libraries that remove stop words from the text. These libraries gather all the stop words in English and all of the stop words are removed from the text. We prefer the NLTK tool for preprocessing operations. NLTK is a widely used tool for NLP practitioners and is designed for different kinds of text preprocessing operations (Loper and Bird, 2002). Moreover, when we look at our data corpus carefully, we realized that there are some words that do not carry any meaning and should be added to the stop words. So, we extended the stop words list by adding these words manually. The last step of the noise-removing operation is if there are single chars in the dataset, they are also removed before going to the next steps.

After completely removing the noisy pieces from the text, the next step is standardizing the text. All the words in the text should be in the same standard format. In English, the same words can be in different forms like adjectives, adverbs, etc. by adding suffixes or prefixes. For example, 'take', 'took', and 'taken' are the three different forms of the same word and represent the same action. A word can be in different forms and using all these forms at the same time conduces to confusion in the vectorization step. So, to prevent this, a standardization process is needed to degrade the different forms of words into a single base form. As the first step of standardization, tokenization is a process of splitting texts into small text pieces like words, subwords, etc. After tokenization, the individual pieces, which are called tokens, can be processed. In order to obtain the standard base form of the words, Lemmatization or Stemming, which are the most used methods, are applied. Lemmatization stands for converting words to their base forms by analyzing words morphologically. Stemming is a heuristic process to obtain the root of a word by removing morphological variations. Compared to stemming, lemmatization performs better (Balakrishnan and Lloyd-Yemoh, 2014). Some applications use stemming and lemmatization together but to make it simple, we decided to use lemmatization only, which performs better

overall. After obtaining the lemmas a length checking function is applied to filter the words that are too long or too short. Words that have more than 25 chars in length and less than 2 chars in length can be considered noise and removed from the text.

The preprocessing step is done and the dataset is now ready for applying embeddings to obtain vectors. In the literature, many embeddings are proposed that reach great performance for representing words in numerical format. Selva Birunda and Kanniga Devi (2021) reviews embeddings such as count vectors, tf-idf, Word2vec, Glove Hugging Face, Elmo, Bert, etc. To use the state-of-the-art embeddings there are some open-sourced and commercialized tools. These tools provide APIs for users to utilize pre-trained state-of-the-art embeddings. Huggingface is a community that researches and implements state-of-the-art embeddings for NLP tasks. Their transformers are one of the most popular models that are used in the NLP world (Wolf et al., 2019). For example, their most downloaded model has been downloaded more than 37 million times and the well-known Bert model is downloaded more than 22 million times. Another well-known open-source research community, also the releaser of the well-known GPT-3, is Open-AI (Floridi and Chiriatti, 2020). Open-AI is conducting research in AI for the human good. They are the leading communities in this field and most of the top companies use their tools and also develop open-source models as well. These tools are our main resources for implementing pre-trained embedding models. From HuggingFace and Open AI, it is possible to use the most popular models via their APIs. We use their models to find the best-performing embedding model for our dataset. The full list of the embeddings we tried during our experimentations can be found in Appendix A. Since this is not in our study of interest we decided to continue with a good-performing one for our modeling approach. The main performance criterion here is the optimization potential of the similarity between instances.

In order to make our oversampling method perform well, the similarity between within-class instances and between-class instances should be discriminated well. We expect to see a high similarity score for the instances that belong to the same class and a low score for the ones in different classes. As explained in Section 2, the embeddings convert the texts into the numerical format in a way that the similar texts

are placed close to each other in the vector space, and vice versa. Our aim is to distinguish the instances in the same area with the help of the similarity function. In a vector space, if we can find the unlabeled instances that are placed in the area where a class's instances are placed intensively, then we can label them as potential instances for the related class. If we fine-tune our embeddings so that vectors of the instances that belong to a class should be placed closely and the ones that belong to different classes should be placed far from each other. In that way, within-class and between-class similarities are more distinguishable, so the instances belong to different classes.

In Figure 3.1, a model in two-dimensional vector space is presented to demonstrate the explanations. In the Figure, a sample of labeled instances that belong to the same class are marked as red circles and it can be clearly seen that they are located close to each other. The positions of the instances in the vector space are optimized by fine-tuning the embeddings. By maximizing the in-class similarities and minimizing the between-class similarities, the instances that belong to the same class locate close to each other. The unlabeled instances, which are marked as a cross, are distributed all around the space but our algorithm will try to find the closer instances to consider them as potential instances. The blue points are in the interest of our algorithm and the algorithm will try to find the ones that help to improve the overall performance. The green points are insignificant for the class that is oversampled.

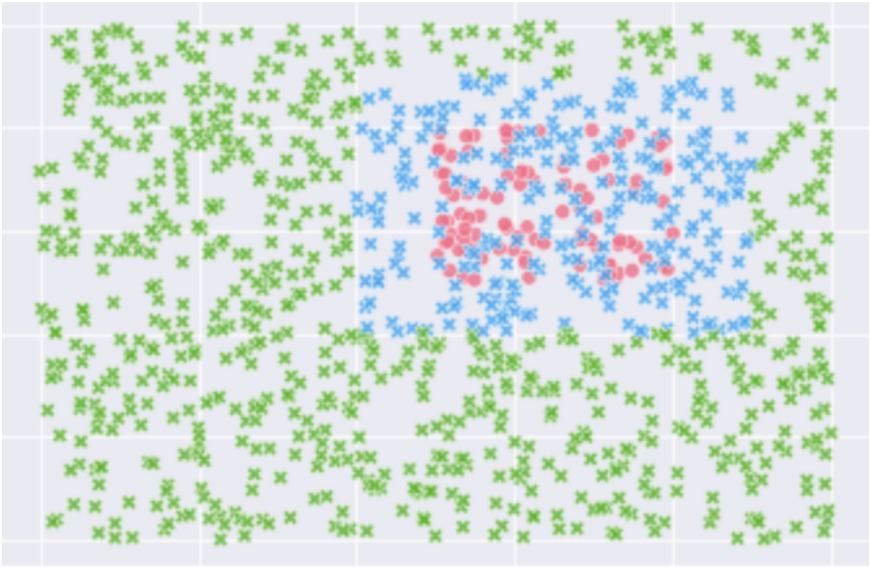

Figure 3.1: A representation of the labeled and unlabeled instances.

As a reference point, to understand our proposed method and find new instances accurately, a control mechanism is required. As the reference point, an initial classifier is trained to observe the current performance level of the dataset. After completion of the oversampling method, it is possible to measure the performance improvements by comparing the final results with the initial classifier. After training the initial classifier, the data is oversampled using the Algorithm 2. The details of the oversampling algorithm are described in Section 3.3.

The next step is optimizing the parameters of the oversampling algorithm given in Algorithm 2. It has several parameters and the performance of the oversampling algorithm strongly depends on the parameter setting. So, there is a need for parameter optimization to improve the performance of the algorithm. It is not possible to make a suggestion for parameters since different datasets have different characteristics and the parameter settings can vary for each dataset. So, we strongly suggest optimizing the parameters to improve the performance of the algorithm. Each parameter of the algorithm and possible values they can take is explained in Section 3.3. The details of the strategy we employ for parameter optimization are given in Section 4.

Lastly, a final classifier is trained to compare the final results with the initial results. If the final results are improved, we can say that our proposed algorithm performs well of finding new instances from the unlabeled set. The flow chart of the overall algorithm is presented in Figure 3.2.

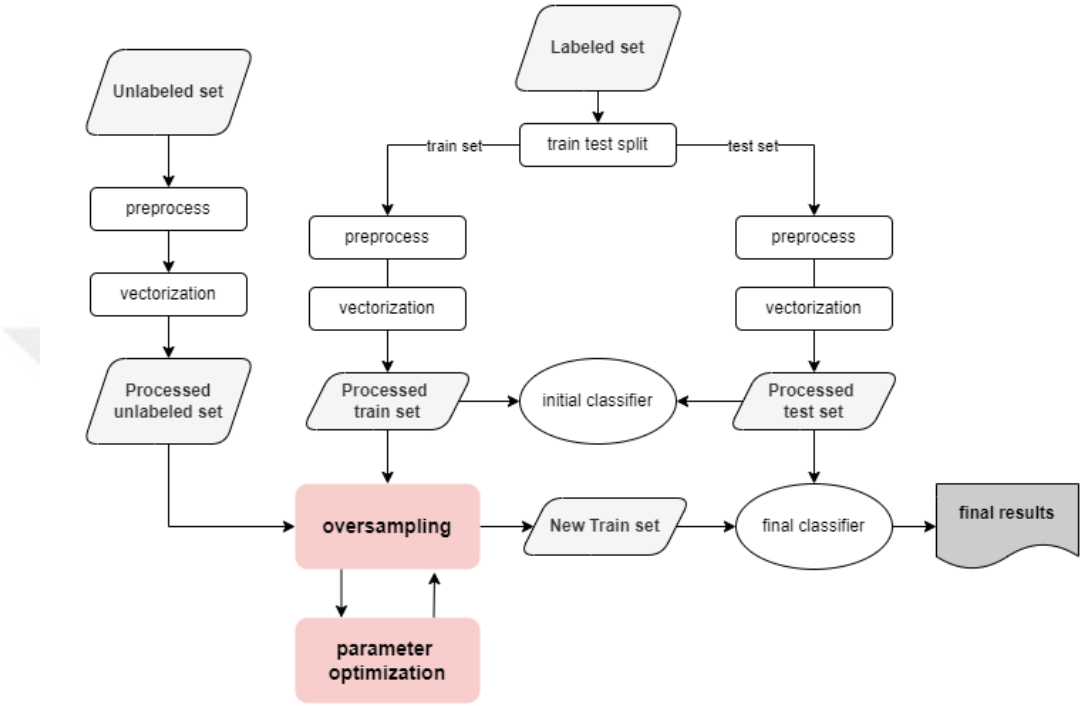

Figure 3.2: Flow chart for the overall algorithm.

3.3 The Oversampling Algorithm

The proposed oversampling algorithm tries to find new instances from the unlabeled set by utilizing the similarity scores calculated with the labeled set. The instances that belong to the same class, in other words, the instances that have similar meanings, tend to be close to each in vector space as well (Clark, 2015). By measuring the similarity between these vectors, it is possible to find the instances that are close to each other. With this logic, our oversampling algorithm searches the unlabeled set and finds instances that are close to the existing ones as depicted in Figure 3.1. These closely placed instances are candidate instances for labeling to extend the labeled set. An iterative improvement checking procedure is used to check whether newly

added candidates improve the overall performance or not. Iteratively, it searches the unlabeled set to find new instances until a number of iterations are repeated. The pseudocode for our oversampling algorithm is presented below.

Algorithm 2 The Oversampling Algorithm

```
Calculate class similarities
Calculate similarity factors
Train a baseline classifier
for number of iterations do
    Calculate the required number of instances with a performance measure
    Find candidate instances
        Calculate selection probabilities for classes using the required number of instances
        Select a class randomly using the selection probabilities
        Find a batch of instances from the unlabeled set that has greater similarity than the similarity threshold calculated by class similarity times similarity factor
    Train a new classifier to look for improvement
        If the results get better
            Add instances to the labeled set
            Increase similarity factors
        If the results do not get better
            Decrease similarity factors
    Update unlabeled set and iteration number
end for
```

The oversampling algorithm has some parameters that shape the functioning of the algorithm. Similarity calculation type, similarity function, batch size, number of iterations, balance ratio, and performance measure are the parameters of the algorithm. The algorithm takes the labeled set, unlabeled set, and the parameters to oversample the dataset and it returns the labeled set, unlabeled set, and metric history as output. The final labeled set is oversampled set and the unlabeled set is the remaining unlabeled set that the algorithm did not consider as similar points. Metric history is an array and it is the list of performance measures that are measured during the over-

sampling step at each iteration. The history of measures will be used for performance analysis.

The algorithm starts with calculating the class similarities. Class similarities are calculated for each class individually using all the instances in a class. The binary combination of the instances is used to calculate the similarity between two embedding vectors with the preferred similarity function. There are alternatives for the similarity function like euclidean similarity, cosine similarity, etc. in the literature, and it is a parameter for our method. Any kind of similarity function is applicable at this point and in the experimentations, the best performing similarity function will be found. After calculating the similarity between vector pairs, an array of similarity scores is present at hand and there are two alternatives to degrade this array of similarities to a single score. This is another parameter of the algorithm which is called similarity calculation type. We defined two alternatives for similarity calculation type: 'average' and 'safe interval'. If 'average' is used, the average of the array of similarity scores is calculated and returned as the final similarity score for the class. Alternatively, 'safe interval' is tighter compared to 'average' and it finds the 3rd quartile of the similarity score array. The 'safe interval' will produce higher similarity scores compared to the 'average' in most cases. It is because the mean is expected to be less than the 3rd quartile in the normal distribution.

