Sentiment Analysis of IMDb Movie Reviews

CS7CS4 / CSU44061 Machine Learning – Group Project

School of Computer Science and Statistics, Trinity College Dublin, MT 2021

Group 46
Milan Miletic, miletim@tcd.ie, 21361558
Pradnya Shinde, shindep@tcd.ie, 20309561
Ameya Kale, akale@tcd.ie, 21331310
github.com/Mikki99/IMDB-Sentiment-Analysis

1 Introduction

Sentiment analysis is the process of identifying the emotion (i.e. sentiment) that a piece of text conveys. It is an important part in understanding the customer feedback, as it allows companies to improve their products or services in order to meet their customers' needs. Successfully automating this task is essential for all big companies nowadays. Therefore, we decided to explore several machine learning techniques and their performance in the given problem. The following machine learning models were used: logistic regression, linear support vector classifier, (multinomial) naive Bayes classifier, as well as LSTM, a deep (recurrent) neural network architecture. The task was to successfully classify movie reviews into one of two categories, *positive* or *negative*, depending on their sentiment.

2 Dataset and Features

2.1 Dataset

The data that we used for training our models was scraped from the Internet Movie Database[1]. First of all, a file was obtained from the website, that contained a list of all movies stored on IMDb (around 30 million at the moment) with some additional information about each movie. Table 1 gives an example of what information was stored for a single movie.

Feature	Value
titleId	tt0000001
ordering	6
title	Carmencita
region	US
language	\N
types	imdbDisplay
attributes	\N
isOriginalTitle	0

Table 1: Caption

The relevant information in our case was stored in `titleId` and `region` fields. The `titleId` feature uniquely identifies the movies according to all other feature values and helps us gain access to the reviews for a desired movie. The `region` feature, when fixed to "US"[2], allowed us to filter only movie reviews in English. From the list described above, a random sample of movies (with `region="US"`) was taken, for which we obtained the reviews. Each review consisted of the text and a numerical label representing the rating on the scale of 1-10. In total, three different datasets were created, each containing different number of reviews (roughly 1.5k, 8k, and 16k), as we also wanted to examine how the size of the dataset would impact the performance of our models. The code for this part of the task can be found in `scraper.py` file in the github repository.

2.2 Features

Features represent the attributes which aid to describe the data and predict the classes or labels. Reviews and Ratings were used as features whereas binary Ratings column was added to bifurcate the reviews that have rating greater than 5 as +1 and reviews with rating smaller than 5 as -1. Data preprocessing is a significant step which is used to eliminate noisy and inconsistent data. Redundant data like punctuations, special characters, numbers and stop words were removed from the reviews in order to achieve quality results. Also, each word was converted to lowercase. Furthermore, after stemming and lemmatizing the data, tf-idf vectorization operation was performed on the data to figure out how important are the words are in each review.

Data preprocessing steps are showcased below with a sample review.

1. Sample review:

```
This is a beautifully sculpted slow
paced romantic adorable and lovable
movie.  Halitha shameem deserves all the
appreciation.  Wish to see more movies
coming like this in the future
```

[1] `https://www.imdb.com/`

[2] While there were also other regions, such as "GB", that would direct us to English movie reviews, the number of ones labeled as "US" was substantially larger and also sufficient by itself for our purposes

2. Lowercase conversion:

```
this is a beautifully sculpted slow
paced romantic adorable and lovable
movie.  halitha shameem deserves all the
appreciation.  wish to see more movies
coming like this in the future
```

3. Punctuation Removal:

```
this is a beautifully sculpted slow paced
romantic adorable and lovable movie halitha
shameem deserves all the appreciation wish
to see more movies coming like this in the
future
```

4. Stopwords Removal:

```
beautifully sculpted slow paced romantic
adorable lovable movie halitha shameem
deserves appreciation wish see movies
coming like future
```

5. Stemming:

```
beauti sculpt slow pace romant ador lovabl
movi halitha shameem deserv appreci wish
see movi come like futur
```

6. Lemmatizing:

```
beauti sculpt slow pace romant ador lovabl
movi halitha shameem deserv appreci wish
see movi come like futur
```

For feature f_i:

$$TF - IDF(f_i) = \frac{ff_i}{log(df_i)}$$

ff_i is the feature frequency of f_i and df_i is the document frequency of f_i

Data scraped through Internet Movie Database was split into train-test ratio of 80:20 respectively with the help of `train_test_split()` function of sklearn.

