IMDB report

Aly Swidan(3433), Mohamed Mahmoud(3431), Zeina elhayah Nada (3490)

April 2018

1 Introduction

Feature extraction turned out to be the deciding step in the results of sentiment analysis, as a result of this fact our work was mainly focused on trying different feature extraction techniques, specifically we tried three different methods to represent a document: TF-IDF sparse vectors, doc2vec and word vectors weighted with TF-IDFs.In the following sections we discuss how we went about using each of these methods, following this we would discuss our usage of different classifiers.

2 Preprocessing

2.1 Tokenization

We used some regular expressions to remove punctuation followed by a simple call to split to split the words into tokens.

2.2 Lemmatiztion

We used the nltk wordnet lemmatizer since after some researching we came up to the conclusion that lemmatization results in better accuracies than stemming since it considers the word's context insteda of merely stripping of the suffix as stemming does.

2.3 Removing Stop Words

We used nltk's stop words list to remove stop words from the corpus.

2.4 What we have so far

After doing the above cleaning steps, we got the following word clouds for positive and negative documents presented in figure 2 and figure 1 respectively, from inspecting those figures we noticed that some of the there are words that appeared very frequently in both classes that they contributed no meaning to the classifier, so we tried removing the 50 most frequent words in both classes after taking the intersection of the frequency dictionaries of both classes, the resulting word clouds are shown in figure 4 and figure 3, as could be seen, much of the noise has been removed, and the type of words in both classes started to give some indication of the sentiment, increasing the number of words we removed increased made things clearer, however it also decreased the size of the corpus drastically which had negative effects on accuracy, which is discussed later when we speak about how we classified the data.

2.5 Part of Speech Tagging

As another preprocessing step, we appended the part of speech tags to their corresponding words and treated the resulting compound words as our vocabulary, we only tried this approach with the word2vec model.

