Analysis of the Relationship between Movie Titles and Corresponding Ratings and Reviews

Klara Banić, Ana Marija Ereš, Dominik Kos, Petra Omrčen, Petra Zanetti

Natural Language Processing course,
Faculty of Electrical Engineering and Computing, University of Zagreb

Abstract

In modern times, a great amount of data is being generated and collected each second using different tools such as surveys, reviews, comments all over the web. All this information is widely used by many companies and organisations in order to improve their services and products based on people's feedback. An example of such data are movie reviews which show personal opinion and give movie a rating. This paper's focus is divided in two parts – the first one does the analysis of movie titles and corresponding ratings, while the other performs analysis on the connection between movie reviews and its average rating. Sentiment analysis is included since it is an emerging research area nowadays, as well as word embedding and emotions vectors. Two datasets are used – one which holds movie titles and their ratings, and the second which represents movie reviews for a particular movie. The used data is preprocessed and partitioned into training and testing sets. The results are then studied in terms of accuracy.

Keywords: Movies, Sentiment Analysis, Word Embedding, Neural Network, Natural Language Processing

1. Introduction

In today's world, in the last decade and even more, there has been a lot of impact made by technology in everyday life. An immense growth in human's knowledge, as well as in area of new technologies research and development can be noticed. Another contribution to the society is the willingness of people to give honest opinions and sentimental content easily online which results in a lot of meaningful data being made each second. Social networks, along with numerous websites are the main source of information, especially when it comes to product ratings or movie reviews. One can cleverly gather the relevant information in a proper way in order to analyse the market and its behaviour. Data needs to be carefully collected and properly cleaned by using common ways such as scrubbing for incorrect data, handling missing data, standardizing and normalizing. Only then the proper analysis can take place. It is important to define the target of data analysis to know what has to be done and in which way. By using this kind of information, companies can understand the market trends and customer's feedback and therefore easily customize their products and offers to give response to the audience's needs.

The aim of this paper is to tackle and analyse movie titles with their corresponding ratings and reviews. First part of the paper focuses on exploring if the movie's rating can be induced and predicted based on its title. Four different approaches are given and analysed. Correlation between movie titles as word embeddings and ratings is explored following by analysing movie titles as emotion vectors. Research of correlation between the number of words in the title and its rating is provided, as well as correlation between the type of words in the title and its rating. This was done using the dataset which holds 755 records, where each of those inputs represents a movie through a unique ID, title, and rating, taken from a popular movie reviewing website Internet Movie Database (IMDb). The next part of the paper applies sentiment analysis and word embedding on movie reviews and its average rating. This is done on the second dataset which has 10 movies and for each one 1000 reviews.

2. Related work

B. Liu, 2016 [1] describes sentiment analysis as the field of study that analyses people's opinions, sentiments, attitudes, and emotions towards entities such as products, services, and their attributes. Although linguistics and natural language processing have a long history, there has been done little research about people's opinions and sentiments before the year 2000. Since then, the field has become a very active research area and there are several reasons for this. It has a wide arrange of applications, almost in every domain. The industry surrounding sentiment analysis has also flourished due to the proliferation of commercial applications. Accordingly, this provides a strong motivation for research. It also offers many challenging research problems, which had never been studied before. The most important thing about mentioned flourishing is that now there is a huge volume of opinionated data in the social media on the Web. Without this data, a lot of research would not have been possible.

Jiang et. al, 2012 [2] proposed an improved KNN algorithm for text categorization having combination of one pass clustering algorithm and K Nearest Neighbours algorithm. As stated, K Nearest Neighbours and Support Vector Machine had much better performance than other classifiers. However, KNN was the sample-based learning technique, which made use of all the training documents in order to predict the labels of the test document and had very huge text similarity computation. As a result, it could not be used widely in real-world applications. In order to tackle this problem, they proposed an improved KNN algorithm for text categorization based on one pass clustering algorithm and KNN algorithm. The proposed algorithm classifies test documents based on the cluster vectors instead of original text samples by using KNN approach which boosts way up the effectiveness of KNN approach.

M. Mamtesh & S. Mehla, 2019 [3] analyse movie reviews using K Nearest Neighbours, Naïve Bayes, and Logistic Regression. They are implemented in order to train the data set, to analyse the reviews and predict the sentiment of the reviews either positive or negative. The dataset is gathered from several sources for analysis, and stopping and Porter stemming are used to preprocess the data. Moreover, words with frequency greater than 80% of the dataset are ignored, as they are likely to be a stop-word. Likewise, words having very low frequencies should also be ignored. All in all, they have concluded that Naïve Bayes can be used ultra-efficiently and appropriately in implementing sentiment analysis on movie reviews / brand reviews and many more in order to understand the emotions and behavioural preferences of the consumer so as to provide better customer experience.

G.S. Brar & A. Sharma, 2018 [4] have done sentiment analysis of movie reviews using feature-based opinion mining and supervised machine learning. Their main focus is to determine the polarity of reviews using nouns, verbs, and adjectives as opinion words. Reviews are be classified into two different categories - positive and negative. Reviews of Open Movie Database is used as source data set and Natural Language Processing Toolkit for Part of Speech Tagging. The system is tested for 50 plus different movie titles each with maximum of 10 reviews and final results of total 500 reviews. The average accuracy of proposed system for test review is 81.22%. The system proposed by author in the paper can be used to classify a huge database of movie reviews. Best thing about their system is that it is a web-based API for sentiment analysis for movie reviews with JSON output to display results on any operating system.

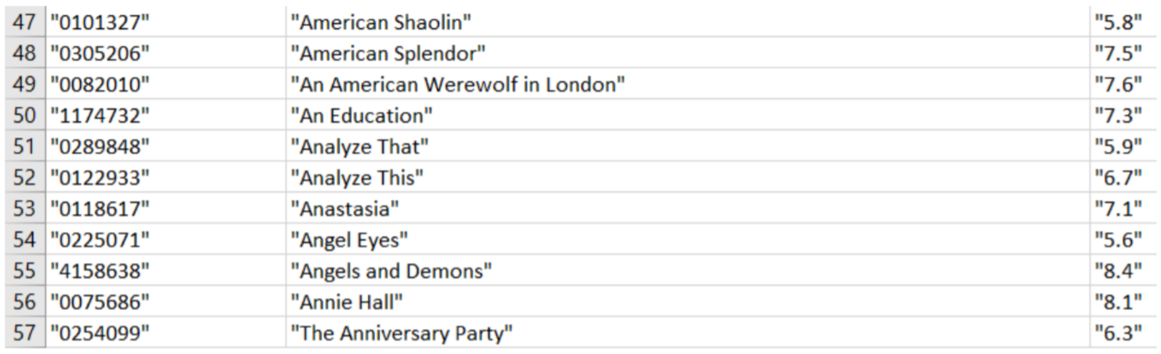
S.M. Qaisar, 2020 [5] uses Long Short-Term Memory classifier with Adam optimizer to automatically categorize the preprocessed IMDb movie reviews. The data set containing 50k movie reviews from IMDb, created by Andrew Maas, is utilized. The data have already been split into 25k reviews for training purposes while the other 25k is intended for testing the classifier. In addition, both sets contain 12.5k positive and negative reviews. The reviews are classified into positive and negative in reference to the IMDb rating system. Afterwards, the testing set is used to quantify the classification precision. Each movie review is encoded "vectorized" into a numeric value. It is achieved by utilizing the genism, a Python library for topic modelling and NLP. Three layers of LSTM are employed. The first layer employs 50 nodes to present the intended words. The second layer is the LSTM with 101 units of memory and the final layer creates 2 outputs corresponding to the considered classes. The results of confusion matrix are summarized and presented, and the accuracy score is found to be around 89.9%. Considering the accuracy score of the classifier, these results show that the classifier achieves an appropriate precision for the intended dataset. A better precision could be achieved by further cleaning the data and employing the ensemble classification approaches.

J.Y. Wu & Y. Pao, 2012 [6] experiment with different machine learning algorithms to predict the sentiment of unseen reviews from the improved corpus that has the additional sentiment information of all sub-phrases. They use the Rotten Tomatoes movie review corpus1 that has been greatly improved upon, and annotated with a fine sentiment score via Amazon Mechanical Turk. Their wish was to see whether having a finer sentiment annotation for every span of a parsed sentence in the training set would help improve the accuracy in predicting the overall sentiment of unseen sentences. They have trained and tested using Multinomial Naïve Bayes, Support Vector Machine and Deep Learning. It is worth nothing that their model tends to predict more positive than negative. This is because in the first setup, they rounded the AMT ratings to 0 and 1, which rendered an imbalance of the postive/negative polarity in the dataset (around 20% of the neutral data were rounded up to positive). They concluded that, with regards to achieving a performance better than that of their MNB and SVM models, their current 3-layer neural network is insufficient. Their results showed that this network performed well on unigrams and bigrams but suffered for most larger n-grams. An improved neural network could include representations for how words act on their neighbours, as well as which types of edges were used to construct the parse of the n-gram. Deep learning has a lot of potential for sentiment analysis, but their results here proved that a basic neural network with only word vector representations as features will perform worse than other classification techniques that build upon just bigram and unigram data.

3. Datasets

The first dataset for this paper consists of 755 records. Each record represents a movie through a unique ID, title, and rating, taken from a popular movie reviewing website Internet Movie Database (IMDb). The format of each record can be seen in Figure 1.

