University of Passau

Text Mining Project 5981P

Learning Sentiment-Specific Word embedding for Twitter Sentiment Classification

Author:	*Student Number:*
Abdelaziz Ben Othman	79229
Montasser Ajam	75997
Mounir Ouled Ltaeif	79173

October 13, 2017

Contents

Abstract

Most of the tasks in natural language processing involve looking at individual words and try to extract information from different factors, namely their distribution in the documents,syntactic contexts,their frequency weighting but always ignore the sentiment of the continuous form of these words.
In this paper we represent a method that learns word embedding for Twitter sentiment classification.
Word embedding for social media sentiment analysis and classification has gained increasing importance as it gives better results than the other related existing methods.

keywords : *sentiment analysis ; tweet sentiment ; sentiment-specific word embedding ; text classification*

1 Introduction

Recently, the social web network has experienced a huge growth in the number of its users where they are daily sharing their opinions and expressing their feelings. These data are extremely valuable and from them we can extract so many information.

Twitter sentiment classification is one of the popular and attractive tasks that is widely used in order to analyze feedbacks, reviews and opinions towards products, persons, events, etc. This kind of analysis is very important in decision making specially with nowadays connected world.

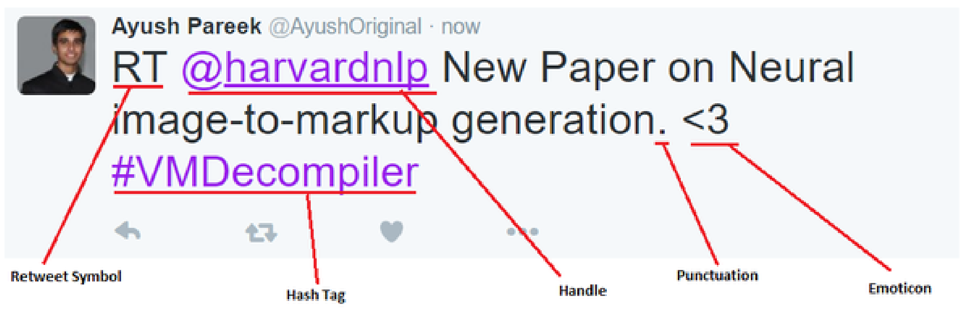
However, the existing approaches for twitter analysis are generally based on training a machine learning model and using it to predict the new tweet polarity. The features used to train the machine learning classifier are typically extracted by the traditional NLP methods such as Bag of words and TF-IDF. These traditional features did not yield to a robust model and fail to predict unseen data. So that researchers tried to find another feature representation to improve the model learning performance. In this context, the Word Embedding comes as a promising alternative and becomes the most used technique to construct a vector representation of word. Word2Vec and C&w models are two words embedding models that map word to vector representation which integrates the context and the syntactic relationship between words but ignore the sentiment polarity of words.
Specific-Sentiment word Embedding (SSWE) is an embedding algorithm that encodes the sentiment information in the vector representation. In this project, we will explore this technique to learn continuous representations of tweet words to train a twitter sentiment classifier. Our approach is based on training a SSWE model and then used to extract features from tweets and use them to train twitter sentiment classifier.

2 Tweet preprocessing

The Pre-processing step is fundamental in all Natural Language Processing tasks. It strongly depends on the targeted requirements and differs from an application to another. For instance, when dealing with social media corpus, it is important to notice that the users tend to use very informal language knowing that they have the total freedom to write whatever they like (i.e without any grammatical or syntactical restrictions), this factor introduces to any extracted dataset from a social media both grammatical and spelling errors which make the data very noisy from a semantic perspective. Added to this, social media users tend always to express themselves using "emoticons", acronyms, hash-tags, URLs, etc [13].

A good starting point when designing a pre-processing scheme should consider a good knowledge of the user's writing behavior over the social medias as well as a good understanding of the treated problem. For instance, It could be necessary to pre-process data and filter an existing expression or a pattern for a certain NLP problem because it doesn't contain any information needed to solve the problem, and decisive to keep it for another different task.

When treating the case of analyzing tweets posted on the Worldwide Micro-blog Twitter, one can remark that the posts present often some repetitive patterns, namely, **Hash-tags** before words, **Retweet symbols Handles**, a remarkable use of **Punctuations**, **Emoticons**, as well as the use of grammatically correct words but also misspelled written words, some other patterns as **Repeated characters** are also very common. We can also notice that **Abbreviations** are widely used in tweets. This will naturally make the information extraction process more complex and for that reason we followed the following preprocessing steps present in the sub-sections of this part of the report.



Ayush Pareek @AyushOriginal · now

RT @harvardnlp New Paper on Neural image-to-markup generation. <3 #VMDecompiler

Figure 1: unprocessed tweet

2.1 Removing Links/URLs

The URLs and Links are maybe the most widely used patterns which appears to be the least useful for our Task. A Word embedding which is considering to embed Links and URLS in vectors is obviously very wrong, as these web directions don't have any impact on the sentiment of the tweet. Hence, We have removed these patterns using the following regular expression and the **re** module providing regular expression operations:

```
def removeUrls(tweet):
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+) |
    ↪  (http?://[^\s]+))','URL',tweet)
```

2.2 Removing usernames

User mentions are used to refer to a person name. names begin with an at sign "@" symbol followed by the user-name. User-names are used as a unique reference to a person's account and add no information to the collected entropy about the sentiment of the tweet so that's why they're removed.

```
def removeUsername(tweet):
  tweet = tweet.lower()
  tweet = re.sub('@[^\s]+','AT_USER',tweet)
```

2.3 Removing Hashtags

Users use hashtag symbol (#) before a relevant keyword or phrase in their Tweets to categorize or prioritize it over the other. Keeping the hashtags may mislead any embedding learning which can consider these "hashtagged" words as new words rather from their original form without Hashtagging. Therefore, we decided to remove all hashtags and keep the words that come after. We have used here also **re** module:

```
def removeHashtags(tweet):
  tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
```

2.4 Repeated characters

People always use repeated characters when they want to express their emotional state, It is very common to find tweets like "I'm happyyyyyy","We won, omgggggg!". The repeated characters present in these kind of tweets may mislead the embedder which will consider these forms of words, if not preprocessed and the original form as two different words.