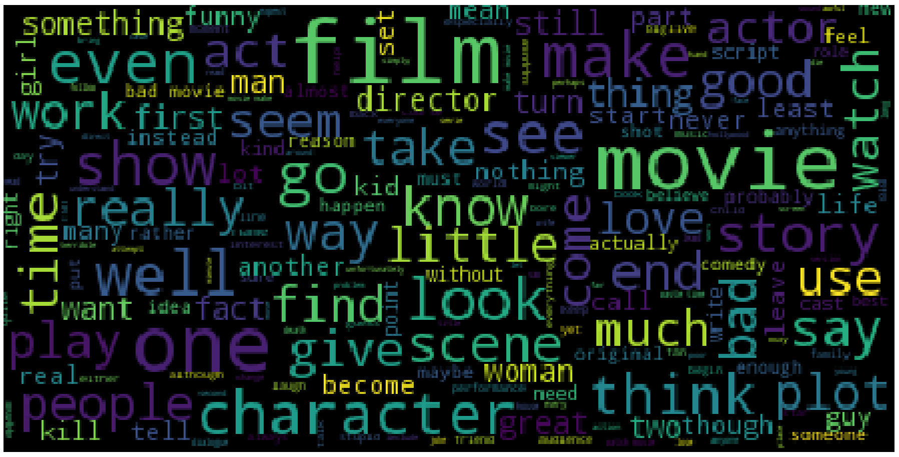


Figure 1: A word cloud of the words in negative docs just after cleaning

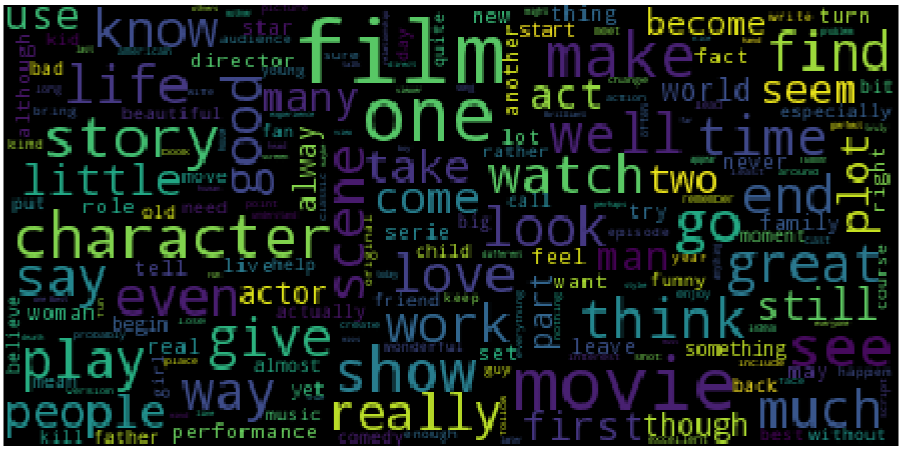


Figure 2: A word cloud of the words in positive docs just after cleaning

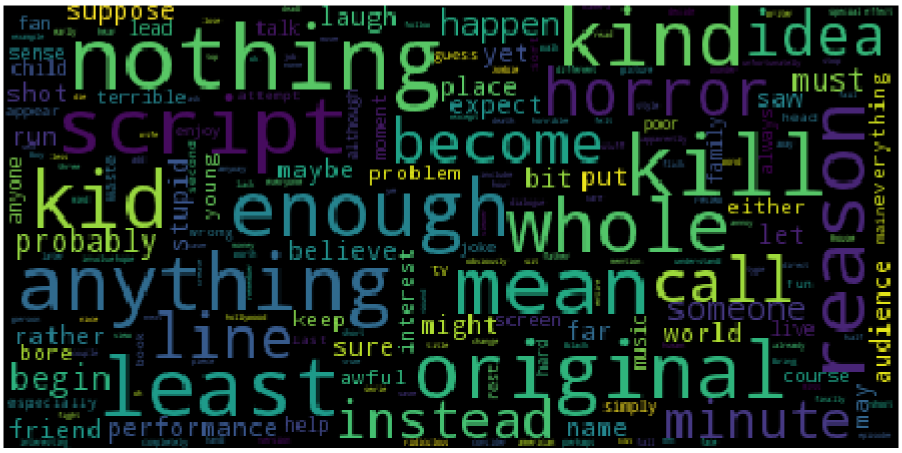


Figure 3: A word cloud of the words in positive docs after removing noisy words

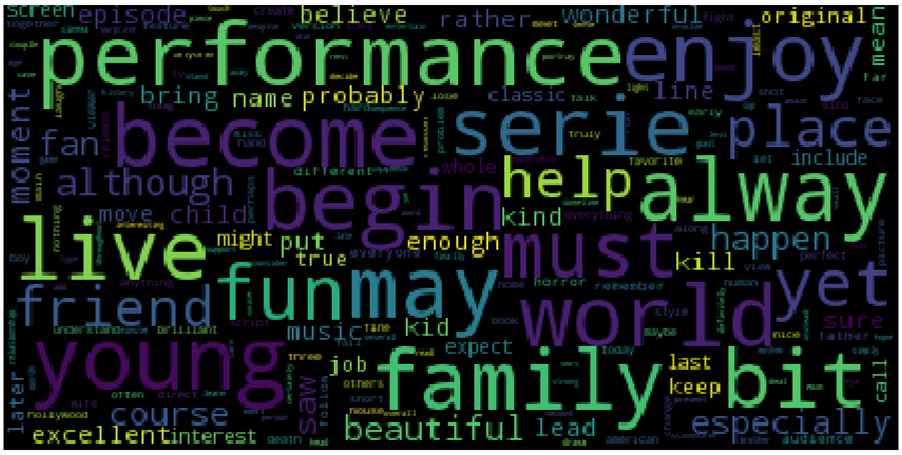


Figure 4: A word cloud of the words in positive docs just after removing noisy words

