Adil Mukhtar adilmukhtar@fccollege.edu.pk

Machine Learning Methods for Sentiment Analysis and Emotion Detection

Given dataset *D* which contains *N* number of tweets where each tweet $x_i \in D$ have label y_i where $y_i \in \{0,1,2,3\}$, a word embedding is learned through skip-gram model using *word2vec* available in *gensim.* It is known that skip-gram model works better for smaller dataset than CBOW. As CBOW learns the probability of word given context. In case of small dataset, there will be more rare cases, and CBOW won't be able to predict a word correctly. On the contrary, skip-gram model is designed to learn the context. Given a target-word, it must know in which context it fits.

Before implementing *word2vec*, little bit of preprocessing is done, which is to deEmojify the tweet and removal of stop words. Following are the hyper parameters for *word2vec* model.

```
model = Word2Vec(tokenized, min_count=3, size = 300, negative
         = 10, workers = multiprocessing.cpu_count())
```

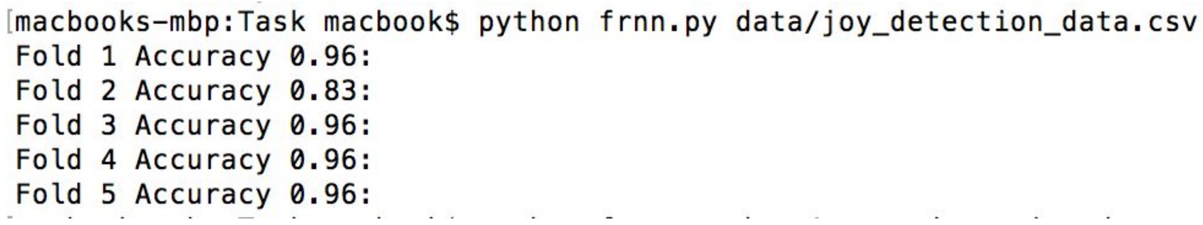
Each word has learned coefficients $c_k \; where \; k \le K$ in above setting K=300 (hidden layer neurons). After learning word embeddings, data matrix is reconstructed in order to represent $\forall x_i \in D$ tweets in terms of learned coefficients.

Given W_L words of dataset *D*, reconstructed datamatrix $X_{N\times L} \; where \; N = Total \; number \; of \; tweets, \; L = \; Total \; number \; of \; words$ is defined as:

$$x_{n,l} = \frac{\sum_{k=1}^{K} c_{l,k}}{K}$$
$$where \; n \; : n^{th} \; tweet \, , \; k \; \le \; K \; and \; l : l^{th} \; word$$
$$c_{l,k} : k^{th} \; coefficient \; of \; word \; (l) \;\; \in x_n$$

Also target column $Y_{n,} \; n \le N$ appended as column vector into data matrix X. Now each tweet is mapped onto words dimensional space. This data matrix numerical representation will help us in calculating fuzzy tolerance relation between two objects $R(x,y)$, as previously the length of each tweet was different. When two vectors have different lengths e.g. strings then we can calculate partial similarity. But in our case we have numerical vector of each tweet so I chose to calculate fuzzy tolerance relation $R(x,y)$, which can also be used to calculate the degree of similarity of two objects given attributes by calculating normalized distance of two points and subtracting from 1 (*1-normalized_distance*). Also FRNN is dependant on $R(x,y)$.

