# Project Report

## Introduction:

This project was an introduction to text classification. In order to achieve that, we worked with different tools and techniques, recording the outcome of our different models as we went. Before we delved into the actual classification problem however, we needed to preprocess our data. We kept the general preprocessing infrastructure shown in class and described by the exercise, adding our own touch at times given that each problem requires its own unique approach. We then trained and tested four different models (Dummy Classifier as baseline, Logistic Regression, Naïve Bayes and KNN classifier), giving a relatively extensive report of our different outcomes for each one, including the learning curves for our different subsets (train, test and development subset). The dataset selected was "the Large Movie Review Dataset" given by Stanford University (http://ai.stanford.edu/~amaas/data/sentiment/) which was also one of the datasets proposed.

## Data preprocessing:

Even before preprocessing our data, it was already split into training and test subsets, equally large, at 25.000 reviews each. The reviews themselves were also split exactly in half, with 12.500 being positive and 12.500 being negative. Originally the sentiment column had the values "neg" and "pos", which were then replaced with 0s and 1s respectively, since our models can only recognize numbers. Although the dataset was evenly split, we thought that 25.000 test cases would be a bit of an overkill for our (mostly) linear models. Thus, we concatenated the train and test data into a single Data Frame which we then split again into 3 subsets. We used train-test-split twice. First, we kept 70% of our data to train our models on. After that we performed a 60%-40% split on the remaining 30%, the larger part of which we used as test and the smaller as development data. After the rearranging performed above, the size of our train data is 35000 documents, with test and development data being 9000 and 6000 documents respectively. Furthermore, to get a better overview of our data we then estimated the average document length in our training dataset, which came at 232 words per review.

After deciding on the final shape of our data, came data cleaning and preprocessing. To give them their final morphology that would be transformed through the TFIDF vectorizer, we created a preprocessing function which did the following: It lowercases all text, removes html tags through regex, removes punctuation, numbers, single characters and multiple spaces, in that order. Finally, it tokenized the different reviews, used a Lemmatizer on them and reconstructed them giving us our final document form. We ran our preprocessing function on all of our subsets, and then initialized our vectorizer setting the n-grams hyperparameter to unigrams and bigrams, removing English stop words, as well as setting maximum features to 7.500.

As we wanted to further investigate our data, we initialized another TFIDF vectorizer object, using only unigrams and removing stop words, that we used to calculate our vocabulary size (number of features) which ended up being 78098 words in total. After fit-transforming the vectorizer on our training data, and also using it to transform our development and test data as well, only a few steps remained before we could start testing our models. We created a new Data Frame that only contained the train and development subsets, which would later be used in the grid search function to tune our parameters. Also, we used a mutual info classifier, to further reduce the number of features selected from 7500 to 2500, specifically the ones that present the highest information gain. The number of features kept was selected through several tests of our models' performance on the development subset, with the number of features ranging from 500 to 3750. The tests were not kept in the code, as that would make it very time-inefficient.

Finally, we created a function that plots the learning curves for a specific model given the current state of our datasets. Essentially, given a specific model it breaks the training set into 5 parts, namely 20%, 40%, 60%, 80% and 100% of the whole set. Then, it trains the model on those subsets, testing it on the subset itself as well as the entire test and development sets for each loop. It plots the resulting macro average F1 scores, giving us the 3 learning curves as requested.

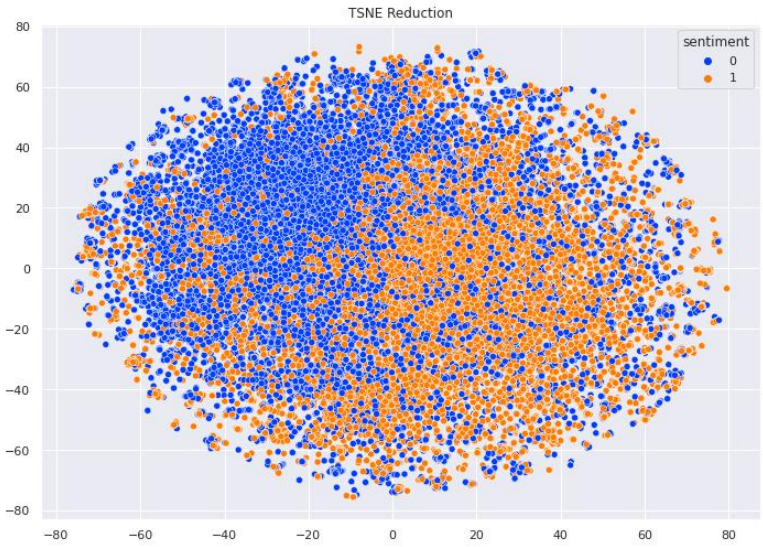


*Figure 1 Graphical representation of the data sentiments using the TSNE dimensionality reduction method*

## Model Selection:

The model training, testing and evaluation procedure was pretty similar amongst all models. First, we perform a grid search, with a predefined split based on a Data Frame we created in our preprocessing steps, in order to tune our parameters with our validation set, should there be parameters to be tuned. Afterwards, using the parameters calculated above, we fit our model to our training data, and test it on the test data. To assess our model, we then estimate different metrics (precision, recall, F1-score, ROC-AUC score etc.) on all of our different subsets as requested by the exercise. Finally, the F1 curves are plotted to give us a more intuitive understanding of our model's behavior.

