BUSEM at SemEval-2017 Task 4
Sentiment Analysis with Word Embedding and Long Short Term Memory RNN Approaches

Deger Ayata[1], Murat Saraclar [1], Arzucan Ozgur[2]
[1]Electrical & Electronical Engineering Department, Bogaziçi University
[2]Computer Engineering Department, Bogaziçi University
Istanbul , Turkey
{deger.ayata, murat.saraclar, arzucan.ozgur}@boun.edu.tr

Abstract

This paper describes our approach for SemEval-2017 Task 4: Sentiment Analysis in Twitter. We have participated in Subtask A: Message Polarity Classification subtask and developed two systems. The first system uses word embeddings for feature representation and Support Vector Machine, Random Forest and Naive Bayes algorithms for the classification of Twitter messages into negative, neutral and positive polarity. The second system is based on Long Short Term Memory Recurrent Neural Networks and uses word indexes as sequence of inputs for feature representation.

1 Introduction

Sentiment analysis is extracting subjective information from source materials, via natural language processing, computational linguistics, text mining and machine learning. Classification of users' reviews about a concept or political view may bring different opportunities including customer satisfaction rating, making right recommendations to right target, categorization of users etc. Sentiment Analysis is often referred to as subjectivity analysis, opinion mining and appraisal extraction with some connections to affective computing. Sometimes whole documents are studied as a sentiment unit (Turney and Littman, 2003), but it's generally agreed that sentiment resides in smaller linguistic units (Pang and Lee, 2008).

This paper describes our approach for SemEval-2017 Task 4: Sentiment Analysis in Twitter. We have participated in Subtask A: Message Polarity Classification subtask. We have developed two systems. The first system uses word embeddings for feature representation and Support Vector Machine (SVM), Random Forest (RF) and Naive Bayes (NB) algorithms for classification Twitter messages into negative, neutral and positive polarity. The second system is based on Long Short Term Memory Recurrent Neural Networks (LSTM) and uses word indexes as sequence of inputs for feature representation.

The remainder of this article is structured as follows: Section 2 contains information about the system description and Section 3 explains methods, models, tools and software packages used in this work. Test cases and datasets are explained in Section 4. Results are given in Section 5 with discussions. Finally, section 6 summarizes the conclusions and future work.

2 System Description

We have developed two independent systems. The first system is word embedding centric and described in subsection 2.1. The second is LSTM based and described in subsection 2.2. Further details about both systems are given in Section 3.

2.1 Word Embedding based System Description

In word embedding centric system approach, each word in a tweet is represented with a vector. Tweets consist of words and vectorial values of words (word vectors) are used to represent tweets as vectorial values. Word Embedding system framework is shown Figure 1. Two methods are used to obtain word vectors in this work. The first method is based on generating word vectors via constructing a word2vec model from semeval corpus as depicted in Figure 1 steps 1 and 2. The second

Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017), pages 777–783,
Vancouver, Canada, August 3 - 4, 2017.

method is based on Google News pre-trained word vectors model (step 3).

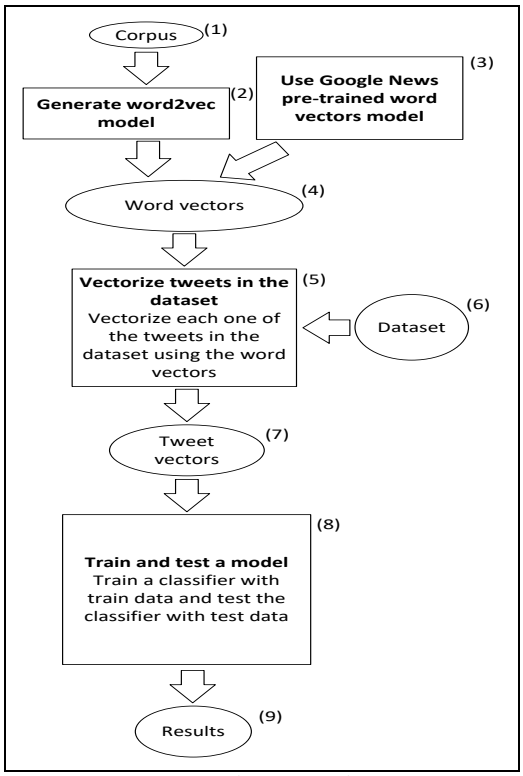

Figure 1: General framework of the system

In the first method, a word2vec model is construted by using the entire semeval tweet corpus and a vector space (word2vec model) has been created. This model contains vector values for all unique words in the corpus. Words which have similar contexts are positioned closer on this space. The parameters used in training word2vec model effect the performance of the whole framework. Therefore it is important to find optimal parameter values. This work is focused on the parameter named feature vector dimension size and its impact on the general performance. This parameter determines the dimensionality of the word vectors, which are generated via the word2vec model.

