Name: Truong Giang Khang

Email: khangtg.hust.it@gmail.com

Detecting Sara Comment Problem

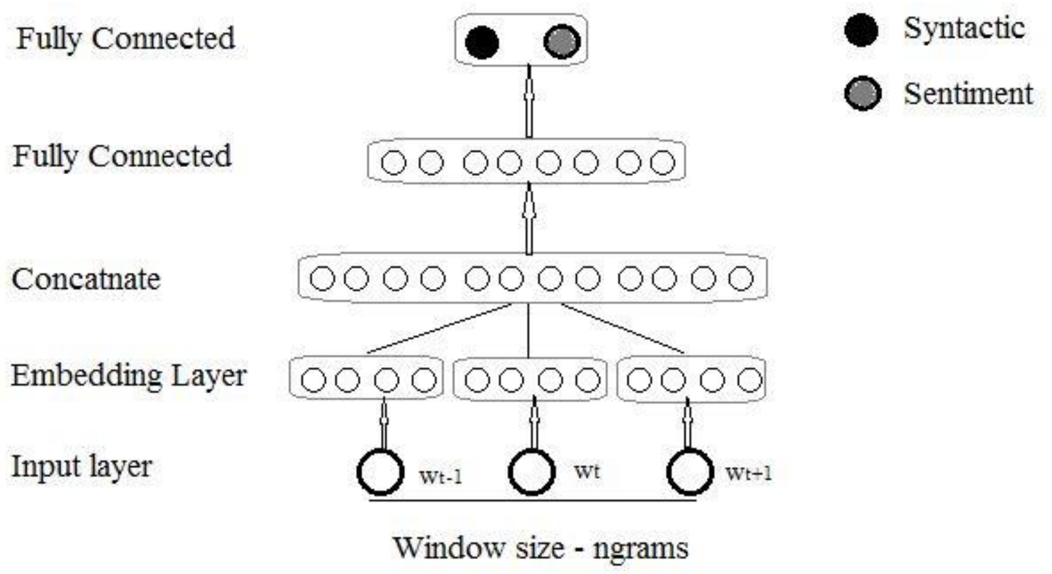
1. Model

There exists many models for solving the document classification problem. Today, I have based on a model proposed by Tang at el [1] for this problem. First, I use the SSWE (Sentiment Specific Word Embedding) model to learn the word embedding and use it to represent a feature vector for each document (comment). After that, we'll train a SVM classifier to do the classification task.

1.1. Word Embedding

Word2Vec [2] is a popular model for learn word embedding, but in this case, it maybe not efficient because of the small training dataset. SSWE is a neural network model which is originally used for sentiment analysis for twitter data and this is quite similar to the comments data in Babe app. An interesting point of SSWE is that it utilities the label information to learn a good representation for word.

The architecture of SSWE as follow:



Intead of using whole sentence (comment) as input for neural network, SSWE uses a window approach to create a input ngrams with fixed size – window size. Window approach assumes that the sentiment label of a word depends mainly on its neighboring words. So, the input layer take a ngram and map each word to a embedding vector which represented in embedding layer.

Each word is represented by a unique feature vector and if corpus have a vocabulary V words, we'll have a lookup table L with size VxK to map each word to a vector K-dimentional. Our task is that to learn this lookup table. We will concatenate the embedding vectors into a single vector represented for window. The following layers are standard Neural network layers, the current layer is fully connected to the next layer. We have the formula maths for SSWE as following

With a ngram t, Lt is the representation vector for ngram after the concatenation step.

$$a = hTanh(w_1 L_t + b_1) \quad (1) \qquad \text{with } hTanh(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \le x \le 1 \\ 1 & \text{if } x > 1 \end{cases}$$

$$f^u(t) = w_2 a + b_2 \quad (2)$$

f^u is the 2-dimentional output of the network (f_0^u, f_1^u) . f_0^u stands for languge model score and f_1^u stands for sentiment/semantic score of the input ngram. One problem is that how we can define a loss function for them. This solved by constructing a corrupted ngram from original ngram t. Assume that ngram t has the center word w_t , we will replace this center word with a random word w_r and derives a corrupted ngram t^r . The training objective is that the original ngram is expected to obtain a higher language model score than the corrupted ngram and the sentiment score of original ngram should be more consistent with truth sentiment label of sentence then corrupted ngram. So, the syntactic loss function as follow:

$$\text{loss}_{\text{syntactic}} = \max(0, 1 - f_0^u(t) + f_0^u(t^r))$$

When we minimize this loss, it means that $f_0^u(t) - f_0^u(t^r)$ will be very close to 1 or it is always greater than 1 and the syntactic score of ngram t will always be higher than the corrupted ngram t^r .

