University of Athens
Dept. of Informatics & Telecommunications

Artificial Intelligence II (Deep Learning for Natural Language Processing)

Rontogiannis Dimitrios – `sdi1900165`

Fall 2022 - Homework 1: Sentiment Classifier

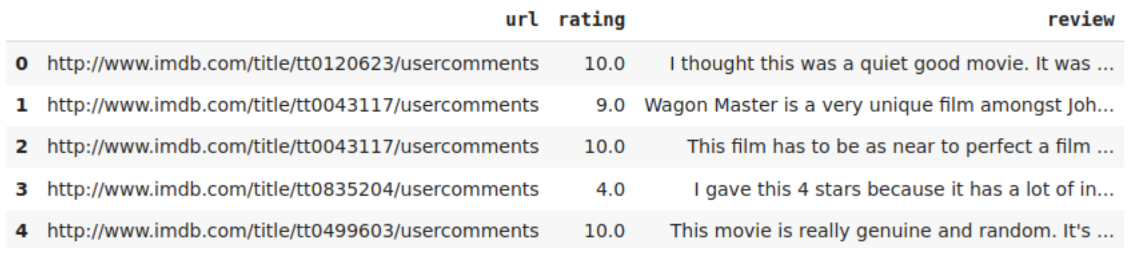
1 Introduction

We implement a sentiment classifier using logistic regression for the dataset *imdb-reviews.csv* which is provided. In order to choose all the details for developing a good model we run different experiments, with various vectorization algorithms, hyperparameters, train sizes, features etc.

2 Preprocessing - Cleaning data

Preprocessing the data is an important part of the process. Text data contains noise in various forms like punctuation, text in a different case, symbols with no meaning etc. We perform **seven** different steps to make sure that our data will be ready to be fed to our model.

Initial head of data frame:



	url	rating	review
0	http://www.imdb.com/title/tt0120623/usercomments	10.0	I thought this was a quiet good movie. It was ...
1	http://www.imdb.com/title/tt0043117/usercomments	9.0	Wagon Master is a very unique film amongst Joh...
2	http://www.imdb.com/title/tt0043117/usercomments	10.0	This film has to be as near to perfect a film ...
3	http://www.imdb.com/title/tt0835204/usercomments	4.0	I gave this 4 stars because it has a lot of in...
4	http://www.imdb.com/title/tt0499603/usercomments	10.0	This movie is really genuine and random. It's ...

2.1 HTML

These reviews are taken straight from the IMDB site, so there are some HTML tags that do not affect the sentiment of the sentence. We use regular expressions to filter the tags.

2.2 Number

Numbers might indicate the number of stars a user rated the movie, but also they could indicate the time they watched it, it's price when it was released in the cinemas, etc. Therefore, we use reguler expressions to filter every numeric character.

2.3 Punctuation

Punctuation could show both positive and negative emotion. For example the exclamation mark in the sentence ”I wish I could watch this movie again!” has a positive sentiment, but in the sentence ”I turned off the tv after 5 minuntes!” has a negative sentiment. We use regular expressions to replace any non-word and non-sentence.

2.4 Uppercase

We lowercase every letter because our model would treat upper case and lower case. We would not want ‘A’ and ‘a’ to be seen as different characters.

2.5 Tokenization - Bonus step

Before we continue with the next preprocessing techniques we need to tokenize our text into separate tokens which are essentially individual words.

2.6 Stopwords

We remove these words, which have low-level information, from our text, in order to give more focus to the important information.

2.7 Lemmatization

Lemmatization helps us to achieve the root forms of words. We aim to remove inflectional endings only and to return the base or dictionary form of a word. This way, words like ”give”, ”gave”, ”giving” will be treated equally.