3 Feature extraction

3.1 TF-IDF sparse vectors

3.1.1 Overview

It's firstly built on tokenization, text preprocessing and filtering of stopwords with the use of countvectorizer function that is able to build a dictionary of features and transform documents to feature vectors. The index value of a word in the vocabulary is linked to its frequency in the whole training corpus. We then transform the occurrences to frequencies. Occurrence count is a good start but there is an issue: longer documents will have higher average count values than shorter documents, even though they might talk about the same topics. To avoid these potential discrepancies it suffices to divide the number of occurrences of each word in a document by the total number of words in the document: these new features are called tf for Term Frequencies. Another refinement on top of tf is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus. This downscaling is called tf–idf for "Term Frequency times Inverse Document Frequency". and here we used the skikit-learn function called "TfidfTransformer".

3.2 Word vectors weighted with TF-IDFs

3.2.1 Overview

This representation mainly relies on building word vectors for each word in the corpus-a word vector is nothing but a way to represent a word as a point in some high dimensional space, those vectors are then used to convert each document into a matrix of constant length vectors, since this matrix has a different length for each document we couldn't just stretch the matrix into one long vector and use it as the input to the classifiers, so after some researching we found out that the best results in literature was obtained with a weighted average of the rows of the matrix using the TF-IDF (described in the previous section) of the word represented by the vector as the weight.

3.2.2 Word2Vec

The exact method we used to produce the word vectors was the word2vec model[1], to be specific we used the implementation provided by the gensim library.This model has 2 variants the continuous bag of words or CBOW for short and the skip-gram model, at the core of those two lies a 2 layer neural network that receives as input a binary vector with a one in the position of the word or words we want to input. As for the output of the network, this is what differentiates the two variants, as for CBOW it takes as input a window of words and learns a function that predicts the next word following the window-the output is a vector of probabilities that has the length of the vocabulary.The skip-gram model differs from the other variant in that it tries to predict the probability that a word in the vocabulary appears in the window centered at the input word to the network.The network also uses a softmax layer for scoring.After the network is trained the hidden layer wights are considered our word vectors, Figure 5 shows a subset of the word vectors projected into 2 dimensions using

PCA, the proximity of points should indicate how close they are in meaning for example bananas and apples should have very close vectors- the network extracts this from the context of the words in the corpus, however due to the relatively small size of our corpus the vectors aren't very accurate.

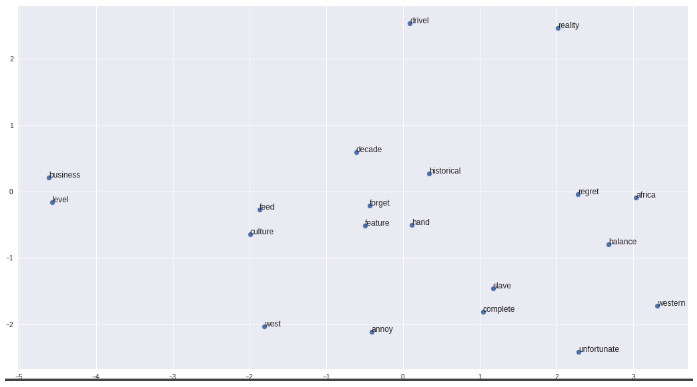


Figure 5: A plot of some word vectors

3.3 Doc2vec

3.3.1 Overview

Doc2Vec is to create a numeric representation of a document, regardless of it's length. But unlike words, documents do not come in logical structures such as words. The concept used is simple,use word2vec model, and added another feature vector (document-unique). So, when training the word vectors W, the document vector D is trained as well, and in the end of training, it holds a numeric representation of the document.

3.3.2 Combined Doc2Vec

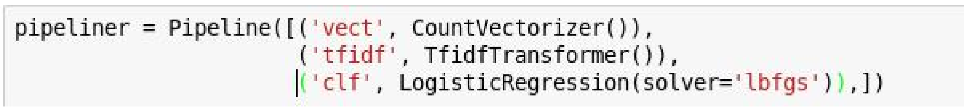
There are several algorithm used in doc2vec, one is Distributed Memory version of Paragraph Vector (PV-DM). It acts as a memory that remembers topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document. Another algorithm which is similar to skip-gram may be used, Distributed Bag of Words version of Paragraph Vector (PV-DBOW). The authers of an article using Doc2Vec recommend using a combination of both algorithms.