Figure 1: Example of Records in the First Dataset

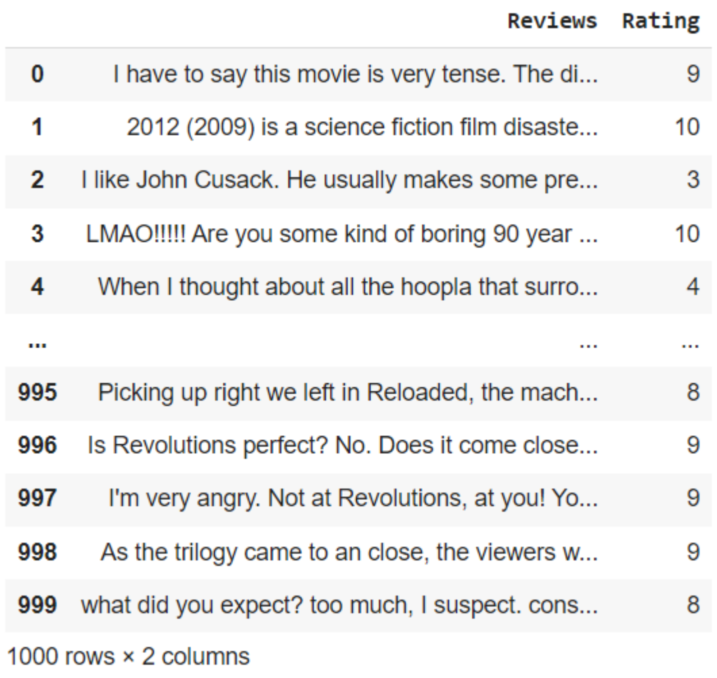


47	"0101327"	"American Shaolin"	"5.8"
48	"0305206"	"American Splendor"	"7.5"
49	"0082010"	"An American Werewolf in London"	"7.6"
50	"1174732"	"An Education"	"7.3"
51	"0289848"	"Analyze That"	"5.9"
52	"0122933"	"Analyze This"	"6.7"
53	"0118617"	"Anastasia"	"7.1"
54	"0225071"	"Angel Eyes"	"5.6"
55	"4158638"	"Angels and Demons"	"8.4"
56	"0075686"	"Annie Hall"	"8.1"
57	"0254099"	"The Anniversary Party"	"6.3"

This dataset is useful for this paper as it has direct mapping between a movie title and corresponding rating. It is used in the first part of paper where there is an exploration between colleration of movie titles and its ratings.

The second dataset used in this paper consists of 10 movie titles, and for each one there are corresponding 100 reviews. Each movie can be seen as a dataset where each record contains a movie review and its rating which is an integer between 1 and 10 which is presented in Figure 2.

Figure 2: Example of Records in the Second Dataset



	Reviews	Rating
0	I have to say this movie is very tense. The di...	9
1	2012 (2009) is a science fiction film disaste...	10
2	I like John Cusack. He usually makes some pre...	3
3	LMAO!!!!! Are you some kind of boring 90 year ...	10
4	When I thought about all the hoopla that surro...	4
...	...	...
995	Picking up right we left in Reloaded, the mach...	8
996	Is Revolutions perfect? No. Does it come close...	9
997	I'm very angry. Not at Revolutions, at you! Yo...	9
998	As the trilogy came to an close, the viewers w...	9
999	what did you expect? too much, I suspect. cons...	8

1000 rows × 2 columns

This dataset is needed for the second part of this paper where movie reviews are analysed through multiple methods.

Both datasets are cleaned and transformed in such way that is required for the methods used in this paper. Even though both datasets are not as huge as they could be, for the purpose of this paper we rely on them in hope to produce quality results.

4. Methods

4.1. Analysis of the relationship between movie titles and ratings

In this chapter we want to explore the possibility of predicting a rating of the movie based on its title. This problem is approached in four different ways: searching for correlation between movie titles as word embeddings and ratings, movie titles as emotion vectors and ratings, the number of words in the title and ratings, the type of words in the title and ratings.

In this part we are using a dataset that has 755 movie titles and corresponding ratings. The dataset is preprocessed leaving only relevant features for the analysis which are the column “title” and the column “rating”. Afterwards, the number of words in each title is counted and stored in a list. Every missing rating is replaced with an average value which is 5.

4.1.1 Correlation between movie titles as word embeddings and ratings

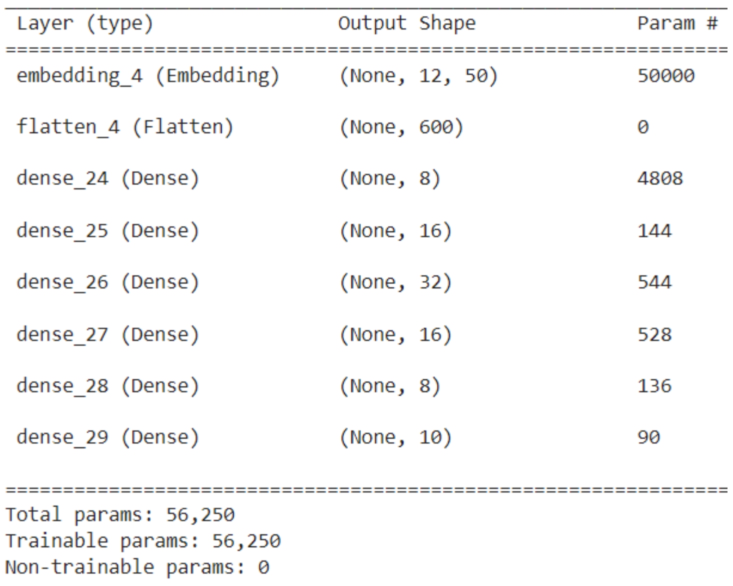
Machine learning models used in NLP take vectors, or more precisely arrays of numbers, as input. Since the data we use for NLP is strictly textual, the first step in applying a machine learning algorithm is coming up with a strategy to convert those strings into numbers, also referred to as vectorizing.

In recent years, word embedding has become the method to use to combat this problem, since it gives us an efficent, dense representation in which similar words have a similar encoding. Word embeddings are a class of techniques where individual words are represented as real-valued vectors, often tens or hundreds of dimensions, in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network. Words that have the same meaning have a similar representation. The learning process is either joint with the neural network model on some tasks, such as document classification, or is an unsupervised process, using document statistics.

In this part, embedding layer from Keras library is used as a word embedding algorithm. Firstly, the dataset is cleaned and prepared such that each word is one-hot encoded. Keras provides the *one_hot()* function that creates a hash of each word as an efficient integer encoding. We will estimate the vocabulary size of 1000, which is much larger than needed to reduce the probability of collisions from the hash function. The sequences have different lengths and Keras prefers inputs to be vectorized and all inputs to have the same length. We will pad all input sequences to have the length of 12 which is number of words in the longest title. The embedding layer is used on the front end of a neural network and there we specified our vector dimension to be 50.

2/3 of the dataset is training set and the remaining is testing set. Using TensorFlow we make a model. The model is shown in Figure 3. It is doing multiclassification with *keras.Sequential()* model. Other layers are added sequentially. Firstly, the Embedding layer is added which was described earlier and it is flatten to one dimension. After that 5 hidden layers are added: 2 with 8 neurons, 2 with 16 and 1 with 32. All of them have activation function "relu". The output layer has 10 neurons where each neuron represents one rating. Activation function of that layer is "sigmoid". Optimizer "Adam" is used and the loss function for the model is "SparseCategoricalCrossentropy". Inputs are embedded titles and outputs are ratings.

Figure 3: Model for Correlation between Movie Titles as Word Embeddings and Ratings