In the second step of the algorithm, similarity factors are calculated with class similarities and it is an adjusting parameter for acceptance of the instances as the potential for labeling. Class similarities are constant values calculated with the training set at hand at the beginning and cannot be changed during the iterations. In order to make our algorithm more flexible, we defined a variable, similarity factor, that can be adjustable. In the beginning, it is calculated with the below Equation 3.1 and adjusted in each iteration depending on whether new instances are added or not. When the new instances are added to the labeled set, the similarity factor is increased to tighten the acceptance level and decreased to loosen the acceptance level. So, it is a parameter to control the acceptance mechanism.

$$f_i = (1/s_i)^{0.5} \tag{3.1}$$

In Equation 3.1, f_i represents the similarity factor for class i and s_i stands for similarity of class i. In this formula, an increase in the similarity will cause a decrease in the similarity factor. If the similarity, which takes values between 0 and 1, is close to 1 the similarity factor takes a value very close to 1 and does not affect the acceptance level too much. If the similarity is close to 0, which means no similarity, the similarity factor will be high and the acceptance level will be driven up to tighten the acceptance level. With the help of the similarity factor, it is possible to configure the acceptance level. We believe that the acceptance level should be adjustable to be sure that only similar instances are considered candidates. If the class similarity is very low for a class then most of the instances are welcomed as candidates, which is not the desired case.

The next step is training a classifier to obtain initial performance measures as a baseline. These initial performance measures will be used to calculate the number of required instances to use only in the first step of the loop. The number of instances will be calculated in each step of the loop with the updated labeled training set. In the next step, the algorithm will loop until the given stopping conditions are met. The first stopping condition is met if the number of instances in the unlabeled set is run out. In other words, the algorithm considers all the instances in the unlabeled set until no instance is left for the next iterations. The second stopping condition is looping for a certain number of iterations. The first stopping condition is hard to meet because the algorithm should iterate over all the instances for all the classes to stop. So, we limit the number of iterations to set an early stopping condition. In that way, we can control our algorithm for how long it will run in our analysis. As an alternative to looping for a certain number of iterations, a convergence condition can be defined. In this condition, the algorithm will stop if there is no improvement in the results for a certain number of iterations. The algorithm will converge and the results will not get better.

The first step in the loop is calculating the number of required instances. It is the number of instances required if the dataset was not imbalanced. Ideally, a dataset is balanced if all the classes have the same number of instances. In other words, the imbalance ratio, the ratio between the number of instances for each class is 1 (Amin et al., 2016). This is the ideal scenario in classification problems but in the real world, it is very unlikely to have such a ratio. Still, it is desired to have a close ratio to 1 to prevent performance problems because of the imbalancedness. In most cases, it is not easy to reach that, especially for the highly imbalanced datasets. For some cases, where the positive class is a very rare event like fraud detection, disease detection, etc., the imbalance ratio can reach up to one out of thousand or one out of ten thousand. In such cases, it is nearly impossible to balance the dataset because finding the rare cases is very hard. So, reaching the ideal scenario is almost impossible and what should be the imbalance ratio is still a big question in this case. In our method, in order to find this ratio experimentally, we defined a parameter which is called the balance ratio. This parameter controls the number of instances to balance the dataset. For example in a binary classification case, if the number of instances is 10 and 40, respectively for positive and negative classes, the number of required instances is 30 for the positive class to reach to 0.5 balance ratio. The formula to calculate the number of required instances to balance a class is given in Equation 3.2. The output of this formula can be negative for the classes there is no need for oversampling. Our algorithm will ignore these classes and will only focus on the classes that need to be oversampled.

$$(n \cdot r - n_i) \cdot 2 \tag{3.2}$$

In Equation 3.2, the n represents the total number of instances in the dataset. r is the balance ratio, and n_i stands for the number of instances for the minority class. With formula 3.2 it is possible to calculate the number of required instances to balance a dataset numerically. In other words, it tries to balance the number of instances in each class equally. In some cases, while an imbalance problem in the dataset arises, the performance measures still can be good. Despite the dataset suffering from the imbalance problem, the output of the classification does not suffer. So it is not necessary to oversample that class to be in search of performance improvement.

Or, congruently, the instance numbers can be adjusted by the performance measure. So, in order to focus on the classes both imbalanced and have a poor performance in the classification task, we introduced Equation 3.3 to calculate the required number of instances. In Equation 3.3, the performance measure is brought into play and has an effect on the number of required instances.

$$max(0, (n \cdot r - n_i) \cdot 2 \cdot \rho) \tag{3.3}$$

ρ represents the performance factor in Equation 3.3. If the performance measure increases ρ needs to decrease and vice versa to be able to reflect the effect of the performance measure on the formula. For example, the performance measure is chosen as the accuracy score and it is maximized for better performance. The maximum score of accuracy score is 1 which means a perfect classification for that class. In that case, the number of required instances to oversample should be zero since the best possible performance is reached. Or conversely, when the accuracy score is close to 0 and the number of instances in that class is insufficient and so the number of required instances should be high. Accordingly, we set ρ to $1 - performance\ measure$ where the performance measure is maximized and equals to $performance\ measure$ where it is minimized. For the multi-label classification case, Equation 3.3 is calculated for each class using the related class's performance measure. The maximum function is used to ignore if the number of required instances is negative. A negative score means undersampling and in our case, our algorithm will ignore these classes to just focus on the classes that need to be oversampled. So, the formula is put into a maximum function to get rid of the negative results.

At this point, we need to clarify that the number of required instances is not the number of instances that the oversampling algorithm tries to reach. As mentioned, it is the ideal scenario but unfortunately, it is not possible to find that number of labeled instances. The main aim of our algorithm is to find as many possible instances to balance the dataset with the help of this number of required instances. In the best scenario, if the algorithm finds enough instances to balance the dataset, it will stop iterating for that class. So, we used these instance numbers as the priority score and our

algorithm will focus on the classes considering this priority score. In the next step, the selection probabilities are calculated by using the required number of instances. It is calculated by normalizing the number of required instances for all classes within the 0-1 range. Hence, the class that has the highest number of required instances will have the highest selection probability. Using the selection probabilities, the algorithm chooses a random class to oversample. This mechanism is designed to give higher chances to the classes that are highly imbalanced and have poor performance measures. This selection is done in every iteration and the algorithm will try to find new instances for the chosen class. The reason why a probabilistic process is chosen instead of a deterministic process that focuses only on the prior classes is that the algorithm would miss instances that have a higher potential to improve overall performance. If a deterministic process is designed, the algorithm will focus on only one class for a certain number of iterations, and focusing only on that class may harm the overall performance while the focused class is being improved. Put differently, we can associate this mechanism with the exploitation and exploration trade-off in heuristic algorithms. Exploration is searching for new solutions across the feasible space to diversify the solution set. On the other hand, exploitation is searching for new solutions in the local optima to intensify the solution set (Črepinšek et al., 2013). In our method, we can link the probabilistic selection of the classes to exploration and deterministic selection to exploitation. Since our algorithm search for the new instances iteratively the need for exploration is more important than exploitation. Using a probabilistic selection strategy will add to the exploration mechanism and would benefit our algorithm to find new good instances.

The selected class is the prior focus of the algorithm and the next step is finding a batch of potential instances from the unlabeled set. The unlabeled set is shuffled in each iteration and a loop iterates over it to find the most similar instances for the current class. For each instance in the unlabeled set, the similarity between the instance and the selected class is calculated. The acceptance criterion is the similarity between an unlabeled instance and the selected class should exceed the similarity threshold. The similarity threshold is calculated by multiplying the class similarity and similarity factor for that class. As explained in section 3, the class similarity is the calculated

similarity for the related class. The similarity factor, f_i, is calculated with Equation 3.1. The similarity threshold is the filtering mechanism to choose similar instances. Also, the similarity threshold is adjustable by updating the similarity factor in each iteration. It has two components: class similarity and similarity factor. The class similarity is constant during a run because it is calculated with the training data at the beginning. The similarity threshold is initialized at the beginning also, but it is adjusted in every iteration according to the improvement in the performance measure. The batch of instances that have high similarity to the class is now the candidate instances for labeling.

Candidate instances are now the potential instances that are labeled by the class label if they contribute to the performance. Since our problem is for a multi-label classification case, the labels for the other classes are determined with the same logic. If the similarity of the instances in the batch exceeds the similarity threshold for the other classes, they are also potential instances for those classes as well. So the potential label vectors for the candidate instances are prepared. A classifier is trained with the combination of labeled data and candidate instances with their potential labels. The chosen measure is calculated for the test set with the trained classifier. If the result gets better compared to the previous result, the candidate instances are added to the labeled set. It can be said that the instances that have high similarity to the existing ones help to improve the overall score and can be added to the labeled set. If the performance is not improved then the instances are not added.

In the next step, after deciding whether the candidate instances are added or not, the similarity factor is updated accordingly. It is mentioned that the similarity factor controls the filtering mechanism of finding the new potential instances. It is used in the calculation of similarity thresholds directly and has the effect of choosing similar instances. If the potential instances help to improve the performance and are added to the labeled set, the similarity factor is increased as indicated in Equation 3.4. Otherwise, if the instances do not help performance improvement and are not added to the labeled set, the similarity factor is decreased to loosen the acceptance filter (Equation 3.5).

$$f_i = f_i \cdot (1 + (1 - f_i)^4) \tag{3.4}$$

$$f_i = f_i \cdot (1 - (1 - f_i)^4) \tag{3.5}$$

The last step in the iteration is updating the iteration number and removing the candidate instances from the unlabeled set. No matter whether the candidate instances are added or not to the labeled set, they are removed from the unlabeled set. So that the algorithm will not make calculations for the same instances every time. It is because the algorithm checks for all potential labels that can be given to them in the previous step and clears off the possibility of being a potential candidate in the next iterations.

After oversampling algorithm is completed, the oversampled data is returned to the overall algorithm. The final classification is trained with the oversampled data to compare the performance obtained with the initial version of the data. A flow chart of the proposed oversampling algorithm can be seen in Figure 3.3.

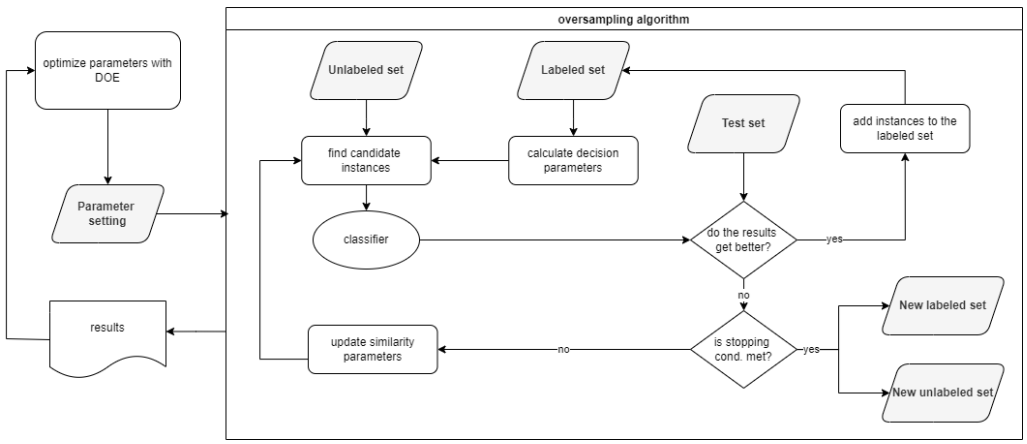

Figure 3.3: Flow chart for oversampling algorithm.

CHAPTER 4

EXPERIMENTAL RESULTS

In this section, we start with a toy example to explain our proposed method for the sake of clarity. In this toy example, the steps of the algorithm will be described with examples. Also, the performance measures for multi-label classification are presented, and why we chose the f1-score as our main performance measure is explained. In order to optimize our algorithm's parameters, we set up an experimentation design on a dataset. The details of the experimentation, the important parameters of the algorithm, and finding the optimum parameters to improve performance are presented. Finally, a comparative analysis is conducted to see whether our algorithm performs well or not.

4.1 A Toy Example

An example of our proposed oversampling method will be presented in this chapter. The steps of the algorithm will be more intelligible with the help of this example. Since this is a toy example and presented for a better understanding of the recommended method, some steps will be ignored which we think do not require explanation. We use some samples from the sentiment analysis dataset obtained from UCI Machine Learning Repository. A set of movie reviews taken from the IMDB webpage is presented and these reviews are labeled as positive and negative according to their sentiment. A sample set is randomly selected from this dataset for our toy example.

Let's assume that the labels for the instances presented in Table 4.1 are known and we want to find new instances from the unlabeled set, which can be seen in Table 4.2. As mentioned, our algorithm's main aim is to extend the labeled set with the instances

from the unlabeled set by utilizing the similarity between instances. The first step to do this is splitting the labeled set as train and test sets. Totally, we have 6 instances for negative class and 3 instances for positive class, and the dataset is randomly split as 6 for training and 3 for testing.

Table 4.1: A sample for labeled instances from the sentiment dataset.