Class imbalancing problem did arrive as the scraped data contained more positive reviews than the negative ones. Hence, some of the positive reviews were dropped in order to balance the data.

Below is the visualization for the actual data and downsampled data

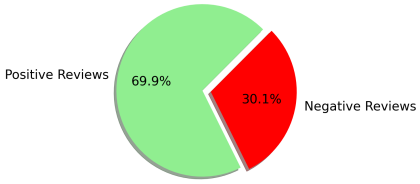

Figure 1: Actual Data.

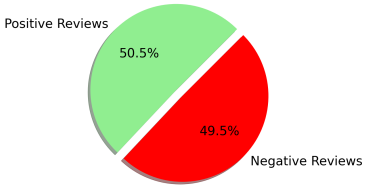

Figure 2: Downsampled Data.

3 Methods and Experiments

3.1 Logistic Regression

3.1.1 Model

Generally used for classification problem a logistic regression is supervised machine learning classification.It uses a binary variable where 1 can be determined as success and 0 is failure. The LogisticRegression() function from the sklearn was used with the IMDb movie data. Depending on the data the decision boundary here can be linear or no-linear. In logistic regression the cost function id known as: **'Sigmoid function'** which can be calculated as

$$\frac{1}{1+e^{-\theta^T x}} = \frac{e^{-\theta^T x}}{e^{-\theta^T x}+1}$$

3.1.2 Penalty

If there are many variables in data a penalty is assigned to logistic model. The solution to it is shrinking the coefficient of variables which are not contributing to zero. The process is also known as **'regularization'**. For adding the penalty L1 and L2 is used depending on the dataset.

3.2 Linear SVC

3.2.1 Model

Another approach in classification, relatively similar to the one described in the previous subsection, are Support Vector Machines (SVMs). In particular, the `LinearSVC()` model from the sklearn library was used (SVC standing for Support Vector Classifier). `LinearSVC` is just one particular implementation of an SVM designed for classification purposes, that uses the linear kernel. The model looks the same as previously, $h(\theta) = sign(\theta^T x)$, however, unlike Logistic Regression, SVMs use hinge loss as the cost function. The hinge loss function is defined as $J(\theta) = max(0, 1 - y\theta^T x)$. In addition, there is a penalty $\theta^T\theta/C$, whose value decides how much weight to give to miss-classified reviews, so the final cost function in our case is the average hinge loss across reviews including the penalty:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} max(0, 1 - y^{(i)}\theta^T x^{(i)}) + \theta^T\theta/C$$

3.2.2 Hyper-parameter tuning

The C parameter in our penalty is a hyper-parameter of our model, thus, its value has to be carefully chosen. Performing 5-fold cross-validation (Figure 3), we decided to use the $C = 0.1$ value for our penalty parameter, as we could see that it gave the best F1-score with a low standard error.[3]

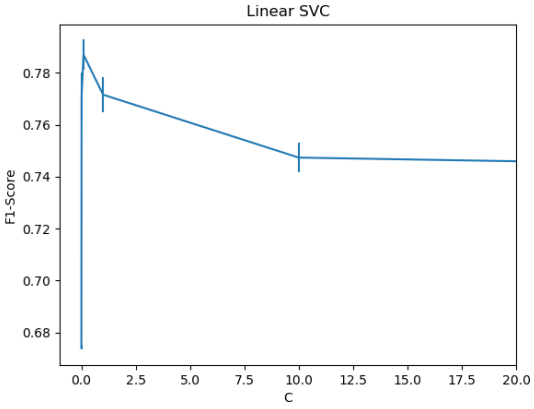

Figure 3: Linear SVC: Choice of C

3.3 Naive Bayes

Naive Bayes is a machine learning algorithm which classifies the attribute based on various features. This algorithm is regarded as naive as it is based on the assumption of feature's conditional independence. It is probabilistic model based on Bayes theorem.