The second method is based on Google News pre-trained word vectors model. This method uses the Google News pre-trained word vectors model to obtain word vectors as shown in Figure 1 step 3. The Google news pre-trained model is a dictionary which contains word and vectorial value pairs, and it is generated via a word2vec model trained on the Google News text corpus.

Next stage includes using the obtained word vectors to vectorize tweets in the dataset which contains both training data and test data (steps 5 and 6). This stage includes also the preprocessing of the tweets, e.g. deleting http links and twitter user names included in the tweets, deleting the duplicate tweets which occur multiple times on the dataset etc. Later, preprocessed and formatted tweets are iterated to generate a tweet vector for each tweets by using the words they contained. Therefore, inputs of this stage are the dataset and the model which includes word vectors, while its output is a set containing tweet vectors, both for the train and the test data.

Outputted tweet vectors are in a numerical format which can be given as an input to multiple machine learning algorithms with the purpose of classifying them into categories or testing an existing model (step 8). At this stage, each tweet in the dataset is represented as a vector with multiple dimensions (step 7). It is possible to train a new classifier model or load a pre-trained and saved model. SVM, RF, and NB classifier models are trained in this work. Tweets are categorized into three classes which are negative, neutral and positive (step 9).

2.2 LSTM Based System Description

The pipeline of the second system consists of many steps : reading Tweets from Semeval datasets, preprocessing Tweets, representing each word with an index, then representing each Tweet with a set of word index sequence and training a LSTM classifier with sequence index array. The Flowchart of this system is shown in Figure 2.

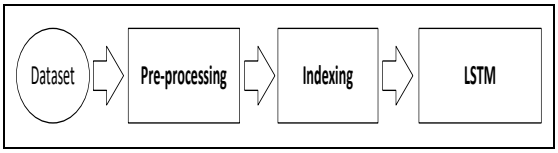

Figure 2: LSTM based system pipeline

3 Methods and Tools

3.1 Word Embedding

Word embedding stands for a set of natural language processing methods, where words or phrases from the vocabulary are mapped to vectorial values of real numbers (Bengio et al.,2003). Embeddings have been shown to boost the performance of natural language processing tasks such as sentiment analysis

(Socher et al., 2013). Vector representations of words can be used in vector operations like addition and subtraction. Vectors generated by word embedding can be used to represent sentences, tweets or whole documents as vectorial values. There are multiple methods to generate sentence vectors using the word vectors, a modified version of the sum representation method which is proposed by Blacoe is used in this work (Blacoe et al., 2012).

The sum representation model, in its original state, is generated via summing the vectorial embeddings of words which a sentence contains. The related equations are given below with E.1, E.2 and E.3:

Twt: *tweet*, $twtVec$: *tweet vector*,
w: *word*, $wdVec$: *word vector*,
n: *number of words in tweet*,
$Twt_i = \left(w_1{}^{(i)}, \dots, w_n{}^{(i)}\right)$: *words in tweet*

$$twtVec[j] = \sum_{k=1,\dots,n_i} wdVec_{w_k}[j] \qquad \text{(E.1)}$$

A modified version is used in this work. Derived version considers the number of words. The related equations are given below:

$$Twt_i = \left(w_1{}^{(i)}, \dots, w_n{}^{(i)}\right) \qquad \text{(E.2)}$$

$$twtVec[j] = \frac{\sum_{k=1,\dots,n_i} wdVec_{w_k}[j]}{n} \qquad \text{(E.3)}$$

3.2 Classification Models

3.2.1 Support Vector Machine

SVM finds a hyper plane seperating tweet vectors according to their classes while making the margin as large as possible. After training, it classifies test records according to which side of the hyperplane their positions are (Fradkin et al, 2000). We have used SVM with the following parameters = {Kernel = PolyKernel, batchSize=100}

3.2.2 Random Forest

Random forests, first proposed by Ho (Ho, 1995) and later improved by Breiman (Breiman, 2001), operate by generating multiple decision trees and generate the final decision by evaluating the results of these individual trees. The mathematical expression is given in equation (E.4). We have used Random Forest with the following parameters ={bagSizePercent =100, batchSize=100}

$\{(x_i, y_i)\}_{i=1}^{n}$: *training set*,
$y*$: *predictions*,
x' : *new points to classify*,
$W(x_i, x')y_i$: *weight of the i'th function*,
W : *weight function*,

$$y^* = \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n} W_j(x_i, x')y_i = \sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=1}^{m} W_j(x_i, x')\right)y_i \qquad \text{(E.4)}$$

3.2.3 Naïve Bayes

Naïve-Bayes is a probabilistic classifier based on Bayes' theorem, based on independence of features (John et al, 1995). Mathematical expression is given in equation (E.5).

