NATIONAL AND KAPODESTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

MSC INFORMATICS AND TELECOMMUNICATIONS

MSC THESIS

Real-time Fake-news Detection in Greek using a Browser Extension

Odysseas A. Trispiotis

Supervisor: **Alex Delis,** Professor NKUA

ATHENS

MARCH 2021

ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΠΣ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εύρεση Ψευδών Ελληνικών Ειδήσεων σε Πραγματικό Χρόνο με Επέκταση Φυλλομετρητή

Οδυσσέας Α. Τρισπιώτης

Επιβλέπων: **Αλέξης Δελής**, Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2021

MSC THESIS

Real-time Fake-news Detection in Greek using a Browser Extension

Odysseas A. Trispiotis
S.N.: M1599

SUPERVISOR: **Alex Delis,** Professor NKUA

EXAMINATION COMMITTEE: **Alex Delis,** Professor NKUA
Panagiotis Rondogiannis, Professor NKUA
Alexandros Ntoulas, Assistant Professor NKUA

March 2021

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εύρεση Ψευδών Ελληνικών Ειδήσεων σε Πραγματικό Χρόνο με Επέκταση Φυλλομετρητή

Οδυσσέας Α. Τρισπιώτης
Α.Μ.: Μ1599

ΕΠΙΒΛΕΠΩΝ: **Αλέξης Δελής**, Καθηγητής ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Αλέξης Δελής**, Καθηγητής ΕΚΠΑ
Παναγιώτης Ροντογιάννης, Καθηγητής ΕΚΠΑ
Αλέξανδρος Ντούλας, Επίκουρος Καθηγητής ΕΚΠΑ

Μάρτιος 2021

ABSTRACT

Fake news have become more prevalent in recent years with the increased popularity and use of social media. Such news sometimes can be very dangerous, as they deceive readers and can push the public towards dangerous actions. So we deem critical to detect this type of publicly available information in real-time.

This thesis outlines our work in creating an experimental web browser extension that recognizes if a web page containing a Greek news article is fake with the process being entirely transparent to the user. We do that by carrying out our analysis real-time and ascertain the probability of this article to be illegitimate using contemporary ML techniques.

Initially, we collected a good amount of Greek articles (~35,000) and we marked the fake ones, in order to create a dataset. Then we used this dataset along with basic feature extraction techniques in order to create an input for training various classification algorithms. The result of the above process produces a machine learning model, which can be stored in a file and used for predictions in new unseen data. After that, we compare the results of produced models based on some common metrics. Then we chose the model, which gave us the best results, and we create a backend REST A.P.I. based on this. Finally, we create a separate browser extension as a U.I. client to preview the results.

The results of the above process were quite encouraging considering the amount of available data and showed that our extension can predict fast (~35ms) and with great accuracy (~95%) if an article is fake news or not. There are several open issues for improvement and future research, such as the fake news detection by using various neural networks instead of classification algorithms. Also, the automatic retrain of the model with new data and the handling of which part of web page's content is an article are some open issues from current thesis.

SUBJECT AREAS: Machine Learning, Automatic Data Tagging

KEYWORDS: classification, feature extraction, model, probability prediction, browser extension, data extraction, data tagging

ΠΕΡΙΛΗΨΗ

Η διασπορά ψευδών ειδήσεων είναι ιδιαίτερα δημοφιλής τα τελευταία χρόνια, κυρίως λόγω της αυξανόμενης δημοτικότητας και χρήσης των κοινωνικών δικτύων (social media). Τέτοιου είδους ειδήσεις κάποιες φορές μπορεί να είναι πολύ επικίνδυνες, καθώς εξαπατούν τους αναγνώστες και μπορούν να τους ωθήσουν σε επικίνδυνες ενέργειες. Έτσι η ανίχνευση σε πραγματικό χρόνο τέτοιου είδους ειδήσεων είναι πολύ σημαντική.

Η συγκεκριμένη εργασία περιγράφει τη διαδικασία υλοποίησης ενός πειραματικού browser extension που αναγνωρίζει αν ο χρήστης βρίσκεται σε κάποια ιστοσελίδα ελληνικού ειδησεογραφικού άρθρου και προβλέπει σε πραγματικό χρόνο, χωρίς να γίνει αντιληπτή από τον χρήστη η διαδικασία της πρόβλεψης, την πιθανότητα το συγκεκριμένο άρθρο να αποτελεί ψευδή είδηση χρησιμοποιώντας machine learning.

Αρχικά, για τη δημιουργία της συγκεκριμένης επέκτασης, συλλέξαμε έναν σχετικά καλό αριθμό από ελληνικά ειδησιογραφικά άρθρα (~35.000) και ξεχωρίσαμε ποια από αυτά αποτελούν αληθινή και ποια ψευδή είδηση, ώστε να δημιουργήσουμε ένα dataset. Στη συνέχεια χρησιμοποιήσαμε το συγκεκριμένο dataset μαζί με τεχνικές για feature extraction από κείμενο ώστε να εκπαιδεύσουμε διάφορους αλγόριθμους ταξινόμησης. Το αποτέλεσμα της παραπάνω διαδικασίας παράγει ένα μοντέλο μηχανικής μάθησης, το οποίο μπορεί να αποθηκευτεί σε κάποιο αρχείο και να χρησιμοποιηθεί για προβλέψεις σε νέα δεδομένα. Έπειτα συγκρίναμε τα αποτελέσματά των παραγόμενων μοντέλων με βάση κοινές μετρικές. Τέλος επιλέξαμε το μοντέλο που έδινε τα καλύτερα αποτελέσματα και το χρησιμοποιήσαμε σαν βάση για να κατασκευάσουμε μια επέκταση browser που επικοινωνεί με το συγκεκριμένο μοντέλο αναγνώρισης ψευδών ειδήσεων.

Τα αποτελέσματα της παραπάνω διαδικασία ήταν αρκετά ενθαρρυντικά για το πλήθος των διαθέσιμων δεδομένων και έδειξαν πως η επέκτασή μας μπορεί να προβλέψει με αρκετά μεγάλη ακρίβεια (~95%) και γρήγορα (~35 ms) αν ένα άρθρο αποτελεί ψευδή είδηση. Βέβαια υπάρχουν αρκετά ανοιχτά κομμάτια προς βελτίωση και μελλοντική έρευνα, όπως η ανίχνευση ψευδών ειδήσεων με χρήση διαφόρων νευρωνικών δικτύων αντί για αλγορίθμους ταξινόμησης. Ακόμη κάποια ανοιχτά κομμάτια που χρήζουν επέκτασης στην παρούσα εργασία είναι το retrain του μοντέλου και η αναγνώριση για το ποιο μέρος της ιστοσελίδας αποτελεί ειδησεογραφικό άρθρο.

ΘΕΜΑΤΙΚΕΣ ΠΕΡΙΟΧΕΣ: Μηχανική Μάθηση, Αυτόματη Σήμανση Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ταξινόμηση, εξαγωγή χαρακτηριστικών, μοντέλο, πρόβλεψη πιθανότητας, επέκταση φυλλομετρητή, εξαγωγή δεδομένων, σήμανση δεδομένων

To my loving mother and late father as well as to my friends…

ACKNOWLEDGEMENTS

I would like to thank the supervisor, professor Alex Delis, and my friend Michalis Papakonstantinou for cooperation and their valuable contribution on fulfillment of this thesis.

CONTENTS

FIGURES LIST

TABLES LIST

PREFACE

The project was challenging, as we had to create a real-time browser extension that handles web pages that contain Greek article and tries to predict if this article is fake by using machine learning. Therefore, we had to collect the data and to create various separate components. We spent a lot of time crawling the web in order to collect data for our dataset. Then comparing various ML algorithms along with feature extraction and data preprocessing techniques in order to set up a generic RESTFUL API was also a challenge. Finally, we had to explore a completely new world of web browser extensions. At the end of the day, conducting extensive investigation on all these separate fields has allowed us to answer the question that we identified.

I would like to thank my supervisor for their excellent guidance and support during this process.

To all my colleagues at UOA and at my job: I would like to thank you for your wonderful cooperation as well. It has been always helpful to discuss ideas about my research with you.

My mother and my friends deserve a particular note of thanks. Your wise counsel and kind words have served me well. I also want to dedicate this thesis to my father who has passed away but he always inspires me.

I hope you enjoy reading this thesis.

1. INTRODUCTION

Nowadays most of the people prefer to read news from various sources like social media rather than traditional news outlets due to low cost and easy access. Most of the time, the readers consume those news without filtering the reliability of source or their content. Because of this many fake news are transmitted as true. In recent years, fake news have increasingly become a dangerous prospect made for online users. A characteristic example of this is the spreading of various kinds of fake news during the 2016 US presidential elections [1] that helped popularize the term.

In the Cambridge dictionary, fake news is defined as false stories that appear to be news, spread on the internet or using other media, usually created to influence political views or as a joke. Also fake news can be defined as the prediction of the chances of a particular news article (news report, editorial, expose, etc.) being intentionally deceptive [2]. All the above are only unofficial definitions and there is not a proper definition of fake news, so this is the first challenge in detecting fake news. Another challenge is that this type of news are spread almost automatically in the web and especially the social media. In addition, the language used in fake news and legitimate articles are very similar because fake news are created with the intention to be trusted. Therefore, fake news detection has become a critical matter that yetis technically very challenging.

Our main goal is to create a stable and accurate model to predict Greek fake news in real-time. We have to carefully adopt the most suitable and promising of the ML models so as to empower a viable solution to the problem through our work. In our study, we consider applying various feature extraction techniques combined with traditional classification algorithms, in order to avoid or to overcome those limitations for our problem. We can say that our approach detects fake news by representing articles content in terms of features within a classification framework and focus mainly on the linguistic indicators of fake news [3].

An extension to our main goal is to get the combination of feature extraction method and machine learning algorithm that fits better to our problem and use it in a browser extension, in order to make prediction in real-time. By real-time we mean that the user will have a prediction if the content of the webpage she reads is fake just when she enters the webpage and without discernible understanding of the service. Our overall objective has been to offer realistic computational solutions to the problem of instantly identifying fame news and along the way to materialize an amply test, well behaving and useful application.

2. DATASET CREATION

The first step to achieve our target was to identify an appropriate amount of data, in order to train and evaluate our machine learning models. There are plenty of datasets available for widely spoken languages including English, French etc. Unfortunatelly, for Greek there are very limited resources that may be not applicable to our objective, so we had to create our own dataset. In this section, we describe the steps that we follow in order to collect and process various Greek articles to create our dataset. Figure 1 depicts the general overview of our approach.

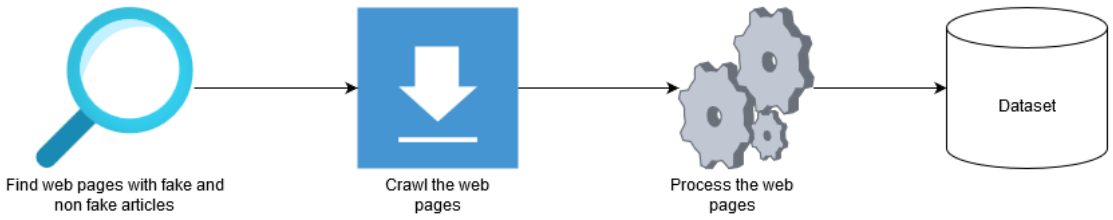


Figure 1: Dataset Creation Process

- The first step is the search of web pages with surely fake or non-fake content. There are many reliable online newspapers, so we can say that the tricky part on our implementation was to find fake articles. Fortunately, there are some “troll” websites like tokoulouri.com and tovatraxi.com with valid amount of content (~5,100 fake articles). In addition, there is ellinikahoaxes.gr which presents fake articles published by 3rd parties and articulates why these pieces are not real.
- The second step is to crawl the web in order to download articles’ pages and to process them. For this purpose, we used *Crawler4J* [4] web crawler, a powerful tool for web crawling using Java. The way our crawling layer works is as follows:
  - First we need to identify our seed url, used by the crawler to initiate its work
  - Then we have to define 2 sets or patterns. One for the pattern of the urls to follow and another for the pattern of the ones to store.

