Data representation in sentiment analysis task

Hani Hamdan[1], Pierre Vigier[1], and Frédéric Wantiez[1]

[1]Department of Signal and Statistics - *CentraleSupelec*
{*Hani.Hamdan, Pierre.Vigier, Frederic.Wantiez*}*@supelec.fr*

January 30, 2017

Abstract

Sentiment analysis, or opinion mining, refers to the task of exctracting subjective informations contained within text materials. The basic form of sentiment analysis is polarity classification. Intuitively, it aims to determine whether a piece of text expresses a positive sentiment or not with respect to a given subject. This process is already used in the industry. It has applications in a wide range of sectors such as marketing, online sales or opinion forecasting. These technics have gained popularity thanks to the essort of modern social networks and the ever growing amount of labeled data. Most industrial applications use traditionnal classification algorithm combined with refined data representations such as Logistic Regression or Random Forest. Even with not so complex classifiers, industrial applications achieve great score on classification tasks. This is mainly due to the effort put in preprocessing and representing input data. We thus discuss how refined representations of text materials can lead to great performances.

1 Description, data and technologies

1.1 Description

The sentiment analysis task is a supervised learning problem. We consider a set of texts and their associated sentiment or opinion. Since mesuring sentiments and opinions may be tough most datasets use a numerical value derived from the rating on a comment or the number of votes on a review. Specifically, we can define the sentiment analysis task in its polarity classification variation as follows: Let V be the set of every possible words and $V^* = \cup_{n\geq 0} V^n$ the set of all the possible texts using the vocabulary V. We note $x_1, ..., x_N \in V^*$ the texts and $y_1, ..., y_N \in S$ the corresponding labeled sentiment. Sentiments can take their value in $S = [0, 1]$ where the 0 value express a very negative sentiment and the value 1 an utterly positive sentiment. This can be reduced to a binary classification task if the labels take their value in $S = \{0, 1\}$. The classification task aims at determining a function f such that $\forall i \in 1, ..., N, y_i \approx f(x_i)$. We also want that function to generalize to unseen piece of texts. Indeed, in real-world applications our model will get unlabeled data and it will have to predict the corresponding sentiment. In order to mesure the quality of a given classifier or reprensation of data we define the accuracy metrics : $A(y_1, ..., y_N, \hat{y}_1, ..., \hat{y}_N) = \frac{1}{N} \sum_{i=0}^{N} 1_{y_i = \hat{y}_i}$. Our goal is to maximize the accuracy given a text representation and a classifier model.

1.2 Data

In order to train our models we need a large amount of labeled data. It is fairly easy to create such training sets. In fact, one can use websites with rated comment systems or rated products sections. The grade on the comment or on a given product expresses the correpsonding sentiment. Websites such as IMDB or the e-commerce platform Amazon present such features. In our work we opted for the IMDB dataset released by Maas and al. [4]. It contains 50 000 comments in english extracted from the IMDB website with their associated sentiment.

The dataset is divided between a training set and a validation set. The label is a binary value, a 0 represents a grade lower than 4 and the 1 represents a grade greater than 7. Values 5 and 6 are not considered because they represent a neutral opinion. Distribution of postive and negative sentiments is even.

2 Models and pipeline

Our work aimed at comparing text representations within the sentiment analysis task. Hence, we had to create different representations of the IMDB dataset and feed them in classifier models with and increasing complexity into the representations. The approach was straight-forward : We took a set of features to represent the reviews, we trained classifiers on them, mesure the accuracy and analyze the results. Once we determined flaws in our model we moved on more complex representations in order to improve the performances. However, as most classifiers require fixed-length numerical vectors as input data, we had to find numerical representations of the reviews which have variable length.

Accordinglyn, we had to preprocess the reviews, remove all the hyperlinks and all the fancy typography. Then we had to model the reviews, which are variable length pieces of english text using features that we may use within our learning algorithms.

3 Feature ingineering

Our work focused on creating and combining good features for the reviews. We present here the set of representations we used and describe them in a formal way. These features are sorted by complexity. We first introduce some simple ways to model text and move on to more complex representations.