$c*$: *predicted class* x : *sample*

h_{NB}: *naïve bayes func*
$c^* = h_{NB}(x)$

$$= argmax_{j=1\dots m} P(c_j)\prod P(X_i = x_i|c_j) \qquad \text{(E.5)}$$

3.2.4 Long Short Term Memory Recurrent Neural Nets

LSTM networks have similiar architecture to Recurrent Neural Nets (RNNs), except that they use different functions and architecture to compute the hidden state. They were introduced by Hochreiter & Schmidhuber (1997) to avoid the long-term dependency problem and were refined and popularized by many people in next studies.

LSTMs have the form of a chain of repeating modules of a special kind of architecture. The memory in LSTMs are called *cells*. Internally, these cells decide what to keep in and what to erase from memory. They then combine the previous state output, the current memory and the input to produce current state output. It turns out that these types of units are very efficient at capturing

long-term dependencies. Before feeding the LSTM Network, preprocessing and indexing steps have been applied as shown in Figure 2.

Preprocessing

We have pre-processed the dataset before we input it into the LSTM classifier. We used Deeplearning4J library[1] to remove punctuations from tweets, and convert all content into lowercase.

Indexing

Indexing is iterating over all tweets contained in the dataset to determine words used in them and enumerate them. The index values of words are combined sequentially so that each tweet is presented as a sequence of word index numbers.

The program iterates through the dataset, enumarates each word which has not been indexed before and generates a dictionary that contains word – index pairs. As a result, each tweet is represented as set of sequential indexes, each representation contains same number of values as the tweet word count.

Indexed tweets are in sequential structure and they can be given as input to neural networks directly. LSTM networks make it possible to take the data sequentially and take in consideration the order of words in the training and classifying stages. Therefore, we used LSTM upon indexed tweets. We have used categorical crossentropy as loss function and softmax function. Our model parameters are given in Table 1 and the model is shown in Figure 3.

3.3 Used Tools and Software Packages

3.3.1 Deeplearning4J

Deeplearning4j is a commercial-grade, open-source, distributed deep-learning library written for Java and Scala[1]. There are multiple parameters to adjust when training a deep-learning network. Deeplearning4j is used for generating a vectorized format of the Semeval dataset using the Google News trained word vectors model.

Table 1: Parameters for classifier stage

`max_features`	86000 : Maximum integer value of indexed dataset.
`maxlen`	25 : Indexed tweets padded into this value.
`batch_size`	32
`model`	Sequential(): sequential model
`Embedding`	max_features : Input dimension, size of the vocabulary. 128 : Dimension of the dense embedding. dropout=0.2
`LSTM`	128 : dimension of the internal projections and the final output dropout_W=0.2 : Fraction of the input units to drop for input gates. dropout_U=0.2 : Fraction of the input units to drop for recurrent connections.
`Dense`	3 : Output dimensions.
`Activation`	'softmax' : Normalized exponential function
`loss`	'sparse_categorical_crossentropy' :.
`optimizer`	'adam' : Adam optimizer.

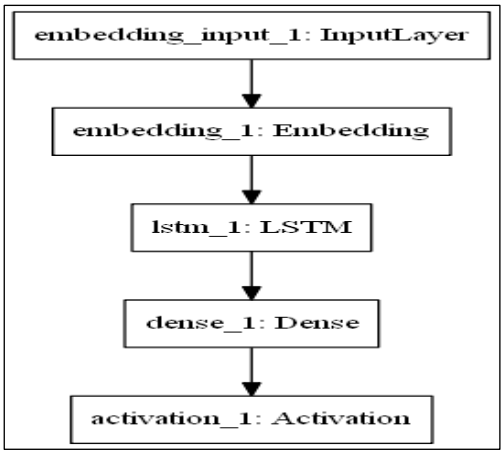

Figure 3: Plotted diagram of the LSTM classifier

3.3.2 Keras

Keras[2], developed by Chollet et al., is a high-level, open-source neural networks library written in Python (Chollet, 2015). It can use Theano or TensorFlow libraries as its backend. It is focused on fast experimentation with data. It includes implementations of commonly used neural network blocks such as layers, activation functions, etc., to enable its users to implement various neural networks in their work. We have used Keras library to develop LSTM network for tweet polarity classification.

