TEXT CLASSIFICATION WITH DIMENSION REDUCTION ON NEWS ARTICLES

NG KANG WEI

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

TEXT CLASSIFICATION WITH DIMENSION REDUCTION ON NEWS ARTICLES

NG KANG WEI

RESEARCH PROJECT SUBMITTED TO THE FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY UNIVERSITY OF MALAYA, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF DATA SCIENCE

2019

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: (I.C./Passport No.:)

Registration/Matric No.:

Name of Degree:

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Field of Study:

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:
Designation:

TEXT CLASSIFICATION WITH DIMENSION REDUCTION ON NEWS ARTICLES

ABSTRACT

Text classification is one of the main tasks in Natural Language Processing (NLP). Text classification assigns text into pre-defined categories. The common document representation method used to extract features from text usually produce a large and sparse matrix. The high dimensionality of the matrix would be a hindrance to the accuracy of the classification models. Thus, dimension reduction algorithm is applied to reduce the dimension of these large and sparse matrix. This study analyse effect of dimension reduction on the matrix from different document representation algorithm and subsequently the performance of the classification models trained with the original large and sparse matrix and the performance of the classification models trained with the resulting matrix from different dimension reduction algorithms. The performance of the classification models in these different scenarios are evaluated, in order to identify the optimized dimension representation algorithm, dimension reduction algorithm and classification model.

Keywords: Natural Language Processing (NLP), text classification, dimension reduction, classification models, accuracy, term frequency, Term Frequency-Inverse Document Frequency (TF-IDF), k-Nearest Neighbours (kNN), Support Vector Machine (SVM), Neural Network (NN), truncated Single Value Decomposition (SVD).

ABSTRAK

Pengklasifikasian teks adalah salah satu tugas utama dalam pemprosesan bahasa semulajadi (NLP). Klasifikasi teks mengklasifikasi teks ke dalam beberapa kategori. Kaedah perwakilan dokumen yang digunakan untuk mengekstrak ciri dari teks biasanya menghasilkan matriks yang besar dan jarang. Dimensi matriks yang besar adalah halangan kepada ketepatan model klasifikasi. Oleh itu, algoritma pengurangan dimensi akan digunakan untuk mengurangkan dimensi matriks yang besar dan jarang ini. Kajian ini menganalisis kesan pengurangan dimensi pada matriks dari algoritma perwakilan dokumen yang berlainan dan seterusnya prestasi model klasifikasi yang dilatih dari matriks besar dan jarang asal dan matriks yang terhasil daripada algoritma pengurangan dimensi yang berbeza. Prestasi model klasifikasi dinilai untuk mengenal pasti algoritma perwakilan dimensi, algoritma pengurangan dimensi dan model klasifikasi yang optimum.

ACKNOWLEDGEMENTS

I would like to take this opportunity to appreciate the helps and guidance given freely by my supervisor, Dr Hoo. Without his guidance and encouragement, this study would not has seen the light of day.

I am thankful and indebted to the lecturers who persevered and pour their hearts out to educate fellow students like me. It is through their hardwork and sacrifices that I have gain knowledges that are crucial to this work.

I am also grateful to my fellow friends who are by my side as we work on our own projects. It is comforting to have someone working by your side towards the same goal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	:	Artificial Intelligence.
BoW	:	Bag of Words.
BPLion	:	Back Propagation Lion.
BSS	:	Blind Source Separation.
kNN	:	k-Nearest Neighbours.
LDA	:	Linear Discriminant Analysis.
LSA	:	Latent Semantic Analysis.
NLP	:	Natural Language Processing.
NMF	:	Nonnegative Matrix Factorization.
NN	:	Neural Network.
PCA	:	Principal Component Analysis.
RBM	:	Restricted Boltzmann Machine.
SVD	:	Single Value Decomposition.
SVM	:	Support Vector Machine.
TF-IDF	:	Term Frequency-Inverse Document Frequency.

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

1.1 Introduction

In this digital age, data is being generated rapidly and in large quantity. Much of the data generated is unstructured data which includes text messages, scientific articles, news articles, blog posts and others. (Çakir & Güldamlasioğlu, 2016). The data is generated in large quantity and in high velocity that is beyond human's capability to analyse each of them in time. Thankfully, as the data generation capacity increases so do the capability of Artificial Intelligence (AI). A branch of AI is created specifically to automate the process with text and speech. The branch of AI is Natural Language Processing (NLP).

The objective of NLP is to allow computer to understand human natural languages. If the objective is achieved, NLP would be able to read and understand all form of human natural languages including text and speech as well as any human, if not better. In the meantime, although the capability of NLP has not reach the ideal yet, it has improved over the years. NLP would be able to analyse the sentiment of text, sentiment analysis; recognized named entity, named entity recognition; classify text into different groups, text classification among others.

Text classification is the main topic in this research. It is one of the NLP method that group text into different topics or categories. This classification would be helpful for users to narrow down a search quickly, or to know the main topic of the text in a short time.

The technology breakthrough in recent years, machine learning algorithms, processing power of processors have been a boost to the NLP field. With the breakthroughs, there has been a rapid development of new techniques in NLP and AI that can work without domain experts' supervision.

There are several approaches to the text classification problem, multi-label where each

document can belong to several categories or classification, where each document can only belong to one category. Besides classification and multi-labeling, there is clustering, which is an unsupervised machine learning method. In clustering, the data is not labeled and is grouped by similarities based on a similarity measure. This research shall focus on classification rather than multi-label and clustering.

In text classification, most of the algorithms used vector space model to represent the unstructured textual data. (Ababneh, Almanmomani, Hadi, El-Omari, & Alibrahim, 2014). This vector space model represent the sequence of the textual features and their weight, it is easy to implement and provide uniform representation for documents. However, it has a drawback, since it represent all the words in the documents, the dimension of the vector would be huge. This huge vector space model would impact the performance of the machine learning tasks. (Moldagulova & Sulaiman, 2018). Therefore, this study would explore the effect of dimension reduction on vector space model has on text classification.

1.1.1 Problem Statement

Vector space model is one of the most commonly used document representation algorithms, but dimension of the feature space can be too large and the vectors can be too sparse which would affect the accuracy of classification models.

1.1.2 Research Objectives

1. To identify a document representation algorithm that is optimized to extract features from news articles.
2. To investigate the effect of dimension reduction has on high dimensional matrix and the performance of text classification algorithms.
3. To evaluate the performance of the several text classification algorithms and identify the best one.

1.1.3 Research Questions

1. Which document representation and dimension reduction algorithm is optimized for news article?
2. How do complexities of the features influence the performance and accuracy of classification algorithm?
3. Which classification models perform best in text classification?

1.1.4 Research Motivation

In the age of big data, the amount of data generated and collected is growing at an explosive rate. Much of the data generated and collected is in the form of unstructured text. The value contained within the text need to be extracted so that it can be useful to the users. Natural Language Processing (NLP) would be able to do so. Text classification is one of the pillars of NLP.

In text classification, the unstructured text would be given a label or multiple labels depend on the method used. These labels would make the data more meaningful and searchable. Users can search for a topic just by selecting the text with the particular label rather than performing a manual key word search on all the text document. In another scenario, users could know what is the topic of a text in real time if they feed the text to a text classification model.

Bag of Words (BoW) is the most commonly used document representation method in text classification. BoW would produce a vector space model of the textual data representation. The dimension of this vector space model would be huge because the amount of words in the documents is large. This huge dimension of vector space model would be a problem to text classification models as the models need a large amount of memory to store this huge and sparse matrix. Furthermore, in order for the classification models to learn from the patterns of the large matrix and able to classify another text correctly, this would need an immense amount of computing power to process the huge and sparse matrix.

With the emergence of dimension reduction algorithms, the dimension of the vector space could be decreased, less memory would be needed to store the matrix and less computing power would be needed to process the matrix. Compressing or transforming the huge matrix into a matrix with lesser dimension would need some computing power

as well. This is the cost to reduce the dimension. Would this cost incurred increase the performance of the text classification models? Or the performance of the classification models would deteriorate?

This study would investigate the effect of dimension reduction algorithms on different document representation method and subsequently the performance of text classification models.

1.1.5 Research Significance

This research would classify news articles into different categories. In order to do that, first the features have to be extracted from the text. The features might need to be compressed or transformed. Then the transformed features are used to train classification models. After that, the trained models are validated and tested to evaluate its performance.

Bag of words (BoW) approach is the most commonly used document representation method. In this approach, the documents are converted into vector space models. Due to the large amount of words in the documents, the vector space would be large and sparse. This large and sparse matrix would negatively impact the performance of classification models which is known as the curse of dimensionality.

By applying dimension reduction algorithms to the vector space model, the vector space and sparsity of the vector would be reduced. With this reduced vector, the classification models would need to process less information. The performance of the classification models would certainly be influenced by this reduction in dimension. Different dimension reduction algorithms also would result in different reduced vector space. The initial matrix from different bag of words or term vector approach also would produce vector space models with distinct differences.

This study is trying to answer these few questions on which combination of document

representation method, dimension reduction and classification model would have the best performance. The way to get to the bottom of these is to apply the few methods to a dataset and analyse the result.

1.1.6 Expected Outcome

1. A prototype document classification model that can achieve at least 80% accuracy in optimum condition
2. A working pipeline of converting raw text to vector space model or features to be processed by classification models
3. A suitable text classification algorithm is applied on the text classification application
4. A performance evaluation on the text classification models when dimension reduction algorithms are applied

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This study is investigating the effect of different document representation methods pair with different dimension reduction methods have on the performance of text classification models. The 3 aspects mentioned cover the whole pipeline of text classification, which are feature extraction from raw text, document representation; reduce the dimension of the vector space model, dimension reduction; and the classification models. The main focus on this study remain on the effect of dimension reduction has on text classification models.

2.2 Document Representation

2.2.1 Term Frequency

Term frequency is the simplest form of document representation method in BoW approach. It simply convert the words appeared in all the documents or text into a matrix or vector space model.