We have chosen to remove these repeated characters to end-up with a normalized text representation. This is done following the rule that a letter can't be repeated more than three consecutive times. To do what has just been mentioned, we used the built-in module **re** providing **regular expression matching operations**, The following function implementation looks for 2 or more repitions of character in an input string and outputs a normalized text:

```
def replaceTwoOrMore(s):
        #look for 2 or more repetitions of character
        patt = re.compile(r"(.)\1{1,}", re.DOTALL)
        return patt.sub(r"\1\1", s)
```

2.5 Removing stop-words

Stop Words are words which do not contain important significance to be used in Search Queries or in text analysis. Usually these words are filtered out from the corpus because they return vast amount of unnecessary information. Every language has a set of stop words , and for instance, Words like ”This”, ”here”, ”I” belong to the list of stop words of the English language. Learning how to eliminate these words from the tweet is tricky due to the following facts [6]:

- Learning phase becomes much faster since we eliminated many features from the tweet.
- Prediction can be also more accurate since we are eliminating noise and distracting features.

It is also important to mention that the list of stop words is not unique for a specified language, and one can find different stop words lists for the english language, what matters is to know which stop word list to use for our application. For instance, we found for example stop words lists containing words like ”good” or ”bad”, we have estimated that this kind of lists won't be suitable as we may need the embedding of these words for the sentiment classification part. This has conduct us to choose carefully the list of the stop-words in order to improve the power of the prediction of our classifier.

2.6 Conclusion

In this chapter, we have described the preprocessing that we have done to our dataset. To summarize, We have performed three subtasks. The first one consists of removing URLs/Links, usernames and Hashtags. The second one removes repeated letters of a word and the last one deals with removing stop-words. We have also done other filtering steps of some patterns. For instance, datetimes, days, months, years and other similar words which are filtered. More detailed information about the pre-processing, in the module we have implemented for this step and namely **preprocess**.

3 Dataset

This section describes the datasets that we used during this project. As our project is composed of 2 main parts: the first part consists on learning a **specific sentiment word embedding (SSWE)** model and **the second part consists on training a classifier** for predicting the sentiment expressed by tweets. Therefore, tow different datasets were used in order to achieve these objectives.

3.1 sentiment140 dataset

[1] For training the SSWE model we need a large number of tweets as it s specified in the published paper of SSWE [7]. So we chose the dataset sentiment140 [2] which contains 1600000 tweets collected using Twitter API. The approach followed to collecting this dataset was based on the assumption that any tweet with positive emoticons, like "**:)**", were positive, and tweets with negative emoticons, like "**:(**", were negative. The detailed description of this approach is reported in [3]. The sentiment140 dataset is available on a CSV format and its fields are described below in the table 1. As We have used the **deepnl** [3] to train a SSWE model whcih requires

column	values
0	the polarity of the tweet: 0 for negative, 2 for neutral and 4 for positive)
1	the id of the tweet (e.g : 2087)
2	the date of the tweet (e.g: Sat May 16 23:58:44 UTC 2009)
3	the query (e.g: lyx). If there is no query, then this value is NO_QUERY.
4	the user that tweeted (e.g: robotickilldozr)
5	the text of the tweet (e.g: Lyx is cool)

Table 1: sentiment140 dataset fields

and uses the presence of the tweets polarity to construct the word representation, this dataset provides us with the necessary data and informationto construct these vector representations.

deepnl needs as well a **large training** set annotated with polarity. In our case the training set is a large number of tweets. Deepnl defines the format of the training input. In fact, the tweets training file should be in the format of the SemEval 2013 Sentiment Analysis in Twitter, i.e. one tweet per line, in the following format: $< SID >< tab >< UID >< tab > polarity < tab >$TWITTER_MESSAGE The polarity values can be positive, negative, neutral or objective.

[1]http://help.sentiment140.com/for-students/

[2]http://help.sentiment140.com/for-students

[3]https://github.com/attardi/**deepnl**

Before using sentiment140 dataset, we cleaned because there are data rows with a distorted format and we also removed the too short tweets (less than 3 words). So after cleaning, we obtained a new dataset with 1102286 tweets. Then we removed the unnecessary fields from sentiment140 dataset, and we kept only the tweets text and their polarities. As a result we get the new format of the training dataset:
TWITTER_MESSAGE < *tab* >polarity
The table 2 describes the sentiment140 dataset after cleaning:

Property	values
Number of tweets	1102286
Number of positive tweets	534590
Number of negative tweets	568102
Start date	Mon Apr 06 22:19:45 PDT 2009
End date	Tue Jun 16 08:40:49 PDT 2009

Table 2: sentiment140 dataset statistics

3.2 SemEval17 dataset

SemEval is an international workshop on Semantic Evaluation, and it proposes each year a set of semantic tasks. Some of these tasks are related to twitter data analysis. It provides a dataset to evaluate the proposed tasks.
We worked essentially with the SemEval17 dataset for twitter sentiment analysis. It is composed of train and test data. Both train and test data contain the fields described on the table 3: Before using the SemEval17 dataset, we cleaned it and

SID	language	sentiment	tweet
262163168678248449	en	negative	its not that I'm a GSP fan, i just hate Nick Diaz. can't wait for february

Table 3: semeval17 dataset fields

we set the polarity "objective or neutral" to "neutral". The SemEval train data contains 18051 tweets with the polarity distribution described in the table4 The

positive	negative	neutral
7755	2751	7545

Table 4: semeval17 train dataset

SemEval test data contains 11987 tweets with the polarity distribution described in the table5

positive	negative	neutral
3843	2360	5784

Table 5: semeval17 test dataset

4 Sentiment-Specific Word Embedding

Vector space models have been used in distributional semantics since the 1990s. Many models have been described for estimating the continuous representations of word embedding, Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA) being two such examples. The term **word embeddings** was originally described by Bengio et al. in 2003 [1] who trained them in a neural network language model together with the model's parameters.