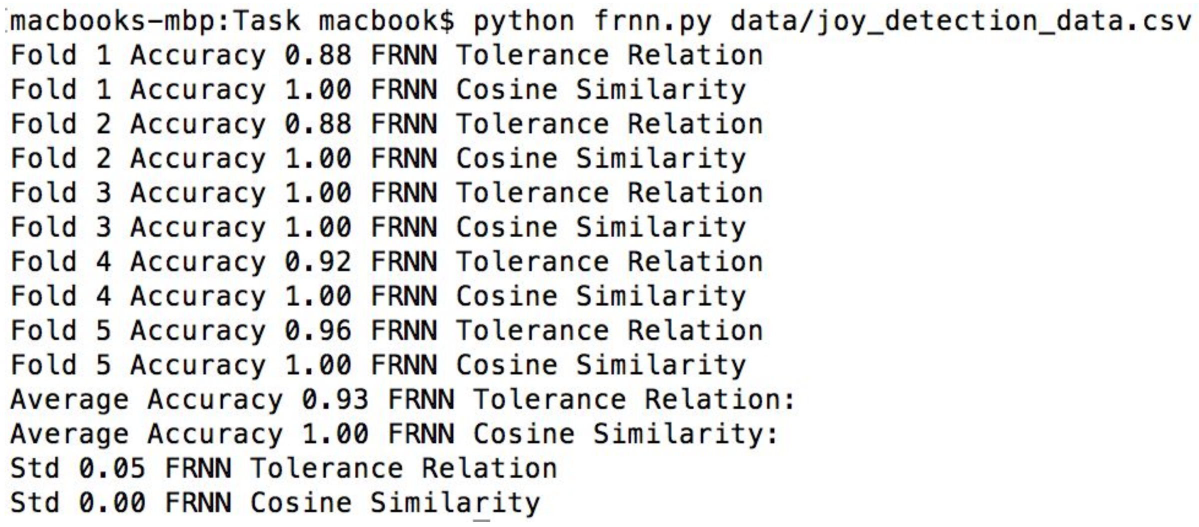
Following are the results of 5-fold cross validation when tolerance relation $R(x, y)$ is used:



```
macbooks-mbp:Task macbook$ python frnn.py data/joy_detection_data.csv
 Fold 1 Accuracy 0.96:
 Fold 2 Accuracy 0.83:
 Fold 3 Accuracy 0.96:
 Fold 4 Accuracy 0.96:
 Fold 5 Accuracy 0.96:
```

Mean is 0.85 and standard deviation is 0.052.

Results with cosine similarity also calculated in the python script, results are also given below.



```
macbooks-mbp:Task macbook$ python frnn.py data/joy_detection_data.csv
Fold 1 Accuracy 0.88 FRNN Tolerance Relation
Fold 1 Accuracy 1.00 FRNN Cosine Similarity
Fold 2 Accuracy 0.88 FRNN Tolerance Relation
Fold 2 Accuracy 1.00 FRNN Cosine Similarity
Fold 3 Accuracy 1.00 FRNN Tolerance Relation
Fold 3 Accuracy 1.00 FRNN Cosine Similarity
Fold 4 Accuracy 0.92 FRNN Tolerance Relation
Fold 4 Accuracy 1.00 FRNN Cosine Similarity
Fold 5 Accuracy 0.96 FRNN Tolerance Relation
Fold 5 Accuracy 1.00 FRNN Cosine Similarity
Average Accuracy 0.93 FRNN Tolerance Relation:
Average Accuracy 1.00 FRNN Cosine Similarity:
Std 0.05 FRNN Tolerance Relation
Std 0.00 FRNN Cosine Similarity
```

Other metrics like precision and recall can be used in order to check that the model is not biased towards any decision class.

As our goal is to classify the test object, given universe of objects, by calculating averaged upper and lower approximation. If averaged upper and lower approximation is greater than or equal to the previous one then we update the decision class and τ, means testing object is more close the training object.

Classification algorithm can be improved through following suggestions:

1. We can use machine learning model instead of crisp set (*A)* for equation (8) and (9) along with fuzzy tolerance relation in order to calculate upper and lower approximation for test object. As machine learning models can learn complex patterns in a given data. We can predict the probability of the decision class of testing object and then use the t-norm and implicator with correspondence to fuzzy tolerance relation.

2. Using optimal hyper parameters to generate word embeddings and then modelling those word embeddings using deep learning algorithm (RNN, LSTM etc).
3. We can use weighted K nearest neighbour or Gaussian mixture model with Expectation maximization for soft assignments and then use it instead of crisp set (0 or 1).
4. Rather than calculating fuzzy tolerance relation of testing object with all the given object (training), we can find the centroid of given universe of objects using word embeddings and calculate the tolerance relation of each testing object with centroid vector of the universe only. Then find the upper and lower approximations. It might improve the efficiency.
5. We can use norms to calculate the similarity (1-distance) e.g. euclidean, manhattan or minkowski instead of fuzzy tolerance relation.

***Note:** I have implemented the cosine similarity measure also. By default python (frnn.py) file runs for fuzzy tolerance relation (*R(x,y)*). If you want to run for cosine similarity measure just comment the function call line for fuzzy tolerance relation and uncomment the cosine similarity in loop in main code, where FRNN function call is made.