3.1 Bag of Words (BoW)

Bag of Words [7] are a really simple representation. If we number the elements of V we have then $V = \{w_1, ..., w_D\}$. Let's $t \in V^*$ be a text and note $tf_{i,t} = card(\{j, t_j = w_i\})$ the number of time the word w_i appears in the text t. The Bag of Words for the text t is the vector b in $\mathbb{R}^D$ such that $\forall i \in 1, ..., D, b_i = tf_{i,t}$. In other words, the i^{th} coordinate of the vector b is the number of occurences of the word w_i in the text t. Let's consider a toy example. Given the two following sentences

$$S_1 = \text{"Bob likes action movies"}$$
$$S_2 = \text{"Alice prefers romantic movies"}$$

the vocabularay is:

$$\{\text{Bob}, \text{likes}, \text{action}, \text{movies}, \text{Alice}, \text{prefers}, \text{romantic}\}$$

Keeping this order we have the related Bag-of-Words b_1 and b_2 for the sentences 1 and 2:

$$b_1 = (1, 1, 1, 1, 1, 1, 0, 0, 0)^T$$
$$b_2 = (0, 0, 1, 1, 1, 0, 1, 1, 1)^T$$

There is also another well-known version of Bag-of-Words, the tf-idf, short for term frequency–inverse document frequency [7]. In this version, the coordinates of the vector are now weighted by a measure of the scarcity of a the coresponding word : the IDF. We have :

$$\forall i \in \{1, ..., D\}, idf_i = log(\frac{N}{N_i})$$

where $N_i = card(\{k, tf_{i,x_k} > 0\})$ is the number of reviews containing the word w_i and N the total number of reviews. Intuitively, the idf weighting gives a greater weight for rare words as $\frac{N}{N_i}$ increases with a decreasing number N_i. The i^{th} coordinate of the BoW is then given by $b_i = tf_{i,t} \times idf_i$.

3.2 Word Vectors

Another approach may be to find a continuous representation for the words. In the BoW model, coordinates of our vectors were discrete and thus the quality of the model was linked with the size of the vocabulary.On the contrary, continuous models result in a very compact way to represent texts. Thus, with the BOW representation, input data is a vector of length $card(V)$ which can lead to a tremendous amount of parameters to tune within our models. Contrarywise, with continuous representations we can set the length of the input vectors. With longer vectors we get subtler representations. In our case we chose a dimension of 300 to get a pretty complex representation and still be able to compute on reasonable time. In the following, we note $v_i \in \mathbb{R}^{300}$ the vectors representing the word w_i.

Several methods exist but we chose the Skip-Gram model introduced by Mikolov and al. [5] [6]. The basic idea behind the Skip-gram model is to train a neural network, see figure 1, to predict surrounding words of a given word in a sentence. Once the model is trained the hidden layer contains the numerical representation of words.

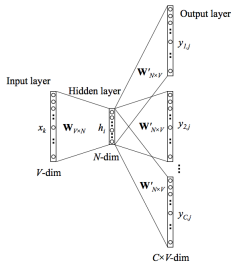

Figure 1: Skip-gram model. Mikolov and al. [5]

This model gives us semantic and syntaxic informations about the words. Indeed, words with similar contexts will get closer vectors in the feature space. Words with similar contexts are words with similar grammatical function or similar meaning, this means that the feature space will contain word clusters related to a given subject. To visualize this we ploted a 2-dimensional representation of our vectors using the t-SNE algorithm [8]. This visualization offers the advantage of preserving distances (figure 10).

In order to get a representation for the reviews we can use a linear combination of word vectors. First we can assume that every word get the same weight and thus the text vector is given by:

$$h_{vec}(t) = \frac{\sum_{w_i \in t} tf_{i,t} v_i}{\sum_{w_i \in t} tf_{i,t}}$$

Where $tf_{i,t}$ is the term-frequency of the word i in the text t as described above and v_i is the word vector associated with the word w_i obtained using the Skip-gram model.