However, Collobert and Weston were the first to proove the efficiency of pre-trained word embeddings in their paper [2], in which they established word embdedding e as a highly effective tool to represent words in vector spaces for later use in classification or other tasks, they have also rather than that described a neural network architecture that most of today's embedding approaches are built upon. However, It was Mikolov et al. (2013), who really brought a big constribution that has been made to this area through the invention of word2vec, a powerful two-layer neural network model, that is trained to reconstruct linguistic contexts of words with a very precise and efficient way.

Lately, the SSWE model which stands learning sentiment-specific word embedding, and which encodes sentiment information in the continuous representation of words has made a breakthrough in Vector space models. In this report we give an overview about the classical model architectures C&W and Word2vec, Later on, a more detailed overview will be given for the SSWE model, citing its variants $SSWE_h$ and $SSWE_r$.

4.1 Classical technique of Word Embedding

4.1.1 C&W model

Collobert and Weston, a.k.a C&W, prove that word embeddings trained on a sufficiently large dataset carry syntactic and semantic meaning.
Their solution consist of training a neural network that gives a higher score f_θ for the correct next word sequence given the previous word.
They use a ranking criterion that looks like this :

$$J_\theta = \sum_{x\in X} \sum_{w\in V} \max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\}$$

x is a sample window that contains n words and X is the set of all possible windows of the corpus.
For each window x, they then produce a corrupted, incorrect version $x^{(w)}$ by replacing x's centre word with another word w from V.
The objectif of ranking criterion is to maximize the distance between the correct and the incorrect window with a margin of 1.[2]
While this model eliminates the computation's complexity of the softmax layer, it preserves an intermediate fully-connected hidden layer which itself constitutes another level of complexity.
C&W used seven weeks to train their model with a vocabulary $|V| = 130000$.

4.1.2 Word2Vec model

The most popular word embedding model proposed by Mikolov et al. in 2013.[5]
Technically, word2vec is not considered to be part of deep learning because it doesn't have multiple non-linear layers and can't learn feature hierarchies.
However, Word2vec has other training strategies that takes additional contexts into account.
Those training strategies enhance both the speed and accuracy of the prediction and get rid of the complexity computation of the previous word embedding models.
In the following, we will look further at those strategies.

Continuous bag-of-words (CBOW) : To predict a targeted word W_t , CBOW focuses not only on the previous words but also on the next ones as shown in the figure.

To evaluate the model, an objective function is generated that invokes a strong (naive Bayes) conditional independence assumption between the words.

The objective function is as following :

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \log p(w_t|w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n})$$

After that, the objective function is used to compute the gradients with respect to the unknown parameters and at each iteration update them via Stochastic Gradient Descent.

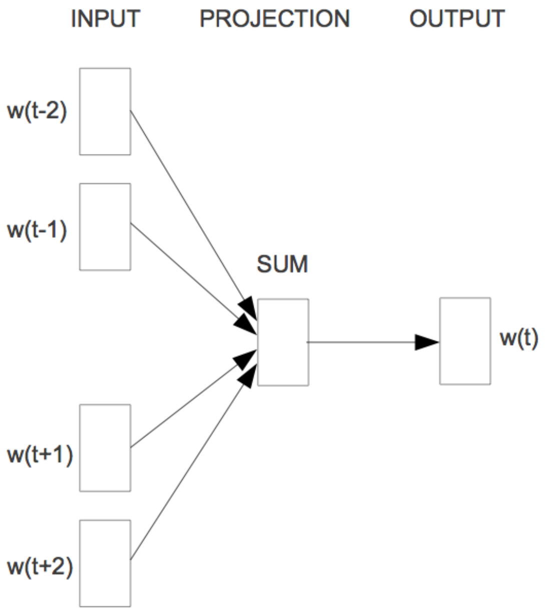


Figure 2: Continuous bag-of-words

Skip-gram : Instead of using the surrounding words to predict the target words, Skip-gram uses the center word to predict the surrounding words as show in figure. The objective function is as following :

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \geq j \leq n, \neq 0} \log p(w_{t+j}|w_t)$$

knowing that the next equation is equivalent to the softmax function defined in :

$$p(w_{t+j}|w_t) = \frac{\exp(h^T v\prime_{w_{t+j}})}{\sum_{w_i \in V} \exp(h^T v\prime_{w_i})}$$

Knowing that the skip-gram model doesn't have a hidden layer that produces an intermediate vector , here h is simply the word embedding v_{w_t} of the input word w_t.

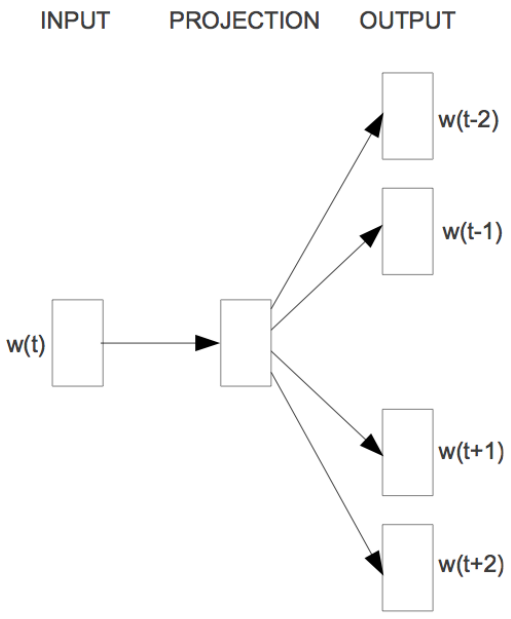


Figure 3: Skip-gram

4.2 SSWE model

The models described in the previous section yield continuous representations of words without considering the sentiments expressed by words. It means that the most word embedding algorithms ignore the sentiment information while learning the embedding and only model the syntactic context of words. Thus, Sentiment-Specific word embedding comes to overcome this problem. In fact, it is an extension of the existing word embedding learning algorithm C&W model to encode sentiment information in the continuous representation of words.
SSWE is introduced in [7], and it is designed to incorporate the sentiment information of sentences to learn continuous representations for words and phrases. It propose three models based on three neural networks with different approaches to integrate the sentiment information of tweets in their loss functions.

