Machine Learning Mini Project 2: IMDB Sentiment Analysis

Humayun Khan

Boury Mbodj

Michael Segev

Group 47

February 23, 2019

1 Abstract

The classification task of predicting the sentiment of IMDB movie reviews was studied, where a sentiment can either be positive or negative. The task in question can typically be split into four sub-tasks. The first is the pre-processing phase which parses the raw text data and removes invalid characters and other text anomalies which could hurt performance. The second sub-task consists of producing meaningful features which will be used as input to the classifier. Next, the classifier takes the features as input to be trained (supervised learning) and produces a classification prediction. Lastly, the predictions must be evaluated in a systematic way to obtain an accuracy score of the entire model. Several ways of implementing the feature extractions were investigated in order investigate the pros and cons of each. These feature extraction pipelines can be very simple in nature, such as a binary word feature which asserts if a word is present in a review, or can be very complex such as a doc2vec algorithm which produces a numeric vector representing the meaning of a review. Next, different classifier models were investigated in order to compare performance and computational requirements of each. We implemented a Bernoulli Naive Bayes model from scratch, and made use of the Scikit Learn package to experiment with Logistic Regression, Decision trees, Support Vector Machines (SVMs) and neural networks. Finally, a k-fold cross validation pipeline was implemented to properly validate the different models. Experiments using more sophisticated lemmatization and/or word embeddings techniques only added compute cost with no performance gain. The configuration which ultimately yielded the best performance was one where lemmatization was used, followed by TF-IDF, into a 2 hidden layer neural network (60 nodes, 30 nodes). This produced an accuracy score of 91.76% measured using hold one out validation and 91.09% on kaggle.

2 Introduction

Natural language processing (NLP) is a machine learning subfield which attempts to understand and predict the underlying meaning or intent of a piece of text. This is a very broad and challenging task to which its full completion is outside the scope of this project. This report will specifically focus on the NLP task of sentiment analysis, which attempts to predict whether a given text expresses a positive or negative sentiment using supervised learning algorithms. This is a relatively simple NLP task which only requires a binary classifier, and ignores the intensity of the sentiment and also ignores neutral texts. Even though the scope of the task is relatively narrow, we are still able to compare different techniques to which our conclusions can be extrapolated to more challenging tasks.

This report summarizes the investigation and implementation of several systems which seek to predict the sentiment of IMDB movie reviews. The provided dataset included 12500 labelled negative, 12500 labelled positive, and 25000 unlabelled movie reviews. The reviews came in individual files, and typically include 4-5 sentences of people describing their opinions of a given movie.

NLP classification systems can be seen as 4 segments placed in series, each having a number of different implementations and parameters. The first segment consists of the preprocessing step where the raw movie review text files are opened and cleaned in order to reduce noise. The cleaning process should not remove any relevant information from the data, only characters and anomalies that contain no meaning and have no connection to the sentiment of the text. We implemented a cleaning step that removed html and xml tags, punctuation and any other non-word characters such as brackets. The second segment consists of the feature extraction step in which the cleaned text is now processed in some way in an attempt to extract meaningful inputs to the next segment (classifier). A meaningful feature is one which has a strong correlation with the label. For example, finding the word amazing numerous times in a comment intuitively has a strong correlation with the comment having a positive sentiment. A simple feature extraction which was considered was a bag of words feature set which counts how many times words appear in each review. The third segment consists of the actual classifier which takes input feature data and outputs a sentiment prediction. We investigated several models including a Naive Bayes classifier made from scratch. Each of these models operate fundamentally differently and make different underlying assumptions pertaining to the data. The models also have hyper-parameters which tune the way they calculate their cost function and at the end of the day also affect performance. Finally, the fourth segment is the cross validation pipeline which is simply used to measure the accuracy/performance of the previous 3 segments. If one is not careful, it is easy to misinterpret results which is why a robust cross validation such as the k-fold method ensures that the classifier is not only reporting results on a small constant validations set or even worse validating using data it has trained with.

Our experiments show that a strong pre-processing step is crucial to obtaining a good performing model. Several models were investigated, and ultimately, a 2 hidden layer neural network yielded the best performance. When used in combination with Lemmatization and TF-IDF, it resulted in an accuracy score of 91.76% measured using

hold one out validation.