  To make this process as agnostic to the website crawled as possible we expanded the code of *Crawler4J* to take as input a YML configuration file with all the necessary properties. In total, 184 websites are currently crawled, and ~130,000 pages have been collected so far.
- The third step is the processing of each page, in order to extract the real article and its title. That was the trickiest step because there is no standard formalization for which part of the web page is the article, so we need to define that. In addition, as we said earlier one of our main sources does not provide fake articles, but it explains why various articles from other sources are fake, so we need to extract these articles from the pages’ content. The following algorithm describes the steps we follow in order to extract the articles from the downloaded webpages.

```
FUNCTION boolean isArticlePage(String pagePath) {
 SET articles TO newspaper.articles(pagePath)

 IF articles is Empty
   RETURN false;
 ELSE
  RETURN true;
}

FUNCTION List handleArticles(String pagePath) {
 INITIALIZE results TO empty List
 SET articles TO newspaper.articles(pagePath)

 FOR each article in articles{
  IF article.metadata.tag is 'article' and article.metadata.
     language = 'greek'
   ADD article TO results
 }

 RETURN results
}

FUNCTION List getAllArticles(String pageFolderPath){
 INITIALIZE allArticles TO empty List

 FOR each pagePath in pageFolderPath{

  IF isArticlePage(pagePath){
   SET results TO handleArticles(pagePath)
   ADD ALL results TO allArticles
  }
  ELSE{
   SET urls TO BeautifulSoup.find_all('a_tags', pagePath)
   FOR each url in urls{
    SET newPagePath TO crawl(url)
    IF isArticlePage(newPagePath){
     SET results TO handleArticles(pagePath)
     ADD ALL results TO allArticles
    }
   }
  }
 }

 RETURN allArticles;
}
```

Figure 2: Web Page Processing Algorithm

Figure 2 depicts the algorithmic steps to process a web page and extract its articles. As we can see, the algorithm is very simple and contains only two main steps as all our input pages are article pages or pages that refer to an article page.

- The algorithm firstly checks if the given webpage contains one or more articles. If so, it handles all articles by extracting their title, text ,author etc. from the web page. For this purpose, we used Python's *newspaper* [5] library along with a set of rules. The *newspaper* library is a powerful library that finds out possible articles on a web page along with various information like title, text, author, metadata etc.
- Then on those possible articles, we check if their metadata contains the "article" tag and if they are written in Greek. If the web page is not an article we get all page's urls that may refer to the original article page and download them. Finally we process the new downloaded pages on the same way as the first step. In order to achive that firstly we used Python's *BeautifulSoup* [6] library to get page's urls that may refer to the original article from html's a-tags. Then we crawl those urls by using our web crawler or with traditional wget for archive pages.

• The final step is to categorize our data in fake and non fake and store them in a useful format. We prefered to store them as a csv file for easy and fast handling and processing on our code.

In order to categorize a downloaded articles as fake or non-fake, we used a blacklist logic based on the source of the article. We know that some sources like ellinika-hoaxes.gr, tokoulouri.com and tovatraxi.com has only fake content, so we can easily catgorize articles from those sources as fake (blacklisted source) and the other articles as non-fake.

We used Python's *Pandas* library [7] in order to store our categorized articles in a csv file. *Pandas* is a powerful library for storing and handling huge files and also has some very useful features for handling csv files.

Our final csv file contains the following attributes:

- **id:** unique id for a news article
- **title:** the title of a news article
- **author:** author of the news article; could be incomplete
- **text:** the text of the article
- **label:** a label that marks the article as potentially fake (1 for fake and 0 for non-fake)

You can find our generated dataset here: fake-news-detection-public-dataset

Figure 3 depicts the algorithmic steps to categorize a downloaded article as fake or non-fake and to store the final result in a csv file.

```
1
2  FUNCTION void categorizeAndStoreToCSV(String articlesFolderPath){
3
4   INITIALIZE allCategorizedArticles TO empty List
5   INITIALIZE id TO 1
6   INITIALIZE fakeSources TO
7    [ellinikahoaxes, tokoulouri, tovatraxi]
8
9   FOR each article in articlesFolderPath{
10
11   INITIALIZE categorizedArticle TO empty Article
12
13   SET categorizedArticle.id TO id
14   SET catgorizedArticle.title TO article.title
15   SET categorizedArticle.author TO article.author
16   SET categorizedArticle.text TO article.text
17
18   IF article.source IN fakeSources
19    SET categorizedArticle.label TO 1
20   ELSE
21    SET categorizedArticle.label TO 0
22
23   SET id TO id + 1
24
25   ADD categorizedArticle TO allCategorizedArticles
26  }
27
28  pandas.to_csv(allCategorizedArticles)
29 }
```

Figure 3: Article Categorization and Store Algorithm

3. FAKE NEWS DETECTION

In this section, we outline our general approach to detect fake news using various feature extraction methods combined with various classifiers. Figure 4 shows an overview of our proposed model.

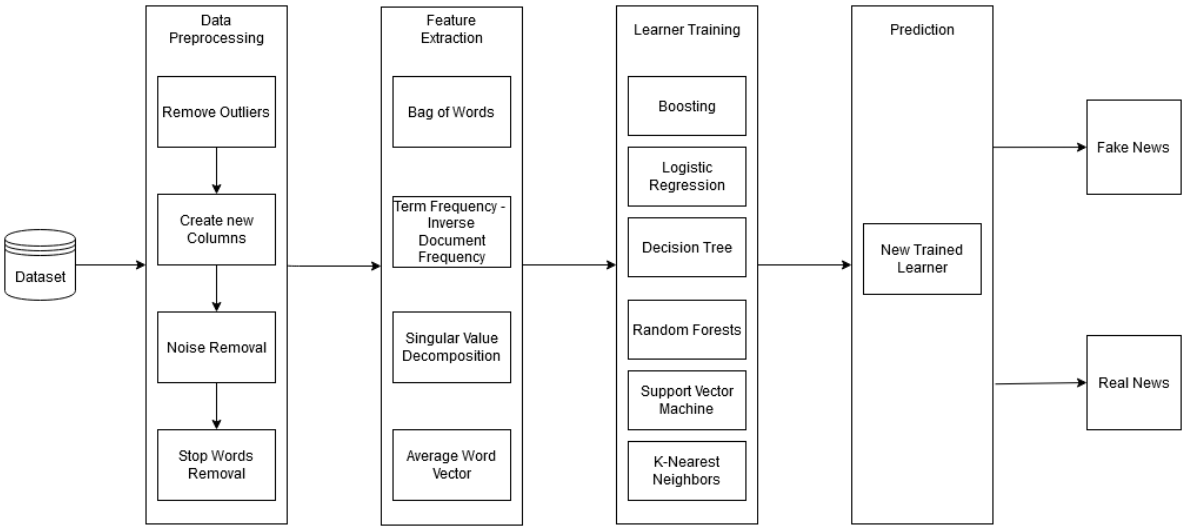


Figure 4: Model Overview

- The first step is preprocessing the dataset by removing useless columns or generating new ones and removing invalid characters that are not necessary for text mining or classification purposes.
- The second step is the feature extraction. We used in total four different text mining techniques [8] to extract our features including Bag of Words(BoW) [9], Term Frequency-Inverse Document Frequency(TF-IDF) [10], Singular Value Decomposition (SVD)[11] and Average Word Vector(Word2Vector) [12, 13]. BoW and TF-IDF are word frequency techniques that extract features in form that can easily be combined with classification algorithms. On the other hand Average Word Vector is a word embedding techniques that extracts features in n-dimensional sequence array and can be combined with classifiers. SVD is a popular technique for dimensionality reduction and a method to identify a subspace in which the data approximately lies, it can be used an extension to word frequency techniques.
- The third step is training the classifiers using the previous extracted features, in order to determine new coming data into fake or real. We used 10-fold cross-validation in training process of every algorithm and so the dataset is been split into 90% training data and 10% testing data on each fold. We trained and evaluated six classifiers [8] from different algorithm families including Boosting [14], Logistic Regression [15], Decision Tree [16], Random Forests [17], Support Vector Machines [18] and K-Nearest Neighbors [19]. For each combination of learning algorithm and feature method, a *sklearn* pipeline [20] was created. Intuitively, a pipeline is a sequence of steps to be followed in order to get the results we want. The output of each step is the input to next step. In this way, we made our framework modular and managable. Thus, the performance evaluation was the same for all cases and the results as much as possible fairly. In this work, the first steps of the pipeline are the features options and then one final ML algorithm as learner. Each feature and classifier is the same, wherever it is used.

3.1 Data Preprocessing Phase

We applied various data preprocessing steps to both our training and test data to reduce the size of the actual data and to improve feature extraction. We split data preprocessing into two components. The first one is a custom implementation that is executing only once during data loading. This component is used to remove useless information from our data and transform them to a more useful format. The second component is based on some standard parameters of Python's *sklearn* library and it is executed after data loading and before feature extraction. This component is used to remove more specific information from our data, like language's stopwords.

For our custom implementation, we first remove various columns from dataset, including the author, id, empty columns etc., as this kind of information is useless in our content-based approach to detect fake news. So practically, we maintain only the article's title and text content.

After that, we append article's title to its content in order to achieve better classification results because it is common for fake news to have "clickbait" titles with specific formal, so we created a new column called total.

Raw text and title are an unstructured form of data, so our new column can contain noisy context like consecutive spaces, newline after html tags, invalid html tags etc. [21]. Finally, we apply the following rules to remove noisy content from total column using regular expressions:

1. Remove spaces after a tag opens or closes
2. Replace consecutive spaces
3. Add newline after a
4. Add newline after and and <h1/>
5. Remove <head> to </head>
6. Show links instead of texts
7. Remove remaining tags like , etc.
8. Remove spaces at the beginning

After the data loading and before the beginning of feature extraction we remove the stop words [22] that are insignificant in a language and create noise when used as features in text classification, using ntlk standard stopwords. We used stopword parameter for both standard sklearn transformers and our custom implemented transformers.

3.2 Feature Extraction Techniques

We outlined the following main feature extraction approaches: Word frequency (or weighted words) and word embedding [8]. Word embedding techniques learn from sequences of words by taking into consideration their occurrence and co-occurrence information. Also, these methods are unsupervised models for generating word vectors. In contrast, weighted words features are based on counting words in documents and can be used as a simple scoring scheme of word representation.

We used four different techniques to extract features from our preprocessed dataset based on various text-mining algorithms including Bow, TF-IDF, SVD and Average Word Vector (Word2Vec). BoW and TF-IDF are word frequency techniques that extract features in form that can easily be combined with classification algorithms. On the other hand Average Word Vector is a word embedding techniques that extracts features in n-dimensional

sequence array and can be combined with classifiers. SVD is a popular technique for dimensionality reduction and a method to identify a subspace in which the data approximately lies, it can be used an extension to word frequency techniques. In this work feature extraction is the first step to our pipeline. We used python's *sklearn* and *gensim* libraries to implement our feature extraction methods.

3.2.1 Bag of Words (BoW)

Bag of words is one the simplest text mining technique. This method is based on counting number of the words in each document, and assign it to feature space. The following models a text document using bag-of-words. Here is a simple text document with its BoW representation:

Simple Text = John likes to watch movies. Mary likes movies too.
BoW = [”John”,”likes”,”to”,”watch”,”movies”,”Mary”,”likes”,”movies”,”too”]

After transforming the text into a BoW, we can calculate various measures to characterize the text. The most common type of characteristics, or features calculated from the Bag of Words model is the frequency of a term (F), which measures the number of times a term appears in the text. For the example above, we can construct the following list to record the term frequencies of all the distinct words.

F = [1, 2, 1, 1, 2, 1, 1, 0, 0]

In the previous example, the Bag of Words representation will not reveal that the verb ”likes” always follows a person's name in this text. As an alternative, the n-gram model can store this spatial information. Applying to the same example above, a bigram model will parse the text into the following units and store the term frequency of each unit as before.

BoW = [”John likes”, ”likes to”, ”to watch”, ”watch movies”, ”Mary likes”, ”likes movies”, ”movies too”]

BoW is a very simple method to extract the most descriptive terms in a document is easy to compute and works fine with unknown words. In addition, it is very easy to compute the similarity between two documents by using it. On the other hand, BoW has syntactic and semantic limitations, as it does not capture the position and the meaning of a word in the text. Also common words effect on the results.

We used *CountVectorizer* [23] of python *sklearn* library to implement our BoW feature extraction method as the first step of a pipeline. We also used *ngram_range* parameter giving it a value of (1, 2), to indicate that each word can be packed alone (unigram model) or along with its sibling (bigram model).

3.2.2 Term Frequency – Inverse Document Frequency (TF-IDF)

We combined BoW technique with TF-IDF to overcome limitations of BoW [24] and to improve recall and precision [24].