4.2.1 SSWE$_h$

The first approach SSWE$_h$ has the same neural network architecture as C&W model with an additional $softmax$ layer as it is shown in Figure4(b). The idea behind this model is to predict the sentiment distribution of the tweet based on input ngram using the continuous vector of the top layer of the neural network. As a result, it yields a feature vector for the specific input. In fact, it uses a window to go through the input sentence and then predict the sentiment polarity based on each ngram with a shared neural network.

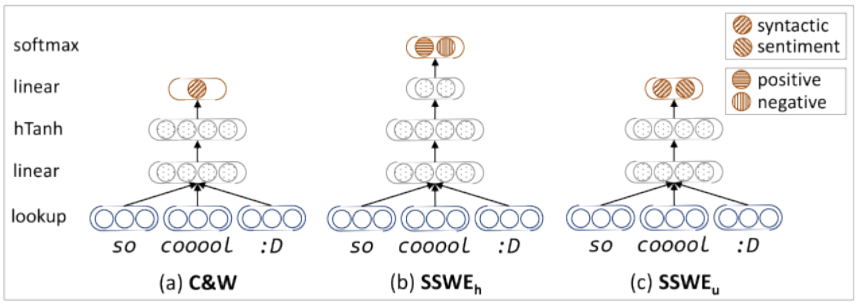


Figure 4: C&W, SSWE$_h$ and SSWE$_u$ models

The *softmax* layer is used to map the neural network outputs into conditional probabilities. SSWE$_h$ assumes that the positive distribution is of the form [1,0] and the negative distribution is of the form [0,1]. So it is trained by predicting the positive ngram as [1,0] and the negative ngram as [0,1] which are too strict constraints. The letter h in the $SSWE_h$ refers to the hard constraints of this model. The cross-entropy error of the *softmax* layer is:

$$loss_h(t) = -\sum_{k=\{0,1\}} f_k^g(t).log(f_k^h(t)) \tag{1}$$

Where $f^g(t)$ is the gold sentiment distribution, and $f^h(t)$ is the predicted sentiment distribution.

4.2.2 SSWE$_r$

It is an improvement of the first model, and it overcomes the hard constraints of the $SSWE_h$ to yield more relaxed constraints. Unlike $SSWE_h$, $SSWE_r$ model assumes that the positive tweet should have a higher predicted positive score than the predicted negative score and vice versa. For example the distribution of [0.7,0.3] is considered as a positive label because the positive score is larger than the negative score and the distribution of [0.2,0.8] indicates negative polarity.
SSWE$_r$ introduces a ranking objective function to relax the constraints. Like SSWE$_h$ model, it uses the bottom four layers described in the figure4(b), but it removes the

softmax layer as there is no need to the probabilistic interpretation.
The new loss function is described below:

$$loss_r(t) = max(0, 1 - \delta_s(t)f_0^r(t) + \delta_s(t)f_1^r(t)) \tag{2}$$

Where f_0^r is the predicted positive score, f_1^r is the predicted negative score and $\delta_s(t)$ is an indicator function reflecting the sentiment polarity of a sentence which is described below :

$$\delta_s(t) = \begin{cases} 1 & \text{if } f^g(t) = [1,0] \\ -1 & \text{if } f^g(t) = [0,1] \end{cases}$$

4.2.3 SSWE$_u$

Both SSWE$_h$ and SSWE$_r$ do not generate the corrupted ngram. So they learn sentiment-specific word embedding by integrating the sentiment polarity of sentences but They ignore the syntactic contexts of words. SSWE$_u$ is a unified model that capture both the sentiment information of sentences and the syntactic contexts of words. It is described in the Figure4(c).
SSWE$_u$ uses the original (or corrupted) ngram and the sentiment polarity of a sentence as the input to predict a two-dimensional vector for each input ngram. It defines two scalars (f_0^u,f_1^u) to compute respectively the language model score and the sentiment score of the input ngram. The training of SSWE$_u$ model aims to achieve the following goals:

- The original ngram should obtain a higher language model score $f_0^u(t)$ than the corrupted ngram $f_0^u(t^r)$
- The sentiment score of original ngram $f_1^u(t)$ should be more consistent with the gold polarity annotation of sentence than corrupted ngram $f_0^u(t^r)$.

The new function is a linear combination of two losses as it is illustrated below:

$$loss_u(t, t^r) = \alpha.loss_{cw}(t, t^r) + (1 - \alpha).loss_{us}(t, t^r) \tag{3}$$

Where $loss_{cw}(t, t^r)$ is the syntactic loss given by c&W model, and $loss_{us}(t, t^r)$ is the sentiment loss as described below:

$$loss_{us}(t, t^r) = max(0, 1 - \delta_s(t)f_1^u(t) + \delta_s(t)f_1^u(t^r)) \tag{4}$$

5 Word Embedding Visualization

In this section, we describe the approach that we have chosen to prove the semantic relationship between the word representations that we have already obtained as described in the previous part of the report.

Many features and metrics were used in the literature to determine the semantic efficiency of different word embedding algorithms. The choice of such metrics appears to be as crucial as the choice of the embedding algorithm itself, as these metrics will be the evaluators of how good a semantic structure is constructed between words and how the learning of this structure is done.

Our approach to study the semantic efficiency of the sentiment-specific word embedding is divided in two parts, namely **a Global Visualization** in which we try to understand the overall distribution of our vector representations as well as to explain the created structure of the whole set of vectors, and a second step consists of a **Specific Visualization** in which we further our analyzing of the word embedding by investigating the relationship between individual word representations in order to track the similarities between semantically close words as for example synonym words.

5.1 Global Visualization

In order to visualize the set of the word embeddings as a whole, we have first to patch it in a way that it becomes visually clear and more intuitive so that we can later infer the general structure and distribution of the data. One problem is to how visualize the **High-dimensional** word representations as the chosen size of the Embedding's vectors extends by far the 2-Dimensional or 3-Dimensional spaces. This High-dimensional dataset can be very difficult to visualize and even impossible considering the complexity of the problem and the available resources. To aid visualization of the structure of the word embedding, the dimension must be reduced in some way.