3 Related work

In the past, a Naive Bayes classifier with bag of words or n-grams would have been the state of the art method of obtaining a sentiment classification from a given text. Different techniques of grouping similar words together would have been used to increase performance, as well as many other hand crafted methods of better extracting the underlying meaning of a sentence into input features.

In recent years, deep learning has allowed computer scientists to perform more advanced NLP. One of the most prominent techniques is to automatically extract the meaning of a word or phrase into a series of numbers, called an embedding in the Vector Space Model(VSM). In the Vector space model, each word is represented by a vector and the continuous similarities between words are represented by the distance between their respective word vectors. One of the popular works in unsupervised VSMs is where the context in which a word appears is used to guess the meaning of the word [2]. Thus, words that appear in similar contexts are assumed to be similar in meanings. The semantic similarity between words is captured by computing the distance between their word vectors.

However, because no sentiment polarity information is associated with the training samples used for learning word vectors, the word vector representation does not capture the sentiment similarity between the words. Thus, the work in [2] was extended in [3] which uses the label information of documents to learn vector representation of words that capture semantic as well as sentiment similarities between words. For example, the algorithm in [2] will capture that terrible and awful are similar in meaning, but it will not capture the negative sentiment associated with these words. However, the word vectors representation for the two words using the algorithm in [3] will not only capture the semantic but also the sentiment similarity.

4 Dataset and setup

We were given a IMDB movie review dataset with labelled and unlabeled movie reviews in separate text files. The provided dataset included 25000 labeled reviews, which had half positive and half negative sentiments, and 25000 unlabeled reviews which we were tasked to classify. The dataset had some html tags, had lots of non-letter characters and made use of a lot of slang words. Given that the html tags and non-letter characters provides little to no information on the meaning of a review, the first step of our preprocessing step involved removing them. For many of our feature extraction pipelines, tokenization of the review words was required, some using n-grams and some without.

5 Proposed Approach

A number of experiments were performed in order to gain understanding of their performance in the context of IMDB movie review sentiment classification. As previously described, this is a multi-layered algorithm, and in order to learn the impact of changing each layer, the other layers were kept constant, such that only one parameter is changing at a time. This is perhaps an oversimplification, since there could be cross-correlation between the performance of a given type of model and the feature extraction used, but it served as an adequate starting point.

Before investigating could begin, the data first had to be cleaned, in order to make it easier to process. By finding and removing noise words in advance, model performance is known to improve.The cleaning process includes removing the HTML tags (using the Python package BeautifulSoup), unnecessary punctuation and convert to lowercase. We experimented using an off-the-shelf stop words dictionary which systematically resulted in lower performance since it is likely that valuable information is thrown away in this process.

In order to capture the sentiment as accurately as possible, we experimented with three different feature extraction pipelines:

1. Feature Presence (FP) a binary value, indicating whether the feature exists in the text or not. In the text hello world, only the features hello and world are set to 1, and all the other words in the vocabulary are set to 0. The vocabulary is the set of all the words we see in the corpus.

2. Feature Frequency (FF) - A real value, indicating the frequency of the feature in the given example, normalized by the size of the text (in words).

3. TF-IDF - a real value, indicating the frequency of the feature in the given text, divided by the logarithm of the number of examples from the corpus containing this feature. In other words, for feature f_i :

$$TF - IDF(f_i) = FF_i/log(DF_i)$$

Where FF_i is the feature frequency of f_i as before, and DF_i is the document frequency of f_i , that is, the number of documents containing f_i . The intuition behind this weighting function is to give a larger weight to features that occur less often in the corpus rather than the common ones. This way, we are increasing the impact of rare words over common words.

We compared the performance of these feature extraction pipelines using Logistic Regression and found TF-IDF to be the best-performing pipeline - the different results can be found in the table below .