4 Classifiers Choice

4.1 TF-IDF Model

4.1.1 Overview

Now that we have our features, we can train a classifier to try to predict the positivity of a review. for all the previous three steps we used pipeline. In order to make the vectorizer transformer classifier easier to work with, scikit-learn provides a Pipeline class that behaves like a compound classifier so we used it as a one shot excutioner. Below you will find an example of how to implement a piepline class.



```
pipeliner = Pipeline([('vect', CountVectorizer()),
                      ('tfidf', TfidfTransformer()),
                      ('clf', LogisticRegression(solver='lbfgs')),])
```

Figure 6: A pipeline example

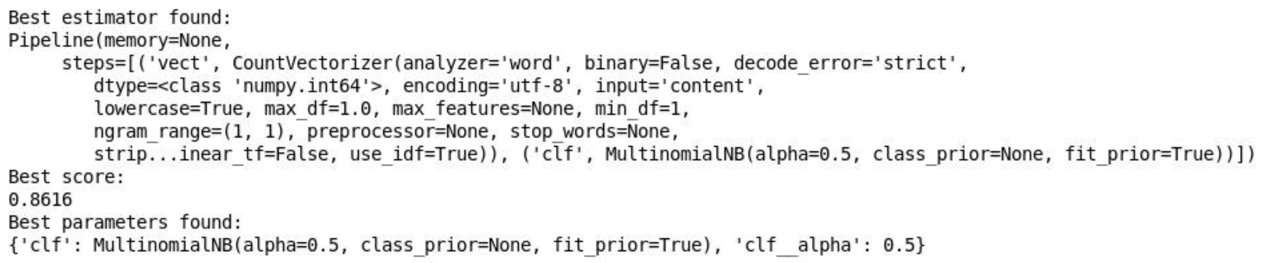
4.1.2 Hyperparameter Tuning and crossvalidating the training data

Hyper-parameters are parameters that are not directly learnt within estimators. We will use a great technique to get the best estimator with the best parameters which is also in the scikit-learn library. A search consists of:

- an estimator (regressor or classifier such as sklearn.svm.SVC())
- a parameter space
- a method for searching or sampling candidates
- a cross-validation scheme and
- a score function.

4.1.3 Our group of classifiers:

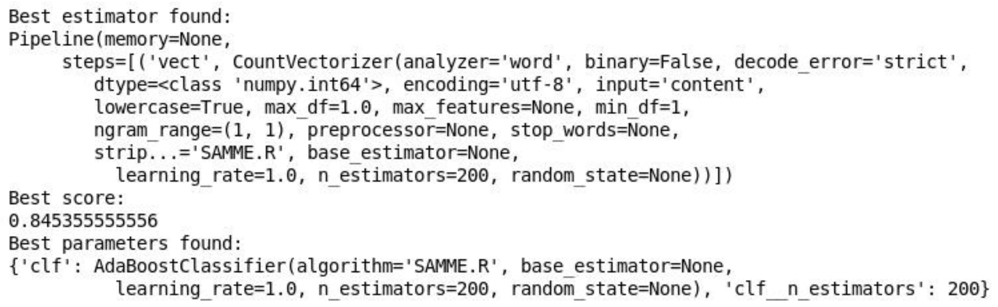
- Multinomial Naive Bayes Classifier



```
Best estimator found:
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...inear_tf=False, use_idf=True)), ('clf', MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True))])
Best score:
0.8616
Best parameters found:
{'clf': MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True), 'clf__alpha': 0.5}
```