Term Frequency (TF): The ratio of the number of times a specific word appears in a document compared to the total number of words in that document. TF increases as the number of occurrences of that word within the document increases and each document has its own TF. Term Frequency can be defined mathematically as follows:

$$TF(i,j) = \frac{\text{Frequency of term i in document j}}{\text{Total words in document j}}$$

Inverse Document Frequency (IDF): Measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, may appear many times but have little importance. Thus, we need to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. Inverse Document Frequency can be defined mathematically as follows:

$$IDF(i) = \log(\frac{\text{Total documents}}{\text{Documents with term i}})$$

Based on the above, the mathematical representation of weight of a term in a document by TF-IDF is given:

$$W(i,j) = TF(i,j) * IDF(i)$$

Although TF-IDF tries to overcome the problem of common terms in document, it still suffers from syntactic and semantic limitations. In particular, TD-IDF cannot account for the similarity between words in the document since each word is presented as an index. We used *CountVectorizer* [23] and *TfidfTransformer* [25] of python *sklearn* library to implement our BoW feature extraction method as the first step of a pipeline. We also used *ngram_range* parameter giving it a value of (1,2), to indicate that each word can be packed alone (unigram model) or along with its sibling (bigram model).

3.2.3 Singular Value Decomposition (SVD)

In order to overcome semantic limitations of both BoW and TF-IDF we extended our implementation using words co-occurrence matrix. Words co-occurrence matrix is computed simply by counting how two or more words occur together in a given corpus. As an example of words co-occurrence, consider a corpus consisting of the following documents:

- penny wise and pound foolish
- a penny saved is a penny earned

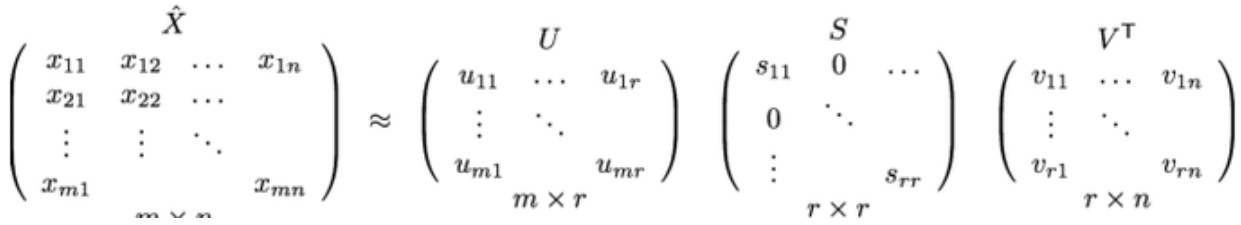
Table 1 shows how many times the words “a” and “penny” are followed by other words of the corpus. We can summarize co-occurrence statistics for words “a” and “penny” as:

Table 1: Co-occurrence Matrix

	a	**and**	**earned**	**foolish**	**is**	**penny**	**pound**	**saved**	**wise**
a	0	0	0	0	0	2	0	0	0
penny	0	0	1	0	0	0	0	1	1

We can see at the above table that “a” is followed twice by “penny” while words “earned”, “saved”, and “wise” each follows “penny” once in our corpus. The count shown above is called bigram frequency; it looks into only the next word from a current word. Therefore, given a corpus of N words, we need a table of size *NxN* to represent bigram frequencies of all possible word-pairs. Such a table is highly sparse as most frequencies are equal to zero.

The co-occurrence matrix is not the word vector representation that is generally used. Instead, this Co-occurrence matrix is decomposed into factors using techniques like SVD. The combination of these factors forms the word vector representation. Singular Value Decomposition (SVD) is a concept from linear algebra widely used in statistics and data analysis as a noise reduction algorithm and to avoid the over-fitting problem [26]. SVD is based on the following matrix equation:



The above states that a rectangular matrix X can be decomposed into three other matrices U, S and V where U and V are both orthogonal matrices. The product of U and S gives the word vector representation and V gives the word context representation, so we have semantic information using matrix V.

- **U:** consists of the orthonormal eigenvectors of:

$$XX^T$$

- **V:** consists of the orthonormal eigenvectors of:

$$X^TX$$

- **S:** is a diagonal matrix consisting of the square root of the eigenvalues of:

$$U \text{ or } V$$

In SVD, the matrix X is typically a word document matrix; it is a way of representing a document and text as a highly dimensional vector space model referred to as hyperspace document representation. SVD takes high dimensional variable data and reduces it to a lower dimensional space that more clearly depicts the underlying structure of the data. SVD reduces noise and redundancy in the data leaving you with new dimensions that capture the essence of existing relationships.

SVD analysis also preserves the semantic relationship between words [27] and produces more accurate word vector representations than BoW and TF-IDF [27]. Although SVD tries to overcome the semantic limitations of the previous methods, it still suffers from syntactic limitations. In addition, SVD requires huge memory compared to the other two techniques in order to store the *NxN* co-occurrence matrix. For the corporate of our problem with size 35,000 documents (*N=35,000*) we need an array of floats with size 35,000x35,000 for the co-occurrence matrix, so we need about 10GB of memory on a 64bit system only to store the co-occurrence matrix.

We used *TruncatedSVD* [28] method of *sklearn* along with previous TF-IDF method in order to implement SVD and use it as the first step of a pipeline. First, we obtained the number of features using the BoW and then we applied the *TruncatedSVD* method, which works better on sparse arrays. For the purposes of this project, we obtained the 20% of the features for memory and performance reasons, because as we said earlier we need a lot of memory just to store the co-occurrence matrix for our problem. Also we saw that the usage of less than 20% of the features gave as clearly worst results and the usage of more features until the number of 70-80% gave as the same results with 20%. Finally, we used the randomized algorithm (default) for the SVD because it appeared to be slightly more efficient.

3.2.4 Average Word Vector (W2V)

T. Mikolov presented ”word to vector” representation as a word embedding architecture [12, 13]. Word2vec is a simple, one hidden layer neural network that sums word embeddings and instead of minimizing a multi-class logistic loss (softmax), it minimizes a binary logistic loss on positive and negative samples, allowing to handle huge vocabularies efficiently [29].

Average Word Vector is a common technique to generate new embeddings to sentences, paragraphs or documents, using an existing pre-trained Word2Vec model, by averaging the word vectors to create a single fixed size embedding vector as we can see in the following equation:

$$\begin{bmatrix} W_{11} \\ W_{12} \\ \\ \\ W_{1n} \end{bmatrix}^{W_1} + \begin{bmatrix} W_{21} \\ W_{22} \\ \\ \\ W_{2n} \end{bmatrix}^{W_2} + \dots + \begin{bmatrix} W_{n1} \\ W_{n2} \\ \\ \\ W_{nn} \end{bmatrix}^{W_n} = \begin{bmatrix} \frac{W_{11}+W_{21}+\dots+W_{n1}}{n} \\ \\ \\ \frac{W_{1n}+W_{2n}+\dots+W_{nn}}{n} \end{bmatrix}^{D}$$

Generally, word embedding techniques like Word2Vec or Average Word Vector can capture the meaning in the words (semantics) and their position in the text (syntactics). As we can see Average Word Vector tries to overcome the semantic and syntactic limitations of BoW and SVD but it cannot capture the meaning of the word from the text and also fails to capture out of vocabulary words from a corpus [8]. Also sometimes it needs a huge corpus for trainning in order to be efficient [8].

We used a pre-trained Word2Vec model provided by Google by a Python’s library called *gensim* [30] as the core to our Average Word Vector implementation. This model was trained on 100 billion words of Google News and contains 300-dimensional vectors for 3 million words and phrases. Based on that model we built up a new transformer (Average Word VectorTransformer) in order to transform the data in average word vectors and to be compatible with sklearn pipeline.

3.3 Machine Learning Algorithms

In this section, we outline machine learning algorithms and their implementation we used in this thesis. Our problem is a text classification problem we experimented with six different machine learning algorithms for text classification [8], in order to compare them and see exactly which one fits better to our problem and why. All of them are traditional classification algorithms from different families including Boosting, Logistic Regression, Decision Tree, Random Forests, Support Vector Machines and K-Nearest Neighbors.

3.3.1 Boosting

Boosting is an Ensemble learning meta-algorithm for primarily reducing variance in supervised learning. Boosting is basically a family of machine learning algorithms that creates a strong learner from a number of weak learners. It was first introduced by R.E. Schapire [14] in 1990 as a technique for boosting the performance of a weak learning algorithm.

A weak learner is defined to be a Classification that is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong

learner is a classifier that is arbitrarily well correlated with the true classification.

For this thesis, we used *sklearn's AdaBoostClassifier* [31] that implements the AdaBoost-SAMME algorithm [32]. It starts by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. Therefore, the result is a new strong classifier.

3.3.2 Logistic Regression

Logistic Regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. Logistic Regression is used to describe data and to explain the relationship between one dependent binary variable and one, more nominal, ordinal, interval, or ratio-level independent variables. The logistic regression classifier was introduced and developed by statistician David Cox in 1958 [15] and predicts probabilities rather than classes [33]. The goal of LR is to train from the probability of variable Y being 0 or 1 for given x.

For this thesis, we used *sklearn's LogisticRegression* [34] classifier that implements regularized logistic regression [35].

3.3.3 Decision Tree

One of earlier classification algorithm for text and data mining is decision tree [36]. Decision trees (DTC's) are used successfully in many diverse areas of classification, such as text and document classification [16]. The structure of this technique includes a hierarchical decomposition of the data space (only train dataset). Decision tree as classification task was introduced by D. Morgan and developed by JR. Quinlan [37]. The main idea is creating trees based on the attributes of the data points, but the challenge is determining which attribute should be in parent level and which one should be in child level. To solve this problem, De Mantaras [38] introduced statistical modeling for feature selection in tree.

For this thesis, we used *sklearn's ExtraTreesClassifier* [39], which is a decision tree classifier. During classification process, classifier fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

3.3.4 Random Forests

Random Forests (RF) or random decision forests technique is an ensemble learning method for text classification that was firstly introduced by T. Kam Ho in 1995 [17]. The technique was later developed by L. Breiman in 1999 [40] who found converged for RF as a margin measure. The main idea of RF is generating random decision trees in order to avoid the overfitting problem of DTC [41]. Therefore, we can say that RF is an extension to Decision Trees.

For our thesis, we used *sklearn's RandomForestClassifier* [42] that implements the Random Forests algorithm. During classification process, the RFC fits a number of DTCs on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

3.3.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a ML algorithm that was firstly introduced by Vapnik and Chervonenkis [18] in 1963. Original version of SVM was designed for binary linear classification problems, but many researchers have worked on multi-class problems using this authoritative technique [43]. In addition, a nonlinear version of SVM was introduced

by BE. Boser [44] in early 1990s. The main idea behind linear and non-linear SVM is the same. The data are placed on the n-dimensional space based on their characteristics and SVM tries to find the hyperplane that best divides the data into several classes. A hyperplane in an n-dimensional Euclidean space is a flat n-1 dimensional subset that divides the space into two disconnected parts, so for linear problems this hyperplane is a line. In order to find the best hyperplane SVM algorithm finds the points closest to the line from all classes, which are called support vectors. After that, the algorithm computes the distance between the line and the support vectors. The hyperplane for which this distance is maximum is the optimal hyperplane.

For this thesis, we used *sklearn's inear SVC* [45] which is an implementation of SVM algorithm, setting the probability argument to true, in order to get the classification results as probabilities.

3.3.6 K-Nearest Neighbors

K-Nearest Neighbors algorithm (k-NN) is a non-parametric method proposed by Thomas Cover [19] which for classification or regression. In both classification and regression, the input consists of the k closest training examples in the feature space [46]. The output depends on whether k-NN is used for classification or regression.

The basic concept behind k-NN is given a test document x, the KNN algorithm finds the k nearest neighbors of x among all the documents in the training set, and scores the category candidates based the class of k neighbors. The similarity of x and each neighbor's document could be the score of the category of the neighbor documents. After sorting the score values, the algorithm assigns the candidate to the class with the highest score from the test document x [46]. The algorithm relies on distance for classification so normalizing the training data can improve its accuracy.

For this thesis, we used *sklearn's KNeighborsClassifier* [47], which implements the k-NN algorithm.

The overall implementation of our fake news detection process is available here: fake-news-detection

4. EVALUATION

In this section, we outline our evaluation process and provide key results. Our evaluation process is separated into two sections. The first one is concerened with the selection of parameters for every ML algorithm and its optimization. The second one is the comparison of every selected model based on widely used metrics. These metrics are precision, recall, F-measue which is a combination of precision and recall, accuracy and AUC ROC curve [48] and are based on true-false positive and true-false negative values [49]. We also used mean absolute error and MAE folds curve in order to detect overfitting or underfitting problems [50]. Overfitting referes to the situation where a model learns the data but also the noise that is part of training data to the extent that it negatively impacts the performance of the model on new unseen data. We can say that an overfitting machine learning model has learned the training data very well and fails on testing unseen data. On the other hand underfitting means that our machine learning model can neither learn the training data nor generalize to new unseen data [50].