Binary Counts	**Counts**	**TF-IDF**
0.8964	0.9004	0.9152

Table 1: Feature extraction comparisons using logistic regression

One of the main feature extraction methods that we used is lemmatization, which is the process of finding the root of a given word which still holds the meaning of the original word. The motivation behind this is to group up all word features with the same meaning, but that simply have a different conjugation, or plural versus singular, etc. By doing so, a model can find trends more easily with less training data. We implemented NLTKs lemmatization as a preprocessing step in hopes that it would improve model performance. It is of interest to note that the Lemmatizer itself is a model trained with machine learning, and that it is used to generate better features for a subsequent model, a common practice in machine learning tasks.

Along with the use of Lemmatizers, during our feature design stage we experimented with word embeddings, which are essentially spatial coordinates that contain the meaning of a given word. If two words have similar meanings, the coordinates will be close to each other. A popular model created by Google is known as Word2Vec which takes a word as input and outputs the associated word embedding. We can then use this word embedding as input features to a model and should provide improved results according to literature[1]. A further play on this idea is the use of a library called Doc2Vec, which is capable of taking an entire sentence, and obtaining a single coordinate representing its meaning. This is an attractive proposition for us as it would allow the entire movie review to have a single coordinate to represent its meaning. The performance if this technique will ultimately depend on the accuracy and performance of the libraries used.

A Bernoulli Naive Bayes model was implemented from scratch in Python using only numpy as external library dependency. It was designed in an object oriented approach, inspired by NLTKs, such that one can simply instantiate an object of class NaiveBayes, train it and use it to predict new inputs with class methods. The string parsing and tokenization were also done from scratch such that the class can be used without additional external dependencies for feature extractions if one so desired. The trainFromXY method can be used if one has already converted the raw data into numpy feature matrix X and label matrix Y. The training phase takes the training matrix X and first computes the prior probability of each label, $P(pos)$ & $P(neg)$. For $P(pos)$ this is done by taking the number of positive samples and dividing by the total number of samples. $P(neg)$ can then easily be determined by taking $1 - P(pos)$ since we only have two labels. Next, the conditional probabilities with Laplace smoothing of each feature,$P(x_i|pos)$ and $P(x_i|neg)$, is computed. For $P(x_i|pos)$, this is done by counting the number of data points (reviews) that contain a given feature (word), adding 1 to it (Laplace smoothing) and dividing by the total number of positive reviews plus the vocabulary size. The same is done for the negative case to get all the $P(x_i|neg)$. Laplace smoothing is important to implement or else if, for example, the model tries to predict the sentiment of a review which includes a word it has never seen before in the positive training data, it will automatically label it as having negative sentiment, which is undesired. Once trained, the runFromX method can be called to predict labels for new data matrix X. This is done by computing a delta score, which is done by taking the $log_{10}(P(pos)/P(neg))$ and for each feature, adding $log_{10}(P(x_i|pos)/P(x_i|neg))$ if the feature is 1 or adding $log_{10}(1 - P(x_i|pos)/1 - P(x_i|neg))$ if the feature is 0. If the delta score for a given review is greater than 0, the the predicted label will be positive, else negative. For a large dataset, the algorithm was computationally expensive, so in order to improve performance, the log ratios used in the prediction step were pre-computed in the training phase, which resulted in a dramatic performance increase since it was only computed once instead of once per data point. Using the sci-kit learn library, we also implemented the following supervised learning models: logistic regression, decision trees, support vector machines and neural networks.

Finally, Grid Search was implemented, which is a method to perform hyper-parameter optimization, that is, to find the best combination of hyper-parameters for a given model and test dataset. In this scenario, we have several models, each with a different combination of hyper-parameters. Each of these combinations of parameters, which correspond to a single model, can be said to lie on a point of a "grid". The goal is then to train each of these models and evaluate them e.g. using cross-validation. You then select the one that performed best.

Normally in a machine learning process, data is divided into training and test sets; the training set is then used to train the model and the test set is used to evaluate the performance of a model. However, this approach may lead to variance problems which refers to the scenario where our accuracy obtained on one test is very different to accuracy obtained on another test set using the same algorithm.

The solution to this problem is to use K-Fold Cross-Validation for performance evaluation where K is any number, we used K = 5 in our case. The process of K-Fold Cross-Validation is straightforward. The data is divided into K folds. Out of the K folds, K-1 sets are used for training while the remaining set is used for testing. The algorithm is trained and tested K times, each time a new set is used as testing set while remaining sets are used for training. Finally, the result of the K-Fold Cross-Validation is the average of the results obtained on each set.

