UNIVERSITY OF HERTFORDSHIRE
School of Computer Science

BSc Honours in Computer Science

6COM1056 - Software Engineering Project

Final Report
April 2019

Categorising Movies into Genres Using Subtitles

R.C. Bromfield

Supervised by: Raimund Kirner

Abstract

This project explores the categorisation of movies using their subtitle. Subtitles are usually found in text files which means that they can be classified using document classification techniques. This repot delves into the subjects of word embedding, document classification techniques and Support Vector Machine. As well as offering a novel to the term frequency–inverse document frequency formula in respect to finding the importance of words within a corpus.

This report finds that the new formula created outperforms the term frequency–inverse document frequency formula in all but one genre. And the use of a SVM allows for the categorisation of some genres but not others.

Contents

Chapter 1 -Introduction

1.1 – Overview

Natural language processing is an ever-growing field within computer science. The study of how computers can be used to understand and categorise human language dates back to the 1950s and yet is still growing to this day. Computer categorisation of human concepts can be difficult as often the human language is seemingly random and complex meaning that ideas and concepts can be challenging for machines to grasp. However, this report will look into how a program may categorise movies into different genres using a subtitle file of the words spoken in said movie.

1.2 – Research Questions

Before starting this project, the author of this report started with the intention of answering these four questions:

1. Using subtitles is it possible to predict a movies genre?
2. Which genres are more easily classified?
3. What is the best method to categorise large corpora of text?
4. Can a similar process be used to determine other characteristics?

By the end of this project, the author of this paper hopes to have answered these questions either through research of other papers in this field or through experimentation on the project itself.

1.3 – Motivations

As online entertainment grows as does the need for categorisation of said entertainment. With movies, the standard way to categorise them is by sorting them into genres. These genres are sorted via themes and ideas which are present in the text, and therefore this project hypothesises that it is possible to gather said genres via the text alone. With the increasing accuracy of speech recognition automatic subtitling, this may be used to categorise films which no meta-data easily.

With movies often come subtitle which for the viewer appear at the bottom of the screen these subtitles will often come in a text file often with the extension ".srt". These are created from lines of dialog spoken in the movie with a time stamp which will tell the video player when the line is meant to be displayed.

As the purpose of this exercise is only to look at the text, the program will ignore these timestamps, and the lines of dialogue will be tokenised into unigrams meaning that an array of single word elements will be created.

1.4 – Report Structure

There are eight parts to this report including this current chapter. As well as these chapters there are many figures all of which have a corresponding larger figure in Appendix A.

This report will attempt to explain and analyse the processes of creating a machine learning program to classify movies into their correct genres via their subtitles.

The report will involve the following chapters:

Chapter 2 – A literature review of the previous research into classifying movies into genres using machine learning techniques.

Chapter 3 – A detailed description of the work that proceeded the building of this program and the gathering of the data.

Chapter 4 – Key concepts this project explored and analysed.

Chapter 5 – A look into natural language processing and how those concepts influenced this work.

Chapter 6 – Will look at the creation of the SVM models and the building of the system.

Chapter 7 – Will evaluate the results given from the system and analyse what these results mean.

Chapter 8 – Will conclude the work looking at how well this project answered the hypothesis and research questions along with what could be done in the future to build off of this work.

The results of this project will be written in this report in four significant figures when needed for the sake of simplicity and readability.

Chapter 2 – Related Work

In ‘Movie Classification Using SVM with Audio and Video Features’ by Yin-Fu Huang and Shih-Hoa Wang (2012) explores the topic of using machine learning techniques to classify movies into their genres. However, unlike this project, this conference paper attempts to do so via the audio and video output of the film. Here they utilise a process first discussed in their other paper ‘Self-adaptive Harmony Search Algorithm for Optimization’ (2010).

The paper outlines an extensive and methodology which seems to gather reliable results as their method gains a 91.9% accuracy rate. However, it should be noted that the experiments are on movie trailers, not the movies themselves. While the same logic may be applied to movies and movie trailers as data, they are very different.

This is because movie trailers are cut and written to give a clear, distinct idea of what the movie is in a limited space of time. Due to this, trailers are shot differently often having distinct editing styles and music choices based on their genre. There will be less irrelevant data than there would be in the movie itself hence less chance of misclassification making it easier to classify movie genres rather than the movies themselves.

While it stands to reason that if it were possible to classify movie trailers, then it should be possible to classify actual movies this does make the title somewhat of a misnomer as the authors are classifying movie trailers can classify movies themselves.

A predating paper focusing only on the audio component only can be found in the ‘Journal Of Computer Science And Applications’ named ‘Audio Based Movies Characterization Using Neural Network’ in Jain and Jadon (2008). Here like in Huang and Wang (2012) no linguistic data is used to classify the documents instead the author uses techniques such as Volume Standard Deviation, Volume Root Mean Square and Zero Crossing Rate to analyses pitch. An issue this article, however, is that movie genre themselves is not compared as the experiment only attempt to classify whether the movie is an action movie or not an action movie. Later in this report, will cover the issues with such a technique as some genres are easier to classify than others.

As much of the analysis is on volume and pitch it is understandable that this method would be more productive with this genre more than others as action movies often have loud noises, upbeat music and explosions making it likely easier to classify via this method. However, Jadon and Jain (2010) do not attempt to find if this method would be as efficient on other genres such as biographies or romance films. Due to this, it may be hard to find a practical application to this work as any genre classification system would need to do more than classify one particular genre.

Nevertheless, this paper does show the potential is using audio in the classification of movie genres and the methodology and results are completed concisely and interestingly. The authors of this article would take the year after following this by looking at both the video and audio parts of a movie. In their conference paper ‘Movies genres classifier using neural network’ (2009). However, this took a different approach to Huang and Wang (2012) in that a neural network was used instead of an SVM building on this they extracted visual features via many aspects such as motion computation as well as colour dominance; building on the previously established audio feature selection methods.

A minor issue with these papers is that they both state the statistic that every year over 4500 movies or 9000 hours movies is released each year. Which was seen throughout many of the papers dealing with this subject these are both taken though in the case of Huang and Wang (2012) via the paper ‘The Challenges of Continuous Capture, Contemporaneous Analysis, and Customized Summarization of Video Content’ by Howard D. Wactlar (2001). From the executive summary by Berkeley University called ‘How much Information’ from the year 2000. While these statistics are not crucial to the

papers, it is probably worth noting that this data was likely very out of date at the time of writing. Also that the webpage where this information was hosted was taken down the same year Huang, and Wang (2012) was published and now only survives as a pdf of the website in Berkeley University's archives.

Chapter 4 – Concepts

4.1 – Document Classification

The concept of document classification may very well be as old as the written word and is most likely at least as old as the first library. As document classification is in many ways a sub-category of library science, we can see that the problem of where to put specific categories of text is an old one.
With the rise of computers, this becomes both much more efficient and much more needed as the number of documents available rises exponentially.

Due to this, we can see a significant increase in Automatic Document Classification (ADC) where programs use different kinds of methods to classify documents. These come in three types these being. Supervised document classification where an external actor such as a user or database trains the program on what the correct category in using labelled data the machine can learn the categories (Krithara *et al.*, 2018).

Unsupervised document classification where clusters found where documents are alike in a way without an external actor to give labels to the classes found. Semi-supervised Classification as the name would suggest is a mix of both types where only some of the data is labelled, and it is up to the program to find which data point belongs to which cluster (Krithara *et al.*, 2018).

As this project aims to find which movies are part of already pre-established groups which through the OMDB API it is possible to label the data with the said group this would make this task a supervised document classification problem.

4.2 – Word Embedding

Word embedding is a term used for a set of modelling techniques mapping words or phrases to a multi-dimensional vector to use for machine learning. The concept of representing a word in vector space can be dated back to at least Rumelhart, Hinton and Williams (1986). However, while we now use word embeddings for machine learning tasks that are used in our everyday life, it should be noted that at the time the authors of this paper were trying to model how learning may happen in the brain and do not spend much time on the word vectors themselves.

Around a decade later however Bellegarda (1998) outlined something similar in Latent Semantic Analysis (LSA) while these matrices still held no real semantic meaning it used a count based algorithm which displays a vector based on the number of words per paragraph.

It was not however until Bengio *et al.* (2003) however introduced a Neural Network Language Model (NNLM) that allowed for a language to be genuinely modelled into a vector space as this would use a neural network to predict the probability of the final word in a sequence given the ones before it.

This would have many word embedding methods build on this however as of writing the widely used was developed by Mikolov *et al.* (2013) is known as Word2Vec and can be found in the scikit-learn python library.

4.2.1 – How Word2Vec Works

There are two methods used by word2vec to create these vectors both of which uses distributional semantics. The concept behind this, however, is best summarised in a quote the quote “You shall know a word by the company it keeps” by Firth (1957) to expand on this the method works via working out the probabilities based on the words around it. In Word2Vec this is done via two algorithms the first being the Continuous Bag Of Words (CBOW) named as the order of the words itself does not matter which takes the surrounding words and tries to predict the word itself using a shallow neural network. The second can be seen as the inverse of this and is called the Skip-Gram model (Mikolov *et al.*, 2013). Here the neural network attempts to calculate the context words given a

centre word. Using the Skip-Gram model, we can calculate the probability of $p(w_{t+j}|w_t)$ with the formula:

$$p(o|c) = \frac{\exp(u_o{}^T V_c)}{\sum_{w=1}^{V} \exp(u_w{}^T V_c)}$$

Where v_c and u_o are the centres and outside vectors respectively, from here, we can see that each word has two vector representations at this point one v for when it is at the centre and u for when the word is represented in context.

The architecture of both models can be seen in Figure 1 from Mikolov *et al.* (2013) as this figure illustrates both models are necessarily the inverses of each other and yet work for the same outcome.

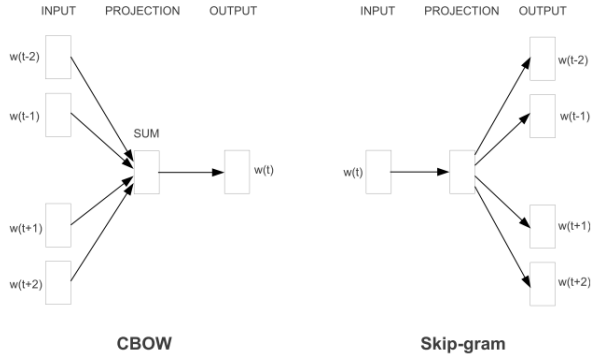


Figure 1 - CBOW and Skip-gram architecture

4.2.1 – What is Word2Vec

Word2Vec is a model for creating such word embeddings. Here words are turned into vectors based on their semantics in many aspects; each aspect is given a vector which is changed via its meaning. This means that words which are similar in meaning have a smaller Euclidian distance than words which are less similar and also means that words can have maths performed on them based on their semantics.

An example of this would be looking at the difference between the words ‘man’, ‘woman’,’king’ and ‘queen’. These words can have similar means or different meaning based on their context. Take for example the phrase “The *king* ruled over the country” the word king could easily be replaced with the word queen, and the sentence would still hold the same meaning, and these words are more likely to appear in that sentence than ‘man’ or ‘woman’. Given this, a program could be mistaken in thinking that the two were synonymous. However, this may cause issues with the phrase “The *queen* gave birth to a beautiful baby” here king and queen are not synonymous as if ‘queen’ was replaced with king this would imply either some miracle of biology or a transgender monarchy both of which are rare. However, if ‘queen’ is replaced with ‘woman’ then the sentence is again become a lot more ordinary.

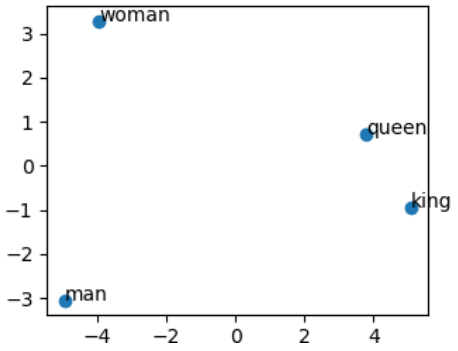


Figure 2 - PCA Diagram of words "woman", "man", "queen", "king"

Hence from here, we can gather that ‘king’ and ‘queen’ are close in one aspect and ‘queen’ and ‘woman’ are close in another. Given this, we can map these words to a vector using these aspects and then group them. This can be seen when using PCA analysis on these words using word2vec in Figure 2 here the more positive the x-axis, the higher the status and the higher the y-axis, the more feminine the word.

Due to these properties this leads to a case where mathematics can be performed on words a prominent example by the which can be found in Mikolov, Yih and Zweig (2013) is when converted to vectors “king” – “man” + “woman” equals a vector nearest to “queen”. This example can be seen to be true when looking at Figure 2.

4.2.3 – Potential Issues

However, the nature of word2vec leads to the issue that a word not previously used to train the model cannot be turned into a vector. The list of words that can be used with a model is called a vocab and cannot be updated once the model is trained.

Due to this if more words are to be added to the vocab the model must be trained again which can lead to further problems as the vectors for each world will be different each time a new model is created. This led to a problem while creating the SVM as it been trained on an old model and therefore did not work with the word vectors of the new one. An example of the difference between the vectors of the two models can be seen with the word ‘apple’ which has a Euclidean distance of around 5.07 between the old and new models.

4.4 – Support Vector Machines

A Support Vector Machine (SVM) is a method for supervised learning which provides a binary classification of data. This means that given a set of vectors and labels for each vector an SVM can be trained to see where distinctions can be made between the two sets. This is done by first finding the support vectors which are points which are closest to the opposing data. Then two hyperplanes are drawn between these support vectors with the maximum distance between them after the decision boundary is drawn directly in the middle of these two hyperplanes.