4.1 Metrics Definition

On our problem, we have two different classes “Fake” and “No Fake”. We define “Fake” as positive class and “No Fake” as negative class, so can summarize our fake prediction model using a 2x2-confusion matrix that depicts our four possible outcomes:

Table 2: Fake Prediction Model

True Positive (TP): Classification of a fake news as fake	**False Positive (FP):** Classification of a no fake news as fake
False Negative (FN): Classification of a fake news as no fake	**True Negative (TN):** Classification of a no fake news as no fake

Now we can define our metrics:

- **Precision:** The portion of positive identifications that was actually correct. It is defined mathematically as follows:

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

- **Recall:** The portion of actual positives that was identified correctly. It is defined mathematically as follows:

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

- **F-measure:** Precision and recall can be combined to produce a single metric known as F-measure, which is the weighted harmonic mean of precision and recall. The main advantage of using F-measure is that we can rate a model using one unique rating instead of two. It is defined mathematically as follows:

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

- **Accuracy:** The fraction of predictions our model got right. Formally, accuracy has

the following mathematical definition:

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

- **Mean Absolute Error (MAE):** The absolute difference between the actual or true values and the values that are predicted. Absolute difference means that if the result has a negative sign, it is ignored. MAE has the following mathematical definition:

$$MAE = |\text{True values} - \text{Predicted values}|$$

- **AUC ROC curve:** A performance measurement at various thresholds settings. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting positives as positives and negatives as negatives.
- **MAE Folds curve:** Mean Average Error or MAE folds curve represends the mean average error on each fold during trainning process. For each fold we have two different MAE curves, one for the trainning error and one for the test error. As we said earlier an overfitted model has learned the training data too well, so the the model's error on the training MAE diagram will be very low but the model's error on the test MAE diagram will be high. On the other hand an underfitted model can neither learn the training data nor generalize to new unseen data, so the model's error on both train and test MAE will be high.

4.2 Hyper-parameter Tuning

A ML model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyper-parameters, and have to be tuned so that the model can optimally solve the ML problem. The optimal solution is the solution that gives the best result for a predifined metric like accuracy. Hyper-parameter optimization is the process of choosing a set of optimal hyper-parameters for a ML algorithm. It finds a tuple of hyper-parameters that yields an optimal model, which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyper-parameters and returns the associated loss.

In this thesis, we used *sklearn's GridSearchCV* method [51] to compare various combinations of hyper-parameters for each ML algorithm on a predefined grid and to choose which one of these applies to our problem [52]. As scoring metrics for the above process, we used accuracy, precision and recall. As target metric, we choose the accuracy. We also combined the *GridSearchCV* function with 10-fold split to get more accurate results. At the end of this process, we get a tuned learner along with the set of best parameters.

4.3 K-Fold Evaluation

In K Fold cross -validation, the data is divided into k subsets. Now the method is repeated k times, such that each time, one of the k subsets is used as the validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get total effectiveness of our model. As can be seen, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times. This significantly reduces bias and variance as we are using most of the data for fitting and validation. Therefore, we avoid overfitting and underfitting problems. Interchanging the training and test sets also adds to the effectiveness of this method.

The dataset was splitted into 90% training data and 10% testing data on each fold during

training process for every algorithm. We wanted to avoid imbalance problem, so we splitted our data in order each fold to contains approximately the same percentage of samples of each target class as the complete set [53]. This is called stratified cross-validation. For this thesis, we used *sklearn's StratifiedKFold* method [54] to perform 10-Fold cross validation which is an implementation of stratified cross-validation. As scoring metrics for the above process, we used accuracy, precision, recall and F-measure which is a combination of precision and recall. We also used k fold's split during this, to get train-test data pairs in order to train our tuned model process and calculate mean true/false positive rate for our AUC ROC curve an mean absolute error for MAE Folds curve. At the end of this process, we get a fully trained model along with its calculated metrics.

4.4 Models Comparison

In this section, we compare models using the metrics that we defined earlier in following two steps:

- The first step is the comparison between feature extraction techniques for every learning algorithm, to find out which technique fits better to the specific learner and to get the best model for that learner.
- The second step is the comparison between the best models that we find out on the previous step, in order to get our final model for the specific problem. We wanted the comparison to be as fair as possible, so every model is tuned with the best hyperparameters and fully trained using 10-fold method.

4.4.1 Comparison per Learner

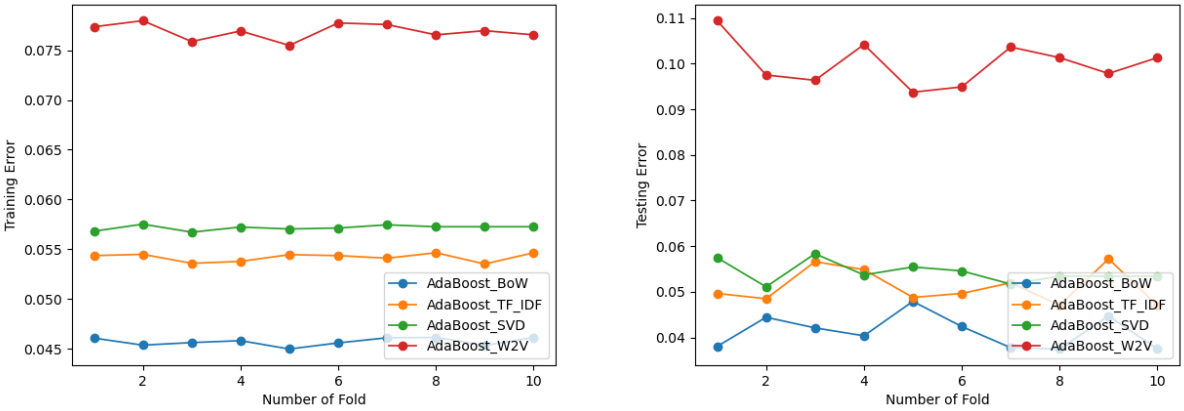
In this section, we perform comparison of feature extraction techniques for every learning algorithm, to find out which technique fits better to the specific learner and to get the best model for that learner. At the end of the section, we provide conclusions about how every feature extraction technique performs and a comparison based on our expectations.

Table 3 shows the results of every feature extraction method for AdaBoost classifier. We can easily see that Average Word Vector(W2V) gives the worst results and the other methods give almost the same results, with TF-IDF being marginally better on accuracy and precision and BoW being better on recall. TF-IDF method is also marginally better on F-measure that combine precision and recall, so we can say that TF-IDF method gives the best results.

Table 3: AdaBoost Model Results

Boosting (AdaBoost)				
	Accuracy	**Precision**	**Recall**	**F-measure**
BoW	0.927	0.908	**0.914**	0.911
TF_IDF	**0.928**	**0.914**	0.911	**0.912**
SVD	0.912	0.910	0.868	0.889
W2V	0.900	0.878	0.878	0.878

Figure 5 shows the MAE diagrams for both training and testing process. We can spot a small overfitting situation for Average Word Vector as the testing error is marginally higher in almost every fold. The other models have almost the same error on both training and testing process for every fold, so there isn't any overfitting or underfitting situation. As we can see Figure 6 shows the AUC ROC curve for every method and verifies the results of Table 3 as BoW and TF-IDF have very good curves close to perfect, with TF-IDF being marginally better with AUC score close to 0.98.



(a) Training MAE Folds Diagram (b) Testing MAE Folds Diagram

Figure 5: MAE Folds Diagrams for AdaBoost Models

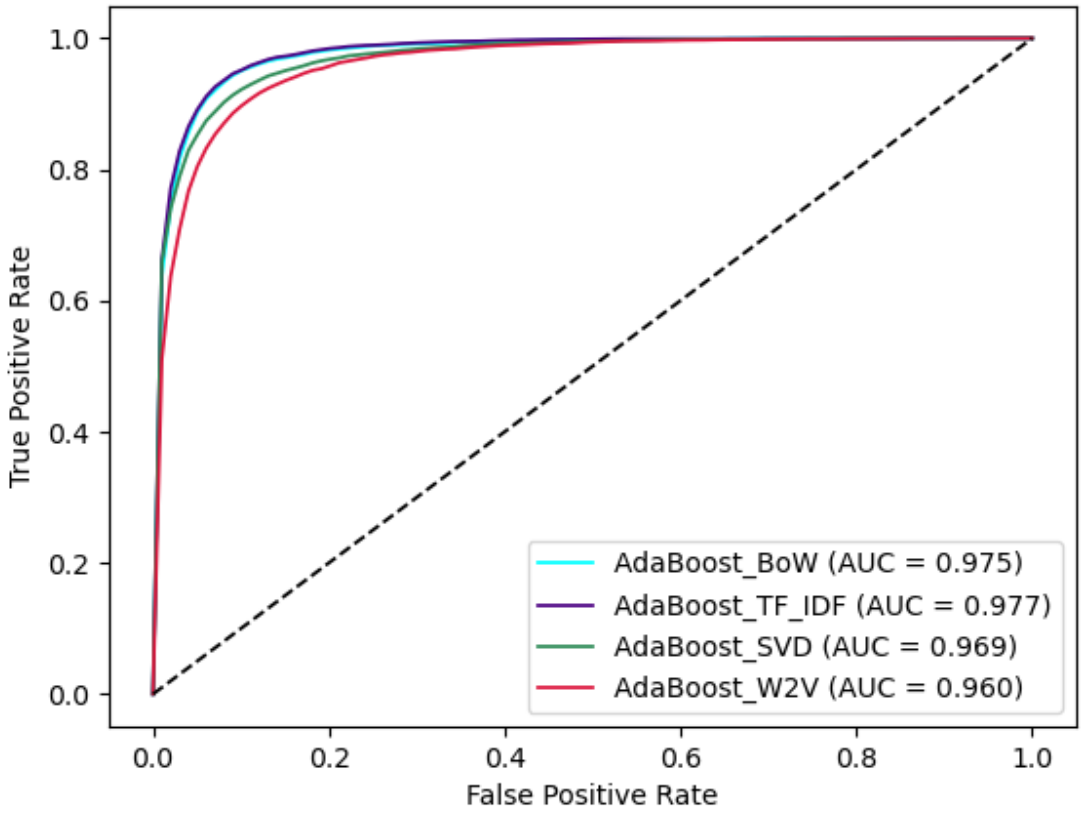


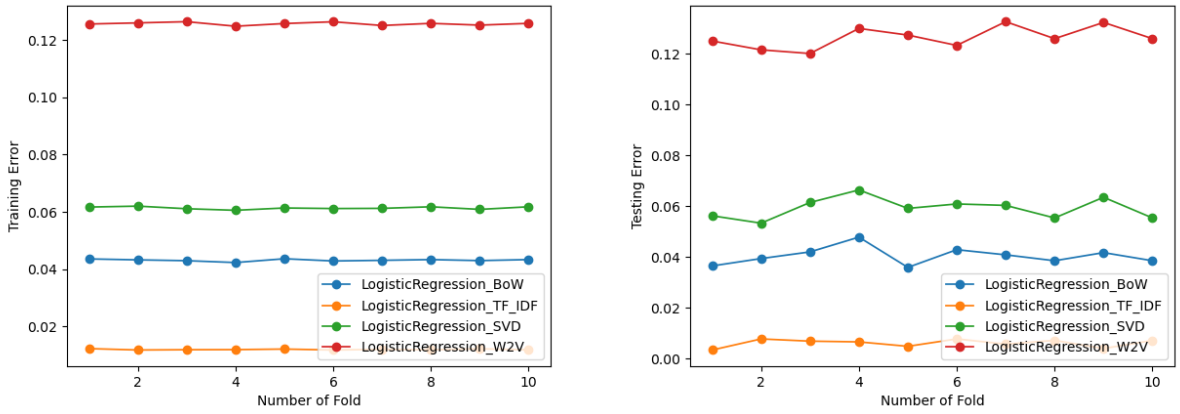
Figure 6: ROC Curve Diagram for AdaBoost Models

Table 4 shows the results of every feature extraction method for LogisticRegression classifier. We can easily see that Average Word Vector(W2V) gives the worst results and the other methods give almost the same results, with TF-IDF being marginally better on accuracy and recall and BoW being better on precision. TF-IDF method is also marginally better on F-measure that combine precision and recall, so we can say that TF-IDF method gives the best results.