However, as we used with the TD-IDF BoW, we can also consider the scarcity and weight the words using the Idf. We have:

$$h_{vec+idf}(t) = \frac{\sum_{w_i \in t} tf_{i,t} idf_i v_i}{\sum_{w_i \in t} tf_{i,t} idf_i}$$

3.3 N-gram

The two representations described above do not express word order. It seems reasonable to think that word order may contain semantic information. Thus, previous representations considered words independantly and if we consider the phrase "It was not amazing", it appears that some really meaningful words are present like "amazing" and the classifiers will probably misclassify this sentence as positive. However, if we take into account word order i.e if we consider sentences like sequences we should be able to refine our representation. One solution may be the N-gram model. The n-grams of a given piece of text t are the n-tuples of consecutive words of t. Let's consider the sentence t = "Bob likes action movies." the coresponding bigrams or 2-grams are $\{(\text{Bob}, \text{likes}), (\text{likes}, \text{action}), (\text{action}, \text{movies}), \}$
As we see, n-grams take into account syntaxic structures such as "not like". With increasing dimension n, the model can represent more complex structures. However, the number of n-tuples increase quickly with n like $O(card(V)^n)$. But with a great value of n we would get scarce n-tuples with most of them appearing no more than a single time throughout the text. These rare n-tuples are not meaningful features for the analysis, thus we can limit n to small values. Our tests show that $n = 3$ represents a good compromise.

To get numerical vectors we wonsider bag of n-grams instead of bag-of-words i.e a Bag-of-N-Gram representation for a given review is $b \in \mathbb{R}^D$ such that $\forall i \in 1, ..., D,\ b_i = tf_{i,t}$ where $tf_{i,t}$ is the number of times the n-gram i appears in the text t. Practically, we considered the most frequent 2-grams and 3-grams to get reasonable computation time.

3.4 Paragraph Vectors

Another solution is to consider our paragraphs as a whole and try to find features for the paragraph. Mikolov and al. [3] presented a framework to extend the Word2Vec model to larger pieces of text like sentences or paragraphs. The idea is basically the same, the network tries to figure out the following word given a context. However this model includes a Paragraph Matrix which contains the information about the whole piece of text. This

representation presents the advantage of modelling complex structure within the text material. As the piece of text is considered as a whole, its inner structure is preserved and thus it can model complex syntaxic dependancies within the text. This representation also provides fixed-length representations for the reviews. We get vectors in $\mathbb{R}^n$ with $n = 300$ to feed into our classifiers.

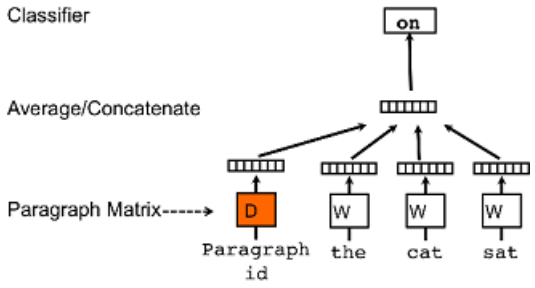

Figure 2: Learning framework for Pargraph Vectors (source : Mikolov et al. [3]).

Another advantage of this representation lies in the fact that it considers word order and preserves the semantic meaning. If we look at similar words of "interesting" with this model we get the figure 3.

```
In[1]: model.most_similar('interesting')
Out[1]:
[('enjoyable', 0.549770712852478),
 ('entertaining', 0.5360784530639648),
 ('important', 0.5295416712760925),
 ('intriguing', 0.4986931085586548),
 ('amazing', 0.49587947130203247),
 ('exciting', 0.4942770004272461),
 ('excellent', 0.4942253828048706),
 ('awesome', 0.46036389470100403),
 ('amusing', 0.4536857306957245),
 ('impressive', 0.448122501373291)]
```

Figure 3: 10 most similar words to 'interesting' for a PV-DM model using mean.

4 Experimental results

In our experimentations we used two classifiers: the Logistic Regression and Random Forest algorithm. Logistic Regresion is a basic classifier which models linear dependancies in our data [1]. However we can analyze coefficients of the model to get a better understanding of which features we get. These coefficients may be expressed as the log-probability to belong to a certain class. Thus, we can subsequently analyze the relative importance of some groups of word. On the contrary, the Random Forest algorithm can model non-linear dependancies [2] but its parameters can not be interpreted easily. To carry out our experimentations we used Python 3.5 and some of its available libraries:

- *pandas*: Data manipulation;
- *Numpy*: Linear Algebra;
- *Matplotlib*: Ploting and graphics;
- *scikit-learn*: Learning algorithms;
- *TensorFlow*: Neural Network;
- *NLTK*: Natural language processing.

