Sentiment Classification on Product Reviews

FIT5149 S2 2019 – Assignment 2

Pouria Ebrahimnezhad – 30035678
Hristina Todorovska – 29193869
Asghar Mustafa- 28905644

October, 2019

Table of Contents

1 Introduction

Sentiment analysis also known as opinion mining is a subfield within Natural Language Processing (NLP) that builds machine learning algorithms to classify a text according to the sentimental polarities of opinions it contains, e.g., positive or negative. In recent years, sentiment analysis has become a topic of great interest and development in both academics and industry. Analyzing the sentiment of texts could benefit, for example, customer services, product analytics, market research etc. In this data analysis challenge, we are interested in developing such an automatic sentiment classification system that relies on machine learning techniques to learn from a large set of product reviews provided by Yelp. The levels of polarity of opinion we consider include strong negative, weak negative, neutral, weak positive, and strong positive. So, the challenge for us was to develop a sentiment classifier that can assign a large set of product reviews to the five levels of polarity of opinion as accurately as possible, given a small amount of labeled reviews and a large amount of unlabeled reviews.

Within our team of three we split the task into three main subtasks, Pre-processing the data and feature extraction, preliminary analysis of the data and experiment with models that are known to perform well in relation to sentiment analysis. Comparing and fine tuning the models and testing the results on the final test set. We held meetings to check progress on the tasks and due to learning curve, we all focused on the same subtasks at a time and tried to help with ideas as well as the code.

2 Pre-processing and feature generation

Since the task was sentiment analysis, we decided to make some modifications to the usual method of text preprocessing that is usually done for document topic classification, these included coming up with our own custom list of stop words as well as finding a list of common contractions that could help replace the usual apostrophe at the end of words such as don't or shouldn't, in this way we could make sure the generated list of features includes some of the words that are important in terms of sentiment analysis. We then made a custom function to apply the preprocessing required on each document. This function includes removing url, numbers, punctuation, replacing short words, removing white spaces, tokenization using the tokenizer from nltk and lemmatization using lemma. This resulted in our final tokenized version of each document.

We then compute the term frequency and Inverse document frequency (TFIDF) of the words which we then used to create a data frame of our features for modeling stage, after multiple iterations and exploratory data analysis specially the PCA which we will discuss in next section we decided we are using a maximum number of features of 10000 for our modelling and analysis. We used the tfidfvectorizer method of sklearn library and used ngrams ranging from 1 to 3.

2.1 Principal Component Analysis

Here we try to perform Principal Component Analysis on the features with an aim to understand how much of the variance of our data is explained by what number of components of our features. in order to do this, we scale the data and apply the analysis to both scaled and non-scaled data just to observe the differences and try to demonstrate how much of the variance is explained by how many components which can give us some information of what models may help us in this task.

Below (Figure 1 and 2) we have shown the result of the analysis using PCA and standard scaler form sklearn library. Looking at the non-scale version of the PCA we can see that there is not a lot of separation of the classes using these two components meaning the variance explained by the first two components is not significant enough

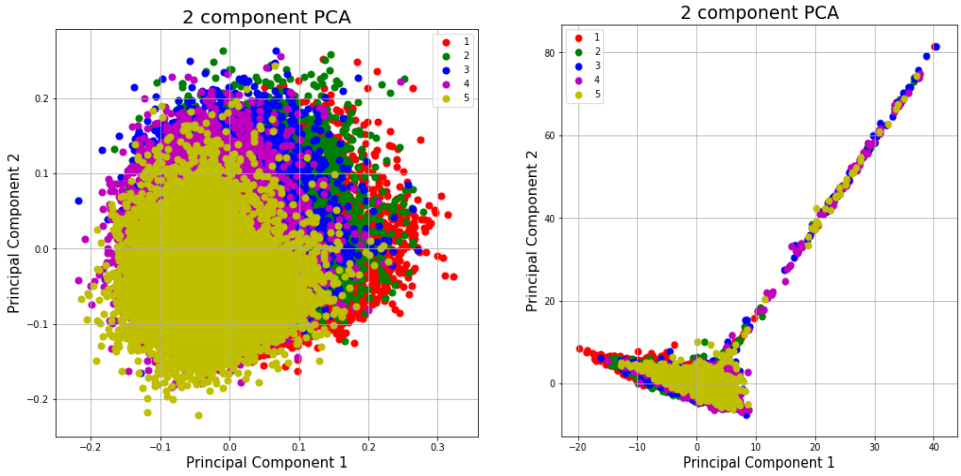

Figure 1 – 2 component PCA (non-scaled vs scaled)