Table 4: LogisticRegression Model Results

Logistic Regression (LogisticRegression)				
	Accuracy	**Precision**	**Recall**	**F-measure**
BoW	0.937	**0.934**	0.911	0.922
TF_IDF	**0.946**	0.925	**0.946**	**0.935**
SVD	0.926	0.923	0.890	0.906
W2V	0.873	0.857	0.829	0.843

Figure 7 shows the MAE diagrams for both training and testing process. We can't spot any overfitting or underfitting situation every model has the same error on both training and testing process for every fold. Figure 8 shows the AUC ROC curve for every method and verifies the results of Table 4 as BoW and TF-IDF have very good curves close to perfect, with TF-IDF being marginally better with AUC score more than 0.98.



(a) Training MAE Folds Diagram

(b) Testing MAE Folds Diagram

Figure 7: MAE Folds Diagrams for LogisticRegression Models

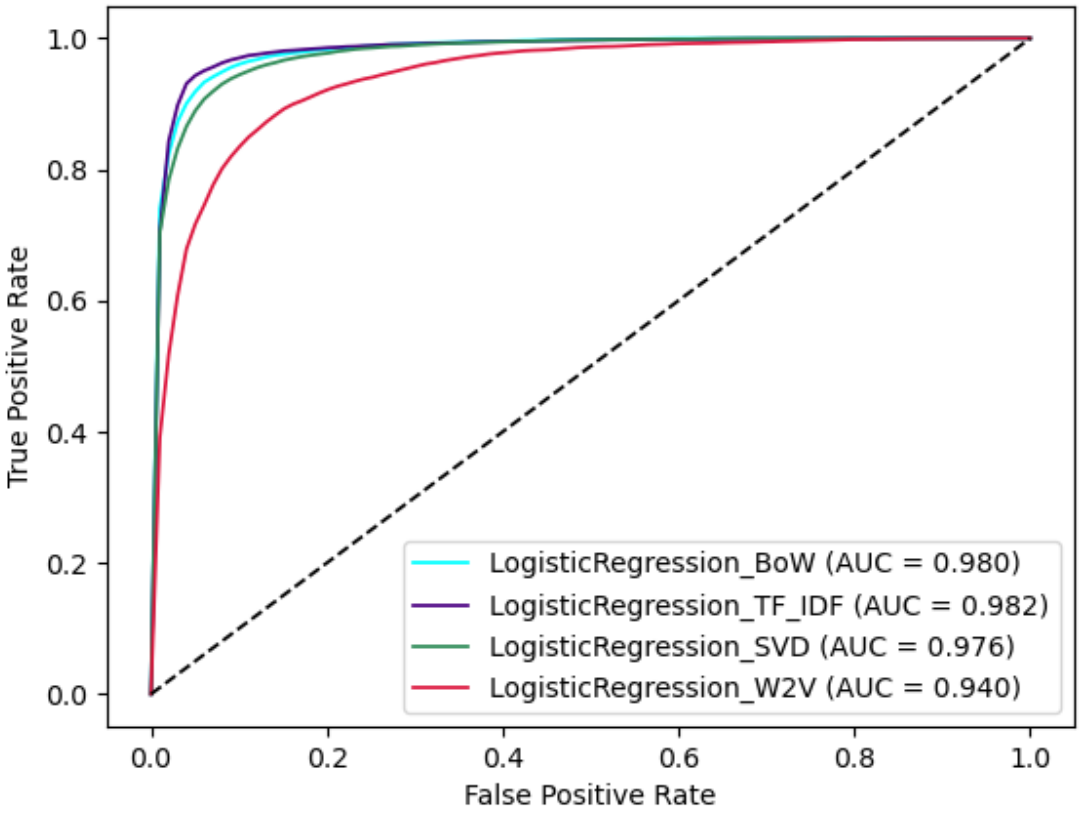


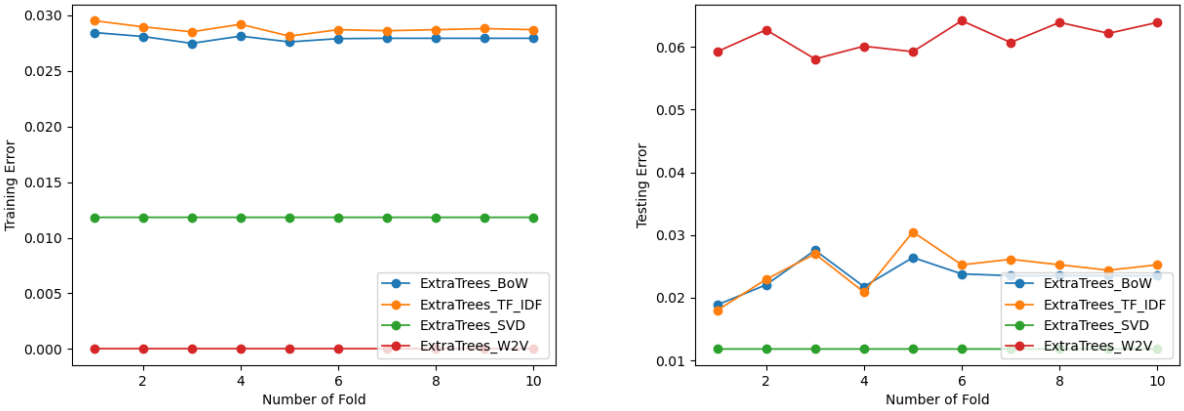
Figure 8: ROC Curve Diagram for LogisticRegression Models

Table 5 shows the results of every feature extraction method for ExtraTrees classifier. We can see that Average Word Vector(W2V) gives the best results for almost every metric, but that's not true at all because this model is overfitted as we can see at Figure 9. The other methods give almost the same results, with TF-IDF being marginally better on accuracy and recall and SVD being better on precision. TF-IDF method is also marginally better on F-measure that combine precision and recall, so we can say that TF-IDF method gives the best results.

Table 5: ExtraTrees Model Results

Decision Tree (ExtraTrees)				
	Accuracy	**Precision**	**Recall**	**F-measure**
BoW	0.936	0.968	**0.872**	0.918
TF_IDF	**0.937**	0.970	**0.872**	**0.919**
SVD	0.899	**0.999**	0.756	0.861
W2V	**0.939**	**0.963**	**0.884**	**0.922**

Figure 9 shows the MAE diagrams for both training and testing process. We can spot a huge overfitting situation for Average Word Vector as the training error is close to 0.0 for every fold and the testing error is very high in every fold. Figure 10 shows the AUC ROC curve for every method and verifies the the results of Table 5 as BoW and TF-IDF have very good curves close to perfect with AUC score more than 0.98.



(a) Training MAE Folds Diagram (b) Testing MAE Folds Diagram

Figure 9: MAE Folds Diagrams for ExtraTrees Models

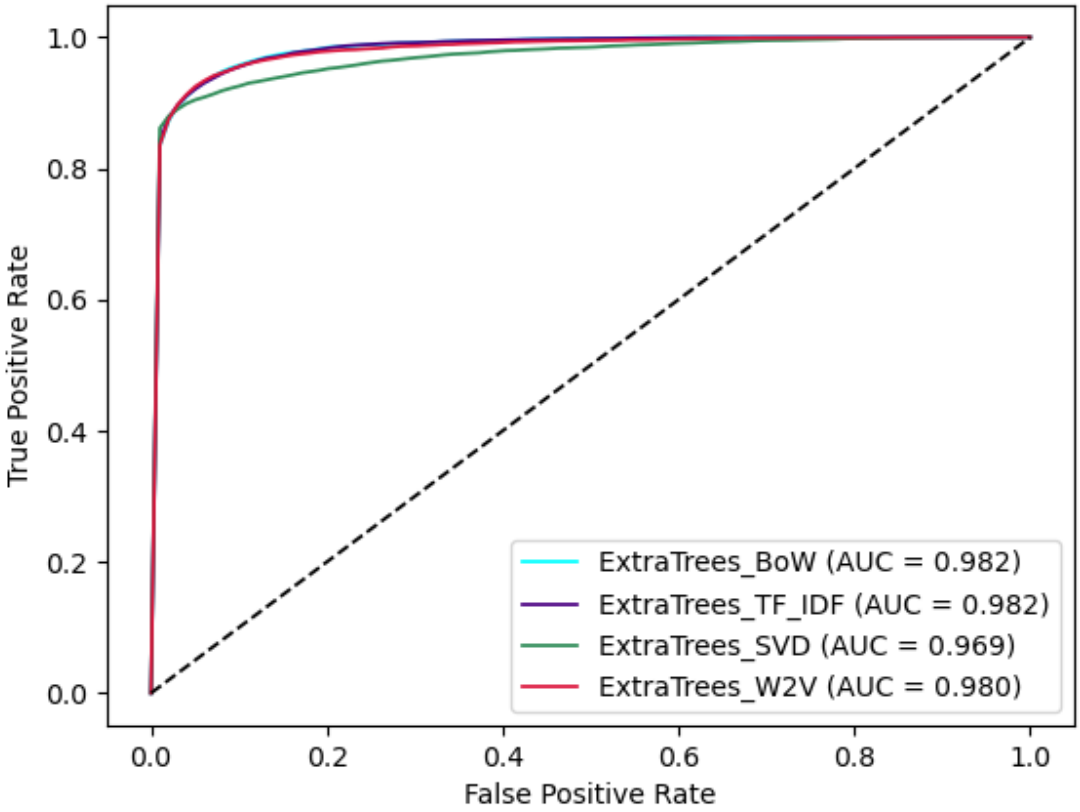


Figure 10: ROC Curve Diagram for ExtraTrees Models

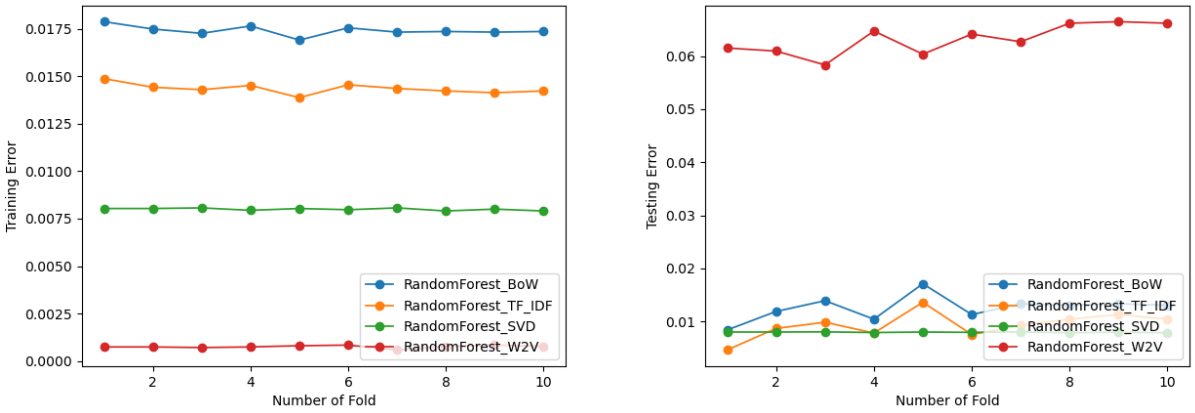
Table 6 shows the results of every feature extraction method for RandomForest classifier. We can see that Average Word Vector(W2V) gives the best results for almost every metric, but that's not true at all because this model is overfitted as we can see at Figure 11. The other methods give almost the same results, with TF-IDF being marginally better on accuracy and recall and SVD being better on precision. TF-IDF method is also marginally

better on F-measure that combine precision and recall, so we can say that TF-IDF method gives the best results.

Table 6: RandomForest Model Results

Random Forests (RandomForest)				
	Accuracy	**Precision**	**Recall**	**F-measure**
BoW	0.939	0.945	0.906	**0.925**
TF_IDF	**0.940**	0.943	**0.908**	**0.925**
SVD	0.934	**0.986**	0.851	0.913
W2V	**0.937**	**0.946**	**0.898**	**0.921**

Figure 11 shows the MAE diagrams for both training and testing process. We can spot a huge overfitting situation for Average Word Vector as the training error is close to 0.0 for every fold and the testing error is very high in every fold. Figure 12 shows the AUC ROC curve for every method and verifies the the results of Table 6 as BoW, TF-IDF and SVD have very good curves close to perfect with AUC score close to 0.98.