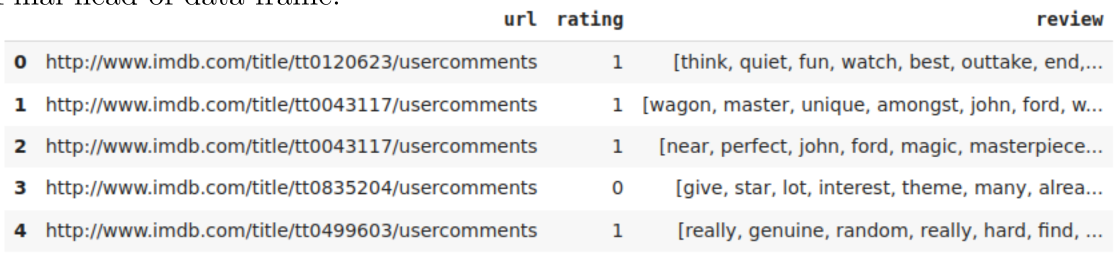
2.8 Rating - Bonus step

We know that a rating in the range $[0, 4.0]$ is categorized as negative and a rating in the range $[7.0, 10.0]$ is categorized as positive. As we will use Logistic Regression, we transform every negative rating to 0 and every positive rating to 1.

2.9 Rare/Frequent words

Rare and frequent words almost always offer no information for the sentiment analysis. For example words like "this", "and", "so" etc. are not useful for our model training.

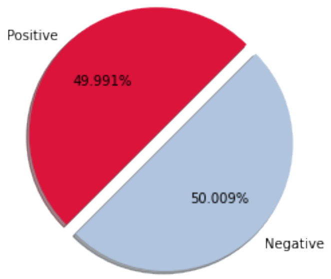
Final head of data frame:



	url	rating	review
0	http://www.imdb.com/title/tt0120623/usercomments	1	[think, quiet, fun, watch, best, outtake, end,...
1	http://www.imdb.com/title/tt0043117/usercomments	1	[wagon, master, unique, amongst, john, ford, w...
2	http://www.imdb.com/title/tt0043117/usercomments	1	[near, perfect, john, ford, magic, masterpiece...
3	http://www.imdb.com/title/tt0835204/usercomments	0	[give, star, lot, interest, theme, many, alrea...
4	http://www.imdb.com/title/tt0499603/usercomments	1	[really, genuine, random, really, hard, find, ...

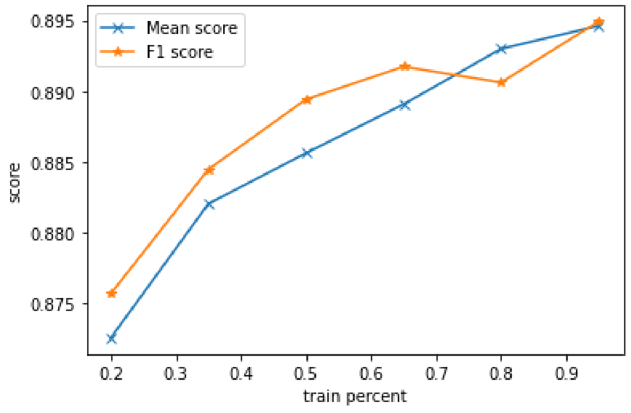
3 Training

We train our model with data with almost the same number of positive and negative reviews:

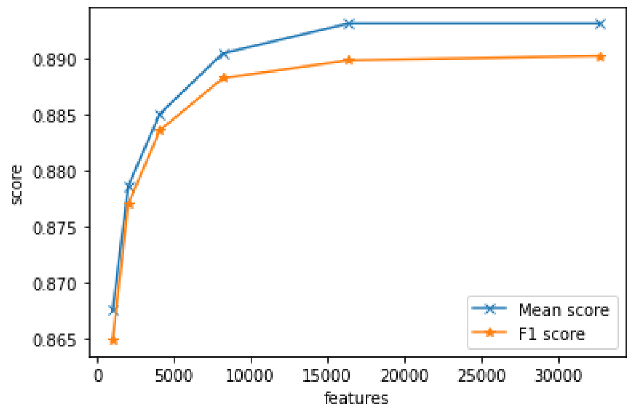


3.1 TFIDF

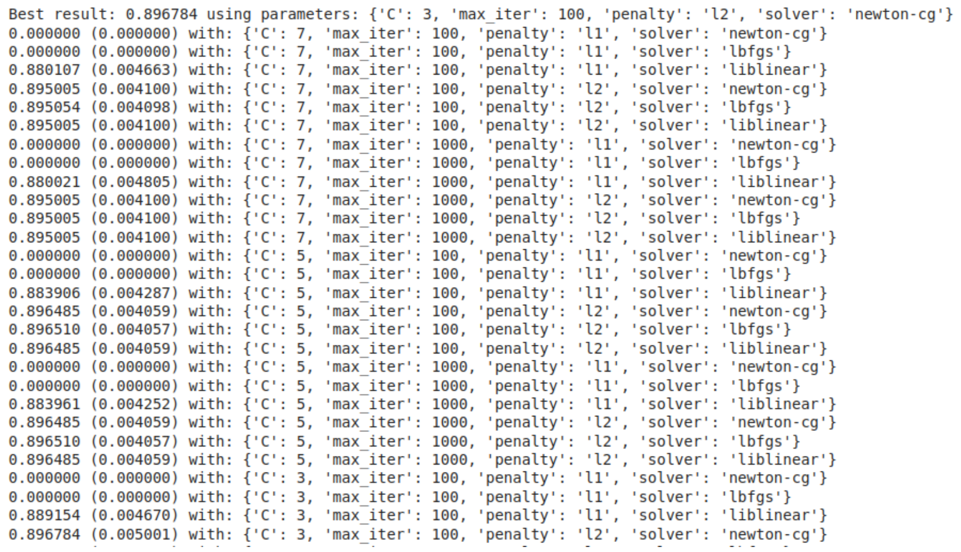
TFIDF or term frequency-inverse document frequency, reflects how important a word is to a document. We experiment with TFIDF vectorizer, which vectorizes text into a format more agreeable for training. We test the model's accuracy for different training sizes, different number of features, and use the algorithm for experimenting with the hyperparameters for Logistic Regression. For our testing we use cross validation with 10 different samples.



Here we can see that the mean score increases as the train size increases, as was expected. However, for the rest experiments for TFIDF vectorizer we use 80 percent of the dataset in order to avoid overfitting or underfitting.



In this graph we experiment with different powers of 2 as the number of max features in the TFIDF algorithm. We can see that the best value is around the number 16384, so for out model with TFIDF we will use 16500 features.

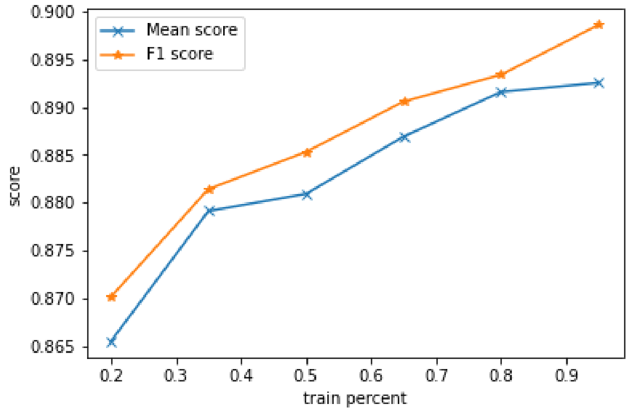


```
Best result: 0.896784 using parameters: {'C': 3, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 7, 'max_iter': 100, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 7, 'max_iter': 100, 'penalty': 'l1', 'solver': 'lbfgs'}
0.880107 (0.004663) with: {'C': 7, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
0.895005 (0.004100) with: {'C': 7, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.895054 (0.004098) with: {'C': 7, 'max_iter': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.895005 (0.004100) with: {'C': 7, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.000000 (0.000000) with: {'C': 7, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 7, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'lbfgs'}
0.880021 (0.004805) with: {'C': 7, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'liblinear'}
0.895005 (0.004100) with: {'C': 7, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'newton-cg'}
0.895005 (0.004100) with: {'C': 7, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.895005 (0.004100) with: {'C': 7, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
0.000000 (0.000000) with: {'C': 5, 'max_iter': 100, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 5, 'max_iter': 100, 'penalty': 'l1', 'solver': 'lbfgs'}
0.883906 (0.004287) with: {'C': 5, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
0.896485 (0.004059) with: {'C': 5, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.896510 (0.004057) with: {'C': 5, 'max_iter': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.896485 (0.004059) with: {'C': 5, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.000000 (0.000000) with: {'C': 5, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 5, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'lbfgs'}
0.883961 (0.004252) with: {'C': 5, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'liblinear'}
0.896485 (0.004059) with: {'C': 5, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'newton-cg'}
0.896510 (0.004057) with: {'C': 5, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.896485 (0.004059) with: {'C': 5, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
0.000000 (0.000000) with: {'C': 3, 'max_iter': 100, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 3, 'max_iter': 100, 'penalty': 'l1', 'solver': 'lbfgs'}
0.889154 (0.004670) with: {'C': 3, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
0.896784 (0.005001) with: {'C': 3, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
```