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)}$$

The hypothesis in this algorithm is formed by counting the frequency of different data combinations that are present in the training dataset. Hence, this unique feature lured us to use Naive Bayes for sentiment analysis. It is a generative classifier which can be used for classification purpose on complex and incomplete datasets. Also, it has the power to handle continuous and discrete data.

Multinomial Naive Bayes is one of the popular variants of Naive Bayes used to classify text data based on word vector counting or tf-idf vectorization.

$$\theta_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

θ_{yi} is the probability $P(x_i|y)$ of feature i appearing in a sample belonging to class y

$N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample class y in the training set T_i

$N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y

[3] C values in the range $10^{-4} - 50$ were explored, but x-axis only expands until 20 for easier visualisation, as the performance just kept getting worse from there on

3.4 LSTM

3.4.1 Model

With the growing popularity of deep learning models, it felt too tempting not to try and explore one such architecture. As text is a sequence of tokens, and no matter what we defined those tokens to be (e.g. sentences, words or something else), it is evident that those tokens are not independent (what was previously stated in the text influences what is being stated and what will be stated). Recurrent Neural Networks (RNNs) are a deep learning architecture that can capture this context-dependent nature of the textual data. However, as these dependencies in reality can be quite far apart, plain RNNs, they way they are defined, won't be able to capture them. Thus, a similar architecture, a modification of a plain RNN, called Long Short-Term Memory (LSTM) was introduced to facilitate the problem.

The problem of plain RNNs are the unstable gradients that can occur in gradient descent during backpropagation. The way that LSTMs avoid this is by using a "memory cell" that can store information for a longer time and a set of so-called "gates" that would control what to remember or forget from previous hidden layers. This showed to be helpful for memorizing longer-distance dependencies.

Another thing worth mentioning – LSTMs, as described above, can only capture the dependencies from the "past", but we know that a lot of times in language whatever is being said in a certain moment can also depend on what will be said after (e.g. to predict a missing word from a sentence, we gain valuable information by looking at both words that appear before, as well as after the missing word). There is a further modification of an LSTM, called BiLSTM (Bidirectional LSTM), that can even include this information, by running the inputs in both ways (start to end and end to start). As a result, this was the model we decided to use for this task.

Before describing the exact structure of our architecture, one additional thing has to be noted. For training a deep neural network, our machines weren't powerful enough to train the model using the TF-IDF vectorized data in a reasonable time. Truncating the TF-IDF vectors wasn't a solution, either, as the model was struggling to learn (the original vectors are sparse, so truncated vectors don't provide a lot of useful information). Alternatively, we used the Keras tokenizer to encode the texts. It assigns an integer value to each word, such that the lower the integer the more frequently the word appeared across the texts. Finally, we truncated and padded the encoded texts, so that they are all the same length, as this was required by Keras.

3.4.2 Hyper-parameter tuning

Alright, now we can move on to the details of our LSTM model. Our architecture consisted of the following layers [4]:

- **Embedding** – input_dim*, emb_dim*, input_shape=(max_len,)
- **BiLSTM***[5] – BiLSTM_neurons*

[4] Things marked with an asterisk (*) were the hyper-parameters of the model and, thus, carefully selected

[5] The number of BiLSTM layers was also explored as a hyper-parameter

- **Dense** – dense_neurons*, activation='relu'
- **Dropout** – dropout_rate*
- **Dense** – units=1, activation='sigmoid'

Table 2 gives a description of what each hyper-parameter represents, the range of values that were explored and, finally, the selected best values:

Hyper-Parameter	Meaning	Value range	Best Value
input_dim	size of the vocabulary	5k−15k	10k
emb_dim	dimension of the embedding	32 – 256	64
num-_BiLSTM-_layers	number of BiLSTM layers to be used	1 – 4	1
BiLSTM-_neurons	number of neurons in the output of the BiLSTM layer	2 – 64	22
dense-_neurons	number of neurons in the output of the Dense layer	32 – 256	96
dropout_rate	fraction of units to drop from the previous layer	0 – 0.5	0.5

Table 2: LSTM Hyper-parameters

The described hyper-parameters were tuned using the Keras tuner and a random search algorithm, which explored random combinations of hyper-parameter values from the given range. Therefore, not all combinations were explored – in deep NNs, that is often not computationally feasible – but it nonetheless allowed to make more informed choices compared to random initialization.

