ML HW4 Report B03901056 孫凡耕

Preliminaries:

1. Data Preprocess:

```
data = open(data_filename, 'r').read().splitlines()        #read data
data = [ re.sub("\.+\s", " ", datum) for datum in data ]   #replace any '.' that is followed by white-space(s)
data = [ re.sub("\.+$", " ", datum) for datum in data ]    #replace any '.' at the end of sentence
data = [ re.sub("[^\w]", " ", datum) for datum in data ]   #replace anything besides alphanumeric and '_'
stemmer = SnowballStemmer("english")                       #using SnowballStemmer from nltk
data = [[stemmer.stem(word) for word in datum.lower().split()]  for datum in data] #stem, to lowercase, split
data = [ [ word for word in datum if word not in STOPWORDS ] for datum in data ]   #remove stopwords
```

2. sklearn.feature_extraction.TfidfVectorizer:

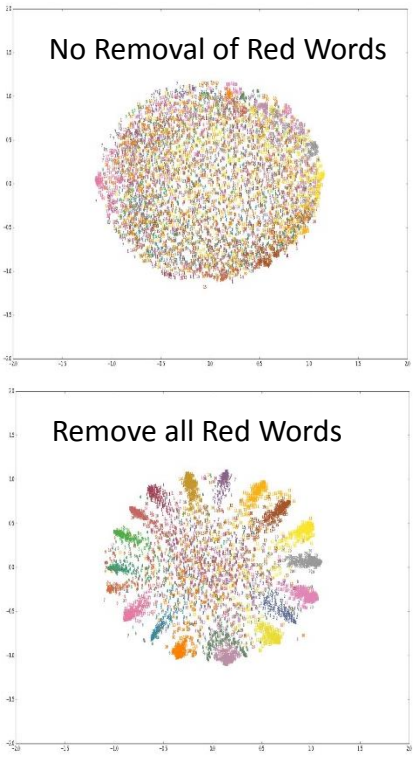
```
min_df=8 stopwords=STOPWORDS sublinear_tf=True binary=True analyzer=lambda x: x
```

3. sklearn.decomposition.TruncatedSVD `n_components=19`
4. sklearn.cluster.KMeans `n_init=100 n_jobs=-1`
5. gensim.models.Word2Vec `size=200 min_count=2 workers=8 negative=20 window=12`

Q #1.

Not removing any stopwords while preprocessing data!

