Fake News Detection using Natural Language Processing

Mohamed Ayman[†]
Computer Engineering
The American University in Cairo
mohamedayman15069@aucegypt.edu

Abdelrahman Shaaban
Computer Engineering
The American University in Cairo
abdelrahmanfawzy@aucegypt.edu

1. Introduction

In phase 2, we were challenged with the problem of storing a Bag of words for each word. In detail, the dictionary has nearly 2781 unique words, meaning that each sentence would be represented as numbers using a vector of 2781 x 1 having the frequency of each word. After preprocessing- deleting the Stop words, and doing stemming, and other ways of preprocessing- the Fakeddit dataset, we have nearly 546 thousand examples. This needs a near of 282 GB Memory to store 546,000 x 2781 matrix and do useful manipulations on it. After that, we tried to do the solution of storing the words as a map to avoid the problem of numerous zeros in each row matrix temporarily. In this phase, we have been thinking of using the Word2Vec and Doc2Vec embedding. The discernible reasons for using them are the small size of embedding for each word compared to the bag of words and retention of the semantic meaning of different words in the dataset. To see which one is more influential and surges the accuracy, we have created two embeddings using Doc2Vec, and Word2Vec respectively. After representing the words as representative, numerical embedded vectors, we have trained the dataset on different models explained in the class- Logistic regression, SVM, Naive Bayes, Decision Trees, Perceptron, and deep neural networks- with the result of each embedding technique. We did not use KNN, as it is not accurate with big, high dimensional dataset.

2. Reasons for dropping all columns except the title

Regarding the comments provided in phase 2, the author is from fake authors, and ID are generated from machines according to the collectors of the dataset itself. In this sense, we dropped them. Also, Regarding the other columns, there are some valid reasons to do so. Subreddit, and domain column are meaningless, as they do not have any additional information to identify the authenticity of the news. Upvote ratio, score, num comments are misleading in the datasets, because there are several fake examples having high scores and upvote ratio. Regarding linked submission id, we have found that all of them are NaN. We dropped the title column since clean title is the same as title other than the fact that clean title is cleaned from hashtags and URLs. Last two features the image and its URL are dropped since we do not deal with images for now.

WOODSTOCK'18, June, 2018, El Paso, Texas USA

https://doi.org/10.1145/1234567890

3. Data Representation

3.1 Word to Vector Embedding

With all the challenges and the disadvantages of the data representation using BOW and TF-IDF, such as the memory limitations and the lack of context' knowledge, Word2Vec embedding is one of the most efficient models for learning word embeddings. We used the CBOW model, as shown in figure 1, for training the embedding layer as it has a better generalization and faster performance compared to the skip-gram model. In the CBOW model, the objective is to predict the current word from a window of context words. For training the CBOW model, we used an embedding vector size of 500 with the default size of words' window =4. By training the model, we looped over the whole training and testing samples and applied the embedding on each word to get a training set of (400660, 500) and testing set of (133554, 500).

```
In [85]: modell.wv.most_similar("book")

Out[85]: [('newspap', 0.6760290265083313),
          ('copi', 0.6708730459213257),
          ('textbook', 0.6647543907165527),
          ('page', 0.6197590231895447),
          ('card', 0.6169708967208862),
          ('letter', 0.5677825212478638),
          ('text', 0.5625329613685608),
          ('album', 0.5575341582298279),
          ('calendar', 0.5557220578193665),
          ('error', 0.5418707728385925)]
```

Fig. 1: Screenshot to a piece of code

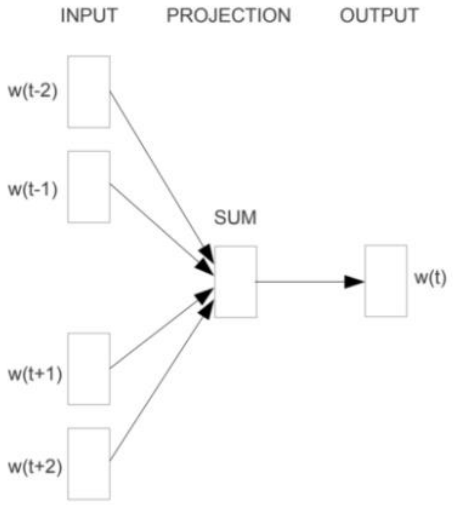

Fig. 2: Continuous Bag of words Model

Also, from figure 2, we can see an example to the most similar words to word "book". Besides the CBOW model, we worked on implementing the doc2vec model, training each of their resulting datasets on multiple classifiers and making comparisons. These classifiers are logistic regression, perceptron, neural network, naive bayes, decision tree, and SVM. By running each of these models, we used the sklearn classification report to observe the accuracy, precision, and recall of each of them.