[1] Deeplearning4j, http://deeplearning4j.org (Referenced Nov 2016)

[2] Keras, https://keras.io/ (Referenced February 2017)

3.3.3 Word2vec

Word2vec[3], is a group of models used to generate word embeddings (Mikalov et al., 2013). Word2vec models are based on two-layer neural networks which takes a large corpus of text as its input and produces a vector space of several hundred dimensions. Each unique word in the corpus is assigned a corresponding vector in this space. Embeddings are used to represent words as vectors which are closer to each other when words have similar meanings and far apart when they do not. Therefore, the system can generalize similar words.

Representing words as vectorial values makes it possible to treat words as vectors and use vectorial operations on them. A properly trained Word2vec model can calculate an operation like *[king] - [man] + [woman]* and give the approximate result of *[queen]*.

We have generated word2vec models in our tests from semeval datasets. We have run word2vec models with the parameters shown in Table 2.

Table 2: Parameters for word2vec generation stage

`minWordFreq`	MIN_WORD_FREQ = 5 : Minimal element frequency for elements found in the training corpus.
`iterations`	NETWORK_ITERATION = 25/50 :How many iterations should be done over batched sequences.
`layerSize`	FEATURE_VECTOR_DIMENSION_SIZE = 300/600/900 : Number of dimensions for outcome vectors.
`seed`	RANDOM_SEED = 42 : Sets seed for random numbers generator.
`windowSize`	WINDOW_SIZE = 25 : Sets window size for skip-Gram

3.3.4 Google News Trained Word2vec Model

Google news trained word vectors compose a word vector model which has been pre-trained on part of Google News corpus that includes 100 billion words. The model contains 300-dimensional vectors for 3 million words and phrases[4]. It is 3.39 GB in size which is observable from the equality, *3 million words * 300 features * 4bytes/feature = ~3.39GB*. Some stop words like "a", "and", "of" are excluded, but others like "the", "also", "should" are included. It also includes misspellings of words. For example, it includes both "mispelled" and "misspelled".

We have used Google News pre-trained word vectors to generate vector representations of tweets with `FEATURE VECTOR DIMENSION SIZE` equals to 300 configuration.

4 Dataset and Test Cases

4.1 Dataset

SemEval-2016 Task4 Subtask A's twitter train and test datasets have been used in this work[5]. The given datasets are dynamic which don't include the tweets that are deleted by their authors. Thus, the available data changes dynamically as users make their tweets available or deleted. We have used all previous years' tweets to construct the word embedding and classification models.

4.2 Test Cases

We have tested many configurations to find the best configuration to achieve the highest accuracy rate. We have conducted five main test cases. In the first, second and third test cases we have used word2vec model that has been constructed with previous years' semeval tweet datasets.

In Test 01, Test 02 and Test 03 we have trained word2vec model with SemEval Tweet dataset corpus. Also we have used different vector dimension sizes including 300, 600 and 900. In test case 04 we have used google news based(trained) word vectors. Also in each test case classification has been done with SVM, RF and NB. Test 05 id done with LSTM classifier on SemEval dataset. The test cases are listed in Table 3.

Table 3: Results obtained from tests

Test No.	Word Vectors	Dimension Size	SVM	RF	NB	LSTM
01	SemEval	600	√	√	√	
02	SemEval	300	√	√	√	
03	SemEval	900	√	√	√	
04	Google News trained	300	√	√	√	
05	N/A (index)	30				√

[3] Word2vec https://deeplearning4j.org/word2vec.html

[4] Google News trained word vectors model, https://code.google.com/archive/p/word2vec/

[5] SemEval Dataset, alt.qcri.org/semeval2017/task4/index.php?id=data-and-tools

5 Results and Discussion

5.1 Tests with Word Embedding and SVM, TF, NB

Purpose of the first three tests was observing the parameter feature vector dimension size's and classifier type impact on the general performance. We have used SemEval training and test datasets pertaining to 2013, 2014, 2015 and 2016 years to construct SemEval word2vec model. The tests have been done using this SemEval cumulative dataset. Results obtained from the classification tests are shown in Table 4.