The term hyperplane is being used as the data can of high dimension as it is when using Word2Vec vectors. However there another reason why the decision boundary may be of higher dimension this being that if the data is not linearly separable, then it would be impossible to draw a straight line to classify them. However, if the data is projected onto a higher dimensional space, then it is possible to draw a linear hyperplane which intersects with the feature space which is now folded into a higher dimension. This is what is called the ‘kernel method’ and because of this means that we can classify data non-linearly using different kernels (Boser, Guyon and Vapnik, 1992).

There are several kernels which can be used when creating a support vector machine; the linear kernel is the simplest as it only draws a linear decision boundary without projecting the data to a higher dimension. This would be unfeasible for this project as much of the data intersected, and therefore this would be unlikely to be easily classified. Of the other non-linear kernels tested it was shown that the Gaussian Radial Basis function would serve the best.

This can be characterised by the formula $K(x, x') = \exp(-\lambda \| x - x' \|^2)$ however with this formula λ or simply written as gamma is an adjustable parameter. If this is set too high, then there is a risk of overfitting however if it is too low then the function may become too similar to a linear classifier as the size of the gamma is inverse to the size of the curve it creates meaning that a small gamma can be less useful for highly mixed data.

As this data is quite mixed, this would imply a high gamma score for the data which was proven right when creating many SVM models for different genres and then testing them finding that $\lambda = 2048$ was the best parameter.

As SVM is binary classifier and there will tell if something does or does not fall into a particular category this project uses a different model for each genre rather than just training one model on each. While it is possible to have an SVM categorise data into more than two categories movies may have more than one genre and therefore require more than one output. Also, this would require making more decision boundaries which may cause more false positives or negative.

Instead, the genres have been iterated through, and then for each genre, it takes the top two-hundred words of each subtitle file, Using the OMDB API the program checks if the movies are of that genre. If so then it is added to the genre top words array. Otherwise, it is added to the not genre top words array. These arrays are then used to create the SVM.

After this, the model can then be used to find if a particular word is likely to fall into that genre or not. For instance, when checking the word "Cowboy" into the 'Western SVM', it will provide a positive result whereas the word 'Alien' will only provide a positive result if compared to the 'Sci-Fi SVM'. However, as natural language is random, only a certain percentage of words will be positive regardless of if the movie falls into that genre or not and false positive can also arise on movies that are not of that genre.

However, if the movie is of that genre, then it will have a higher percentage than if it is not. Therefore, a second SVM can be used to find the threshold needed to categorise the movie correctly.

Chapter 5 – Natural Language Processing

5.1 – The Bag-of-words Model

A bag of words model is a method of natural language processing which focuses on the individual words of a text rather than its sequence or grammar. While this may mean some meaning is lost it also provides a fast and efficient way of classifying text. These models usually focus on word distributions and work to figure which words are more important than others.

For this task, it can self-evident why a model such as this one would be useful. If for instance, a subtitle file uses the word "alien" many times within its text then it is likely to be a Sci-Fi movie. However, if the word "love" appears many times, then there is a high chance it is a romance movie.

An issue with this can, of course, be that that words may be slightly different and yet mean the same thing the obvious example to this when words are pluralised by adding an "s" or made past or future tense if we take the verb talk for instance. Without taking into account the information of when the action is taking place it can be agreed that "talked", "talk" and "talking" mean the same thing. If counted without lemmatisation, it would be gathered that each word is worth up to a third of its overall significance as the frequency is being split between each word.

This is where lemmatisation comes in as the process will convert each of these words into a single word that represents all three words and hence the concept itself. This is similar to the concept of stemming turns a word into a root word; however, with this concept, the word itself can often not be a real word. An example of this would be the words "laziness" and "lazy" both of which are stemmed to "lazi" but lemmatised too "lazy" (Lovins, 1968). This means more semantic significance can be given to words after lemmatisation.

Bag-of-words models usually involve what is known as tokenising a corpus which breaks it down into n-grams. Then what is known as stop-words are taken out these are words which are regarded as semantically meaningless such as "the", "you", "be" and so on. After this, each word is counted, and the meaning of the text is derived from how many of which word is the highest or what it shares in common with other texts.

While this can be useful for small bodies of texts with a little variation the word with more randomised text, this is less useful. The first issue is that stop-words are usually taken from an array and are not generated via the machine itself; instead, they are a group of words previously decided to be semantically meaningless.

This can cause issues, firstly in that this means these stop words will have to be changed depending language but also when these words are pre-decided it can mean that the program is unadaptable. For instance, if the word "you" appeared in 90% of the corpus it would be significant; however, with just removing stop words, the program would never notice this. In Chapter 5.3 this report will look at two possible methods to solve this problem as well as making this process automatic regardless of language.

Another issue is that with large corpora which may contain many words that are not relevant for classification this method can be less than useful as while similar words may be shared between texts the exact words may be low hence the need for more sophisticated methods as seen in Chapter 5.3.1.

5.2 – N-grams

An n-gram model is a model within natural language processing where a corpus is split into sections of n words. This program will only use uni-grams where n = 1 this because of a few reasons. The first being that storing and categorising bi-grams would increase both space and computational time by what could be the significant amount as if we take d as the total about of words in a corpus then using a bigram model, we increase the number of n-grams or phrases potentially by up to d^2.

The second reason is that there is a large amount of research that bi-grams or other higher n-grams are either not significantly effective or are less effective. For instance, in Pang, Lee and Vaithyanathan (2002) we can see that unigrams outperformed bigrams by 82.9% to 77.1% respectively when using an SVM. When combining unigrams with bigrams so that phrases such as "Computer Science" are not split into "Computer" and "Science" but less relevant phrases are kept as unigrams the results are more mixed Tan, Wang and Lee (2002) found a 1.3% increase in precision when using this method on the Yahoo-Science Corpa however with Pang, Lee and Vaithyanathan (2002) 0.3% decrease when using this method. Either way, a 1.3% increase would not justify the added computational expense.

5.3 – Inverse Document Frequency

Due to this a formula derived from looking at word distributions from large bodies of text. Zipf's law states all large corpora follow a similar distribution no matter which language this distribution can be characterised that $r \times R$ roughly equals a constant. As all languages follow this distribution and roughly all languages have their stop words as the word which appear with the highest frequency, ,it stands to reason that meaning can be derived by seeing how the word distribution of a single text differentiates from a large word distribution. Hence importance can be derived from the formula.

$$\left(\frac{1000}{P_n + P_s} \times P_s\right)$$

Here P_n is the probability of the word in significance to the word frequency and P_s is the probability of the word in word frequency of the subtitle file. This not only shows which words are used the most but will mean that any numbers lower than five hundred are used less frequently than the average. Which numbers the rankings fall between can be changed by changing the constant '1000', and it may be wise to minus half that constant to make the words which are less frequent than average negative however for this program this was not necessary. However, a further increase in accuracy can be found by removing the subtitle files of the genre being tested from the set n.
When using this with the subtitles for the file Star Trek Beyond, we can see its effectiveness. Words that appear in no other place but this file are given the score 1000, but also words which people would highly associate with Star Trek though are not unique to it have very high scores as well. For instance, the word 'federation' has a score of 999.47, 'unified' 999.19 and more scientific words which may connote a sci-fi movie but are not special to just Star Trek can be seen around the 995 such as 'electromagnetic' and 'plasma'.

After further reading; however, it became apparent that this issue had already been solved by Jones (1972) in this paper the author outlines a different method to rank the importance of word within the text.

$$w_{t,d} = tf_{t,d} \log\left(\frac{N}{df_t}\right)$$

Here instead of counting the overall frequency of the words in all of the dataset instead the number of documents that use these words are counted and then divide by the number of overall documents.

This, however, could be inefficient for this task as due to subtitles containing many more words than something like a search term where this is mainly utilised it is much more likely for df_t to be higher than it should be. Take for instance the term "love" this may be mentioned in every movie at least once and therefore make $N \approx df_t$ which would put the word on the same level as a word such as "the". However, this word could be important when checking if a film falls into the genre of 'romance'. Due to this SVM models have been created with both formulas and in Chapter 7 – Evaluation both sets of results will be compared with each other.

Issues with both formulas, however, can arise as if the word is misspelt or otherwise is wrongly considered unique to the subtitle file then it will be given a high score and therefore be the most significant. If this is not caught this can cause outliers as these vectors are usually far removed from the rest with the SVM though these can be removed by using z-score outlier detection before training and testing.

5.3 – First Experiments

5.3.1 – Comparing Words Without SVM

A bag-of-words model was first used to classify the subtitles into genres. This model would add the frequencies of each word then compare them to a word frequency vocabulary of each genre and see which word frequencies were the most similar.

To test this idea, CSV files were created which held individual words in its first column and the frequency of those words in its second. Then in the folder 'word_freq' similar CSV files were created that held the word frequencies of the subtitles files.

The function getGenreScore() was then created which when passed an IMDb ID would create a sorted dictionary where the keys were the genres, and the values were the sum of all the scores of each word. Then the lowest scores were checked against the genres given by OMDB API, and if for instance, the lowest score matched the genre given to it via API it was considered a correct if not wrong.

When running against the training set, this experiment has a 70% success rate while this may not be high it should be noted this was out of 22 different genres so the likelihood of success of random chance would be roughly 4.54%. This showed a proof of concept which unfortunately did not hold when testing against the testing set which had not had their word frequencies added to the different genres.

The reason this would seem to be despite common words being found which depend on genres most words would seem to be different enough that they just comparing the words on their own would be meaningless at least with the current dataset.

Due to this, it seems clear that using a Support Vector Machine may be better with an embedding which will give similar values based on meaning. Hence the move to Word2Vec however this algorithm would then be used in isolating the top two hundred words which are explained in the next chapter.

5.3.2 – Checking Data with PCA Analysis

The first step in seeing if it was possible to linearly group movies via their subtitles using a Support Vector Machine was to put the unigrams of two different movies through PCA analysis to see if any

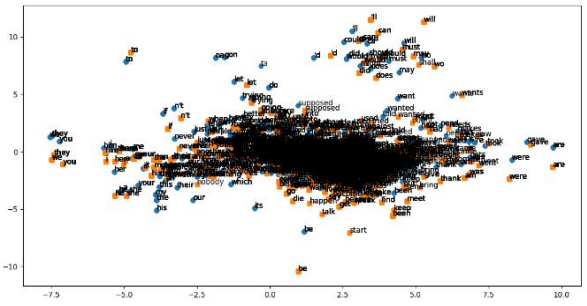


Figure 4 - PCA Diagram of all words in Star Trek Beyond (orange) and Gandhi (blue)

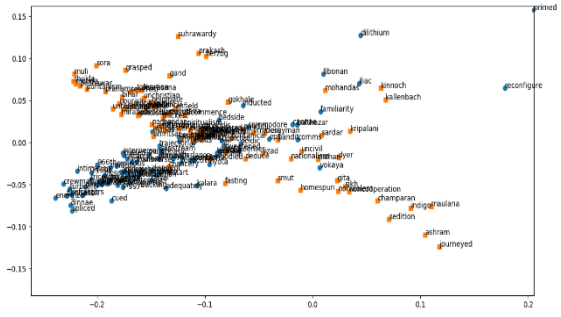


Figure 3 - Top hundred words from Star Trek Beyond (orange) and Ghandi (blue)

clusters arose. For this the words of Star Trek Beyond (2016) and Gandhi (1982) were chosen due to them being very different movies if this possible at all it should be evident most with these two. However, the issue spoken about in Chapter 4.1 arises that most words in a corpus do not signify meaning this can be seen in Figure 4 where a PCA diagram has been created of all of the words from both movies. Here it can be seen there is no clear separation between the two and most of the words are over each other showing no distinct way of separating them.

From here it is evident that only the words which hold the most meaning to the text should be used for classification. Using the formula $\left(\frac{1000}{P_n+P_s} \times P_s\right)$ again it was possible to find the top 100 words of both files. Hence in Figure 3, it can be seen that these create two different distinct clusters which can be easily identified.

More improvement, however, can be made as if instead of comparing all the words to the word frequencies of all the text we compare them to all the words minus those of genre Sci-Fi we can see can improve the accuracy of choosing words which are more science fiction like hence enabling more straightforward classification. In Chapter 7 – Evaluation it can be seen that when this is done much easier clusters can be made out; meaning that classification is much simpler.

From here the next move was to try comparing words of a particular genre vs words which was not of that genre. With these, the top two hundred words of each subtitle file were added either a Sci-Fi array or a Not Sci-Fi array. After this a PCA diagram was created that can be seen in Figure 5 this diagram has much more overlap however it still distinct clusters can be seen and hence classification with a Support Vector Machine is more likely.

