Sentiment analysis of Amazon reviews
Comparing three word Embedding methods
word-document matrix, word2vec, and Transformers

Shaghayegh Safar
Shaghayegh.afar@aalto.fi
Student number: 613183

Final project for the course
ELEC-E5550-Statistical Natural Language Processing
April 2020

Table of Contents

1. Introduction

Sentiment analysis refers to the use of natural language processing (NLP) to systematically study affective states and subjective information about a textual input. Given an input text, we might be interested to know polarity (positive, negative, neutral), feelings, emotions (sad, angry, happy), or intentions (interested or not interested) behind the text. This information can be very important as it can help a webpage to better investigate the comments of its posts or an online shopping site to better understand whether its customers like a product or not. In other words, sentiment analysis allows us to extract subjective sentiment from complex textual inputs.

Being able to accurately summarize a textual input into subjective information is very important as there is a huge amount of unstructured textual data generated everyday and it would be very expensive to hire humans to label the data. In this report we are interested in machine learning solutions for sentiment analysis. In other words, we assume that we have a fixed set of textual data with known sentiments, which can be used to train a machine learning method and then use the trained method on some new textual data to predict the sentiment and validate the method. The core challenge for any method to use textual data is how the textual data is represented. This representation, known as embedding, projects the text (or its components such as words) to a dimension where similar texts (words) are close to each other. In this report we are interested in different methods for word embeddings for the task of sentiment analysis.

A good sentiment analysis system needs to understand the unstructured textual input to be able to correctly classify it. In the next section we look at three general commonly used approaches for word embedding. After that in Section 3, we introduce the problem investigated in this report: predicting review score for Amazon reviews. Section 4 reports the performance of the methods introduced on the training data and Section 5 provides the lesson learned from the studies.

2. Methods for embedding

Given a review text such as “Alexa was fine, but I will not use it again” how can a machine predict the review score? Let us assume that we have hundreds of similar reviews with known sentiments, here the review score. In this section we study how to train a model to predict review scores.

The first step for any method is to represent the textual input as a vector of numbers, which are understandable for machines. There are many different methods for word or sentence embedding but in this report, we consider three general approaches, from old to new:

Word-document matrix. In this method we create a matrix where the number of rows is the number of reviews and the number of columns is the total number of unique words found in the reviews. Then we fill element (i,j) of that matrix with the number of times that the jth unique word appeared in the ith document. This representation is also known as bag-of-words. The ith row of this matrix will be a vector representation (embedding) for the ith review.

After having this representation, we can use standard supervised learning methods to solve the sentiment classification problem. In this project I used logistic regression, as a baseline linear model, and random forest, as a highly non-linear and robust model.

Word2vec. Wrod2vec is a word embedding model that converts a given word to a vector of real values [1]. To generate this embedding, we usually need a huge corpus of textual data. Then we can train a neural network to predict words against other words that neighbor them in the input corpus. The resulting embedding for each word can then be used as a vector representation of that word. There are many pre-trained word2vec models available. In this project we use an embedding trained on 100 billion words from a Google News dataset using Genism library. In this dataset each word is representing as a vector of length 300. After tokenizing a review, we check if each word (token) is in the Google News dataset, and if it is, we keep the vector representing that word. If a word is not in the Google vocabulary or in the other words it is out of vocabulary, we simply remove it.

After extracting the word2vec representation for all the words in the review, we need to somehow combine them to create one feature representation for the review. There are different approaches to combine words of a review. We use weighting average to have a vector of length 300 for each review. I used term frequency-inverse document frequency (tf-idf) weighting scheme which is a good weighting method. If a word is frequent in one document but is rare in other documents, then it sems to carry lot of information specifically for that document, therefore, that word should get more weight. We perform this weighting averaging of worf2vec vectors for all reviews. Now that the review matrix is ready, like the previous method, we use logistic regression and random forest on the training set as the classification models.

Working with the whole sentence. Recent advances in neural network for NLP has resulted on training huge models that receive the whole sentence or paragraph as the input. The Transformers is a novel deep neural network architecture that aims to solve a problem of converting a sequence of one type to a sequence of text of another type (sequence-to-sequence task) while handling long-range dependencies in input texts. These models have been trained on huge datasets and can be use as pre-trained model for the starting point to other NLP applications [2]. In particular, these pre-trained models can be just fine-tuned by hundreds of training data for our task[1] [3]. This fine-tuning is done as follows: the input sentence is fed to the pre-trained model (with weights fixed), then the value of some output layer of the model observed. Roughly speaking, these values have an embedding of the input sentence and we can train another model (with hundreds of data) on top of those (fine-tuning). Transformers have shown to be quite successful, and current state-of-the-art in many NLP applications [2].

[1] In this project, I first planned to use Long short-term memory (LSTM) models for sentence level sentiment analysis but a google search showed that Transformers are the current state-of-the-art.

3. Experiments

The task is to classify the review score given the text of the review. I have used one of the data sets in Kaggle [4]. The dataset consists of 3150 Amazon reviews of Alexa products such as Alexa Echo. The dataset contains reviews texts, star rating, variant, date of review, and feedback. I will use only two columns of this dataset, the reviews texts, and the rating star which can be 1, 2, 3, 4, and 5. It is important to consider that the dataset is not balanced as the reviews with score 5 are more common than other scores. The class distribution of the dataset is shown in Table 1.