Table 4: Accuracy / Results obtained from tests

Test No.	Word Vectors	Dim. Size	SVM %	RF %	NB %
01	Semeval	600	58.3	54.4	52.3
02	Semeval	300	57.3	45.7	51.8
03	Semeval	900	58.7	53.7	51.7
04	Google News trained word vectors	300	62.8	57.2	53.1

For SVM, the difference is minimal, but the value 900 worked best. For RF, the value 300 drastically reduced the overall performance while the value 600 worked best. NB accuracies are close to each other but it is observed that this method has the lowest overall accuracy values among three. With a word2vec model which is trained on the same dataset with the classifier, SVM method obtained the best results.

The fourth test has a different approach, which is not generating a word2vec model but obtaining the Google News pre-trained word vectors instead. This model has the standard value 300 for the feature vector dimension size and resulted in better accuracies for each one of the classification methods. It is observed that the model has positive impact on the overall system performance.

5.2 Tests with LSTM

Keras library is used to train and test LSTM Recurent Neural Net. Test 05 id done with LSTM classifier on SemEval cumulative dataset and 62.6% accuracy rate has been achieved.

5.3 Results over the SemEval 2017 Test Set

The test dataset is used to test the system's capability of predicting categories for unlabeled tweet data, and give them as an output. The original test dataset includes 12379 records, 95 of which are confirmed to be duplicates. These duplicate records are deleted from the dataset. Remaining 12284 records are evaluated in this test.

Preprocessing stage strips all punctuation from the dataset and converts all tweets into lower case. This means, twitter user names, e.g. @username, are stripped from their '@' symbol, but the user names themselves are preserved.

In SemEval 2017, the results are given with three scores: average F_1 (F_1 averaged across the positives and the negatives), average R (recall averaged across the three classes) and accuracy. The F_1 score measures test accuracy by considering precision and recall where a F_1 score reaches its worst value at 0 and best value at 1.

Using SemEval 2017 test data we have achieved the following scores : Average F_1 = 0.587, Average R = 0.605 and Accuracy = 0.603.

6 Conclusion and Future Work

The best result is obtained via support vector machine classifier, when Google News pre-trained word vectors are used, which is 62.8% accuracy in average when applied to previous years' training and test data.

On the Semeval 2017 Test Dataset by using same Word embedding + SVM pipeline (the first system), we have obtained 60.3% accuracy rate with the following scores scores : Average F_1 = 0.587, Average R = 0.605 and Accuracy = 0.603.

There may be many approches to create a better system. One possible way to further improve our system could be to transfer word embedding features to other classifiers (Recurrent Tensor Neural Networks, combining LSTM and Convolutional Neural Networks etc.). Another possible line of the future research is the combination of hand crafted features (bag of words, n-grams, lexicons) with word embedding features.

References

B. Pang, L. Lee, S. Vaithyanathan. 2002. *Thumbs up? Sentiment Classification using Machine Learning Techniques*, Proceedings of EMNLP 2002, pp. 79–86.

B. Pang and L. Lee. 2008. *Opinion mining and sentiment analysis*. Foundations and Trends in Information Retrieval.

Turney, P. & Littman, M. 2003. *Measuring praise and Criticism: Inference of semantic orientation from association.* ACM Transactions on Information Systems, 21(4), 315-346.

Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, "*A Neural Probabilistic Language Model*, Journal of Machine Learning Research 3 1137–1155, 2003

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, C. Potts, *"Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank"*, Conference on Empirical Methods in Natural Language Processing, 2013

W. Blacoe, M. Lapata, *"A Comparison of Vector-based Representations for Semantic Composition",* Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 546–556, 2012

T. Mikolov, K. Chen, G. Corrado, J. Dean, *"Efficient Estimation of Word Representations in Vector Space"*, 2013

D. Fradkin, I. Muchnik, *"Support Vector Machines for Classification"*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 2000

T. K. Ho, *"Random Decision Forests",* ICDAR '95 Proceedings of the Third International Conference on Document Analysis and Recognition Vol.1, 1995

G. H. John, P. Langley, *"Estimating Continuous Distributions in Bayesian Classifiers"*, Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, 1995

S. Hochreiter and J. Schmidhuber, *"Long Short Term Memory", Neural Computation*, vol. 9, no. 8, p. 1735–1780, 1997

F. Chollet, *"Keras: Deep Learning library for Tensor Flow and Theano"*, GitHub, https://github.com/fchollet/keras, 2015

L. Breiman, (2001). *"Random Forests". Machine Learning*. 45 (1): 5-32. doi:10.1023/A:1010933404324.