Text	**Label**	**Set**
It was so BORING!	negative	train
It's a long time since I was so entertained by a movie.	positive	train
The good cinematography also makes her, and Monica Bellucci look very beautiful.	positive	train
This movie is a solid example of a bad plot and a very, very bad idea all the way.	negative	train
All in all, its an insult to one's intelligence and a huge waste of money.	negative	train
The acting is terrible, and the writing is worse.	negative	train
To be honest with you, this is unbelievable nonsense and very foolish.	negative	test
This is definitely a cult classic well worth viewing and sharing with others.	positive	test
Top line: Don't waste your time and money on this one, its as bad as it comes.	negative	test

After the lowering operation, all the noisy words like stop words, punctuations, and emojis are removed from the text and the lemmatization operation is applied. The text is now cleaned and standardized to be ready to convertible to vectors. The text is converted to vectors and fine-tuned to increase in-class similarities while decreasing between-class similarities. Lastly, dimensionality reduction is applied and final vectors are obtained before starting the oversampling algorithm. Since this is a toy example to demonstrate the working mechanism of the algorithm, we skip the training classifier part and will use assumptions in order to make it simple. Now, the preprocessing part is completed and oversampling algorithm can work to find new instances.

Table 4.2: A sample for unlabeled instances from the sentiment dataset.

#	**Unlabeled Text**
1	This was one of the worst films i have ever seen.
2	But it wasn't until I watched this film that I realised how great he actually was.
3	However, this didn't make up for the fact that overall, this was a tremendously boring movie.
4	It is wonderful and inspiring to watch, and I hope that it gets released again on to video or DVD.
5	I must say I have taped most of the episodes and i find myself watching them over and over again.
6	This movie was kind of long in length, but I enjoyed every minute of it.

The oversampling algorithm starts with calculating class similarities and similarity factors using Equation 3.1. In the example, we have only two classes: positive and negative, and calculated class similarities are 0.42 and 0.49 respectively. According to Equation 3.1, the similarity factors are 1.54 for positive class and 1.43 for negative class. After calculating the class similarities and similarity factors, it is possible to calculate the similarity threshold for each class. The next step is training a baseline classifier and the obtained single performance measure for this classifier is 0.33 (Equation 2.12). The class measures are 0.0 and 0.5 for positive and negative classes respectively. Now, the algorithm will loop for a certain number of iterations and will try to find new instances from the unlabeled set. For this example, we go for several iterations, and please keep in mind that this is a parameter of the algorithm and will be optimized for better performance.

The first step in the loop is calculating the number of required instances using the formula 3.2. In this case, the number of total instances, m, is 6 and m_1 is 2 and m_2 is 4. The balance ratio is another parameter of the algorithm and it controls what will be the oversampling ratio. It will be fine-tuned for better performance but for the sake of simplicity, we take that as 0.5 here. From equation 3.2, the number of required instances for the positive class is 2 and the negative class is -0.5 which will be ignored. In formula 3.3, we update the number of required instances by using the

Table 4.3: Cleaned and standardized text.

Text	Label	Set
boring	negative	train
long time entertained movie	positive	train
good cinematography monica bellucci beautiful	positive	train
movie solid bad plot bad idea	negative	train
insult intelligence huge waste money	negative	train
acting terrible writing worse	negative	train
honest unbelievable nonsense foolish	negative	test
cult classic worth viewing sharing	positive	test
waste time money bad	negative	test

performance factor, p. The reason we put the performance into consideration is that a class maybe is imbalanced but still can exhibit good performance. In that case, when a class performs well while it is imbalanced, the required number of instances should be low or none. So, the updated required number of instances by using the formula 3.3 are 2 and 0 for positive and negative classes respectively. Now, the candidate instances from the unlabeled set will be found that can be added to the labeled set. According to the calculated required number of instances, the selection probabilities will be calculated. Since we have two classes and their number of required instances are 2 and 0, after normalizing them in the 0-1 range we obtain 1.0 and 0.0 probabilities. It means that the algorithm will only consider the positive class and search for new instances for only the positive class. It searches the unlabeled set iteratively and if an instance has a greater similarity than the similarity threshold calculated by multiplying the class similarities with similarity factors. It is calculated as 0.65 for the positive class and 0.70 for the negative class. Now, the similarity between an instance and the positive class is calculated and if that similarity exceeds the 0.65 threshold, it will be added to the candidate instances list. Batch size is another parameter here

and it restricts the length of the candidate list that will be found from the unlabeled set. The higher the batch size, the algorithm works more efficient while increasing the risk of losing promising instances in a group of bad instances. Here we take the batch size as 1 and the algorithm will look for instances one by one.

Table 4.4: The similarity scores of the unlabeled set with the positive class.

#	**Unlabeled Text**	**Similarity**
1	This was one of the worst films i have ever seen.	0.21
2	But it wasn't until I watched this film that I realised how great he actually was.	0.57
3	However, this didn't make up for the fact that overall, this was a tremendously boring movie.	0.33
4	It is wonderful and inspiring to watch, and I hope that it gets released again on to video or DVD.	0.76
5	I must say I have taped most of the episodes and i find myself watching them over and over again.	0.69
6	This movie was kind of long in length, but I enjoyed every minute of it.	0.63

The algorithm iterate over the shuffled list to find instances and let's say Table 4.4 represents the list that the algorithm iterates over. The first three instances have lower similarity than the threshold, so the algorithm skips them and finally, the fourth instance has a 0.76 similarity score with the positive class. It is now on the candidate list and it will be added to the training set temporarily. We need to intervene here and should remark that our toy example contains two classes to simplify the illustration of the proposed method.

The actual method is designed for multi-label classification cases and here, there is a need to demonstrate what happens if the case is multi-labeled. In the multi-label case, the algorithm will find the other labels for these potential instances. It finds the potential instances to extend the selected class for oversampling class but these instances may have more than one label due to the nature of the problem. The algorithm finds the other labels using the same similarity threshold logic and these labels

also have an effect on the classification performance. Now that we have made this small addition, we can return to the mechanism of the algorithm. After adding that instance to the training set, a classifier is trained to check for improvement. If the new single performance score is better than the older one, then this instance helped to improve the performance and will be added to the labeled set, otherwise not. In this case, the new performance measure is 0.33 which is the same as the older score and didn't help in improving the performance. So, this instance will not be added to the labeled set. The similarity factor is updated to adjust the similarity threshold. Since the performance is not improved, the similarity factor will be decreased with equation 3.5. The new calculated similarity factor is 1.41 and the new similarity threshold is 0.59. In the next iteration, if the instances exceed the 0.59 level, they will be considered potential instances. Since there is no change in the performance measures, we can use the same number of required instances and selection probabilities are calculated in the previous iteration. The next instance is the fifth instance and its similarity is 0.69 which is greater than the similarity threshold. It is another candidate instance and another classifier is trained to check for performance improvement. In this case, the performance reached 0.67. The class scores also reached 0.67 for both classes as well. It means adding this instance to the training data improved the classifier's performance and it will be added to the labeled set. So the labeled set is extended and the similarity factor will be increased. Using Equation 3.4, the new similarity factor is 0.61.

In the next iteration, the extended labeled set will be used and the new set has 3 positive and 4 negative instances. So, the required number of instances is changed and, thus, all the other parameters. Now, the number of required instances for the positive class is 1 and 0 for the negative class. Again, the probabilities are 1 and 0 for positive and negative classes because the positive class is the still minority class. Similarly, the selected class is the positive class and the algorithm will iterate over to find new instances for the positive class. The only candidate instance left in the unlabeled set has a similarity of 0.63 to the new set. The current similarity threshold is 0.61 and that instance is now in the potential instance list. A new classifier is trained to check improvement and the new single score is 1.0 which is better than the previous score

and so this instance is also added to the labeled set. Since this was the last iteration of our toy example, the algorithm is finalized by returning the extended labeled set and unlabeled set. A final classifier is trained to the present final results of the algorithm.

This is a toy example to demonstrate our algorithm on a small and limited dataset. The multi-label datasets contain several labels and thousands of instances. The similarity between instances is not easy to capture and the algorithm needs hundreds of iterations to find several instances.

4.2 Choosing the Proper Performance Measure

Before we go into our experimentation process, it is obvious that we need to define the performance measure used in our algorithm. There are several measures used in multi-label classification and imbalanced data classification. We can classify measures under two categories: classification measures and rank-based measures. Classification measures are used in any kind of classification problem and some of them resolve the data imbalance problem. Rank-based methods are designed for multi-label classification problems, on the other hand, they have weaknesses when the data imbalance problem arises. We mentioned the measures used in multi-label classification in section 2.

Some classification measures like accuracy and hamming loss only consider the proportion of the correctly classified examples overall. These measures are not sensitive to the class imbalance problem. Their main drawback is not considering the minority classes' performances. For example, a highly imbalanced dataset that has too many instances for the majority class and very less for the minority class will have high scores if all the predictions are made as the majority class. Accuracy is a good example of it. It counts the correctly classified instances and finds the ratio over the total number of instances. Similarly, the hamming loss is calculated as the difference between the true labels and predictions, with the same logic with accuracy.

The exact match ratio is the ratio of correctly classified label sets to all numbers of instances. The problem with the exact match ratio is it considers the label set as a whole and it is a very strict measure and not sensitive to little changes. For example, it considers a label set is wrong if only one label is wrongly predicted out of many labels. Also, if all the labels are wrongly predicted for a label set, it again considers that label set as wrong. It gives the same importance to both cases which is not the desired scenario, especially in case of data imbalance problem exists.

Ranking-based methods can be listed as one error, coverage, ranking loss, and average precision. They are calculated with the prediction probabilities instead of predicted classes. One error only considers the top-ranked label and does not care about the others. So, it is inadequate to measure the overall performance unless the focus is on the top-ranked label. Coverage measures the average step needed in the ordered predicted probabilities list to cover all the true labels. It's not easily interpretable and its working mechanism is a little bit different than the traditional measures. Lastly, average precision is also a measure that measures the average fraction of labels ranked than a specific label. During our literature survey, we realized that ranking-based measures are designed for specific tasks and most of the studies mentioned them instead of explaining and analyzing them. Also, they are not easily interpretable and not sensitive for the data imbalance case.

On the other hand, precision, recall, and their harmonic mean, f1-score are other classification measures. Precision is the ratio of correctly predicted outputs to the total number of predictions. It measures the predictive success of the classifier. The recall is the ratio of correctly predicted outputs to the total number of true outputs and is used to measure the detection success of the classifier. Precision and recall are designed for specific purposes and, otherwise, f1-score, another measure that maximizes precision and recall at the same time is used. F1-score is one of the most used measures in case of data imbalance (Wardhani et al., 2019). So, we choose f1-score as our single measure to optimize.

When the number of classes is greater than two, the weighting of the classes is a problem in classification measures. The decision of weighting the classes or weighting the instances is important, especially in case of data imbalance. Micro weighting considers all instances to have the same weight and macro averaging gives equal weights to all classes.

When we evaluate all the measures to choose the proper one or ones to use in our further analysis, most of the measures are to evaluate the results in general. Since we are working on solving the class imbalance problem, we need a measure that measures performance for each class individually as well. So, we can consider accuracy, precision, recall, or f1-score that comprises both (Equations 2.8, 2.10, 2.11, and 2.12). For the cases to evaluate the algorithm's performance on the whole dataset, we can choose the exact match ratio, accuracy, hamming loss, or f1-score. The exact match ratio is a very strict measure and it only accepts correct when the whole label set matches exactly. Accuracy does not consider class imbalance problems and so the hamming loss. Ranking-based measures focus on the probabilities of ranks rather than the classification success. In our case, we decided to continue further analysis with the macro average f1-score measure. M.-L. Zhang et al. (2020), Spyromitros-Xioufis et al. (2011), Tahir, Kittler, and Yan (2012) are the studies that use macro averaging f1-score to handle class imbalance problems in multi-label settings.

4.3 Experimentation Setup

In order to test our proposed algorithm, we conduct experiments and perform analyses to test our algorithm. These analyses are conducted on a dataset and the results are evaluated and interpreted. We present the OPP-115 dataset, the dataset we use in our analyses, and its characteristics. As indicated in the overall algorithm of the proposed method, the parameters should be optimized to improve the performance. The performance of the proposed method highly depends on the choice of these parameters. As an optimization strategy, an experimental design is used to measure the effectiveness of the parameters on the algorithm's performance. After finding the important parameters, we try to optimize them to obtain the best performance. Finally,

we interpret the results of the algorithm and shed light on future work. Before starting to deep dive into the experiments, we briefly explain the parameters of the algorithm to refresh the reader's memory. Lastly, as the main classifier, we use Linear Support Vector Machines (SVM) which is introduced in Section 2.

4.4 The OPP-115 Dataset

The dataset we use in our experimentations is the OPP-115 dataset. The OPP-115 dataset is a collection of website privacy policies to analyze the data practices (Wilson et al., 2016). Website privacy policies are plain texts formed by a group of paragraphs. Generally, each paragraph covers some topic related to regulations, like the collection of user data, sharing, processing, data security, etc. In order to degrade users' effort, rather than reading each paragraph and understanding what it covers, a machine learning algorithm can be employed to make a classification in order to assign each paragraph to a related topic. After assigning paragraphs to topics, it is easier to understand a privacy policy that covers which topics and has an incompleteness for which of them. Or building a tool that rates the privacy policies in terms of completeness to rescue the users from reading them is possible by using classification results.