The sentiment loss function can be defined as follow:

$$\text{loss}_{\text{sentiment}} = \max(0, 1 - \delta_s f_1^u(\text{t}) + \delta_s f_1^u(\text{t}^r)) \quad \text{where } \delta_s = \begin{cases} 1 & \text{if sentiment is normal} \\ -1 & \text{if sentiment is sara} \end{cases}$$

So, when we're trying to minimize this loss:

- if the original ngram t is normal sentiment then $f_1^u(t) - f_1^u(t^r)$ is always greater than 1 or very close to 1, it means that the sentiment score of t is always high if t is a normal sentiment.
- if t is a sara sentiment, $f_1^u(t) - f_1^u(t^r)$ will be lower than -1 or it will be very close to -1. So, the sentiment score of sara ngram is always low.

Therefore, the word embedding learnt will capture all the property above. The total loss function of SSWE$_u$ is the linear combination of two hinge losses:

$$\text{loss}_u(t, t^r) = \alpha * loss_{syntactic} + (1-\alpha) * loss_{sentiment}$$

The objective training function will be:

$$J = \sum_{t \in D} \sum_{x \in neg-samples} loss(t, t^x)$$

D is the whole training ngrams and neg-samples is the negative samples for each ngram t. The negative samples is a set of words which be chosen randomly from vocabulary.

To minimize J, the back propagation algorithm in neural network will be used

1.2. Classification

After we learn the word embedding, we will use that to represent a feature vector for each comment and use SVM to do classification task. The feature vector of each comment is constructed by concatenating the word embedding of its words. Each word is multiplied by its computed tf-idf (tem frequency inverse document frequency). Tf-idf is also a popular representation for document and the tfidf of word w in document d will be computed as follow:

$$tf_{w,d} = \frac{\text{frequency of word w in document d}}{\text{total words in document d}}$$

$$idf_w = \log \frac{\text{total documents}}{\text{number of documents that word w belong to}}$$

$$tfidf_{w,d} = tf_{w,d} * idf_w$$

So if vocabulary size is V and word embedding size is K, then each comment will be represented by a sparse vector K*V.

2. Experiment and results

2.1. Requirements

I write a Python3 code to implement the model above. Some python packages required are: numpy, scipy, nltk, scikit-learn, tensorflow, keras, gensim

Computer must have the RAM $\geq$ 8 GB and the OS is ubuntu

2.2. Preprocessing

First, I need to preprocessing the raw dataset. This task includes: tokenizing, removing stop words and words is not in the vocabulary. One more problem is the original vocabulary includes 202115 words, but there are some words which don't appear in the training and test dataset. So to decrease computation, I filtered and extract a real vocabulary including 25773 words.

After preprocessing step, there are some comment will be removed because all words of it is the stop words or the words not appear in the original vocabulary. The new information about dataset is:

- Training data: 58972 normal comment and 14810 sara comment. Total: 73872
- Test data: 6623 normal comment and 2798 sara comment. Total: 9421

2.3. Running code

First, preprocessing data by:

```
python3 preprocessing.py
```

This program will extract 3 file: real_vocab.txt, training_data_real_vocab.txt, test_data_real_vocab.txt (**Do not** rename these file)

After that, running SSWE to build up word embedding:

```
python3 sswe.py [window_size] [embedding_size]
```

```
(for example: python3 sswe.py 3 50)
```

The word embedding will be saved in binary file 'word_embedding.npy'. And finally, we can run classification by SVM:

```
python3 classifier.py [word_embedding_file]
```

```
(python3 classifier.py word_embedding.py)
```

You also can try run word2vec [2] to create word embedding by:

```
python3 word2vec.py [window_size] [embedding_size]
```

The result is saved in file 'word2vec.npy'. I implement word2vec by using gensim package. After that, you can trying classification by SVM is similar to above.

You also try visualization words by:

```
python3 visualization.py [word_embedding_file]
```

2.4. Result

Setting for the parameters:

- the size of word embedding: 50
- number of hidden units of Dense layer: 20
- learning rate: 0.01 and using Adagrad to optimize loss function
- number of epochs = 100 and batch_size = 10000.
- Number of negative samples per word: 15
- α = 0.5 (parameter for the linear combination of losses function)