6 Results

In the search of a hypothesis with the best fit, we used different classification models.

Logistic regression

We began our experiments by training our data on a logistic regression classifier which uses a discriminative learning approach that models the log-odds with a linear function. We observed the following results from our experiments on Table 3 and Figure 2 in the appendix.

Num data	Parameters	Accuracy	Precision	Runtime	Big O
2500	norm=l2, c=1,cv=5	0.9036	0.90	7.36 s	O (mn)

Table 2: Logistic regression perfomance results

Decision Trees

Our second approach consisted in using decision trees, another eager learning approach which has a sequential decision making process. We observed that this model doesnt provide great accuracy in contrast to our other models. The construction of the decision trees doesnt involve any pruning and a sample logic is demonstrated in Figure 1 in the appendix. We can assume that the loss of performance can be due to irrelevant features or outliers from our training data. Another possible explanation of the low accuracy is the sensitivity of the output as well as the high dimensionality of our matrix which consists of approximately 300 000 features. A summation of our analysis with regards to this model is that it creates overly complex trees that do not generalize data well and thus overfits.

Num data	Accuracy	Precision	Runtime	Big O
2500	0.7196	0.72	2min 24s	$O(m.n^2)$

Table 3: Decision trees perfomance results

Support vector machines

Our third model uses a non probabilistic approach with the goal to partition the feature space into different regions and classify points based on the region where they lie (Hamilton, 2019) The accuracy of our model was highly dependent on the choice of hyperparameters. Training the data with the assumption that the data was linearly separable during our first try gave us very bad results (below 50%). The resulting support vector machine from our first trial attempted an exact separation of the training data in the original input space without allowance for misclassification. By implementing a soft SVM, varying the value of C, which is analogous to the regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity (Bishop, 2007), as well as our kernel and gamma, we were able to obtain better results. The choice of a lower C equal to 10.0 modified the algorithm of the support vector machine in order to allow some misclassification which is good in practice as class conditional distributions may overlap. We used a radial basis kernel and relatively medium gamma in order to smoothen the function and still be able to capture the high dimensionality. Our results using this model are captured on Table 5 and Figure 3 in the appendix.

Num data	Parameters	Accuracy	Precision	Runtime	Big O
2500	C= 10.0,Gamma=0.1,kenel=rbf	0.9168	0.91	51min 55s	O (n^3)

Table 4: Support vector machine perfomance results

Neural networks

In our last experiment we used supervised neural networks form the sci-kit learn framework using two levels of hidden layers of size 60 and 30 and trained it with backpropagation. Neurons were made to use the sigmoid activation function, and two hidden layers were chosen since literature suggests that this satisfies the majority of simple problems. The time complexity of our back propagation neural network is in the table 6 where m=(30000) is the number of features, k the number of hidden layers(two), o the output of neurons and i the number of iterations which is a max of 200 in our case. In terms of runtime, our neural networks is expensive to compute and has a non-convex loss function which makes our model hard to reproduce as it can converge to different local minima. However its performance (Table 6) beats by far our other models and thus qualifies it as our best performing model. Furthermore, we can observe significantly lower false positive and false negatives in the confusion matrix compared to out other models(Figure 4 in the appendix).

Num data	Parameters	Accuracy	Precision	Runtime	Big O
2500	Hidden layers (60,30)	0.9176	0.92	20min 49s	O ($n.m.h^k.o.i$)

Table 5: Neural networks perfomance results

7 Discussion and Conclusion

After having performed numerous experiments with different models, feature extractions and text pre-processing, we found that obtaining a system that had nominal performance was relatively straightforward to obtain. A simple tokenization with binary word presence feature was enough to obtain accuracy scores in the high 80% range using a logistic regression model. In order to achieve higher accuracy, we employed more sophisticated Natural Language Processing libraries which could extract more meaningful features from the text. Optimal model Hyper-Parameters were also obtained through time consuming grid searches, which ultimately re-trains the model and performs k-fold cross validation while sweeping a certain variable.