(a) Training MAE Folds Diagram

(b) Testing MAE Folds Diagram

Figure 11: MAE Folds Diagrams for RandomForest Models

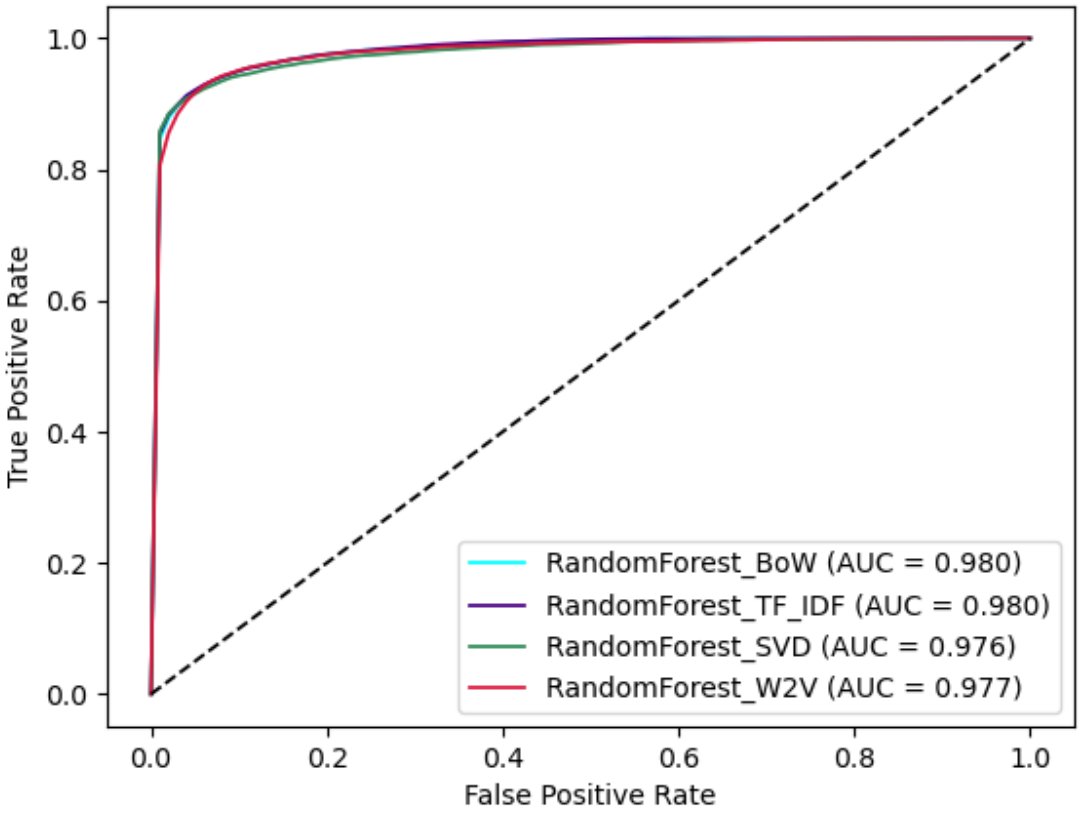


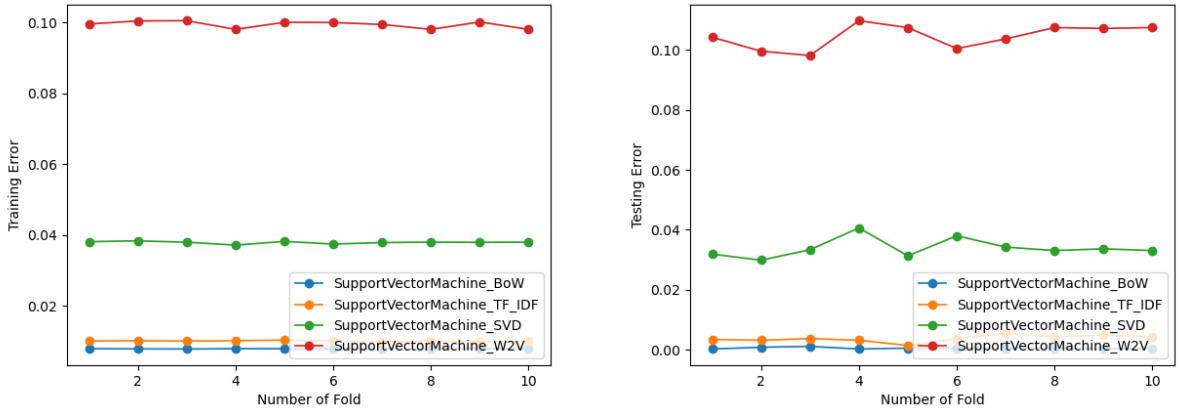
Figure 12: ROC Curve Diagram for RandomForest Models

Table 7 shows the results of every feature extraction method for SupportVectorMachine classifier. We can easily see that Average Word Vector(W2V) gives the worst results and the other methods give almost the same results, with TF-IDF being marginally better on accuracy and precision and BoW being better on precision. TF-IDF method is also marginally better on F-measure that combine precision and recall, so we can say that TF-IDF method gives the best results.

Table 7: SupportVectorMachine Model Results

Support Vector Machine (SupportVectorMachine)				
	Accuracy	**Precision**	**Recall**	**F-measure**
BoW	0.933	0.897	**0.946**	0.921
TF_IDF	**0.942**	**0.916**	0.945	**0.930**
SVD	0.933	**0.916**	0.921	0.919
W2V	0.896	0.878	0.866	0.872

Figure 13 shows the MAE diagrams for both training and testing process. We can't spot any overfitting or underfitting situation every model has the same error on both training and testing process for every fold. Figure 14 shows the AUC ROC curve for every method and verifies the the the results of Table 7 as BoW, TF-IDF and SVD have very good curves close to perfect with AUC score close to 0.98.



(a) Training MAE Folds Diagram (b) Testing MAE Folds Diagram

Figure 13: MAE Folds Diagrams for SupportVectorMachine Models

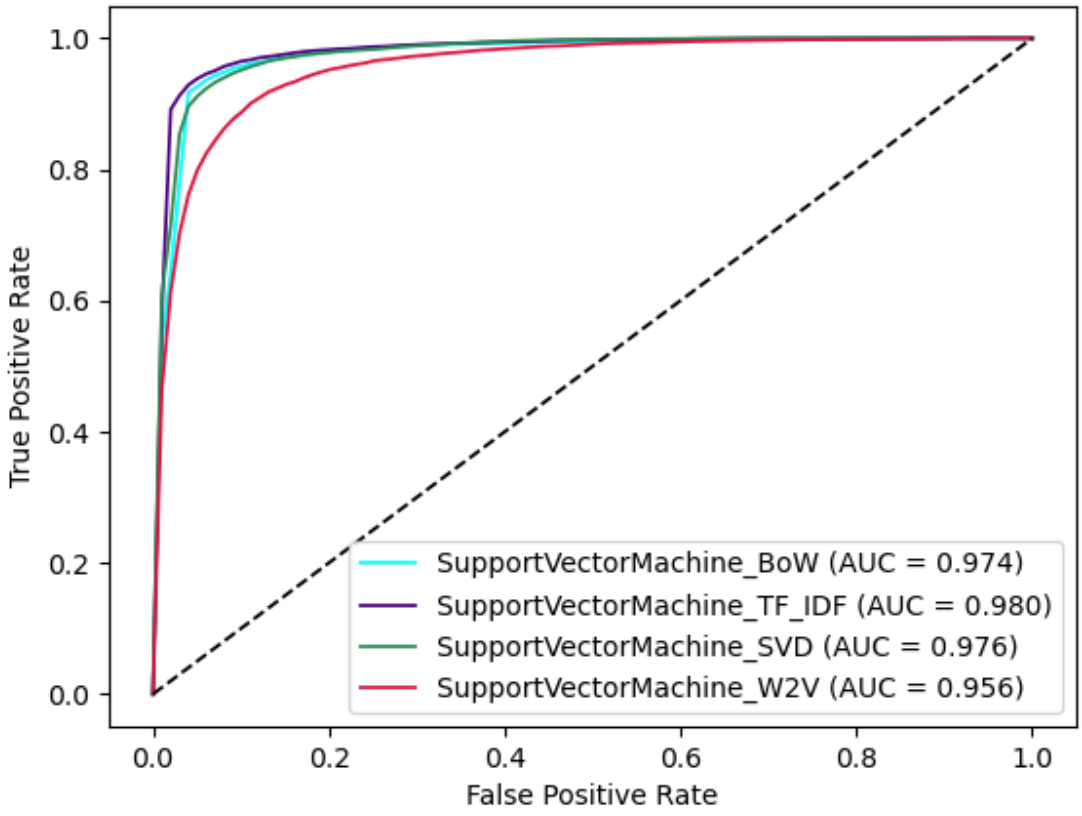


Figure 14: ROC Curve Diagram for SupportVectorMachine Models

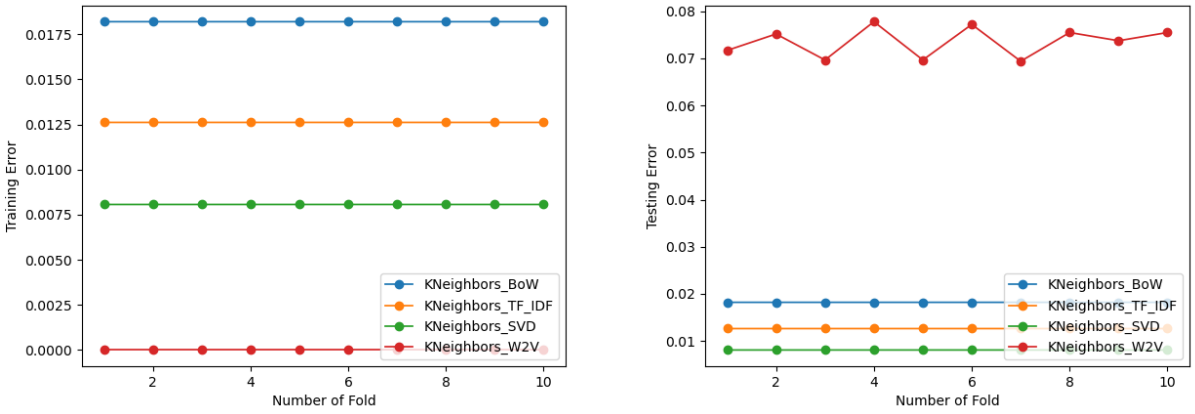
Table 8 shows the results of every feature extraction method for KNeighbors classifier. We can see that Average Word Vector(W2V) gives the best results for almost every metric, but that's not true at all because this model is overfitted as we can see at Figure 15. We can also see that SVD is clearly better on all metrics. The performance of KNN is dependent on finding a meaningful distance function, thus making this technique a very dataset dependent algorithm [55] , so we expected a technique like SVD to perform very well because it

expands the feature space and produces more accurate word vector representations than BoW and TF-IDF [56].

Table 8: KNeighbors Model Results

K-Nearest Neighbors (KNeighbors)				
	Accuracy	**Precision**	**Recall**	**F-measure**
BoW	0.834	0.855	0.880	0.813
TF_IDF	0.888	0.844	0.890	0.867
SVD	0.923	0.915	0.898	0.906
W2V	**0.926**	**0.892**	**0.933**	**0.912**

Figure 15 shows the MAE diagrams for both training and testing process. We can spot a huge overfitting situation for Average Word Vector as the training error is close to 0.0 for every fold and the testing error is very high in every fold. Figure 16 shows the AUC ROC curve for every method and verifies the the results of Table 8 as SVD has very good curve with AUC score close to 0.97.