Hence, we have opted to map our representation from the original chosen size of the Embedding's vectors to the 2D-space, and because a simple random projection of the data or even using PCA is likely to cause a loss in the more interesting structure within the data. We have chosen to map our data using the **t-distributed Stochastic Neighbor Embedding (t-SNE)** which is a very effective way to preserve the neighborhood structure of the data i.e the clustering in the high dimensional space is preserved in the low dimensional space (2D-space in our case).

t-SNE converts affinities of data points (i.e co-occurrence relationships) to probabilities. The affinities in the original space are represented by Gaussian joint probabilities and the affinities in the low-dimensional space are represented by t-distributions, which is the sampling distribution of the t statistic given by this formula:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}} \tag{5}$$

where $\bar{x}$ is the sample mean, μ is the population mean, s is the standard deviation of the sample, and n is the sample size. This allows t-SNE to be particularly sensitive to local structure but also to reduce the tendency to crowd points together at the center and more importantly to reveal data that lie in multiple, different, manifolds or clusters.

To add more interactivity with the user, we added some features to the visualization such as the display of the corresponding word when the user hover on the data point, this is done using an association of the vectors to their corresponding words, to do so , our visualizer has inherited the class Keyedvectors from the module **gensim.models.keyedvectors** and we stored the words and vectors in a Keyedvectors instance.

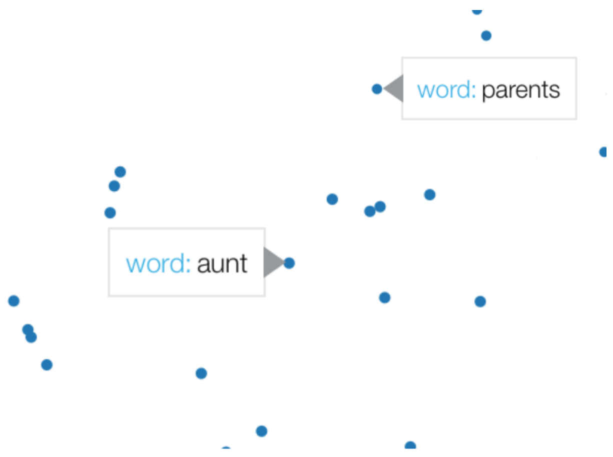


Figure 5: Zoom-in and click over two word representation to show the associated vocab words

5.2 Specific visualization:

This section describes the approach we have followed to analyze the relationship between individual word representations. The main goal of any word embedding scheme is to enclose the semantic similarity into the word representations. To verify this property on our embedding, different similarity measures where studied and the best fitted to our problem were chosen to analyze the behavior of the embedding algorithm when comparing between similar (i.e in the sense of words synonymy) words.

Very much similar to the case of the Word2Vec embedding, and because of the simple fact that the SSWE embedding uses also the context of words when learning the embedding along with the polarity of the sentences in which they appeared, **similar words tend to be close to each other in the sense of the geometric distance between them**. For that reason, it seemed necessary to measure the distance similarity between different words of our embedded vocabulary.

5.2.1 Cosine Similarity:

The cosine similarity is a widely used similarity measure of two non-zero vectors of an inner product space that measures the cosine of the angle between two vectors. A normalization of the input vectors can be used to get a range of cosine similarity output results between 0 and 1. Hence, the values close to 0 (i.e geometrically perpendicular when 0 is reached) refer to two completely uncorrelated vectors , this means in the case of the word embedding problem that the two words have no common semantic relationship. On the other hand, the more the two input vectors are positively correlated , the bigger the cosine similarity value is.

Mathematically, the normalized cosine similarity can be defined as follows:

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} \tag{6}$$

When testing on our word representation, we could be able to prove that the words' semantics learning was successfully done. In the following figures, we demonstrate some of the normalized cosine similarity measure results of some chosen emotion words:

5.2.2 The Multiplicative combination objective similarity:

The Multiplicative combination objective similarity measure was proposed by Omer Levy and Yoav Goldberg in [4]. And because it proved its efficiency to give relatively precise measures in other word embedding systems as for example the Word2Vec embedding, we have decided to use it to measure distance between different words.

6 Experiments and results

In this section we will illustrate the implementation details of the word embedding algorithm and the classification model, Afterwards, we will demonstrate and explain the results that we have obtained.

6.1 SSWE Model training

we train the Specific sentiment embedding model using the sentiment140 presented in the dataset section. In fact, the SSWE needs a huge train dataset in order to obtain large scale training corpora.
We used the **deepnl** library which implement the SSWE neural networks architecture. We used the script **dl-sentiwords.py** which allows creating sentiment specific word embedding. It uses essentially the **dl-conv.py** script which a convolutional neural networks with the same layers specified by SSWE model. Then we integrate them into our project as an Embedding module. Before using the sentiment140 dataset, We cleaned and pre-processed as it is described in the preprocessing section. We also remove remove the too short tweet which have less then 5 words.
The training parameters are described in the table6

Embedding size	epochs	Hidden layer	window size	learning rate
50	100	20	3	0.01

Table 6: SSWE training parameters

The outputs of the first model are two files with the same size:

- Vocabulary file: text file contains the vocabulary extracted from the dataset.
- Vector file: text file contains the embedding vectors of the words existing in the vocabulary file

6.2 Embedding evaluation

6.3 Visualization results:

As described in the last chapter, A **global visualization** was done as we wanted to study the distribution of our word representation. After training the SSWE model and constructing the words' feature vectors using **different vector space representation (i.e different size of embedding for the word representation: 200, 100, 50, 30)**. The visualization of our word embedding data in a 2D-space using t-SNE (described in details the last chapter can be shown in the figure [13]