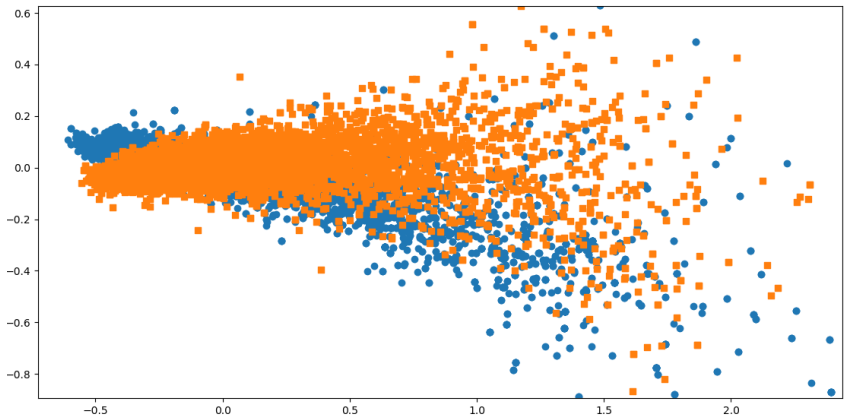


Figure 5 - PCA Diagram of Sci-Fi (orange) and not Sci-Fi (blue)

Chapter 6 – Creating SVM Models

6.1 – Downloading Subtitles

To gather the dataset the previously mentioned OpenSubtitles API was utilised along with a dataset downloaded from IMDB which gave a list of IMDB IDs which were movies as opposed to TV Series or Short Films. With this, over 1200 subtitles were able to be downloaded and then sorted into different genres via the OMDB API.

After these, another 100 subtitles were downloaded for each genre to be split in half for both training the second layer and then for overall testing. Another 100 were also downloaded to test against the subtitles which were of the genre.

While a relational database could have been used for this application, it seemed much more efficient and more efficient to use simple Comma Separated Files as they were easily readable and could be easily copied from one system to another making use of the computer cluster much easier

6.2 - Finding Word Frequencies

After the training set was downloaded, the word frequencies were gathered. Here a dictionary datatype was utilised which held the word as its key and incremented by one each time it was found in a subtitle file. For the inverse document frequency formula, this was replaced by a simple if statement which would then increment the element by one if the word was in the document at all.

Two functions were created for these tasks to keep the data persistent and allowing for the CSV files to be easily read and written to these were csvToDictionary() which would read a CSV file then turn the contents into a dictionary and dictionaryToCSV() which performed the opposite operation.

6.3 - Adding word frequencies to not genre CSV files

To use the formula outlined in this report, a ‘not genre’ word distribution must be found. This is the total word frequencies of all subtitle files which are not in the genre which is being tested. To do this, the OMDB API was used to find the genres that the movie was in this was then turned into an array of which the elements of was subsequently removed from an array of all the genres.

From here the word frequencies were added to the CSV files of these genres which could be found in the ‘not genre’ folder. These CSV files were not used when using the inverse document frequency formula however as with this formula only the count of how many times the document appears is necessary not the word frequency itself.

6.4 - Creating ‘top words’ arrays

After this, it is possible to find the most important words of the subtitles using both formulas. Then the top two-hundred words were taken from each subtitle and added to an array. Each genre had two arrays associated with it. These were an array of the top words of subtitle files that fit in that genre and an array of the top words which were not in the genre for purposes of training. These arrays were then saved using the python serialisation technique known as pickling which allowed the arrays to be read quickly. Creating CSV files of these arrays was less necessary as reading the data in the arrays was less necessary for debugging purposes.

6.5 - Using ‘top words’ arrays to create the first layer

Once these arrays were created, they are then used to train the Support Vector Machine. The arrays are cut so they are of equal size and then iterated through so that they can be turned into vectors using Word2Vec. This can take several hours in not a day to run as these models take a long time to train the computer cluster was used so that the software could be made multithreaded so these models could be created in parallel.

Once they were created, it could be seen that with the best values for all genres was a cost value of 10 and a gamma value of 2048. The high gamma value indicates how well mixed the data was even utilising the importance formula.

However, once created the model enables for the classification of single words, not whole corpora. For instance, if the word “Cowboy” is given to the SVM for Western, then it will return positive. If, however, it is given the word “Alien” it will return negative.

While this is good for finding which words can be signifiers for a particular genre, it does mean when given a list of 200 words from the subtitle files only a certain percentage will be classified as positive to the genre as only a certain amount of them will be genre signifiers. Another issue is that false positives can arise from the algorithm when testing the first layer against subtitles not of that genre making this more difficult to classify. Due to this, a second SVM was needed to interpret the results that were given from the first.

6.6 – Creation of the Second Layer

As the number of false positives is lower than the number of true positives for all subtitles, this means that any subtitle file which has several positives above a set threshold is most likely in the genre that is being tested.

Due to this, it is possible to use a second SVM model this time only needing a linear kernel. By feeding it the percentage, the first layer perceives as positive for both subtitle files that are of the genre and those which are not the model can predict which genre the subtitle file fits into overall hence creating the second layer. This means that when the two layers work together, they can give a single binary output which says if the movie is in the genre or not.

Chapter 7 – Evaluation

7.1 – Evaluation Metrics

The second layer of the program was then given roughly the same number of subtitles which it would then have to classify. In Chapter 7.3 – Comparison and Results the results of both formulas being compared can be seen in a confusion matrix. This a type of table that visualises the performance of a machine learning task breaking the results down into:

- True Positives (TP) – Results that were categorised as the genre and were, in reality, the genre.
- True Negatives (TN) – Results that were categorised not of the genre and were in reality not of the genre.
- False Positives (FP) – Results that were categorised as the genre but were not.
- False Negatives (FN) – Results that were categorised as not being in the genre but were in the genre.

Due to this project classifying many genres the confusion matrix is not set up in its usual 2x2 or NxN configuration and instead has columns of TP, TN, FP, FN along with the calculated performance measurements.

7.1.1 – Recall and Precision

The two standard measurements recall, and precision for information retrieval systems was first outlined in Kent *et al.* (1955).

The first of these measurements is known as recall. This the fraction of documents that are successfully classified by the program due to this with binary classifiers this can also be known as the measure of sensitivity. The equation for which is: (Davis and Goadrich, 2006)

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

For a binary classifier such as this program, this can be seen as the probability of retrieved relevant items in relation to all relevant items and therefore can be seen as measuring the effectiveness in retrieving or including relevant items (Buckland and Gey, 1994).

Precision, on the other hand, is the relevant items over the overall number of retrieved items hence can be taken as a measure of excluding non-relevant items and can, therefore, be calculated like this: (Davis and Goadrich, 2006)

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

7.1.2 – F-Measure

An issue arises with using these two measures as a way to judge a system; however, in that, they can often conflict with each other meaning that as one increases the other decreases. Due to as a system retrieves more relevant items, It may also retrieve more irrelevant ones. Meaning this also works contrariwise too meaning that it can be challenging to measure a system on these two factors alone accurately.

Hence when measuring the effectiveness of a system, it is best to use both in tandem hence the creation of the F-measure also known as the F_1 score which is a harmonic average of both the precision and recall and like these two measurements scale from the worst result of zero to the best result of one.

Which be expressed by the following formula: (Pitakrat, van Hoorn and Grunske, 2013)

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

7.3 – Comparison and Results

	TP	FN	TN	FP	Recall	Precision	F1 Score
Western	49	5	41	8	0.907407	0.859649	0.882882883
Romance	26	25	40	10	0.509804	0.722222	0.597701149
Sci-Fi	32	8	71	4	0.8	0.888889	0.842105263
Biography	6	43	48	1	0.122449	0.857143	0.214285714
Fantasy	19	31	46	4	0.38	0.826087	0.520547945
Horror	27	11	38	12	0.710526	0.692308	0.701298701
					Average		0.626470276
					Standard Deviation		0.244862607

Figure 6 - Confusion matrix using new importance formula

	TP	FN	TN	FP	Recall	Precision	F1 Score
Western	53	1	44	5	0.981481	0.913793	0.946428571
Romance	28	23	26	24	0.54902	0.538462	0.54368932
Sci-Fi	30	10	65	10	0.75	0.75	0.75
Biography	0	50	49	49	0	0	0
Fantasy	28	18	24	26	0.608696	0.518519	0.56
Horror	1	37	47	3	0.026316	0.25	0.047619048
					Average		0.474622823
					Standard Deviation		0.378970116

Figure 7 - Confusion matrix using tf-idf formula

7.3.1 – Comparison of Overall Performance

The results from the SVM were mixed but mostly positive here we can see that more distinct differences in the ability to predict which genre the file is of. With more genres such as Westerns or Sci-Fi, we can see that the F_1 score is significantly higher with scores of 0.8829 and 0.8421 respectively. Most likely this is due to the different types of words used in a Western movie as opposed to other movies.

However, as we can see with movies that have a less distinctive dialogue using a bag of words model is less likely to provide accurate results. With the Biography model, for instance, it almost always categorises a film as being Not a Biography. As Biographical films are defined by being about a single person's life, there are less apparent signifiers which can be used to tell if the movie does or does not fall into that genre. This can be seen when plotting the word vectors to a PCA Diagram as seen in Figure 9 here we can see that these vectors share almost the same distribution patterns as words which are not found in that genre. Contrast this with Figure 8 which shows words in the genre Western compared to

words which are not here we can see a small cluster of words of that genre indicating that there is a distinct difference between words of that genre and words which are not.

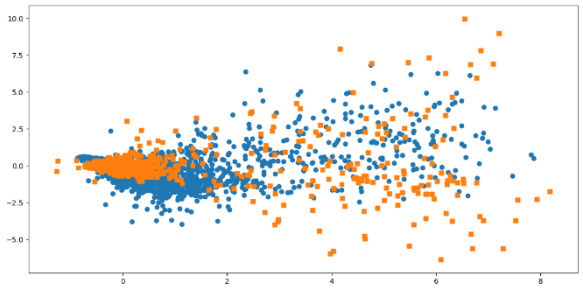


Figure 8 - PCA Diagram of Western (orange) and not Western (blue)

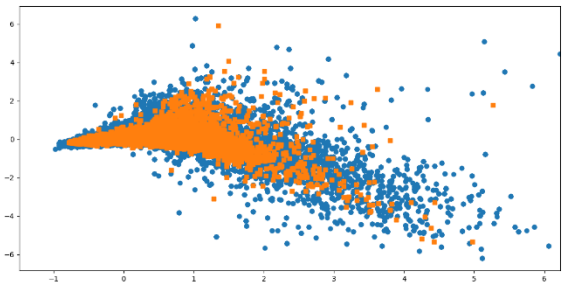


Figure 9 - PCA Diagram of Biography (orange) and not Biography (blue)

Due to this, it stands to reason that genres which display ideas that are not found in other genres would be easier to classify; however, as we can see with Fantasy, this is not the case. A reason for this could be the relative broadness of the genre Fantasy. When a layperson thinks of the genre Fantasy, they may think of a distinct set of genre connotations such as magic, elves, and so forth. However, the term fantasy is used to describe anything which contains any unrealistic setting, and due to this broadness, it may be hard for a machine to see the distinction between them.

Taking this into account it may be best to take this method of classification as more a feature selection method rather than a way to classify movies on their own. While this method works well for Sci-Fi and Western movies, other methods could be used to find films in different genres. For instance, as Biography is categorised a detailed look into a non-fictional person's life, it may be best to perform a search of proper nouns and places and then search them against a database to see if they genuinely existed.

7.3.2 – Comparisons of different formulas

In Figure 6 and Figure 7 we can see that the formula created for this project mostly outperformed the tf-idf formula with an average F_1 score 0.1493 higher. This was surprising as the tf-idf formula created by Jones (1972) has been the standard when defining the importance of words however when the top two-hundred of each are used to create an SVM we can see a significant increase the average F_1 scores when using this formula rather than tf-idf.

Using the genre Sci-Fi as an example, it can be seen that using the formula defined in this paper the F_1 score is 0.8421 whereas with the 'inverse document frequency' formula the F1 score is only 0.75. However, it can be seen that for this project the formula created for this project is superior in all but two genres those being Western where the F_1 score is higher by 0.07686 and Fantasy where the difference is 0.009275 which could be considered marginal.

One reason for this may be that the with the original formula each word is being compared with a word frequency of subtitles that are not in the genre that is being tested. This means that words which are seen frequently in that genre but are less frequent in others are given more weight. For example, we can take the word 'alien' when comparing Sci-Fi movies. If Sci-Fi movies were included in P_n then every time the word 'alien' is used in another Sci-Fi the weight would become lower this is not the case here; however if the genre is not a subset of n.

Another reason may be due to that lack of precision of the inverse document frequency function. With this function, if a word is used once or many times in another document they are counted as the same. This may cause issues; however when an individual corpus contains many words as the possibility of irrelevant words appearing in the text go up significantly. For instance, to go back to the example of the word 'alien' if this is used in a non-science fiction movie once as either a joke or figure-of-speech then df_t will increment the same; as if it is used many times in a context where it is more relevant. This explanation may also go some way into the reasoning why Western ranked higher with the tf-idf formula as many of the words spoken in Western movies are not spoken at all in other movies hence tf-idf's advantage with this particular genre.

Chapter 8 – Conclusion

8.1 – Hypothesis

This paper hypothesised that it was possible to classify movies in genres using the text or subtitles alone. With this in mind, this report has found mixed results depending on the genres. As each model has two possible outputs, we can say that the probability and F_1 Score for randomly choosing a value would be 0.5. Therefore, any value above that shows some degree of significance.

Using only the formula created for this project we can see that every F_1 score but Biography managed to be higher showing that this method works at least in part.

However, even despite this, it is likely that this would not be useful for commercial use as while it shows that there is some validity to this method even without the genre biography the average recall is still only 0.6905 which means that around 30.94% of documents are not successfully retrieved. This would be an issue if a system relied upon this method. However, the results are positive enough to show that this could be useful if implemented with other techniques.

8.2 – Research Questions

In Chapter 1.2 this report outlined four research questions which it was the purpose of this project in answering. This chapter will hope to summarise and reliably answer these questions.

8.2.1 – Is it possible to predict a movies genre using machine learning techniques?