Table 1: Review score distribution.

class	***Size of the class***
1	161
2	96
3	152
4	455
5	2286

Every text needs preprocessing before working with them, in order to have a clean data and increase the performance of the methods. Therefore, for the models that work at a word level (word2vec and word-document models) we perform the following analysis[2]. First thing to do is removing the punctuations and special characters since they are not informative. Then we lower case all the text so that there will be no difference between "Echo" and "echo". Next step would be removing the stop words. Stop words are words that happens in the corpora many times (have very high frequency than other words) and do not have good amount of information. For example, "the" is a word that occurs a lot, but its weight is not heavy. For this matter, there is a built-in list of words in Python's nltk package [5] that incudes these words. Therefore, we only need to check if any word in our corpus is in this list and needs to be removed. After that we tokenize all the reviews, and we keep a set of all the tokens as our vocabulary [3]. At the end there were 4402 unique tokens in the data which also specifies the dimension of the word-document matrix.

It is interesting to see top 50 words that were used in the reviews by users in Figure 1.

[2] There is no need to perform these steps for Transformers as it has its own pre-trained tokenizer which receives the unprocessed text.

[3] Note that stemming and lemmatization are also important in preprocessing the data. However, since the data in this project is not very big removing more words by these two methods is not recommended.

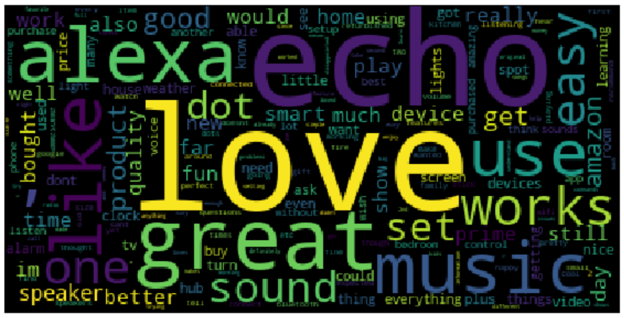

Figure 1: top 50 words used by users in reviews

We use 80% of the data as training and the remaining as test data. The training and test division is the same for all the methods we compare. We use accuracy, average precision, average recall, and average F1 score based on confusion matrix as the performance measures to compare different methods. The average is computed as an unweighted average of the five classes.

To summarize, we test the following methods:

- Logistic regression model with reviews represented as word-document matrix.
- Random forest model with reviews represented as word-document matrix.
- Logistic regression model with reviews represented as feature generated from word2vec average.
- Random forest model with reviews represented as feature generated from word2vec average.
- A fine-tuned transformers model using BERT pre-trained model as its base [3].

4. Results

The summary of the results are shown in Table 2. As it is evident, the Transformers have achieved the highest performance measures.

Table 2: summary of the performance measures on test data for different models. Highest performances are shown in bold font.

Method	***classifier***	***Accuracy***	***Precision***	***Recall***	***F1 Score***
Word-document matrix	*Logistic regression*	0.78	0.69	0.46	0.53
Word-document matrix	*Random Forest*	0.81	0.72	0.48	0.53
Word2Vec (weighted tf-idf average)	*Logistic regression*	0.72	0.43	0.27	0.36
Word2Vec (weighted tf-idf average)	*Random Forest*	0.79	**0.86**	0.43	0.51
Transformers with BERT pretrained model		**0.82**	0.65	**0.52**	**0.65**

Looking deeper at the results it is evident that the average precision values are not high (compared to accuracy). This is indeed because of the unbalances of the data. For example for logistic regression in word-document model the confusion matrix and individual class precision and recall are shown in Table 3. The results show the unbalances of the data and also the fact that for close scores (for example 4 and 5) it is easier for the model to make mistakes (compare to errors for prediction class 5 to 1 or 2).

Table 3: Confusion matrix for model word-document matrix, logistic regression method

predicted / ***Actual***	***1***	***2***	***3***	***4***	***5***
1	13	2	1	0	17
2	0	5	4	4	8
3	1	0	15	7	16
4	4	0	0	29	57
5	2	0	1	13	431

5. Conclusions

Considering the huge amount of textual data generated every day, automatically predicting the sentiments behind the text can have a large value. The task, like any other tasks that work with textual inputs, is difficult. This is because textual data are generally difficult to work with (compared to numerical data). However, recent advances in natural language processing have made text representations straightforward and accessible. Rather than representing a review as a sparse vector that counts the number of appearances of words, we can now directly convert words to vectors using pre-trained models. Out of the models that we tried, Transformers (which are also the newest methods in NLP) achieved the best performance. This is because these models use a huge number of prior information as they have been trained on huge corpuses. Thanks to available libraries, it is very easy to download these pre-trained models and just fine-tune them on the task at hand.

6. Acknowledgments

I would like to thank Pedram Daee for his consultation in this project.

7. References

1. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
2. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv, abs/1910.03771*.
3. Simple Transformers, GitHub repository: https://github.com/ThilinaRajapakse/simpletransformers

8. Kaggle dataset: https://www.kaggle.com/sid321axn/amazon-alexa-reviews
9. Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. *O'Reilly Media Inc.*

8. Appendix

The code of the project is attached to the report in the zip file. You can also find it in my Github using the address:

https://github.com/Shaghas1/Sentiment_Analysis_Transformers