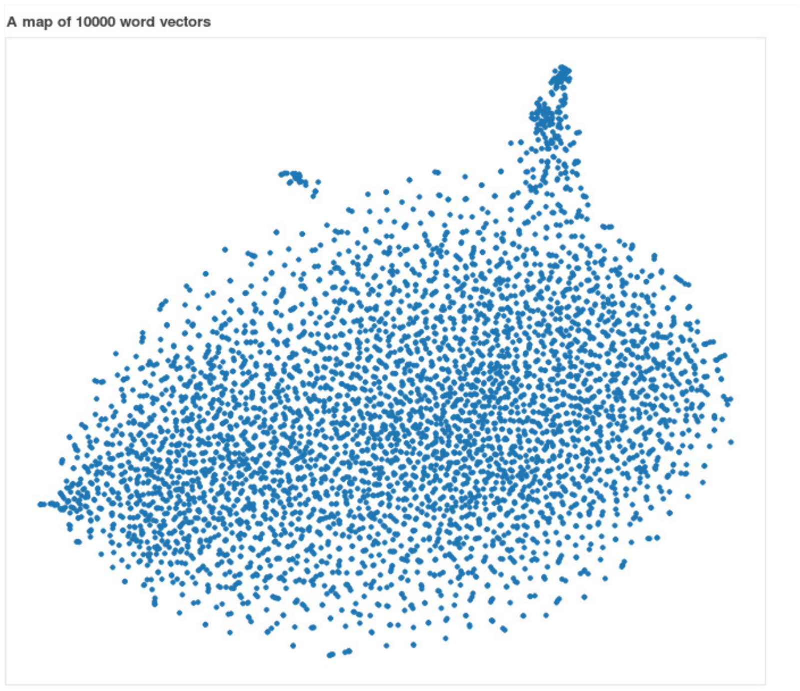


Figure 6: word representation distribution after being mapped from a vector space of size 50 to a 2D-space using t-SNE

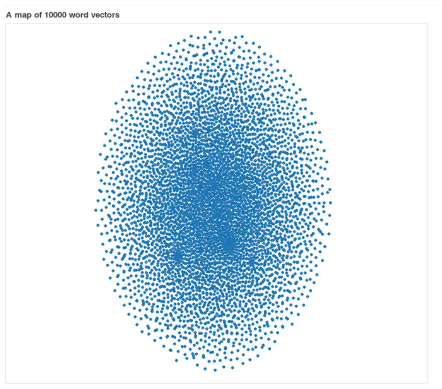
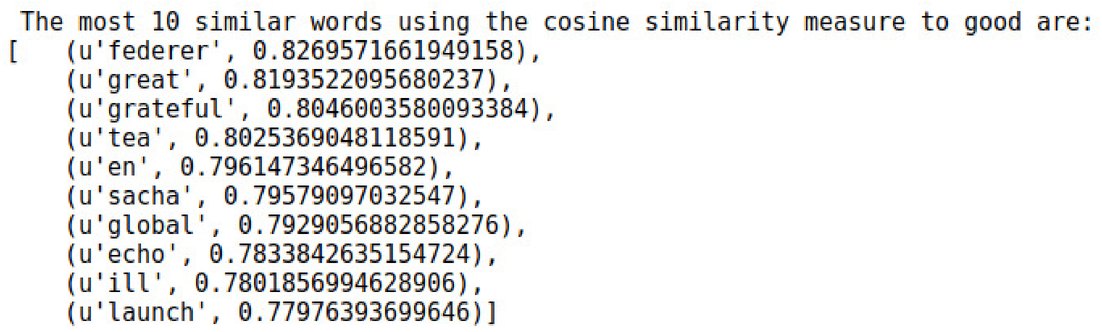


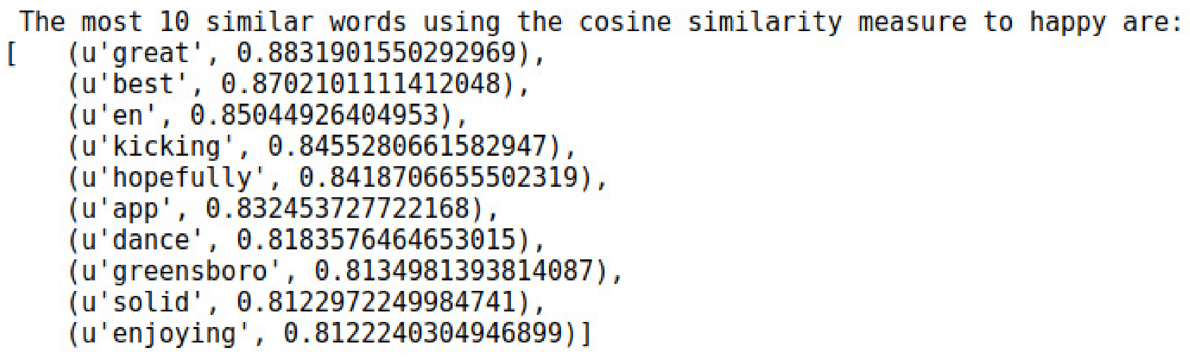
Figure 7: word representation distribution after being mapped from a vector space of size 200 to a 2D-space using t-SNE

We note that the general word representations distribution has a particular oval form for an embedding size of 200, it is more coarse in the representation of words in a vector space of dimension equal to 50. this latter representation presents more vectors density in the x-axis part of the manifold but also presents some special parts in its extremities. Given the fact that the initialization of the feature vectors before learning was done using pseudo-random values following the **Normal distribution**, we can prove that a learning of a model is done and the vector distribution doesn't follows anymore a normal distribution (i.e for a size of embedding of 50) like its initialization values. On the hand, it is insufficient to study our embedding only through studying the shape of the distribution and thus we needed to study the relationships between different words in the embedding space, for that reason, we have ,as described in the specific visualization of the last part, calculated for some keywords the cosine similarity of some chosen cosine similarities and here is a list of the most similar words to some of the keywords described in the next page.