## Dummy Classifier

In selecting our model, we first needed a baseline. Using a classic majority classifier as a baseline, namely Dummy classifier, we ran the necessary training and testing, as well as plotted its learning curves. Due to the training, test and development sets being balanced amongst the two classes, we got pretty much the exact same macro average F1 scores in all of them at 0.33. The learning curves are a bit more chaotic. Given that this is a majority classifier, this is absolutely logical and caused by the slight difference in balance between the sentiment values in the train, test and development sets. As we can see, the fluctuation is only evident in a very small scale, since they all range from 0.3325 to 0.3350. A majority classifier however is still a solid baseline in any single-label two-class classification problem so we decided it to be an appropriate standard.

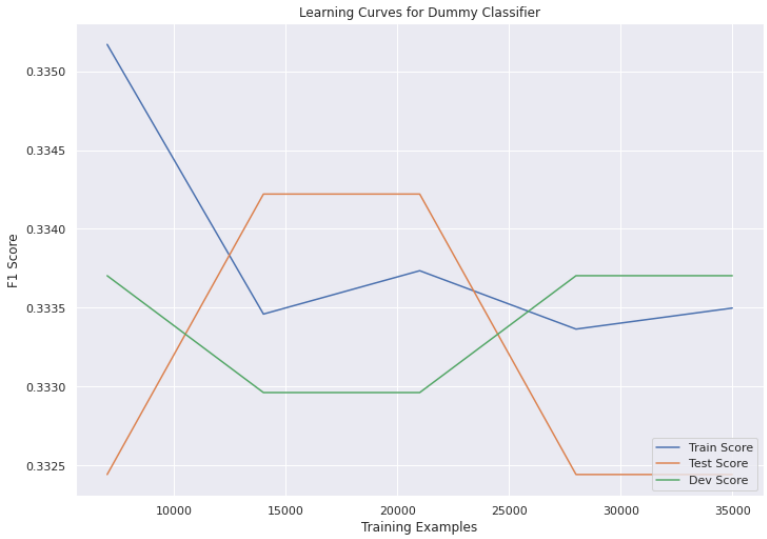


*Figure 2 Learning Curves -Logistic Regression Model*

| Dummy Classifier Model | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sets/Metrics** | **Precision** | | **Recall** | | **F1 score** | | **AUC** | Macro Average | | | |
| | Positive | Negative | Positive | Negative | Positive | Negative | | **Precision** | **Recall** | **F1** | **AUC** |
| **Test** | 0,5 | 0 | 1 | 0 | 0,66 | 0 | 0,5 | 0,25 | 0,5 | 0,33 | 0,5 |
| **Dev** | 0,5 | 0 | 1 | 0 | 0,67 | 0 | 0,5 | 0,25 | 0,5 | 0,33 | 0,5 |
| **Train** | 0,5 | 0 | 1 | 0 | 0,67 | 0 | 0,5 | 0,25 | 0,5 | 0,33 | 0,5 |

*Table 1 Dummy Classifier Classification Report*

## Multinomial Naïve Bayes

The next classifier we tried was the Naïve Bayes classifier. First, we tried to select the best hyperparameter value for alpha through the grid search method, which returned a value of alpha equal to 1.803. Then, using that alpha value we trained and tested the model as described above. The results for our model were pretty uniform all in all. With 0.88 ±2% in all macro averaged scores in the train, test and validation sets, as well as similar scores in all metrics across the board, it seemed like a pretty "well behaved" model at first. The learning curves results however differ from the expected.