Compared to other text classification problems, like sentiment analysis or, user comment classification, privacy policy classification has a more complex and rich vocabulary corpus because of its context. The challenge that humans face, appears for machine learning algorithms too. Inadequacy of the data for training, complexity, and copiousness of the vocabulary makes the problem harder. Also, in real life, the number of instances belonging to each class is distributed differently. For example, the collection or sharing of information is considered more important than other categories from both authorities and users compared to the retention of the data. While most of the privacy policies cover data collection and sharing, some of them do not indicate anything for data retention. So, the number of instances for each class differs, meaning a class imbalance problem arises in the dataset. Another aspect of the problem, each paragraph may cover more than one category or does not cover any

of them. Having more than one label at the same time converts the problem into a multi-label, multiclass classification problem.

When we analyze the labels of the dataset, 12 different labels are present in the dataset. Each instance takes at least 1 and at most 5 different labels. The distribution of the instances according to the number of labels they take is presented below in Figure 4.1.

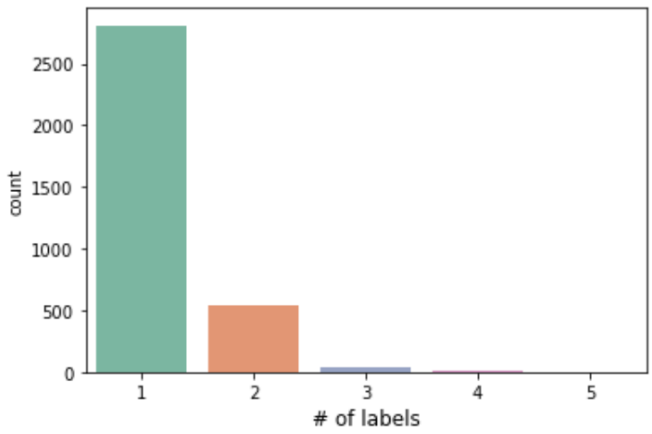

Figure 4.1: The distribution of length of the label set.

The number of instances belonging to each class is not uniformly distributed. While a class has 1,181 instances, others have very few numbers like 31 and 78. There is an imbalanced distribution between the instances and imbalanced distribution will make the classification task very hard. This point is the beginning point of our problem. We want to solve this class imbalance problem. The distribution of the classes is represented in Figure 4.2.

In Table 4.5, the summary statistics for the dataset are given. Cardinality is defined as the average number of labels per example and density is the average number of labels for each sample obtained by dividing by the total number of labels (Bernardini et al., 2014). Also, the average number of words per instance is given. In the last column, the maximum imbalance ratio is given to demonstrate how imbalanced the dataset is.

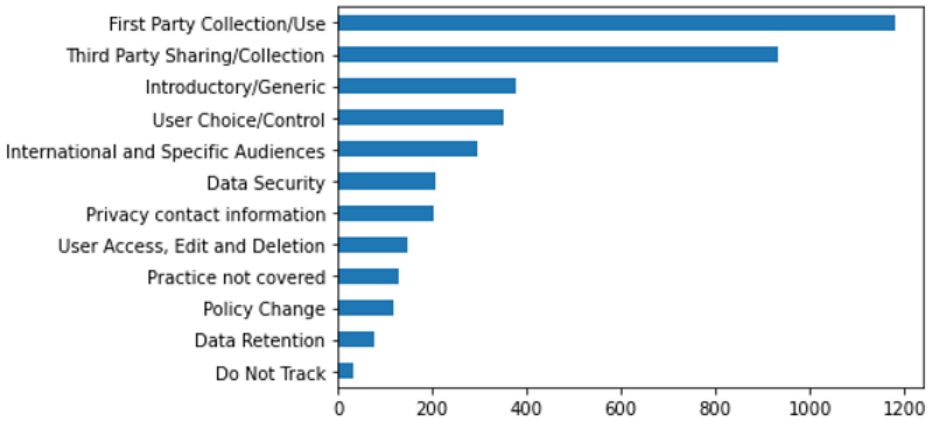

Figure 4.2: The distribution of the classes.

Table 4.5: Summary statistics for the dataset.

Dataset	Domain	# of labels	# of instances	Cardinality	Density	avg. # of words	Max Imb. Ratio
Opp-115	Legal text	12	3,399	1.19	0.099	82	38:1

4.5 DOE and Parameter Optimization

As we mention the importance of optimizing the parameters, in this section, an optimization strategy is presented to the reader's attention. First, an experiment is designed to find the effect of the parameters and, if possible, to find the optimal values for the parameters. The result obtained with the optimal parameters is interpreted to evaluate the algorithm's performance.

4.5.1 Design of Experiments for Parameters

Our oversampling algorithm has several parameters that we think have an effect on the performance of the algorithm. Because of the limitation of time and resources, we need to employ an efficient way to design our experiment. For categorical parameters, all the levels are included in the experiments but for continuous parameters, only two levels are included: high and low levels. After determining the levels of parameters, a general full factorial design is set up, and the results are analyzed with ANOVA. Ac-

cording to the ANOVA results, it is possible to detect important parameters, and we optimize them for further performance improvements. Before starting the analysis, in order to refresh the reader's memory, we explain the algorithm's parameters and which levels we choose for experimentation.

Balance Ratio

The first parameter is the **balance ratio**. The **balance ratio** is the parameter used during the calculation of the number of required instances. Using Equation 3.3, the number of required instances to balance the dataset is calculated and these numbers are used as the selection probability of the classes during oversampling. Our algorithm's main aim is to find as many as possible instances from the imbalanced set to balance the dataset, not to reach this **balance ratio**. It is because the unlabeled set is a limited set and the number of instances that belong to a class may not be sufficient to reach this ratio. The ideal level for the **balance ratio** is 0.5 which means all the classes have the same number of instances. In our experiments, we chose 0.5 and 0.2 as our levels and the analysis will be done by using these two levels.

Batch Size

The second parameter is the **batch size**. **Batch size** is the number of instances that will be added to the candidate list from the unlabeled set in each iteration. The instances are found from the unlabeled set by considering their similarity and their contribution to the performance of the overall classifier will be tested by training a classifier. If the **batch size** is low, then the algorithm will test instances more sensitively while the efficiency decreases. If a high **batch size** is employed, the algorithm will be efficient but the effect of potential instances may not be measured carefully. For our experiments, we want to measure the effect of **batch size** more precisely so we chose 3 levels. We chose 1, 3, and 5 as low, medium, and high levels.

Number of Iterations

The **number of iterations** is a parameter that controls the cycle count of the algorithm. The more iterates the algorithm, it will find more instances and the dataset

reaches a more balanced state. So, the **number of iterations** is another parameter that should be controlled to improve performance. The levels for this parameter are chosen as 100 and 500.

Similarity Calculation Type

The **similarity calculation type** is the similarity calculation method parameter. During the calculation of class similarities, the similarities between a group of pair vectors are calculated and an array of similarities is obtained. **Similarity calculation type** comes to play at this point and it controls how to calculate the single similarity score to represent the final similarity from this array. 'average' takes the average of this array and 'safe_interval' takes the 3rd quartile of this array. 'safe_interval' is a more strict compared to 'average' because 3rd quartile of a sample is higher than the mean under the assumption of normal distribution.

Similarity Function

The **similarity function** is the last parameter to include in our analysis. In the literature, there are several similarity functions used to measure similarity between instances. We mentioned the similarity functions in section 2 and we included the three of them in our analysis. Cosine similarity is nearly the mostly used **similarity function** among the studies because its accuracy and efficiency, given in equation 2.15 (Park et al., 2020). Euclidean distance is also a well-known function to calculate the distance between two vectors, given in equation 2.14. It is easy to interpret and can be calculated efficiently. Jensen Shannon distance is another measure that calculates the distance between two probability vectors as given in equation 2.16. It is the square root of the Jensen Shannon divergence which is a symmetrized and smoothed version of Kullback–Leibler divergence (Fuglede and Topsoe, 2004). The distances will be converted to the similarity with the Equation 2.17.

The summary of the values for the parameters are given in Table 4.6. Three parameters have two levels and two parameters have three levels. In total, 72 different combinations exist for the parameters.

Table 4.6: Summary for parameter settings.

Parameter	# of levels	Levels
balance ratio	2	[0.2, 0.5]
batch size	3	[1, 3, 5]
the number of iterations	2	[100, 500]
similarity calculation type	2	['average', 'safe_interval']
similarity function	3	['cosine', 'euclidean', 'jensen-shannon']

After explaining the parameters of the algorithm we can move on to the design setup. In total, we have 5 parameters and these parameters have different levels. Three of them have two levels and two of them have three levels which make our design general full factorial design. In our analysis, for each parameter setting, we will use k-fold cross-validation and take 10 runs for each split. K-fold cross-validation is a technique to validate machine learning models by splitting data into k folds (Jung, 2018). One of the folds is used for testing and the remaining splits are used for model training. So, k number of test results are obtained to evaluate model performance. The logic of k-fold cross-validation is presented in Figure 4.3. With the cross-validation strategy, the effect of data splits is removed from the results. For example, if only one split is used for validating the model, there is a risk of obtaining good results thanks to the randomness. In order to remove the possible effect of the data splits arising from randomness, the cross-validation strategy is applied to validate ML models on a dataset. So that the model that performs well all around the dataset is obtained, not on a single data split. In k-fold cross-validation, we chose k as 5 in our analysis. From parameters, we have 72 different parameter settings and for each parameter setting 5x10 runs are taken and 3,600 runs are conducted for analysis.

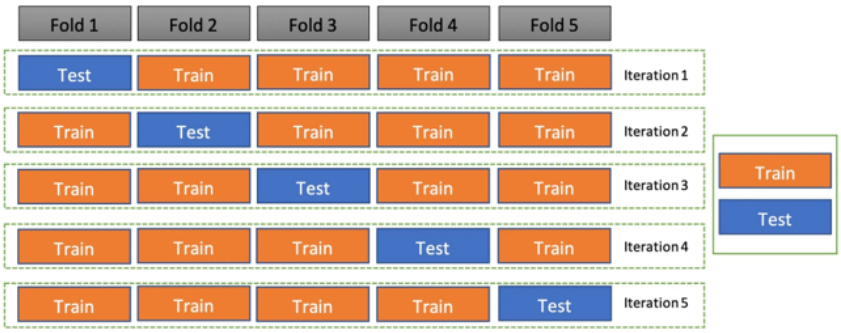

Figure 4.3: K-fold cross-validation for k=5.

4.5.2 Results

The response variable is defined as the **percentage improvement** in the chosen measure after oversampling. It is calculated with the initial measure obtained with the baseline classifier before oversampling and the final measure obtained with the final classifier after oversampling with Equation 4.1. In Equation, p_0 represents the initial performance measure and p_1 represents the final performance measure. It is needed to clarify that, both the initial and final performance measures should be included in the response variable. Our aim is to evaluate the contribution of the algorithm to the performance measure by oversampling the dataset. So, we need to include the initial scores to see the effect.

$$\%\ improvement = \frac{p_1 - p_0}{p_0} \tag{4.1}$$

With the given parameter settings in Table 4.6, the analyses are conducted using the Minitab software, with $alpha = 0.05$ significance level. A general full factorial design is set to analyze the effect of parameters. The parameters and their two-way and three-way interactions are included for analysis. We decided not to include four-way and above interactions because we think they are insignificant. After obtaining the initial model, which has the Pareto Chart presented in Figure 4.4, significant terms

are found. In order to reduce the model to improve adjusted R^2, the insignificant terms are removed from the model and a new model is created with only significant terms. Using the ANOVA results, presented in Appendix B, the terms that have a p-value greater than 0.05 are found and reduced from the model. Besides the significant terms, the insignificant terms that are a part of interactions are included to make the model hierarchical.

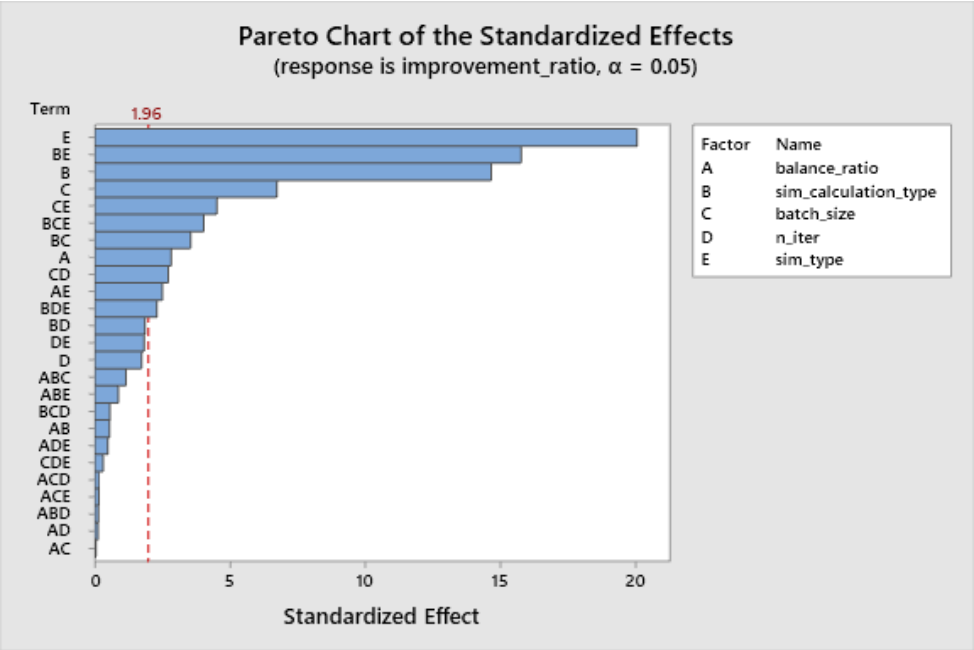

Figure 4.4: Pareto chart for the initial model.