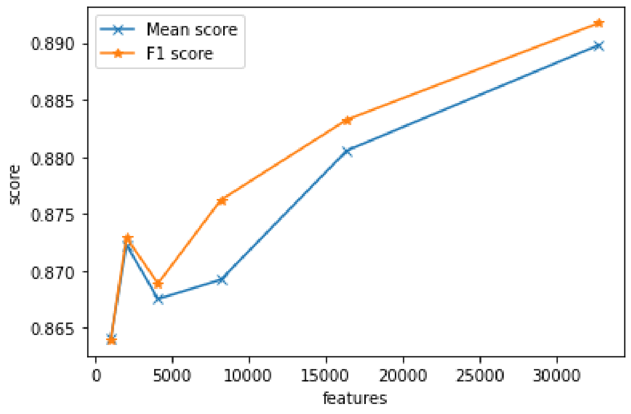
We run GridSearch in order to find best parameters for Logistic Regression, using as vectorizer the TFIDF. Above are some of the results. For our final model we use the parameters of the most accurate experiment.

3.2 Count Vectorizer

Count vectorizer is used to transform our data into a vector on the basis of the frequency of each word in the entire text. We test the model's accuracy for different training sizes and different number of features.



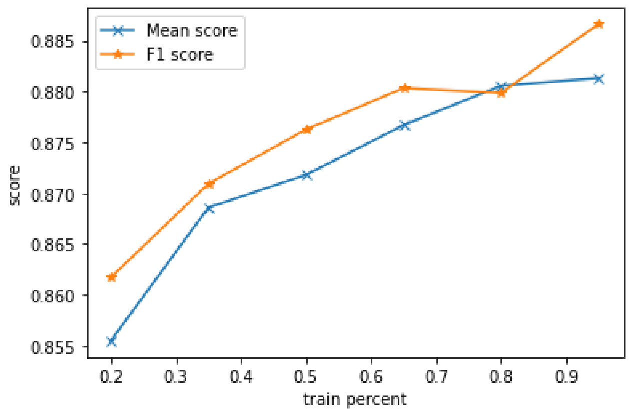
Here we can see that the mean score increases as the train size increases, as was expected. However, one can notice that F1 score's line in this graph is above mean score's line, in contrast with the corresponding graph of TFIDF.



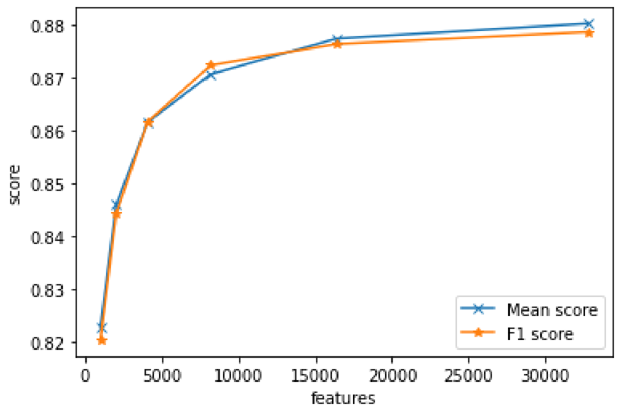
In this graph we experiment with different powers of 2 as the number of max features in the Count Vectorizer algorithm. We can see that the best value is probably for max features. Moreover, one can notice an interesting decline in mean and f1 score in the third experiment. Also, the increase is linear and not exponential.

3.3 Hashing Vectorizer

Hashing vectorizer applies a hashing function to term frequency counts in each document. We test the model's accuracy for different training sizes and different number of features.