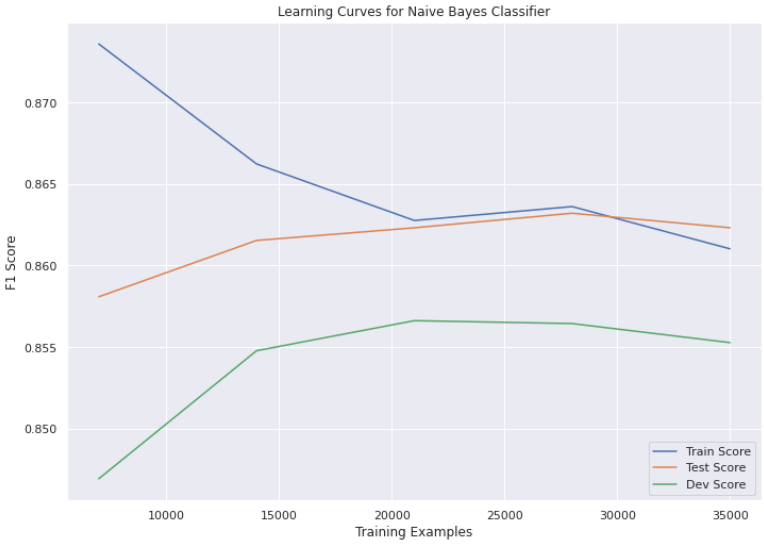


*Figure 3 Learning Curves -Multinomial Naive Mayes Model*

It is quite apparent that the F1 score for our test set is above the train set's one in the final stretch of the plot, even if only by less than $3 \cdot 10^{-3}$. That unnatural and counter intuitive result leads us to a couple of

observations. This could be caused by a large variance in our data, given how we are tackling an NLP problem. In such a case, we could be seeing the model being confused even in its own training data, as it cannot follow the large fluctuation of the datapoints, yet exactly due to that, we could have gotten a bit "lucky" with the training set at some point. Another interesting note is that the dev score is substantially lower than both the train and test scores, even though the hyperparameter tuning was performed to maximize its efficiency in particular.

| Naïve Bayes Classifier Model | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sets/Metrics** | **Precision** | | **Recall** | | **F1 score** | | **AUC** | Macro Average | | | |
| | Positive | Negative | Positive | Negative | Positive | Negative | | **Precision** | **Recall** | **F1** | **AUC** |
| **Test** | 0,85 | 0,87 | 0,88 | 0,85 | 0,86 | 0,86 | 0,86 | 0,86 | 0,91 | 0,86 | 0,86 |
| **Dev** | 0,84 | 0,87 | 0,87 | 0,84 | 0,86 | 0,85 | 0,85 | 0,86 | 0,86 | 0,86 | 0,85 |
| **Train** | 0,85 | 0,88 | 0,88 | 0,84 | 0,86 | 0,86 | 0,86 | 0,86 | 0,86 | 0,86 | 0,86 |

*Table 2 Multinomial Naive Bayes Classifier- Classification Report*

## Logistic Regression

Up next was the model that we were instructed to use, that being Logistic Regression. Logistic Regression in general procured very encouraging results. Running its grid search on both "C" and "penalty" parameters, we ended up with a C value of 1 and an "l2" penalty as the best approach for our data. We then fit our model as before, returning the relevant metrics as well as learning curves. We had a very stable macro averaged F1 score, ranging from 0.89 ±1% on the validation, train set. The ROC-AUC scores are also quite high, with 0.88 on both test and dev sets and 0.9 on the train set. The behavior of the learning curves is quite orthodox as well, giving us an overall very satisfactory model. It should be noted that the slight drop in the training curve is mathematically consistent. As we add more points to our training set, the logistic function we try to fit to our data moves slightly to a better fit, still "missing" quite a few points. Those slight moves however further generalize the function, giving us a better result in the test and development sets, perfectly explaining the plot. We specifically selected the "saga" solver as it seems to be uniformly the best in most situations, especially in sparse datasets such as ours, since we have used tfidf.

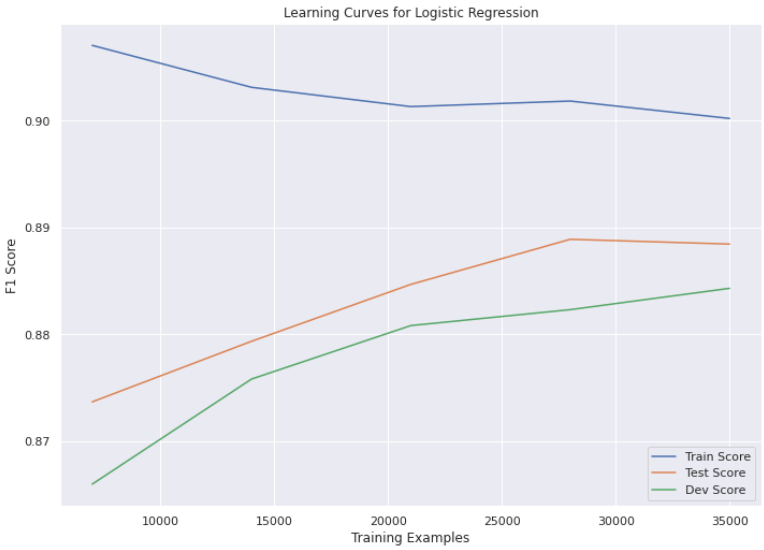


*Figure 4 Learning Curves-Logistic Regression Model*

| Logistic Regression Classifier Model | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sets/Metrics** | **Precision** | | **Recall** | | **F1 score** | | **AUC** | Macro Average | | | |
| | Positive | Negative | Positive | Negative | Positive | Negative | | **Precision** | **Recall** | **F1** | **AUC** |
| **Test** | 0,88 | 0,9 | 0,9 | 0,88 | 0,89 | 0,89 | 0,88 | 0,89 | 0,89 | 0,89 | 0,88 |
| **Dev** | 0,87 | 0,9 | 0,9 | 0,87 | 0,89 | 0,88 | 0,88 | 0,88 | 0,88 | 0,88 | 0,88 |
| **Train** | 0,89 | 0,91 | 0,91 | 0,89 | 0,9 | 0,9 | 0,90 | 0,9 | 0,9 | 0,9 | 0,90 |

*Table 3 Logistic Regression Classifier- Classification Report*

## KNN

Finally, we turned to a non-parametric classifier, that being KNN. After performing a grid search for k in the range from 8 to 20, we ended up with 19 as the best number of neighbors in that range. Using that k to fit the model, we ended up with relatively lower scores in all macro averaged metrics. With our macro averaged F1 on test and dev sets being 0.7 and 0.69 respectively and 0.78 for train, we did not have a model as successful as its predecessors. Looking at the graph, we can see a very interesting difference to the previous ones. Other than the test and dev ones, the training curve also has an upward trend. This is quite sound mathematically as well. By adding more possible neighbors to choose from, we can only help the KNN model achieve better results regardless of whether or not it is tested on the train or the test set. This is innate in its non-parametric nature, which also causes the relatively low initial score.

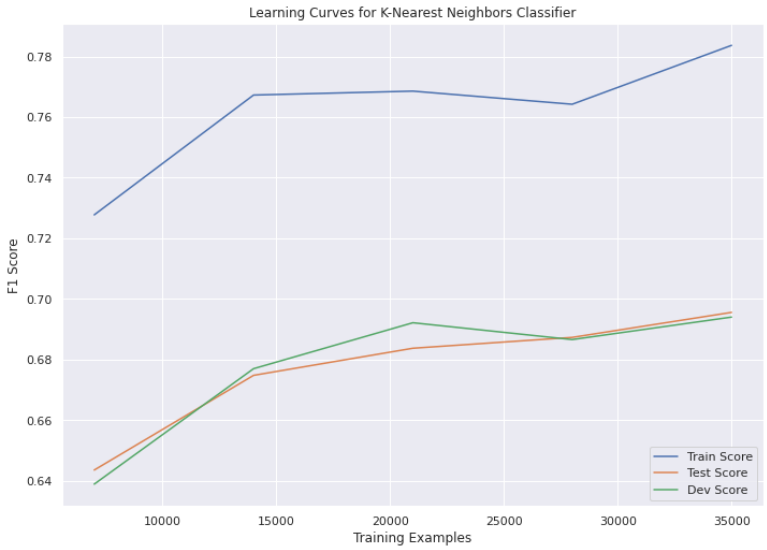


*Figure 5 Learning Curves -KNN Model*

| Sets/Metrics | KNN Classifier Model | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | | Recall | | F1 score | | AUC | Macro Average | | | |
| | Positive | Negative | Positive | Negative | Positive | Negative | | Precision | Recall | F1 | AUC |
| **Test** | 0,69 | 0,7 | 0,7 | 0,69 | 0,7 | 0,69 | 0,69 | 0,7 | 0,7 | 0,7 | 0,69 |
| **Dev** | 0,69 | 0,69 | 0,7 | 0,69 | 0,69 | 0,69 | 0,69 | 0,69 | 0,69 | 0,69 | 0,69 |
| **Train** | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 | 0,78 |

*Table 4 KNN Classifier- Classification Report*

## Conclusion:

In this Project we experimented with data preprocessing and text classification. The different tools and methods' necessity or usefulness became evident through a practical approach of a real-world problem. It was especially apparent how oftentimes, pure numbers and metrics told a different tale than the learning curves and how important grasping abstract mathematical concepts such as machine learning through visual, intuitive means can be. Experimenting with grid search and hyperparameter tuning also tied into the importance of parameters and computational methods, as we saw how much different parameters can affect the results of a model, as well as how the best parameter could be almost impossible for us to decide on without methods such as grid search. It is impressive how NLP intertwines mathematics in its most practical form, with understanding of real language contradictions and behaviorisms.