The columns are all the words appear in the document, each column represent one word. Each of the rows represent each of the document. The value in the cells represent the number of times a word appear in a document, in other word, the value is the term frequency.

Even though it is a simple document representation method, it is proven to be able to achieve a satisfactory results with the right processing and methods. (Moldagulova & Sulaiman, 2018). In addition, term frequency would produce a big and sparse matrix which is exactly what this study is focusing on. Therefore, term frequency would be a suitable document representation algorithm to be applied in this study.

2.2.2 Term Frequency - Inverse Document Freqeuncy (TF-IDF)

Term frequency-Inverse document frequency (TF-IDF) is also a BoW approach. It is build on the concept of term frequency but take it a step further to overcome the setback of term frequency.

Besides term frequency, TF-IDF also take into account the inverse document frequency which means that it also take the frequency of words in other documents into account. TF-IDF provide a measure of weight or importance to the words. The value of TF-IDF estimate the amount of information provided by each word in its document. The value of TF-IDF increases proportionally to the number of times a word appears in a document but is offset by the frequency of the word in the corpus. (Vijayarani, Ilamathi, & Nithya, 2015) The characteristic of TF-IDF can counter the high frequency of some common words, such as the stop words in a language.

The main differences between term frequency and TF-IDF is that in term frequency, if a word has a high frequency in many documents it would be a prominent feature of the data. In TF-IDF however, term frequency alone would not make a word a prominent feature. If a word has high frequency in many documents, the inverse document frequency of TF-IDF would offset its high frequency therefore the word would not be a prominent feature. A word with high TF-IDF score would appear many times in a document but not in many documents.

The value in the vector space model resulting from TF-IDF would be in decimals as the values assigned are scores or weights for each of the words. The values in the resulting vector space model from term frequency on the other hand, consists of integers which represent the frequency of the words appear in the text.

TF-IDF is a simple document representation method that has proven to be efficient and reliable. It has a few setbacks as it only take into account the term frequency and

inverse document frequency and not the semantic of the words. (Qu, Wang, & Zou, 2008). However, the focus area of this study is on the sparse and huge vector space model which TF-IDF produces.

Formula for TF-IDF is shown below:

$$TFIDF = tf \times \log_e \frac{N}{df} \tag{2.1}$$

where:

tf = the number of times a word appears in a document

N = the total number of documents in the corpus

df = the number of documents that contain the word

2.2.3 Summary

Both term frequency and TF-IDF are chosen as the document representation method in this study. Term frequency and TF-IDF are simple and relatively easy to implement. Both of the methods are efficient and tried and true over the years. Most importantly, the output of huge and sparse matrix from term frequency and TF-IDF is the topic of interest in this study.

2.3 Dimension Reduction

2.3.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is probably one of the most popular multivariate statistical technique in dimension reduction. PCA would transform a data table with values from inter-related variables into new orthogonal variables. The variables would be the principal components of the data. In other words, PCA transform data from high dimensional space into low dimensional space with linear transformation while preserving the original data features as much as possible. (Ma & Yuan, 2019) The output from PCA is the principal components of the data which hold 80% to 90% of the information of the original data.

PCA has several objectives, it would extract the most important information from the dataset, the dimension of the dataset would be compressed so that only the important information remained. The dataset is also simplified after PCA has processed it. (Abdi & Williams, 2010)

PCA has been successfully applied to image classification problems. After the features are extracted from the images, PCA is applied to extract the important features from the images. This reduction in dimension allow the researchers to save storage space for the images and still able to process the images in a satisfactory manner. While PCA do reduce the dimension of the input data but this reduction has a cost too. PCA need to consume a large amount of memory and take some time to reduce a data with high dimension. (Ma & Yuan, 2019)

PCA is proven to be efficient in pattern recognition and image analysis and has been extensively applied in face recognition system. However, it is found that PCA is not capable of processing data with high dimensionality and sparsity. PCA is only effective when reducing tens or few hundreds of dimensions. (Rajashekharaiah, Chikkalli, Kumbar, &

Babu, 2018). Thus, PCA might not be suitable to be applied in this study which involves text data and the text data would be converted into a large and sparse matrix.

2.3.2 Nonnegative Matrix Factorization (NMF)

Nonnegative Matrix Factorization (NMF) is a multiplicative updating solution to decompose a nonnegative temporal-frequency data matrix into the sum of individual components. The sum of individual components are calculated from a nonnegative basis matrix. (Chien, 2019)

NMF and its variant have found to be applied in many fields such as feature extractions, segmentation and clustering, dimension reduction and others. In PCA and SVD, the signs of the data is not restricted in any way but NMF has a non-negativity constraint. This means that NMF can only described by using additive components only. This is due to the influences of classical studies where most of the values are in the positives. This non-negativity constraint of NMF has proved to be problematic when the data matrix is not strictly in the positives. (Allab, Labiod, & Nadif, 2017)

NMF has been applied to learn the latent space of the input text data from Twitter. With NMF the data is decomposed into bi-orthogonal non-negative 3-factor, the rows and columns from the different axis are simultaneously clustered. (Lahoti, Garimella, & Gionis, 2018)

NMF might be more suitable for clustering as it took the least amount of time in clustering the data compared to SVD and PCA. (Li & Ding, 2018). This advantage of NMF over SVD and PCA is negligible since this study would focus on classification rather than clustering.

2.3.3 Truncated Single Value Decomposition (SVD)

Single value decomposition (SVD) is one of the most commonly used dimension reduction algorithms. It generalizes a complex matrix with many dimensions into a matrix of lower dimension via an extension of the polar decomposition. SVD detects the part of the data that contains the maximum variance in a set of orthogonal basis vectors. The data with the maximum variance would be the most prominent features of the data. (Sweeney et al., 2014)

The mathematical equation for SVD of a matrix X is shown as follows:

$$X = USV^T \tag{2.2}$$

where:

U = an $m \times n$ matrix, columns are left singular vectors

S = an $n \times n$ non-zero diagonal matrix, the singular values

V^T = an $n \times n$ matrix, rows are right singular vectors

Latent Semantic Analysis (LSA) a technique applied in natural language processing that apply SVD in its process. SVD is applied in LSA to transform the features by dropping the least significant values in the matrix thus reducing the dimensions of the matrix. (Karami, 2017)

Truncated SVD is a variant of SVD. Similar to PCA, truncated SVD is a matrix factorization technique that factors matrix M into 3 matrices namely, U, Σ and V. There is a slight difference between truncated SVD and PCA. PCA performed the factorization on the covariance matrix while truncated SVD performed the factorization on the data matrix directly. Truncated SVD differ slightly from SVD in the way of that SVD would always produce matrices of n columns if given $n \times n$ matrix but truncated SVD given the same

matrix can produce matrices with specified number of columns. (Hauck, 2014)

Truncated SVD can reduced the dimension of the data into lesser dimension when compare to other dimension reduction algorithms such as Linear Discriminant Analysis (LDA) and PCA. The classification model trained with the output of SVD is also higher than that of LDA and PCA. (Rajashekharaiah et al., 2018). Thus, SVD would be a great choice for the dimension reduction algorithm in this study.

2.3.4 Summary

The 3 dimension reduction algorithms above are considered to be Blind Source Separation (BSS) methods, unsupervised learning algorithms. Out of the 3 dimension reduction algorithms, truncated SVD is the most suitable to be applied in this study. This is because PCA is not efficient in reducing high dimensional and sparse data and NMF is only efficient in clustering algorithm. Most importantly, truncated SVD overcome the drawback of PCA and has been proven to be effective and achieve better result than PCA and others. Thus, truncated SVD is chosen to be the dimension reduction algorithm to be applied on the feature matrix extracted from the text.

2.4 Document Classification

2.4.1 Support Vector Machine

Support Vector Machine (SVM) is a machine learning algorithm that construct a hyperplane to separate the data points into different classes. It has been proven to be very effective in dealing with high dimensional data. (Shinde, Joeg, & Vanjale, 2017). It is also proven to produce dramatically better results in text classification shown in experiments with the Reuters dataset. (Dumais, Platt, Heckerman, & Sahami, 1998). However, various issues need to be considered when applying SVM in text classification, the processing of the data, which kernel to use, and the parameters of SVM. A variant of SVM, called one-class SVM which is trained only with positive information has been used in text classification. (L. M. Manevitz & Yousef, 2002). The authors experimented with different kernels of SVM with different type of document representation method. The kernels tested include linear, sigmoid, polynomial, and radial basis. The document representation methods applied are binary representation, frequency representation, TF-IDF, and Hadamard representation. The best result, F1 score of 0.507 is achieved with binary representation, feature length 10 and with linear kernel function.

In another research, the researchers apply SVM in the classification on web document instead of news or ordinary text document. The document representation method used in this research is vector space model, just the nouns term on the web pages. The researchers experimented with different SVM kernels and varying the size of the training sets. Expectedly, the precision, recall and accuracy increased as the size of the training set increase. Linear kernel achieved the best result out of the various SVM kernels, a classification accuracy of 80% is achieved. (Shinde et al., 2017).

SVM is proven to be effective in many fields including text classification. The only drawback with SVM is that it can be tricky to find an appropriate kernel for the problem,

but from the result of several researches above, the most suitable kernel in text classification is most probably the linear kernel.

2.4.2 k-Nearest Neighbours (kNN)

K-Nearest Neighbours (kNN) is a classification machine learning algorithm that classify data based on Euclidean distance between the new data point with the existing data points in the feature space. It is a simple yet effective classification technique, as it only need 3 prerequisites. The 3 prerequisites are training dataset, similarity measure and the value of k which is the number of closest neighbours to be considered.

