REPORT ON THE WORK OF SENTIMENT ANALYSIS

Introduction

NLP stands for Natural Language Processing which is the task of mining the text and find out meaningful insights like Sentiments, Named Entity, Topics of Discussion and even Summary of the text.
By definition, sentiment analysis is the use of text analysis, computational linguistics, or Natural Language processing (NLP) in order to get semantic quantification of the studied information. Sentiment analysis aims to indicate the opinion of a specific text (a tweet, or a product review)

This work applies sentiment analysis over a dataset of English movies reviews (IMDB dataset) using deep learning techniques in order to classify this dataset files into positive and negative reviews. The notebook given to us uses the following techniques for classification: Logistic Regression, SVM, and Naïve Bayes.

We have tried to improve the performance of the first two models and then came up with the following models: Recurrent Neural Networks like LSTM, One-Dimensional Convolutional Neural Network, CNN_LSTM with dropout.

Beyond the provided baseline, what other approaches did you decide to pick?

We have plotted some figs in order to understand the distribution of the data and how they are dispatched (word cloud) and picked up one review to show step by step why we will have to do on the whole dataset. (Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance)

In order to increase the accuracy of the models given in the baseline with have chosen to clean up the data:

Normally any NLP task involves following text cleaning techniques

1. **Removal of HTML contents** like "< br>".
2. **Removal of punctuations**, special characters like '\'.
3. **Removal of stop words** which do not offer much insight.
4. **Stemming/Lemmatization** to bring back multiple forms of same word to their common root like 'coming', 'comes' into 'come'.
5. **Vectorization** - Encode the numeric values once you have cleaned it.
6. **Fit the data to the ML model.**

We have added:

- **Stemming**: Stemming is considered to be the cruder/brute-force approach to normalization (although this doesn't necessarily mean that it will perform worse). There are several algorithms, but in general they all use basic rules to chop off the ends of words.

- **Lemmatization**: Lemmatization works by identifying the part-of-speech of a given word and then applying more complex rules to transform the word into its true root.

  But lemmatization is better than stemming method because it preserves spelling correction (lexicon models like **Textblob** do the same job as lemmatization: spelling correction, tokenization...)

  In order to vectorize the text we have added Keras Tokenizer(embedding) to CounVectorizer and TfidfVectorizer. In some cases, Counvectorizer is better than TfidfVectorizer, we will use the both method and see the accuracy.

  TfidfVectorizer and CountVectorizer are both Bag of Words models (A Bag of Words Model is used to convert text document to numerical vectors or bag of words)

How did you develop a training/validation and test set?

We have chosen 75% of the dataset to train our models and 25% to test our models. So, we have 35000 data for the training set and 12500 for the test set. But my computer takes too much time to perform some tack.

What different models did you try? What were the observed results?

1. **Logistic Regression**
   Accuracy with tfidf: 0.88472
   Accuracy with count_vect: 0.89872

2. **LinearSVM**
   Accuracy with tfidf: 0.90272
   Accuracy with count_vect: 0.87736

3. **RNN (simple LSTM)**
   Accuracy: 0.5025

4. **One Dimentional CNN**
   Accuracy:0.86

5. **CNN_LSTM with droupout.**
   Accuracy: 0.88

What performance metric did you pick to measure how well your model did. Why?

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. Accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Accuracy = TP+TN/TP+FP+FN+TN

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all reviews that labelled as 'positive', how many actually are 'positive'? High precision relates to the low false positive rate.

Precision = TP/TP+FP

Recall is the ratio of correctly predicted positive observations to the all observations in actual class. The question recall answers is: Of all the reviews that truly labelled 'positive', how many did we label?

Recall = TP/TP+FN

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Confusion matrix : The results of a confusion matrix are classified into four main categories : true positives, true negatives, false positives and false negatives. The true positives or TP (true positive) indicate the cases where the predictions and the actual values are indeed positive.

For the different models you picked, what were their relative strengths and weaknesses?

linearSVM has best performance than non-linear SVM (which uses sGDClassifier).

Linear SVM	**Non-Linear SVM**
It can be easily separated with a linear line.	It cannot be easily separated with a linear line.
Data is classified with the help of hyperplane.	We use Kernels to make non-separable data into separable data.
Data can be easily classified by drawing a straight line.	We map data into high dimensional space to classify.

we can notice that LinearSVM has the best performance with TfidfVectorization

CNN_LSTM is better than simple LSTM but LSTM is faster.
We have noticed that vactorization with Tfidf is faster than vectorization with Counvectorizer

Conclusion

we used 5 models of algorithms LR, LinearSVM, LSTM, CNN, CNN_LSTM and it appears that of the first three groups LinearSVM has the best performance.

We cannot compare the last two models with the first three because they do not come from the same distributions. for what it is about the last two models one notes that CNN_LSTM is more powerful.