As noted in Chapter 2 – Related Work there has already been significant work done in this area and while these methods may not have been done on entire movies the proof of concept is still there showing that this most likely possible. With this project however the results are more mixed it would be accurate to say that it is possible to see if a movie is a Western or Sci-Fi with a reasonable degree of reliability; other genres it seems cannot be classified using this method or at least cannot be classified with a reasonable degree of reliability.

8.2.2 – What is the best method to categorise large corpora of text?

While this project has not covered every type of text classification, it has looked into many and tried at least one approach. While it would most likely be inaccurate to say this is the best method for categorising large corpora of text this method does show some promise and could be used along with other methods in the future.

8.2.3 – Which genres are more easily classified?

Of the genres tested it is clear that the genre Western is the easiest to classify. However, looking at genres overall, we can assume from looking at the data that a genre must have two aspects to be classified well by this method. The first being that works in the genre must deviate from reality in a significant way as they must be words within the corpus of the genre that are unlikely to be found in others. The second is that there must be a distinct set of rules on what the genre is hence why we can see that the genre Fantasy performs relatively poorly.

8.2.4 – Can a similar process be used to determine other characteristics?

This research question was unfortunately not followed up on; however given the focus on different words it would be interesting to see if this process would work in determining particular screenwriters or which decade a film is from.

8.3 – Future Work

As discussed in previous chapters the results of this project were mostly positive yet could use some room for improvement. There are several things which could be done to build on this project going forward.

8.3.1 – Test different genres

This project only went so far as to test the classification ability of this method on seven genres. While this gives a good understanding of how an SVM can be used to classify subtitles into genres, it still misses out twenty-one other genres that IMDb classifies their movies. Hence the next step would be to see how this handles other genres and see if the hypothesis made by looking at the results of these seven remain true.

8.3.2 – Use Neural Network to classify genres

The concept of using neural networks to classify movies into genres would not be new as seen in Jain and Jadon (2008) which was discussed in Chapter 2 and their use in classifying text has also been researched. An interesting paper on this is 'Effective Use of Word Order for Text Categorization with Convolutional Neural Networks' by Johnson and Zhang (2015) in this paper the author discusses how the bag-of-words (bag-of-n-grams) approaches can fail due to lacking word order which is something that can be seen when we look at this report. Due to this, the authors utilise a Convolutional Neural Network (CNN).

These types of neural networks were initially created for image recognition however as Johnson and Zhang (2015) discuss it can classify text as results on a sentiment analysis task show that it outperformed SVM models when given the same data. Hence it would be interesting to see if when the same method is applied to this task if it also provides better results.

References

Bellegarda, J. R. (1998) 'A multispan language modeling framework for large vocabulary speech recognition', *IEEE Transactions on Speech and Audio Processing*, 6(5), pp. 456–467. doi: 10.1109/89.709671.

Bengio, Y. *et al.* (2003) 'A Neural Probabilistic Language Model', *Journal of Machine Learning Research*, 3, pp. 1137–1155. doi: 10.1007/3-540-33486-6_6.

Boser, B. E., Guyon, I. M. and Vapnik, V. N. (1992) 'A training algorithm for optimal margin classifiers', in *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*. New York, New York, USA: ACM Press, pp. 144–152. doi: 10.1145/130385.130401.

Buckland, M. and Gey, F. (1994) 'The relationship between Recall and Precision', *Journal of the American Society for Information Science*. John Wiley & Sons, Ltd, 45(1), pp. 12–19. doi: 10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L.

Davis, J. and Goadrich, M. (2006) 'The relationship between Precision-Recall and ROC curves', in *Proceedings of the 23rd international conference on Machine learning - ICML '06*. New York, New York, USA: ACM Press, pp. 233–240. doi: 10.1145/1143844.1143874.

Firth, J. R. (1957) *Studies in linguistic analysis.* Blackwell.

Huang, Y.-F. and Wang, S.-H. (2012) 'Movie Genre Classification Using SVM with Audio and Video Features', in *Proceedings of the 8th international conference on Active Media Technology*, pp. 1–10. doi: 10.1007/978-3-642-35236-2_1.

Jadon, R. S. and Jain, S. (2010) 'Intelligent Schemes for Indexing Digital Movies', in. Springer, Berlin, Heidelberg, pp. 518–529. doi: 10.1007/978-3-642-14834-7_49.

Jain, S. and Jadon, R. s. (2008) 'Audio Based Movies Characterization Using Neural Network', *International Journal of Computer Science and Applications*, 1(2), pp. 87–90. Available at: https://www.researchgate.net/publication/228984777_Audio_Based_Movies_Characterization_Using_Neural_Network [Accessed: 22 Apr. 2019].

Jain, S. K. and Jadon, R. S. (2009) 'Movies genres classifier using neural network', in *2009 24th International Symposium on Computer and Information Sciences*. IEEE, pp. 575–580. doi: 10.1109/ISCIS.2009.5291884.

Johnson, R. and Zhang, T. (2015) 'Effective Use of Word Order for Text Categorization with Convolutional Neural Networks', in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 103–112. doi: 10.3115/v1/N15-1011.

Jones, K. S. (1972) 'A Statistical Interpretation of Term Specificity and Its Application in Retrieval', *Journal of Documentation*, 28(1), pp. 11–21. doi: 10.1108/eb026526.

Kent, A. *et al.* (1955) 'Machine literature searching VIII. Operational criteria for designing information retrieval systems', *American Documentation*. John Wiley & Sons, Ltd, 6(2), pp. 93–101. doi: 10.1002/asi.5090060209.

Krithara, A. *et al.* (2018) 'Semi-supervised Document Classification with a Mislabeling Error Model', in *Advances in Information Retrieval*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 370–381. doi: 10.1007/978-3-540-78646-7_34.

Lovins, J. B. (1968) 'Development of a Stemming Algorithm', *Mechanical Translation and*

Computational Linguistics, 11(1), pp. 22–31. Available at: https://www.semanticscholar.org/paper/Development-of-a-stemming-algorithm-Lovins/6b3853f08c482fe1bfbe39d656d50a8c73976f3c [Accessed: 30 Apr. 2019].

Mikolov, T. *et al.* (2013) *Efficient Estimation of Word Representations in Vector Space*. Available at: http://ronan.collobert.com/senna/ [Accessed: 12 Mar. 2019].

Mikolov, T., Yih, W.-T. and Zweig, G. (2013) *Linguistic Regularities in Continuous Space Word Representations*. Association for Computational Linguistics. Available at: http://research.microsoft.com/en- [Accessed: 30 Apr. 2019].

Pang, B., Lee, L. and Vaithyanathan, S. (2002) 'Thumbs up? Sentiment Classification using Machine Learning Techniques', *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, 10, pp. 79–86. doi: 10.3115/1118693.1118704.

Pitakrat, T., van Hoorn, A. and Grunske, L. (2013) 'A comparison of machine learning algorithms for proactive hard disk drive failure detection', in *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems - ISARCS '13*. New York, New York, USA: ACM Press, p. 1. doi: 10.1145/2465470.2465473.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) 'Learning representations by back-propagating errors', *Nature*, 323(6088), pp. 533–536. doi: 10.1038/323533a0.

Tan, C.-M., Wang, Y.-F. and Lee, C.-D. (2002) 'The use of bigrams to enhance text categorization', *Information Processing & Management*. Pergamon, 38(4), pp. 529–546. doi: 10.1016/S0306-4573(01)00045-0.

Wactlar, H. D. (2001) 'The Challenges of Continuous Capture , Contemporaneous Analysis , and Customized Summarization of Video Content'. Available at: https://www.semanticscholar.org/paper/The-Challenges-of-Continuous-Capture-%2C-Analysis-%2C-Wactlar/9de40d04535729707aaad62410ed8a6b2fdfc25e#citing-papers [Accessed: 25 Apr. 2019].

Wang, C.-M. and Huang, Y.-F. (2010) 'Self-adaptive harmony search algorithm for optimization', *Expert Systems with Applications*. Pergamon, 37(4), pp. 2826–2837. doi: 10.1016/J.ESWA.2009.09.008.

Appendix A – Figures

Figure 1 - CBOW and Skip-gram architecture

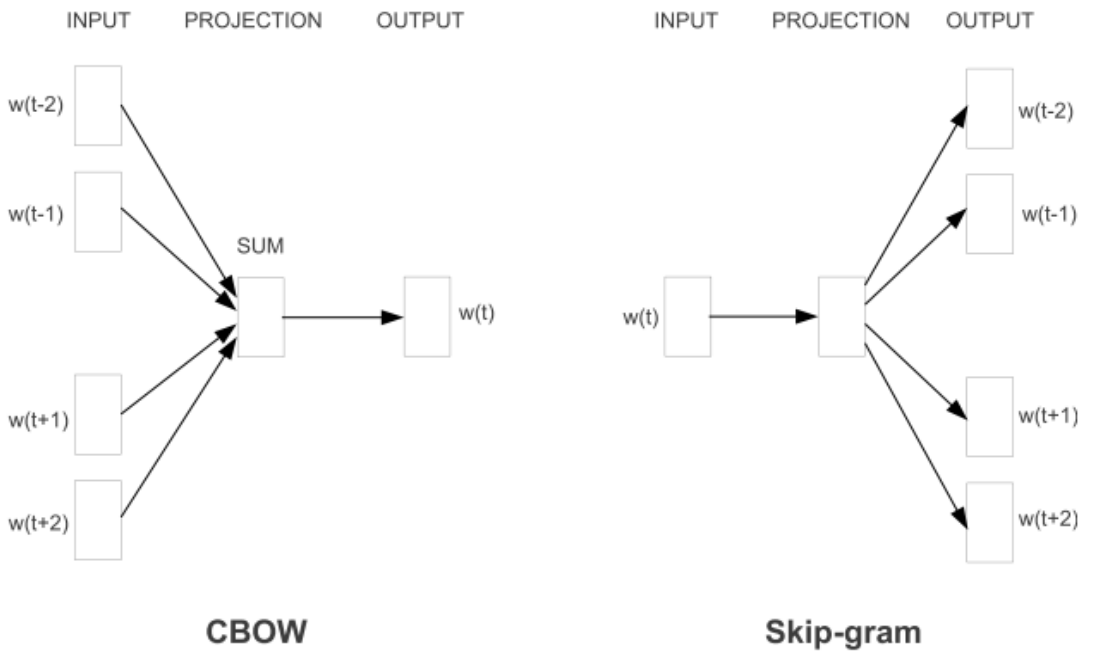


Figure 2 - PCA Diagram of words "woman", "man", "queen", "king"

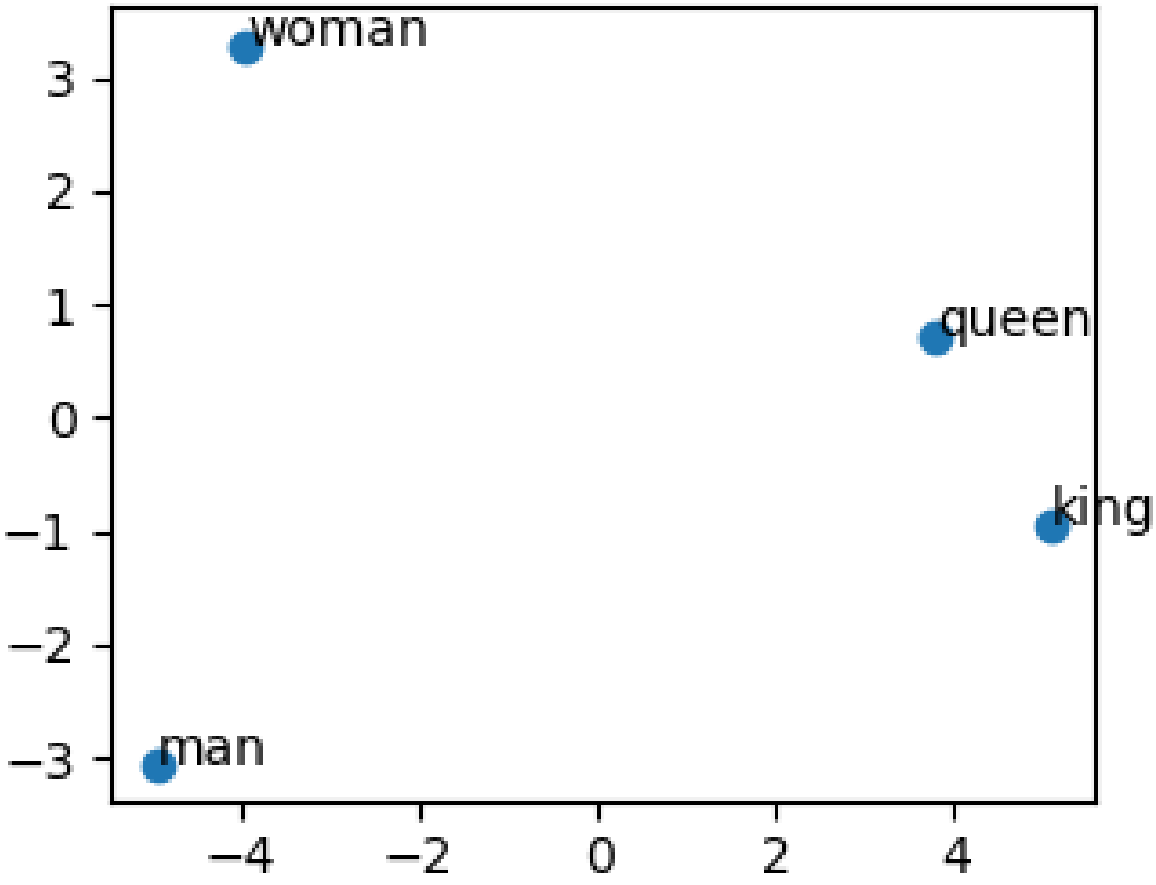


Figure 4 - PCA Diagram of all words in Star Trek Beyond (orange) and Gandhi (blue)