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 12, 50)            50000

 flatten_4 (Flatten)         (None, 600)               0

 dense_24 (Dense)            (None, 8)                 4808

 dense_25 (Dense)            (None, 16)                144

 dense_26 (Dense)            (None, 32)                544

 dense_27 (Dense)            (None, 16)                528

 dense_28 (Dense)            (None, 8)                 136

 dense_29 (Dense)            (None, 10)                90

=================================================================
Total params: 56,250
Trainable params: 56,250
Non-trainable params: 0
```

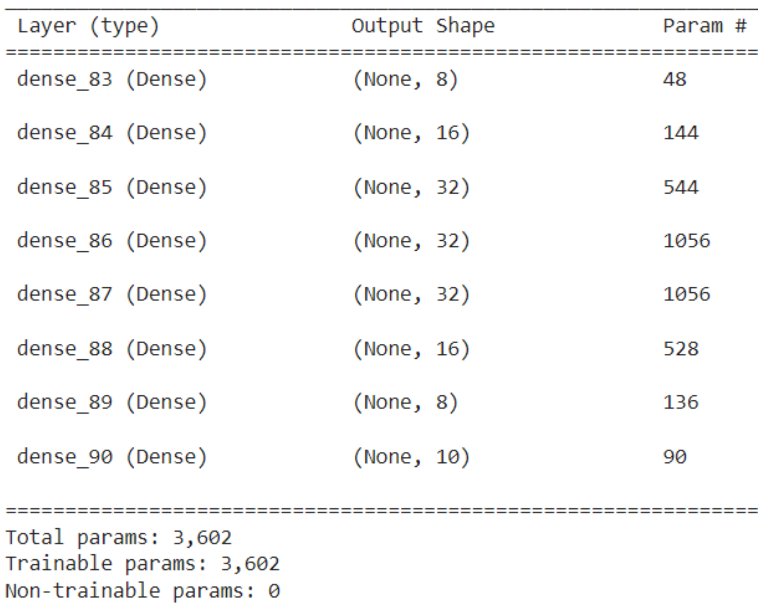
4.1.2 Correlation between movie titles as emotion vectors and ratings

Emotion Detection and Recognition from text is a recent field of research that is closely related to Sentiment Analysis. Sentiment analysis is a technique in natural language processing used to identify emotions associated with the text and determine whether data is positive, negative or neutral. Common use cases of sentiment analysis include monitoring customers' feedbacks on social media, brand and campaign monitoring. Sentiment analysis focuses on the polarity of a text (positive, negative, neutral) but it also goes beyond polarity to detect specific feelings and emotions (angry, happy, sad, etc.), urgency (urgent, not urgent) and even intentions (interested vs. not interested). Since humans express their thoughts and feelings more openly than ever before, sentiment analysis is fast becoming an essential tool to monitor and understand sentiment in all types of data.

Text2Emotion is a python library that is used to extract emotions from the titles in 3 steps: text preprocessing, emotion investigation and emotion analysis. It is compatible with five emotion categories: happy, angry, sad, surprise and fear. It takes a movie title and turns it into a list of 5 values. Each value represents the percentage of one out of five emotions the title has. We split our dataset into training set and testing set. Using TensorFlow a model is made. It is shown in Figure 4. The model is doing multiclassification with *keras.Sequential()* model.

Other layers are added sequentially. It has 7 hidden layers: 2 with 8 neurons, 2 with 16 and 3 with 32. All of them have activation function "relu". The output layer has 10 neurons where each neuron represents one rating. Activation function of that layer is "sigmoid". Optimizer "Adam" is used and the loss function for the model is "SparseCategoricalCrossentropy". Inputs are emotion vectors and outputs are ratings.

Figure 4: Model for Correlation between Movie Titles as Emotion Vectors and Ratings



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_83 (Dense)            (None, 8)                 48

 dense_84 (Dense)            (None, 16)                144

 dense_85 (Dense)            (None, 32)                544

 dense_86 (Dense)            (None, 32)                1056

 dense_87 (Dense)            (None, 32)                1056

 dense_88 (Dense)            (None, 16)                528

 dense_89 (Dense)            (None, 8)                 136

 dense_90 (Dense)            (None, 10)                90

=================================================================
Total params: 3,602
Trainable params: 3,602
Non-trainable params: 0
```

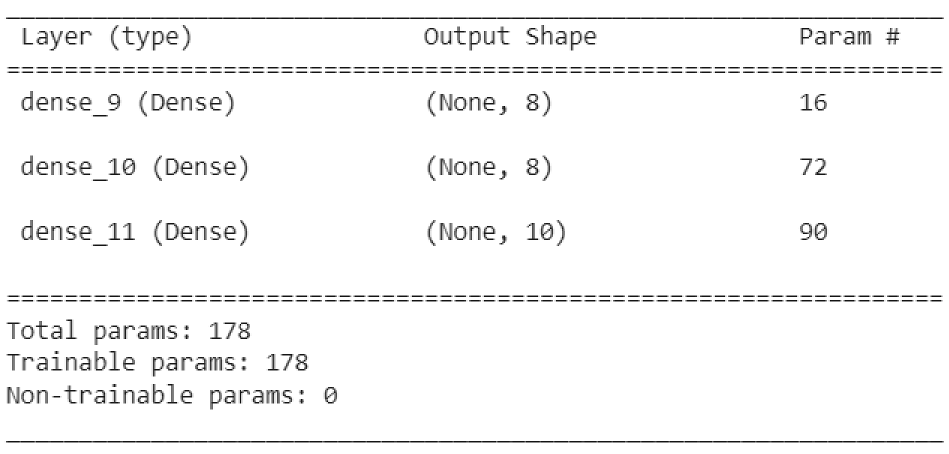
4.1.3 Correlation between the number of words in the title and ratings

In this chapter three different machine learning approaches are used to determine correlation between the number of words in the title and the rating. The first approach is multiclassification with deep learning models, the second is linear regression with a built-in model from *keras* library and the third is linear regression with a built-in model from *sklearn* library.

Dataset is first split into three subsets. The first subset is a training set containing 556/756 examples (74% of the initial dataset), the second subset is a test set containing 100/756 examples (13% of the initial dataset) and the third subset is a validation set containing 100/756 examples (13% of the initial dataset).

The first approach is multiclassification with a deep learning model. Neural network for multiclassification is done with *keras.Sequential()* model. It has 2 hidden layers with 8 neurons and an activation function "relu". The output layer has 10 neurons (each neuron represents one rating) and an activation function "sigmoid". Optimizer used in the model is "Adam" and the loss function for the model is "SparseCategoricalCrossentropy". Model is fitted over 200 epochs, with a batch size 256 (each epoch contains 256 examples). After each epoch, the model is tested with the examples from validation set. After fitting the model, it is evaluated on the test set. Evaluation displays its loss and accuracy.

Figure 5: Multiclassification keras.Sequential() Model



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 8)                 16

 dense_10 (Dense)            (None, 8)                 72

 dense_11 (Dense)            (None, 10)                90

=================================================================
Total params: 178
Trainable params: 178
Non-trainable params: 0
_________________________________________________________________
```

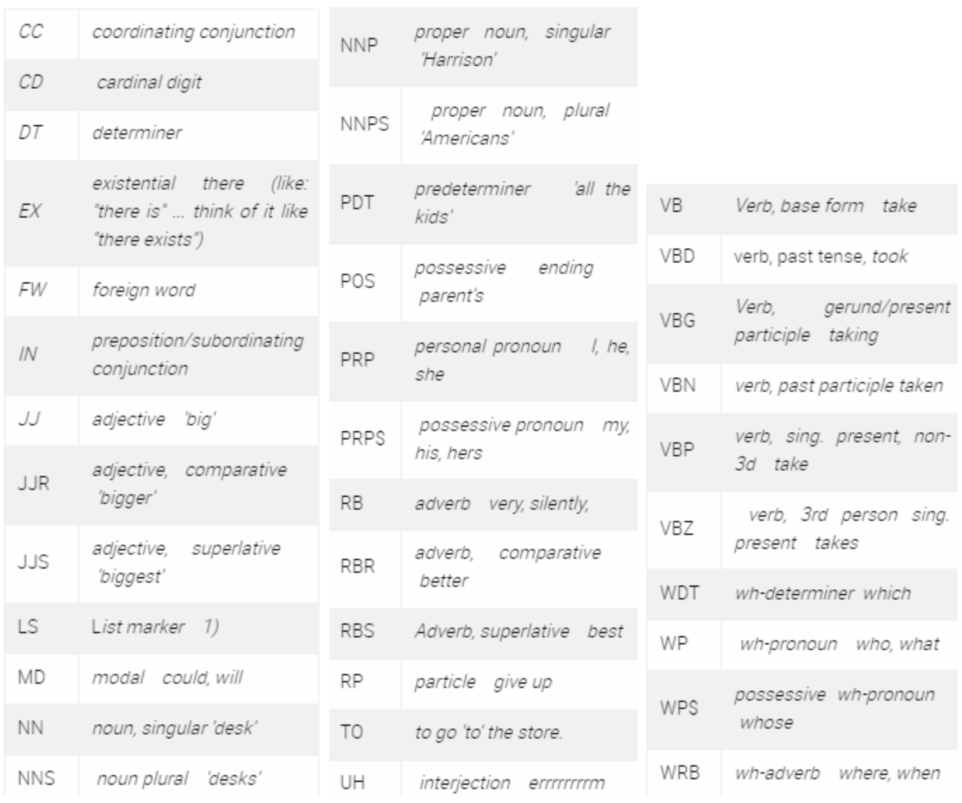
The second approach is linear regression model using *tf.keras.experimental.LinearModel()*. Optimizer is “stochastic gradient descent”, and the loss function is “mean squared errors”.

The third approach is linear regression using *sklearn.linear_model.LinearRegression()*. Difference between this approach and the second approach is that fitting is not done over epochs and therefore, the test set is bigger than the test set in the second approach (26% percent of the initial dataset).

4.1.4 Correlation between the type of words in the title and ratings

In this chapter the words in the movie title are divided into categories such as adjectives, verbs, nouns, adverbs etc. This process is called entity extraction, this is done in order to draw a clearer conclusion about a relation between a title of a movie and its rating. A title is first split into the list of smaller parts called “tokens” with functions from *nltk* library. After that POS Tagging, that is “Part of Speech” tagging, is performed on each title. POS Tagging is the most complex task in entity extraction. The goal is to match the tokens (words) with the corresponding tags (nouns, verbs, adjectives, etc.). The collection of tags used is known as a tag set which can be seen in Figure 6.

Figure 6: A Tag Set



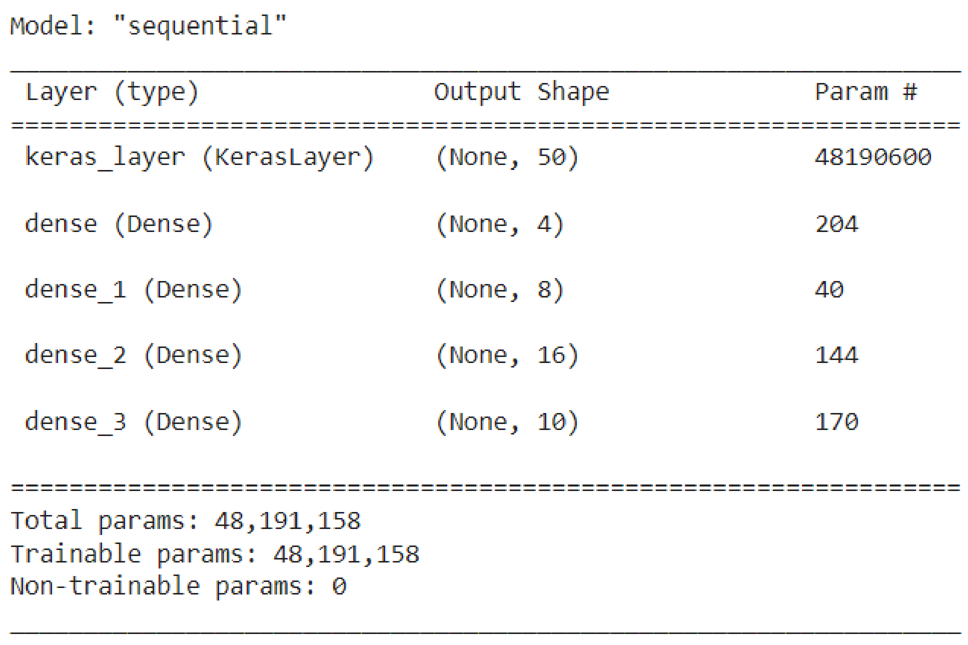
CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there (like: "there is" ... think of it like "there exists")
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective 'big'
JJR	adjective, comparative 'bigger'
JJS	adjective, superlative 'biggest'
LS	List marker 1)
MD	modal could, will
NN	noun, singular 'desk'
NNS	noun plural 'desks'
NNP	proper noun, singular 'Harrison'
NNPS	proper noun, plural 'Americans'
PDT	predeterminer 'all the kids'
POS	possessive ending parent's
PRP	personal pronoun I, he, she
PRP$	possessive pronoun my, his, hers
RB	adverb very, silently,
RBR	adverb, comparative better
RBS	Adverb, superlative best
RP	particle give up
TO	to go 'to' the store.
UH	interjection errrrrrrrm
VB	Verb, base form take
VBD	verb, past tense, took
VBG	Verb, gerund/present participle taking
VBN	verb, past participle taken
VBP	verb, sing. present, non-3d take
VBZ	verb, 3rd person sing. present takes
WDT	wh-determiner which
WP	wh-pronoun who, what
WP$	possessive wh-pronoun whose
WRB	wh-adverb where, when

In these approaches dataset is split into subsets the same way as in 4.1.3. chapter. Dataset is split into three subsets. The first one is a training set containing 556/756 examples (74% of the initial dataset), the second one is a test set containing 100/756 examples (13% of the initial dataset) and the third one is a validation set containing 100/756 examples (13% of the initial dataset).

Inputs in these approaches are a list of tokens (POS tags) for each movie title and outputs are ratings. As the input size depends on the number of words in the title, one way to represent them would be converting list of tokens (words) into embedding vectors. In this chapter four approaches have been used to determine the correlation between a type of words in the movie title and its rating.

In the first approach all input vectors are processed through previously implemented *hub_layer*. After being processed through the *hub_layer* all vectors will have the same size. This model performs multiclassification with *keras.Sequential()* model. The layers are stacked sequentially to build the classifier. The first layer is a TensorFlow Hub layer. This layer uses a pre-trained Saved Model to map a token into its embedding vector. The model that is used (google/nnlm-en-dim50/2) embeds each token and then combines the embedding. The resulting dimensions are (num_examples, embedding_vector). This fixed-length output vector is piped through a fully connected hidden layer with 4 neurons. This fixed-length output vector is piped through a fully connected hidden layer with 8 neurons. This fixed-length output vector is piped through a fully connected hidden layer with 16 neurons. The last layer is densely connected with 10 output nodes.

Figure 7: Model for the First Approach

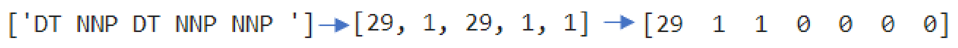


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 50)                48190600

 dense (Dense)               (None, 4)                 204

 dense_1 (Dense)             (None, 8)                 40

 dense_2 (Dense)             (None, 16)                144

 dense_3 (Dense)             (None, 10)                170

=================================================================
Total params: 48,191,158
Trainable params: 48,191,158
Non-trainable params: 0
_________________________________________________________________
```