Figure 7: GridSearch Result for MultiNB with alpha=0.5,1

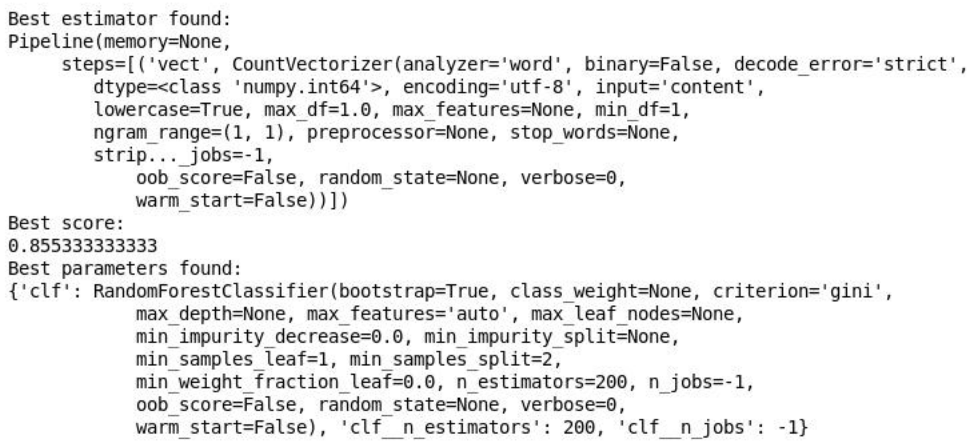
- Adaboost



```
Best estimator found:
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=200, random_state=None))])
Best score:
0.845355555556
Best parameters found:
{'clf': AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=200, random_state=None), 'clf__n_estimators': 200}
```

Figure 8: GridSearch Result for Adaboost with n_estimator=50,100,200

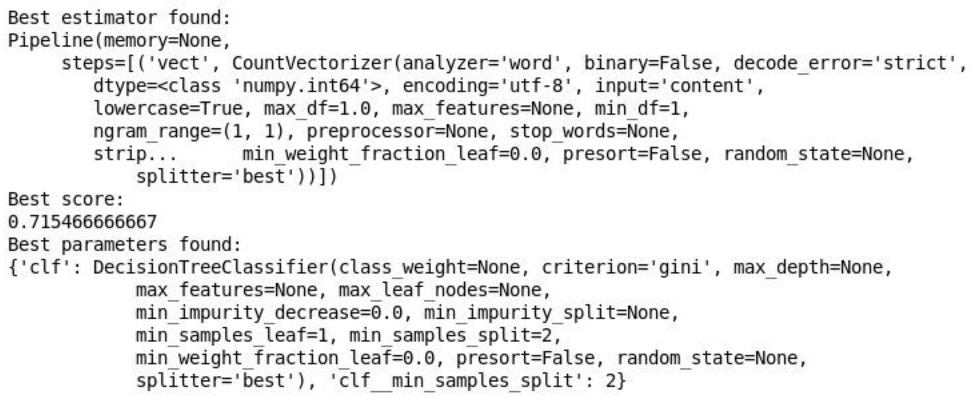
- Random forest



```
Best estimator found:
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip..._jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False))])
Best score:
0.855333333333
Best parameters found:
{'clf': RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False), 'clf__n_estimators': 200, 'clf__n_jobs': -1}
```