In Figure 4.5, the Pareto chart for the standardized effects is presented for the reduced model. Also, the detailed ANOVA results are presented in Appendix C. The regression equation for the reduced model is given in Equation 4.2. The adjusted R^2 of the model is 0.31. This is not a high score as we expected and we think that the result of the model is questionable. Because of the low R^2 score, the model is not adequate and the inferences of the model are questionable.

From the obtained results, for the main effects, the p-values for all parameters except $numberofiterations$, which has a p-value of 0.083, are less than 0.05 and we can say that all of them are significant except $numberofiterations$. The significant pa-

rameters affect the performance of the algorithm and will be tuned for performance improvements in the next section.

$$
\begin{aligned}
\textbf{improvement\_ratio} = {} & 0.004642 + 0.002285A_{-1} - 0.002285A_1 - 0.011846B_{-1} \\
& + 0.011846B_1 - 0.01142C_{-1} + 0.00720C_0 + 0.00423C_1 + 0.001397D_{-1} \\
& - 0.001397D_1 - 0.016205E_{-1} + 0.016205E_1 + 0.002025A_{-1}E_{-1} \\
& - 0.002025A_{-1}E_1 - 0.002025A_1E_{-1} + 0.002025A_1E_1 + 0.00376B_{-1}C_{-1} \\
& + 0.00031B_{-1}C_0 - 0.00406B_{-1}C_1 - 0.00376B_1C_{-1} - 0.00031B_1C_0 \\
& + 0.00406B_1C_1 + 0.001502B_{-1}D_{-1} - 0.001502B_{-1}D_1 - 0.001502B_1D_{-1} \\
& + 0.001502B_1D_1 - 0.012747B_{-1}E_{-1} + 0.012747B_{-1}E_1 + 0.012747B_1E_{-1} \\
& - 0.012747B_1E_1 - 0.00321C_{-1}D_{-1} + 0.00321C_{-1}D_1 + 0.00017C_0D_{-1} \\
& - 0.00017C_0D_1 + 0.00305C_1D_{-1} - 0.00305C_1D_1 + 0.00539C_{-1}E_{-1} \\
& - 0.00539C_{-1}E_1 - 0.00143C_0E_{-1} + 0.00143C_0E_1 - 0.00397C_1E_{-1} \\
& + 0.00397C_1E_1 + 0.001474D_{-1}E_{-1} - 0.001474D_{-1}E_1 - 0.001474D_1E_{-1} \\
& + 0.001474D_1E_1 + 0.00479B_{-1}C_{-1}E_{-1} - 0.00479B_{-1}C_{-1}E_1 \\
& - 0.00383B_{-1}C_0E_{-1} + 0.00383B_{-1}C_0E_1 - 0.00097B_{-1}C_1E_{-1} \\
& + 0.00097B_{-1}C_1E_1 - 0.00479B_1C_{-1}E_{-1} + 0.00479B_1C_{-1}E_1 \\
& + 0.00383B_1C_0E_{-1} - 0.00383B_1C_0E_1 + 0.00097B_1C_1E_{-1} \\
& - 0.00097B_1C_1E_1 + 0.001857B_{-1}D_{-1}E_{-1} + 0.001857B_{-1}D_{-1}E_1 \\
& + 0.001857B_{-1}D_1E_{-1} + 0.001857B_{-1}D_1E_1 - 0.001857B_1D_{-1}E_{-1} \\
& + 0.001857B_1D_{-1}E_1 + 0.001857B_1D_1E_{-1} - 0.001857B_1D_1E_1
\end{aligned}
\tag{4.2}
$$

The coefficients of the regression model show the effect of the parameters on the response variable. For continuous variables, the coefficients indicate the amount of change in the response variable. In order to find the effect of a variable on the response variable, a partial derivative of the regression equation should be taken with respect to the related variable. $balance\ ratio$, is a significant term and also, it is a component of a two-way interaction, which is $balance\ ratio$*$similarity\ type$. So, a change in $balance\ ratio$ will affect the response variable by its coefficient and de-

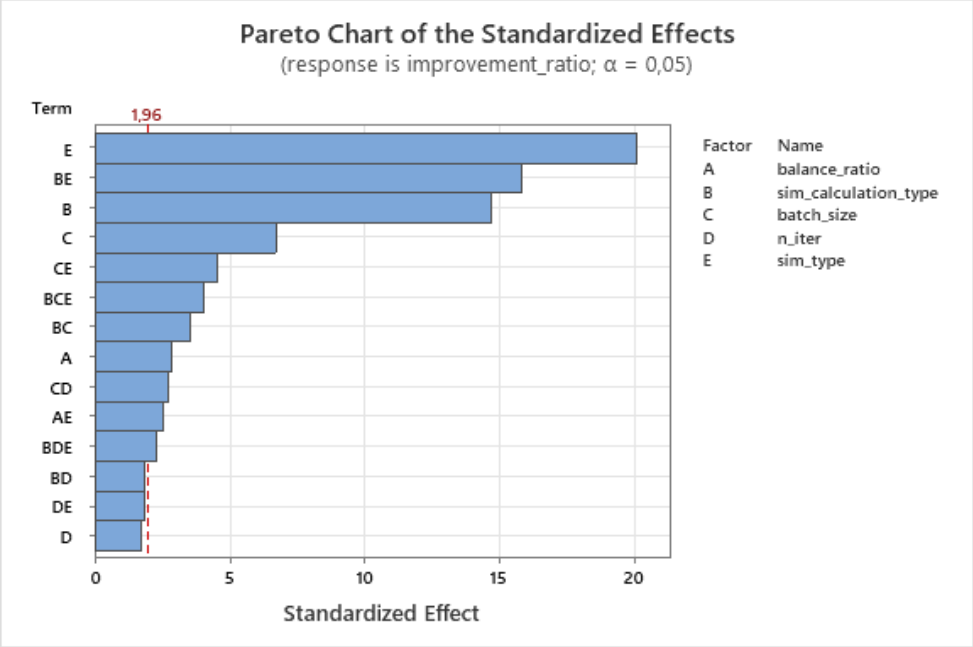

Figure 4.5: Pareto chart for the reduced model.

pending on the value of *similarity type*. The other continuous variable is *number of iterations*. It is an insignificant term in the model but its a part of a significant two-way interaction. It affects the response variable with *batch size* together.

Also, F-Value is another measure to interpret the effect of the parameters. It is the ratio of variation between sample means to variation within the samples. The higher the F-value, the higher the deviation between samples means proximate to the deviation within the samples. Also, the higher F-value corresponds to the lower p-value. From the ANOVA table presented in Appendix B, the *similarity type* has the highest F-value which is 405. Secondly, *similarity calculation type* has the second highest F-value score, with a score of 216. The third score is *batch size* with a score of 51. *balance ratio* has a very low F-value, which is 8, and *number of iterations* has 3. Also, it can be seen that *number of iterations* is insignificant in the Pareto chart given in Figure 4.5.

When we look at the 2-way and 3-way interactions, there are some interactions that are significant. *Balance ratio* and *similarity type*, *similarity calculation type*

and $batch\ size$, $similarity\ calculation\ type$ and $similarity\ type$, $batch\ size$ and $number\ of\ iterations$, $batch\ size$ and $similarity\ type$ pairs are the significant 2-way interactions according to their p-values. Rest of them looks insignificant according to Figure 4.5 and Appendix B. In 3-way interactions, $similarity\ calculation\ type$, $similarity\ type$, $batch\ size$, and $similarity\ calculation\ type$, $similarity\ type$, $number\ of\ iterations$ trios are the only significant 3-way interactions. When we carefully look at the 2-way and 3-way interactions, we can see that the main effects that have the highest F-values are generally part of the interactions. $similarity\ calculation\ type$, $similarity\ type$ have the highest F-values according to Appendix B and have the highest standardized effect in Figure 4.5 and they are generally part of the significant 2-way and 3-way interactions. It can be said that $similarity\ type$ plays an important role in the algorithm's performance as we indicated before. Choosing the right similarity function has vital importance in the algorithm's performance. Also, $similarity\ calculation\ type$ has a great effect on the performance. It is a part of the decision mechanism in choosing the most similar instances from the unlabeled set, so it is expected to be significant.

In Equation 4.2, the coefficients of each interaction level can be seen. From this equation, we can find the importance of each parameter's levels. $similarity\ type$ has the highest coefficient in this equation. When we look at the coefficients, the coefficient for the level equivalents to $euclidean$ is the highest among the interactions. The coefficient for the $cosine$ is high but it is negative. For JS, it is also negative and very low. From here, we can say that when the $similarity\ type$ is $euclidean$, it has the highest effect on the algorithm's performance. The second highest effect belongs to $similarity\ calculation\ type$ in the main effects. $average$ has a negative effect, because the coefficient is negative, and conversely, for $safeinterval$ has a positive effect. Since there are only two levels for $similarity\ calculation\ type$, we can say that $safe\ interval$ has a high and positive effect on the algorithm.

For the $batch\ size$, the highest positive effect is derived when the $batch\ size$ is 3 or 5. For 1, it is negative, so we can say that the $batch\ size$ should be around 3 or 5. The last main effect that is significant in the algorithm's performance is $balance\ ratio$. It has

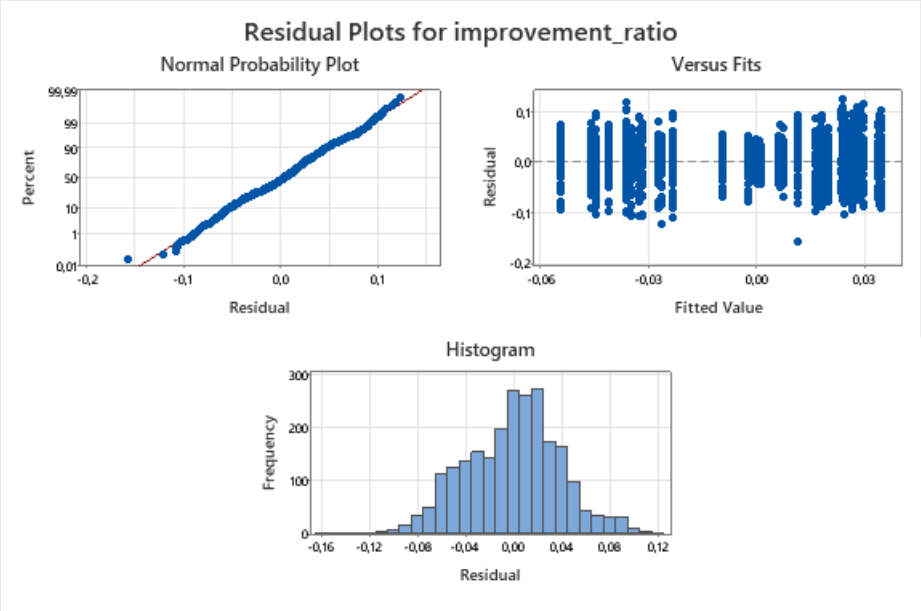

Figure 4.6: Residual plots for the response variable.

two levels and while the low level has a positive effect, the high level has a negative effect. Other than the main effects, *similarity type*, *similarity calculation type* pair has a high coefficient.

In the ANOVA results, we need to analyze carefully the residual and normality plots to validate the normality assumption for the residuals. According to the figure, we can assume that the residuals are distributed normally. In the normal probability plot, left upper plot in Figure 4.6, the residuals lie on the red line very closely, with a small variation. The red line is the line of the perfectly-homoscedastic model, and the actual residuals of the model, The histogram plot, located on the left lower side of the figure, shows the distribution of the residuals. This histogram is similar to the normal distribution plot, meaning we can assume that the residuals are normally distributed. Also, the upper right plot in Figure 4.6 indicates that the residuals are located on both sides of 0 with similar distances. In other words, the error variance can be considered constant. All the plots in the figure show that the residuals can be assumed as distributed normally and the model fits the assumption of homoscedasticity. So, we can assume that the analysis obtained from the ANOVA is valid.

We suspect some narrowing in the residual plot which may mean the variance is not constant. This narrowing is a little bit and may not be too critical. If the variance is not constant there are some methods to solve this problem. We applied box-cox transformation and logit transformation are applied to check whether the results are changed or not. After trying both box-cox transformation and logit transformation, the results are not changed and for all practical purposes, we assume that the assumptions behind the ANOVA model are valid.