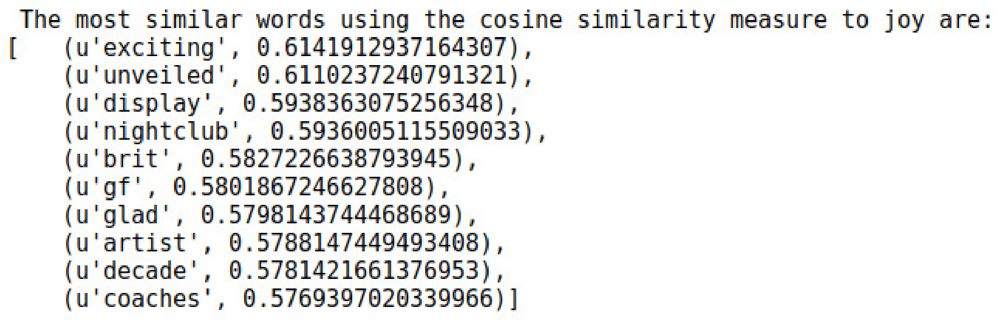
```
 The most 10 similar words using the cosine similarity measure to good are:
[   (u'federer', 0.8269571661949158),
    (u'great', 0.819352209568O237),
    (u'grateful', 0.8046003580093384),
    (u'tea', 0.8025369048118591),
    (u'en', 0.796147346496582),
    (u'sacha', 0.79579097032547),
    (u'global', 0.7929056882858276),
    (u'echo', 0.7833842635154724),
    (u'ill', 0.7801856994628906),
    (u'launch', 0.77976393699646)]
```

Figure 8: The most similar words using the cosine similarity of the word: **good** in **a vector space of dimension 50**



```
 The most 10 similar words using the cosine similarity measure to happy are:
[   (u'great', 0.8831901550292969),
    (u'best', 0.8702101111412048),
    (u'en', 0.85044926404953),
    (u'kicking', 0.8455280661582947),
    (u'hopefully', 0.8418706655502319),
    (u'app', 0.832453727722168),
    (u'dance', 0.8183576464653015),
    (u'greensboro', 0.8134981393814087),
    (u'solid', 0.8122972249984741),
    (u'enjoying', 0.8122240304946899)]
```

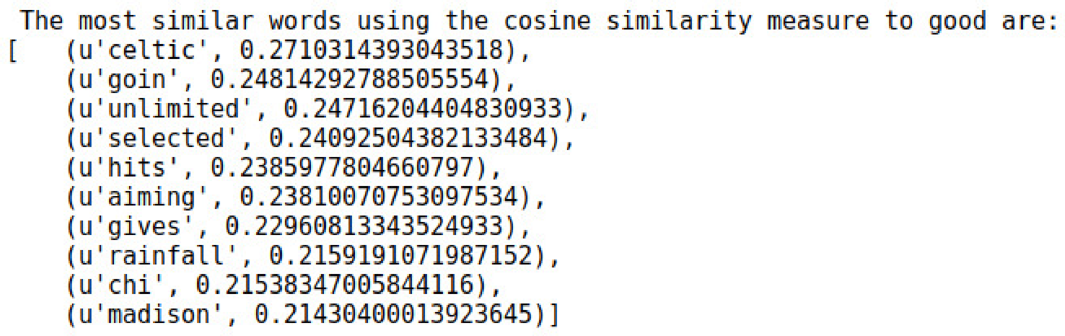
Figure 9: The most similar words using the cosine similarity of the word: **happy** in **a vector space of dimension 50**



```
 The most similar words using the cosine similarity measure to joy are:
[   (u'exciting', 0.6141912937164307),
    (u'unveiled', 0.6110237240791321),
    (u'display', 0.5938363075256348),
    (u'nightclub', 0.5936005115509033),
    (u'brit', 0.5827226638793945),
    (u'gf', 0.5801867246627808),
    (u'glad', 0.5798143744468689),
    (u'artist', 0.5788147449493408),
    (u'decade', 0.5781421661376953),
    (u'coaches', 0.5769397020339966)]
```

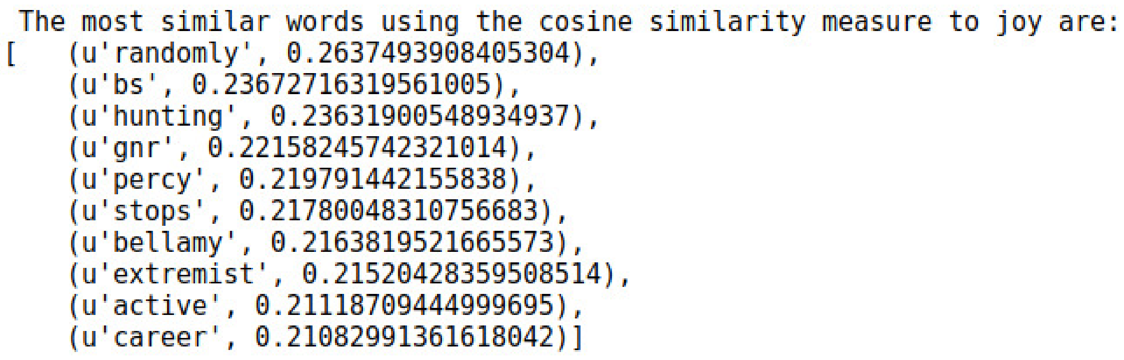
Figure 10: The most similar words using the cosine similarity of the word: **joy** in **a vector space of dimension 50**

The most similar words of some chosen keywords in a 200 dimensional vector space are presented as follows:



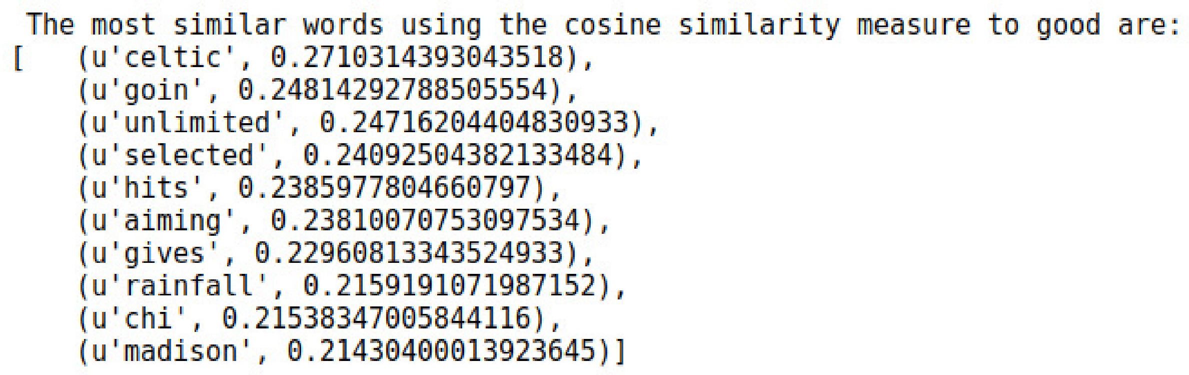
```
 The most similar words using the cosine similarity measure to good are:
[   (u'celtic', 0.2710314393043518),
    (u'goin', 0.24814292788505554),
    (u'unlimited', 0.24716204404830933),
    (u'selected', 0.24092504382133484),
    (u'hits', 0.2385977804660797),
    (u'aiming', 0.23810070753097534),
    (u'gives', 0.22960813343524933),
    (u'rainfall', 0.2159191071987152),
    (u'chi', 0.21538347005844116),
    (u'madison', 0.21430400013923645)]
```