For the output layer, we used the sigmoid activation function which outputs a value in the range 0-1 that represents the probability of the review belonging to the positive class. That is why the number of output units is set to 1. Based on this number, we will classify the review as either positive or negative by setting the cut-off value at 0.5.

The loss function that was used was binary cross-entropy, which returns a normalized sum of log probabilities of a review being of category y:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i \cdot log(p(y^i)) + (1-y^i)\cdot log(1-p(y^i))\right]$$

It is also worth mentioning that the way this loss function is defined, it expects the value of category y to be either 0 or 1 (not -1 or 1), so this was another slight modification that had to be made in our data.

Instead of also considering the number of epochs as a hyper-parameter and exploring its values, we used an early stopping mechanism, monitoring the accuracy on validation data and stopping the training if the val_accuracy hasn't improved for three consecutive epochs. The number of batches was fixed to 64 and the optimizer used was Adam. As is standard in LSTMs, Keras uses stochastic gradient descent and backpropagation for training the model.

4 Results and Discussion

As mentioned in the beginning of the report, one of our ideas was to explore how our models would behave when trained on different sized data. That is the reason why we created three data sets, with roughly 1.5k, 8k, and 16k rated movie reviews. Figure 4 shows the ROC curve plot (true positive rate against false positive rate) for the Naive Bayes classifier trained on the three data sets.

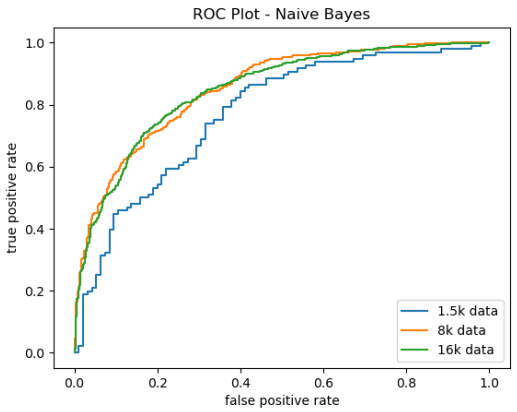

Figure 4: Naive Bayes classifier trained on different sized data

The improvement in the performance is evident when augmenting the data from 1.5k to 8k reviews, and arguably there is a slight further improvement when using 15k data set. Similar results were observed with the other models, thus additional plots were omitted. Thus, we decided to use the 16k data set for training our models and, from now on, all the results reported refer to models trained on this large data set.

Before moving onto the results, let us mention two models used as baseline classifiers for comparison:

- `DummyClassifier` – classifier from sklearn library; with `strategy` parameter set to "most-frequent"; as a result, this classifier always predicts the most frequent category in our data.
- `TextBlob` – a library in Python providing API useful for many NLP tasks, including sentiment analysis; the classifier was used as a baseline as we expected it to perform worse than our tuned models, as it was trained on different data

Finally, let us present the ROC plot for all of our models described above, as well as for the baseline classifiers (Figure 5).

Additionally, table 3 gives F1-scores for the mentioned models:

The appendix also provides confusion matrix plots for all the models. We can see from the provided metrics that logistic regression and linear SVC models demonstrated the best performance. Naive Bayes and LSTM also did a solid job. LSTM

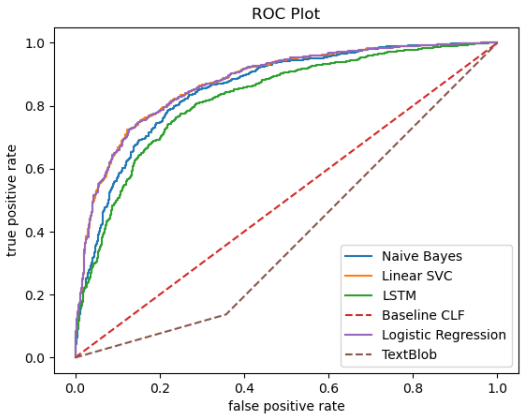

Figure 5: ROC plot for all the models

Model	F1-score
Logistic Regression	0.80
LinearSVC	0.80
Naive Bayes	0.78
LSTM	0.75
Dummy CLF	0.67
TextBlob	0.19

Table 3: Model evaluation scores

can potentially show an even better performance, but would require even larger data (currently it shows solid performance on the test set, but overfits the train set). However, obtaining the 16k data set took around two hours on our machines, thus we didn't explore this further. Still, we managed to obtain performance significantly better than that of baseline classifiers.