3.2 Document to Vector Embedding

Doc2Vec is simply a generalized form of the word2vec method, representing each document in the corpus as a vector irrespective of each document's length. Doc2Vec model is based on Word2Vec only with adding another vector called paragraph ID to the input. By analogy, the Doc2Vec model can rely on one of the following architectures: distributed Memory version of Paragraph Vector (PV-DM) and Distributed Bag of Words version of Paragraph Vector (PV-DBOW). The figure 4 illustrates the architecture of PV-DM. It is based on a continuous bag of words with a little modification. Instead of using nearby words to predict the word, it holds a numeric representation of the whole sentence- document. For further explanation, the input has the word vectors as a hot encoding and the document ID vector. Moreover, we use Gensim and SKlearn libraries to implement the Doc2Vec technique. More clearly, we have dimensioned the number of features in the embedding to be 300 for each document.

```
In [50]: def get_vectors(m, corp_sz, v_sz, v_type):
             vectors= np.zeros((corp_sz,v_sz))
             for i in range(0, corp_sz):
                 lb1= v_type + '_' + str(i)
                 vectors[i]=m.docvecs[lb1]
             return vectors

In [51]: train_vecs=get_vectors(model, len(X_train), 300, 'Train')
         test_vecs=get_vectors(model, len(X_test), 300, 'Test')

         C:\Users\ABDELR~1\AppData\Local\Temp/ipykernel_25184/34870
         ecs` (The `docvecs` property has been renamed `dv`.).
           vectors[i]=m.docvecs[lb1]

In [52]: train_vecs.shape

Out[52]: (400660, 300)
```

Fig. 3: Screenshot to a piece of code

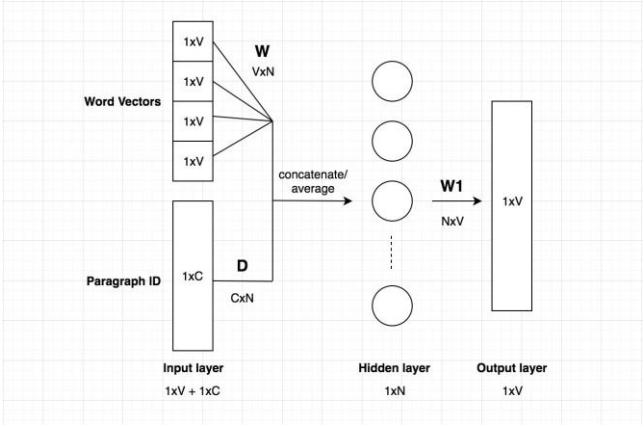

Fig. 4: Distributed Bag of Words Model

4. Experimental analysis for each Machine learning model with comments on each of model's performance

After creating the embedding vectors, we use the numerical representation of sentences to be considered as an input for each model, calculate the accuracy, precision, and recall and comment on each model's results. For each model, we have done several variations for each hyperparameter to be able to get the highest accuracy. For each model, we mention the best hyperparameters we get with respect to the accuracy, precision, recall, and F1 score. Most importantly, we need to identify whether precision or recall is more crucial. In our problem, it is better for the user to get notified that the news is fake although it is not (FP) rather than get notified that the news is real although it is fake (FN). There is a huge trade-off here, but we need to decrease the number of FN. In this sense, recall is more important in terms of our context. However, we will try to maximize F1_Score, as precision is essential as well. Moreover, luckily, the performance evaluation of each model shows all models converged to a solution, meaning that the cost function was minimized at each model accuracy; therefore, we can conclude that increasing the number of iterations will not improve the accuracy much.

4.1 Logistic Regression

	Word2vec CBOW	doc2vec
Accuracy	0.7	0.68
Precision	0.7	0.68
Recall	0.7	0.68
F1_Score	0.7	0.68

Table 1: Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score.