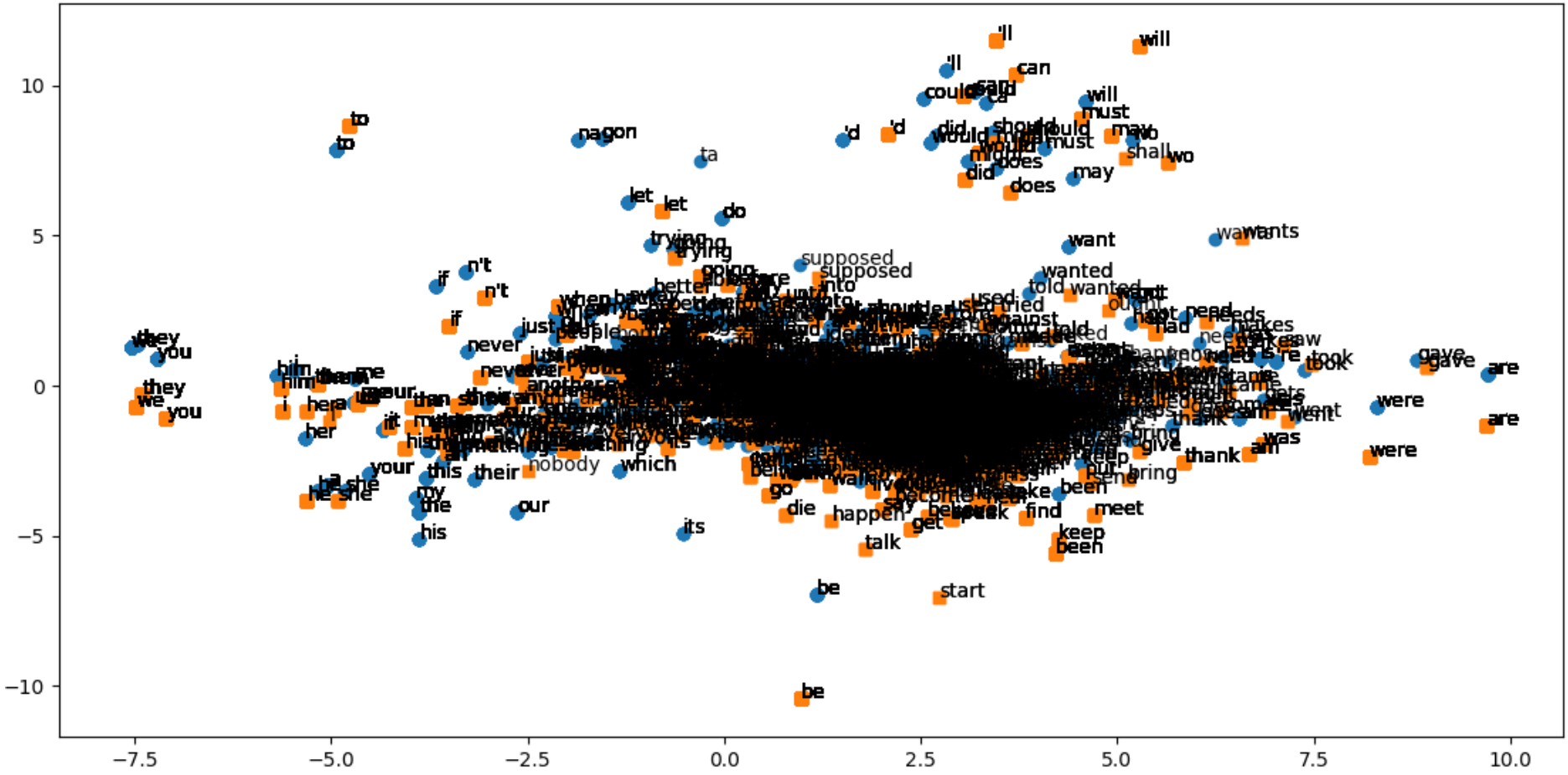


Figure 3 - Top hundred words from Star Trek Beyond (orange) and Ghandi (blue)

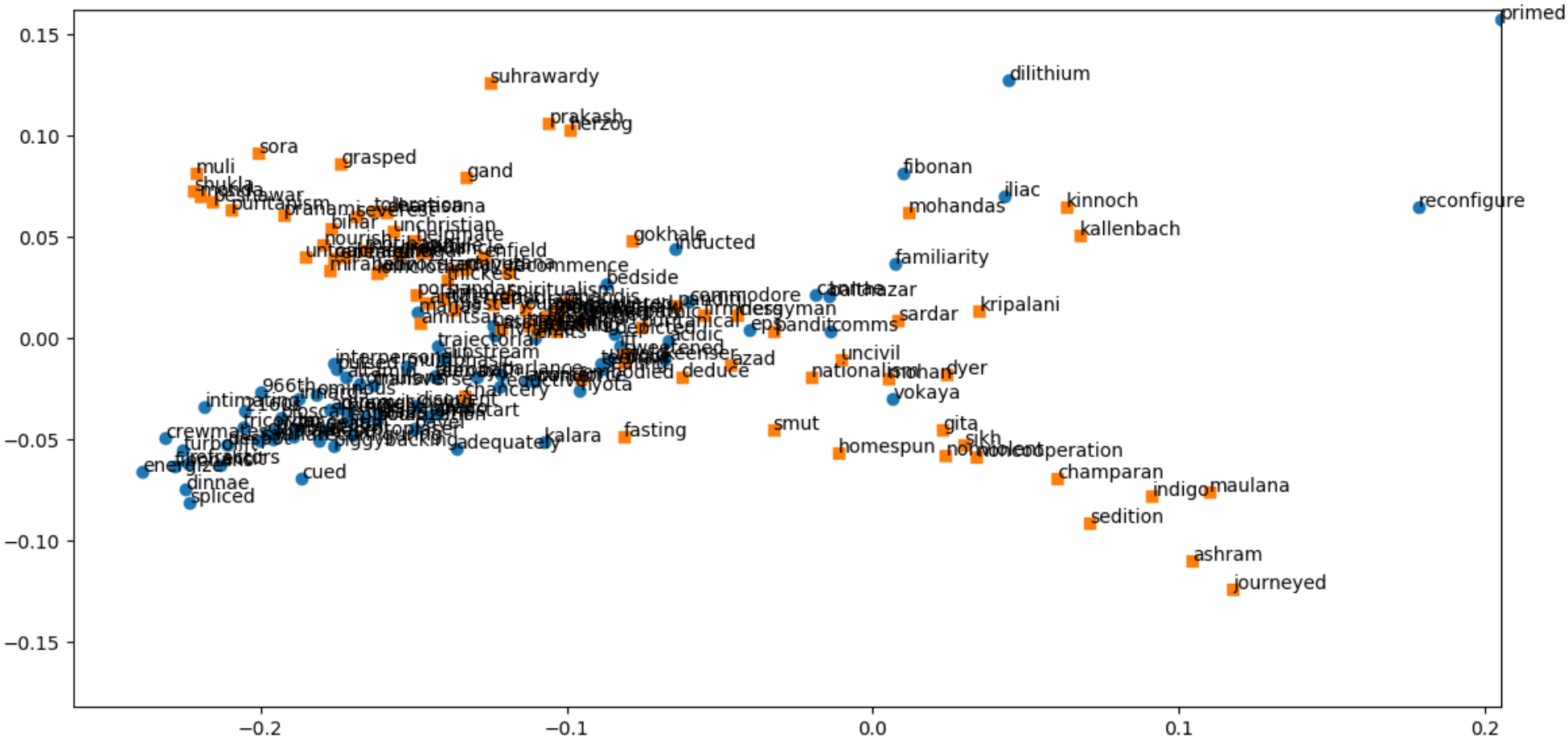


Figure 5 - PCA Diagram of Sci-Fi (orange) and not Sci-Fi (blue)

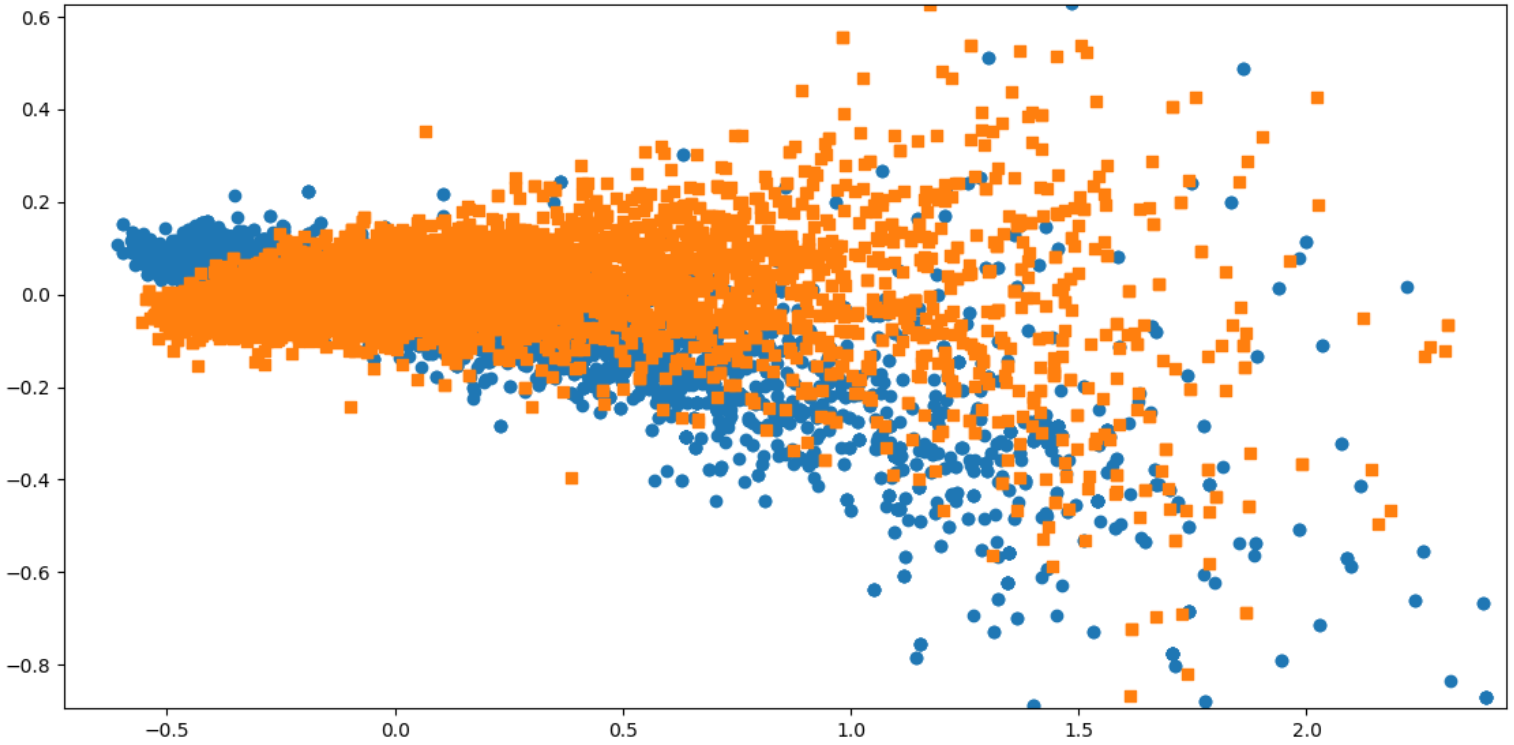


Figure 6 - Confusion matrix using new importance formula

	TP	FN	TN	FP	Recall	Precision	F1 Score
Western	49	5	41	8	0.907407	0.859649	0.882882883
Romance	26	25	40	10	0.509804	0.722222	0.597701149
Sci-Fi	32	8	71	4	0.8	0.888889	0.842105263
Biography	6	43	48	1	0.122449	0.857143	0.214285714
Fantasy	19	31	46	4	0.38	0.826087	0.520547945
Horror	27	11	38	12	0.710526	0.692308	0.701298701
					Average		0.626470276
					Standard Deviation		0.244862607

Figure 7 - Confusion matrix using tf-idf formula

	TP	FN	TN	FP	Recall	Precision	F1 Score
Western	53	1	44	5	0.981481	0.913793	0.946428571
Romance	28	23	26	24	0.54902	0.538462	0.54368932
Sci-Fi	30	10	65	10	0.75	0.75	0.75
Biography	0	50	49	49	0	0	0
Fantasy	28	18	24	26	0.608696	0.518519	0.56
Horror	1	37	47	3	0.026316	0.25	0.047619048
					Average		0.474622823
					Standard Deviation		0.378970116

Figure 8 - PCA Diagram of Western (orange) and not Western (blue)