(a) Training MAE Folds Diagram

(b) Testing MAE Folds Diagram

Figure 15: MAE Folds Diagrams for KNeighbors Models

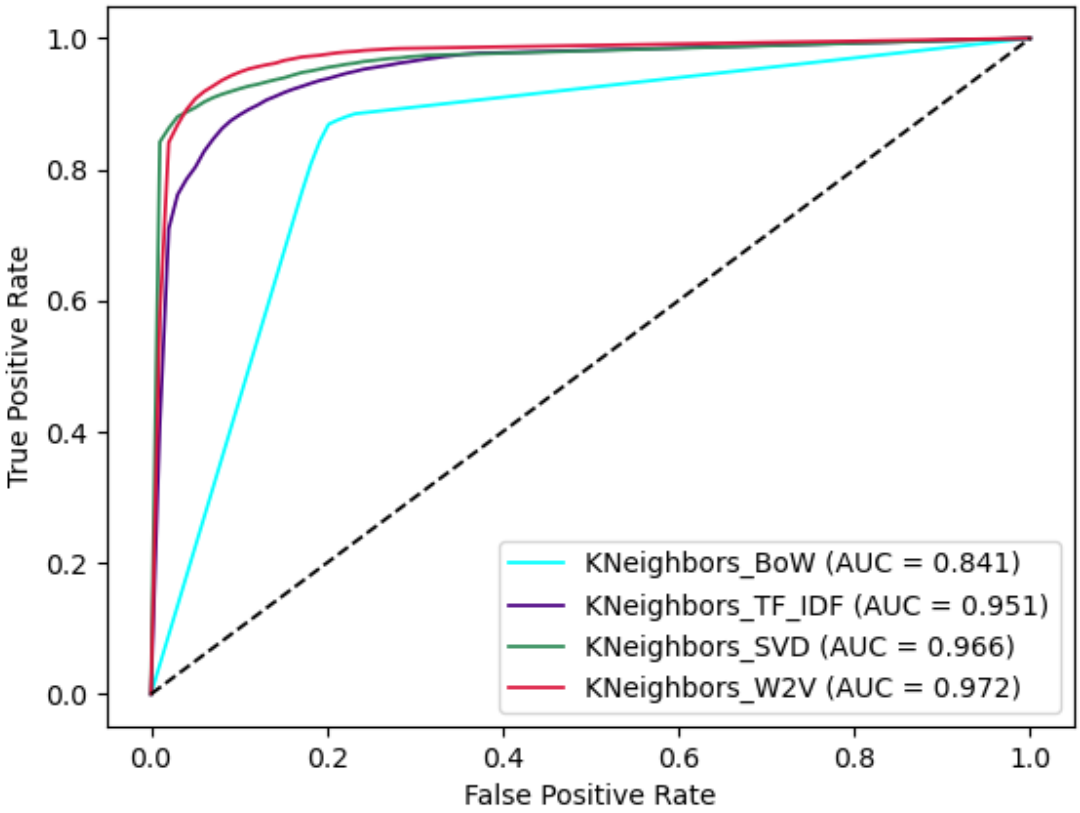


Figure 16: ROC Curve Diagram for KNeighbors Models

From the previous comparisons, we can see that for every learner we get the poorest results by using the Average Word Vector feature extraction technique. The cases that this technique gave us the best results were overifitting situtations, so the model in every case had learned the training data too well and failed to generalize to new unseen data. When we started our research, we believed that this method could give the best results for almost every algorithm, because it overcomes the semantic and syntactic limitation off other methods, but that is the half true. This method surely tries to overcome those two limitations, but it is based on Word2Vec method that so it requires a large corpus to pre-train this model [57]. On our situation, the corpus was limited. Also as a word embedding method, is ideal for problems involving a single word, such as translation problems. Based on the above, we realized why we did not get the expected results.

In addition, we can observe that every algorithm works fine with BoW, TF-IDF and SVD methods with very similar results and TF-IDF being marginally better in almost every case. When we started our research, we believed that the SVD method would perform better than the other two methods because it expand the feature space, preserves the semantic relationship between words and produces more accurate word vector representations than BoW and TF-IDF [56]. We still believe that is true, because for memory reasons we only used the 20% of the available features, we get very similar, and in some cases better results with the previous methods that used all the available features. We would like to explore how much better could this method perform by using a larger amount of the available features but it requires huge amount of memory to store the co-occurrence matrix.

4.4.2 Overall Comparison

In this section, we perform comparison between the best models that we find out on the previous section, in order to get our final model for the specific problem. We also provide our conlcusive remarks on how every model performs and a comparison.

Table 9: Overall Results

Overall Model Comparison				
	Accuracy	**Precision**	**Recall**	**F-measure**
AdaBoost_TF_IDF	0.928	0.914	0.911	0.912
LogisticRegression_TF_IDF	**0.946**	0.925	**0.946**	**0.935**
ExtraTrees_TF_IDF	0.937	**0.970**	0.872	0.919
RandomForest_TF_IDF	0.940	0.943	0.908	0.925
SupportVectorMachine_TF_IDF	0.942	0.916	0.945	0.930
KNeighbors_SVD	0.923	0.915	0.898	0.906

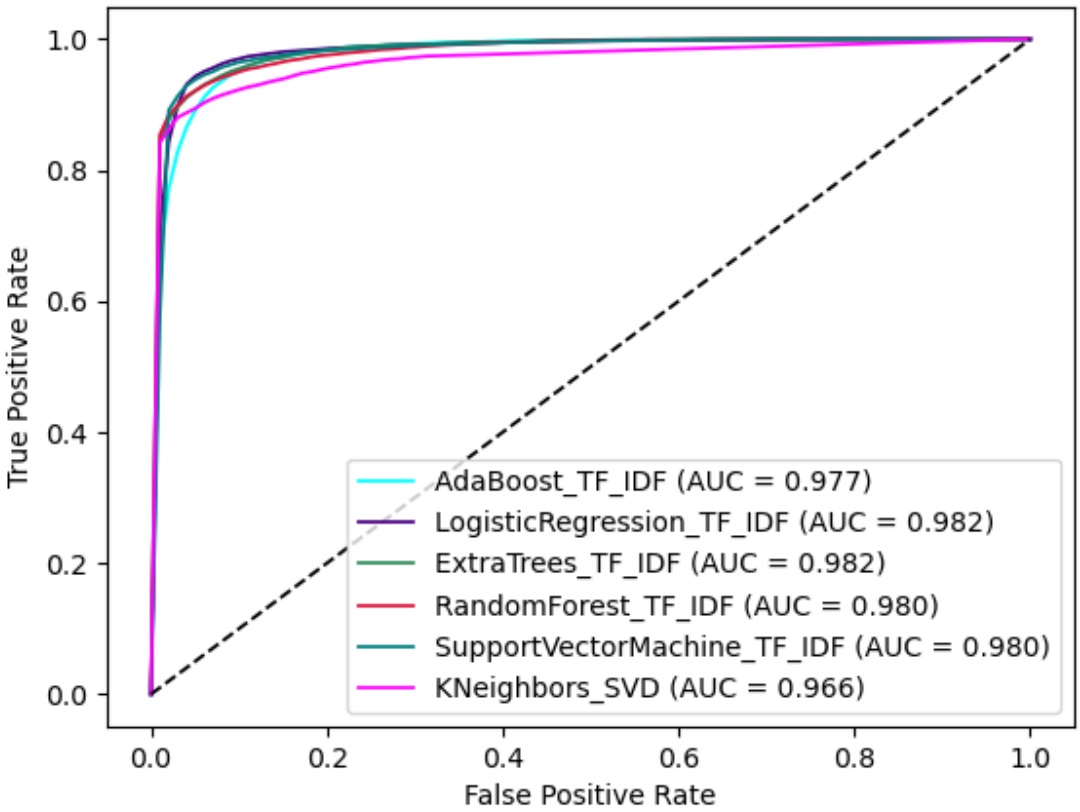


Figure 17: ROC Curve Diagram for Best Models

Table 9 shows results of the best models for every algorithm. We can easily see that TF-IDF works perfect for every classification algorithm with accuracy from 0.928 to 0.946, precision from 0.914 to 0.970, recall from 0.872 to 0.946 and F-measure from 0.912 to 0.935. The only exception is the KNN classifier where it seems to work better using non-sparse features of SVD method, because is a dataset dependent algorithm. Figure 17 shows the AUC ROC curve for every method and verifies the above as all models that

use TF-IDF as feature extraction technique have very good curve with score close to 0.98 and KNN-SVD combination has a curve with AUC score close to 0.97.

We can divide our models in three different groups based on their performance. In particular, we can see that LogisticRegression classifier and SVM have very good and similar performance for every metric and we can add them to the first group. We expected LogisticRegression's good performance because as a method, for linear problems, it does not require input features to be scaled or any hyper-parameter tuning [58]. On the other hand SVM performs similar with LogisticRegression for linear problems [59].

On the second group, we can add ExtraTrees and RandomForest with almost same performance on accuracy and precision. ExtraTrees has worst performance on recall metric due to its sensitivity to small perturbations in the data [60]. We We expected RandomForest to better than ExtraTrees because, as we said in section 3 RFC is an extension of DTCs as ExtraTrees [41].

On the third group we have AdaBoost and KNN. AdaBoost as a boosting algorithm has many limitations and disadvantages, such as the computational complexity and loss of interpretability [61]. On the other hand the performance of KNN is dependent on finding a meaningful distance function, thus making this technique a very dataset dependent algorithm [55]. Therefore, we believe that the small lack on their performance is due to the above limitations.

Finally we had to chose a model, in order to use it as the core module to our browser extension. Our main goal was to find a stable, accurate model to predict Greek fake news in few milliseconds and without user discernible understanding of the service (real-time). For this purpose we choose the combination of Logistic Regression algorithm along with TF-IDF feature extraction method because as we can see on Table 9 it gives the best accuracy (0.946), the best F-measure (0.935) and the best ROC AUC score(0.982). Those mean that the model is very accurated on its predictions and also capable of distinguishing between classes.

Figure 7 shows that the previous metrics are not a result of overfitting, as the model which combines LogisticRegression with TF-IDF gives the same error on both training and testing process for every fold. Also models like the selected one which are based on LogisticRegression algorithm are easy to retrain because they don't require input features to be scaled or any hyper-parameter tuning [58]. In addition, we checked the response time of the selected model by using Python's *time()* function and we outlined that it can predict if a Greek article is fake or not in ~30 ms. All the above mean that the selected model is accurate and can generalize to new unseen data. Also the selected model is fast enough on its predictions, as a user does not have to wait for a prediction, and it can be retrained with new data any time we want which is very important on a production environment.

The overall implementation of our evaluation process is available here: fake-news-detection-evaluation

5. RESTFUL A.P.I.

As we said earlier, the purpose of this this is to create an application that detects fake news in few milliseconds and without user discernible understanding of the service (real-time). Following all previous steps we outlined, we created a set of stable and accurate ML models that given an article's text and title can predict the probability this article to be fake.

Therefore, the next step to our target was to pick up the best model based on our evaluation process that we described earlier, and use it under real-time conditions. In addition, we wanted to separate our backend implementation from various clients, in order to make it modular and scalable compared to traditional monolithic approach. To achieve that we thought that a good idea was to create a RESTFUL API based on our chosen model. We choose to use REST because is the most logical, efficient and widespread standard in the creationof APIs for Internet services.

For this thesis, we used Python's *flask* library in order to create our REST API. We choose *flask* [62] because it provides all the tools to create a lightweight, fast, secure and scalable application, such as handling off HTTP requests, JSON responses etc. It also provides lightweight Python server that can easily been deployed on every environment.

We implemented two different http POST methods on our API scraper and predict.

- **predict:** The first method that gets the text and the title of an article from request's JSON body and calls our model, in order to predict the probability the given article to be fake. It returns to client a JSON with the predicted probability on the body or an error response if something goes wrong during prediction.
- **scraper:** The second method that gets from body a URL, using the same way as previous and tries to scrap an article from the given URL's webpage. If this page contains an article, the method returns to client a JSON with the text and the title of this article on the body, otherwise it returns an error response. For article's scraping from URL, we used Python's newspaper library on the same way as we used it during dataset's creation process.

Our application gets as parameter only the name of the chosen ML algorithm and the name of the feature extraction method. Once server starts, the application loads the trained model that we saved previously on a pickle file. If the file does not exist, the training process that we described earlier is executed in order to get our trained model. After those steps, we are ready to serve requests from every client. We also checked the response time of our REST API by using *Apache Jmeter* [63] and we outlined that the whole prediction process takes ~35 ms.

The implementation of our Restful A.P.I. is available here: fake-news-detection-rest-api

6. BROWSER EXTENSION

The next and final step to our target was to use our API from a client, under real-time conditions. As we said earlier, fake news has been very popular in recent years because of the increasing popularity and use of social media, were that kind of news can be spread out in a short amount of time. Also on the internet, anyone can find a plethora of websites that contain articles of doubtful quality from untrustworthy sources. Therefore, we thought that we should handle that kind of news during browsing session in order to warn users before they read one of those articles. For that purpose, we implemented a web browser extension that runs during browsing session and can has access to web page's metadata. We used React programming language for our extension, because is simple, easy to learn and can be easily combined with google's libraries for web extensions in order to create a lightweight and fast extension.

React is based on components. Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. They are also like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen. On the other hand, a browser extension is made of different, but cohesive, elements. These elements can include background scripts, content scripts, an options page, UI elements and various logic files.

On our extension, we implemented only one React component that works as UI page and as content script. This component displays results and various other information to users as we can see in Figure 18. It also prepares and sends some messages to the background script, in order to get results and display them, so we can say that this component does only jobs that change the DOM.