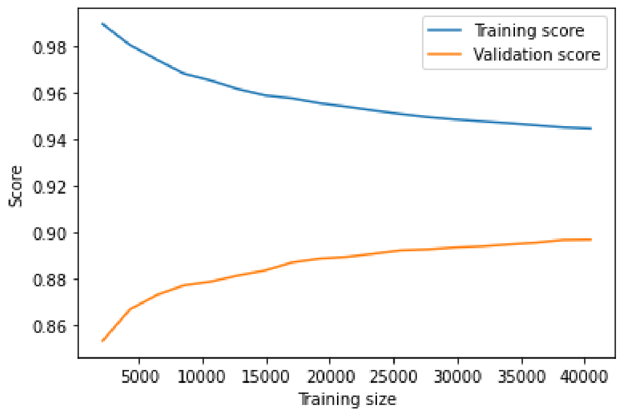
Here we can see that the mean score increases as the train size increases, as was expected. However, one can notice that F1 score's line in this graph is above mean score's line, in contrast with the corresponding graph of TFIDF. It has almost the same behaviour as the Count Vectorizer (only on experiment differentiates, where the F1 score becomes slightly less than the mean score)



In this graph we experiment with different powers of 2 as the number of max features in the Hashing Vectorizer algorithm. We can see that the best value is probably for max features. The behaviour is similar with the TFIDF algorithm.

4 Learning Curve

We test our best model's (TFIDF with 16500 features, LogisticRegression with solver=newton-cg, penalty=l2, C=3, iterations=100) train score and validation score and build the following learning curve:

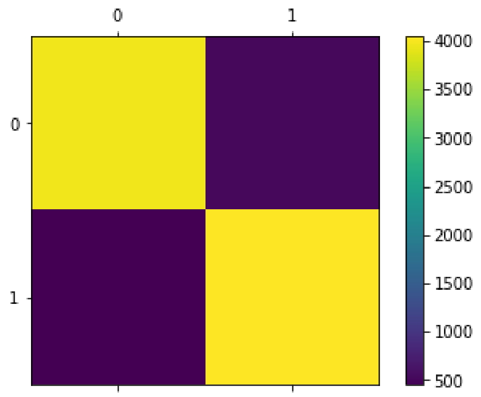


Training score starts high and decreases as the training size increases. In contrast Validation score starts lower and increases as the training size increases.

5 Confusion Matrix

We test our best model (TFIDF with 16500 features, LogisticRegression with solver=newton-cg, penalty=l2, C=3, iterations=100) with 20 percent of the dataset and build the following confusion matrix:

- True positive - 3977
- False positive - 523
- True negative - 4051
- False negative - 451

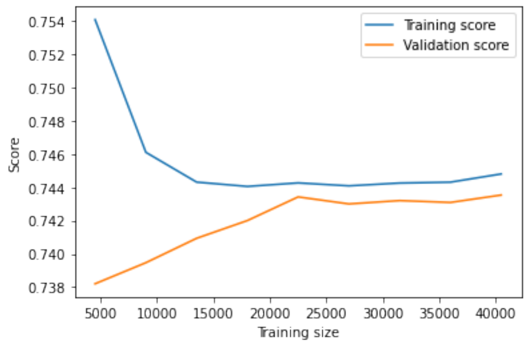


6 Testing

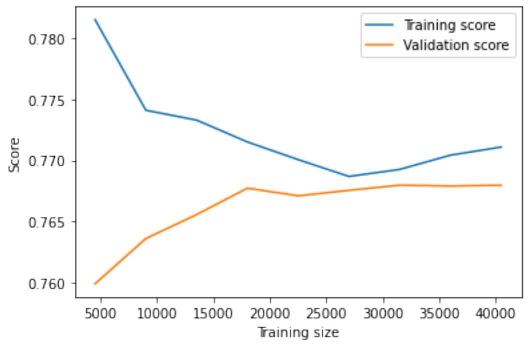
The reviewer is welcome to change the values of dataset and testset path in the beginning of the ipynb file. The testing can be done by executing the last cell, which trains the model with the hyperparameters found in the above experiments, with the dataset from the dataset path and then predicting the sentiment of the reviews from the testset from testset path.