Logistic Regression is the basic binary classifier to be used. We used the SKlearn logistic regression model with all its default parameters except for the C=1e-5 which is determined to control the model's generalization. By observing the results, it achieves on the two kinds of the embedded datasets (as shown in table 1), it showed how effective it is for our problem using the sigmoid function to compute the probability of each class. The table below shows accuracy, precision, recall, and f1 score for the logistic regression on each embedding output.

Logistic Regression

```
In [53]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import classification_report, accurac
         clf=LogisticRegression(C=1e5)
         clf=clf.fit(train_vecs,y_train)
         yp=clf.predict(test_vecs)
         yp

Out[53]: array([0, 1, 1, ..., 0, 0, 1], dtype=int64)

In [54]: print(classification_report(y_test,yp))

                    precision    recall  f1-score   support

                 0       0.68      0.74      0.71     70538
                 1       0.68      0.62      0.65     63016

          accuracy                           0.68    133554
         macro avg       0.68      0.68      0.68    133554
      weighted avg       0.68      0.68      0.68    133554
```

Fig. 5: Screenshot to the Classification report for Logistic regression model using Doc2Vec.

```
In [86]: test_predictions_w2vec = clf.predict(w2f_test_df.values)
         print(classification_report(y_test.values,test_predictions_w2vec))

                    precision    recall  f1-score   support

                 0       0.71      0.71      0.71     70421
                 1       0.68      0.68      0.68     63133

          accuracy                           0.70    133554
         macro avg       0.69      0.70      0.69    133554
      weighted avg       0.70      0.70      0.70    133554
```

Fig. 6: Screenshot to the Classification report for Logistic regression model using Word2Vec.

Now. we can notice that from the performance evaluation table and screenshots above, Word2Vec embedding gives a higher accuracy than doc2vec embedding. Also, the table shows how accuracy does not hinder the results of precision and recall. Also, all performance metrics- recall, precision, and accuracy- for each class in word2vec is better than the Doc2Vec. As mentioned above, the logistic regression model gives a solution since it could minimize the cost function with gradient descent. One of the advantages in logistic regression is very simple in terms of complexity, and so it takes less time. Not only this, but it is one of the highest accurate models for detecting fake news.

4.2 Naïve Bayes

Naive Bayes is another model widely used for text classification using probabilistic estimates. In this project, we used two naive bayes models which are the Gaussian NB and the Bernoulli NB with their default parameters. In Gaussian NB, the model follows Gaussian normal distribution and supports continuous data. However, Bernoulli is used for classifying discrete data, such as binary values and it works using Bernoulli distribution. Based on the following results, Naive Bayes is not the best choice for the classification problem of this project although it shows some good potential. Multinomial NB can't be used due to the negative values resulting from the words' embedding.

Gaussian NB:

	Word2vec CBOW	doc2vec
Accuracy	0.64	0.62
Precision	0.66	0.62
Recall	0.64	0.62
F1_Score	0.63	0.62

Table 2: Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score

Bernoulli NB:

	Word2vec CBOW	doc2vec
Accuracy	0.64	0.66
Precision	0.66	0.66
Recall	0.64	0.66
F1_Score	0.63	0.66

Table 3: Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score

From the screenshots and tables above, we can notice that in Gaussian NB, Word2Vec is better than Doc2vec in terms of accuracy, precision, recall, and F1_Score. Meanwhile, Bernoulli NB accuracy, recall, and F1_Score are much better when using Doc2Vec technique for embedding rather than Word2Vec CBOW, which is the reverse truth in Gaussian NB and Logistic Regression. The screenshots show that the model is not biased to a specific class, and the cost function is minimized somehow to give good

results. The table shows that using Gaussian NB model is better than using the other one in terms of all types of performance metrics. Furthermore, there is no huge difference between evaluating the recall, precision, and F1_Score macro average or micro average. One of disadvantage to the model is its claim that the features are completely independent- although the features are related to each other in our problem. In this sense, the accuracy is not that good compared to logistic regression or neural network.

```
GAUSSIAN NB
              precision    recall  f1-score   support

           0       0.63      0.72      0.67     70538
           1       0.62      0.52      0.57     63016

    accuracy                           0.62    133554
   macro avg       0.62      0.62      0.62    133554
weighted avg       0.62      0.62      0.62    133554

BERNOULLI NB
              precision    recall  f1-score   support

           0       0.68      0.65      0.67     70538
           1       0.63      0.66      0.65     63016

    accuracy                           0.66    133554
   macro avg       0.66      0.66      0.66    133554
weighted avg       0.66      0.66      0.66    133554
```