Box-cox transformation is a technique that helps to reduce non-normality and heteroscedasticity in a sample (Sakia, 1992). If the data does not follow the normal distribution or does not fit the homoscedasticity assumption, box-cox transformation is used to transform the data to a normal distribution. In our analysis, we also tried to apply the box-cox transformation to our response variable but the results does not change. So, we decided to move on without transformation.

4.5.3 Parameter Optimization

In the previous section, the effect of the parameters is analyzed and the important parameters that positively affect the algorithm's performance are found. After the detailed analysis of coefficients, in order to obtain optimal values for the parameters, a Response Surface Design is fit in Minitab. Response Surface Design is used when the goal is optimizing the response variable (DONNELLY, 1984). The results of Response Surface Design are very similar to the results obtained with coefficient analysis. The $batch\ size$ is 5, $similarity\ type$ is $euclidean$, $similarity\ calculation\ type$ is $safe\ interval$, $balance\ ratio$ is 0.2 and lastly, the $number\ of\ iterations$ is 100. We think other interactions are not worth mentioning here and we will continue to our optimization step with the findings obtained here.

In this section, in order to improve the algorithm's performance, different values of the algorithm's parameters are tried and the effect of these parameters are investigated. The algorithm has five parameters in total. Two of them are categorical

and can take a limited number of values. In the earlier analysis, all possible values for these categorical variables are tried. In the previous section, we find that for $similarity\ type$, $euclidean$ has the highest effect. $safe\ interval$ has the highest coefficient for $similarity\ calculation\ type$. The best level for $batch\ size$ is around 5. Lastly, the best level for $balance\ ratio$ is 0.2. Since the $number\ of\ iterations$ is not significant, we do not focus on it and take the low level 100 which has a positive effect according to the equation presented in 4.2.

It is obvious that finding the optimal parameters of the algorithm is not easy. It requires many runs with different parameter combinations. Our strategy is to set up a factorial design to find significant parameters. By analyzing the coefficients of the fitted model and the results obtained from the response surface model, we try to find optimal parameters in the given parameter combinations. From the results, the promising values or ranges for the parameters are interpreted. For the sake of simplicity, we try several more alternatives for these parameters in the direction of the findings.

For categorical variables, we find the levels that have a positive effect on the results. For $balance\ ratio$, the low level has a positive effect so, we try 0.2 and additionally 0.3 as the new values for the new alternatives. For the $batch\ size$, 5 and 3 have positive effects so, other than these two values, we also try 7 as an alternative. Lastly, the $number\ of\ iterations$ is insignificant and we continue with the initial value. As a summary, the final values for the parameters are listed in Table 4.7.

Table 4.7: Parameter settings for parameter optimization.

Parameter	Values
balance ratio	[0.2, 0.3]
similarity calculation type	'safe_interval'
batch size	[3, 5, 7]
number of iterations	100
similarity type	'euclidean'

For each setting, we take 10 runs with cross-validation. The best results are obtained with $balance\ ratio$ equals 0.3, $batch\ size$ equals 5 and number of iterations is 100. The average f1-score before oversampling is 0.5961 and after oversampling, it becomes 0.628. The average improvement in the f1-score is 5.34% and the number of added instances is 90. The initial sample size was 135 and after oversampling, it becomes 225. We can say that our algorithm found new instances and improved the performance of the classification with a 5.34% improvement in the f1-score.

4.5.4 Computational Time

Our proposed method requires taking a lot of runs to find optimal parameter settings. So, the computational time is another aspect of our method. We believe it is needed to present an efficient method that gives the correct results in the required time. A method may present feasible or optimal results but may not present the solutions in a feasible time. For this reason, efficiency is important not only for the performance but also for the run time.

While taking the runs to optimize parameters, we also measure the run times to evaluate the performance of the method. We take the runs on a local machine that has AMD Ryzen 9 3950X CPU and 64 GB memory. The average run time is 2.34 minutes with a standard deviation of 0.71 minutes for 100 iterations. We can say that the

algorithm finds new instances from the unlabeled set in a very short period of time. Compared to manual data annotation, a method that finds new instances in minutes can be considered efficient.

Algorithm parameters do not have an effect on the run time, except the number of iterations, as anticipated. We try two levels for the number of iterations in the parameter optimization step: 100 for the low level and 500 for the high level. The above run time is obtained when the number of iterations is 100. For the high level, run time becomes 12.21 minutes with a standard deviation of 1.26. It nearly increases linearly because when the number of iterations increases 5 times, the run time increases 5.21 times. We can say that the relation is nearly linear.

CHAPTER 5

CONCLUSION

In this study, an oversampling algorithm for multi-label imbalanced text classification is proposed. The algorithm's main aim is to find instances from an unlabeled set to extend the labeled set by utilizing the similarity of the instances. The algorithm takes the initial imbalanced set and after calculating the parameters of the algorithm, it finds new instances from the given unlabeled set by searching the unlabeled data space using the similarity parameters. It has an iterative mechanism to find instances from the unlabeled set and after checking performance improvements, it decides on to add the instances to the labeled set or not. Macro averaged f1-score is used as a performance metric and the parameters of the algorithm are optimized for better results. The results are analyzed to see whether the proposed algorithm helped to improve the performance of the classifier.

The assumption behind our algorithm is the instances that belong to the same class are located close to each other in the data space. The important point is here to capture the similarity between instances accurately. When the text data is converted to numerical format, it is formed as high-dimensional vectors. So, we use the similarity functions that are used for high-dimensional vector space such as euclidean distance, cosine similarity, and Jensen-Shannon distance. In our experiments, euclidean distance performed best compared to others. As a future study direction, trying different similarity functions, or defining a new similarity function that captures the similarity between instances more accurately will improve the performance of the algorithm. There are a lot of different similarity functions in the literature and an extensive survey can be done to find different similarity functions that help to improve performance.

The proposed method has several parameters that highly affect the performance. We test the method on one dataset and try to optimize the parameters to improve performance. In future studies, the proposed method can be tested on more datasets to make inferences about the parameters. There can be a relation between the parameters and characteristics of the datasets. By analyzing this relation, it is possible to make suggestions to users in choosing the right parameters for better performance.

In self-learning or semi-supervised learning methods, the new instances are labeled by the predictions of the initial classifier that was trained with the highly-imbalanced and insufficient dataset. When the dataset is highly imbalanced and insufficient, the results for the initial classifier are very poor and the predictions of the classifier for the unseen data instances are not reliable. In our method, we consider the initial classifier's predictions very risky and avoid using it as the labeler. Instead, the new instances are found using a similarity-based method and a classifier is trained to check whether the overall performance is improved or not. To the best of our knowledge, there does not exist any study in the literature that employs such a strategy to find new instances to oversample an imbalanced text dataset.

Another aspect of our method is applying it to a multi-label text dataset which makes the problem more complex compared to a single labeled case. In single-label classification, the output is a single class which means the instance can belong to only one class. From the similarity point of view, an instance can belong to only one class that is most similar to it. On the other hand, in a multi-label case, an instance can be close to different classes so it can belong to more than one class. A threshold or an acceptance mechanism should be applied to decide whether an instance belongs to a class or not. This aspect makes our problem more complex compared to single-class cases.

As for further future directions, active learning can be integrated with our method. For the instances that the algorithm is not highly confident to assign a class, they can

be asked by a decision maker to label them. We mentioned the difficulties in finding experts in the field to label a dataset, so the proposed method can be used as an assistant to help the labelers to label a dataset. In that way, the labels will be more confident and by only asking the potential instances from an unlabeled set instead of all the instances, the effort on the labeler will be eased.

We think that the results look promising and this can be a baseline study for future studies. New methods can be built based on our method. In our test cases, the performance of the classifiers is improved and we believe it is open for further improvements. Because our method finds real instances from the unlabeled set instead of creating synthetic instances. Finding real instances with the correct labels enriches the dataset and will help to improve the performance of the algorithm like they are labeled by a human annotator.

Lastly, another contribution of our study to the literature is that we investigated widely-used performance metrics for multi-label classification and criticized them. They can be divided into two categories: classification metrics and ranking-based metrics. Classification metrics are widely-used in any kind of classification problem and some of them are also good for imbalanced classification problems. However, ranking-based metrics are designed for multi-label classification problems but we realized that there is a gap in the literature that explains and clarifies these metrics properly. We haven't encountered sufficient studies that use ranking-based metrics because they are not easy to interpret and also they are not sensitive to data imbalance problems which is a case for most multi-label classification problems.

REFERENCES

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, *2*(4), 433–459.

Aggarwal, C. C., & Zhai, C. (1970). A survey of text clustering algorithms. https://link.springer.com/chapter/10.1007/978-1-4614-3223-4_4

Amin, A., Anwar, S., Adnan, A., Nawaz, M., Howard, N., Qadir, J., Hawalah, A., & Hussain, A. (2016). Comparing oversampling techniques to handle the class imbalance problem: A customer churn prediction case study. *IEEE Access*, *4*, 7940–7957.

Anandarajan, M., Hill, C., & Nolan, T. (1970). Text preprocessing. https://link.springer.com/chapter/10.1007/978-3-319-95663-3_4

Andoni, A., Indyk, P., & Krauthgamer, R. (2008). Earth mover distance over high-dimensional spaces. *SODA*, *8*, 343–352.

Balakrishnan, V., & Lloyd-Yemoh, E. (2014). Stemming and lemmatization: A comparison of retrieval performances.

Bedre, R. (2021). *Support vector machine (svm) basics and implementation in python*. Retrieved August 11, 2022, from https://www.reneshbedre.com/blog/support-vector-machine.html

Bernardini, F. C., da Silva, R. B., Rodovalho, R. M., & Meza, E. B. M. (2014). Cardinality and density measures and their influence to multi-label learning methods. *Submitted to Learning and Nonlinear Models*.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, 144–152.

Cao, P., Liu, X., Zhao, D., & Zaiane, O. (2016). Cost sensitive ranking support vector machine for multi-label data learning. *International Conference on Hybrid Intelligent Systems*, 244–255.

Charte, F., Rivera, A., Jesus, M. J. d., & Herrera, F. (2013). A first approach to deal with imbalance in multi-label datasets. *International conference on hybrid artificial intelligence systems*, 150–160.

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2015). Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, *163*, 3–16.

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2019). Dealing with difficult minority labels in imbalanced mutilabel data sets. *Neurocomputing*, *326*, 39–53.

Charte, F., Rivera, A. J., Jesus, M. J. d., & Herrera, F. (2014). Mlenn: A first approach to heuristic multilabel undersampling. *International Conference on Intelligent Data Engineering and Automated Learning*, 1–9.

Chau, M., & Chen, H. (2008). A machine learning approach to web page filtering using content and structure analysis. *Decision Support Systems*, *44*(2), 482–494.

Chauhan, V. K., Dahiya, K., & Sharma, A. (2019). Problem formulations and solvers in linear svm: A review. *Artificial Intelligence Review*, *52*(2), 803–855.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, T., Xu, R., Lu, Q., Liu, B., Xu, J., Yao, L., & He, Z. (2014). A sentence vector based over-sampling method for imbalanced emotion classification. *International Conference on Intelligent Text Processing and Computational Linguistics*, 62–72.

Clark, S. (2015). Vector space models of lexical meaning. *The Handbook of Contemporary semantic theory*, 493–522.

Consultant, T. P. S. (2022). *Apply a simple bag-of-words approach*. Retrieved August 11, 2022, from https://openclassrooms.com/en/courses/6532301-introduction-to-natural-language-processing/6980811-apply-a-simple-bag-of-words-approach

Cooper, K. (2022). *Openai gpt-3: Everything you need to know*. Retrieved August 11, 2022, from https://www.springboard.com/blog/data-science/machine-learning-gpt-3-open-ai/

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Crammer, K., & Singer, Y. (2002). On the learnability and design of output codes for multiclass problems. *Machine learning*, *47*(2), 201–233.

Črepinšek, M., Liu, S.-H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, *45*(3), 1–33.

Daniels, Z. A., & Metaxas, D. N. (2017). Addressing imbalance in multi-label classification using structured hellinger forests. *Thirty-First AAAI Conference on Artificial Intelligence*.

Desmond, M., Duesterwald, E., Brimijoin, K., Brachman, M., & Pan, Q. (2021). Semi-automated data labeling. *NeurIPS 2020 Competition and Demonstration Track*, 156–169.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Deza, M. M., & Deza, E. (2009). Encyclopedia of distances. *Encyclopedia of distances* (pp. 1–583). Springer.

DOMO. (2018). *Data never sleeps*. Retrieved August 11, 2022, from https://www.domo.com/assets/downloads/18_domo_data-never-sleeps-6+verticals.pdf

DONNELLY, P. (1984). Design and analysis of experiments, -montgomery, dc.

Donyavi, Z., & Asadi, S. (2020). Using decomposition-based multi-objective evolutionary algorithm as synthetic example optimization for self-labeling. *Swarm and Evolutionary Computation*, *58*, 100736.

Dumais, S. et al. (1998). Using svms for text categorization. *IEEE Intelligent Systems*, *13*(4), 21–23.