looking at the scaled version of the PCA again we could not see a clear differentiation between the classes. We now demonstrate the graph indicating the cumulative explained variance with respect to the first 1000 features in our data we could see that only around 0.25 of the variances is explained by this many features indicating that our choice of 10000 features could be a good choice.

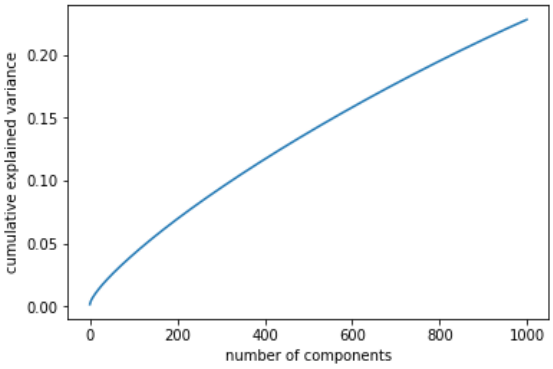

Figure 3 – cumulative explained variance

3 Models

In this section we discuss the modelling approach that we took and how we implemented some of the chosen models and experimented with to investigate which one would give us the best accuracy

3.1 Conventional models

We first experimented with implementing a few well-known models that can help in classification tasks. This included implementing a Multinomial Naive Bayes, Logistic Regression and Linear SVM which in multiple publications have been well referenced for similar tasks including text classification and sentiment analysis. For details of the implementation please refer to the notebook provided with this report. Table 1 shows how these models performed on using the labeled data and a split of 20% for testing and 80% for training:

Models	Accuracy on 10k test data
Multinomial-NB	57%
Logistic Regression	60%
Linear SVC	56%

Table 1 – Models Test Accuracy

We can see that amongst these model Linear regression was consistently performing better that the other two.

3.2 Neural Network

As we were aiming for further improvements to the accuracy of our model we decided to study and implement a Neural Network model to see how it would perform on our labeled test data, we chose the implementation by Keras library and we tried constructing various layers and number of Neurons as well as different optimizer functions to compile the model such as Adam and rmsprop which according to publications are well known for providing good results but each time we used a softmax activation function in the last layer along with 6 nodes for 5 classification groups.

So basically, after testing various implementations the best implementation for our data included a 1 hidden layer with 31 neurons and input dimension of our 10k features which then fed in to the output layer with 5 classification output nodes. This model when testing showed accuracy levels of up to 65%

on our test split data and we managed to reach accuracy scores of 61% on the Kaggle test data. We also applied Cross validation testing with 10 folds to make sure the testing results we are getting is statistically significant the details of the implementation are included in the notebook. We also implemented a Convolutional Neural network to see if it would perform better than the simple neural network but the results were not as significant of the neural network.

3.3 Deep learning with ULMFIT

To further increase the performance of the classifier, a deep learning recurrent Neural Network was set up using the FAST AI library. This model uses ULMFIT (Universal Language Model Fine-Tuning) transfer learning technique to Natural Language Processing Applications. From the previous analysis, it was found out that YALP reviews dataset contains text from multiple languages. The main reason for choosing this model was its inherent capability to perform well on text containing multi language data. As per the fast.ai, this sort of model outperforms on multilingual data with low training data compared to high training data applied to algorithms like BERT, which do not perform well with dissimilar data.