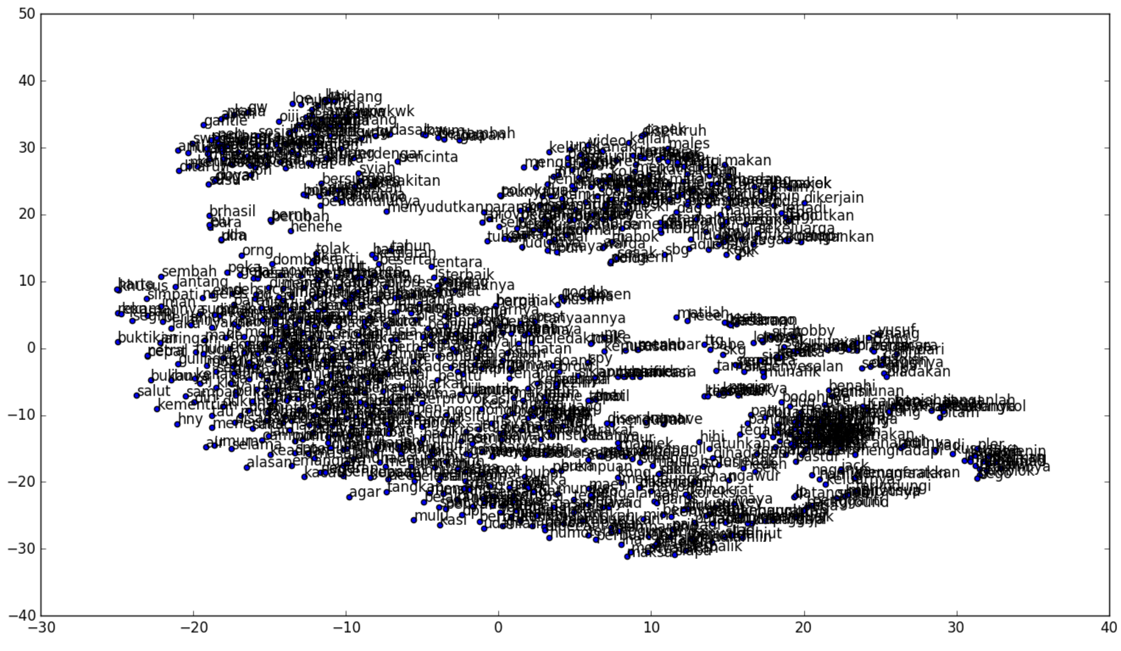
This table is results when changing window size:

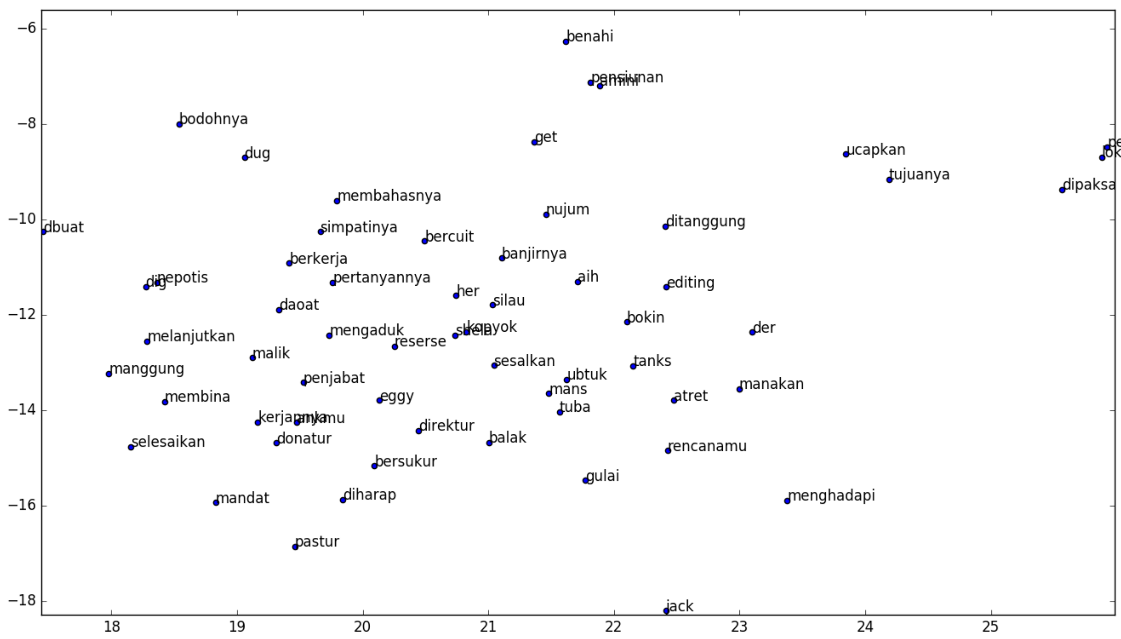
Window size	Macro-Precision	Macro-Recall	Macro-F1	Accuracy
3	**86.52%**	**79.76%**	**82.11%**	**86.32%**
5	85.97%	79.67%	81.90%	86.1%
7	85.76%	79.44%	81.67%	85.94%

I also test with word2vec and the result isn't good as follow:

Macro-Precision	Macro-Recall	Macro-F1	Accuracy
54.49%	**50.09%**	**41.82%**	**70.19%**

Visualization for 1000 first word in vocabulary with word embedding learnt by SSWE:





3. Reference

[1] Tang, Duyu, et al. "Learning sentiment-specific word embedding for twitter sentiment classification." Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2014.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013