Er, M. J., Venkatesan, R., & Wang, N. (2016). An online universal classifier for binary, multi-class and multi-label classification. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 003701–003706.

Fautsch, C., & Savoy, J. (2010). Adapting the tf idf vector-space model to domain specific information retrieval. *Proceedings of the 2010 ACM Symposium on Applied Computing*, 1708–1712.

Femmer, H., Kučera, J., & Vetrò, A. (2014). On the impact of passive voice requirements on domain modelling. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–4.

Feng, Y., Zhou, M., & Tong, X. (2021). Imbalanced classification: A paradigm-based review. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *14*(5), 383–406.

Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, *61*, 863–905.

Floridi, L., & Chiriatti, M. (2020). Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, *30*(4), 681–694.

Forman, G. et al. (2003). An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, *3*(Mar), 1289–1305.

Fortune. (2021). *Natural language processing (nlp) market size growth*. Retrieved August 11, 2022, from https://www.fortunebusinessinsights.com/industry-reports/natural-language-processing-nlp-market-101933

Fredriksson, T., Mattos, D. I., Bosch, J., & Olsson, H. H. (2020). Data labeling: An empirical investigation into industrial challenges and mitigation strategies. *International Conference on Product-Focused Software Process Improvement*, 202–216.

Fuglede, B., & Topsoe, F. (2004). Jensen-shannon divergence and hilbert space embedding. *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, 31.

Ganda, D., & Buch, R. (2018). A survey on multi label classification. *Recent Trends in Programming Languages*, *5*(1), 19–23.

Gao, W., & Zhou, Z.-H. (2011). On the consistency of multi-label learning. *Proceedings of the 24th annual conference on learning theory*, 341–358.

Gibaja, E., & Ventura, S. (2015). A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)*, *47*(3), 1–38.

GMI. (2022). *Machine translation market size: Industry analysis, 2022-2030*. Retrieved August 11, 2022, from https://www.gminsights.com/industry-analysis/machine-translation-market-size

Gomaa, W. H., Fahmy, A. A. et al. (2013). A survey of text similarity approaches. *international journal of Computer Applications*, *68*(13), 13–18.

Goudjil, M., Koudil, M., Bedda, M., & Ghoggali, N. (2018). A novel active learning method using svm for text classification. *International Journal of Automation and Computing*, *15*(3), 290–298.

Gupta, M., Varma, V., Damani, S., & Narahari, K. N. (2020). Compression of deep learning models for nlp. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 3507–3508.

Guzmán-Cabrera, R., Montes-y-Gómez, M., Rosso, P., & Villasenor-Pineda, L. (2009). Using the web as corpus for self-training text categorization. *Information Retrieval*, *12*(3), 400–415.

Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., & Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert systems with applications*, *73*, 220–239.

Hajmohammadi, M. S., Ibrahim, R., Selamat, A., & Fujita, H. (2015). Combination of active learning and self-training for cross-lingual sentiment classification with density analysis of unlabelled samples. *Information sciences*, *317*, 67–77.

Herrera, F., Charte, F., Rivera, A. J., & Jesus, M. J. d. (2016). Multilabel classification. *Multilabel classification* (pp. 17–31). Springer.

Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd international conference on document analysis and recognition*, *1*, 278–282.

Ilić, S., Marrese-Taylor, E., Balazs, J. A., & Matsuo, Y. (2018). Deep contextualized word representations for detecting sarcasm and irony. *arXiv preprint arXiv:1809.09795*.

Jang, J., Kim, Y., Choi, K., & Suh, S. (2021). Sequential targeting: A continual learning approach for data imbalance in text classification. *Expert Systems with Applications*, *179*, 115067.

Jia, S., Lansdall-Welfare, T., Sudhahar, S., Carter, C., & Cristianini, N. (2016). Women are seen more than heard in online newspapers. *PloS one*, *11*(2), e0148434.

Jiang, Z., Pan, T., Zhang, C., & Yang, J. (2021). A new oversampling method based on the classification contribution degree. *Symmetry*, *13*(2), 194.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *European conference on machine learning*, 137–142.

Joshi, S., & Abdelfattah, E. (2021). Multi-class text classification using machine learning models for online drug reviews. *2021 IEEE World AI IoT Congress (AIIoT)*, 0262–0267.

Jung, Y. (2018). Multiple predicting k-fold cross-validation for model selection. *Journal of Nonparametric Statistics*, *30*(1), 197–215.

Kalyanathaya, K. P., Akila, D., & Rajesh, P. (2019). Advances in natural language processing–a survey of current research trends, development tools and industry applications. *International Journal of Recent Technology and Engineering*, *7*(5C), 199–202.

Kandhro, I. A., Jumani, S. Z., Lashari, A. A., Nangraj, S. S., Lakhan, Q. A., Baig, M. T., & Guriro, S. (2019). Classification of sindhi headline news documents based on tf-idf text analysis scheme. *Indian Journal of Science and Technology*, *12*(33), 1–10.

Kang, Y., Cai, Z., Tan, C.-W., Huang, Q., & Liu, H. (2020). Natural language processing (nlp) in management research: A literature review. *Journal of Management Analytics*, *7*(2), 139–172.

Karisani, P., & Karisani, N. (2021). Semi-supervised text classification via self pretraining. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 40–48.

Kaur, H., Pannu, H. S., & Malhi, A. K. (2019). A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Computing Surveys (CSUR)*, *52*(4), 1–36.

KDnuggets. (2018). *Multi-class text classification with scikit-learn*. Retrieved August 11, 2022, from https://www.kdnuggets.com/2018/08/multi-class-text-classification-scikit-learn.html

Kim, S.-B., Rim, H.-C., Yook, D., & Lim, H.-S. (2002). Effective methods for improving naive bayes text classifiers. *Pacific rim international conference on artificial intelligence*, 414–423.

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information*, *10*(4), 150.

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, *29*(1), 1–27.

Lanquillon, C. (2000). Partially supervised text classification: Combining labeled and unlabeled documents using an em-like scheme. *European Conference on Machine Learning*, 229–237.

Leah. (2021). *What do your customers actually think about chatbots?* Retrieved August 11, 2022, from https://www.userlike.com/en/blog/consumer-chatbot-perceptions

Lewis, D. D., & Ringuette, M. (1994). A comparison of two learning algorithms for text categorization. *Third annual symposium on document analysis and information retrieval*, *33*, 81–93.

Li, H., Caragea, D., & Caragea, C. (2021). Combining self-training with deep learning for disaster tweet classification. *The 18th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2021)*.

Li, J., & Zhu, Q. (2020). A boosting self-training framework based on instance generation with natural neighbors for k nearest neighbor. *Applied Intelligence*, *50*(11), 3535–3553.

Li, Y., & Yang, T. (1970). Word embedding for understanding natural language: A survey. https://link.springer.com/chapter/10.1007/978-3-319-53817-4_4

Li, Y., Guo, H., Zhang, Q., Gu, M., & Yang, J. (2018). Imbalanced text sentiment classification using universal and domain-specific knowledge. *Knowledge-Based Systems*, *160*, 1–15.

Li, Z. (2019). A classification retrieval approach for english legal texts. *2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, 220–223.

Lilleberg, J., Zhu, Y., & Zhang, Y. (2015). Support vector machines and word2vec for text classification with semantic features. *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, 136–140.

Liu, S. M., & Chen, J.-H. (2015). A multi-label classification based approach for sentiment classification. *Expert Systems with Applications*, *42*(3), 1083–1093.

Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. *arXiv preprint cs/0205028*.

Luo, M., Shi, X., Ji, Q., Shang, M., He, X., & Tao, W. (2019). A deep self-learning classification framework for incomplete medical patents with multi-label. *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, 566–573.

Luo, Y., Feng, H., Weng, X., Huang, K., & Zheng, H. (2019). A novel oversampling method based on seqgan for imbalanced text classification. *2019 IEEE International Conference on Big Data (Big Data)*, 2891–2894.

Ly, A., Uthayasooriyar, B., & Wang, T. (2020). A survey on natural language processing (nlp) and applications in insurance. *arXiv preprint arXiv:2010.00462*.

Magara, M. B., Ojo, S. O., & Zuva, T. (2018). A comparative analysis of text similarity measures and algorithms in research paper recommender systems. *2018 conference on information communications technology and society (ICTAS)*, 1–5.

Martınez, E., Mollica, F., & Gibson, E. (2022). Poor writing, not specialized concepts, drives processing difficulty in legal language. *Cognition*, *224*, 105070.

Meng, Y., Shen, J., Zhang, C., & Han, J. (2018). Weakly-supervised neural text classification. *proceedings of the 27th ACM International Conference on information and knowledge management*, 983–992.

Meng, Y., Zhang, Y., Huang, J., Xiong, C., Ji, H., Zhang, C., & Han, J. (2020). Text classification using label names only: A language model self-training approach. *arXiv preprint arXiv:2010.07245*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mohammad, S. (2020). Nlp scholar: A dataset for examining the state of nlp research. *Proceedings of the 12th Language Resources and Evaluation Conference*, 868–877.

Mohasseb, A., Bader-El-Den, M., Cocea, M., & Liu, H. (2018). Improving imbalanced question classification using structured smote based approach. *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, *2*, 593–597.

Mollineda, R., Alejo, R., & Sotoca, J. (2007). The class imbalance problem in pattern classification and learning. *II Congreso Espanol de Informática (CEDI 2007). ISBN*, 978–84.

Moreo, A., Esuli, A., & Sebastiani, F. (2016). Distributional random oversampling for imbalanced text classification. *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 805–808.

Morris, J. X., Lifland, E., Yoo, J. Y., Grigsby, J., Jin, D., & Qi, Y. (2020). Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *arXiv preprint arXiv:2005.05909*.

Mukherjee, S., & Awadallah, A. (2020). Uncertainty-aware self-training for few-shot text classification. *Advances in Neural Information Processing Systems*, *33*, 21199–21212.

Nareshpalsingh, J. M., & Modi, H. N. (2017). Multi-label classification methods: A comparative study. *International Research Journal of Engineering and Technology (IRJET)*, *4*(12), 263–270.

Nasiri, M., Minaei, B., & Sharifi, Z. (2017). Adjusting data sparsity problem using linear algebra and machine learning algorithm. *Applied Soft Computing*, *61*, 1153–1159.

Nazmi, S., Yan, X., Homaifar, A., & Doucette, E. (2020). Evolving multi-label classification rules by exploiting high-order label correlations. *Neurocomputing*, *417*, 176–186.

Nielsen, F. (2010). A family of statistical symmetric divergences based on jensen's inequality. *arXiv preprint arXiv:1009.4004*.

Norouzi, M., Fleet, D. J., & Salakhutdinov, R. R. (2012). Hamming distance metric learning. *Advances in neural information processing systems*, *25*.

Omar, A., Mahmoud, T. M., Abd-El-Hafeez, T., & Mahfouz, A. (2021). Multi-label arabic text classification in online social networks. *Information Systems*, *100*, 101785.

Papadimitriou, P., & Garcia-Molina, H. (2010). Data leakage detection. *IEEE Transactions on knowledge and data engineering*, *23*(1), 51–63.

Park, K., Hong, J. S., & Kim, W. (2020). A methodology combining cosine similarity with classifier for text classification. *Applied Artificial Intelligence*, *34*(5), 396–411.

Paul, A., & Chaki, N. (2019). Dimensionality reduction of hyperspectral images using pooling. *Pattern Recognition and Image Analysis*, *29*(1), 72–78.

Pavlinek, M., & Podgorelec, V. (2017). Text classification method based on self-training and lda topic models. *Expert Systems with Applications*, *80*, 83–93.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, *2*(11), 559–572.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Pereira, R. M., Costa, Y. M., & Silla Jr, C. N. (2020). Mltl: A multi-label approach for the tomek link undersampling algorithm. *Neurocomputing*, *383*, 95–105.

Pranckevičius, T., & Marcinkevičius, V. (2016). Application of logistic regression with part-of-the-speech tagging for multi-class text classification. *2016 IEEE 4th workshop on advances in information, electronic and electrical engineering (AIEEE)*, 1–5.

Qiu, S., Xu, B., Zhang, J., Wang, Y., Shen, X., De Melo, G., Long, C., & Li, X. (2020). Easyaug: An automatic textual data augmentation platform for classification tasks. *Companion Proceedings of the Web Conference 2020*, 249–252.

Raghavan, A., Umaashankar, V., & Gudur, G. K. (2019). Label frequency transformation for multi-label multi-class text classification. *KONVENS*.

Ruzzetti, E. S., Ranaldi, L., Mastromattei, M., Fallucchi, F., & Zanzotto, F. M. (2021). Lacking the embedding of a word? look it up into a traditional dictionary. *arXiv preprint arXiv:2109.11763*.

Rydning, J. (2021). *Worldwide global datasphere and global storagesphere structured and unstructured data forecast, 2021–2025*. Retrieved August 11, 2022, from https://www.idc.com/getdoc.jsp?containerId=US47998321

Sakia, R. M. (1992). The box-cox transformation technique: A review. *Journal of the Royal Statistical Society: Series D (The Statistician)*, *41*(2), 169–178.