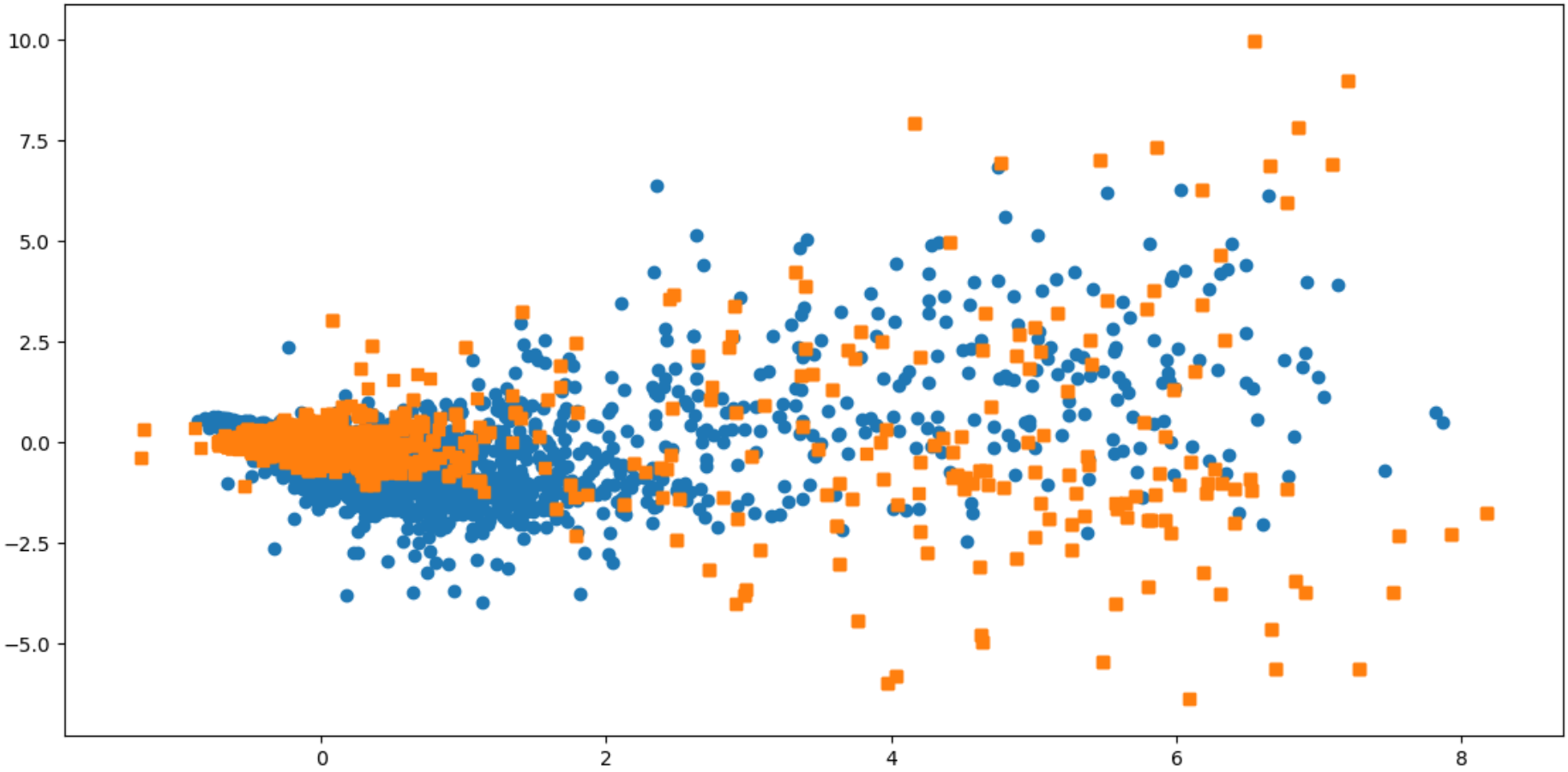
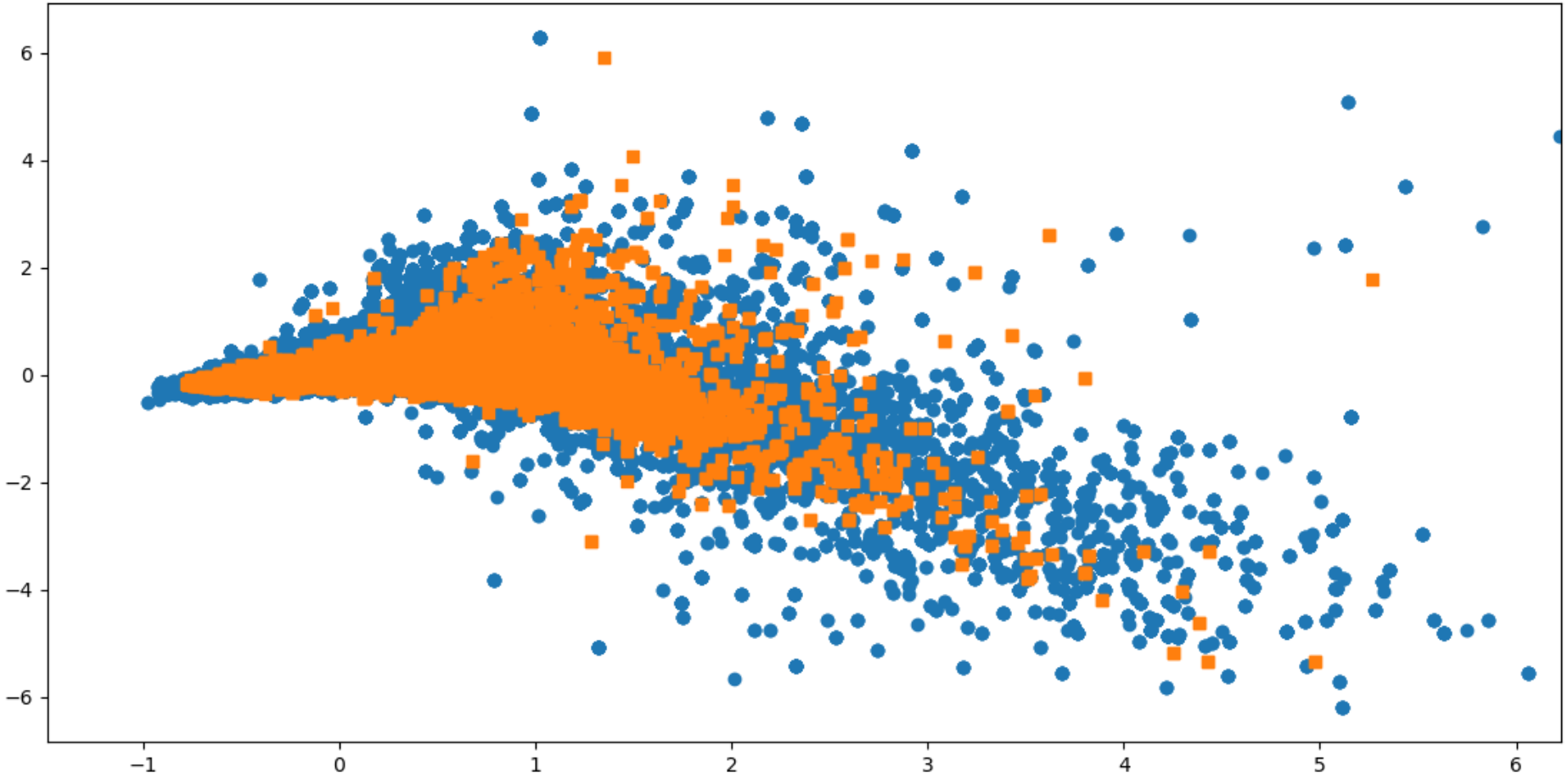


Figure 9 - PCA Diagram of Biography (orange) and not Biography (blue)