To get the results shown below for the different models described above we used the following workflow : we preprocessed our data, built a given representation, fed it into our classifiers and measured its accuracy.

4.1 Bag-of-Words

The results obtained with the Bag-of-Words (BOW) model is summarized in figure 4. To get reasonable computation time, the model only considered the 5000 most frequent words. As we can see, it does not present great results. This is mainly due to the extreme simplicity of the BOW model. It does not consider word order or any semantic informations about the text. Its underlying structure is not represented. It also appears that with so little informations about the semantic content classifiers tend to converge too fast. If we look at the learning curves of the logistic regression it appears that the BoW representation lacks variance i.e parameters.

The parameters of the logistic regression may give us some insight. They represent the relative

Method	Accuracy
RF + BoW	0.84356
RF + BoW + TF-IDF	0.83952
LR + BoW	0.84748
LR + BoW + TF-IDF	0.88308

Figure 4: BOW model accuracy. (LR : Logistic Regression, RF : Random Forest, BOW : Bag-of-Words, TF-IDF : TF-IDF weighting). Remark : Random Forest with 100 estimators.

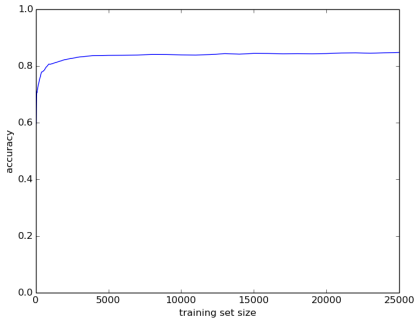

Figure 5: Learning curve with BoW input.

weight of a certain word in our model i.e a great value expresses a very positive sentiment related to the given word. If we look at the 5 most negative words in our models listed below (figure 6) we see that some words like "worst" or "bad" have a really strong importance. Their presence in a sentence will probably force the classifier towards negative sentiment classification whereas constructions like "not bad" are not considered.

```
NEGATIVE
1. worst (-9.207327806842622)
2. bad (-7.259958398881469)
3. awful (-6.415897979257145)
4. waste (-6.330645081158189)
5. boring (-5.972322478351235)
```

Figure 6: Most negative words and their associated weight in the logistic model.

4.2 Word vector

The Word Vector model gave slightly worse results (figure 7) and the TF-IDF weighting did not improve the performances. These results show that the Word Vector model does not increase the complexity of the representation. It is basically a continuous version of the BoW model. Words are represented with too little semantic information and we find the same flaws as those we may find within the BoW model. This representation however presents the interest of being really compact and can generalize well and give results equivalent to the BoW model.

Method	Accuracy
Random Forest + Vec	0.7954
Random Forest + Vec + TF-IDF	0.79416
Logistic Regression + Vec	0.81652
Logistic Regression + Vec + TF-IDF	0.81688

Figure 7: Accuracy (Vec : words vector, TF-IDF : IDF weigthing)

4.3 N-gram

Precedent representations did not take into account the semantic construction of the sentence and to improve performances input data should take into account word order and consider sentences as variable length sequences. We considered bag-of-n-grams and fed them into our classifiers. We only kept the most frequent 1-grams and 2-grams to get a reasonable computing time. Our results are summarized in figure 8.

Here we see that performances increase significantly with the number of bigrams and trigrams considered. This due to the fact that n-grams hold more information about semantic constructions. This representation can model constructions such as "not bad" or "not so terrible" which are really meaningful. It appears that increasing the number of bi-grams improve largely performances. The TF-IDF weighting process also improve significantly overall performances. Reasons evoked for the BoW model also hold here, scarce bigrams or trigrams can be really meaningful. The 0.89 performance is already decent considering our basic model.

C	n	TF-IDF	Accuracy
5000	1	No	0.85164
10000	1	No	0.85432
15000	1	No	0.85664
5000	2	No	0.8582
10000	2	No	0.86844
15000	2	No	0.87544
5000	1	Yes	0.88308
10000	1	Yes	0.88364
15000	1	Yes	0.88412
5000	2	Yes	0.88848
10000	2	Yes	0.89312
15000	2	Yes	0.89432
20000	2	Yes	0.8954
30000	2	Yes	0.89564

Figure 8: N-grams accuracy with Logistic Regression

4.4 Paragraph vectors

With Paragraph vectors we achieve high performances for these relatively simple classifiers. Results are shown in figure 9.