Selva Birunda, S., & Kanniga Devi, R. (2021). A review on word embedding techniques for text classification. *Innovative Data Communication Technologies and Application*, 267–281.

Severyn, A., & Moschitti, A. (2015). Learning to rank short text pairs with convolutional deep neural networks. *Proceedings of the 38th international ACM SI-*

GIR conference on research and development in information retrieval, 373–382.

Shaikh, S., Daudpota, S. M., Imran, A. S., & Kastrati, Z. (2021). Towards improved classification accuracy on highly imbalanced text dataset using deep neural language models. *Applied Sciences*, *11*(2), 869.

Shi, G., Feng, C., Xu, W., Liao, L., & Huang, H. (2020). Penalized multiple distribution selection method for imbalanced data classification. *Knowledge-Based Systems*, *196*, 105833.

Singh, A. K., & Shashi, M. (2019). Vectorization of text documents for identifying unifiable news articles. *International Journal of Advanced Computer Science and Applications*, *10*(7).

Singh, S., & Mahmood, A. (2021). The nlp cookbook: Modern recipes for transformer based deep learning architectures. *IEEE Access*, *9*, 68675–68702.

Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*.

Spyns, P. (1996). Natural language processing in medicine: An overview. *Methods of information in medicine*, *35*(04/05), 285–301.

Spyromitros-Xioufis, E., Spiliopoulou, M., Tsoumakas, G., & Vlahavas, I. (2011). Dealing with concept drift and class imbalance in multi-label stream classification. *Twenty-Second International Joint Conference on Artificial Intelligence*.

Sulaiman, S., Wahid, R. A., Ariffin, A. H., & Zulkifli, C. Z. (2020). Question classification based on cognitive levels using linear svc. *Test Eng Manag*, *83*, 6463–6470.

Sun, S., Gao, L., & Liu, Y. (2010). Enhanced dye-sensitized solar cell using graphene-tio 2 photoanode prepared by heterogeneous coagulation. *Applied physics letters*, *96*(8), 083113.

Tahir, M. A., Kittler, J., & Bouridane, A. (2012). Multilabel classification using heterogeneous ensemble of multi-label classifiers. *Pattern Recognition Letters*, *33*(5), 513–523.

Tahir, M. A., Kittler, J., & Yan, F. (2012). Inverse random under sampling for class imbalance problem and its application to multi-label classification. *Pattern Recognition*, *45*(10), 3738–3750.

Tarekegn, A. N., Giacobini, M., & Michalak, K. (2021). A review of methods for imbalanced multi-label classification. *Pattern Recognition*, *118*, 107965.

Thormundsson, B. (2022). *Global chatbot market value 2018-2027*. Retrieved August 11, 2022, from https://www.statista.com/statistics/1007392/worldwide-chatbot-market-size/

Tian, J., Chen, S., Zhang, X., & Feng, Z. (2020). A graph-based measurement for text imbalance classification. *Ecai 2020* (pp. 2188–2195). IOS Press.

Tsoumakas, G., & Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, *3*(3), 1–13.

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2009). Mining multi-label data. *Data mining and knowledge discovery handbook*, 667–685.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(11).

Vapnik, V. (1998). The support vector method of function estimation. *Nonlinear modeling* (pp. 55–85). Springer.

Vijayarani, S., Ilamathi, M. J., Nithya, M., et al. (2015). Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, *5*(1), 7–16.

Wan, S., Duan, Y., & Zou, Q. (2017). Hpslpred: An ensemble multi-label classifier for human protein subcellular location prediction with imbalanced source. *Proteomics*, *17*(17-18), 1700262.

Wang, C., Song, Y., Li, H., Sun, Y., Zhang, M., & Han, J. (2017). Distant meta-path similarities for text-based heterogeneous information networks. *Proceedings of the 2017 ACM on conference on information and knowledge management*, 1629–1638.

Wang, J., & Dong, Y. (2020). Measurement of text similarity: A survey. *Information*, *11*(9), 421.

Wang, S., Zhou, W., & Jiang, C. (2020). A survey of word embeddings based on deep learning. *Computing*, *102*(3), 717–740.

Wang, Z.-Q., Sun, X., Zhang, D.-X., & Li, X. (2006). An optimal svm-based text classification algorithm. *2006 International Conference on Machine Learning and Cybernetics*, 1378–1381.

Wardhani, N. W. S., Rochayani, M. Y., Iriany, A., Sulistyono, A. D., & Lestantyo, P. (2019). Cross-validation metrics for evaluating classification performance on imbalanced data. *2019 international conference on computer, control, informatics and its applications (IC3INA)*, 14–18.

Weston, J., Watkins, C. et al. (1999). Support vector machines for multi-class pattern recognition. *Esann*, *99*, 219–224.

Wikipedia. (2022). *Reliability of wikipedia*. Retrieved August 11, 2022, from https://en.wikipedia.org/wiki/Reliability_of_Wikipedia

Wilson, S., Schaub, F., Dara, A. A., Liu, F., Cherivirala, S., Leon, P. G., Andersen, M. S., Zimmeck, S., Sathyendra, K. M., Russell, N. C., et al. (2016). The creation and analysis of a website privacy policy corpus. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1330–1340.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wu, D., Luo, X., Wang, G., Shang, M., Yuan, Y., & Yan, H. (2017). A highly accurate framework for self-labeled semisupervised classification in industrial applications. *IEEE Transactions on Industrial Informatics*, *14*(3), 909–920.

Wu, G., Tian, Y., & Liu, D. (2018). Cost-sensitive multi-label learning with positive and negative label pairwise correlations. *Neural Networks*, *108*, 411–423.

Xu, J. (2011). An extended one-versus-rest support vector machine for multi-label classification. *Neurocomputing*, *74*(17), 3114–3124.

Xu, Z., & Iwaihara, M. (2021). Semantic space-based self-training for semi-supervised multi-label text classification. *DEIM Forum E24-2*.

Yang, X., Fu, H., Zha, H., & Barlow, J. (2006). Semi-supervised nonlinear dimensionality reduction. *Proceedings of the 23rd international conference on Machine learning*, 1065–1072.

Ye, Z., Geng, Y., Chen, J., Chen, J., Xu, X., Zheng, S., Wang, F., Zhang, J., & Chen, H. (2020). Zero-shot text classification via reinforced self-training. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 3014–3024.

Yıldırım, F. M., Kaya, A., Öztürk, S. N., & Kılınç, D. (2019). A real-world text classification application for an e-commerce platform. *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1–5.

Yuan, P., Chen, Y., Jin, H., & Huang, L. (2008). Msvm-knn: Combining svm and k-nn for multi-class text classification. *IEEE international workshop on Semantic Computing and Systems*, 133–140.

Zhang, B., Bai, B., & Su, J. (2007). Semi-supervised text classification based on self-training em algorithm. *JOURNAL-NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY*, *29*(6), 65.

Zhang, H., Cao, L., Madden, S., & Rundensteiner, E. (2021). Lancet: Labeling complex data at scale. *Proceedings of the VLDB Endowment*, *14*(11), 2154–2166.

Zhang, M.-L., Li, Y.-K., Yang, H., & Liu, X.-Y. (2020). Towards class-imbalance aware multi-label learning. *IEEE Transactions on Cybernetics*.

Zhang, M.-L., & Zhou, Z.-H. (2013). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, *26*(8), 1819–1837.

APPENDIX A

EMBEDDINGS

Huggingface Embeddings 'stsb-roberta-large', 'all-MiniLM-L6-v2', 'all-MiniLM-L12-v2', 'all-mpnet-base-v1', 'all-mpnet-base-v2', 'all-roberta-large-v1', 'all-distilroberta-v1', 'albert-base-v2', 'ALBERT-xxlarg', 'bert-base-nli-mean-tokens', 'all-roberta-large-v1', 'distiluse-base-multilingual-cased-v1', 'multi-qa-mpnet-base-dot-v1', 'all-distilroberta-v1', 'bert-base-uncased', 'bert-base-nli-mean-tokens', 'distiluse-base-multilingual-cased-v1', 'distilbert-base-nli-mean-tokens', 'multi-qa-mpnet-base-dot-v1', 'nlpaueb/legal-bert-base-uncased', 'paraphrase-multilingual-MiniLM-L12-v2', 'paraphrase-mpnet-base-v2', 'paraphrase-MiniLM-L6-v2', 'paraphrase-xlm-r-multilingual-v1', 'saibo/legal-roberta-base', 'ALBERT-xlarge', 'sentence-t5-large', 'sentence-transformers/average-word-embeddings-glove.6B.300d', 'sentence-transformers/average-word-embeddings-glove.840B.300d'

Openai Embeddings 'text-similarity-babbage-001', 'text-similarity-ada-001', 'text-similarity-curie-001', 'text-similarity-davinci-001'

Google Embeddings 'universal-sentence-encoder'

APPENDIX B

INITIAL ANOVA RESULTS FOR OPP-115

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	36	1.68175	0.046715	29.86	0.000
Linear	6	1.14432	0.190719	121.89	0.000
balance_ratio	1	0.01253	0.012531	8.01	0.005
sim_calculation_type	1	0.33677	0.336766	215.22	0.000
batch_size	2	0.16012	0.080062	51.17	0.000
n_iter	1	0.00468	0.004683	2.99	0.084
sim_type	1	0.63021	0.630215	402.76	0.000
2-Way Interactions	14	0.48881	0.034915	22.31	0.000
balance_ratio*sim_calculation_type	1	0.00047	0.000465	0.30	0.586
balance_ratio*batch_size	2	0.00011	0.000053	0.03	0.967
balance_ratio*n_iter	1	0.00002	0.000024	0.02	0.901
balance_ratio*sim_type	1	0.00985	0.009846	6.29	0.012
sim_calculation_type*batch_size	2	0.02459	0.012296	7.86	0.000
sim_calculation_type*n_iter	1	0.00542	0.005415	3.46	0.063
sim_calculation_type*sim_type	1	0.38995	0.389952	249.21	0.000
batch_size*n_iter	2	0.01572	0.007861	5.02	0.007
batch_size*sim_type	2	0.03748	0.018738	11.98	0.000
n_iter*sim_type	1	0.00522	0.005218	3.33	0.068
3-Way Interactions	16	0.04862	0.003039	1.94	0.014
balance_ratio*sim_calculation_type*batch_size	2	0.00441	0.002203	1.41	0.245
balance_ratio*sim_calculation_type*n_iter	1	0.00003	0.000030	0.02	0.889
balance_ratio*sim_calculation_type*sim_type	1	0.00120	0.001196	0.76	0.382
balance_ratio*batch_size*n_iter	2	0.00044	0.000219	0.14	0.870
balance_ratio*batch_size*sim_type	2	0.00040	0.000198	0.13	0.881
balance_ratio*n_iter*sim_type	1	0.00037	0.000372	0.24	0.626
sim_calculation_type*batch_size*n_iter	2	0.00177	0.000885	0.57	0.568
sim_calculation_type*batch_size*sim_type	2	0.03085	0.015427	9.86	0.000
sim_calculation_type*n_iter*sim_type	1	0.00827	0.008273	5.29	0.022
batch_size*n_iter*sim_type	2	0.00089	0.000444	0.28	0.753
Error	2363	3.69747	0.001565		
Lack-of-Fit	11	0.01599	0.001453	0.93	0.512
Pure Error	2352	3.68149	0.001565		
Total	2399	5.37923			

Figure B.1: Initial ANOVA results for Opp-115 dataset.

APPENDIX C

REDUCED ANOVA RESULTS FOR OPP-115

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	19	1,67166	0,087982	56,48	0,000
Linear	6	1,14432	0,190719	122,43	0,000
balance_ratio	1	0,01253	0,012531	8,04	0,005
sim_calculation_type	1	0,33677	0,336766	216,18	0,000
batch_size	2	0,16012	0,080062	51,39	0,000
n_iter	1	0,00468	0,004683	3,01	0,083
sim_type	1	0,63021	0,630215	404,55	0,000
2-Way Interactions	10	0,48822	0,048822	31,34	0,000
balance_ratio*sim_type	1	0,00985	0,009846	6,32	0,012
sim_calculation_type*batch_size	2	0,02459	0,012296	7,89	0,000
sim_calculation_type*n_iter	1	0,00542	0,005415	3,48	0,062
sim_calculation_type*sim_type	1	0,38995	0,389952	250,32	0,000
batch_size*n_iter	2	0,01572	0,007861	5,05	0,007
batch_size*sim_type	2	0,03748	0,018738	12,03	0,000
n_iter*sim_type	1	0,00522	0,005218	3,35	0,067
3-Way Interactions	3	0,03913	0,013042	8,37	0,000
sim_calculation_type*batch_size*sim_type	2	0,03085	0,015427	9,90	0,000
sim_calculation_type*n_iter*sim_type	1	0,00827	0,008273	5,31	0,021
Error	2380	3,70756	0,001558		
Lack-of-Fit	28	0,02608	0,000931	0,60	0,954
Pure Error	2352	3,68149	0,001565		
Total	2399	5,37923			

Figure C.1: Reduced ANOVA results for Opp-115 dataset.