Appendix B - Code

main_functions.py

```
import json
import urllib.request
import csv
import pandas as pd
import numpy as np
from os import path
import os
from pythonopensubtitles.utils import File
from pythonopensubtitles.opensubtitles import OpenSubtitles
import gzip
import requests
import shutil
import re
import string
import operator
import glob
import nltk
from nltk.stem import WordNetLemmatizer
from pathlib import Path
from collections import Counter
from nltk.stem import WordNetLemmatizer
import pickle
import random
lemmatizer = WordNetLemmatizer()
ost = OpenSubtitles()


##
# Trims the words in an array
#
# @param    array   An array of strings
# @return   array   An array of strings without whitespace either side of
elements
#
def arrayTrim(array):
    output = []
    for a in array:
        output.append(a.strip())
    return output

##
# Turns a dictionary into a CSV file and then saves it
#
# @param    dict    The dictionary to be saved to a CSV as 'Word' =>
'Frequency'
# @param    string  The name of the CSV file
# @return   void
#

def dictionaryToCSV(dictionary, csvName):
    f = open(csvName, mode='w', encoding='utf-8')
    f.write("Words,Frequency\n")
    for word, freq in dictionary.items():
          f.write(word+","+str(freq)+"\n")

##
```

```
# Retrives a dictionary based on a CSV filename
#
# @param    string  The name of the CSV file
# @return   dict    The dictionary with the data in the format of 'Word' =>
'Frequency'
#

def csvToDictionary(filename):
    csv_file = pd.read_csv(filename, encoding='utf-8',quoting=3,
error_bad_lines=False)
    words_array = np.asarray(csv_file[['Words']])
    frequency_array = np.asarray(csv_file[['Frequency']])
    output = dict()
    for i in range(len(words_array)):
        output[ words_array[i][0] ] = frequency_array[i][0]
    return output

##
# Used to see if it is a movie or not
#
# @param    int     The ID given to it by IMDB
# @return   string  The type of media it is 'movie','episode',etc.
#
def getMediaType(imdbid):
    url = "http://www.omdbapi.com/?apikey=1cb8fb66&i=tt"
    data = urllib.request.urlopen(url+imdbid)
    for line in data:
        json_data = line
    y = json.loads(json_data)

    return y['Type']
##
# Gets an array of the genres IMDB says the movie is
# @param    string     The ID given to it by IMDB
# @return   array   The genres the movie is
#
def getGenres(imdbid):
    url = "http://www.omdbapi.com/?apikey=1cb8fb66&i=tt"

    data = urllib.request.urlopen(url+imdbid)
    for line in data:
        json_data = line
    y = json.loads(json_data)

    genres = (y['Genre']).split(",")
    genres = arrayTrim(genres)
    with open('Movie Genres.csv', 'a', newline='') as csvfile:
        genreWriter = csv.writer(csvfile)
        for genre in genres:
            genreWriter.writerow([imdbid,genre])

    return genres

##
# Creates a filename which will not throw an error by removing illegal
characters
#
# @param    string  The name of the file which may have illegal charcters
# @return   string  The filename but with "_" replacing illegal characters
#
def validateFileName(filename):
```

```
    badCharacters = ['#','%','&','{','}','\',<','>','*','?','/',' 
','$','!',"'",'"',':','@']
    for b in badCharacters:
        filename = filename.replace(b,"_")
    return filename
##
# Returns and saves subtitle files of movies with the corrisponding IMDB ID
#
# This is dependent on the OpenSubtitles API and therefore may cause issues
if limit is reached
# Subtitles are saved as "subs/{imdbid}--{Move Name}.srt"
# @param    string     The ID given to it by IMDB
# @return   string  The contents of the subtitle file
#
def getSubtitles(imdbid, folder_name = 'subs'):
    token = ost.login('thebobbrom', 'timetravel27')

    data = ost.search_subtitles([{'sublanguageid': 'eng', 'imdbid': 
str(imdbid)}])

    if(data == None or len(data) == 0):
        return True
    if(data[0].get('Response') == 'False'):
        return False
    sub_link = data[0].get('SubDownloadLink')
    movie_name = data[0].get('MovieName')
    r = requests.get(sub_link)
    filename = 'temp/'+sub_link.split("/")[-1]

    with open(filename, "wb") as f:
        r = requests.get(sub_link)
        f.write(r.content)

    srt_name = filename.split(".")[0]
    with gzip.open(filename, 'rb') as f_in:
        with open(srt_name, 'wb') as f_out:
            shutil.copyfileobj(f_in, f_out)

    f = open(srt_name, "r")
    subs = f.read()
    f.close()
    movie_name = validateFileName(movie_name)
    new_srt_name = folder_name+"/"+imdbid+"--"+movie_name+".srt"

    open(new_srt_name, mode='w', encoding='utf-8').write(str(subs))
    os.remove(filename)
    return subs


##
# Sorts a dictionary array in decending order
# @param    dict    Unsorted dictionary
# @return   dict    Sorted dictionary
#

def arsort(d):
    output = dict()
    sorted_list = sorted(d.items(), key=lambda x: x[1], reverse=True)
    for item in sorted_list:
        output[ item[0] ] = item[1]
    return output
```

```
##
# Prepares subtitles removing unwanted characters
# @param    string  The unprepared subtitles
# @param    string  A string representation of all the punctuation which
needs removing
# @return   string  The subtitles with all unwanted characters removed
#

def prepareText(string, punc):
     punc = punc.replace("'","")
     string = string.replace("'s","")
     remove = ["<font>","</font>","-",".","¡","»","¿"]
     string = string.lower()
     for r in remove:
          string = string.replace(r," ")
     for char in punc:
          string = string.replace(char," ")
     return string

##
# This strips things such as timecodes from subtitles
# @param    string  The subtitles with their timecodes still in
# @return   string  The subtitles without timecodes
#
def filterSubs(subs):
    pattern = re.compile("\d")
    i = 0
    lines = subs.split("\n")
    output = ""

    for line in lines:
        if(re.match(pattern, line) or not line):
             if(len(lines) > i):
                 lines.pop(i)
             if(len(lines) > i):
                 lines.pop(i)
        else:
             output += "\n" + line
        i = i + 1
    return(output)

##
# Counts how many of each word there and returns a dictionary with the word
as the key
# This also saves the word frequency to a CSV file
# @param    string   The subtitles which need their words counted
# @param    string   The ID given to it by IMDB
# @param    string   Optional parameter which will determine which folder
the CSV goes to
# @return   dict     A dictionary of 'Word' => 'Frequency'
#
def getWordFrequency(subs,imdbid,folder="word_freq",writeToFile=True):
     words = dict()
     table = str.maketrans(dict.fromkeys(string.punctuation+"\n"))

     newLines = [];

     output = filterSubs(subs)
     output = prepareText(output, string.punctuation)
     arr = output.split()
```

```
     freq = dict()
     lemmatizer = WordNetLemmatizer()
     for word in arr:
          word = lemmatizer.lemmatize(word)
          if word not in freq:
               freq[word] = 1
          else:
               freq[word] = freq[word] + 1

     sorted_d = arsort(freq)
     if(writeToFile):
         csv_name = folder+"/"+imdbid + "--word_frequency.csv"
         dictionaryToCSV(sorted_d, csv_name)
     return sorted_d

def makeCSV(filename):
    if(filename.split(".")[-1] != "csv"):
        filename = filename+".csv"
    f = open(filename, "w")
    f.write("Words,Frequency")
    f.close()

def addWordsToCSV(sorted_freq,genre,foldername='genre_corpus'):
    filename = foldername+"/"+genre+".csv"
    my_file = Path(filename)
    original_dict = dict()

    if (not my_file.is_file()):
        makeCSV(filename)
    csv_file = pd.read_csv(filename,encoding = "utf-8")
    original_frequency = np.asarray(csv_file[['Frequency']])
    original_words = np.asarray(csv_file[['Words']])

    for i in range(len(original_words)):
        original_dict[ str(original_words[i][0]) ] =
original_frequency[i][0]
    new_dict = Counter(original_dict) + Counter(sorted_freq)
    new_dict = arsort(new_dict)

    dictionaryToCSV(new_dict, filename)

#-------------------------------------------------------
#   Little functions for running
#-------------------------------------------------------


def addToDone(imdbid):
    f = open("done_imdb_ids.csv", "a")
    f.write("\n"+imdbid)

def openByIMDBID(imdbid):
    imdbid = "{0:07d}".format(int(imdbid))
    srt_name = glob.glob('subs/'+str(imdbid)+'--*.srt')[0]
    f = open(srt_name, "r")
    return f.read()


def getDoneArray():
    doneFileNames = glob.glob("word_freq/*.csv")
    doneFileNames += glob.glob("testing_word_freq/*.csv")
```

```
    done_array = []
    for filename in doneFileNames:
        done_array.append(filename.split('\\')[1].split('--')[0])
    return done_array

def getSubsArray(f = 'subs'):
    doneFileNames = glob.glob(f+"/*.srt")
    done_array = []
    for filename in doneFileNames:
        done_array.append(filename.split('\\')[1].split('--')[0])
    return done_array

def printProgress(i, arr_length):
    print(str(i)+" / "+str(arr_length)+" -- "+str(arr_length - i)+" to go")


def getToDoArray():
    big_imdbid = pickle.load( open( 'imdbid_big.pkl', "rb" ) )
    new_done_imdbs = getDoneArray()
    imdbId_csv = pd.read_csv("imdb_ids.csv")
    imdbId_csv = np.asarray(imdbId_csv[['IMDB_ID']]).reshape((1,-1))[0]
    imdbid_array =[]
    for imdbid in big_imdbid:
        if imdbid in imdbId_csv and imdbid not in new_done_imdbs:
            imdbid_array.append(imdbid)
    random.shuffle(imdbid_array)
    return imdbid_array



#------------------------------------------------------
#   Runnable Functions
#------------------------------------------------------

##
# Downloads subtitles
# Gets the word frequency then adds it to CSV
# Adds to genre frequencies
#
# @return void
#


def downloadAndPrepare():
    imdbId_csv = pd.read_csv("imdb_ids.csv")
    imdbId_array = np.asarray(imdbId_csv[['IMDB_ID']])

    done_array = getDoneArray()
    i = 0
    for imdbid in imdbId_array:
        if imdbid in done_array:
            print(imdbid)
            continue
        try:
            imdbid = str(imdbid[0]).replace("tt","")
            imdbid = "{0:07d}".format(int(imdbid))
            subs = getSubtitles(imdbid)
            if not subs:
                continue
            wordFreq = getWordFrequency(subs,imdbid)
            genres = getGenres(imdbid)
```

```
            addWordsToCSV(wordFreq,"normal_corpus")
            for genre in genres:
                addWordsToCSV(wordFreq,genre)
            i = i + 1
            addToDone(imdbid)
            if i >= 500:
                break
        except Exception as e:
            print("Error on "+imdbid)
            print("--> "+str(e))
            if(str(e) == "<ProtocolError for api.opensubtitles.org/xml-rpc:
429 Too Many Requests>"):
                return False
                break


def downloadTesting(genre = False, limit = False):
    imdbid_csv = pd.read_csv("imdb_ids.csv")
    imdbid_array = getToDoArray()



    if not genre:
        new_imdbid_array = []
        i = 0
        for imdbid in imdbid_array:
            genres = getGenres(imdbid)
            if genre in genres:
                new_imdbid_array.append(imdbid)
                print(imdbid)
            i += 1
            if not limit and i >= limit:
                break
        imdbid_array = new_imdbid_array


    if not limit == False:
        limit = len(imdbid_array)
    i = 0
    for imdbid in imdbid_array:
        try:
            imdbid = str(imdbid).replace("tt","")
            imdbid = "{0:07d}".format(int(imdbid))
            subs = getSubtitles(imdbid,'testing_subs')
            wordFreq = getWordFrequency(subs,imdbid,"testing_word_freq")
            i = i + 1
            printProgress(i, limit)
            if i >= limit:
                break
        except Exception as e:
            print("Error on "+imdbid)
            print("--> "+str(e))


#downloadTesting('Sci-Fi',10)


##
# Gets pre-downloaded subtitles from 'subs' folder
# Gets the word frequency then adds it to CSV
```

```
# Adds to genre frequencies
#
# @return void
#

def prepare():
    imdbId_array = getSubsArray()

    done_array = getDoneArray()
    i = 0
    for imdbid in imdbId_array:
        #imdbid = imdbid[0].replace("tt","")
        #imdbid = "{0:07d}".format(int(imdbid))
        if imdbid in done_array:
            continue
        try:
            subs = openByIMDBID(imdbid)
            subs = lemmatizer.lemmatize(subs)
            wordFreq = getWordFrequency(subs,imdbid)
            genres = getGenres(imdbid)
            addWordsToCSV(wordFreq,"normal_corpus")
            for genre in genres:
                addWordsToCSV(wordFreq,genre)
            i = i + 1

        except Exception as e:
            print("Error on "+imdbid)
            print("--> "+str(e))

        printProgress(i, len(imdbId_array))
```

create_svm.py

```
from sklearn import svm

from sklearn import metrics
from gensim.models import Word2Vec, KeyedVectors
import Word2Vec_PCA as wtv
import pickle
import main_functions as bow
from sklearn.model_selection import GridSearchCV


def createModelSVM(genre, cost=10, g=2048, w2v_model = 'model.bin'):
    model = Word2Vec.load(w2v_model)
    not_genre_output = pickle.load( open( "top_words/not_"+genre+".pkl",
"rb" ) )
    genre_output = pickle.load( open( "top_words/"+genre+".pkl", "rb" ) )
    if(not_genre_output > genre_output):
        not_genre_output = not_genre_output[:len(genre_output)]
    else:
        genre_output = genre_output[:len(not_genre_output)]

    ex = []
```

```
    X = []
    y = []
    i = 0
    for word in genre_output:
        try:
            X.append( model[word] )
            y.append( genre)
            i +=1
            if i % 1000 == 0 :
                bow.printProgress(i, len(genre_output))
        except Exception as e:
            ex.append(e)

    i = 0
    for word in not_genre_output:
        try:
            X.append( model[word] )
            y.append( 'Not '+genre)
            i +=1
            if i % 1000 == 0 :
                bow.printProgress(i, len(not_genre_output))
        except Exception as e:
            ex.append(e)


    clf = svm.SVC(C=cost, gamma=g)
    clf.fit(X, y)

    gamma = (str(g)).replace(".",'')
    pkl_filename = "SVM_models/"+genre+"--
"+"C"+str(cost)+"__G"+str(gamma)+"_new.pkl"
    with open(pkl_filename, 'wb') as file:
        pickle.dump(clf, file)
    return clf

createModelSVM('Western')


def getCostGamma(genre):
    model = Word2Vec.load('model.bin')
    not_genre_output = pickle.load( open( "top_words/not_"+genre+".pkl",
"rb" ) )
    genre_output = pickle.load( open( "top_words/"+genre+".pkl", "rb" ) )
    X = []
    y = []
    i = 0
    ex = []
    for word in genre_output:
        try:
            X.append( model[word] )
            y.append( genre)
            i +=1
            if i % 1000 == 0:
                bow.printProgress(i, len(genre_output))
        except Exception as e:
            ex.append(str(e))

    i = 0
    for word in not_genre_output:
        try:
            X.append( model[word] )
```

```
            y.append( 'Not '+genre)
            i +=1
            if i % 1000 == 0:
                bow.printProgress(i, len(not_genre_output))
        except Exception as e:
            ex.append(str(e))

    try:
        tuned_parameters = [{'kernel': ['rbf'], 'gamma': [10, 100,
2048,3000, 5000],
                             'C': [1, 10, 100, 1000]},
                            {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
        clf = GridSearchCV(svm.SVC(), tuned_parameters, cv=5)
        clf.fit(X, y)


        pkl_filename = "SVM_models1/"+genre+".pkl"
        with open(pkl_filename, 'wb') as file:
            pickle.dump(clf, file)

        f = open('C_Gamma_Score/'+genre+'.txt',"w")
        f.write(str(clf.best_params_))
        f.write("/n")
        f.close()
    except Exception as e:
        f = open("Errors/find_CG_"+genre+".txt","w")
        f.write(str(e))
        f.close()
```

create_second_layer.py

```
import pickle
from gensim.models import Word2Vec, KeyedVectors
from sklearn import svm
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import glob
import main_functions as bow
import testing_script as ts
import random

import warnings
warnings.filterwarnings("ignore")

def hasNumbers(inputString):
    return any(char.isdigit() for char in inputString)



def create_second_layer(genre):
    w2v_model = Word2Vec.load('model.bin')



    subs_genre = glob.glob('testing_subs/'+genre+'/*.srt')
    random.shuffle(subs_genre)
```

```
    subs_non_genre = glob.glob('testing_subs/Non-'+genre+'/*.srt')

    random.shuffle(subs_non_genre)


    if len(subs_genre) > len(subs_non_genre):
        subs_genre = subs_genre[:len(subs_non_genre)]
    else:
        subs_non_genre = subs_non_genre[:len(subs_genre)]

    training_result = [genre]*len(subs_genre)
    training_result += ['Non-'+genre]*len(subs_non_genre)


    sub_array = subs_genre + subs_non_genre


    results = []
    zzz = 0


    print(len(sub_array))
    for sub_name in sub_array:
        try:
            zzz += 1
            f = open(sub_name)
            subs = f.read()
            f.close()


            word_freq =
bow.getWordFrequency(subs,'000DELETE000',"word_freq",False)
            ##word_freq = ts.csvToDictionary('word_freq/0829482--
word_frequency.csv')
            word_perc = ts.dictPercent(word_freq)

            not_genre_dict = ts.csvToDictionary('not_genre/'+genre+'.csv')
            not_genre_percent = ts.dictPercent(not_genre_dict)

            unique_array_dict = ts.uniqueArray(not_genre_percent,word_perc)
            unique_array = list(unique_array_dict.keys())

            top_words = []
            i = 0

            for word in unique_array:
                if word in w2v_model and not hasNumbers(word):
                    top_words.append(word)
                    i += 1
                if i>199:
                    break



            scores = dict()
            clf = pickle.load( open('SVM_models/'+genre+'--C10__G2048.pkl',
"rb" ) )
            i = 0
            for word in top_words:
                q = clf.predict([w2v_model[word]])
                if(q == genre):
```

```
                    i += 1
            score = ((2/len(top_words))*i)-1
            results.append([score])

            bow.printProgress(zzz,len(sub_array))



        except Exception as e:
            print(str(e))
            if(len(subs_genre) >= zzz):
                training_result.remove(genre)
            else:
                training_result.remove('Non-'+genre)

    print(len(results))
    print(len(training_result))



    clf = svm.SVC(C=100)
    clf.fit(results, training_result)



    pkl_filename = "SVM_models/second_layer/"+genre+".pkl"

    with open(pkl_filename, 'wb') as file:
        pickle.dump(clf, file)



    tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4, 10, 100,
2048],
                         'C': [1, 10, 100, 1000]},
                        {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
    clf = GridSearchCV(svm.SVC(), tuned_parameters, cv=5)
    clf.fit(results, training_result)


    pkl_filename = "SVM_models/second_layer/"+genre+"2.pkl"


    with open(pkl_filename, 'wb') as file:
        pickle.dump(clf, file)



    f = open('C_Gamma_Score/'+genre+'_layer2.txt',"w")
    f.write(str(clf.best_params_))
    f.write("/n")
    f.close()
```