Our experiments show that using lemmatization followed by bag of words and 4-gram counter followed by TF-IDF was the best performing feature extraction pipeline. NLTKs lemmatization was used to combine words that have the same meaning which helps the model to find word trends more easily especially if a word with a certain meaning has many synonyms or ways of being conjugated. By counting the word occurrence for individual words and n-grams with $n \leq 4$, the model is able to find words and word sequences which have a correlation with the sentiment. N-grams are useful since the sequence of words is very important in the meaning of a sentence, so its intuitive that performance improves by using n-grams. TF-IDF was used and essentially gives a bigger weight to words which dont generally appear in the entire training set and reduces the impact of words which are very common in the training set. The reasoning behind this is that words which occur frequently, typically dont contain much information and vice-versa. With this feature extraction pipeline, our best performance was obtained using a neural network classifier with two hidden layers of size 60 and 30. Our final submission result based on kaggle 30% testing set is 0.9109 and is captured on Figure 5 in the appendix.

With more compute power, or with GPU acceleration using a library such as PyTorch, we could increase the speed at which we could run experiments and land on a model with better performance. Additionally, using ensemble methods such as *Boosting* or *Stacking* as recently covered in the course could potentially lead to better performing models, where instead of only using one model to fit the data, multiple models could be used which could compensate for each others shortcomings.

8 Statement of Contributions

Michael implemented Naive Bayes from scratch. Humayun worked on the feature extraction pipelines. Boury experimented with different classifiers from SciKit. All three group members collaborated on the report

References

[1] Mukku, S., Mamidi, R. and Choudhary, N. *Enhanced Sentiment Classification of Telugu Text using ML Techniques* International Joint Conference on Artificial Intelligence. New York: CEUR Workshop Proceedings, pp.29-34. July 2016

[2] Andrew L Maas and Andrew Y Ng. *A probabilistic model for Semantic Word Vectors.* NIPS Workshop on Deep Learning and Unsupervised Feature Learning.

[3] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts *Learning word vectors for sentiment analysis.* 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 142-150.

[4] Bishop, C. (2007). *Pattern recognition and machine learning.* New York: Springer, p.332.

[5] Hamilton, W. (2019). *COMP 551 - Applied Machine Learning* Lecture 10 — Support vector machines.

Appendix

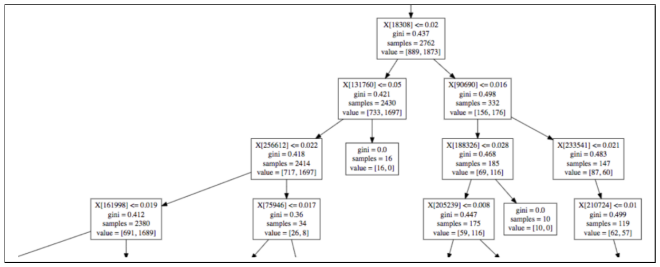

Figure 1: Sample decision tree logic

```
Accuracy score for logistic regression :
0.9036
Classification score for logistic regression :
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      1248
           1       0.90      0.91      0.90      1252

   micro avg       0.90      0.90      0.90      2500
   macro avg       0.90      0.90      0.90      2500
weighted avg       0.90      0.90      0.90      2500

Confusion matrix for logistic regression :
[[1118  130]
 [ 111 1141]]
```

Figure 2: Logistic regression accuracy score report

```
Accuracy score for support vector machines :
0.9168
Classification score support vector machines :
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      1248
           1       0.92      0.92      0.92      1252

   micro avg       0.92      0.92      0.92      2500
   macro avg       0.92      0.92      0.92      2500
weighted avg       0.92      0.92      0.92      2500

Confusion matrix support vector machines :
[[1143  105]
 [ 103 1149]]
```

Figure 3: Support vector machine accuracy score report

```
Accuracy score for neural networks :
0.9176
Classification score neural networks :
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      1248
           1       0.92      0.92      0.92      1252

   micro avg       0.92      0.92      0.92      2500
   macro avg       0.92      0.92      0.92      2500
weighted avg       0.92      0.92      0.92      2500

Confusion matrix neural networks  :
[[1147  101]
 [ 105 1147]]
```

Figure 4: Neural networks accuracy score report

Figure 5: Kaggle submission result