7 Conclusion

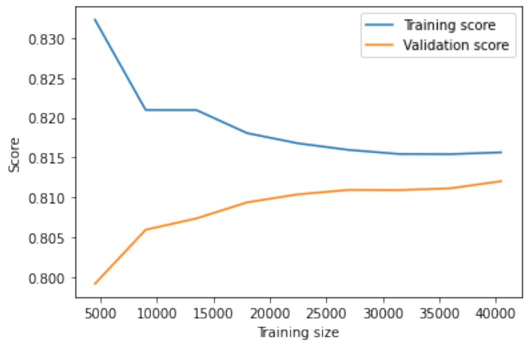
Even though our ”best model” has high accuracy on training dataset, one could argue that the learning curve seems to be underfitting, because of the gap between the two lines. This could be a result of the big number of features provided to the TFIDF vectorizer. Below are some additional experiments in order to find a more appropriate number of features:



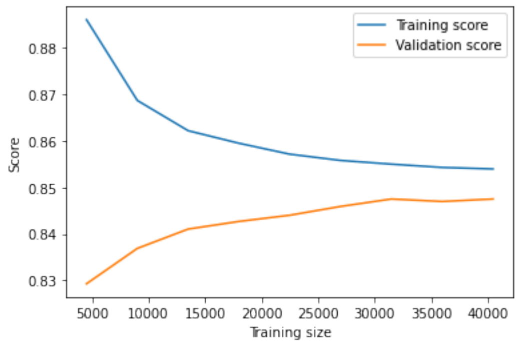
(a) 64 features



(b) 128 features

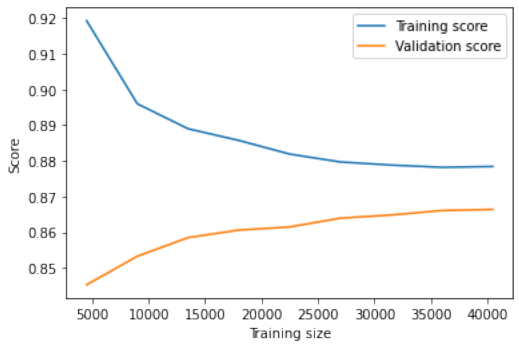


(c) 256 features

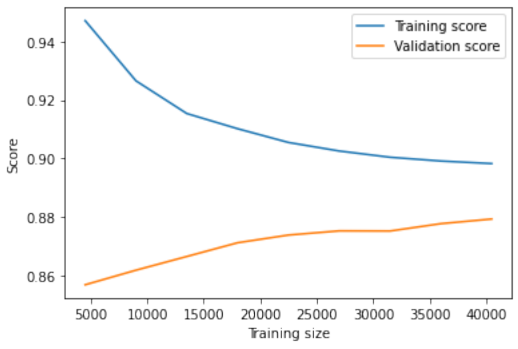


(d) 512 features

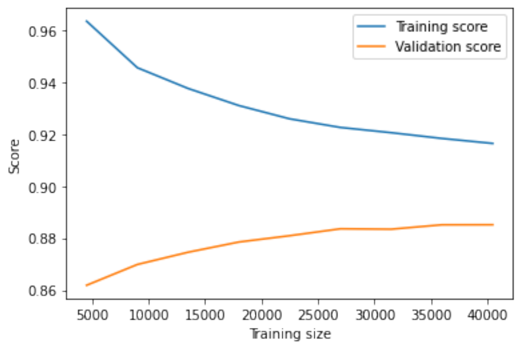
Figure 1: Learning Curves based on number of features in TFIDF



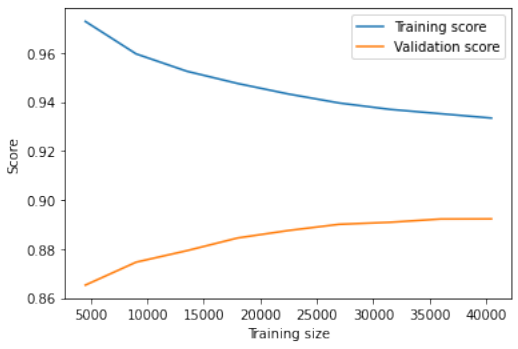
(a) 1024 features



(b) 2048 features



(c) 4096 features



(d) 8192 features

Figure 2: Learning Curves based on number of features in TFIDF

In conclusion, the model that is trained with around 512 features seems to have the best ratio of accuracy and gap between the training and validation score line.