Figure 18: Web Browser Extension

On the other hand, the background script does the entire dirty job. It receives the messages from React component (UI), communicates with the REST API by creating the two REST requests that we analyzed earlier, gets the response and returns the results to the React component. If the wanted results for an article are available from previous execution the background script simple returns those results without making any REST call to the API. In addition, this script can handle various browser events, like tab change or updated and has access to various metadata, so we implement many event handlers that use these metadata, in order to get and save results for the active tab before the mounding of React component.

By using the previous mechanism, we were on position to collect and save all useful data from the browser and the REST API, on tab loading, so we had only to load them during React component's mount or to reload them if the user change page to current tab. As you can see on our background script, we implemented a custom message API for the communication between the UI and the script and a custom cache that keeps the results and other useful data from current page, in order to minimize the communication between the client and the REST API and to make our client faster and more efficient.

The implementation of our browser extension is available here: fake-news-detection-client

7. CONCLUSION

Fake news detection becomes a critical and challenging problem nowadays, so we created a web browser extension that predicts the probability of a Greek article to be fake using machine learning in few milliseconds (~35 ms) and without user discernible understanding of the service (real-time). The browser extension is implemented as a React web client that calls a REST API based on a ML model. To create our API we had to collect and preprocess our data and to compare various machine learning algorithms along with feature extraction techniques, in order to find out which combinations fits better to our problem.

During our evaluation process, we found out that simple word frequency techniques like TF-IDF outperform the more complicated word emending like Word2Vec for smaller amount of data. That was unexpected at the start of our research because the Greek language is very complex and full of semantic and syntactic meaning and that kind of methods have some limitation to process this kind of language. We also confirmed that traditional classification algorithms performs very good with the availablel amount of data. Finally we confirmed the strong and weak points of every classification algorithm and we find out the simple and non data depended Logistic Regression algorithm, along with TF-IDF feature extraction method fits better to our problem.

The evaluation results for the selected model were quite encouraging considering the amount of available data and showed that our extension can predict with great accuracy (~95%) if an article is fake or not. Moreover, our model is also stable and capable of distinguishing between classes as it scores great precission (~93%), recall (~95%) and ROC AUC score (~98.2%). Therefore the selected model is very fast as it can predict if an article is fake or not in ~35 ms and easy to retrain.

There are several open issues for improvement and future research. Unfortunately, they have not been developed mechanisms for automatic retrain of the selected model with new data. In addition after various tests on semi-production environment, we noticed that the rules that we use to decide if a web page contains an article or not, are naive and need to be improved. Also we would like to explore some deep learning approaches like various deep neural networks and see how we can use them in order to solve our problem.

TABLE OF TERMINOLOGY

Browser	Φυλλομετρητής
Extension	Επέκταση
Dataset	Σύνολο Δεδομένων
Retrain	Επανεκπαίδευση

ABBREVIATIONS - ACRONYMS

ML	Machine Learning
REST	Representational State Transfer
A.P.I.	Application Programming Interface
U.I.	User Interface
BOW	Bag of Words
TF-IDF	Term Frequency – Inverse Document Frequency
SVD	Singular Value Decomposition
DTC	Decision Tree Classifier
RF	Random Forests
RFC	Random Forests Classifier
SVM	Support Vector Machine
SVC	Support Vector Classifier
ID	Identifier
HTML	Hypertext Markup Language
WORD2VEC	Word to Vector
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
AUC	Area Under the Curve
ROC	Receiver Operating Characteristics
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
DOM	Document Object Model

REFERENCES

[1] Hunt Allcott and Matthew Gentzkow. 2017. Social Media and Fake News in the 2016 Election. Journal of Economic Perspectives 31, 2 2017, 211-236.

[2] Niall J. Conroy, Victoria L. Rubin, and Yimin Chen. 2015. Automatic deception detection: Methods for finding fake news. In Proceedings of the 78th ASIS & T Annual Meeting: Information Science with Impact: Research in and for the Community,51,1(2015),St. Louis, MO, USA, 1-4.

[3] Supanya Aphiwongsophon and Prabhas Chongstitvatana. 2018. Detecting Fake News with Machine Learning Method. 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Chiang Rai, Thailand, 528-531.

[4] “A Guide to Crawler4j” [Online]. Available: https://www.baeldung.com/crawler4j. [Accessed: Jun 6, 2020].

[5] “Newspaper3k: Article scraping & curation” [Online]. Available: https://newspaper.readthedocs.io/en/latest/. [Accessed: Jul 17, 2020].

[6] “Beautiful Soup Documentation” [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed: Jul 17, 2020].

[7] “Pandas ” [Online]. Available: https://pandas.pydata.org/. [Accessed: Mar 20, 2020].

[8] Kowsari, K.; Jafari Meimandi, K.; Heidarysafa, M.; Mendu, S.; Barnes, L.; Brown, D. Text Classification Algorithms: A Survey. Information 2019, 10, 150.

[9] “A Gentle Introduction to the Bag-of-Words Model” [Online]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/. [Accessed: Mar 20, 2020].

[10] “TF-IDF Simplified” [Online]. Available: https://towardsdatascience.com/tf-idf-simplified-aba19d5f5530. [Accessed: Mar 20, 2020].

[11] “Understanding Singular Value Decomposition and its Application in Data Science” [Online]. Available: https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d. [Accessed: Mar 21, 2020].

[12] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space.arXiv2013, arXiv:1301.3781.62.

[13] Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrasesand their compositionality.Adv. Neural Inf. Process. Syst.2013,26, 3111–3119.

[14] Schapire, R.E. The strength of weak learnability. Mach. Learn.1990,5, 197–227.

[15] Cox, D.R. Analysis of Binary Data; Routledge: London, UK, 2018.

[16] Safavian, S.R.; Landgrebe, D. A survey of decision tree classifier methodology.IEEE Trans. Syst. Man Cybern.1991,21, 660–674.

[17] Ho, T.K. Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada 14–16 August 1995; Volume 1, pp. 278–282.

[18] Vapnik, V.; Chervonenkis, A.Y. A class of algorithms for pattern recognition learning. Avtomat. Telemekh1964,25, 937–945.

[19] Estimation by the Nearest Neighbor Rule. IEEE Transactions on Information Theory 14:50–55.

[20] “sklearn Pipeline” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html. [Accessed: Apr 18, 2020].

[21] Pahwa, B.; Taruna, S.; Kasliwal, N. Sentiment Analysis-Strategy for Text Pre-Processing. Int. J. Comput. Appl.2018,180, 15–18.

[22] Saif, H.; Fernández, M.; He, Y.; Alani, H. On stopwords, filtering and data sparsity for sentiment analysis of twitter. In Proceedings of the Ninth International Conference on Language Resources and Evaluation(LREC 2014), Reykjavik, Iceland, 26–31 May 2014.

[23] “sklearn CountVectorizer” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. [Accessed: Apr 17, 2020].

[24] Tokunaga, T.; Makoto, I. Text categorization based on weighted inverse document frequency. Inf. Process. Soc. Jpn. SIGNL1994,94, 33–40.

[25] “sklearn TfidfTransformer” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html. [Accessed: Apr 17, 2020].

[26] Cao, L.; Chua, K.S.; Chong, W.; Lee, H.; Gu, Q. A comparison of PCA, KPCA and ICA for dimensionalityreduction in support vector machine. Neurocomputing20 03, 55, 321–336.

[27] “Co-occurrence matrix & Singular Value Decomposition(SVD)” [Online]. Available: https://medium.com/@apargarg99/co-occurrence-matrix-singular-value-decomposition-svd-31b3d3deb305. [Accessed: Oct 22, 2020].

[28] “sklearn TruncatedSVD” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html. [Accessed: Apr 17, 2020].

[29] Goldberg, Y.; Levy, O. Word2vec explained: Deriving mikolov et al.’s negative-sampling word-embeddingmethod.arXiv2014, arXiv:1402.3722

[30] “Word2vec embeddings” [Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html. [Accessed: Mar 21, 2020].
[31] “sklearn AdaBoostClassifier” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html. [Accessed: Apr 27, 2020].
[32] Bloehdorn, S.; Hotho, A. Boosting for text classification with semantic features. In International Workshop on Knowledge Discovery on the Web; Springer: Berlin/Heidelberg, Germany, 2004; pp. 149–166.
[33] Fan, R.E.; Chang, K.W.; Hsieh, C.J.; Wang, X.R.; Lin, C.J. LIBLINEAR: A library for large linear classification. J. Mach. Learn. Res.2008,9, 1871–1874.
[34] “sklearn LogisticRegression” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed: Apr 27, 2020].
[35] Lee S, Lee H, Abbeel P, Ng Andrew (2006) Efficient L1 Regularized Logistic Regression.Proceedingsof the 21st National Conference on Artificial Intelligence 21
[36] Morgan, J.N.; Sonquist, J.A. Problems in the analysis of survey data, and a proposal. J. Am. Stat. Assoc.1963,58, 415–434.
[37] Quinlan, J.R. Induction of decision trees. Mach. Learn.1986,1, 81–106.
[38] De Mántaras, R.L. A distance-based attribute selection measure for decision tree induction.Mach. Learn.1991,6, 81–92.
[39] “sklearn ExtraTreesClassifier” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html. [Accessed: Apr 25, 2020].
[40] Breiman, L. Random Forests; UC Berkeley TR567; University of California: Berkeley, CA, USA, 1999.
[41] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning (2nd ed.) 587-589
[42] “sklearn RandomForestClassifier” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: Apr 25, 2020].
[43] G. Bo and H. Xianwu, ”SVM multi-class classification” Journal of Data Acquisition & Processing, vol. 21, pp. 334-339, 2006
[44] Boser, B.E.; Guyon, I.M.; Vapnik, V.N. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, PA, USA, 27–29 July 1992;pp. 144–152.
[45] “sklearn SVC” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: Apr 26, 2020].
[46] Jiang, S.; Pang, G.; Wu, M.; Kuang, L. An improved K-nearest-neighbor algorithm for text categorization. Expert Syst. Appl.2012,39, 1503–1509.
[47] “sklearn KNeighborsClassifier” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed: Apr 26, 2020].
[48] Hanley, J.A.; McNeil, B.J. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology1982,143, 29–36.
[49] Lever, J.; Krzywinski, M.; Altman, N. Points of significance: Classification evaluation. Nat. Methods 2016,13,603–604.
[50] “Is your model overfitting? Or maybe underfitting? An example using a neural network” [Online]. Available: https://towardsdatascience.com/is-your-model-overfitting-or-maybe-underfitting-an-example-using-a-neural-network-in-python-4faf155398d2. [Accessed: Dec 5, 2020].
[51] “sklearn GridSearchCV” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed: May 18, 2020].
[52] Fabian Pedregosa, Ga¨el Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.
[53] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI’95,Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.
[54] “sklearn StratifiedKFold” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html. [Accessed: May 10, 2020].
[55] Sahgal, D.; Parida, M. Object Recognition Using Gabor Wavelet Features with Various Classification Techniques. In Proceedings of the Third International Conference on Soft Computing for Problem Solving; Springer: Berlin/Heidelberg, Germany, 2014; pp. 793–804.
[56] Ramiro H. Gálvez, Agustín Gravano. ”Assessing the usefulness of online message board mining in automatic stock prediction systems”. Journal of Computational Science, March 2017, 19:1877–7503.

[57] Altszyler E.,Sigman M., Slezak DF. Comparative study of LSA vs Word2vec embeddings in small corpora: a case study in dreams database. arXivpreprint, arXiv:1610.01520;2016
[58] Genkin, A.; Lewis, D.D.; Madigan, D. Large-scale Bayesian logistic regression for text categorization.Technometrics 2007,49, 291–304.
[59] Chen, K.; Zhang, Z.; Long, J.; Zhang, H. Turning from TF-IDF to TF-IGM for term weighting in text classification.Expert Syst. Appl.2016,66, 245–260.
[60] Giovanelli, C.; Liu, X.; Sierla, S.; Vyatkin, V.; Ichise, R. Towards an aggregator that exploits big data to bid on frequency containment reserve market. In Proceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2017), Beijing, China, 29 October–1 November 2017; pp. 7514–7519.
[61] Geurts, P. Some enhancements of decision tree bagging. In European Conference on Principles of Data Mining and Knowledge Discovery; Springer: Berlin/Heidelberg, Germany, 2000; pp. 136–147.
[62] “Flask web development one drop at a time” [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/. [Accessed: Aug 30, 2020].
[63] “Apache JMeter” [Online]. Available: https://jmeter.apache.org/. [Accessed: Sep 21, 2020].