Optimizer used in the model is “Adam” and the loss function used is “SparseCategricalCrossentropy”. Model is fitted over 40 epochs, with a batch size 256. After training the model, it is evaluated on the test set.

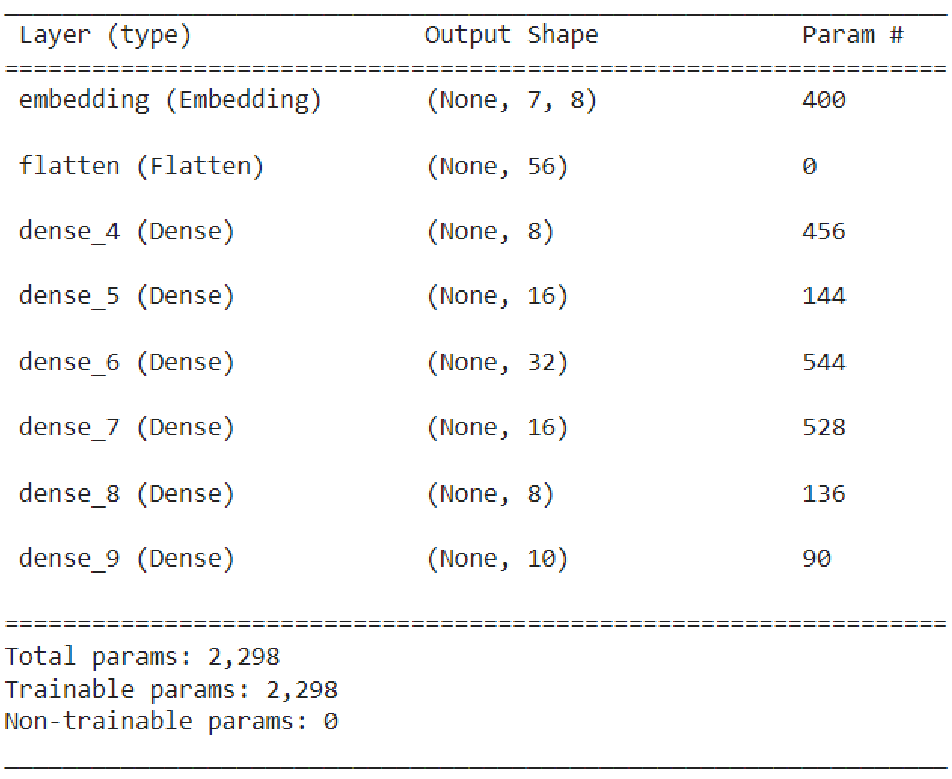
In the second approach word embedding is done differently. This model also performs multiclassification with *keras.Sequential()* model. Keras provides the *one_hot()* function that creates a hash of each word as an efficient integer encoding. Estimated vocabulary size is 50 as the tag collection contains only 35 tags. After this, inputs still have different lengths and Keras prefers inputs to be vectorized, that is prefers to have all inputs with the same length. With the built-in Keras function *pad_sequences()* all inputs will be padded to the *max_length* given in the function. Embedding layer in the model has a vocabulary of 50 and input length of 7. Chosen embedding space for the model is 8 dimensions. Therefore, the output of the embedding layer will be 7 vectors of 8 dimensions each, one for each word. This is flattened to a one 56-element vector to pass on to the Dense hidden layer which has 8 hidden neurons. Output of this layer is passed to the next layer which has 16 hidden neurons. The summary of the whole model is shown in Figure 9.

Figure 8: Enconding of POS Tags with one_hot() and pad_sequences() Functions



```
['DT NNP DT NNP NNP ']→[29, 1, 29, 1, 1] →[29  1  1  0  0  0  0]
```

Figure 9: Model for the Second Approach

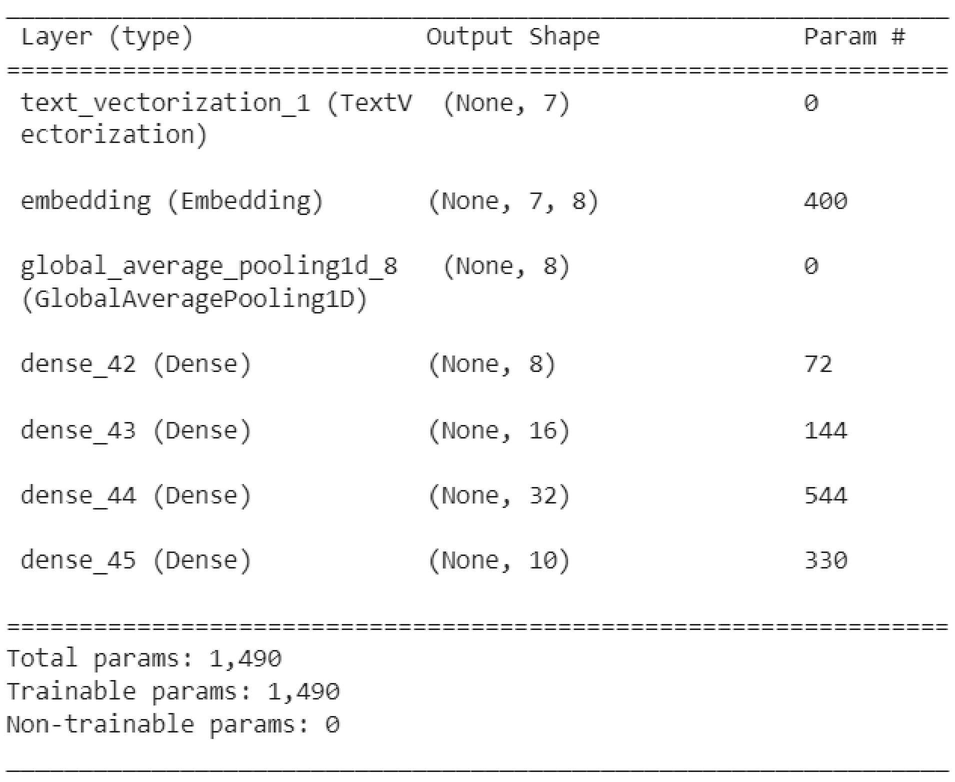


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 7, 8)              400

 flatten (Flatten)           (None, 56)                0

 dense_4 (Dense)             (None, 8)                 456

 dense_5 (Dense)             (None, 16)                144

 dense_6 (Dense)             (None, 32)                544

 dense_7 (Dense)             (None, 16)                528

 dense_8 (Dense)             (None, 8)                 136

 dense_9 (Dense)             (None, 10)                90

=================================================================
Total params: 2,298
Trainable params: 2,298
Non-trainable params: 0
_________________________________________________________________
```

Model optimizer is “Adam” and the loss function is “SparseCategoricalCrossentropy”. Model is fitted over 400 epochs and then evaluated.

In the third approach embedding is also done differently. This model also performs multiclassification with *keras.Sequential()* model. In this approach the built-in *TextVectorization()* layer is used. Text vectorization layer normalizes, splits and maps strings to integers and pins the output vectors to the same length. Model optimizer is “Adam”, and the loss function is “SparseCategoricalCrossentropy”. Model is fitted over 150 epochs, with a batch size 3 and evaluated.

Figure 10: Model for the Third Approach



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization_1 (TextV  (None, 7)                0
 ectorization)

 embedding (Embedding)       (None, 7, 8)              400

 global_average_pooling1d_8   (None, 8)                0
 (GlobalAveragePooling1D)

 dense_42 (Dense)            (None, 8)                 72

 dense_43 (Dense)            (None, 16)                144

 dense_44 (Dense)            (None, 32)                544

 dense_45 (Dense)            (None, 10)                330

=================================================================
Total params: 1,490
Trainable params: 1,490
Non-trainable params: 0
_________________________________________________________________
```

The fourth approach uses a linear regression model from library *sklearn*. Inputs are processed the same way as in the second approach.

4.2. Sentiment analysis and word embedding of movie reviews

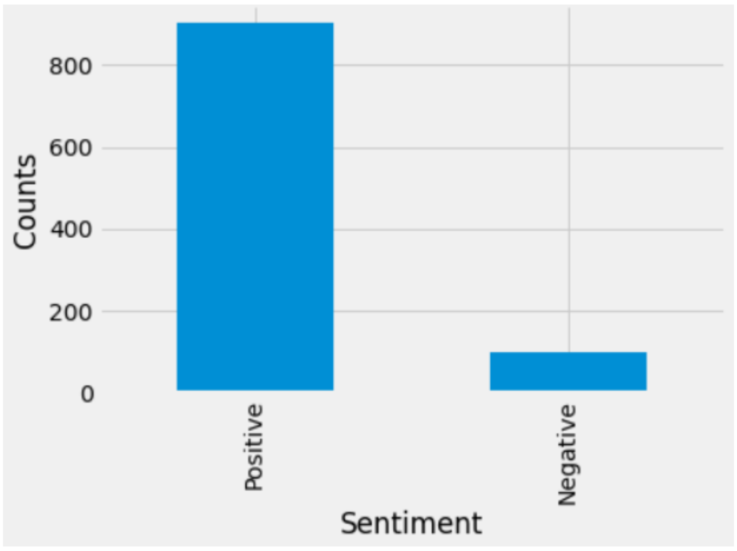
In this chapter, we perform the analysis of movie reviews by applying sentiment analysis and word embedding to text. For this, we use a database that contains movie reviews along with their ratings and sentiment polarity.

4.2.1 Sentiment analysis

First of all, it is necessary to read the data from a dataset. We are dealing with dataset of 1000 movie reviews with attached rating and sentiment. Data is saved using *pandas* library and its DataFrame so it is easy to access and manipulate data. Text is full of unnecessary characters so the next step is to throw them out. By using TextBlob library we are getting subjectivity and polarity for all the reviews. Polarity matches the sentiment we already have in our database and it is used in machine learning later on.

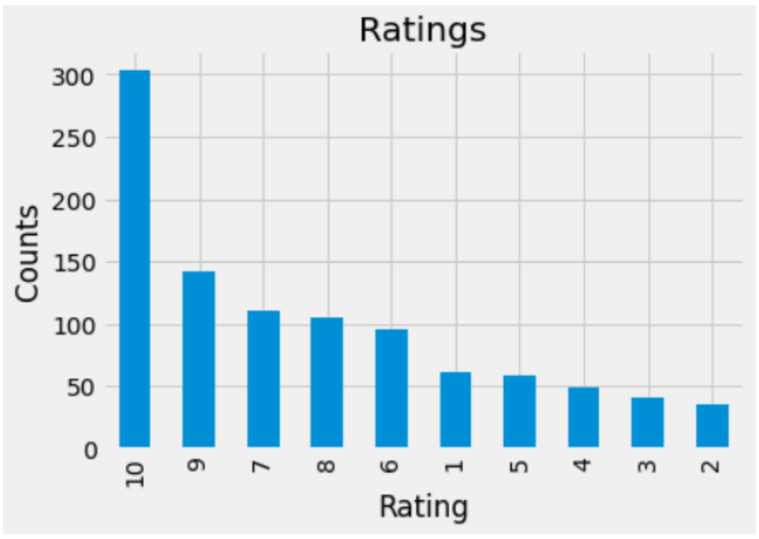
We were curious about what kind of reviews we are working with. We want to know if they are mostly positive or negative and what the distribution of ratings is. We then make a visualization of reviews as shown in the Figure 11. The reviews are mostly positive with an exact stake of 90.2%, which leaves 9.8% of negative reviews.

Figure 11: Distribution of Positive and Negative Reviews



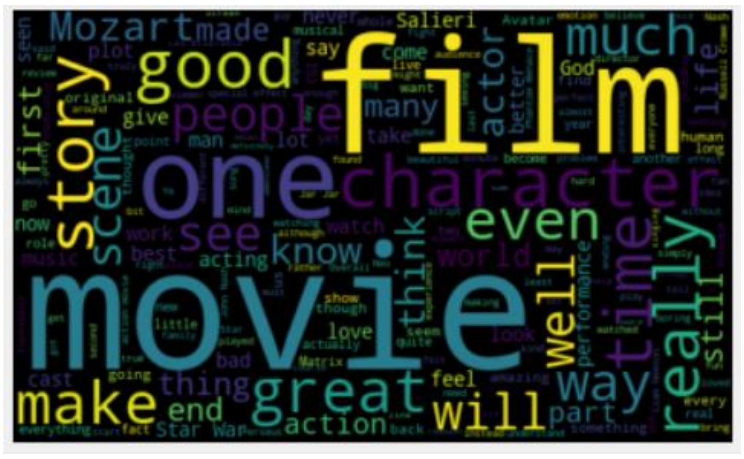
As shown in the Figure 12, most ratings are a 10. As the rating decreases, so does the number of reviews that have that rating. This is expected, as we have already found that most of the reviews are positive.

Figure 12: Distribution of Movie Ratings



To have a better look at the specific text, respectively we use *wordclouds* to see the most frequently used words in the reviews which result can be seen in Figure 13.

Figure 13: Most Frequent Words in Movie Reviews



Also, the same is done for positive and negative reviews separately to see which words appear more frequently in a more positive or negative context which is presented by Figure 14 and Figure 15.

Figure 14: Most Frequent Words in Positive Movie Reviews

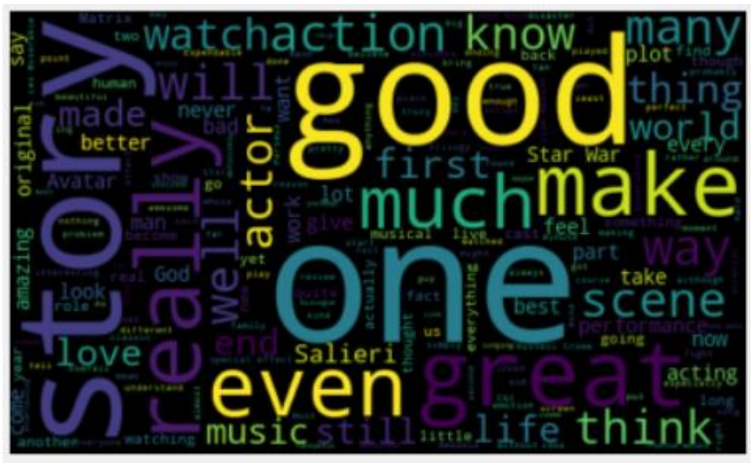
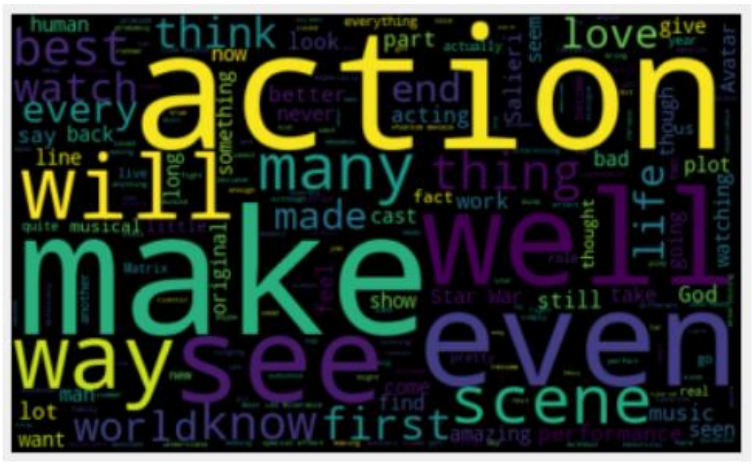


Figure 15: Most Frequent Words in Negative Movie Reviews



As seen above, a positive sentiment word cloud is full of positive (and some neutral) words and a negative word cloud is more neutral due to a smaller set of negative reviews. The words “good” and “great” initially appeared in the negative sentiment word cloud, despite being positive. This happened probably because they were used in a negative context, such as “not good”. Due to this, mentioned words were removed from the word cloud. Also, there are a few more words that were too neutral and common, like “movie”, “film”, “people”, “character”, “story”, “actor”, “much”, etc.

After the analysis, the next step is building the sentiment analysis model. This model takes reviews in as input. It then comes up with a prediction of whether the review is positive or negative. This is a classification task, so a logistic regression model is trained to do it. Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no (in our case positive or negative), based on prior observations of a data set. A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables.

Before anything else, we clean the data of all punctuation and split the data frame in only two columns - “Review” and “Sentiment”. Then we split the data frame into train and test sets. 80% of the data is used for training, and 20% for testing. Next, we use a count vectorizer from the Scikit-learn library. The vectorizer transformed the text in our data frame into a bag of words model, which contains a sparse matrix of integers. This needs to be done due to the logistic regression algorithm not being able to understand text. We build a simple logistic regression model and train the data on it. After that, we make predictions using the model.

Another approach on training sentiment analysis model uses deep learning. To accomplish that, we use *Torch* and *Sklearn* libraries. As in logistic regression, several steps had to be taken before the model could be developed. These steps are data preparation, encoding the sentiment (positive, negative) using Sklearn’s LabelEncoder, splitting the dataset into train, validation and test in ratio of 80%, 10% and 10% respectively, splitting the review text from *pandas* series to list of sentences.

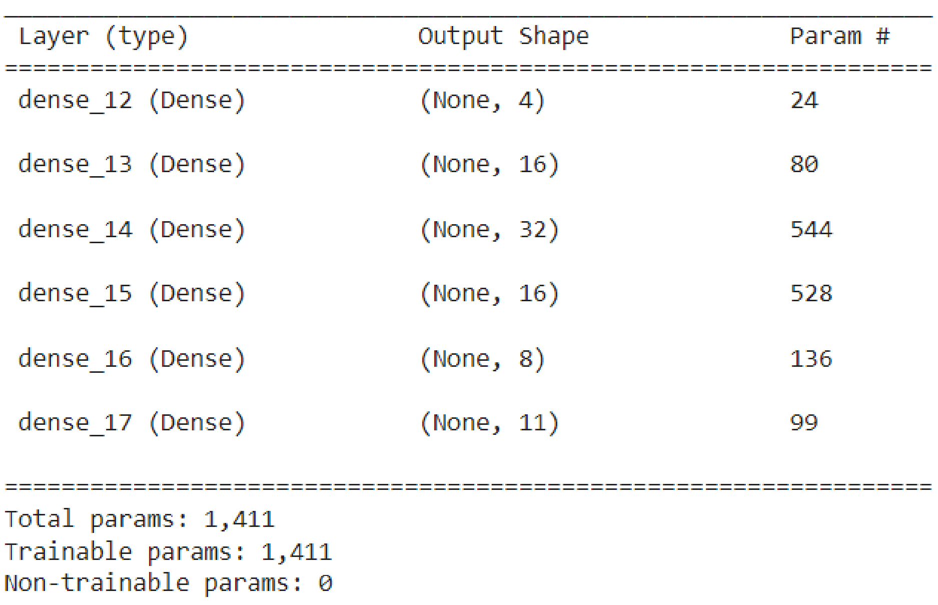
A class DataLoader is created for processing and loading data during the training and inference phase. The DataLoader initializes a pretrained tokenizer and encodes the input sentences. The output data is a dictionary consisting of 3 keys-value. Input_ids contains tensor of integers where each integer represents word from the original sentence. The tokenizer step has transformed the individual words into tokens represented by the integers. Attention_mask corresponds to a token in the same position in the input_ids. 1 indicates that the token at the given position sholud be attended to and 0 indicates that the token at the given position is a padded value. Labels are the target label.

We would like the model performance to be evaluated at intervals during the training phase so we define evaluation metrics with a function that accepts a tuple of (prediction, label) as argument and returns a dictionary of metrics. Next is the training phase where we set up the training arguments. The last thing to do is to instantiate the Trainer and start the training with the function trainer.train(). After the training has begun, evaluation is performed every 50 steps.

The last thing is turning reviews text in emotion word vectors using *text2emotion* similarly as in 5.1.2. Emotional vectors consist of five values that represent five emotions:

“happy, “angry”, “sad”, “surprise” and “fear”. With function *get_emotions()* we generate all the emotions and place them in vectors. That is now our representation of each review. The ratings need to be turned to integers if they are not already. Furthermore, we split the data into training and test sets. Using TensorFlow we make a model. Inputs are lists of emotional vectors and outputs are ratings. After that a small modification is made where the ratings are replaced as the output with polarity (positive and negative).

Figure 16: Model for Correlation between Emotional Vectors and Ratings



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 4)                 24

 dense_13 (Dense)            (None, 16)                80

 dense_14 (Dense)            (None, 32)                544

 dense_15 (Dense)            (None, 16)                528

 dense_16 (Dense)            (None, 8)                 136

 dense_17 (Dense)            (None, 11)                99

=================================================================
Total params: 1,411
Trainable params: 1,411
Non-trainable params: 0
```

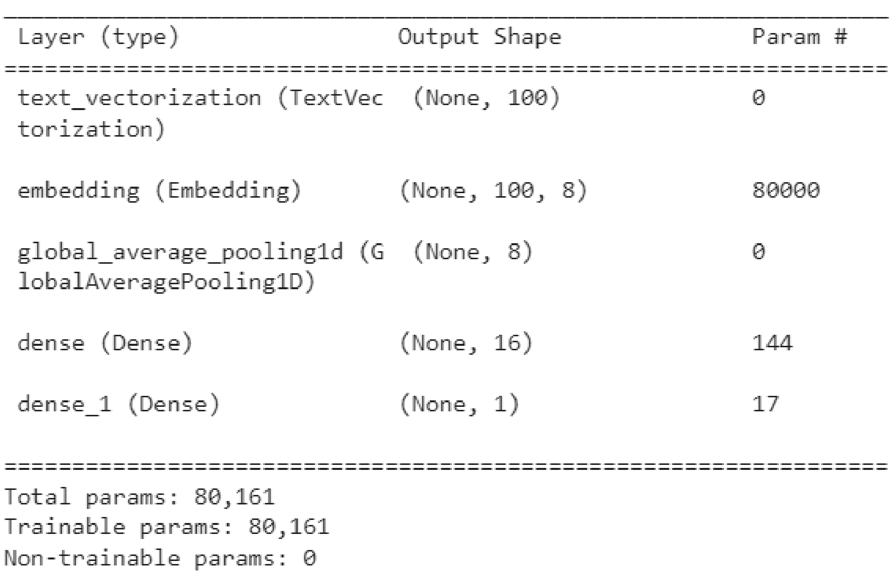
4.2.2 Word embeddings

Word embedding for our research is here performed on the second dataset after which a simple model is created to test the accuracy the model is able to achieve using word embedding.

The dataset is initially preprocessed in a way that it saves all the reviews and polarities of those reviews, so that the test set can easily compare the results of the model, and also ensure that we can measure accuracy of the model at the end since the main idea of this model is to check whether the polarities match, or more precisely whether the model considers the review to be positive or negative and compare it to the actual polarity. The next key part is to split the dataset into a train and test set, the former taking up 70% of the dataset, and the latter taking up the remaining 30%. This means that for all ten movies that persist in the dataset, 70% of the reviews are used for the training set, while the remaining 30% are the reviews that model would be tested on. To enhance performance, both sets are cached using Tensorflows very own AUTOTUNE variable. The last part of the preprocessing is to add an additional starting layer to our Keras model that we are going to build using *tf.keras.layers.TextVectorization*, since we are trying to find out the sentiment of the reviews. As the function for it is created, it is used to adapt our test set data into a vocabulary that will be used for the word embedding. The vocabulary size is set to 10000, while the sequence length is set to a 100.

After preprocessing the dataset, we are finally able to create our model. The model built is a sequential Keras model. The first layer is our layer of vectorized data run through the TextVectorization function. The second layer is the main part of this model, the Embedding layer. It is imported as a standard Keras layer, and the only parameter that it needs is the dimension in which the vectors are going to be saved, in our case, we will use 8-dimensional embeddings, since the dataset is rather small with about 7000 reviews used for the training set. The third layer is the GlobalAveragePooling layer which essentially collects the data from the Embedding layer into a 1-dimensional presentation. That data is then run through the fourth layer, the Dense layer with 16 nodes and the activation function set to ReLu, after which we can get the result of the model for a certain review, 1 being positive, and 0 being negative. The summary of the model can be seen in Figure 18.

Figure 18: Model for Word Embedding Analysis on Movie Reviews



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, 100)              0
 torization)

 embedding (Embedding)       (None, 100, 8)            80000

 global_average_pooling1d (G  (None, 8)                0
 lobalAveragePooling1D)

 dense (Dense)               (None, 16)                144

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
```

5. Results & discussion

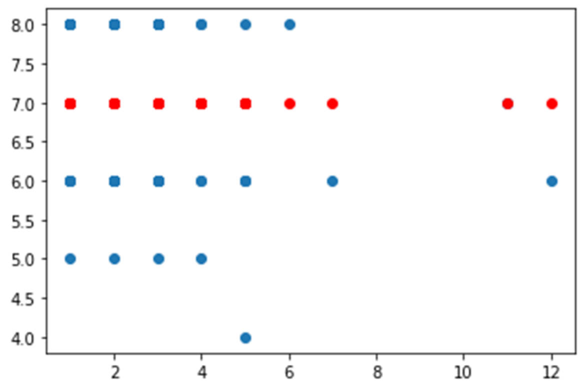
5.1. Analysis of the relationship between movie titles and ratings

The accuracy for multiclassification model testing the correlation between titles as word embeddings and its ratings is 38.31% and the loss is 10.86. Changing the various parameters did not help much, but such bad results were expected. Firstly, the problem with all methods is that the dataset used is too small. Secondly, it does not make sense that the accuracy is too high because movies usually do not get their ratings based on title.

The accuracy for multiclassification model testing the correlation between titles as emotion vectors and its ratings is only 31.58% and loss is 1.66. The problem with this method is in the Text2Emotion library. It does not give representative emotion vectors for movie titles because they are too short and, in some cases, too abstract to analyze.

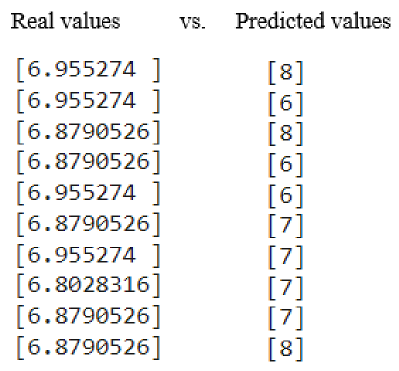
The accuracy for the multiclassification model testing the correlation between a number of words in the title and its rating is only 35% and the loss is 1.768. These results indicate that the correlation between these two features is not strong. X-axis of the graphs represents the number of words in the title and y-axis represents the rating for that movie. Blue dots represent true values while red dots represent predicted values. Figure 19 illustrates that the model is predicting values as an average rating of all movies in general, that is the predicted values conform to a constant line with a value 7 (y=7).

Figure 19: Evaluation of the Multiclassification Model Using Scatter Plot



Training a linear regression model with *keras* and *sklearn* library yields similar results. The loss is 0.9003, it is smaller than in the model for multiclassification.

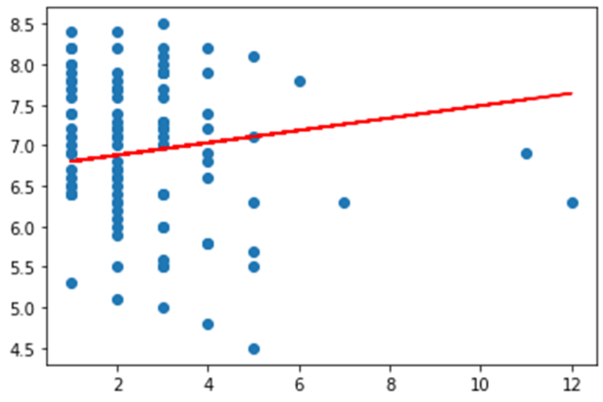
Figure 20: Random 10 Examples from the Test Set Displaying the Loss of the Linear Regression Model



```
Real values      vs.   Predicted values

[6.955274 ]              [8]
[6.955274 ]              [6]
[6.8790526]              [8]
[6.8790526]              [6]
[6.955274 ]              [6]
[6.8790526]              [7]
[6.955274 ]              [7]
[6.8028316]              [7]
[6.8790526]              [7]
[6.8790526]              [8]
```

In Figure 21, blue dots represent true values and a red line represents a collinear dependence between two features. Results are similar to previous case with a slight difference. In this case, since ratings are not rounded to confer to one particular class, line has a slight positive slope.

Figure 21: Evaluation of the Linear Regression Model Using Scatter Plot



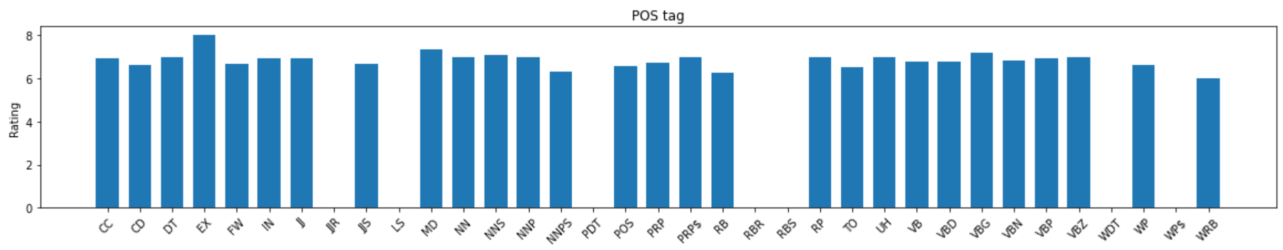
In conclusion, the number of words in the title does not correlate to the rating of the movie. Since more than 90% of the movies have less than 5 words in the title, these movies are taken as a representative sample. It can be observed that each value on the x-axis (number of words in the title) conforms to a normal distribution, meaning that predicted values are expected to conform to a constant or a line with a slight slope, which they do.

The accuracy for the multiclassification model testing the correlation between a type of words in the title and its rating is also around 35%. The accuracy is the same for all multiclassification models regardless of the implementation of the first embedding layer. This is intuitive since the implementation should not influence the result. These results indicate that these two features do not correlate. The loss for models is 1.4292.

The loss for the linear regression model is 0.8, it is smaller than in the multiclassification models. It is also smaller than the linear regression model for the correlation between the number of words in the title and the rating. This means that the correlation between type of words in the movie and the rating is stronger than the correlation between the number of words in the title and the rating.

Figure 22 indicates that all POS tags in the movie titles have the same average rating, meaning that POS tags and the rating of the movie do not correlate.

Figure 22: The Average Rating of a Movie with Specific POS Tag in the Title

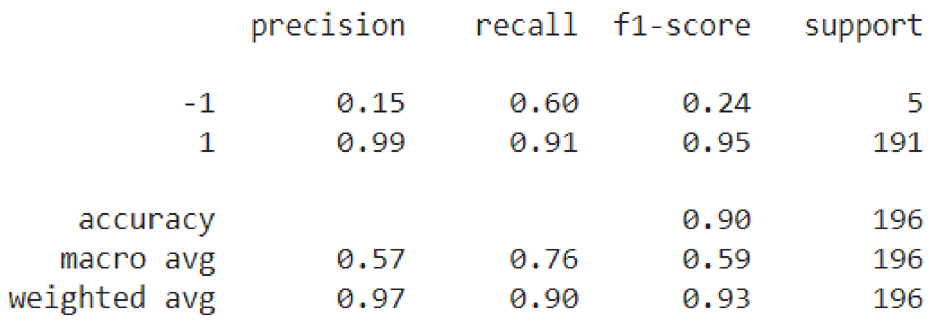


One of the reasons behind these poor results is that the dataset is too small to infer how a title with only a few words affects a rating. Also, another reason is that movies usually do not get their ratings based on title.

5.2. Sentiment analysis and word embedding of movie reviews

Logistic regression used for sentiment analysis gave the following results which are presented in Figure 23.

Figure 23: Results of Logistic Regression Used for Sentiment Analysis

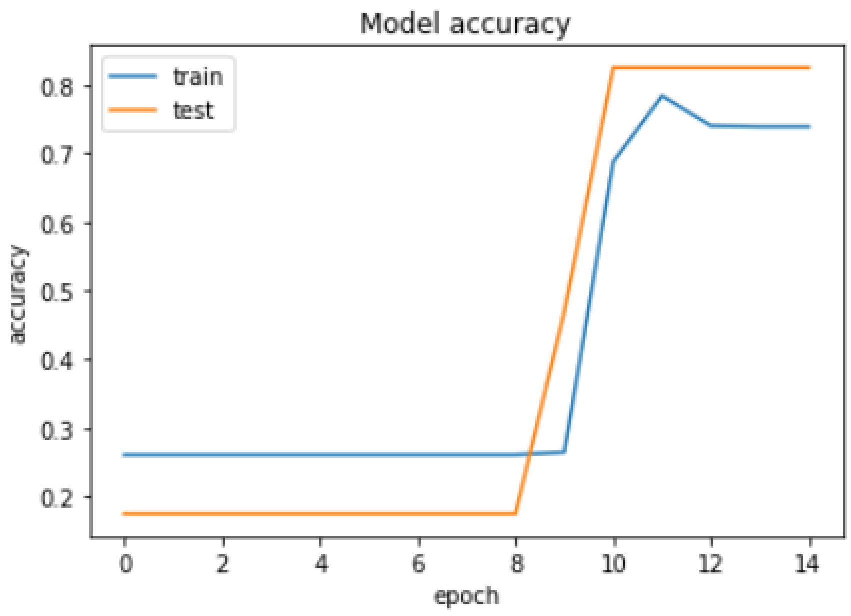


```
              precision    recall  f1-score   support

          -1       0.15      0.60      0.24         5
           1       0.99      0.91      0.95       191

    accuracy                           0.90       196
   macro avg       0.57      0.76      0.59       196
weighted avg       0.97      0.90      0.93       196
```

The accuracy is very good, 90%. Deep learning used for sentiment analysis also gave us great results. The test accuracy is 91% and the loss is 37.87%. Initially a problem occurred when training the model because of RAM deficiency. The problem was solved by decreasing the batch size in training arguments. The only bad result is with emotional vectors. Accuracy is only 34.83%. Changing the size and number of layers did not help much but such poor results were expected. Namely, the vector of emotions cannot really tell us if the movie was good or bad because if someone was sad or scared while watching the movie, that does not mean it was bad, it may be great for the audience to experience those emotions. On the other hand, when we trained the model so that it only outputs whether the film is positive or negative, the accuracy improved significantly, over 80%. We can conclude that the vectors of emotions can still give us information about whether viewers liked the movie or not but not enough to be able to make good predictions on ratings from 1 to 10 that we were working with.