Figure 11: The most similar words using the cosine similarity of the word: **joy** in **a vector space of dimension 200**



```
 The most similar words using the cosine similarity measure to joy are:
[   (u'randomly', 0.2637493908405304),
    (u'bs', 0.23672716319561005),
    (u'hunting', 0.23631900548934937),
    (u'gnr', 0.22158245742321014),
    (u'percy', 0.219791442155838),
    (u'stops', 0.21780048310756683),
    (u'bellamy', 0.2163819521665573),
    (u'extremist', 0.21520428359508514),
    (u'active', 0.21118709444999695),
    (u'career', 0.21082991361618042)]
```

Figure 12: The most similar words using the cosine similarity of the word: **joy** in **a vector space of dimension 200**



```
 The most similar words using the cosine similarity measure to good are:
[   (u'celtic', 0.2710314393043518),
    (u'goin', 0.24814292788505554),
    (u'unlimited', 0.24716204404830933),
    (u'selected', 0.24092504382133484),
    (u'hits', 0.2385977804660797),
    (u'aiming', 0.23810070753097534),
    (u'gives', 0.22960813343524933),
    (u'rainfall', 0.2159191071987152),
    (u'chi', 0.21538347005844116),
    (u'madison', 0.21430400013923645)]
```

Figure 13: The most similar words using the cosine similarity of the word: **joy** in **a vector space of dimension 200**

We can note very clearly that the semantic learning of the embedding gets much better when the size of the embedding is properly chosen. After trying different embedding sizes ranging from 30 to 250 (passing by 100, 150, 180, 200, 230). We can obviously deduce that a size of embedding of 50 is optimal to get the best embedding results in the sense of the semantics understanding of the embedded words.

In the annex pages, we have also visualized the most similar words to other keywords using the cosine similarity measure but also using the MCO similarity (Multiplicative combination objective similarity).

6.4 Twitter sentiment classification

Additionally to the visualization and the similarity measurement, we evaluate the obtained embedding by using it as features for a twitter sentiment classifier. We use the SemEval 2017 dataset, we have done a preprocessing step as described in the first section of this report, afterwards, we have trained two machine learning models, an **SVM classifier** using a linear Kernel and a **neural network classifier**, for its parameters namely, we have chosen 2 as the number of its layers and the the sigmoid function as the type of the activation function of its nodes.

6.4.1 Feature extraction:

For both models (the SVM and the Neural network classifiers), a feature vector of the tweet is constructed by **concatenating the word embedding** of its tokens, each token is multiplied by its computed **term frequency–inverse document frequency(tf-idf)** weight described as the product of these two functions $tf_{i,j}$ and idf_i :

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{7}$$

$$idf_i = \log\frac{|D|}{|d : t_i \in d|} \tag{8}$$

6.5 Results

For the training of the SSWE$_u$ embedding algorithm, we have tried different parametrization metrics, namely:

- The number of epochs of the convolutional neural network of the SSWE model. (N.b: we are not talking about the number of epochs of the classification neural network that we have used to later on to classify the tweets).
- The size of the embedding vector, hence the dimensionality of the embedding space.(As described in deep details in the part 8.1)

When we used the neural network, we tried to train the model using different epochs, so that, we can deduce what is the best epoch to use, in order to get the best accuracy results. The results of using a different epochs and the corresponding accuracies of the models constructed using these epochs are listed in the following table:

To evaluate our machine learning models (SVM and Neural network), using the optimal models' parametrization metrics of the models(e.g number of epochs for the neural network), we used these following metrics:

- The F1-score

Number of epochs	250	270	300	350	450
Accuracy	0,7950	0,7865	0,7813	0,7904	**0,80334**
mean square error	0,21172	0,20719	0,20404	0,203974	**0,195334**
mean absolute error	0,211728	0,20718	0,20388	0,203974	**0,195312**

Table 7: Accuracy results for different epochs' size of the classification neural network

- The Accuracy

In the following table [8], we report the obtained results:

Classifier	Macro-F1	Accuracy
SVM with a linear Kernel	0.71854	0,76423
2-layer Neural network (Sigmoid activation)	0.77342	0.80334

Table 8: F1-score and Accuracy results

7 Future work

First, we would like to further explore and exploit other methods to represent the feature vectors of the tweet (besides the concatenation and the summation) and see the impact on the classification results.
Also, how to learn the SSWE effectively with document level remains an interesting future work.

8 Conclusion

In this report , we implemented a learning continuous word representations as features for Twitter sentiment classification. We showed that the word embedding learned by the traditional neural network are not effective for twitter sentiment classification while adding neural gates boosts the performance. We trained the SSWE model with massive labeled tweets and use the generated vectors and vocabulary features to train a twitter sentiment classifier. Those features have been used later to train another classifiers we have built to measure word similarity in the embedding space for sentiment lexicons. Both classifiers have yield good performance measures.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[2] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[3] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(2009):12, 2009.

[4] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[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] Hassan Saif, Miriam Fernández, Yulan He, and Harith Alani. On stopwords, filtering and data sparsity for sentiment analysis of twitter. 2014.

[7] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *ACL (1)*, pages 1555–1565, 2014.