Figure 9: GridSearch Result for Random Forest with n_estimator=50,100,200

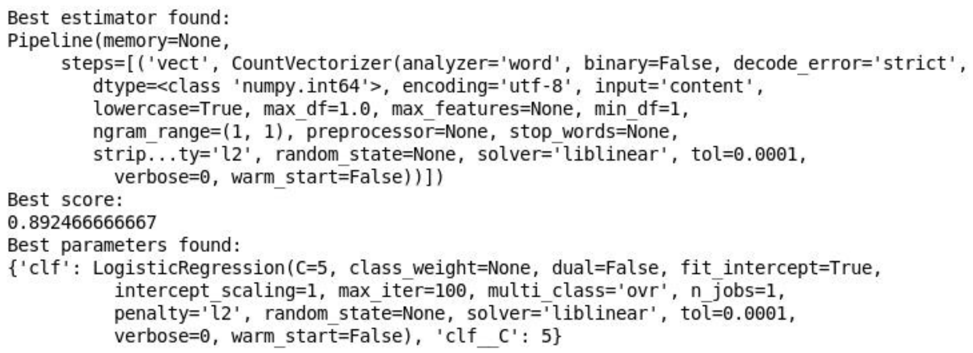
- Decision Tree Classifier



```
Best estimator found:
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...      min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'))])
Best score:
0.715466666667
Best parameters found:
{'clf': DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'), 'clf__min_samples_split': 2}
```

Figure 10: GridSearch Result for MultiNB with min_sample_split=2,10,20

- Logistic Regression Classifier



```
Best estimator found:
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...ty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False))])
Best score:
0.892466666667
Best parameters found:
{'clf': LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False), 'clf__C': 5}
```

Figure 11: GridSearch Result for Logistic Regression with C=0.1,5,10,15,20

4.2 Word2Vec Model

4.2.1 Overview

As previously stated, we have tried three different preprocessing techniques with this feature extraction method, namely: just basic cleaning, appending POS tags and removing words appearing with high frequency in both positive and negative classes.We tuned six classifiers on each of the 3 preprocessed data sets, we used 10-fold cross validation to search with Logistic Regression and only used one fold of size 10% with the rest of the classifiers due to their demand for resources which we couldn't afford.In what follows we briefly discuss the accuracies we got with each method.

4.2.2 Basic Cleaning

Figure 12 shows that the best accuracy achieved on the validation set is just above 77%, this accuracy however dropped by one percent to 76.6 on the test data.This was the best performing classifier, the rest of the classifiers and their validation set performance is in the accompanying jupyter notebook.

4.2.3 Part of speech tagging

The SVM with a RBF kernel achieved the best accuracy when using this method as well which was also just above 77% as shown in Figure 13 which isn't much of an improvement over the basic cleaning method.

4.2.4 Removal of noisy words

This was our last try to improve the accuracy of this method, however we didn't seem to get any improvement either as the accuracy dropped by 0.5% with the SVM as Figure **??** shows.

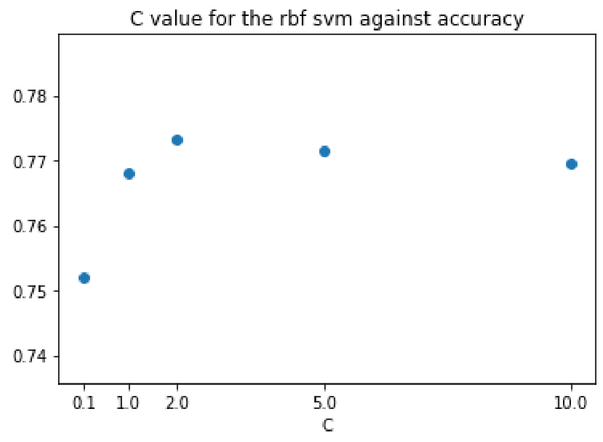


Figure 12: Performance of svm with RBF kernel on the validation set with basic cleaning

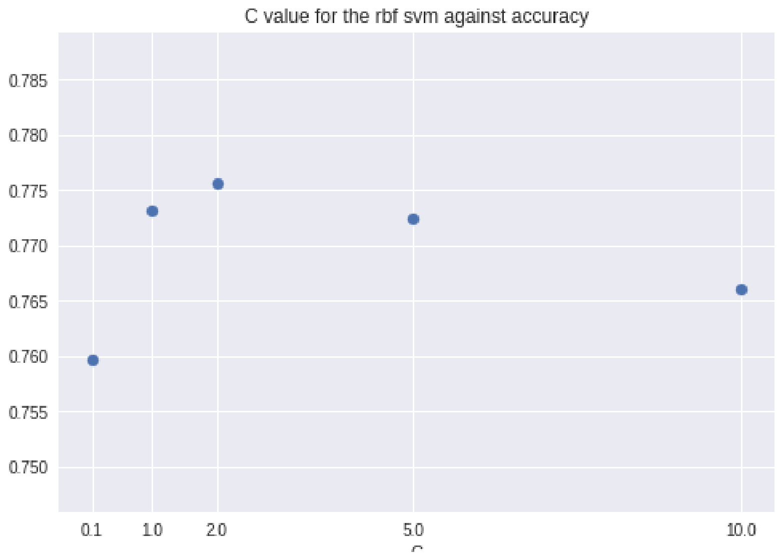


Figure 13: Performance of svm with RBF kernel on the validation set with appending POS tags

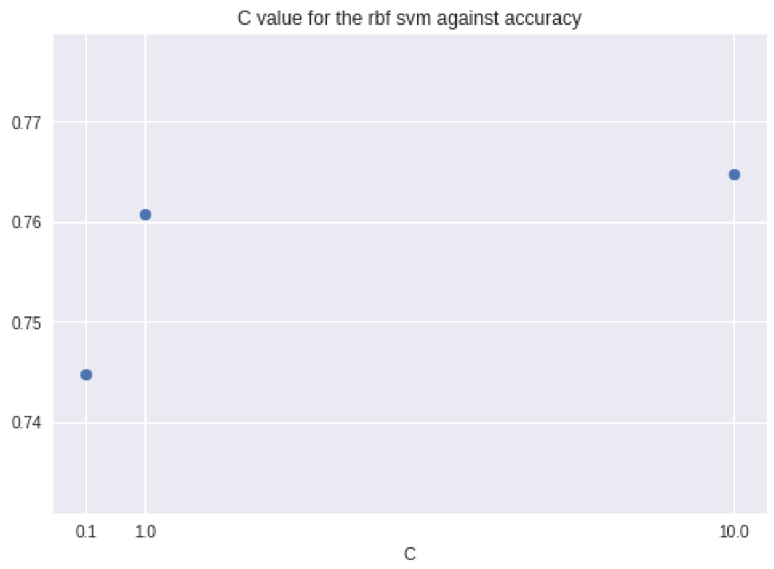


Figure 14: Performance of svm with RBF kernel on the validation set with noisy words removed

4.3 Doc2Vec Model

Here, we just used basic cleaning then used validation set to find best classifier, which was the SVM RBF classifier. Best accuracy using Doc2Vec as features was 85.9%.

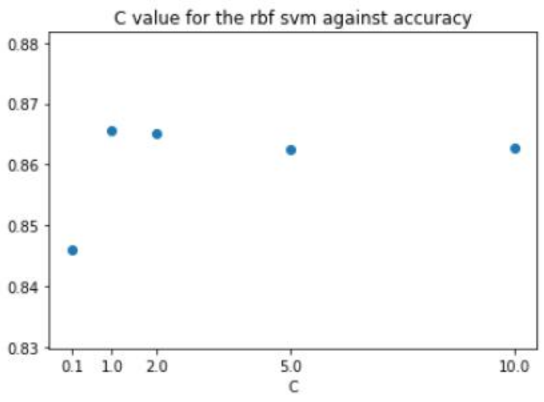
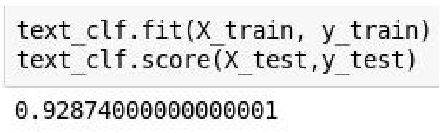


Figure 15: Best accuracy of tuning on SVM RBF kernel]

5 Conclusion

After trying all models, we observed that the TF-IDF has the highest accuracy with the Logistic Regression classifier.



```
text_clf.fit(X_train, y_train)
text_clf.score(X_test,y_test)
```

```
0.9287400000000001
```

Figure 16: Best Accuracy with TF-IDF and Logistic Regression classifier]

References

[1] Distributed representations of words and phrases and their compositionality.