test_second_layer.py

```
import pickle
from gensim.models import Word2Vec, KeyedVectors
from sklearn import svm
from sklearn import metrics
import glob
import bag_of_words as bow
import testing_script as ts
import warnings
from multiprocessing.dummy import Pool as ThreadPool
import numpy as np


def hasNumbers(inputString):
    return any(char.isdigit() for char in inputString)


def test_second_layer(genre):
    w2v_model = Word2Vec.load('model.bin')
    folder = ''

    subs_genre = glob.glob('testing_subs_second_layer/'+genre+'/*.srt')
    subs_non_genre = glob.glob('testing_subs_second_layer/Non-
'+genre+'/*.srt')
    training_result = [genre]*len(subs_genre)
    training_result += ['Non-'+genre]*len(subs_non_genre)

    sub_array = subs_genre + subs_non_genre

    print(len(sub_array))

    second_layer_score = []

    results = []
    zzz = 0
    for sub_name in sub_array:
        try:
            zzz += 1

            f = open(sub_name)
            subs = f.read()
            f.close()

            #print(sub_name)

            word_freq =
bow.getWordFrequency(subs,'000DELETE000',"word_freq",False)
            ##word_freq = ts.csvToDictionary('word_freq/0829482--
word_frequency.csv')
            word_perc = ts.dictPercent(word_freq)

            not_genre_dict = ts.csvToDictionary('not_genre/'+genre+'.csv')
            not_genre_percent = ts.dictPercent(not_genre_dict)

            unique_array_dict = ts.uniqueArray(not_genre_percent,word_perc)
            unique_array = list(unique_array_dict.keys())
```

```
            top_words = []
            i = 0

            for word in unique_array:
                if word in w2v_model and not hasNumbers(word):
                    top_words.append(word)
                    i += 1
                if i>199:
                    break

            clf = pickle.load( open('SVM_models/'+genre+'--C10__G2048.pkl',
"rb" ) )
            i = 0
            for word in top_words:
                q = clf.predict([w2v_model[word]])
                if(q == genre):
                    i += 1
            score = ((2/len(top_words))*i)-1
            second_layer_score.append([score])
            bow.printProgress(zzz,len(sub_array))
        except Exception as e:
            print("Error with "+sub_name)
            print(str(e))
            if(len(subs_genre) >= zzz):
                training_result.remove(genre)
            else:
                training_result.remove('Non-'+genre)

    print("----")
    print(len(second_layer_score))

    f = open(folder+'Results/'+genre+'.txt', "w")

    #-------------------------------------------------------------------------
----
    # For Linear
    #-------------------------------------------------------------------------
----
    genre_total = 0
    genre_correct = 0

    non_genre_total = 0
    non_genre_correct = 0
    clf = pickle.load( open(folder+'SVM_models/second_layer/'+genre+'.pkl',
"rb" ) )
    predictions = clf.predict(second_layer_score)
    for i in range(len(training_result)):
        if training_result[i] == genre:
            genre_total += 1
            if training_result[i] ==  predictions[i]:
                genre_correct += 1
        else:
            non_genre_total += 1
            if training_result[i] ==  predictions[i]:
                non_genre_correct += 1
    f.write("------------------------------"+"\n")
    f.write("Linear"+"\n")
    f.write("------------------------------"+"\n")
    f.write(genre+" Recall: "+str(genre_correct/genre_total)+"\n")

    f.write(genre+" Correct (TP): "+str(genre_correct)+"\n")
```

```
    f.write(genre+" not Correct (FN): "+str(genre_total -
genre_correct)+"\n")

    f.write("Not "+genre+" correct (TN): "+str(non_genre_correct)+"\n")
    f.write("Not "+genre+" not correct (FP): "+str(non_genre_total -
non_genre_correct)+"\n")

    f.write(genre+" correct %: "+str((100/genre_total)*genre_correct)+"\n")
    f.write("Non-"+genre+" correct %:
"+str((100/non_genre_total)*non_genre_correct)+"\n")

    #-------------------------------------------------------------------------
    # For Non-Linear
    #-------------------------------------------------------------------------

    genre_total = 0
    genre_correct = 0

    non_genre_total = 0
    non_genre_correct = 0
    clf = pickle.load(
open(folder+'SVM_models/second_layer/'+genre+'2.pkl', "rb" ) )
    predictions = clf.predict(second_layer_score)
    for i in range(len(training_result)):
        if training_result[i] == genre:
            genre_total += 1
            if training_result[i] ==  predictions[i]:
                genre_correct += 1
        else:
            non_genre_total += 1
            if training_result[i] ==  predictions[i]:
                non_genre_correct += 1
    f.write("------------------------------")
    f.write("Non-Linear")
    f.write("------------------------------")
    f.write(genre+" Recall: "+str(genre_correct/genre_total)+"\n")

    f.write(genre+" Correct: "+str(genre_correct)+"\n")
    f.write("Not "+genre+" correct: "+str(non_genre_correct)+"\n")

    f.write(genre+" total: "+str(genre_total)+"\n")
    f.write("Not "+genre+" total: "+str(non_genre_total)+"\n")

    f.write(genre+" correct %: "+str((100/genre_total)*genre_correct)+"\n")
    f.write("Non-"+genre+" correct %:
"+str((100/non_genre_total)*non_genre_correct)+"\n")
```

genre_downloader.py

```
import getArrays as ga
import bag_of_words as bow
import glob
import Word2Vec_PCA as wtv
import pandas as pd
import os.path
import numpy as np
import pickle
import random
import requests
from bs4 import BeautifulSoup
from multiprocessing.dummy import Pool as ThreadPool

genre = 'Action'

##-------------------------------------------------------------
## Find IMDB ID's for Genre Movies
##-------------------------------------------------------------

imdbid_array = []
pages = 5
for i in range(pages):
    bow.printProgress(i+1,pages)
    # Change IMDB List when changing genre

    page =
requests.get('https://www.imdb.com/list/ls058416162/?sort=list_order,asc&st
_dt=&mode=detail&page='+str(i))
    soup = BeautifulSoup(page.text, 'html.parser')

    movie_names = soup.find(class_='lister-item')

    movie_names = movie_names.find_all('a')

    for a in soup.find_all('a', href=True):
        link = a['href']
        if "http" in link or "?" in link:
            continue
        if "tt" in link:
            link_array = link.split('/')
            link = link_array[2]
            link = link.replace('tt','')
            imdbid_array.append(link)

new_imdbid_array = []
imdbid_array = list(set(imdbid_array))
for imdbid in imdbid_array:
    try:
        genres = bow.getGenres(imdbid)
        if( len(genres) <= 3):
            new_imdbid_array.append(imdbid)
    except Exception as e:
        print(str(e))
random.shuffle(new_imdbid_array)
imdbid_array = new_imdbid_array
print("Amount of IMDB IDs: "+str(len(imdbid_array)))

print("----------------------------------------")
```

```
print("Downloading Subtitles then Arrays -- "+genre)
print("----------------------------------------")

folder_subs = 'testing_subs/'+genre
amount = 50


if not os.path.exists(folder_subs):
    os.makedirs(folder_subs)

new_done_imdbs = bow.getSubsArray()
new_done_imdbs += bow.getSubsArray('testing_subs/'+genre)
new_done_imdbs += bow.getSubsArray('testing_subs_second_layer/'+genre)

if folder_subs != 'subs':
    stop_at = amount - len(glob.glob(folder_subs+"/*.srt"))
else:
    top_words = pickle.load( open( 'top_words/'+genre+'.pkl', "rb" ) )
    stop_at = amount - int( len(top_words) / 200 )
    stop_at = amount
print("Stop At: "+str(stop_at))


i = 0
random.shuffle(imdbid_array)

new_imdbid_array = []


new_imdbid_array = list( set(imdbid_array) - set(new_done_imdbs) )


new_imdbid_array = new_imdbid_array[:(stop_at+20)]


for imdbid in new_imdbid_array:
    if imdbid not in new_done_imdbs:
        imdbid = imdbid.replace('tt','')
        try:
            imdbid = "{0:07d}".format(int(imdbid))
            test = bow.getSubtitles(imdbid, folder_subs)
            if test == False:
                print("Limit ran out")
                break
            elif test == True:
                print("No subtitles for "+imdbid)
            else:
                if folder_subs == 'subs':
                    ga.addToGenreNotGenreArrays(imdbid,True)
                i += 1
                bow.printProgress( i, stop_at )
            if i>= stop_at:
                break
        except Exception as e:
            i -= 1
            f = open("Errors/"+genre+"_Download.txt","w")
            f.write(str(e))
            f.write("\n")
            print(str(e))
```

train_model.py

```
def trainModel(name = 'model.bin'):
    data = getSentenceArray()
    model = Word2Vec(data, min_count=1)
    ## Save Model
    model.save(name)
```

pca.py

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import string
import glob
import re
import main_functions as bow
from gensim.models import Word2Vec, KeyedVectors
from gensim.test.utils import common_texts, get_tmpfile
from nltk.corpus import brown, movie_reviews, treebank
from sklearn.decomposition import PCA
from matplotlib import pyplot
from scipy import stats
import numpy as np

from sklearn.manifold import TSNE

def subFileInto2dArray(filename):
    f = open(filename, "r")
    subs = f.read()
    subs = lemmatizer.lemmatize(subs)
    subs = bow.prepareText(subs, string.punctuation)
    subs = bow.filterSubs(subs)
    lines = subs.split("\n")
    output = []
    for line in lines:
        output.append(nltk.word_tokenize(line))
    return output

def getSentenceArray():
    sub_array = glob.glob("subs/*--*.srt")

    lines = []
    output = []
    i = 0
    for sub_filename in sub_array:
        output += subFileInto2dArray(sub_filename)
        if i % 50 == 0:
            bow.printProgress(i, len(sub_array))
        i = i + 1
    return output



def createPCA(model, labelSets,groups):
    #Create X
    i = 0
    all_data = []
    for labels in labelSets:
        data = []
        all_labels = list(model.wv.vocab)
```

```
        newLabels = []
        for label in labels:
            if label in all_labels:
                data.append(model[label.lower()])
                newLabels.append(label)
        labelSets[i] = newLabels
        all_data.append(data)
        i = i + 1


    markers = ["o","s","P","*","+","x","d","2"]


    pca = PCA(n_components=2)

    i = 0
    for data in all_data:
        result = pca.fit_transform(data)
        pyplot.legend( loc = 'upper right')
        pyplot.scatter(result[:, 0], result[:, 1] , marker=(markers[i]),
label=(groups[i]))
        if len(labelSets[i]) < 35:
            for j, word in enumerate(labelSets[i]):
                pyplot.annotate(word, xy=(result[j, 0], result[j, 1]))
        i += 1
    pyplot.show()



def flattenArray(array2D):
    array1D = []
    for arr in array2D:
        array1D += arr
    return array1D
```

tfidf.py

```
def inverseFrequency(sub_freq):
    total_freq = bow.csvToDictionary('document_frequency/word_freq.csv')
    scores = dict()
    N = len(glob.glob('subs/*.srt'))
    for word, TFtd in sub_freq.items():
        word = str(word)
        if "'" in word or hasNumbers(word):
            continue
        try:
            if word in total_freq.keys():
                DFt = total_freq[word]
            else:
                DFt = 0
            if type(DFt) == 'NoneType':
                DFt = 0
            DFt += 1
            scores[word] = TFtd*math.log(N/DFt)
        except Exception as e:
            print("Word: "+word)
            print(m)
            print(str(e))
    return scores
```

train_program_multithreaded.py

```
import getArrays as ga
import bag_of_words as bow
import glob
import Word2Vec_PCA as wtv
import create_svm as csvm
import pandas as pd
import os.path
import numpy as np
import pickle
import random
from multiprocessing.dummy import Pool as ThreadPool
from functools import partial

print("----------------------------------------")
print("Adding Subtitles to Arrays")
print("----------------------------------------")
done_imdbs = bow.getDoneArray()
if os.path.isfile('part1.pkl'):
    done_so_far = pickle.load( open( 'part1.pkl', "rb" ) )
else:
    done_so_far = []

for imdbid in done_so_far:
    try:
        done_imdbs.remove(imdbid)
    except Exception as e:
        print(str(e))

i = 0
for imdbid in done_imdbs:
    try:
        imdbid = "{0:07d}".format(int(imdbid))
        subs = bow.openByIMDBID(imdbid)
        ga.addToGenreNotGenreArrays(imdbid,subs)
        i += 1
        done_so_far.append(imdbid)
        bow.printProgress(i, len(done_imdbs))
        with open('part1.pkl', 'wb') as file:
            pickle.dump(done_so_far, file)
    except Exception as e:
        print("Error with "+imdbid)
        print(str(e))



print("----------------------------------------")
print("Downloading Subtitles then Arrays")
print("----------------------------------------")
done_imdbs = bow.getDoneArray()
if os.path.isfile('part1.pkl'):
    done_so_far = pickle.load( open( 'part1.pkl', "rb" ) )
else:
    done_so_far = []

for imdbid in done_so_far:
    try:
        done_imdbs.remove(imdbid)
    except Exception as e:
        print(str(e))
```

```
i = 0
for imdbid in done_imdbs:
    try:
        imdbid = "{0:07d}".format(int(imdbid))
        subs = bow.openByIMDBID(imdbid)
        ga.addToGenreNotGenreArrays(imdbid,subs)
        i += 1
        done_so_far.append(imdbid)
        bow.printProgress(i, len(done_imdbs))
        with open('part1.pkl', 'wb') as file:
            pickle.dump(done_so_far, file)
    except Exception as e:
        print("Error with "+imdbid)
        print(str(e))

print("----------------------------------------")
print("Train Word2Vec Model")
print("----------------------------------------")

wtv.trainModel()


print("----------------------------------------")
print("Find Cost and Gamma")
print("----------------------------------------")

genres_fn = glob.glob('genre_corpus/*.csv')
genres = []

for fn in genres_fn:
    genre = fn.split('/')[1].split('.')[0]
    genres.append(genre)


pool = ThreadPool( 36 )

results = pool.map(csvm.getCostGamma, genres)
```