Fig. 7: Screenshot to the Classification report Gaussian and Bernoulli NB models using Doc2Vec.

```
print(classification_report(y_test.values,test_ber_word2vec))
              precision    recall  f1-score   support

           0       0.72      0.53      0.61     70421
           1       0.59      0.77      0.67     63133

    accuracy                           0.64    133554
   macro avg       0.65      0.65      0.64    133554
weighted avg       0.66      0.64      0.64    133554

              precision    recall  f1-score   support

           0       0.72      0.53      0.61     70421
           1       0.59      0.77      0.67     63133

    accuracy                           0.64    133554
   macro avg       0.65      0.65      0.64    133554
weighted avg       0.66      0.64      0.64    133554
```

Fig. 8: Screenshot to the Classification report Gaussian and Bernoulli NB models using Word2Vec.

4.3 Decision Trees:

Decision Trees' model is one of the easiest and most popular classification models to understand and interpret. It is made to learn the complicated rules and patterns and to scale the large data sets. It fits the project's problem with the large number of features it has. First, we used sklearn's Decision tree with all its default parameters' values.

	Word2vec CBOW	doc2vec
Accuracy	0.66	0.60
Precision	0.66	0.60
Recall	0.66	0.60
F1_Score	0.66	0.60

Table 4: Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score

Then, we have tried to change the hyperparameters to boost the accuracy somehow, like limiting the depth of the decision tree = 10, 100. However, the default values give much more adaptive depth to get better results. Regarding the performance metrics, we can find from table 4 that Word2Vector representation gives better accuracy than using doc2vec. Going more depth with commenting on each performance. First, Precision and recall gives good results, especially with two classes as shown in figure 9 and 10. In other words, the learned model is not biased to specific class, which a huge advantage to it, like previous models. One of the disadvantages we can notice from the performance metric is the low accuracy. It means that decision trees cannot achieve interpreting the label class right most of the time. Not only this, but decision tree model is very complex, and takes much time for training (constructing the tree).

```
In [59]: from sklearn.tree import DecisionTreeClassifier
         clf_decision_doc2vec = DecisionTreeClassifier()
         clf_decision_doc2vec.fit(train_vecs,y_train)
         test_predictions_doc2vec = clf_decision_doc2vec.predict(test_vecs)
         print(classification_report(y_test,test_predictions_doc2vec))
              precision    recall  f1-score   support

           0       0.61      0.61      0.61     70538
           1       0.57      0.57      0.57     63016

    accuracy                           0.59    133554
   macro avg       0.59      0.59      0.59    133554
weighted avg       0.59      0.59      0.59    133554
```

Fig. 9: Screenshot to the Classification report to decision tree's classification report using the Doc2Vec technique.

```
In [87]: from sklearn.tree import DecisionTreeClassifier
         clf_decision_word2vec = DecisionTreeClassifier()
         clf_decision_word2vec.fit(w2f_df.values,y_train.values)
         test_decision_word2vec = clf_decision_word2vec.predict(w2f_test_df.values)
         print(classification_report(y_test.values,test_decision_word2vec))
              precision    recall  f1-score   support

           0       0.67      0.71      0.69     70421
           1       0.65      0.61      0.63     63133

    accuracy                           0.66    133554
   macro avg       0.66      0.66      0.66    133554
weighted avg       0.66      0.66      0.66    133554
```

Fig. 10: Screenshot to the Classification report to decision tree's classification report using the Word2Vec technique.

4.4 Support Vector Machine:

	Word2vec CBOW	doc2vec
Accuracy	0.69	0.68
Precision	0.69	0.68
Recall	0.69	0.68
F1_Score	0.69	0.68