The similarity measure used in kNN is usually Euclidean distance. The mathematical formula for Euclidean distance is as follows:

$$p = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.3}$$

where:

p = the distance between 2 data point

x_1 = the x-coordinate of a data point A

x_2 = the x-coordinate of data point B

y_1 = the y-coordinate of data point A

y_2 = the y-coordinate of data point B

KNN needs minimal training, it only needs to plot the training samples on a feature space and calculate the Euclidean distance between the new data point with the training samples to determine which category the new data point should be classified as. KNN has been applied in text classification before, it is found that kNN take significantly longer time to classify a document. This is because kNN need to compute the distance between

the new data point with the existing data points and find the nearest data points. Since the authors are using term vector space document representation method, the dimension of the feature space is high, thus more time is needed for kNN to compute all the distance between the new data point with the training data points. Other than the time taken to compute the distance, the k value is another obstacle in kNN algorithm. In a high dimensionality feature space and the points are not evenly distributed, the k value is hard to be determined.

To overcome the problems mentioned above, the authors applied naive term vector space reduction method, divide the document feature matrix into parts. Term vector space reduction reduces the sparsity of the document term matrix by removing the features less appeared in the corpus. By reducing the term vector space, a slight deterioration in the classification accuracy but the time cost is dramatically reduced. kNN still achieved an accuracy of 92.7% but the time taken reduced from 53 minutes to 11 minutes. (Moldagulova & Sulaiman, 2018)

2.4.3 Neural Network

Neural network (NN) has a resurgence in recent years as there is a breakthrough in the neural network since Geofrey Hinton (et al.) discovered a technique called Contrastive Divergence that could quickly model inputs in a Restricted Boltzmann Machine (RBM). RBM is a 2-layer neural network that model the input by bouncing it through the network. This process is less computationally complex than backpropagation. (Hinton, 2002).

Currently, neural network is applied in deep learning to solve various problems, text classification being one of them. Ranjan (et al.) applied Lion Fuzzy Neural Network on text classification. The researchers used WordNet ontology to retrieve the semantic of the words, and then added context information onto it, thus the features obtained are semantic-context features. The classification part is performed by Lion Fuzzy Neural Network, which is a variant of Back Propagation Lion (BPLion) Neural Network that includes fuzzy bounding and Lion Algorithm. The neural network model used is trained incrementally. It achieves a higher accuracy than Naïve Bayes and other variant of the Lion Neural Network. (Ranjan & Prasad, 2018)

Besides the modified neural network shown above, a simple feed-forward neural network is also proven to be efficient in text classification. By using the Hadamard product as document representation method, a simple neural network also can achieve a good classification accuracy in text classification compare to Naïve Bayes, kNN, and SVM. (L. Manevitz & Yousef, 2007)

2.5 Conclusion

In the document representation algorithm, both term frequency and TF-IDF would be applied in this study. Both of the document representation algorithm would produce matrix with high dimension. This is the purpose of this study and it would be interesting to observe how different document representation algorithm affect the result.

For dimension reduction algorithm, truncated SVD is chosen as the dimension reduction algorithm to be used in this study. SVD has proven to be efficient and achieve satisfactory result from past researches.

In machine learning algorithms for text classification, all 3 machine learning algorithms reviewed above would be applied. One of the objectives of this study is to investigate the performance of different machine learning algorithms in text classification. Therefore, the same dataset would be used to train 3 models and the performance of the 3 classification algorithms would be evaluated in order to identify the best classification model.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

This section would illustrate the process flow to carry out the experiments and fulfill the aforementioned objectives of this study. The following flow charts would illustrates the steps taken to conduct the several experiments.

3.2 Text preprocessing process flow

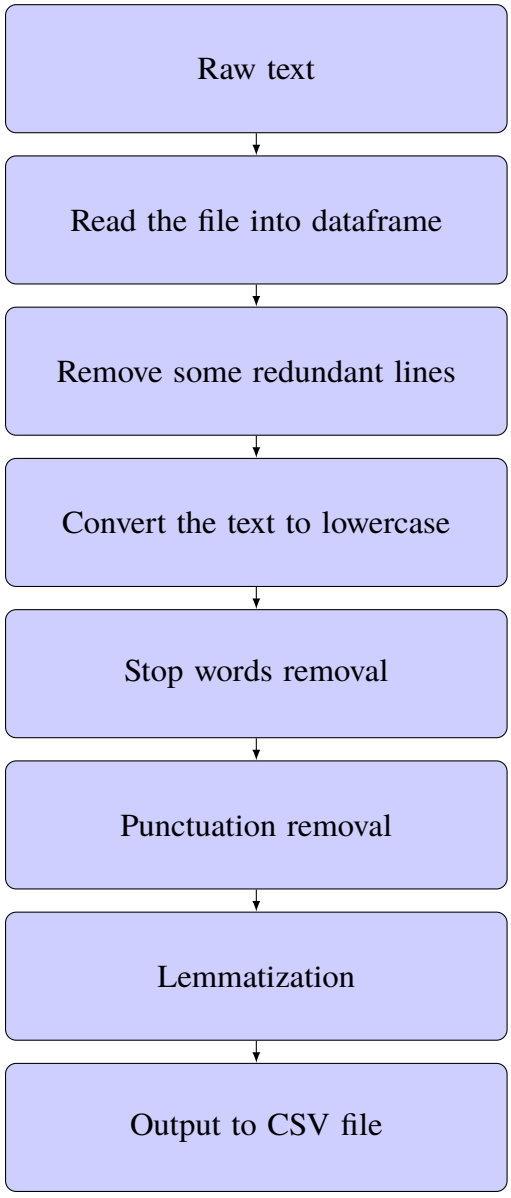

Figure 3.1: Overall process flow

This flow chart above illustrates the preprocessing process. The news articles dataset is

in the form of raw text. Some preprocessing need to be done before the features can be extracted from the text to build the text classification models. First, the raw text would be read into a dataframe or a data structure for ease of access and processing.

In the raw text files, there are some lines that are irrelevant to text classification, these lines would be removed to reduce the noise in the dataset. All the text in the dataset would be converted to lowercase for uniformity and ease of processing. There are some words in the text that are useful in human speech but do not convey meanings in text analysis. These words are stop words, these words have to be removed as well. After stop words removal, the text should contain only the words that are vital to text analysis but there would be some punctuations and symbols in the text. These punctuations and symbols would be removed as well so that the text is cleaner and less noisy.

After that, there is an important step, lemmatization. Lemmatization would transform most of the words in the text to their root form which is known as a lemma. This would reduce the noise in the data by transforming similar words into a single word. The alternative to lemmatization would be stemming. Stemming would be faster and need less processing power than lemmatization but stemming has a drawback. The result of stemming may not be a real word because stemming just chop off the end of the words thus the resulting words may not be a real words. On the other hand, lemmatization uses vocabulary and morphological analysis of words to remove the inflectional endings only and resulting the root form of words. (Nicolai & Kondrak, 2016) Therefore, the output of lemmatization would be better than stemming.

3.3 Overall Process Flow

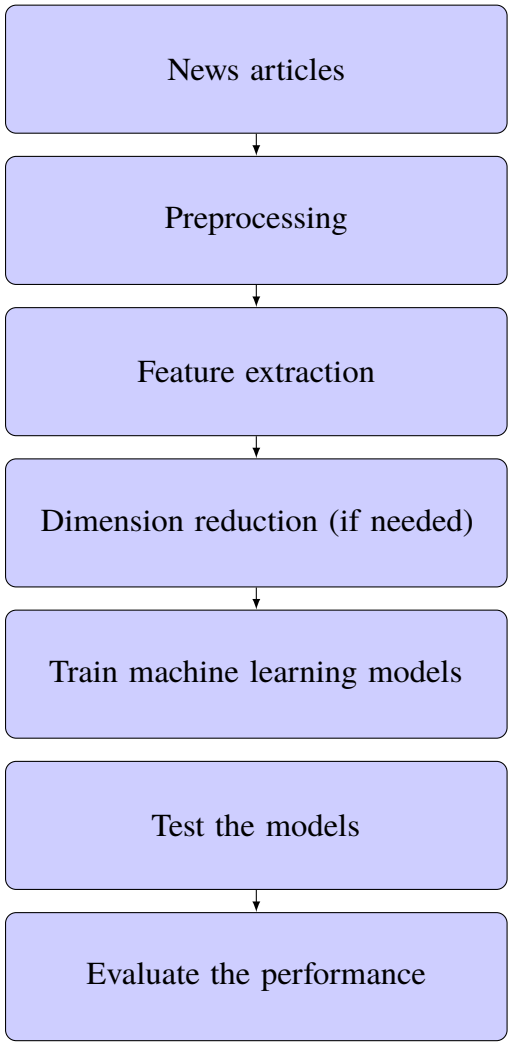

Figure 3.2: Process flow of preprocessing the text dataset

The process flow chart above is the overall process for the few experiments. As mentioned above, the news articles dataset has to be preprocessed before it can undergo feature extraction and being used to train machine learning models.

After preprocessing, the resulting data would be used in several experiments with a few subtle differences. The differences in the experiments would be in the feature extraction stage and the dimension reduction stage.

3.4 The dataset

The dataset used in this research is the 20 news group dataset. It is freely available at `http://qwone.com/~jason/20Newsgroups/`. It is a collection of news articles that can be divided almost evenly to 20 groups. Some of the groups are closely related to one another and some are widely different. Categories with the same prefix for example *comp* would have high similarity with one another but also subtle differences. The categories with the different prefixes, for example *rec* and *talk* would be very different from one another.

This characteristic of the dataset make it a good dataset to test text classification. A good classification model should be able to differentiate the different groups of news article even the groups that are closely resembled one another.

The table below show the number of count for each category. There is a total of 18790 articles and most of the categories have almost 1000 articles in them.

Category	count
alt.atheism	798
comp.graphics	970
comp.os.ms-windows.misc	980
comp.sys.ibm.pc.hardware	978
comp.sys.mac.hardware	957
comp.windows.x	978
misc.forsale	962
rec.autos	988
rec.motorcycles	993
rec.sport.baseball	993
rec.sport.hockey	998
sci.crypt	991
sci.electronics	981
sci.med	988
sci.space	986
soc.religion.christian	997
talk.politics.guns	910
talk.politics.mideast	940
talk.politics.misc	774
talk.religion.misc	628

Table 3.1: The count of each category in the dataset

The bar chart below visualized the data displayed on the table for ease of viewing. Most of categories have almost 1000 articles except alt.atheism, talk.politics.misc and talk.religion.misc.

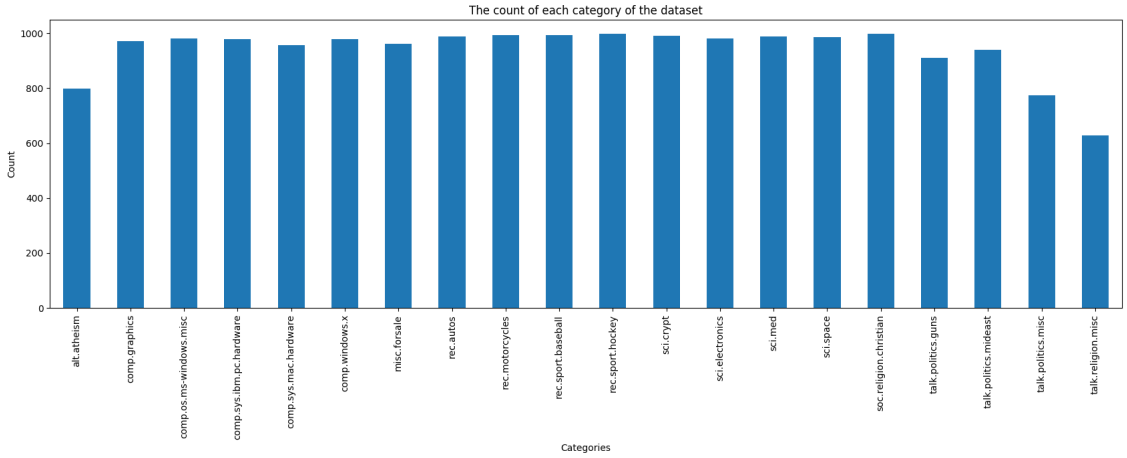

Figure 3.3: The count of each category in the dataset

As mentioned there are 18790 articles in the data, the articles would be split at random into 8:2 ratio. 80% of the data would be used as training data to train the classification model while 20% of the data would be use as the test data. The training data would consist of 15032 articles while the test data would consist of 3758 articles.

3.5 The experiments

The experiments that has been conducted in this research is as follows:

1. Term frequency
2. Term frequency with naive dimension reduction
3. Term frequency with truncated SVD
4. TF-IDF
5. TF-IDF with naive dimension reduction
6. TF-IDF with truncated SVD

Basically, there are 2 feature extraction method used in the experiments, namely term frequency and term frequency - inverse document frequency (TF-IDF). Each of the feature extraction method would be tested with and without dimension reduction algorithm.

There are 2 dimension reduction algorithms involved, one is a naive method, which means that the features or columns lesser than a certain value would be removed. In other words, words or terms that do not appear much in the dataset would be removed. Another dimension reduction algorithm is truncated single value decomposition (SVD). This method would retain the essence of the data, the part of the data with maximum variance.

After feature extraction and dimension reduction (if needed) are applied, the resulting features would be used to train machine learning models. There are 3 machine learning algorithms chosen in these experiments namely k-nearest neighbour (kNN), support vector machine (SVM), and neural network (NN). All 3 of the machine algorithms would be applied to all the different resulting features and the accuracy scores would be evaluated.

3.6 Conclusion

With the method mentioned above, the experiments would be carry out and the results would be recorded. The results would be compared and the differences would be analysed.

CHAPTER 4: RESULTS AND DISCUSSIONS

4.1 Introduction

The results of the experiments are shown in this section. The differences in the accuracy and performance of each of the methods would be discussed and analysed. Hopefully, the results and discussions would be able to answer the research questions posted in the beginning of this research.

4.2 Term frequency

ML	no of features	accuracy	time taken (s)
kNN	8000	0.31	3.74
SVM	8000	0.81	5.27
NN	8000	0.84	91.82

Table 4.1: Term frequency

Term frequency is one of most common feature extraction method in text. It convert all the words in the dataset into a matrix where each column is a word and the value in each of the cells are the number of times each word appear in the text. Each row in the matrix is a document. This vector space model representation of the words would result in a sparse matrix since each of the documents only contains a subset of all the words in the whole dataset.

In this experiment, the number of features of vector space model from term frequency extraction is limited to 8000, this is because of the memory constraint of the machine, if it is unlimited, the resulting matrix would be of bigger size and would have the risk of running out of memory while processing the matrix.

In the results shown above, NN achieved an accuracy of 0.84 which is the best accuracy out of the 3 classification algorithm but it is also the one that takes the longest to process which is 91.82s. KNN took the least time to process, 3.74s but has the lowest accuracy

score, 0.31. Overall performance, SVM is the best, it only took 5s to process, which is 80s less than NN and has an accuracy of 0.81 which is comparable with NN.

KNN accuracy is low in this scenario is possibly due to the sparsity of the feature matrix or vector space model. The data points are few and far in between. KNN is designed for low dimensional data therefore it doesn't perform well in large and sparse data. The underlying reason would be due to kNN classify a new data point based on the distance of the new data point with the nearby data points. Since the other data points are few and sparse, the classification of the new data points would be biased to the few data points that are near. This bias would reduce the accuracy of the classification. (Wang, Lin, Huang, Wang, & He, 2010)

Neural Network (NN) took the longest time to process the features, this is due to the layers of neurons in NN. The vector space model even though sparse has high dimension, NN would need as much if not more neurons as the number of features to process the data. This processing would takes both time and computing power.

SVM depends on the prominent features of the dataset, the support vectors to classify the dataset so it is relatively faster and as accurate.

4.3 Term frequency with naive dimension reduction

ML	parameter value	no of features	accuracy	time taken (s)
kNN	100	2530	0.34	3.94
kNN	500	478	0.42	4.66
kNN	1000	173	0.41	5.17
SVM	10	12705	0.83	6.10
SVM	100	2530	0.76	5.67
SVM	500	478	0.63	20.20
NN	10	12705	0.87	116.78
NN	100	2530	0.80	48.08
NN	500	478	0.65	61.87

Table 4.2: Term frequency with naive dimension reduction

The feature extraction method used in this experiment is same as the experiment above but dimension reduction method is applied to the resulting matrix before training. The dimension reduction algorithm used in this experiment is a naive one which means that the columns with the least occurrence of word are removed assuming those words are unimportant and would not have much influence on the accuracy.

The parameter value is just an integer value passed to the dimension reduction function implemented. It has an inversely proportional relationship with the number of features. The larger the parameter value, the more columns would be removed and the number of features would decrease.

In this experiment, kNN still has the worst performance among the 3 which is at 0.34 with 2530 number of features. As the number of features decreases to 173, the accuracy increases slightly to 0.41. As the number of features decreases, the time taken for kNN to produce the result increases from 3.94s to 5.17s.

SVM and NN have the same trend in this experiment, the accuracy decreases as the number of features decreases. Accuracy of SVM decreases from 0.83 to 0.63 as the number of features decreases from 12705 to 478. The time taken by SVM fluctuates as the amount of features decreases, it decrease from 6s to 5s when the amount of features

decreases from 12705 to 2530, but the time taken increases from 5s to 20s, as the number of features decreases further to 478. The decrement in accuracy in SVM is slight in relative to the decrement in the number of features. Accuracy decreased by 0.20 while the number features has decreased by 12227.

The time taken by SVM increases quite drastically when the number of features decreased. The sparsity of the vector space model decreases, resulting in a denser vector space model. SVM need to take into account more data points to calculate an effective hyperplane to separate the data points into different classes.

On the other hand, the accuracy of NN decreases from 0.87 to 0.65 as the number of features decreases from 12705 to 478. However, the time taken by NN fluctuates from 116.78s to 48.08s and then to 61.87s when the amount of features decreases from 12705 to 2530 then to 478. The overall decrement in time taken is possibly due to the reduction of features, less neurons are needed to process the data and thus the speedup.

In the scenarios where SVM and NN have more features than the first experiment with just term frequency, both of the classification algorithms have better accuracy. These 2 classification algorithms are efficient and able to process features in a large and sparse vector space.

4.4 Term frequency with truncated SVD

ML	no of features	accuracy	time taken (s)
kNN	4000	0.31	482.66
kNN	2000	0.38	216.59
kNN	500	0.49	60.89
kNN	100	0.55	15.38
kNN	10	0.30	2.78
SVM	4000	0.80	370.43
SVM	2000	0.78	172.58
SVM	500	0.77	85.03
NN	4000	0.80	220.76
NN	2000	0.79	91.13
NN	500	0.77	40.86

Table 4.3: Term frequency with truncated SVD

In this experiment, the feature extraction or document representation method used is still term frequency but the dimension reduction algorithm has been changed. Instead of reducing the dimension of the feature matrix naively by removing the terms that has the lowest frequency, truncated SVD is used. Truncated SVD would reduce the dimension of the vector space model by retrieving the features with maximum variance in the data.

The number of features shown in the table above is the number of columns in the resulting matrix after truncated SVD is applied.

Similar with the trend in the previous experiment (term frequency with naive dimension reduction), the accuracy of kNN increases slightly, from 0.31 to 0.55 when the features decreases from 4000 to 100, but the resulting accuracy is still far from satisfactory. The time taken by kNN decreases from 482s to 15s as the number of features decreases. The trend of accuracy increment as the number of features decrease cease when the number of features is reduced to 10. KNN accuracy decrease from 0.55 to 0.30 at that point. The fluctuation of the accuracy would be due to kNN dependency on Euclidean distance between the data points to classify a new data. As the amount of features decrease, the feature matrix becomes less sparse, kNN would need to process less data points thus the

speedup. A less sparse matrix resulting from the decrement of features also make kNN classification more accurate and less biased but when the features decrease to an extent where there is no sufficient data to classify a data point correctly, the accuracy dropped.

The accuracy of SVM and NN also have the same trend with the previous experiment, the accuracy decreases slightly when the number of features decreases. Accuracy of SVM decreases from 0.80 to 0.78 then to 0.77 and the accuracy of NN decreases from 0.80 to 0.79 then further to 0.77 as the number of features decreases from 4000 to 2000 and 500.

As expected, the time taken by the classification model to reduce the dimension and predict the result decreases as the number of features decreases. However, when compare with previous experiments, the time taken in this experiment is astoundingly higher in the case where the features amount to 4000 and 2000. SVD would be the culprit, dimension reduction comes at a cost which is processing power. To calculate the maximum variance of the features and transform the feature matrix into a smaller dimension would require no small feats of calculation, this would consume both time and processing power.

However, the resulting accuracy is not as good as just with term frequency. This is expected because the number of features decreased, the classification would have lesser information to classify a new data point correctly. The reduction in features may save some memory space but in order to achieve that more processing power and time would be needed.

Comparing the performance of the classification models in this experiment with the second experiment, term frequency with naive dimension reduction, the performance of the classification models are almost similar when the amount of features are around 2000. In the scenarios where the amount of features is around 500, SVD has a better performance. The classification models with SVD achieved a higher accuracy than that of naive dimension reduction. At this stage, the advantage of SVD over naive dimension

reduction is shown. SVD transformed the feature matrix into smaller dimension but retaining the maximum variance or the most prominent feature of the data. Therefore, models trained with SVD should have a better performance than those that trained with naive dimension reduction.

The next experiments would apply different document representation algorithm to investigate further the effect of dimension reduction has on classification model performance and compare the performance of different document representation method.

4.5 TF-IDF

ML	no of features	accuracy	time taken (s)
kNN	8000	0.76	3.71
SVM	8000	0.87	2.39
NN	8000	0.88	55.82

Table 4.4: TF-IDF

The previous experiments found that SVD has a slight edge over naive dimension reduction when the number of features decreased to a certain extend. In this and the next experiments, the effect on dimension reduction is further explored. A different document representation algorithm is applied in this and the next 2 experiments.

The document representation algorithm applied in this experiment is TF-IDF. In contrast with term frequency which only take the frequency of each word into account, TF-IDF takes both the frequency of each word and the rarity of it into account. If a term or word appear in high frequency but in many documents, this word may not be of importance and consequently is not a meaningful feature. If a word appear rarely and only in a few documents, this word would have high importance and would be a meaningful feature of the few documents.

With TF-IDF, kNN can achieved a satisfactory accuracy score of 0.76 even though the number of features in the resulting matrix of TF-IDF is the same with term frequency which is 8000. The vector space model of TF-IDF would not be as sparse as that of term frequency which is more suitable for kNN.

NN is still provide the highest accuracy score of 0.88 but the time taken also the longest at 55.82s.

SVM achieved an accuracy of 0.87 which is just 0.01 shy of what achieved by NN and the time taken is the lowest among the 3 which is 2.39s. SVM can achieve high accuracy even with high dimension data, because SVM uses the prominent features or

support vectors from the data to perform classification, SVM's computational complexity is independent of the dimension of the data. (Rajashekharaiah et al., 2018)

In comparison with the first experiment that apply term frequency document representation without dimension reduction, the performance of the classification models significantly improve. KNN accuracy has more than doubled from 0.31 to 0.76. SVM accuracy increases from 0.81 to 0.87 while accuracy of NN increases from 0.84 to 0.88. Besides accuracy, the time taken also improved, time taken by SVM reduces from 5.27s to 2.39s while time taken by NN reduces from 91.82s to 55.82s. All these improvements are achieved with the same number of features in the vector space, which is 8000.

The performance of the classification models in this experiment is also better than those in the 2 experiments above with dimension reduction. However this may not be a fair comparison because the different number of features are used and dimension reductions are not applied in this experiment. Dimension reduction algorithms would be applied to the vector space model from TF-IDF in the following experiments in order to have a fair comparison.

The performance increment from term frequency to TF-IDF seems to prove that TF-IDF is a better document representation method.

4.6 TF-IDF with naive dimension reduction

ML	parameter value	no of features	accuracy	time taken (s)
kNN	50	4221	0.74	4.00
kNN	100	2530	0.71	4.11
kNN	500	478	0.49	5.15
SVM	50	4221	0.86	2.58
SVM	100	2503	0.83	2.63
SVM	500	478	0.66	2.94
NN	50	4221	0.85	38.80
NN	100	2530	0.83	34.28
NN	500	478	0.64	77.12

Table 4.5: TF-IDF with naive dimension reduction

Similar with the experiment with term frequency, naive dimension reduction is applied to the vector space model generated from TF-IDF. The trend over all the 3 machine learning models when the number of features decreases are similar. The accuracy of the classification models decreases and the time taken increases.

The accuracy achieved by kNN with TF-IDF plus naive dimension reduction is still passable at 0.74 when the features reduced from 8000 to 4221. KNN's accuracy dropped slightly to 0.71 when the number of features decreases to 2530. When the number of features decreases from 2530 to 478, as expectedly the accuracy of kNN dropped for quite a large margin, from 0.71 to 0.49. The time taken by kNN increases slightly from 4s to 5s as the features decreases.

SVM has the same behaviour with kNN in this experiment, its accuracy decreases from 0.86 to 0.83 and then to 0.66 when the number of features decreases from 4221 to 2503 to 478. The time taken increases slightly as well as the number of features decreases.

NN also has the similar trend in accuracy as the number of features decreases. NN's accuracy decreases from 0.85 to 0.64 when the number of features decreases from 4221 to 478. The differences in NN is that the time taken almost doubled when the number of features is low compare to when the number of features is high. The time taken is 38s

when there is 4221 features and when there is only 478 features, the time taken doubled to 77s.

In the scenarios where the number of features are almost halved, from 8000 to 4221, the performance of the classification models are still comparable with the performance achieved with just TF-IDF without any dimension reduction. The accuracy are almost the same, the time taken is similar except NN which has quite a speedup when the number of features is reduced to 4221.

It can be deduced that with naive dimension reduction, the number of features and memory needed to store the vector space model is reduced. With this reduced number of features, the classification models can still achieve comparable performance at a slightly decreased capacity.

This is mainly because of the reduction in features. As the amount of information became lesser, less information is available to train a comprehensive model. Therefore, the accuracy of the models decrease. Even though the dimension reduction applied is a naive one and less computing intensive, it still increases the time taken compared with just with TF-IDF. The more reduction is performed, the time taken would increase as well.

4.7 TF-IDF with truncated SVD

ML	no of features	accuracy	time taken (s)
kNN	4000	0.77	457.16
kNN	2000	0.58	208.64
kNN	500	0.55	58.58
kNN	100	0.69	15.09
kNN	50	0.69	7.58
kNN	10	0.55	2.54
SVM	4000	0.87	189.32
SVM	2000	0.86	69.63
SVM	500	0.83	17.54
NN	4000	0.85	215.73
NN	2000	0.84	87.12
NN	500	0.81	39.93

Table 4.6: TF-IDF with truncated SVD

In this last experiment, truncated SVD dimension reduction is applied to the resulting vector space model from TF-IDF. Similar with experiment before, each of the classification models would be tested with several set of vector space model, each with different number of features. To put it in perspective, the number of features without reduction is 8000.

When the number of features are at 4000 which is halved, kNN still can produced an accuracy of 0.77 which is similar with what is achieved with TF-IDF without dimension reduction. Same goes to SVM and NN, at 4000 features, the accuracy are quite similar with TF-IDF without dimension reduction. However, when the dimension of the vector space model is reduced to 2000, the accuracy across the 3 classification models dropped. KNN being the most drastic, its accuracy dropped to 0.58 while SVM and NN dropped to 0.86 and 0.84 respectively, which is slightly worse than before but it is still satisfactory.

Comparing with the results of the experiment with TF-IDF and naive dimension reduction, this performance of the classification models with truncated SVD has a slight advantage. The accuracy is slightly better with truncated SVD than that with naive dimension reduction. This would be due to the advantage of truncated SVD obtaining the

maximum variance of the features over naive dimension reduction.

The time taken in this experiment is much higher than the experiment of TF-IDF without dimension reduction and TF-IDF with naive dimension reduction which is expected. This would be due to SVD reduction takes more time as more calculation is needed to transform the data. However, the time taken decreased when the further reduction is done, kNN take 457s to reduce the vector space model from 8000 to 4000 but just 2s to reduce the vector space model from 8000 to 10. Keep in mind that the time recorded here includes the time taken for the classification model to predict the test dataset as well as the dimension reduction time. This trend appear in SVM and NN as well. SVM took 189s to reduce 8000 to 4000 but just 18s to reduce 8000 to 500 while NN took 215s to reduce 8000 to 4000 and 40s to reduce 8000 to 500.

The decrement in time could be explained by the reduction of features, as the amount of features decreases, the time taken to process them also reduced. Therefore there is a speedup. This speedup that comes with the reduction of features, comes at a cost which is accuracy. For SVM and NN, the reduction in accuracy is meagre thus it would be logical to trade the slight accuracy with the speedup but in kNN, the trade off would be lopsided in the favour of time taken.

4.8 Accuracy vs Dimension of term frequency matrix

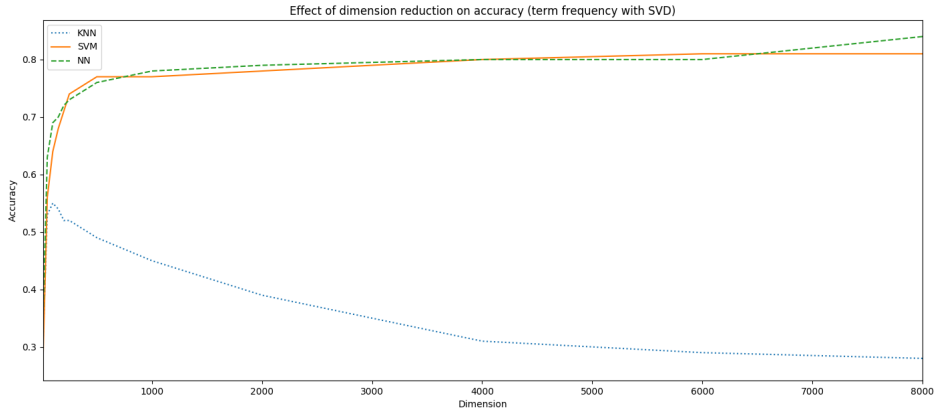

Figure 4.1: The effect of dimension reduction on accuracy with term frequency

This figure illustrates the effect of dimension reduction has on the accuracy of the classification models. The feature extraction algorithm used here is term frequency and the dimension reduction algorithm used is truncated SVD.

