02462 SIGNALS AND DATA, DTU COMPUTE, SPRING 2021

EXAM PROJECT ON TEXT CLASSIFICATION AND SENTIMENT ANALYSIS

Andreas Fiehn (s204125)
August Brogaard Tollerup (s204139)
Johan Böcher Hanehøj (s194495)

Group 17

1. INTRODUCTION

In this project we analyze and classify text using natural language processing tools such as GloVe, FastText and sentiment analysis.

We will try to classify IMDB movie categories based on the movie's plot. Here we will make use of GloVe and FastText. Then we will do a sentiment analysis on Donald Trump's inauguration speech.

First, we will make use of pretrained word embeddings from the Stanford project, GloVe [1]. We use the GloVe word embeddings pretrained on Wikipedia 2014 and the Gigawords database, which gives us a total vocabulary of 400000 words. Furthermore we will implement FastText, which is a Facebook-developed method, and compare it to the GloVe method. In our case the FastText embeddings will be trained on our texts with the purpose of predicting the categories. We suspect that the FastText method will be more precise in it's classification, but also run slower since it is more complex. We use pretrained GloVe word embeddings which means we just have to clean our text and load these embeddings and then we are ready to make classifications. Also we expect that after running PCA on the embeddings, that the principal components of the FastText-embeddings will be more indicative in respect to the categories, since it is trained with this as a purpose.

Lastly we conduct a sentiment analysis on the inauguration speech of Donald Trump. To understand how the sentiment score changes throughout the speech we will split the speech into smaller windows and plot the window-wise sentiment scores. We hypothesize that the speech will be packed with emotion especially towards the end of the speech. And since an inauguration speech is in a sense a *victory* speech, we expect it to be mostly positive.

2. METHODS

2.1. IMDB classification

2.1.1. The dataset

We make our classifications on data retrieved from IMDB with IMDbPy [2]. The data provided contains 4405 movie entries distributed across four genres, Family, Sci-Fi, Thriller and Romance. These are the classes that we strive to predict based on the plot of the movie. The data is split in a 90/10 train/test-split, which means, our train-data consists of 3964 entries whilst we will test our classification model on 441 movies.

2.1.2. GloVe

First, the pretrained GloVe-embeddings are loaded. We make use of the word embeddings with 100 dimensions.

We clean the movie plots by stripping all special characters and making everything lowercase.

The baseline method is used to represent the n'th text as a vector, z_n^{GloVe}.

$$z_n^{\text{GloVe}} = \frac{1}{N_n} \sum_{i=1}^{N_n} w_i$$

This is simply the average GloVe emebedding for the n'th text. We loop through each text and find the average for the text in question.

2.1.3. PCA

After representing the document as a vector we can use PCA to project the data onto the principal components which explains the most variance in the data. This is done by first calculating the mean of the data.

$$\bar{x} = \frac{1}{N} \sum_{j=1}^{N} x_j$$

Next we can calculate the covariance matrix of the data.

$$\Sigma = \frac{1}{N-1}\sum_{j=1}^{N}(x_j - \bar{x})(x_j - \bar{x})^\intercal$$

Since we use 100 dimensional word embeddings, the covariance matrix will be a $\Sigma \in \mathbb{R}^{100\times 100}$ matrix. Now we can find the eigenvectors and eigenvalues of the covariance matrix.

$$\Sigma = VDV^\intercal$$

The eigenvectors are sorted in descending order according to their eigenvalues and since the eigenvalues, λ_i, describe the variance, we pick the K first eigenvectors with the highest variance, and then we can project our data onto this new basis.

$$z_i = V^\intercal(x_i - \bar{x})$$

We will experiment with the amount of principal components, K, and plot the explained variance to get an idea of how much variance is explained by the eigenvalues. Based on this we will choose a suitable amount of principal components, K