Table 5: Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score

Support Vector Machines' models are the most powerful classical machine learning models with the ability to scale to larger data sets, but they are also more complicated, and it can be difficult to know what patterns they rely on. It strongly fits the classification problem as how powerful it is is what this problem with the large number of features needs. By using the sklearn svc model from svm with a linear kernel, it took so much time to run such a model with the large number of features we have and the limited computational resources, even with the help of the PCA for reducing the dimensions of the data. That's why we had to implement this model using the SGDclassifier linear model from sklearn. We choose our limit of 1000 iterations to make the model converge

```
In [55]: from sklearn.linear_model import SGDClassifier
         clf=SGDClassifier(max_iter=100000)
         clf= clf.fit(train_vecs,y_train)
         yp=clf.predict(test_vecs)
         print(classification_report(y_test,yp))
```

```
              precision    recall  f1-score   support

           0       0.69      0.72      0.70     70538
           1       0.67      0.64      0.66     63016

    accuracy                           0.68    133554
   macro avg       0.68      0.68      0.68    133554
weighted avg       0.68      0.68      0.68    133554
```

.

```
In [91]: from sklearn import linear_model
         clf = linear_model.SGDClassifier()
         clf.fit(w2f_df,y_train)
         test_predictions_w2vec = clf.predict(w2f_test_df)
         print(classification_report(y_test.values,test_predictions_w2vec))
```

```
              precision    recall  f1-score   support

           0       0.71      0.70      0.70     70421
           1       0.67      0.68      0.67     63133

    accuracy                           0.69    133554
   macro avg       0.69      0.69      0.69    133554
weighted avg       0.69      0.69      0.69    133554
```

Fig. 12: Screenshot to the Classification report to SVM classification report using the Doc2Vec technique.

Regarding performance metric, from figure 11 and 12, we can see that word2vec is better than the Doc2Vec in terms of all performance metrics as previously proved from the results of other Machine learning model techniques. We can see that SVM accuracy is near to the Logistic regression accuracy. Stochastic gradient descent optimization proves how it is suitable for big datasets. Not only this, but the figures and table above illustrate how the precision and recall for each class are somehow near, meaning that the model is not biased and could result in convergence. Also, it has a great advantage of training compared to the RBF SVM. The performance metric results illustrate how the model is good for checking fake news, but it needs some modification to improve the accuracy somehow. Furthermore, compared to the logistic regression, we can notice that logistic regression is much simpler and more accurate than SVM.

4.5 Neural Networks:

As taken in the class, we tended to use simple deep learning models to see how effective it is in our problem.

4.5.1 multi-Layers:

This time we tended to get multilayer perceptron classifiers to be used for this problem with the activation function of logistic as it is a binary classification problem and maximum activation limit of 5000. It is the most powerful model used in this paper and it is very fitting to the number of features as the hidden layers it contains that require more weights to be learnt. We could use the sklearn MLP with the default parameters of the number of hidden layers and learning rate.

	Word2vec CBOW	doc2vec
Accuracy	0.73	0.70
Precision	0.73	0.71
Recall	0.73	0.70
F1_Score	0.73	0.70

Table 6: **Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score**

```
In [89]: from sklearn.neural_network import MLPClassifier
         NN = MLPClassifier(random_state=1, max_iter=5000, activation='logistic')
         NN.fit(w2f_df.values,y_train.values)
         test_NN_word2vec = NN.predict(w2f_test_df.values)
         print(classification_report(y_test.values,test_NN_word2vec))
```

```
              precision    recall  f1-score   support

           0       0.74      0.75      0.74     70421
           1       0.72      0.70      0.71     63133

    accuracy                           0.73    133554
   macro avg       0.73      0.73      0.73    133554
weighted avg       0.73      0.73      0.73    133554
```

Fig. 13: Screenshot to the Classification report to MLP classification report using the Word2Vec technique.

From the screenshots above, we can notice that using word2vec as representation of dataset is better than Doc2Vec. Regarding the accuracy, recall, and precision, we can notice that how balanced they are in each class and quite high. One of disadvantage of the neural network is taking some time to train.

```
In [58]: from sklearn.neural_network import MLPClassifier
         NN = MLPClassifier(random_state=1, max_iter=5000)
         NN.fit(train_vecs,y_train.values)
         test_NN_doc2vec = NN.predict(test_vecs)
         print(classification_report(y_test.values,test_NN_doc2vec))
```

	precision	recall	f1-score	support
0	0.70	0.77	0.73	70538
1	0.71	0.64	0.67	63016
accuracy			0.71	133554
macro avg	0.71	0.70	0.70	133554
weighted avg	0.71	0.71	0.71	133554

Fig. 14: Screenshot to the Classification report to MLP classification report using the Doc2Vec technique.

