Natural Language Processing: Sentiment Analysis

– Machine Learning From Theory to Practise – Project –

Johann FAOUZI : johann.faouzi@ensae-paristech.fr
Mélanie FINAS : melanie.finas@ensae-paristech.fr
Hicham JANATI : hicham.janati@ensae-paristech.fr
Peter MARTIGNY : peter.martigny@ensae-paristech.fr

Tuesday, January 31th 2017

Abstract

The purpose of this research is to review and implement different methods of sentiment analysis. Sentiment analysis refers to the use of Natural Language Processing (NLP) in order to identify and extract subjective information, such as opinions, in source materials. Our methods of sentiment analysis have been restricted to the tasks of predicting a tri-class outcome (positive, negative, and neutral) using machine learning algorithms. To achieve this goal, this paper involves the implementation of two different methods to predict the sentiment of a review : bag of words models and word embedding models with Word2Vec. After each of this methods, we use three usual classifiers to make the final predictions: Logistic Regression, Random Forest and Naive Bayes.

1 Presentation of the problem

1.1 Natural Language Processing

NLP is a field at the intersection of computer science, artificial intelligence and linguistics. The aim of NLP is to be able to understand natural language in order to perform some tasks such as translation, named entity recognition, sentiment analysis, question answering... In this paper, we focus on NLP applied to Sentiment Analysis.

Sentiment analysis has raised a lot of interest recently, especially in the media industry. Its main goal is to identify opinions that are expressed in textual comments. For example, a fashion clothing company could use the textual reviews of its products online to know what the consumers think of them, in order to build better products, adapted to consumers' needs. Another example lies within presidential elections. Today, a candidate needs to take social media into account, as it is a huge source of textual information. Because these information are produced as textual comments, sentiment analysis is very useful to understand what the citizens think of different candidates and their opinions on different societal issues.

Hence, given the growing number of sources of textual opinions being published on the internet, especially in social media, sentiment analysis has become a major industrial task using machine learning technics. Social media like Twitter have even produced their own api to use sentiment analysis (e.g. tweepy).

In this paper, we restrict ourselves to the task of predicting a tri-class outcome (positive, negative or neutral opinion).

1.2 Several paradigms for Sentiment Analysis

When deciding whether a reviewer will like or not a product, there are mainly two approaches to tackle the task of sentiment analysis:

- **Lexicon based approach**
- **Machine Learning methods**

As [LZ12] explains, lexicon based approaches are based on dictionaries of words manually annotated according to their sentiment polarity. Hence, based on a

list of positive words and a list of negative words, the task is to sum all word-scores in the sentence to assign it a sentiment polarity. These methods are said to perform very well on specific domains. However, it is known to be poorly generalizable to other domains. This approach suffers from overfitting.

The second approach focuses on using machine learning methods to predict the polarity of a text. Using regular machine learning algorithms means that we are able to represent the texts as features, ready to be fed to a learning algorithm.

This paper explores two main methods to represent text as vectors, in order to carry out a learning algorithm.

- **Bag of Words Model**
- **Word embedding with Word2Vec**

The following explains these methods.

2 Strategies used to create features

While the final sentiment prediction will be eventually made with classification algorithms like logistic regression or tree methods, the most important task here is to design the right features, those that the classification algorithms will be able to use to separate the elements from different classes.

2.1 The Bag of Words Model

One of the first attempts to featurize texts dates back to the $50's$ with [Har54]. In his *bag of words model*, the algorithm first learns a vocabulary from all the provided textual data. Then, each text is represented by computing the relative frequencies of its words, both within the document itself and the whole set of documents. This methodology is known to give good results, however it suffers from an important drawback: by considering only the frequencies of the words, we lose the order of the words, which are very informative about the structure of texts and the relations existing between words.

In this paper, we use two ways of measuring the importance of a word in a text:

- By counting the number of its appearance in the text (term-frequency)
- By counting both its term-frequency and its rarity between the different texts (Inverse Term Frequency, IDF). Indeed, we want to take into account the fact that a word that appear often in all documents may not be very discriminant.

The first strategy has the merit to be very simple. If we have a vocabulary V, each text will be represented by a vector of size $|V|$, each coordinate being the number of occurrences of each word in the text. Other approaches, not explored in this paper, can be to use frequencies normalized by the maximum frequency in the text, in order to take the length of the text into account.

However, suppose that a word is often used in a particular text, but it is also regularly used in all other texts. Hence this word will not be very useful if our goal is to classify texts, because it will affect equally all different texts. On the contrary, words that are rare among texts are likely to be very relevant. Hence, we derive a second strategy which takes this into account. In this strategy, we want to define a document frequency, which is high for rare terms and low for frequent words among texts. Let us define d the number of document in which the word appears, and N the number of document, then we can define as $log(\frac{N}{d})$ the document frequency. Indeed, if the word appear in all documents, it will have a weight of 0, and the maximum weight is obtained for a word appearing in only one document.

In order to take both effects into account, which is the fact that a word which appear often in a text may be an important word for this text, and that relevant words should not appear in too many texts, we define the term-frequency/inverse-document-frequency coefficient, which is the product of both terms:

$$w = TF. \log\left(\frac{N}{d}\right)$$

with TF the term frequency of the word in the text.

2.2 Word Embeddings

In the recent years, there have been a large excitement around the task of embedding words into low-dimension spaces, while still preserving contextual information.
Word2Vec is the model that has become very popular. Based on skip-gram and continuous bag of words models, introduced in [MSC+13], the goal is to design a deep neural network that learns jointly a usual task (like classification) and the word vectors. The embeddings that result from this training part provide surprisingly high performance scores on tasks related to semantic and syntactic metrics. Along with another word embedding algorithm, *Glove*, *Word2Vec* is the state of the art for lots of NLP tasks, including *Sentiment Analysis*.

Once these vectors are learned, there is still a need for a strategy to represent the whole text as a vector. In this paper, we explore 2 methods using *Word2Vec* word vectors to build text vectors:

- **Vector averaging**: we take as text vector the average of its word vectors

- **Clustering**: as we built a natural distance between our word vectors, there is a good chance that groups of words will be significant to predict sentiments. Hence, we use the K-Means algorithm to groups the words into clusters. Then, we consider the clusters we created as features. Hence, each text is represented as a vector of size the number of clusters, and for each cluster we assign the number of words in this cluster.

Now, let us have a deeper understanding on the Word2Vec model.

We present here the two main models used to produce word embeddings: Word2Vec and $CBOW$ (Continuous Bag of Words).

In the Skip-Gram model, we are given a text window (typically a few words) around a center word. The goal is to find a vector representation for the words, such that the probability of having the neighboring words is maximized. In other words, the goal is to find the solution of the program:

$$\arg\max_{\theta} \prod_{w \in Text} \left[\prod_{c \in C(w)} p(c|w;\theta) \right]$$

where c are the words in the context of the word w or, in a more compact way:

$$\arg\max_{\theta} \prod_{(w,c) \in D} p(c|w;\theta)$$

with D the set of all words and context pairs from the text.

It is important to notice here that, saying that maximizing this objective will result in good embeddings, i.e. that similar words will have similar vectors, is not a straightforward statement, and remains an assumption from the authors of this model.

In order to parameterizing the skip-gram model, we follow a neural network approach, and take for the conditional probability $p(c|w;\theta)$ the softmax function:

$$p(c|w;\theta) = \frac{e^{v_c.v_w}}{\sum_{c' \in C} e^{v_{c'}.v_w}}$$

where v_c and $v_w \in \mathbb{R}$ are vector representations for c and w respectively. C is the set of all possible contexts in the text. Hence, the parameters to be learned, θ, consists of all v_{c_t} and v_{w_t} for $w \in V$, $c \in C$ and $i \in 1, ..., d$. Hence, in total, there are $|C|.|V|$ parameters to be learned. We want to compute them by maximizing the product of all conditional probabilities. This is the same as maximizing the sum of the log of these probabilities, hence we wish to solve the equivalent problem:

$$\arg\max_{\theta} \sum_{(w,c) \in D} \log(p(c|w))$$
$$= \arg\max_{\theta} \sum_{(w,c) \in D} \left[\log(e^{v_c.v_w}) - \log\left(\sum_{c'} e^{v_{c'}.v_w} \right) \right]$$

From this program to be solved, we see an important issue. While the objective can be evaluated, its computation is not efficient because we must sum over all possible words in the vocabulary. If the vocabulary is large, which is usually the case, the computation time will have in effect that these calculations will not be efficient at all. Hence, there is a need to compute the sum over the whole vocabulary in a more efficient way.

In order to deal with this issue, the authors of the model use another objective function, and define the negative sampling objective by:

$$\sum_{(w,c) \in D} (\sigma(v_c.v_w) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim \mathbb{P}_n(w)} \left[log(\sigma(-{v'_{w_i}}^T.v_{w_i})) \right])$$

where there are k negative sampled for each data sample. According to the authors, a value between 5 and 20 is good for small datasets, whereas for larger dataset a small k can be taken, between 2 and 5. The noise distribution $\mathbb{P}_n(w)$ is a free parameter, the authors chose to use a unigram distribution, raised to the power 3/4.

In the end, the parameters are learned using a feed forward neural network. after initialization of the word vectors, they are applied to a sequence of linear / non linear activations (each pair consisting of a hidden layer of the neural network) until the last layer where a softmax function is used to get the probabilities. Once these probabilities are computed, we compute the negative sampling objective. At this moment, the forward step has reached its limit, and the background step can start, where the gradients of the parameters are successively computed using the chain rule, which enables to update the parameters. The authors have implemented a parallel version of the stochastic gradient descent optimizer in order to enhance the efficiency of the algorithm.

The Continuous Bag of Words model is very similar, but instead of predicting the context words based on the center word, the objective is to predict the center word based on its contexts words.

3 Analysis of performance for the two strategies

3.1 Presentation of the dataset

In order to test the methods presented in this paper, we use the dataset from [Tas16]. The author has gathered reviews from 8 books on Amazon.com, each review being assigned to a rating, ranging from to 1 for very negative feeling, to 5 for very positive feeling. In total, the dataset consists of 219242 rows, each being a textual review with a rating for a given book.

We propose first to develop the methods on a sample of the dataset, which consists of only one of the 8 books, "The Girl on the train", from Paula Hawkins. Then, we adapt the procedure to the whole dataset.

The way of rating a book is something that can vary between people, it is very subjective. Indeed, it would be hard for an algorithm to discriminate between a 1 and a 2, for example. Hence, as the author proposes, we will consider a rating of 1 or 2 as negative, a rating of 4 or 5 as positive, and a rating of 3 as neutral. In the end, the goal will be to perform a tri-class classification task. However, once this sentiment feature is created, we observe that the distribution of sentiment (between negative, neutral and positive) is not uniform at all. There is a dominant class, consisting of positive ratings. We can imagine that readers who loved the book would be more eager to take the time to write a review. In the other hand, only the reader who are really upset with the book would like to criticize it publicly. The problem with imbalanced datasets is that classification algorithms may always predict the dominant class. Indeed, this would lead to a very decent accuracy for the classification task. But this would not lead to a good classification in the sense that the algorithm would not be able to classify correctly an element from the less represented class. One solution is to sample randomly the dominant classes, such that, in the end, we have the same number of elements in each class, hence resulting in a uniform dataset. This paper uses this solution to deal with the imbalanced dataset issue.

3.2 Results

Table 1 and table 2 present the results of both approaches (Bag of Words model and Word2Vec model) for both datasets. For each model, we detail the results for both submodels (TF and TF-IDF for the Bag of Words model, word averaging and clustering for Word2Vec model).

The results are given in terms of accuracy, i.e. the percentage of reviews which are classified to the right sentiment.

Dataset	*Toy Dataset*		
	Logistic	RF	NB
BoW (TF)	0.364	0.360	0.350
BoW (TF-IDF)	0.347	0.343	0.355
Word2Vec average	0.361	0.360	0.366
Word2Vec cluster	0.347	0.353	0.358

Table 1: Results for the Toy Dataset

Dataset	*Total Dataset*		
	Logistic	RF	NB
BoW (TF)	0.448	0.423	0.412
BoW (TF-IDF)	0.414	0.426	0.412
Word2Vec average	0.451	0.429	0.410
Word2Vec cluster	0.439	0.409	0.407

Table 2: Results for the Total Dataset

As expected, we observe that the models resulting from a larger dataset are way better than the results from the toy dataset.

Contrary to our prior belief, we observe that the TF-IDF model does not create better features than the TF model, as the results show.

It seems that the combination Word2Vec + word averaging provides better results than the combination Word2Vec + clustering, for both toy and total datasets.

Then, also contrary to our prior belief, it seems that the Word2Vec model does not lead to better classifications than those produced by the Bag of Words model. This may be partly due to the fact that, even in the larger dataset, there are not enough data. Indeed, deep learning models need very large amounts of data to be successful.

Finally, we observe no clear distinction between the classifiers on the toy dataset. However, on the larger dataset, we observe that the logistic classifier provides the best results most of times. Naive Bayes seems to give poorer results.

4 Conclusion and perspectives

In this paper, we review some very different methods to predict the sentiment of a review, from bags of words to the deep learning Word2Vec model. However, there have been other improvements to the task of sentiment analysis due to deep learning. For example, we have modeled the sentences starting from the representation

of words into vectors, and then we used the word vectors to produce text vectors. In this last process, we lost the ordering of the words, and hence lost some information. Hence, we could learn a representation directly for the sentence level, or even for the paragraph level. This was highly studied in [LM14]. Finally, we could also implement the recursive neural network used in [SPW$^+$13], which is state of the art for many sentiment analysis datasets, and even a more recent paper from [FK17] where the authors develop structural attention neural network for sentiment analysis.

References

[FK17] A. Potamianos F. Kokkinos. Structural attention neural networks for improved sentiment analysis. 2017.

[Har54] Zellig S. Harris. Distributional structure. In *Word*, page 156, 1954.

[LM14] Q. V. Le and T. Mikolov. Distributed Representations of Sentences and Documents. *ArXiv e-prints*, May 2014.

[LZ12] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In *Mining Text Data*, pages 415–463, 2012.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[SPW$^+$13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA, October 2013. Association for Computational Linguistics.

[Tas16] Ahmet Taspinar. Sentiment analysis with bag-of-words. 2016.