5 Summary

In this report, we implemented four machine learning algorithms for performing sentiment analysis on IMDb movie reviews dataset. Based on 16000 reviews, Logistic Regression and Linear SVC performed better than Naive-Bayes and LSTM. However, LSTM is the most superior algorithm if considered from the theory perspective. Furthermore, LSTM requires larger dataset, and its output model requires tuning for obtaining better accuracy. Based on our results section, we used TextBlob and Random baseline classifier for comparison purpose. Future work includes evaluating the above machine learning algorithms on a much larger dataset.

6 Contributions

- **Milan Miletic** – scraped the original and augmented data sets; down-sampled the data; trained and tuned Linear SVC model; trained and tuned LSTM model, baseline classifier, confusion matrix plots ; wrote sections 1, 2.1, 3.2, 3.4
- **Pradnya Shinde** – Trained and tested the Logistic regression model; used the textBlob for performing sentimental analysis on IMDB movie data.Finally used **F1-score** to compare all the models which we used. wrote sections 3.1 and 4.
- **Ameya Kale** – Preprocessed the dataset, tf-idf vectorization, trained and tested Naive-Bayes classifier, Pie chart and Wordcloud plots ; wrote section 2.2, 3.3, 5

7 Appendix

A Project modification

A slight modification of the original project proposal was made after the feedback from prof. Leith. Initially, we wanted to fetch the movie reviews from Reddit, but after professor's feedback we realized that would present a problem, as there would be no clear way to label the reviews, as they didn't come with ratings or any other helpful labels. Therefore, we obtained movie reviews from IMDb, as described in Section 2.1.

B Confusion Matrix Plots

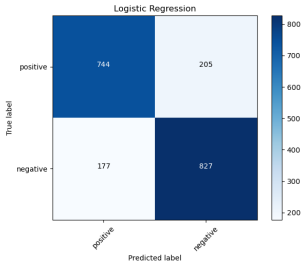

Figure 6: Logistic Regression

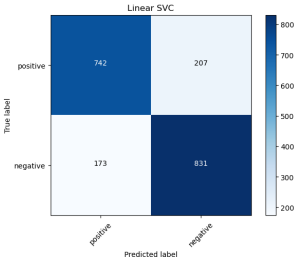

Figure 7: Linear SVC

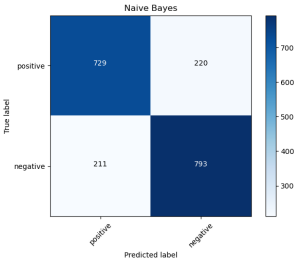

Figure 8: Naive Bayes

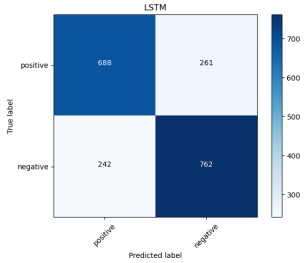

Figure 9: LSTM

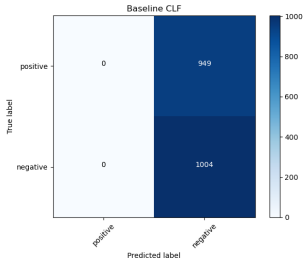

Figure 10: Dummy Classifier

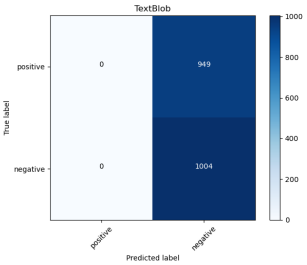

Figure 11: Text Blob