This approach thus uses pre trained language models (AWD LSTM model in this case) to preprocess the data and train a language modeler on the input data set. This is then used by recurrent neural network text classifier to train the classifier. This model when tested on a validation set resulted in an increase in accuracy of up to 65% in just 6 iterations (epochs). These results were also reproduced when this was tested on the Kaggle data set.

---

4 Semi-supervised learning

In this section an attempt was made to appropriately label the 600k unlabeled data and enhance the classifier performance. The following algorithm was used to enhance the performance of the classifier.

1) Use the current model to get probability vectors for each data point (document).
2) Label data point for only the documents with maximum class probability greater than 0.9
3) Recalculate the model and probability vector using the previous and added data points and their corresponding labels.

One thing that was considered was to ensure that the new proportions of all classes are equally added. To do that samples equal to minimum count with thresholds above 0.9 were added from each class.

4.1 Adding unlabeled data to Logistic Regression

When the above labeling approach was tested with the Logistic regression model it was observed that accuracies actually went down or (were at best static at a given point). The figure below shows the accuracy after adding around 150k of 600k unlabeled documents. It shows a clear downward trend with accuracy decreasing by around 1% (from 59.5% to 58.5%). This can be potentially attributed to

dis similarity of the text between labeled and unlabeled data and model's suitability to the unlabeled data since the model was only trained on the labeled data initially.

Figure 4 - Accuracy with unlabeled data (logistic regression)

4.2 Adding unlabeled data to Neural Network

For this section of our experiment with self-learning we decided to slightly change the method in a way explained below.

We got similar results when we apply Keras Neural Network, the results show that the accuracy is decreasing. In the first iteration, we have used only labelled data and the accuracy starts with 61.31% in each iteration we have trained 10000 points from the unlabeled data, we notice that in each iteration the accuracy is dropping where in the 10th iteration the accuracy is 60.35%. In all these 10 iterations the accuracy drop by 0.96%. This method did not show improvement of our model.

5 Conclusion

From our analysis and experiments we can clearly see that the conventional model of choice given our data would be logistic regression and this was further improved by Keras implementation of Neural network. However, given the multi lingual nature of the data provided by YALP it is clear that the ULMFIT by far outperforms the other models used in expense of computational resource and time, our experiment showed that we required in excess of 180 minutes to train this model were as the simple Neural network which also resulted in an acceptable level of performance only took on average 2 minutes to train. Given this task was performed in a competition environment which favors accuracy our model of choice would be the recurrent neural network using ULMFIT. The interesting finding for us was that in contradiction to our expectations we were not able to improve the performance of the models using the unlabeled data and self-learning approach. Now this can be a good topic for us to further look into post the completion of this assignment.

References

Aditya, C.(2019). Using FastAI's ULMFiT to make a state-of-the-art multi-label text classifier. https://medium.com/technonerds/using-fastais-ulmfit-to-make-a-state-of-the-art-multi-label-text-classifier-bf54e2943e83

Jake, V. P. (2016) In Depth: Principal Component Analysis. https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

Javaid, N.(2019).Machine Learning — Text Classification, Language Modelling using fast.ai.https://towardsdatascience.com/machine-learning-text-classification-language-modelling-using-fast-ai-b1b334f2872d

Kavita, G. (2018). Tutorial: Extracting Keywords with TF-IDF and Python's Scikit-Learn.https://kavita-ganesan.com/extracting-keywords-from-text-tfidf/#.XbBFm-gzZPY

Prajwal, S.(2019). Sentiment analysis for text with Deep Learning. https://towardsdatascience.com/sentiment-analysis-for-text-with-deep-learning-2f0a0c6472b5

Prateek, J. (2018). Tutorial on Text Classification (NLP) using ULMFiT and fastai Library in Python. https://www.analyticsvidhya.com/blog/2018/11/tutorial-text-classification-ulmfit-fastai-library/

Sebastian, R. & Julian, E.(2019). Efficient multi-lingual language model fine-tuning. http://nlp.fast.ai/

Usman, M. (2019). Python for NLP: Movie Sentiment Analysis using Deep Learning in Keras. https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras/