SNA STAGE 3

Pre-processing of text corpus:

1. Removed all punctuations, smileys and other unnecessary characters
2. Removed stop words like ‘a’, ‘and’, ‘is’ etc.
3. Did not remove ‘not’ even though it is a stop word since removing not changes the meaning of the sentence which we cannot afford for our sentiment analysis task
4. Cleaned tweets to remove @ mentions, hashtags and URLs.
5. Replaced words like aren’t with are not, wasn’t with was not since presence or absence of not plays an important role in classification of tweet as positive or negative.

Node2Vec:

1. Used only 70067 sentences as input file since node2vec runs very slowly for large files
2. Built edge list from given sentences since node2vec requires graph as input -> pte.py -> ww_making()
3. The edge list was built using PTE as:
    a. Put the above 70067 lines in text_train.txt -> file using which edge list is constructed.
    b. Labels were available for only 42k lines. So, we create a label file having same no of lines as text_train.txt and containing arbitrary labels. Tere is no need for availability of labels for all lines since building edge list does not use labels. (Infact whole Node2Vec is unsupervised)
    c. Keep all 178k lines in text_all.txt - does not matter even if you have only 1 line here since this file is not used in construction of edge list.
    d. Run PTE. The word-word network will be written in a file called ww.net
    e. Using above ww.net file, we construct our edge list by giving each word a unique id which serves as our node id in the edge list.
4. Run node2vec on the above edge list file to get word embeddings.

5. Construct sentence embeddings for the text_train.txt file by taking average of the word embedding vectors of each sentence.
6. Code link: https://github.com/aditya-grover/node2vec

PTE:

1. Used different amount of data to train PTE: Divided labelled dataset into full or ½ or 1/3 partitions and tried to compare the results.
2. Inferred document vectors for all sentences in text_all.txt (contains 178k lines) by running PTE.
3. Also, inferred document vectors for above text_train.txt file (of 70067 lines) so that we can compare PTE and node2vec more accurately and there is fair competition between the two.
4. Code link: https://github.com/mnqu/PTE

We did 2 types of evaluations of all the above word embedding methods:

- Extrinsic Evaluation
- Intrinsic Evaluation

Also, we compare and critically analyze the embedding methods according to values obtained using above evaluations.

Extrinsic Evaluation:

1. Used above document vectors as features for 2 different classifiers for sentiment analysis task.
2. Only those document vectors corresponding to sentences having labelled data are used for training the classifier.
3. Testing is done using 5-fold cross validation.
4. The 2 classifiers used are:
    a. SVM
    b. CNN

5. We calculated precision, accuracy and recall and F-measure for SVM and accuracy loss for CNN.
6. Higher values of these metrics would imply that the embedding is well suited for the sentiment analysis task.

Intrinsic Evaluation:

1. Word Similarity Task:
    a. Dataset used: WordSim353
    b. Dataset link: http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/
    c. Dataset description: Combined.tab dataset from above link is used which contains word pairs along with their similarity value on a scale of 1-10 with 10 being the highest.
    d. For every pair of words in the above dataset, we compute cosine similarity of their word vectors.
    e. Then, we find out the spearman rank correlation coefficient between the ranking of similar words according to humans as given by the dataset and that computed using cosine similarity of word vectors.
    f. Higher correlation coefficient would imply that the embedding is well suited for word similarity task.

2. Word Analogy Task:
    a. Dataset used: Google analogy test set by Mikolov et al. (2013b)
    b. Dataset Link: https://aclweb.org/aclwiki/Google_analogy_test_set_(State_of_the_art)#cite_note-1
    c. Dataset description: 8,869 semantic and 10,675 syntactic questions, with 20-70 pairs per category.
    d. Given *a* : *a** :: *b* : *?*, the task is to find *b**.
    e. We try to find the word which word best fits the above description as:
        i. Get word vectors for a, a*, b
        ii. Get a new vector as a – a* + b

iii. Try to find out the word vector closest to above vector using cosine similarity I.e. word vector whose cosine similarity with above vector is maximum
iv. If a – a* + b = the word present in the analogy dataset, we consider this as correct prediction, otherwise, we consider it as incorrect prediction.

f. We calculate accuracy for the above task as number of correct predictions/total no of predictions made
g. Higher accuracy would imply that the embedding is well suited for word analogy task

3. Word Polarity Task:
    a. Dataset used: Opinion Lexicon by Hu and Liu
    b. Dataset link: https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/tree/master/data/opinion-lexicon-English
    c. Dataset Description: A list of positive and negative English words
    d. We compute cosine similarity between the following types of word pairs:
        i. Same polarity: Both words positive or Both words negative
        ii. Opposite polarity: One word positive and the other word negative
    e. We then calculate AUC for the above task as:
        i. AUC = (n1 + 0.5 * n2) / (n * p)
        ii. N1: Number of times the cosine similarity value for word pairs with same polarity is greater than the value for word pairs with opposite polarity
        iii. N2: Number of times the cosine similarity value for word pairs with same polarity is equal to the value for word pairs with opposite polarity
        iv. N: Total numbers of word pairs with opposite polarity
        v. P: Total numbers of word pairs with same polarity
    f. Higher AUC value would imply that the embedding is well suited for word polarity task

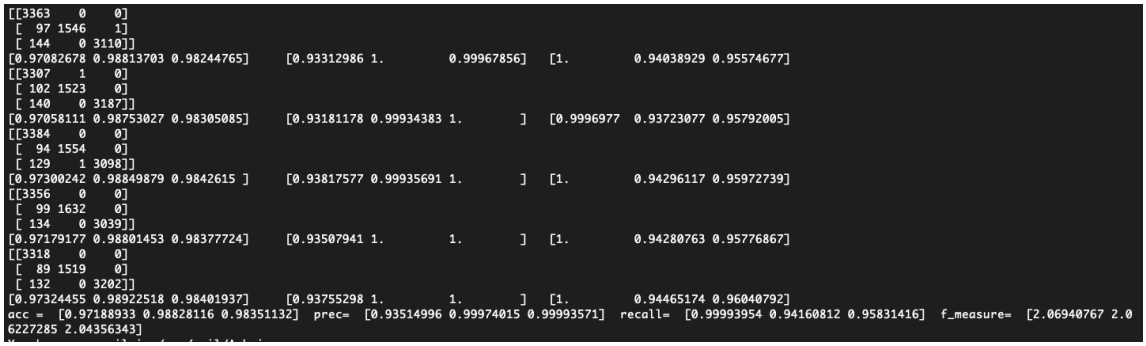
Results and Analysis:

Dataset 1: Twitter dataset:

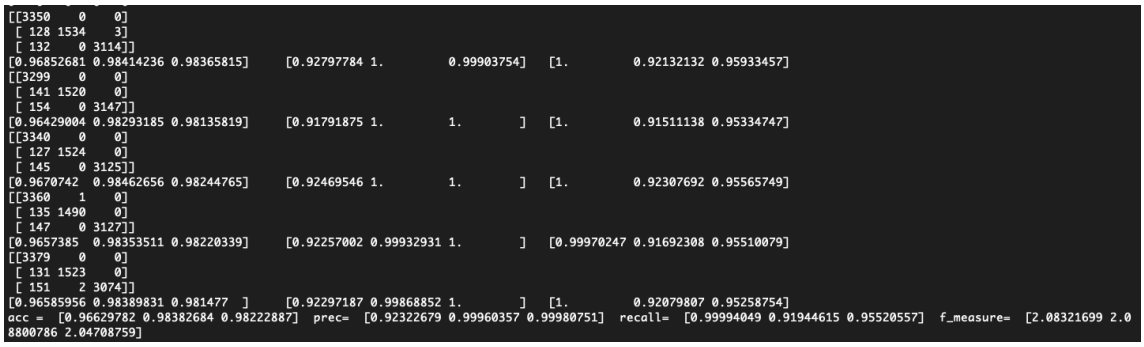
1. **PTE:**
    a. **Extrinsic Evaluation:**
        i. In the PTE extrinsic evaluation, we have used SVM to calculate accuracy, precision by giving full or half or one-third data as a data for 5-fold cross validation. PTE using sentiment label information surely plays a part in getting outstanding results as shown below.

Results:

PTE with full data



PTE with half data



PTE with one third

```
[[3311    0    0]
 [ 104 1544    0]
 [ 143    0 3159]]
[0.97010047 0.98741073 0.98268975]     [0.93057898 1.         1.         ]   [1.         0.9368932  0.95669291]
[[3322    0    0]
 [ 107 1553    0]
 [ 144    0 3135]]
[0.96961627 0.98704757 0.9825687 ]     [0.92975091 1.         1.         ]   [1.         0.93554217 0.95608417]
[[3371    0    0]
 [ 102 1586    0]
 [ 114    0 3088]]
[0.97385304 0.98765283 0.98620022]     [0.93978255 1.         1.         ]   [1.         0.93957346 0.96439725]
[[3400    0    0]
 [ 104 1526    0]
 [ 130    0 3100]]
[0.9716707 0.9874092 0.9842615]     [0.93560815 1.         1.         ]   [1.         0.93619632 0.95975232]
[[3325    0    0]
 [  92 1538    0]
 [ 144    0 3161]]
[0.97142857 0.98886199 0.98256659]     [0.93372648 1.         1.         ]   [1.         0.94355828 0.95642965]
acc =  [0.97133381 0.98767646 0.98365735]  prec=  [0.93388941 1.         1.         ]  recall=  [1.         0.93835269 0.95867126]  f_measure=  [2.0707906  2.0
6569738 2.04311044]
```

The overall results from all the three partitions are quit comparable. As PTE used supervised as well as unsupervised learning that helps it to give better embeddings.

CNN:

While performing CNN we have used supervised information of preprocessed data. The model uses keras library function to perform the operations accuratly. The LSTM model worked well. However, it takes forever to train three epochs. One way to speed up the training time is to improve the network adding “Convolutional” layer. Convolutional Neural Networks (CNN) come from image processing. They pass a “filter” over the data and calculate a higher-level representation. They have been shown to work surprisingly well for text, even though they have none of the sequence processing ability of LSTMs.

```
Train on 24781 samples, validate on 16522 samples
Epoch 1/3
24781/24781 [==============================] - 223s 9ms/step - loss: 0.6320 - acc: 0.6747 - val_loss: 1.8392 - val_acc: 0.0000e+00
Epoch 2/3
24781/24781 [==============================] - 248s 10ms/step - loss: 0.6292 - acc: 0.6746 - val_loss: 1.7794 - val_acc: 0.0000e+00
Epoch 3/3
24781/24781 [==============================] - 236s 10ms/step - loss: 0.6281 - acc: 0.6757 - val_loss: 1.6268 - val_acc: 0.0000e+00
```

The results using CNN migh be low because Convolution layers helps while runnung neural networks on images and not on the texts. But still it worked surprisigly well here.

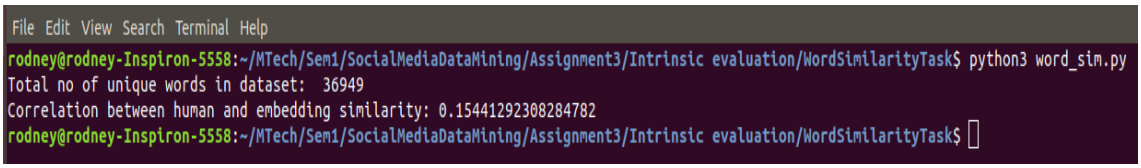
b. **Intrinsic Evaluation:**

i. **Word Similarity Task:**

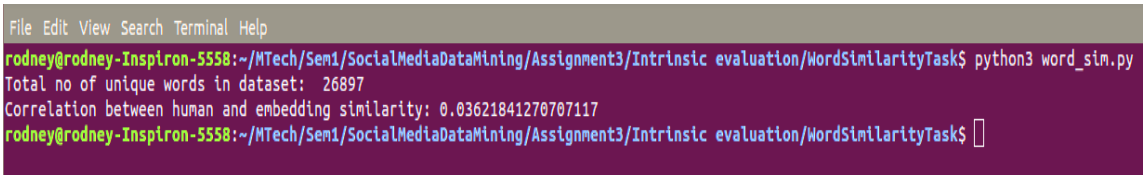
Using entire 42K training set (unprocessed):

```
File  Edit  View  Search  Terminal  Help
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$ python3 word_sim.py
Total no of unique words in dataset:  80507
Correlation between human and embedding similarity: 0.08306619982279925
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$
```

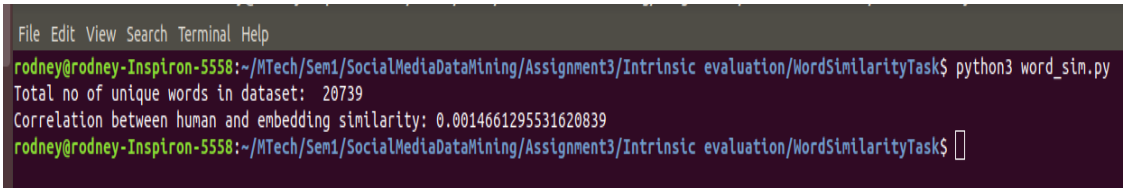
Using entire 42k training set (pre-processed):



Using half of 42k training set:



Using one third of 42k training set:

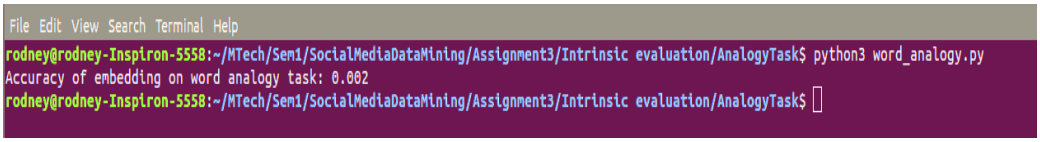


As can be seen from above results, quality of word vectors according to word similarity depends highly on amount of training data and data pre-processing

The more the training data, the better the quality of word vectors.

This shows that pre-processing of text corpus is important for tasks like word similarity. Else, the accuracy will be reduced because of words with just differences in case, stop words, punctuations, tweets containing hashtags, URLs and @ mentions. These words are unnecessary since you would not find URLs or @ mentions or hashtags or punctuations in word similarity dataset

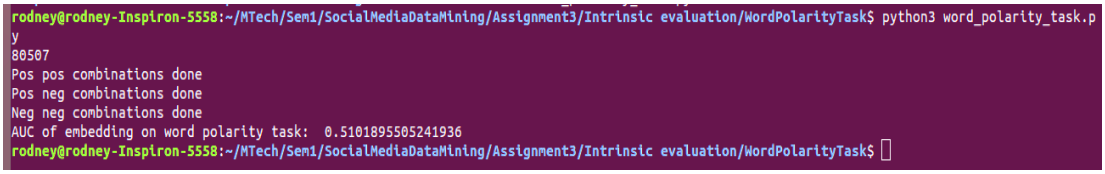
ii. Word Analogy Task:



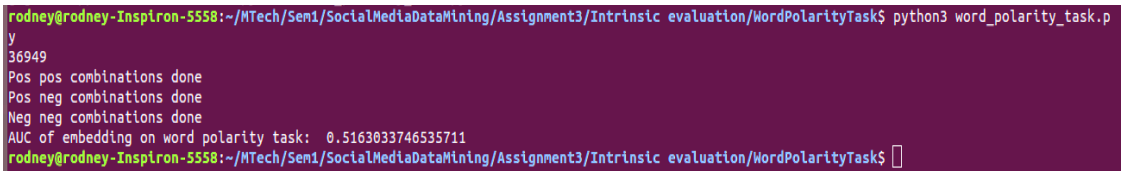
The above numbers do not do justice to the quality of embeddings obtained using PTE. The poor result is most likely because of noise in the twitter dataset with many of the analogies being matched to hindi words or multiple words not separated by space like ”systemgst”, ”itpakisconfuse”, ”everythingthis” etc. Such words are present in the dataset even after pre-processing the dataset. Such words either must have been present in original dataset or must have been generated due to pre-processing since in preprocessing we have replaced punctuations by empty string. So, if a user types ”system,gst” it would be replaced by ”systemgst”.

iii. Word Polarity:

Unprocessed 42K training set:



Pre-processed 42K training set:



It is found that PTE does not do very well in making words vectors of opposite polarity words far from each other while those of same polarity words closer to each other. This is a slightly unexpected result which needs to be analyzed further.

2. **Node2vec:**
   a. **Extrinsic Evaluation:**

i. In Node2vec extrinsic evaluation with SVM, we got less accuracy and precision compared to word embeddings using PTE.

```
[0.63454788 0.80135577 0.69979421]     [0.52691848        nan 0.66721992]
[[3131    0  238]
 [1093    0  529]
 [1615    0 1655]]
[0.64338458 0.80365573 0.71165718]     [0.53622196        nan 0.68331957]
[[3116    0  256]
 [1113    0  526]
 [1611    0 1638]]
[0.63922518 0.80157385 0.71029056]     [0.53356164        nan 0.6768595 ]
[[3122    0  241]
 [1125    0  532]
 [1638    0 1602]]
[0.63631961 0.79939467 0.70811138]     [0.53050127        nan 0.67452632]
[[3064    0  227]
 [1176    0  521]
 [1752    0 1520]]
[0.61803874 0.79455206 0.69733656]     [0.51134846        nan 0.670194  ]
acc =  [0.6343032  0.80010642 0.70543798]  prec=  [0.52771036        nan 0.67442386]
```

b. **Intrinsic Evaluation:**

i. **Word Similarity:**

```
File  Edit  View  Search  Terminal  Help
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$ python3 word_sim.py
Total no of unique words in dataset:  49655
Correlation between human and embedding similarity: 0.17227723044744145
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$
```

ii. **Word Analogy:**

```
File  Edit  View  Search  Terminal  Help
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/AnalogyTask$ python3 word_analogy.py
Accuracy of embedding on word analogy task: 0.001
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/AnalogyTask$
```

Poor result most likely due to noise in dataset.

iii. **Word Polarity:**

```
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordPolarityTask$ python3 word_polarity_task.p
y
49655
Pos pos combinations done
Pos neg combinations done
Neg neg combinations done
AUC of embedding on word polarity task: 0.4924401552587605
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordPolarityTask$
```

From the above result and observation from the output files written by above program (which writes similarity scores of every possible same polarity and opposite polarity word pair

from dataset), it is obvious that node2vec does not do well in word polarity task.

3. **Comparison of above embedding methods (Extrinsic evaluation):**

We got poorer results with node2vec than with PTE. This is expected since PTE uses more of the available information than node2vec I.e. PTE uses word-word co-occurrence graph, word-document graph and word-label graph while ndoe2vec just uses edge list which can be derived from word-word graph. The usage of sentiment labels in learning vectors seems to greatly improve performance in case of PTE.

4. **Comparison of above embedding methods (Intrinsic evaluation)**

   i. **Word Similarity:**

   We find that node2vec slightly outperforms PTE. This must be because of the larger training set used for node2vec. Node2vec outperforming PTE is not an expected result since PTE uses word co-occurrence graph for training while we have just passed edge list to node2vec without the weights of those edges which is used in PTE. Word co-occurrence value should help PTE to learn similar vectors for words in neighborhood and hence increase the accuracy on word similarity task. We did not repeat the experiment with same no of words on node2vec as PTE since running node2vec takes a long time even with a 12GB computer.

   ii. **Word Analogy:**

   We obtain similar results with both embeddings. The most likely reasons are poor quality of original dataset and lack of advanced pre-processing techniques.

   iii. **Word Polarity:**

   PTE outperforms node2vec on word polarity task. This is because PTE uses label information in training word vectors whereas node2vec does not. So, a similar word vector will be learned for words in negative tweets, but dissimilar vectors will be learned for words in positive tweets. This may help in getting similar vectors for words of same polarity and dissimilar vectors for words of opposite polarity.

Dataset 2: IMDB dataset:

5. **PTE:**
    c. **Extrinsic Evaluation:**
        i. **IMBD SVM**

On smaller IMDB training set(25K)

```
[[1206 1341]
 [1149 1304]]
[0.502 0.502]     [0.51210191 0.49300567]   [0.47349823 0.53159397]
[[1262 1203]
 [1306 1229]]
[0.4982 0.4982]     [0.49143302 0.50534539]   [0.51196755 0.48481262]
[[1202 1315]
 [1200 1283]]
[0.497 0.497]     [0.50041632 0.49384142]   [0.47755264 0.51671365]
[[1279 1198]
 [1295 1228]]
[0.5014 0.5014]     [0.496892   0.50618302]   [0.51635042 0.48672216]
[[1263 1231]
 [1270 1236]]
[0.4998 0.4998]     [0.49861824 0.50101338]   [0.5064154  0.49321628]
acc =  [0.49968 0.49968]  prec=  [0.4998923  0.49987778]  recall=  [0.49715685 0.50261174]  f_measure=  [4.01186855 3.99009636]
```

On bigger IMDB training set(42K)

```
[[2020 1320]
 [1238 3822]]
[0.69547619 0.69547619]     [0.62001228 0.74329055]   [0.60479042 0.75533597]
[[2098 1313]
 [1209 3780]]
[0.6997619 0.6997619]     [0.63441185 0.74219517]   [0.61506889 0.75766687]
[[2092 1375]
 [1144 3789]]
[0.70011905 0.70011905]     [0.64647713 0.73373354]   [0.60340352 0.76809244]
[[2050 1309]
 [1221 3820]]
[0.69880952 0.69880952]     [0.62671966 0.74478456]   [0.61030068 0.75778615]
[[2067 1356]
 [1183 3794]]
[0.6977381 0.6977381]     [0.636      0.73669903]   [0.60385627 0.76230661]
acc =  [0.69838095 0.69838095]  prec=  [0.63272418 0.74014057]  recall=  [0.60748396 0.76023761]  f_measure=  [3.22660149 2.66647293]
```

        ii. **IMDB CNN**

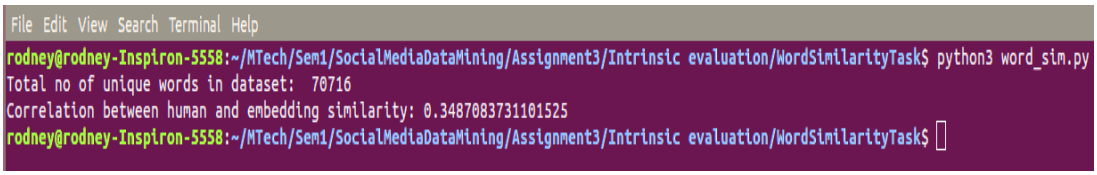
```
Train on 15000 samples, validate on 10000 samples
Epoch 1/3
15000/15000 [==============================] - 137s 9ms/step - loss: 0.6927 - acc: 0.5045 - val_loss: 0.6925 - val_acc: 0.4966
Epoch 2/3
15000/15000 [==============================] - 138s 9ms/step - loss: 0.6920 - acc: 0.4970 - val_loss: 0.6918 - val_acc: 0.4966
Epoch 3/3
15000/15000 [==============================] - 147s 10ms/step - loss: 0.6917 - acc: 0.4988 - val_loss: 0.6916 - val_acc: 0.4964
```

    d. **Intrinsic Evaluation:**
        i. **Word Similarity:**

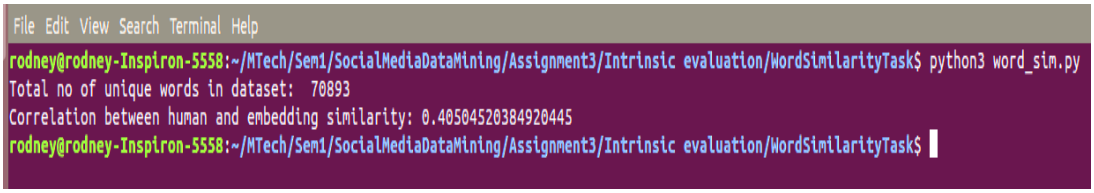
Using unprocessed dataset(25K):

```
File  Edit  View  Search  Terminal  Help
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$ python3 word_sim.py
Total no of unique words in dataset:  71153
Correlation between human and embedding similarity: 0.29553082040012246
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$
```

Using pre-processed dataset(25K):

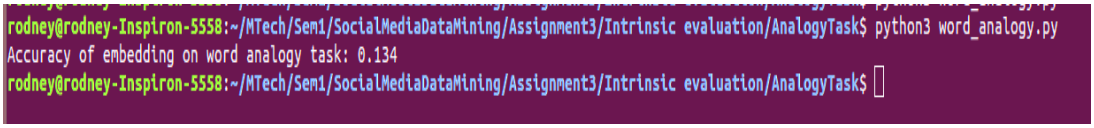


Using a bigger training set(42K):



Even though IMDB dataset requires less pre-processing than twitter dataset, pre-processing still improved accuracy on similarity task by 0.05 which is a significant improvement. This shows that pre-processing of text corpus is vital for tasks like word similarity. Else, the accuracy will be reduced because of words with just differences in case, stop words and punctuations. Also, bigger training set led to even further performance improvement which is expected.

ii. **Word Analogy:**



The above result is on only 1000-word pairs of Google analogy dataset. Running this task takes a lot of time. (Running for 3000-word pairs took about 10 hours). The comparatively poorer performance on analogy dataset than similarity dataset is due to analogy task being harder than similarity task. In analogy task, there are 4 words involved while there are just 2 words in similarity task. Also, for every analogy case, the closest word in the entire dataset corresponding to the analogy needs to be found out which is much more complex than just finding cosine similarity of word pairs in similarity dataset and finding their correlation. There exist many words that may match an analogy. For example in IMDB dataset, father-mother :: nephew-uncle instead of father-mother :: nephew-niece since all words are

related to members of the family. Also, grandpa-grandma :: he-himself instead of grandpa-grandma :: he-she.

iii. **Word Polarity:**

Using 25K training set:

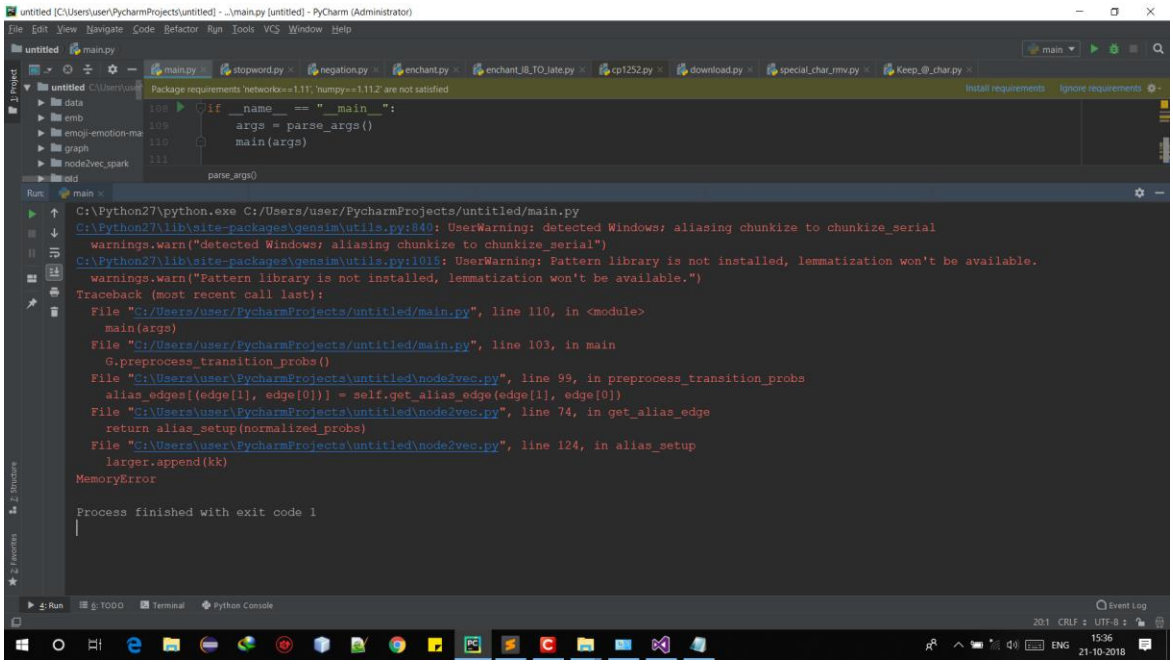
```
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordPolarityTask$ python3 word_polarity_task.p
y
71153
Pos pos combinations done
Pos neg combinations done
Neg neg combinations done
AUC of embedding on word polarity task: 0.5249232471354786
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordPolarityTask$
```

Using 42K training set:

```
File  Edit  View  Search  Terminal  Help
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordPolarityTask$ python3 word_polarity_task.p
y
AUC of embedding on word polarity task:  0.5361019699778277
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordPolarityTask$
```

6. **Node2vec:**

Was not possible due to out of memory error while running node2vec on IMDB dataset. Node2vec was run on a 12 GB computer. Below is the screenshot of out of memory error:



7. **Comparison of above embedding methods (Extrinsic evaluation):**

Not possible due to node2vec giving out of memory error

8. **Comparison of above embedding methods (Intrinsic evaluation):**

Not possible due to node2vec giving out of memory error

Comparison of PTE across datasets:

1. **Extrinsic Evaluation:**
    a. The twitter dataset scores are high compared to imdb dataset because in case of twitter dataset that we had provide has much more data for training hence that can outperform over IMDB data (Which has much lesser rows compared to twitter data).

2. **Intrinsic Evaluation:**
    i. **Word Similarity:**

    We find that performance of PTE embedding improved greatly on IMDB dataset.

    This is expected since IMDB dataset consists of cleaner reviews than twitter dataset hence learning better word vectors on IMDB dataset. This shows that quality of text corpus I.e. less noise in text corpus is vital for learning high quality word vectors.

    ii. **Word Analogy:**

    Twitter dataset fares miserably on this task giving accuracy of about 0.01 for 1000-word pairs of Google analogy set while IMDB dataset gave accuracy of 0.134 which is a significant improvement. The bad accuracy on twitter dataset is due to noise in twitter dataset and may also be due to not performing any advanced text pre-processing.

    iii. **Word Polarity:**

    Here too, IMDB dataset does better than twitter dataset for the same reason as above

Comparison of Node2vec across datasets:

Was not possible due to out of memory error while running node2vec on IMDB dataset. Node2vec was run on a 12 GB computer

Doc2Vec:

Extrinsic Evaluation:

```
[[2020 1320]
 [1238 3822]]
[0.69547619 0.69547619]     [0.62001228 0.74329055]   [0.60479042 0.75533597]
[[2098 1313]
 [1209 3780]]
[0.6997619 0.6997619]     [0.63441185 0.74219517]   [0.61506889 0.75766687]
[[2092 1375]
 [1144 3789]]
[0.70011905 0.70011905]     [0.64647713 0.73373354]   [0.60340352 0.76809244]
[[2050 1309]
 [1221 3820]]
[0.69880952 0.69880952]     [0.62671966 0.74478456]   [0.61030068 0.75778615]
[[2067 1356]
 [1183 3794]]
[0.6977381 0.6977381]     [0.636      0.73669903]   [0.60385627 0.76230661]
acc =  [0.69838095 0.69838095]  prec=  [0.63272418 0.74014057]  recall=  [0.60748396 0.76023761]  f_measure=  [3.22660149 2.66647293]
```

Intrinsic Evaluation:

i. Word Similarity:

```
File  Edit  View  Search  Terminal  Help
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$ python3 word_sim.py
Total no of unique words in dataset:  52767
Correlation between human and embedding similarity: 0.3614082909968745
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/WordSimilarityTask$
```

ii. Word Analogy:

```
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/AnalogyTask$ python3 word_analogy.py
Accuracy of embedding on word analogy task: 0.164
rodney@rodney-Inspiron-5558:~/MTech/Sem1/SocialMediaDataMining/Assignment3/Intrinsic evaluation/AnalogyTask$
```