KNN accuracy increases from 0.3 to 0.55 as the dimension of the feature matrix decreases from 8000 to 100. After that point, further decrease in dimension does not increase the accuracy of kNN further but decrease its accuracy back to 0.3.

SVM and NN have the similar trend as the dimension of the feature matrix reduces. Both of them were at high accuracy, around 0.8 when the dimension of the matrix is at 8000. Both SVM and NN accuracy decrease slightly as the dimension decreases. When the dimension of the matrix is reduced to an extreme, 10, both SVM and NN accuracy dropped to around 0.3.

This showed that kNN is a classification model that is sensitive to dimension changes and is more efficient at lower dimension. SVM and NN, however, are more resilient to the feature matrix dimension changes. Both of SVM and NN accuracy remain stable as the dimension decreases.

4.9 Accuracy vs Dimension of TF-IDF matrix

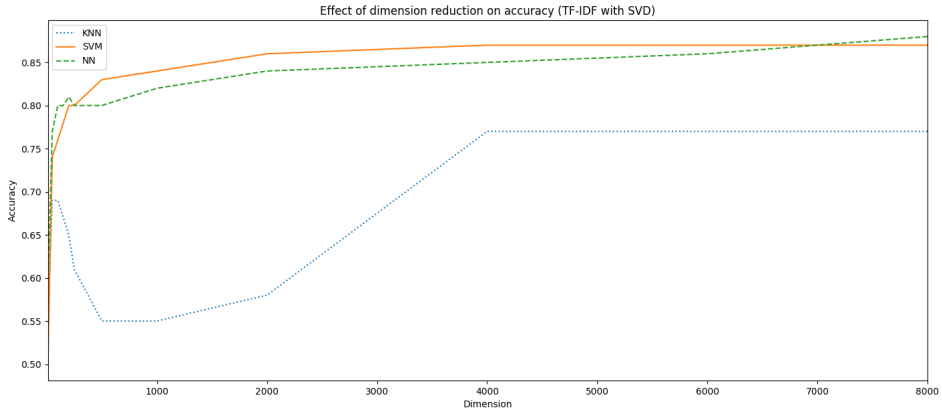

Figure 4.2: The effect of dimension reduction on accuracy with TF-IDF

This figure show the fluctuations of the accuracy of the classification models as the dimension decreases. The feature extraction method used here is TF-IDF and the dimension reduction algorithm used is truncated SVD.

KNN accuracy remain stable at around 0.77 as the dimension decrease from 8000 to 4000. After 4000, further decrement in dimension resulting in a decrease in accuracy. The accuracy of kNN deflected and increase from 0.55 to 0.69 when the dimension reduce from 500 to 10.

Similar to previous result, SVM and NN accuracy remain stable around 0.8 as the dimension reduces from 8000 to 500. When the dimension of the feature matrix is reduced to an extreme of 10 then the accuracy of both SVM and NN deteriorated to around 0.5.

SVM is the best classification model out of the 3, it has slightly better performance than NN in most of the scenarios. It is most resilient to the changes in dimension. This is because SVM depend on the prominent features of the data to classify the data into different classes, the dimension of the data do not has much influence on SVM computational complexity. (Rajashekharaiah et al., 2018). NN has comparable accuracy with SVM but it would take longer time than SVM to produce the similar result.

4.10 Conclusion

From the results of the 6 experiments above, it is found that TF-IDF is a better document representation algorithm than term frequency. The resulting vector space model from TF-IDF can achieve a higher accuracy than that of term frequency.

The effect of dimension reduction on the accuracy of the classification models is analysed. Dimension reduction, naive and truncated SVD, do reduce the dimension of the vector space model, reducing the memory needed to store the matrix. However, this reduction in features and information would result in a loss of accuracy. truncated SVD would be the better dimension reduction algorithm compared to the naive method because the accuracy achieved with truncated SVD is higher than that of the naive method.

Among the 3 classification models tested in the experiments, SVM is the most efficient and versatile. SVM can achieve high accuracy (> 0.80) in most scenarios. NN can also achieve high accuracy in most of the cases tested but NN is more time consuming. SVM has an advantage over NN on the aspect of processing time. Therefore, SVM would be the most efficient text classification model among the 3 classification models.

CHAPTER 5: CONCLUSION

It is known that TF-IDF should be better than term frequency but Moldagulova, Aiman and Sulaiman, Rosnafisah proposed that with term frequency and naive dimension reduction, kNN classification model would be able to achieve a satisfactory result. (Moldagulova & Sulaiman, 2018). Therefore, this research compare term frequency and TF-IDF and apply different dimension reduction methods to both. The implementation of the kNN algorithm in this research might differ from that proposed by the authors. They divided the larger feature matrix into parts and train by part.

From the results of the experiments it is shown that TF-IDF is a better document representation method than term frequency. Classification models can achieved a better result with TF-IDF than term frequency. With term frequency as the document representation method, SVM and NN are able to achieve satisfactory accuracy but the accuracy achieved with TF-IDF is slightly higher. The advantage of TF-IDF is more obvious, with kNN, with term frequency the accuracy is a meagre 0.31 but with TF-IDF, the accuracy is 0.76 which is close to 0.80.

By applying the 2 types of dimension reduction to both the document representation method, the effect of dimension reduction on the performance of the classification models is analysed. The classification models accuracy from both naive dimension reduction and truncated SVD are quite similar in most cases. However, truncated SVD has a slight edge over naive dimension reduction. Truncated SVD can provide classification models with more prominent features and subsequently higher accuracy.

As the dimension of the vector space model is reduced, the accuracy of the classification models would decreased due to the lack of features or information. This reduction of features would reduce the memory used to store the vector space model. However, the

reduction process would need intensive computing power and time, especially in the case of truncated SVD. This the trade off between dimension reduction and classification accuracy. The gain in less memory to store the matrix would cause an increase in computing power consumed and time taken.

Out of the 3 classification models tested in this research, SVM has the best overall performance. SVM can achieve a satisfactory accuracy in most scenarios and the time taken for SVM to predict the test data is among the shortest. Thus SVM would be the best classification models in text classification.

For future works, ways to improve the accuracy of the classification models with TF-IDF should be explored. More document representation method, such as n-gram could be apply alongside TF-IDF and different dimension reduction could be applied. TF-IDF does not take the semantic of the term into account just the frequency and rarity, n-gram would be able to rectify part of the problem.

REFERENCES

Ababneh, J., Almanmomani, O., Hadi, W., El-Omari, N., Alibrahim, A. (2014, 02). Vector space models to classify arabic text. *International Journal of Computer Trends and Technology (IJCTT*, *7*, 219-223. doi: 10.14445/22312803/IJCTT-V7P109

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

Allab, K., Labiod, L., Nadif, M. (2017, Jan). A semi-nmf-pca unified framework for data clustering. *IEEE Transactions on Knowledge and Data Engineering*, *29*(1), 2-16. doi: 10.1109/TKDE.2016.2606098

Chien, J.-T. (2019). Chapter 5 - nonnegative matrix factorization. In J.-T. Chien (Ed.), *Source separation and machine learning* (p. 161 - 229). Academic Press. Retrieved from http://www.sciencedirect.com/science/article/pii/B9780128045664000176 doi: https://doi.org/10.1016/B978-0-12-804566-4.00017-6

Dumais, S., Platt, J., Heckerman, D., Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on information and knowledge management* (pp. 148–155). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/288627.288651 doi: 10.1145/288627.288651

Hauck, T. (2014). *Scikit-learn cookbook*. Packt Publishing.

Hinton, G. E. (2002, August). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, *14*(8), 1771–1800. Retrieved from http://dx.doi.org/10.1162/089976602760128018 doi: 10.1162/089976602760128018

Karami, A. (2017, Nov). Taming wild high dimensional text data with a fuzzy lash. In *2017 ieee international conference on data mining workshops (icdmw)* (p. 518-522). doi: 10.1109/ICDMW.2017.73

Lahoti, P., Garimella, K., Gionis, A. (2018). Joint non-negative matrix factorization for learning ideological leaning on twitter. In *Proceedings of the eleventh acm international conference on web search and data mining* (pp. 351–359).

Li, T., Ding, C.-c. (2018). Nonnegative matrix factorizations for clustering: A survey. In *Data clustering* (pp. 149–176). Chapman and Hall/CRC.

Ma, J., Yuan, Y. (2019). Dimension reduction of image deep feature using pca. *Journal of Visual Communication and Image Representation*, *63*, 102578. Retrieved from http://www.sciencedirect.com/science/article/pii/S1047320319301932 doi: https://doi.org/10.1016/j.jvcir.2019.102578

Manevitz, L., Yousef, M. (2007). One-class document classification via neural networks. *Neurocomputing*, *70*(7), 1466 - 1481. Retrieved from http://www.sciencedirect.com/science/article/pii/S092523120600261X (Advances in Computational Intelligence and Learning) doi: https://doi.org/10.1016/j.neucom.2006.05.013

Manevitz, L. M., Yousef, M. (2002, March). One-class svms for document classification. *J. Mach. Learn. Res.*, *2*, 139–154. Retrieved from http://dl.acm.org/citation.cfm?id=944790.944808

Moldagulova, A., Sulaiman, R. B. (2018, Oct). Document classification based on knn algorithm by term vector space reduction. In *2018 18th international conference on control, automation and systems (iccas)* (p. 387-391).

Nicolai, G., Kondrak, G. (2016, aug). Leveraging inflection tables for stemming and lemmatization. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1138–1147). Berlin, Germany: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/P16-1108 doi: 10.18653/v1/P16-1108

Qu, S., Wang, S., Zou, Y. (2008, Nov). Improvement of text feature selection method based on tfidf. In *2008 international seminar on future information technology and management engineering* (p. 79-81). doi: 10.1109/FITME.2008.25

Rajashekharaiah, K., Chikkalli, S. S., Kumbar, P. K., Babu, P. S. (2018). Unified framework of dimensionality reduction and text categorisation. *International Journal of Engineering & Technology*, *7*(3.29), 648-654.

Ranjan, N. M., Prasad, R. S. (2018). Lfnn: Lion fuzzy neural network-based evolutionary model for text classification using context and sense based features. *Applied Soft Computing*, *71*, 994 - 1008. Retrieved from http://www.sciencedirect.com/science/article/pii/S1568494618304046 doi: https://doi.org/10.1016/j.asoc.2018.07.016

Shinde, S., Joeg, P., Vanjale, S. (2017, 09). Web document classification using support vector machine. In *2017 international conference on current trends in computer, electrical, electronics and communication (ctceec)* (p. 688-691). doi: 10.1109/CTCEEC.2017.8455102

Sweeney, E., Vogelstein, J., Cuzzocreo, J., A Calabresi, P., Reich, D., M Crainiceanu, C., T Shinohara, R. (2014, 04). A comparison of supervised machine learning algorithms and feature vectors for ms lesion segmentation using multimodal structural mri. *PloS one*, *9*, e95753. doi: 10.1371/journal.pone.0095753

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

Wang, J., Lin, L., Huang, T., Wang, J., He, Z. (2010). Efficient k-nearest neighbor join algorithms for high dimensional sparse data. *arXiv preprint arXiv:1011.2807*.

Çakir, M. U., Güldamlasioğlu, S. (2016, June). Text mining analysis in turkish language using big data tools. In *2016 ieee 40th annual computer software and applications conference (compsac)* (Vol. 1, p. 614-618). doi: 10.1109/COMPSAC.2016.203

APPENDIX A: DATA PREPROCESSING IMPLEMENTATION

The code in this section show how the 20 news group dataset is processed before it is used to train classification models.

```
"""
Read the raw dataset downloaded from uci repository into a csv
    file in 2 columns, namely text and category
"""
import os
import pandas as pd
import logging
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import wordnet
import time
import re

nltk.download('stopwords')
nltk.download('wordnet')
ps = PorterStemmer()
lemmatizer = WordNetLemmatizer()
logging.basicConfig(level=logging.INFO, format='%(asctime)s - 
    %(message)s')

output_loc = '../output/20newsGroup18828.csv'


def get_files():
    path = '/home/dante/development/datasets/20news-18828/'
    # path = '/home/db/development/datasets/20news-18828/'

    # Getting all the files path
    files = []
    for r, d, f in os.walk(path):
        for file in f:
            files.append(os.path.join(r, file))

    # Read the content of the file
    text = []
    text_class = []
    for f in files:
        logging.info('reading file: %s', f)
        text_class.append(get_category(f))
```

```
        with open(f, 'r', encoding='utf-8',
            errors='backslashreplace') as reader:
            text_str = reader.read()
            text_str = text_str.encode('utf-8').decode('utf-8',
                'replace')
            # text_str = text_str.replace('\n', ' ')
            text.append(pre_process_text(text_str).strip())

    # Create the dataframe from the data read
    list_of_tuple = list(zip(text, text_class))
    df = pd.DataFrame(list_of_tuple, columns=['text', 'category'])
    # Remove rows with empty text cell
    df = df[df.text != '']
    df.to_csv(output_loc, index=False, encoding='utf8')


def get_category(path):
    """
    Obtain the category of the text by getting it's parent
        directory name
    :param path: the file path
    :return: the category
    """
    split_filepath = path.split('/')
    return split_filepath[-2]


def pre_process_text(string):
    prefixes = ['Xref', 'Path', 'From', 'Newsgroup', 'Subject',
        'Summary', 'Keywords', 'Message-ID', 'Date',
                'Expires', 'Followup-to', 'Distribution',
                    'Organization','Approved', 'Supercedes', 'Lines',
                'X-Newsreader', 'References', 'NNTP-Posting-Host',
                    'In-reply-to', 'Sender', 'News-Software',
                'Article-I.D.', 'Article␣I␣D']
    string_list = string.split('\n')
    new_line = []
    for line in string_list:
        if line.startswith(tuple(prefixes)):
            continue
        else:
            # Remove email addresses
            tmp_line = re.sub('\S*@\S*\s?', '', line)
            # Lower the case
            tmp_line = tmp_line.lower()
            # Remove stopwords
            tmp_line = stopwords_removal(tmp_line)
            # Remove all symbols, retain only alphabets and numbers
            # tmp_line = re.sub('[^A-Za-z0-9]+', ' ', tmp_line)
            tmp_line = re.sub('[^A-Za-z]+', '␣', tmp_line)
```

```
            # Remove all the single letter
            tmp_line = re.sub(r"\s+[a-z]{1}[\s+]", "␣", tmp_line)
            tmp_line = re.sub(r"\s+[a-z]{1}$", "", tmp_line)
            tmp_line = re.sub(r"^[a-z]{1}\s+", "", tmp_line)
            # Remove all extra whitespace
            tmp_line = re.sub(r"\n", "␣", tmp_line)
            tmp_line = re.sub(r"\s+", "␣", tmp_line)
            # Stemming or lemmatization
            # tmp_line = stemming(tmp_line)
            tmp_line = lemmatization(tmp_line)
            # Strip the leading and trailing whitespace
            tmp_line = tmp_line.strip()
            new_line.append(tmp_line)
    new_text = '␣'.join(new_line)
    return new_text


def stopwords_removal(words: str) -> str:
    stop_words = set(stopwords.words('english'))
    more_stop_words = ['article', 'writes', 'write', 'say',
        'nntp', 'posting', 'host', 'berkeley', 'edu', 'hmm', 'll',
                       'wo', 't', 'reply', 'think', 'u', 'go', 've',
                          'repost', 'e', 'mail', 're', 'r', 'o',
                          'hey',
                       'hi', 'n', 'dear', 'reader']
    stop_words.update(more_stop_words)
    tokens = word_tokenize(words)
    filtered_words = [w for w in tokens if not w in stop_words]
    return '␣'.join(filtered_words)


def stemming(words: str) -> str:
    tokens = word_tokenize(words)
    stemmed = [ps.stem(word=w) for w in tokens]
    return '␣'.join(stemmed)


def lemmatization(words: str) -> str:
    tokens = word_tokenize(words)
    pos = nltk.pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word=item[0],
        pos=get_wordnet_pos(item[1])) for item in pos]
    return '␣'.join(lemmatized)


def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
```

```
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN


def main():
    start_time = time.time()
    get_files()
    logging.info("Time␣taken:␣%.2fs", (time.time() - start_time))


if __name__ == "__main__":
    main()
```

APPENDIX B: DATA EXPLORATORY IMPLEMENTATION

The code below is used to check the distribution of the categories in the dataset.

```
import pandas as pd
import matplotlib.pyplot as plt


def input_data_explore():
    input_df = pd.read_csv('../output/20newsGroup18828.csv')

    print(input_df.shape)

    fig, ax = plt.subplots(figsize=(17, 7))
    agg_df =
       input_df.groupby('category').count().sort_values('category')
    print(agg_df)
    fig_plot = agg_df.plot(kind='bar', ax=ax, title='The␣count␣of␣
       each␣category␣of␣the␣dataset',
               legend=False)
    fig_plot.set_xlabel('Categories')
    fig_plot.set_ylabel('Count')

    plt.tight_layout()
    # plt.autoscale()
    # plt.show()
    plt.savefig('../output/count.png', format='png')


def tf_dimension_effect():
    tf_data = {
        'KNN': [0.28, 0.29, 0.31, 0.39, 0.45, 0.49, 0.52, 0.52,
           0.54, 0.55, 0.53, 0.3],
        'SVM': [0.81, 0.81, 0.8, 0.78, 0.77, 0.77, 0.74, 0.71,
           0.68, 0.64, 0.56, 0.27],
        'NN': [0.84, 0.8, 0.8, 0.79, 0.78, 0.76, 0.73, 0.72, 0.7,
           0.69, 0.63, 0.34]
           }

    df = pd.DataFrame(tf_data, index=[8000, 6000, 4000, 2000,
       1000, 500, 250, 200, 150, 100, 50, 10])
    print(df)
    line_styles = [":", "-", "--"]
    fig, ax = plt.subplots(figsize=(17, 7))
    fig_plot = df.plot(kind='line', ax=ax, title='Effect␣of␣
       dimension␣reduction␣on␣accuracy␣(term␣frequency␣with␣SVD)',
                      legend=True, style=line_styles)
    fig_plot.set_xlabel('Dimension')
```

```
    fig_plot.set_ylabel('Accuracy')
    plt.savefig("../output/tfsvd.png", format='png')
    plt.show()


def tfidf_dimension_effect():
    tfidf_data = {
        'KNN': [0.77, 0.77, 0.77, 0.58, 0.55, 0.55, 0.61, 0.65,
           0.67, 0.69, 0.69, 0.55],
        'SVM': [0.87, 0.87, 0.87, 0.86, 0.84, 0.83, 0.8, 0.8,
           0.78, 0.76, 0.74, 0.5],
        'NN': [0.88, 0.86, 0.85, 0.84, 0.82, 0.8, 0.8, 0.81, 0.8,
           0.8, 0.77, 0.59]
           }

    df = pd.DataFrame(tfidf_data, index=[8000, 6000, 4000, 2000,
       1000, 500, 250, 200, 150, 100, 50, 10])
    print(df)
    line_styles = [":", "-", "--"]
    fig, ax = plt.subplots(figsize=(17, 7))
    fig_plot = df.plot(kind='line', ax=ax, title='Effect␣of␣
       dimension␣reduction␣on␣accuracy␣(TF-IDF␣with␣SVD)',
                       legend=True, style=line_styles)
    fig_plot.set_xlabel('Dimension')
    fig_plot.set_ylabel('Accuracy')
    plt.savefig("../output/tfidfsvd.png", format='png')
    plt.show()


def main():
    # input_data_explore()
    # tf_dimension_effect()
    tfidf_dimension_effect()


if __name__ == '__main__':
    main()
```