Logistic regression + PV-DM (concat.)	0.883
Logistic regression + PV-DM (mean)	0.823

Figure 9: Paragraph vectors accuracy.

Finally, our best result was obtained with a concatenation of bag-of-bigram containing 10000 elements and paragrapgh vectors of length 300 We obtained 90.116% which is already a good performance for some really basic classifiers.

5 Conclusion

In this article we presented several representations for text as input data. Starting with basic models and simple classifiers we increased the complexity. Simple models and classifiers have shown to perform well and trying to use mixed representations improve results by a great amount, achieving almost 90%. However these models show some limits especially conerning word order and syntaxic construction modelling. Performances would certainly improve with more complex classifiers. Some of these more complex classifiers, like Recurent Neural Network can model sequences and thus it can take into account semantic features of text classification such as grammatical constructions.

References

[1] Walker SH Duncan DB. Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 1967.

[2] Tin Kam Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*.

[3] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[4] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, 2011. Association for Computational Linguistics.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[6] 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.

[7] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[8] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

6 Appendice

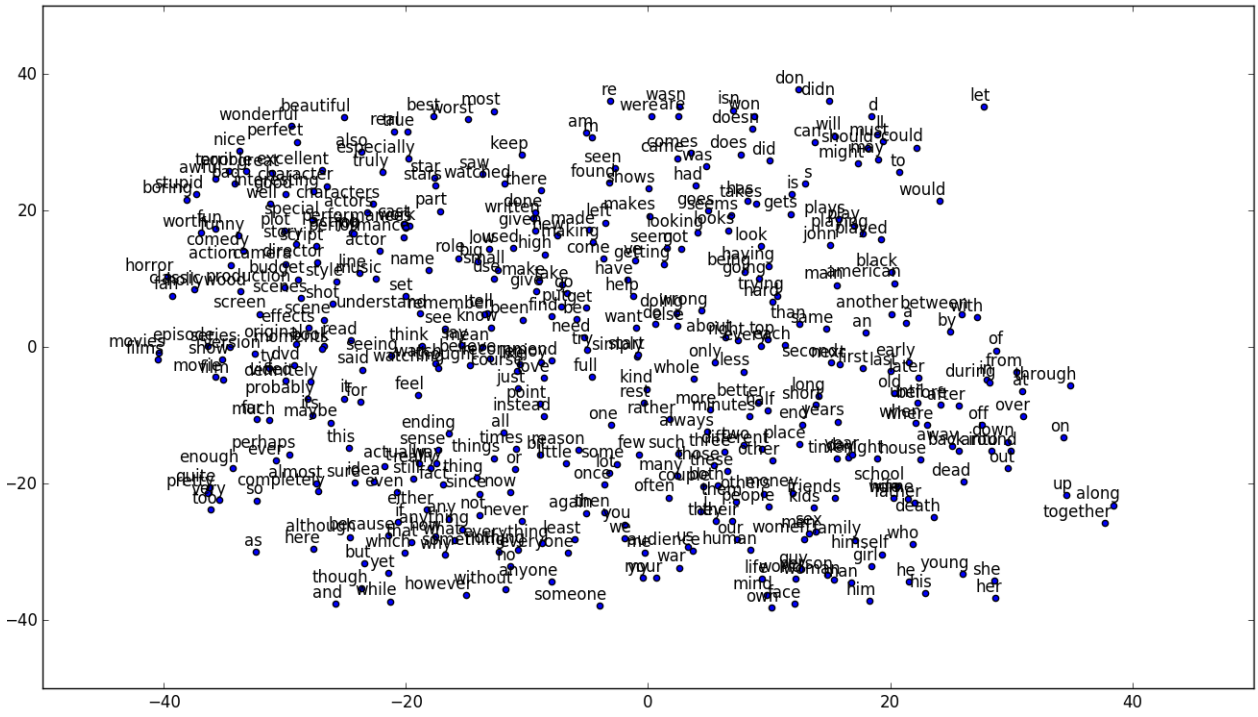

Figure 10: t-SNE visualization of word clusters.