4.5.2 Perceptron:

Perceptron is a basic model to make the NN model and it can be used as a binary classifier model. It is a very weak model, underfitting the big data we have with its large number of features. That's why it is clear below that it suffers to efficiently make a good classification.

	Word2vec CBOW	doc2vec
Accuracy	0.61	0.57
Precision	0.61	0.63
Recall	0.61	0.55
F1_Score	0.61	0.48

Table 7: Comparison between Word2Vec CBOW and Doc2Vec in terms of accuracy, precision, Recall, and F1_Score

```
In [88]: from sklearn.linear_model import Perceptron
         Per_clf = Perceptron(tol=1e-3, random_state=0)
         Per_clf.fit(w2f_df.values,y_train.values)
         test_Per_word2vec=Per_clf.predict(w2f_test_df.values)
         print(classification_report(y_test.values,test_Per_word2vec))
```

	precision	recall	f1-score	support
0	0.63	0.64	0.63	70421
1	0.59	0.57	0.58	63133
accuracy			0.61	133554
macro avg	0.61	0.61	0.61	133554
weighted avg	0.61	0.61	0.61	133554

Fig. 15: Screenshot to the Classification report to Perceptron classification report using the word2Vec technique.

From the screenshots in figure 15 and 16, the recall of word2vec is biased. Not only this, but it is the worst performance results compared to the other models. This makes a little sense, as the data is not necessarily needed to be classified with a line.

```
In [57]: from sklearn.linear_model import Perceptron
         Per_clf = Perceptron(tol=1e-3, random_state=0)
         Per_clf.fit(train_vecs,y_train)
         test_Per_doc2vec=Per_clf.predict(test_vecs)
         print(classification_report(y_test.values,test_Per_doc2vec))
```

	precision	recall	f1-score	support
0	0.56	0.94	0.70	70538
1	0.70	0.17	0.27	63016
accuracy			0.57	133554
macro avg	0.63	0.55	0.48	133554
weighted avg	0.63	0.57	0.50	133554

Fig. 16: Screenshot to the Classification report to Perceptron classification report using the Doc2Vec technique.

5. Comparative analysis

From the tables above that show the results of training multiple classifiers on two different embedding outputs, we can observe that using word2vec embedding can lead to better results than using doc2vec. The opposite was expected but one reason why that happens can be that news' titles differ a lot from each other, with the relations between the nearby words can indicate whether a piece of news is fake or not more than considering their relationship with the whole title. In addition, when it comes to the best performing three models, they are as following:

1- Multilayer Perceptron Classifiers (MLP)

2- Logistic Regression

3- Linear Support Vector Machine

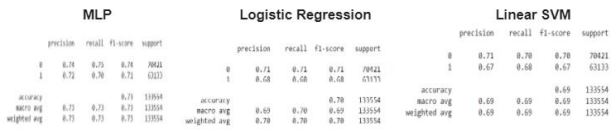

Fig. 17: Screenshot to the Classification report to the best three models.

As shown in the three tables above for the best three classifiers with the word2vec embedding, the difference between their performance when it comes to the accuracy, precision, and recall is not large. MLP could achieve an accuracy of 0.73 (Highest) compared to the logistic regression classifier with an accuracy of 0.7 and the linear SVM with an accuracy of 0.69. That was expected as MLP is a very complex model consisting of multiple layers with many weights to learn, which gives it a great ability to detect the data patterns easily compared to the other models which are not at the same level of complexity. However, achieving 0.7 using a very simple model such as logistic regression made us doubt the efficiency of the MLP and the possibility of overfitting

that led to such a small difference. The same thing applies for the linear SVM as it is also considered as a complex model that is predicted to achieve a higher performance. However, Overfitting might have happened.

6. Conclusion (The best Model)

To sum up, after experimenting two different embedding approaches with multiple classifiers and observing the results, we found that word2vec embedding performs better than doc2vec in while experimenting it with most of the classifiers. That indicates how the words' relationship plays a vital role in the classification process more than the relationship between the words and the title. Regarding the chosen model, we decided to stick to the model with the best performance which is the MLP classifier. It is the fittest for such a kind of classification problems with its high complexity that is essential for getting the many text features' patterns.It is basically a deep learning model that contains an input layer to get the text embedding, multiple hidden layers to detect its pattern, and a sigmoid output layer for the classification. The training process includes optimizing the layers' weights throughout forward and backward propagation.