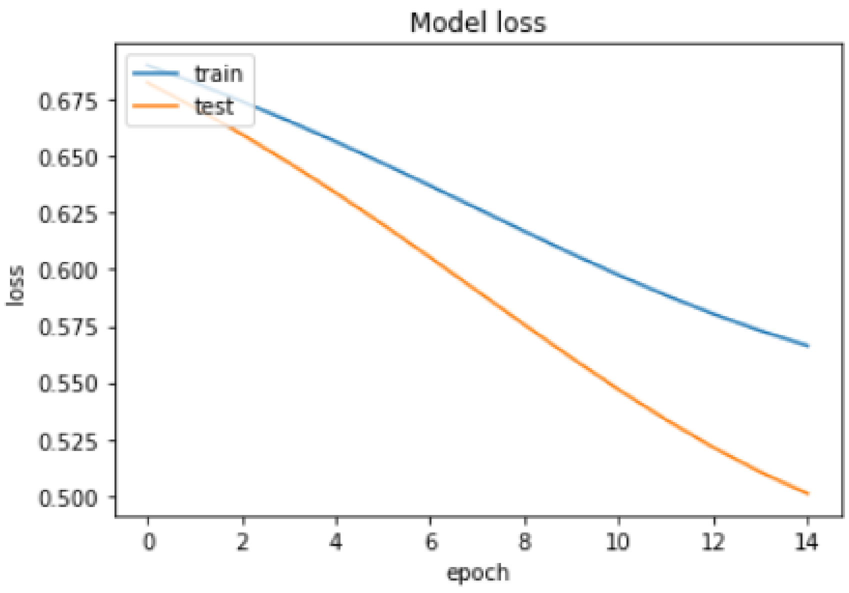
After compiling the word embeddings predicting polarity model using standard Adam optimizer, and using binary cross-entropy loss function, we can test our model on the test set, where it achieves maximum accuracy of about 82.5%, after 15 epochs. The results can be seen in Figure 24.

Figure 24: Accuracy of the Model Using Word Embeddings



Apart from the accuracy we can see the results of our loss function for each epoch in Figure 25.

Figure 25: Loss Function of the Model Using Word Embeddings

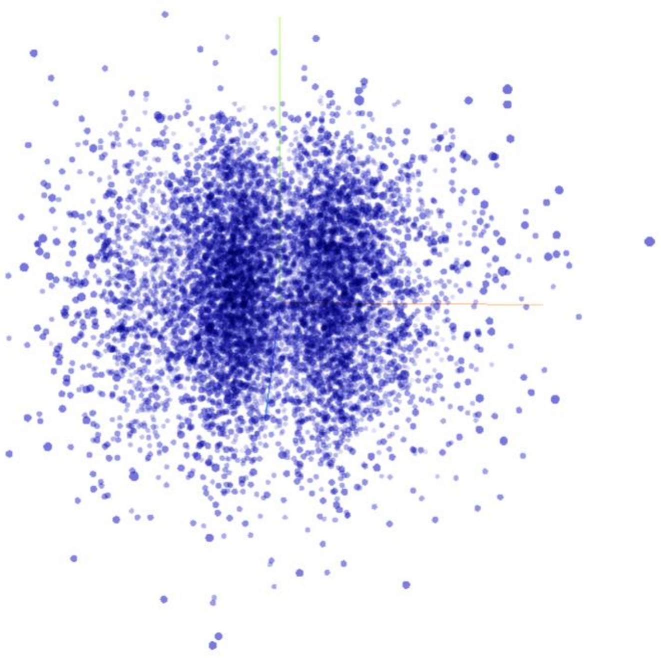


From the results of our accuracy, we can conclude that a much higher accuracy is not really attainable since it essentially reaches its peak after 12 to 13 epochs, and then barely even rises. To combat that effect, we would need a much bigger dataset, that could then give us a basis for using a larger vocabulary, and sequence length. Apart from that, larger embeddings dimension can provide us additional accuracy but that would oversaturate the model with additional unnecessary parameters and would not substantially help improve the accuracy.

On the other hand, our loss function seems to be steadily declining well after the 12th epoch which can indicate potential overfit of the model. This proves that additional dimensions to the embeddings would be unnecessary for the model itself. Potential resizing of the vocabulary might help combat the problem of overfitting but the main solution would be as mentioned previously a much larger dataset of potentially hundreds of thousands or millions of reviews which could then reach higher accuracies than the model used here.

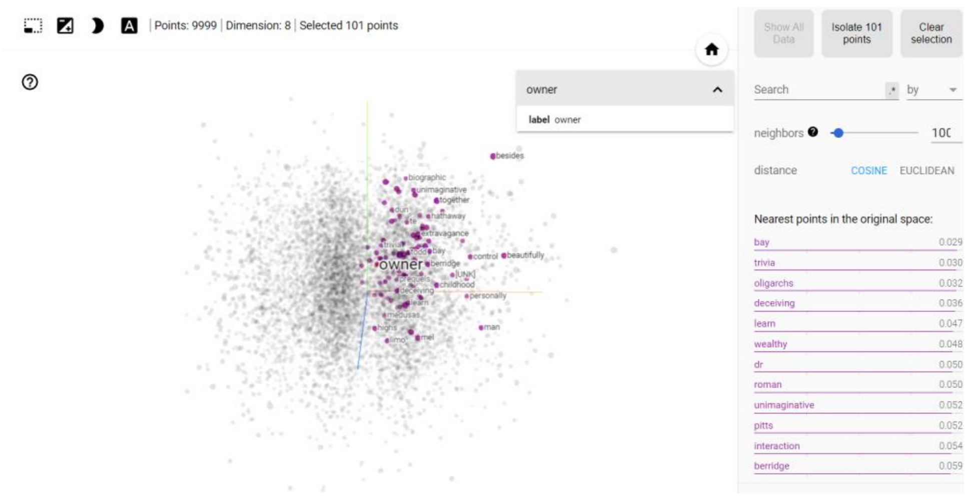
As a part of the analysis of the results, we export the values of the Embedding layer into a .tsv file which can then be used on the website Embedding Projector to visualize all of the embeddings in a vector space as seen in Figure 26.

Figure 26: Visual Vector Representation of the Models Word Embeddings



As presented in Figure 27, the highlighted nodes and their words represent the nearest neighbours of the node, in this example node labelled owner. They can all be found in a similar vicinity from one another. This further proves the initial assumption of the word embedding method itself that words with similar meanings have similar encodings and end up closer to one another on average in the model.

Figure 27: Nearest Neighbours of the Node Labelled Owner



Nearest points in the original space:

bay	0.029
trivia	0.030
oligarchs	0.032
deceiving	0.036
learn	0.047
wealthy	0.048
dr	0.050
roman	0.050
unimaginative	0.052
pitts	0.052
interaction	0.054
berridge	0.059

6. Conclusion

In this paper, movie titles and corresponding ratings and reviews are used to perform different types of analysis. First part of the paper focused on finding a connection between a movie's title and its rating, as well as analysing the movie titles as emotion vectors. The correlation between the number of words in the movie title, following by correlation between number of words in the movie title and its rating is researched. This part does not show meaningful results, which is expected, as movies usually cannot be rated only on their title. Another reason for non-significant results is the size of the dataset which is obviously not big enough to make clear conclusions. Further work would do the same methods but on a different dataset, and only then an improvement could be made. Second part of the paper applied sentiment analysis and word embedding on movie reviews and its average rating. Somewhat better results are achieved here, especially in sentiment analysis, as each movie rating is indeed connected to a movie rating, and therefore predictions can be made more appropriate. The future scope of work could combine the methods presented in this paper, as well as use a larger dataset to obtain more accuracy.

Acknowledgment

This paper is an output of the project within Natural Language Processing course at Faculty of Electrical Engineering and Computing, University of Zagreb.

References

[1] B. Liu, "Sentiment analysis and operation mining," Synthesis Lectures on Human Language Technologies, 2016, pp. 152-153.

[2] S. Jiang, G. Pang, M. Wu, L. Kuang, "An improved K-nearest-neighbor algorithm for text categorization", vol 39, 2012, pp. 1503-1509.

[3] M. Mamtesh, S. Mehla, "Sentiment Analysis of Movie Reviews using Machine Learning Classifiers," International Journal of Computer Applications, 2019, 182, 25-28.

[4] G.S. Brar, A. Sharma, "Sentiment Analysis of Movie Review Using Supervised Machine Learning Techniques", 2018.

[5] S.M. Qaisar, “Sentiment Analysis of IMDb Movie Reviews Using Long Short-Term Memory”, 2020.

[6] J.Y. Wu, Y. Pao, “Predicting Sentiment from Rotten Tomatoes Movie Reviews”, 2012.