APPENDIX C: FEATURE EXTRACTION AND DIMENSION REDUCTION IMPLEMENTATION

The code shown in this section is the code that used to generate the feature matrix and apply the dimension reduction to the feature matrix.

```
"""
Move the feature extraction part out of every script for ease of
    maintenance
"""
from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.decomposition import NMF, TruncatedSVD,
    LatentDirichletAllocation
from sklearn import preprocessing
from sklearn import pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from numpy import ravel
import pandas as pd
import numpy as np
import scipy
from scipy.sparse import csr_matrix, csc_matrix
import logging


logging.basicConfig(level=logging.INFO, format='%(asctime)s - %
    (message)s')
pd.set_option('display.width', 320)
np.set_printoptions(linewidth=320)
input_file = "../output/20newsGroup18828.csv"


def generate_tfidf():
    df = pd.read_csv(input_file)
    label = preprocessing.LabelEncoder()
    df['category_id'] = label.fit_transform(df.category)

    # Some details about the data
    # print(df.groupby(['category',
        'category_id']).count().sort_values('category_id'))

    # Limit the features
    tfidf = TfidfVectorizer(analyzer='word', stop_words='english',
        token_pattern=r'\w+', max_features=8000, lowercase=True,
                            use_idf=True, smooth_idf=True)
    x = tfidf.fit_transform(df.text)
```

```
    y = df['category_id']

    x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size=0.2, random_state=42)
    # To make space in memory
    del df
    return x_train, y_train, x_test, y_test


def generate_tfidf_reduced(factor: int):
    df = pd.read_csv(input_file)
    label = preprocessing.LabelEncoder()
    df['category_id'] = label.fit_transform(df.category)

    # Some details about the data
    # print(df.groupby(['category',
        'category_id']).count().sort_values('category_id'))

    # Limit the features
    tfidf = TfidfVectorizer(analyzer='word', stop_words='english',
        max_features=8000, lowercase=True,
                            use_idf=True, smooth_idf=True)
    # tokenizer=r'\w+'
    x = tfidf.fit_transform(df.text)
    y = df['category_id']

    print(x.shape)
    mat = x.tocsc()
    greaterThanOne_cols = np.diff(mat.indptr) > factor
    new_indptr = mat.indptr[np.append(True, greaterThanOne_cols)]
    new_shape = (mat.shape[0],
        np.count_nonzero(greaterThanOne_cols))
    x2 = csc_matrix((mat.data, mat.indices, new_indptr),
        shape=new_shape)
    x2 = x2.tocsr()
    print(x2.shape)

    x_train, x_test, y_train, y_test = train_test_split(x2, y,
        test_size=0.2, random_state=42)
    # To make space in memory
    del df
    return x_train, y_train, x_test, y_test


def generate_tfidf_svd(no_of_features: int):
    df = pd.read_csv(input_file)
    label = preprocessing.LabelEncoder()
    df['category_id'] = label.fit_transform(df.category)

    # Limit the features
```

```
    tfidf = TfidfVectorizer(analyzer='word', stop_words='english',
       token_pattern=r'\w+', max_features=8000,
                           lowercase=True,
                           use_idf=True, smooth_idf=True)
    x = tfidf.fit_transform(df.text)
    y = df['category_id']

    svd = TruncatedSVD(n_components=no_of_features, n_iter=7,
       random_state=42, tol=0.0)
    x_reduced = svd.fit_transform(x)

    x_train, x_test, y_train, y_test = train_test_split(x_reduced,
       y, test_size=0.2, random_state=42)

    return x_train, y_train, x_test, y_test


def generate_tf():
    df = pd.read_csv(input_file)
    label = preprocessing.LabelEncoder()
    df['category_id'] = label.fit_transform(df.category)
    count_vect = CountVectorizer(analyzer='word',
       stop_words='english', max_features=8000, lowercase=True)
    x = count_vect.fit_transform(df.text)
    y = df['category_id']

    x_train, x_test, y_train, y_test = train_test_split(x, y,
       test_size=0.2, random_state=42)

    return x_train, y_train, x_test, y_test


def generate_tf_reduced(factor: int):
    df = pd.read_csv(input_file)
    label = preprocessing.LabelEncoder()
    df['category_id'] = label.fit_transform(df.category)
    count_vect = CountVectorizer(analyzer='word',
       stop_words='english', lowercase=True)
    x = count_vect.fit_transform(df.text)
    y = df['category_id']

    print(x.shape)
    mat = x.tocsc()
    greaterThanOne_cols = np.diff(mat.indptr) > factor
    new_indptr = mat.indptr[np.append(True, greaterThanOne_cols)]
    new_shape = (mat.shape[0],
       np.count_nonzero(greaterThanOne_cols))
    x2 = csc_matrix((mat.data, mat.indices, new_indptr),
       shape=new_shape)
    x2 = x2.tocsr()
```

```
    print(x2.shape)

    x_train, x_test, y_train, y_test = train_test_split(x2, y,
        test_size=0.2, random_state=42)

    return x_train, y_train, x_test, y_test


def generate_tf_svd(no_of_features: int):
    df = pd.read_csv(input_file)
    label = preprocessing.LabelEncoder()
    df['category_id'] = label.fit_transform(df.category)
    count_vect = CountVectorizer(analyzer='word',
        stop_words='english', max_features=8000, lowercase=True)
    x = count_vect.fit_transform(df.text)
    y = df['category_id']

    svd = TruncatedSVD(n_components=no_of_features, n_iter=7,
        random_state=42, tol=0.0)
    x_reduced = svd.fit_transform(x)

    x_train, x_test, y_train, y_test = train_test_split(x_reduced,
        y, test_size=0.2, random_state=42)

    return x_train, y_train, x_test, y_test
```

APPENDIX D: KNN CLASSIFICATION MODEL IMPLEMENTATION

The code for kNN classification model is shown below.

```
from GenTfIdf import generate_tfidf, generate_tf,
    generate_tf_reduced, generate_tfidf_reduced,
    generate_tfidf_svd, \
     generate_tf_svd
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import numpy as np
import time
import math

pd.set_option('display.width', 320)
np.set_printoptions(linewidth=320)

start_time = time.time()
x_train, y_train, x_test, y_test = generate_tfidf()
# x_train, y_train, x_test, y_test = generate_tfidf_svd(50)
# x_train, y_train, x_test, y_test = generate_tfidf_reduced(50)
# x_train, y_train, x_test, y_test = generate_tf()
# x_train, y_train, x_test, y_test = generate_tf_svd(10)
# x_train, y_train, x_test, y_test = generate_tf_reduced(1000)
print(x_train.shape)

k = int(math.sqrt(x_train.shape[0]))
# Rule of thumb for k is sqrt(sample size)
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(x_train, y_train)

predicted = classifier.predict(x_test)

result = confusion_matrix(y_test, predicted)
print(result)
accuracy = accuracy_score(y_test, predicted)
print("accuracy score: " + accuracy.astype(str))
report = classification_report(y_test, predicted)
print(report)
print('Total time taken: {0:.2f}s'.format(time.time() -
    start_time))
```

APPENDIX E: SVM CLASSIFICATION MODEL IMPLEMENTATION

SVM classification model implementation is shown as follows.

```
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
from sklearn.svm import LinearSVC
import time
from GenTfIdf import generate_tfidf, generate_tf,
    generate_tf_reduced, generate_tfidf_reduced,
    generate_tfidf_svd, \
     generate_tf_svd
import pandas as pd
import numpy as np

pd.set_option('display.width', 320)
np.set_printoptions(linewidth=320)

start_time = time.time()
x_train, y_train, x_test, y_test = generate_tfidf()
# x_train, y_train, x_test, y_test = generate_tfidf_svd(500)
# x_train, y_train, x_test, y_test = generate_tfidf_reduced(500)
# x_train, y_train, x_test, y_test = generate_tf()
# x_train, y_train, x_test, y_test = generate_tf_svd(500)
# x_train, y_train, x_test, y_test = generate_tf_reduced(500)
print(x_train.shape)

clf = LinearSVC(random_state=0, tol=1e-5)
clf.fit(x_train, y_train)

predicted = clf.predict(x_test)
result = confusion_matrix(y_test, predicted)
print(result)
accuracy = accuracy_score(y_test, predicted)
print("accuracy score: " + accuracy.astype(str))
report = classification_report(y_test, predicted)
print(report)
print('Total time taken: {0:.2f}s'.format(time.time() -
    start_time))
```

APPENDIX F: NN CLASSIFICATION MODEL IMPLEMENTATION

NN classification model implementation is shown as follows.

```
from GenTfIdf import generate_tfidf, generate_tf,
    generate_tf_reduced, generate_tfidf_reduced,
    generate_tfidf_svd, \
     generate_tf_svd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
import time
import pandas as pd
import numpy as np

pd.set_option('display.width', 320)
np.set_printoptions(linewidth=320)

start_time = time.time()
x_train, y_train, x_test, y_test = generate_tfidf()
# x_train, y_train, x_test, y_test = generate_tfidf_svd(500)
# x_train, y_train, x_test, y_test = generate_tfidf_reduced(500)
# x_train, y_train, x_test, y_test = generate_tf()
# x_train, y_train, x_test, y_test = generate_tf_svd(500)
# x_train, y_train, x_test, y_test = generate_tf_reduced(500)
print(x_train.shape)

# Have to manually interrupt it to produce result
clf = MLPClassifier(tol=1e-3)
clf.fit(x_train, y_train)

predicted = clf.predict(x_test)

result = confusion_matrix(y_test, predicted)
print(result)
accuracy = accuracy_score(y_test, predicted)
print("accuracy score: " + accuracy.astype(str))
report = classification_report(y_test, predicted)
print(report)
print('Total time taken: {0:.2f}s'.format(time.time() -
    start_time))
```