2.1.4. Classification

Now that the embeddings are projected onto the principal components, we will have to classify it accordingly. For this we will calculate the cosine similarity between the test-data and training-data and classifiy the plot as the most frequent label in the top 10 closest matches in the training-data. The cosine similarity is the cosine of the angle between the two vectors and give us an idea of how the two vectors' directions are in comparison to each other. [3, chapter 2.4.7]

$$\text{similarity} = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Another classification model we will be using on the data is the RandomForest classifier, which is part of the SciKit Learn Python library.[4] We will not elaborate on the theory behind this model, but rather use it as a reference that is easy to set up and use. Briefly explained, it works by generating decision trees based on the features and labels, and from these it will be able to make a majority vote classification. Our main focus will be the cosine similarity of the principal components.

2.1.5. FastText

Another approach we will be using to make classifications is FastText, which is developed by FaceBook's research department.

The idea behind FastText is to represent each document as a vector just as we did with the baseline method for the GloVe embeddings. In FastText our vocabulary will consist of 1-grams, 2-grams and 3-grams, e.g. single words, two words and three words together. An example could be the text consisting of three words (1-grams), "I love data". With this text our domain of data would then look like this, with $b_i \in \mathbb{R}^D$ being our D-dimensional embedding for each n-gram.

$$\mathcal{D} = \{b_{\textbf{I}}, b_{\textbf{love}}, b_{\textbf{data}}, b_{\textbf{I love}}, b_{\textbf{love data}}, b_{\textbf{I love data}}\}$$

By taking the mean of the vector embeddings in $\mathcal{D}$ we can reduce it into a single vector, which then represents our text. If we have more texts we can represent the *n'th* text as the mean of the n-gram embeddings:

$$z_n = \frac{1}{N_n}\sum_{i\in\mathcal{D}_n} b_i$$

We multiply the text-representation z_n with a matrix of weights, A and apply a softmax layer and then we have a probability distribution.

$$\mathcal{P}(y_n|A, z_n) = \text{softmax}(Az_n)$$

We use a FastText classifier, based on an example from PyTorch.[5] and the same approach as earlier where we use PCA on the FastText embeddings and then use the cosine similarity to make a classification.

2.2. Sentiment Analysis

2.2.1. Speech Data

We start by cleaning the speech data so that we can compute the sentiment scores for each word. Just like we did with the IMDB plots. With the cleaned text we can now map each word to it's respective sentiment score from the afinn library.[6] We use the English sentiment lexicon provided by the afinn library since the speech is English.

2.2.2. Sentiment Score

We can compute the sentiment score for a given text using the following formula:

$$S = \sum_{i=1}^{N} x_i s_i = \mathbf{x}^\top\mathbf{s}$$

where **x** is a vector consisting of all words in the cleaned text and **s** is a vector (the afinn lexicon) with sentiment scores for individual words. We can thus compute the sentiment score S as the inner product between **x** and **s**.

2.2.3. Window wise sentiment score and filtering

To understand how the sentiment changes throughout the speech we split the cleaned text into smaller windows of size

L with a stride of J. Resulting in a vector consisting of all the windows where the first element is the sentiment score from word 0 to word $\mathbf{L-1}$, the second element is the sentiment score from word J to word $\mathbf{J+L-1}$, the third element is the sentiment score from word 2J to word $\mathbf{2J+L-1}$, etc.[7] With this windows vector we can now compute the windows wise sentiment scores and visualize the results on a plot. Lastly we apply a smoothing filter to the windows wise sentiment vector. We used a 1D Gaussian kernel to smoothen the data. The kernel fits a gaussian normal distribution with a certain standard deviation to the original signal which will result in smoothened data. The filter is applied through convolution which can be described as the following operation:

$$(y * h)(n) = \sum_{m=0}^{l-1} h[m]y[n-m]$$

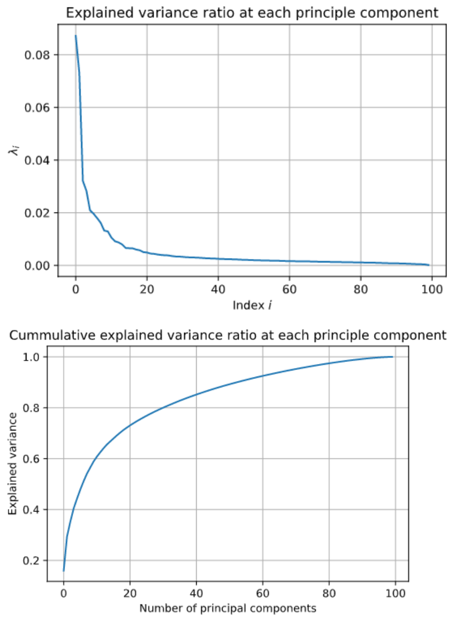
where y is the signal to be filtered, h is the convolution filter and l is the length of a vector.

3. RESULTS

3.1. IMDB classification

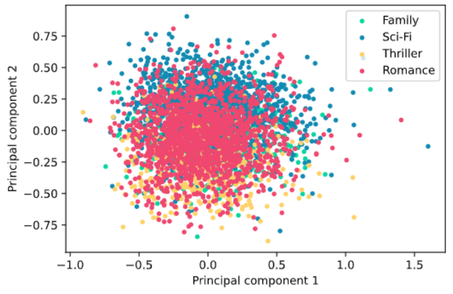
On figure 1 we see a plot of the explained variance ratio and the cummulative explained variance for the GloVe principal components. Here we see, that we would need around 30 principal components to explain 80% of the variance in the data and at around 60 principal components we can explain roughly 90%.

Fig. 1. GloVe: Explained variance ratio plot and cummulative explained ratio plot variance



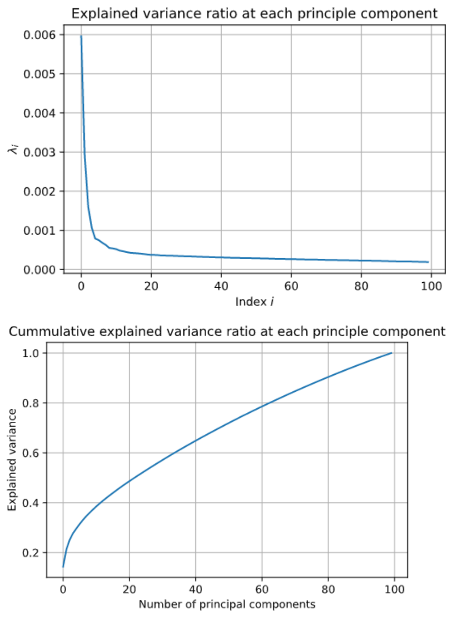
The explained variance plot can be used to decide how many principal components we would need to make our predictions. To get an idea of the principal components we have plotted the first two principal components from the GloVe embeddings on figure 2. We have also plotted the second and third principal component aswell as the third and fourth principal component together. These can be seen in Appendix. Here we see that the the labels are scattered and we cannot tell much from these first two principal components. We have discussed it further in section 4.

Fig. 2. GloVe: First and second principal components plot



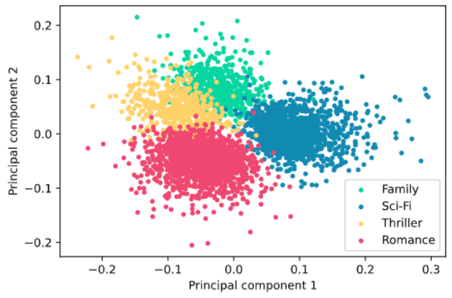
The explained variance for the PCA run on the FastText embeddings shows us a much lower cummulative explained variance, which can be seen on figure 3. At around 60 principal components we can explain 80% of the variance.

Fig. 3. FastText: Explained variance ratio plot and cummulative explained ratio plot variance



In contrast to the GloVe embeddings, we see that the FastText embeddings' first two principal components are clustered with their labels.

Fig. 4. Fasttext: First and second principal components plot



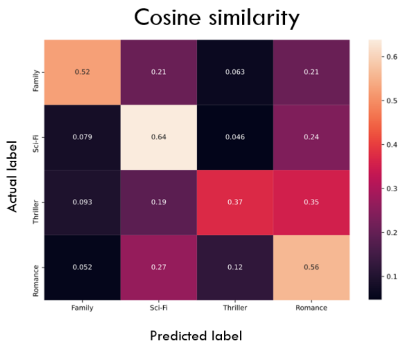
Based on our explained variance and the principal component plots we have decided to test our GloVe model on 2, 5, 10, 20, 30 and 40 principal components, respectively.

In the following table our test accuracy scores for the GloVe embeddings are shown for the different models.

		Test accuracy					
	K, principal components	2	5	10	20	30	40
GloVe	PCA/Cosine similarity	39.23%	44.90%	50.57%	50.11%	55.78%	49.21%
	PCA/RandomForest	39.23%	53.97%	63.72%	65.98%	66.44%	66.44%

We have created a confusion matrix for the GloVe PCA Classifier on figure 5. The columns represent labels, while the rows represent predictions of each label. We see that the best classified labels are ”Sci-Fi” and ”Romance”.

Fig. 5. Confusion Matrix: GloVe/Cosine similarity at $k = 30$ principal components. Confusion matrices for the other classifiers can be seen in Appendix



For the FastText embeddings we have tested our model on 3 and 5 principal components as well, because it seems like more can be told about the genres by the first principal components than with the GloVe embeddings.

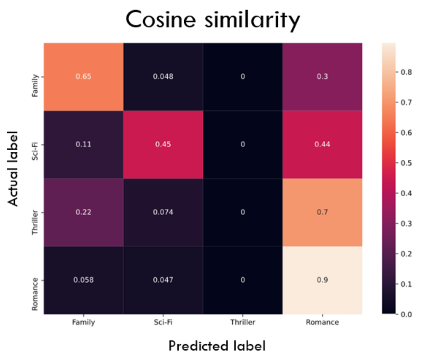
On figure 5 we see the confusion matrix for the cosine similarity classifier on the FastText embeddings. We see that it has not predicted anything to be ”thriller”. and ”romance” is the genre that is predicted the most.

From the above tables we see that peak performance for the GloVe cosine-similarity classifier is reached at approximately 30 principal components. For the FastText cosine-similarity classifier we reach peak performance at approximately 3 principal components. We also got a train accuracy for FastText at 97.3%.

		Test accuracy							
	K, principal components	2	3	4	5	10	20	30	40
FastText	PCA/Cosine similarity	39.00%	59.64%	58.96%	58.73%	51.24%	56.23%	53.06%	46.26%
	PCA/RandomForest	58.05%	62.13%	60.77%	62.59%	63.04%	57.14%	58.73%	57.37%
	Built-in predict (15 epochs)	62.13%							

FastText Train accuracy: 97.3%

Fig. 6. Confusion Matrix: FastText/Cosine similarity at $k = 3$ principal components. Confusion matrices for the other classifiers can be seen in Appendix



3.1.1. Custom Plot Description

We wrote the following custom plot description, which we classified as ”Sci-Fi”:

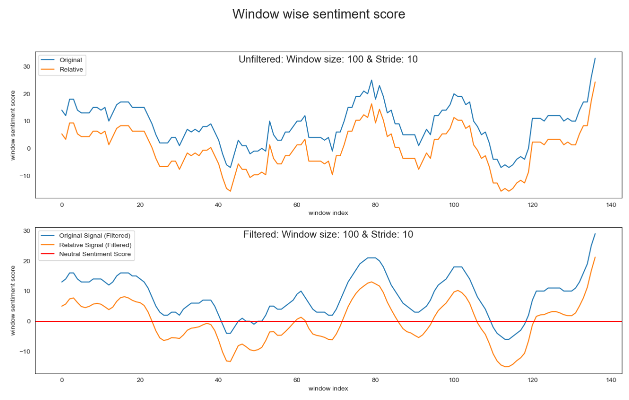
”When the Earth is invaded by alien technology a single human rises up to battle. His name is John Cena. He battles the invasion for weeks, using the most technologically advanced cybernetics. John Cena wins the battle against the aliens, but instantly a new enemy emerges through a time-travelling wormhole. Futuristic robots with laser-eyes. The robots end up killing John Cena, but before he died he planted a quantum bomb inside one of the robots which opens a giant black hole in the middle of Earth killing everything in the Milkyway.”

With our GloVe PCA with 30 principal components our model classified this as ”Romance”. Our FastText model classified this as ”Romance”.

3.2. Sentiment Analysis

On figure 7 you can see the results for the window wise sentiment scores with and without the Gaussian filter. On the figure we have plotted the original sentiment signal as well as the relative sentiment signal (subtracted the mean). It is clear that the sentiment score fluctuates between positive and negative. Furthermore it is clear that the speech starts with a positive sentiment score and ends with a crescendo in the sentiment score.

Fig. 7. The window wise sentiment score and the relative window wise sentiment score (without and with filtering)



4. DISCUSSION

Our test accuracies for the GloVe embeddings using the cosine similarity score as a classifier seems to perform the best, at 30 principal components. Here we explained around 80% of the variance. We cannot predict which features each principal component describes and therefore some may act as noise. This is probably the reason why we see a drop in accuracy when we use 40 principal components. With the Random Forest classifier we achieve accuracies above 60%. We see that 2 and 5 principal components do not tell us enough about the categories to make correct classifications. The reason Random Forest is more precise is probably just that it is a better model than just looking at the cosine distance, which will most likely be more prone to noise.

For the FastText embeddings which are trained with the categories in mind, we achieve a *good* prediction at 3 principal components. At two principal components we should be able to make predictions according to figure 4 where we see the categories clustered. The built-in FastText classifier achieves an accuracy of 62.13%. In our results we saw a *FastText train accuracy* of 97.3%, which in comparison to our relatively low test accuracy might indicate that our model is overfitting quite a lot. This is purely speculation, as we have not examined it in this project.

Our expectation for FastText was that the built-in prediction method would have higher accuracy than cosine-similarity. Furthermore, for this project we hypothesised that the FastText model would get higher accuracy at the cost of runtime. We have not analysed runtime, although the GloVe embeddings are already defined which in theory would make this approach faster with larger datasets, since we would not have to create the n-grams and word embeddings from the text. It seems that PCA on the GloVe embeddings and a decision tree classifier such as Random Forest would be optimal. Though, we cannot make a final conclusion based on only these tests. Cross validation with different random seeds could help make our results more reliable.

On our confusion matrices we see a tendency for both the FastText and GloVe that the ”Romance” category is the most frequently predicted. We can only speculate, but a reason for this could be that ”Romance” is a very broad genre and thus overlaps a lot with the other genres. Another explanation could be the fact that our dataset consists of approximately 40% texts in the ”Romance” category. While the other genres are represented less frequently. For FastText we see that there are no predictions in the Thriller genre, which seems to be a common theme across all our classifiers on these embeddings. This indicates that the embeddings do not capture features that are characteristic for the genre. It might also be because thriller is a vaguely defined genre. To sum up, we see that some movies are easier predicted than others, but we do not see the distribution in our confusion matrix as we would normally expect, but again, we only predict with an accuracy of around 50%-65%.

From our results we also see that with an increasing amount of principal components used for classification we get a lower test accuracy. We actually see that with 3 principal components our FastText cosine-similarity classification model peaks in accuracy. For our GloVe cosine-similarity classifier we need around 30 principal components for peak performance. We suspect that this might be because of *noise* in our data. This essentially means that with an increasing amount of principal components we include a correspondingly increasing amount of possibly miss-leading data. Some principal components might explain other things than the movie genres. We expect this for both FastText as well as GloVe. Therefore, we should not expect the best performance if we include many principal components.

On figure 4 we see that when we use FastText, and project our data onto the first and second principal component, our data is neatly clustered into labeled groups. As mentioned, the FastText model learns the word embeddings during training, which causes the the first two principal components to explain our labeled data very well.

On figure 2 we see that our data projected onto the GloVe PCA’s first and second principal components was clustered incoherently according to categories. We suspect this to be because we used pretrained embeddings. The word embeddings used were *trained* from Wikipedia pages, which would be expected to have a different semantic structure (higher Lix etc.), than IDMB plot descriptions.

Generally, the PCA of FastText and GloVe will not be comparable. This is because each PCA will be describing different aspects of our data. As mentioned, the FastText model is trained to be very good at classifying labels for our IMDB plot descriptions. Whereas, the PCA of our GloVe vectors will not necessarily explain our data corresponding to the labels. This means that the first two principal components might describe something completely different from what we

”want”. Furthermore, the GloVe vectors rely on us cleaning the data which makes the model more prone to error.

One problem, that we face with the GloVe embeddings, is that some words will not exist in the pretrained embeddings. This is mostly due to the fact, that we remove special characters and a word like ”not-so-popular” will be represented as ”notsopopular”. This is not a real word and thus, does not exist as a GloVe embedding. In our implementation we have chosen to skip this word and represented it as a 0-vector.

Even though we get a relatively disappointing test accuracy we see that our data is not evenly distributed since close to 40% of our data is categorised as ”Romance”. This might lead to our model being biased towards ”Romance”. We see that our models gets an accuracy well over 40%, which must mean that it has learned some traits that define the genres.

Another factor that makes our data imprecise is the fact that the labels for our documents might not represent 100% of the true label. A document might be mostly ”Sci-Fi” but also partially ”Romance”. If one looks at a movie on IMDB it is apparent that a movie actually does have several genre labels. Hence, instead of using a binary label we could in fact consider the labels as a distribution of ratios for each genre. Therefore, when working with text classifications we should not expect a very high classification ratio since the real world seldom acts in binary.

In the results of the sentiment analysis we uncovered a somewhat expected pattern where the speech starts and ends on a a high-note. It makes sense to structure a political speech in this way since you will make a good first impression and leave on a high note. In general the sentiment analysis would be more scientifically interesting if compared to other metrics such as outcome, ratings, reviews etc. An interesting comparison would have been between Trump’s speech and Obama’s Speech, since both presidents have been recognised as exemplary speakers, but by widely different social groups. In continuation of this it would have been interesting to compare how the sentiment score of a given speech elicits different reactions/results. The sentiment score lexicons used for sentiment analysis can be prone to subjectivity and bias. In this regard it could have been interesting to compare the results using different sentiment score lexicons. Another interesting implementation could have been adding an arousal score to compare with the sentiment score. Lastly, we could have compared our results with different window sizes. The window size changes the scope of the sentiment analysis where larger window sizes describe the global patterns and smaller windows sizes describe the local patterns.

5. CONCLUSION

In this project we analysed and classified text using *GloVe*, *FastText* and *sentiment analysis*. When analysing and classifying texts with GloVe and FastText, we firstly, used the pretrained GloVe word embeddings to create our GloVe model. Secondly, we implemented the FastText model. When comparing the two methods we found that FastText performs marginally better. Furthermore, we can conclude, from our model, that the FastText model needs less principal components to explain a sufficient amount of our train data. This corresponds somewhat to our hypothesis, that the FastText would be more precise at the cost of runtime.

Lastly, we conducted a sentiment analysis of Trump’s inauguration speech. We used a window size of 100 and a stride of 10 which we used to determine the window wise sentiment scores. From this sentiment analysis we see that the speech oscillates around a sentiment score of 5 until the middle where it peaks. After the peak the sentiment score dips before ending in a crescendo. These results correspond nicely to our hypothesis; that the speech would have a generally high sentiment score, especially towards the end.

6. REFERENCES

[1] Jeffrey Pennington, Richard Socher, and Christopher D Manning, “GloVe: Global Vectors for Word Representation,” Tech. Rep.

[2] “IMDbPY,” .

[3] Jiawei Han, Micheline Kamber, and Jian Pei, “2 - getting to know your data,” in *Data Mining (Third Edition)*, Jiawei Han, Micheline Kamber, and Jian Pei, Eds., The Morgan Kaufmann Series in Data Management Systems, pp. 39–82. Morgan Kaufmann, Boston, third edition edition, 2012.

[4] “sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation,” .

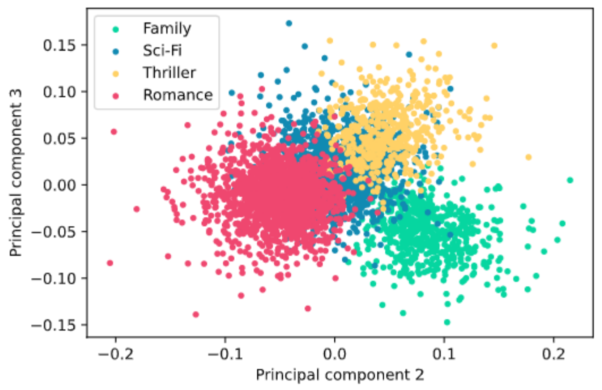
[5] “Text classification with the torchtext library — PyTorch Tutorials 1.8.1+cu102 documentation,” .

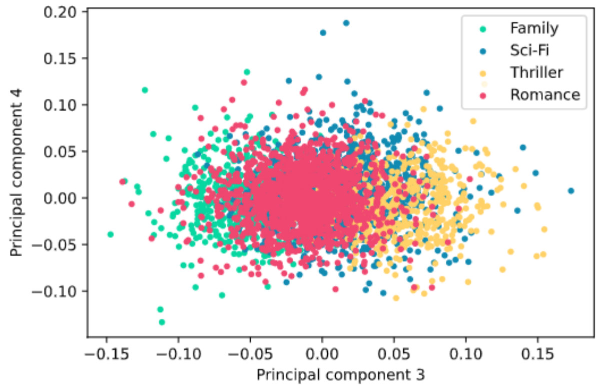
[6] F. Å. Nielsen, “AFINN,” .

[7] Jonathan Foldager and Hiba Nasar, “Genre classification and sentiment analysis,” .

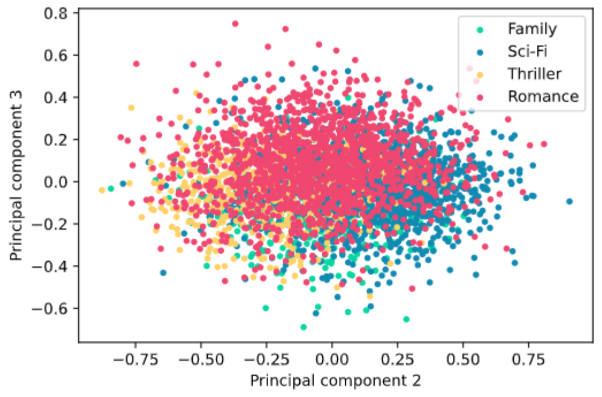
Appendix

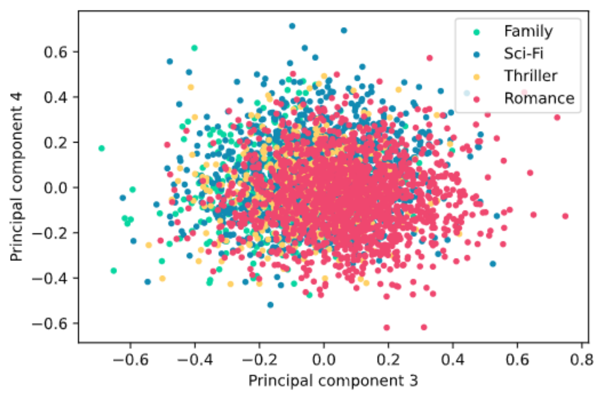
FastText Principal Component plots



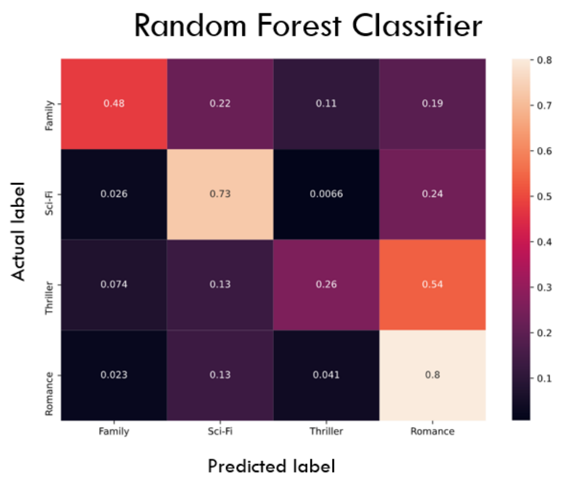


GloVe Principal Components plots



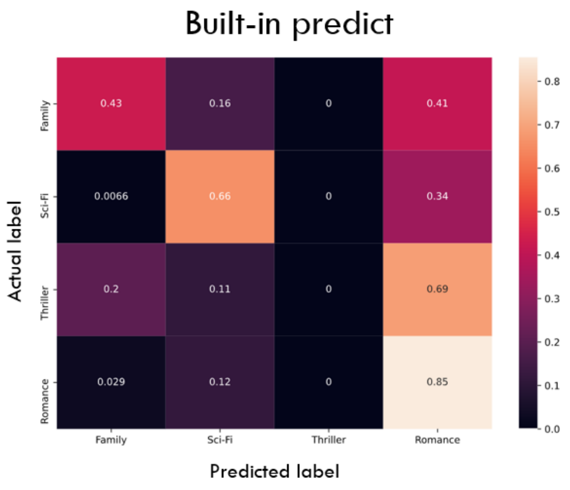


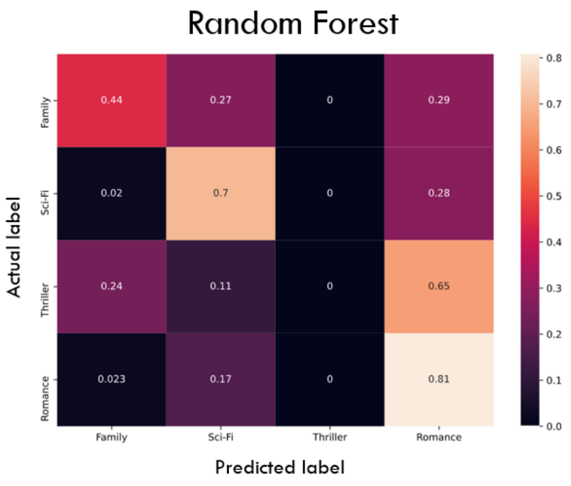
GloVe confusion matrix ($k = 30$)



	Family	Sci-Fi	Thriller	Romance
Family	0.48	0.22	0.11	0.19
Sci-Fi	0.026	0.73	0.0066	0.24
Thriller	0.074	0.13	0.26	0.54
Romance	0.023	0.13	0.041	0.8

FastText confusion matrix ($k = 3$)


Built-in predict
Actual label
Predicted label
Family
Sci-Fi
Thriller
Romance


Random Forest
Actual label
Predicted label
Family
Sci-Fi
Thriller
Romance