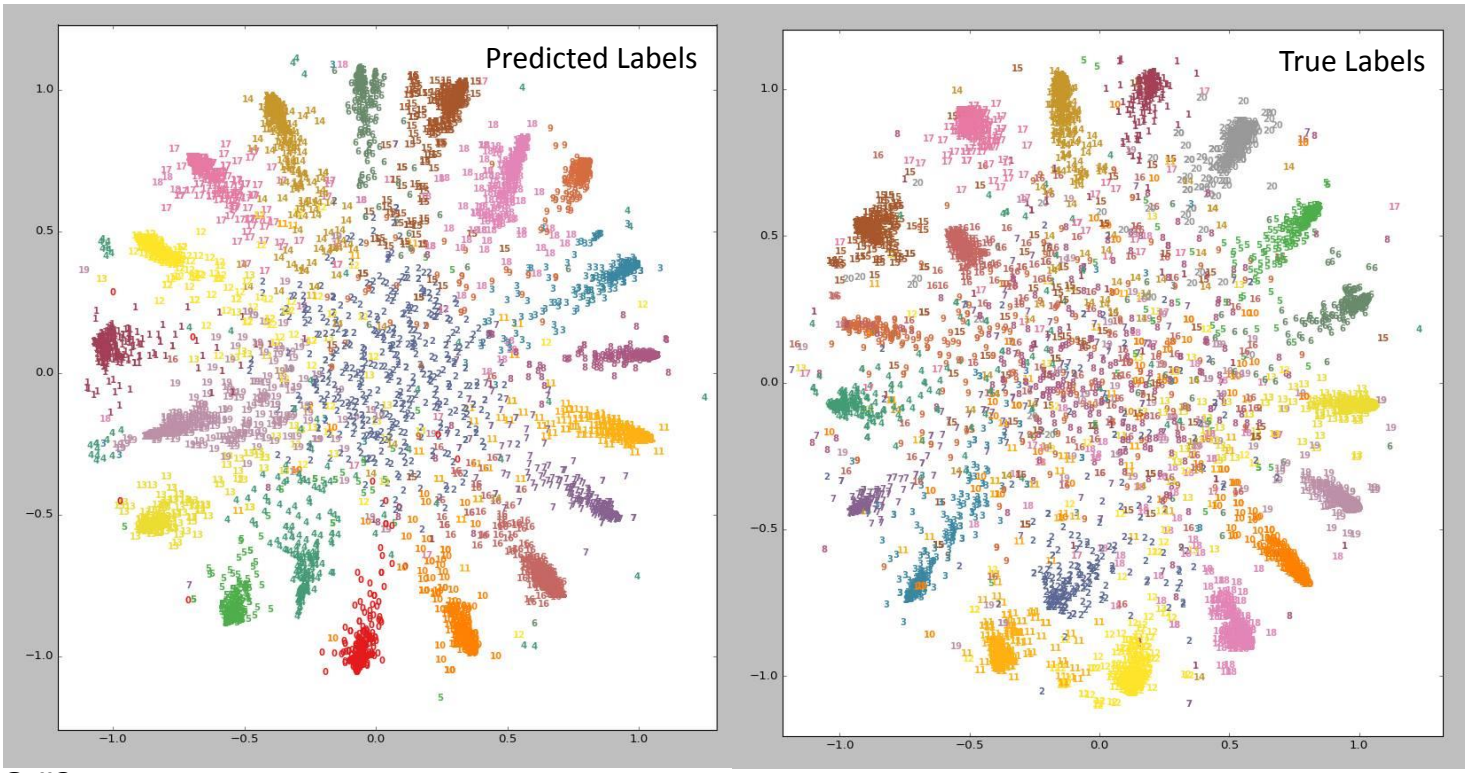
- Steps：
  Tfidf -> TruncatedSVD -> Normalizer -> KMeans (20 clusters) -> most common ten words in a cluster -> sort according to idf score.
- Results：
  Cluster: #0 to in a how sharepoint qt list custom web site
  Cluster: #1 to in a and from file bash apach svn script
  Cluster: #2 to in a how the of use file can get
  Cluster: #3 to in a with use spring ajax apach qt problem
  Cluster: #4 to in a how of use is on from creat
  Cluster: #5 to in a how of scala haskel function list type
  Cluster: #6 to in use and from not ajax call php jqueri
  Cluster: #7 to in a how for drupal view form 6 node
  Cluster: #8 to in a how the on mac x os window
  Cluster: #9 to in a how the on magento not custom product
  Cluster: #10 to in a use for is what oracl best develop
  Cluster: #11 to in a how the use from file an get
  Cluster: #12 to in a the is what s way it there
  Cluster: #13 to in a how on wordpress page post categori plugin
  Cluster: #14 to in a how from file an excel oracl vba
  Cluster: #15 to in a how the for visual studio project 2008s
  Cluster: #16 to in a how with use and hibern spring map
  Cluster: #17 to in a how with use and linq queri sql
  Cluster: #18 to in a how use from matlab qt function array
  Cluster: #19 to in a how the use i do an can



- Discussion：
  It's obvious that there are a lot of redundant, useless words in every cluster, which are highlighted in red color. The graphs show the difference before and after **removing these red words for the true label.** The improvement is significant.

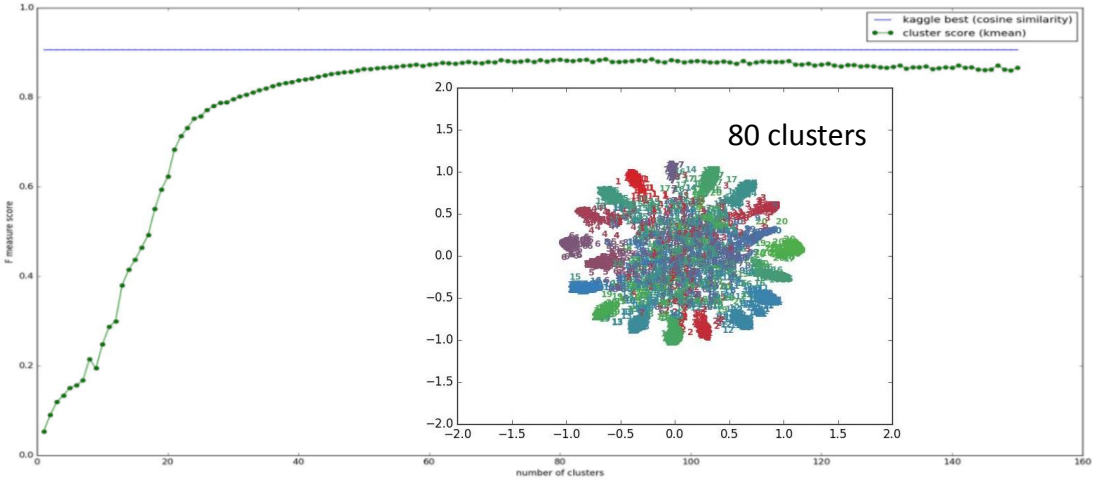
Q #2.

- Step：
  Tfidf -> TruncatedSVD -> Normalizer -> KMeans(20 clusters), MDS -> plot 5000 randomly-chosen data
- Results：
  On the next page.
- Discussion：
  We can see that the main difference between my prediction and the true label is the data that are close to the origin. In my clustered results, most of these points belongs to the same cluster, which is the 'blue 2' cluster in the left graph. However in reality, these points are actually a mixture of many tags. After some observation, I found out the well clustered data are documents that apparently contains one of the 20 key-tags. The points that are close to the origin corresponds to documents that contains none of the key-tags, thus they are in fact very hard to be separated. **There are still some points that are distant to the origin but not in one of the well-clustered groups, these documents actually contains more than one of the key-tags, thus located between well-clustered groups.**



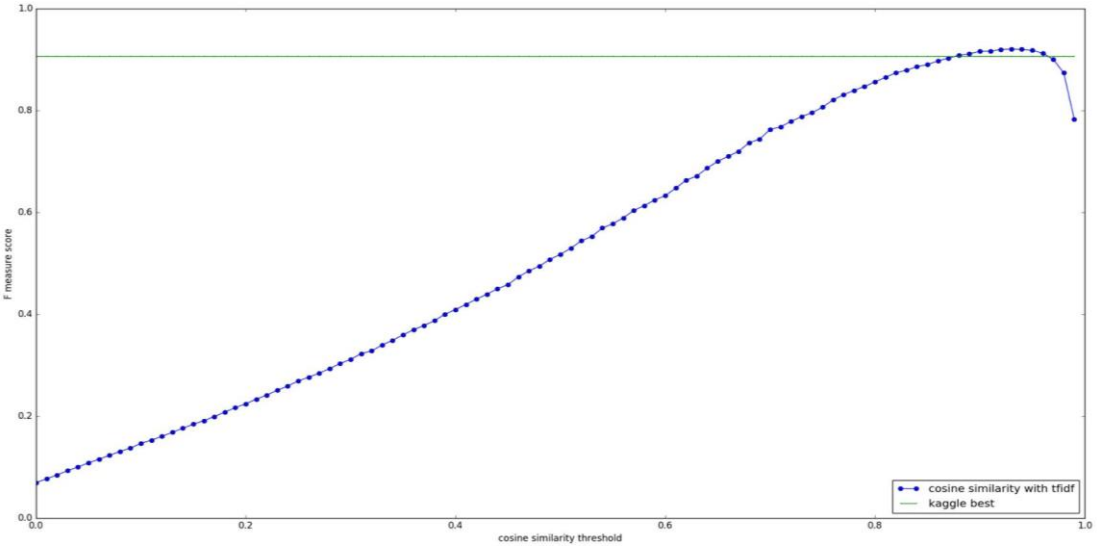
Q #3.

- Steps：
  - A. Tfidf -> TruncatedSVD -> Normalizer -> KMeans (different n_clusters) -> evaluate
  - B. Tfidf -> TruncatedSVD -> Normalizer -> Cosine Similarity (different threshold) -> evaluate
  - C. Word2Vec (trained on all data, including docs.txt) -> Normalizer -> mean vector -> Cosine Similarity (different threshold) -> evaluate
  - D. Word2Vec (trained on all data, including docs.txt), Tfidf -> Normalizer -> Word2Vec multiply by the idf weight -> Cosine Similarity (different threshold) -> evaluate
- Results：(the straight line represents my kaggle-best score = 0.9065)



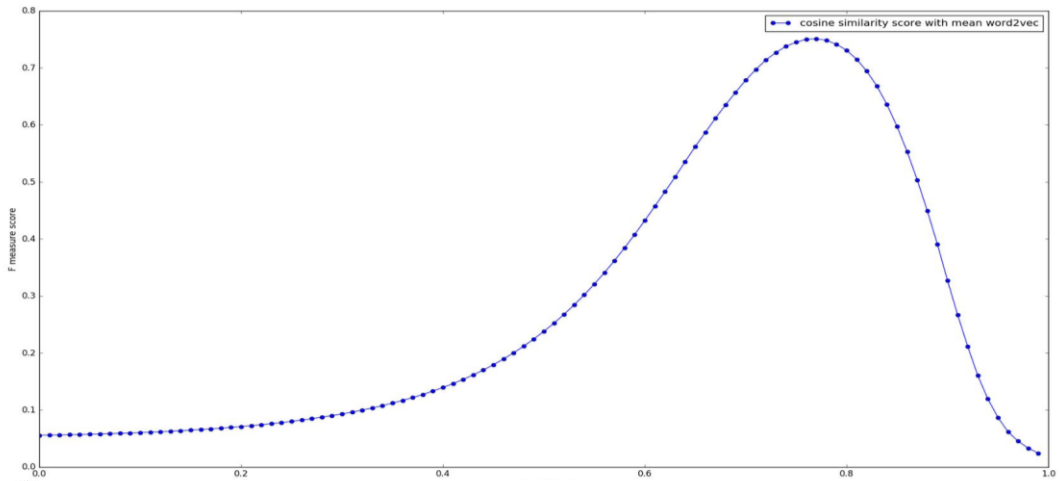
Graph A:

1. Different number of clusters with Tfidf
2. Best at number of clusters ≅ 75~85
3. Best score ≅ 0.88



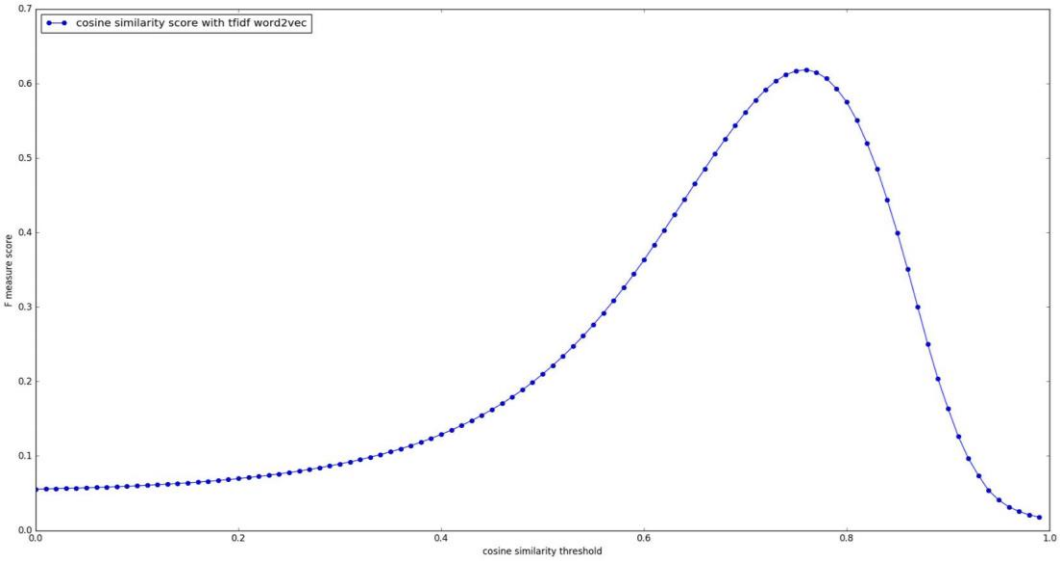
Graph B:

1. Different cosine similarity threshold with Tfidf
2. Best at threshold ≅ 0.91
3. Best score ≅ 0.922



Graph C:

1. Different cosine similarity threshold with mean word2vec vector
2. Best at threshold ≅ 0.75
3. Best score ≅ 0.74



Graph D:

1. Different cosine similarity threshold with Tfidf * word2vec vector
2. Best at threshold ≅ 0.76
3. Best score ≅ 0.61

- Discussion：

The reason why number of clusters more than 20 is better than exactly 20 clusters will be discussed in the next question. So let's first discuss why cosine similarity performs better than clustering. I believe the underlying answer to this problem is what I've presented in the second question. From the 2-D graph, there are a lot of chaotic points that are close to the origin. If clustering is performed, these points will eventually be grouped into a cluster, which results in a large amount of inaccuracy. However, if we utilize cosine similarity to decide whether two points are in the same cluster or not, most of these points will be ruled out as different groups, which are better most of the times.

Next, let's concentrate on graph Cand D. Using Word2Vec to represent a word is magnificent. But the problem is how to use word vectors to represent a sentence. The both naïve way is to take the sum or the mean of all the word vectors within a sentence. Another way is to combine with the idf score from training a Tfidf model. In this case, the naïve way of averaging vectors perform better, but I believe that there is still some rooms of the second ways because perhaps the best model is not simply multiplication, but with different degrees. For example, (word2vec ** 1.5) * (idf ** 0.5) may performs better, but I didn't tried out all the possibilities. Some degrees of information are loss in both ways while compiling the vector representation of sentence, thus are both worse than Tfidf.

Q #4.

- Steps：

Tfidf -> TruncatedSVD -> Normalizer -> KMeans -> evaluate F measure score (average for three times)

- Results：

The graph A. in Q #3.

- Discussion：

The reason why more clusters results in a higher score originates from the mathematical formula of F-measure with beta equivalent to 0.25.

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R} = \frac{(\beta^2+1)(TP)}{(\beta^2+1)(TP)+\beta^2(FN)+(FP)}, \quad where\ \beta = 0.25$$

Since we want to maximize this score, we want the denominator to be as small as possible. Every wrong prediction must fall into one of FN(false negative) or FP(false positive). However, the weight of FN is multiplied by 1/16, this indicates that the effect of FP is 16 times higher than FN. So 16 FN has the same negative impact as a single FP, thus we rather be more confident about our prediction before actually predicting a positive answer. That's why more clusters results in a better score.