CS510: Advanced Topics in Information Retrival **Fall 2017**

Assignment 5

Zhuo Li(zhuol2) *Handed In: Octorber 19, 2017*

1. Brown Clusters

   a. The three applications are:
      - Named entity recognition
      - Dependency parsing
      - Constituency parsing

   b. Basically, whenever we merge words into the same class, we are losing mutual information. To minimize the loss, we need to merge words that are highly dependent on / related to each other. Because with the same mutual information before merging, we can have a larger mutual information with these kind of words after merging, thus minimizing the loss.

   c. We can first take every word as a class, then greedily take the merge between classes with lowest mutual information loss. And for each iteration, we can maintain the current maximum average mutual information cut. Then as we compute from bottom to top, we will have the optimal level that is the best to stop merging clusters.

   d. We can actually take the word classes we learned in the unsupervised way to the training setting of labeled data for supervised learning, then our algorithm on the small labeled dataset can learn and transfer knowledge between those word types/classes.

   e. As the cut-off depth increases, the number of word clusters increase as well, so we will get more and more specific clusters. When the cut-off depth is small, we have more general clusters.
      Assume we are processing plenty of tweets in Twitter. If we want to label in a high level, like sentiment classification, it may be useful to have general clusters just for positive/neutral/negative. If we want to have more specific classification like group the tweets to different topics, more specific clusters can be helpful.

   f. For bigram class model, we need to maximize

$$p(W|\pi) = \prod_{i=1}^{N} p(c_i|c_{i-1})p(w_i|c_i) \tag{1}$$

   where $c_i = \pi(w_i)$.

The log likelihood of (1) is

$$\begin{aligned}
L(\pi) &= \sum_{i=1}^{N} \log p(c_i|c_{i-1})p(w_i|c_i) \\
&= \sum_{w_i,w_j} N_{w_i,w_j} \log p(c_j|c_i)p(w_j|c_j) \\
&= \sum_{w_i,w_j} N_{w_i,w_j} \log \frac{N_{c_i,c_j}}{N_{c_i}} \cdot \frac{N_{w_j}}{N_{c_j}} \\
&= \sum_{w_i,w_j} N_{w_i,w_j} \log \frac{N_{c_i,c_j}}{N_{c_i} \cdot N_{c_j}} + \sum_{w_i,w_j} N_{w_i,w_j} \log N_{w_j} \\
&= \sum_{c_i,c_j} N_{c_i,c_j} \log \frac{N_{c_i,c_j}}{N_{c_i} \cdot N_{c_j}} + \sum_{w_i} N_{w_i} \log N_{w_i}
\end{aligned} \tag{2}$$

While mutual information for each class pair is

$$\begin{aligned}
MI(c,c') &= \sum_{c_i,c_j} p(c_i,c_j) \log \frac{p(c_i,c_j)}{p(c_i)p(c_j)} \\
&= \sum_{c_i,c_j} \frac{N_{c_i,c_j}}{N} \log \frac{N_{c_i,c_j} \cdot N \cdot N}{N \cdot N_{c_i} N_{c_j}} \\
&= \frac{1}{N} \sum_{c_i,c_j} N_{c_i,c_j} \log \frac{N_{c_i,c_j}}{N_{c_i} \cdot N_{c_j}} + \frac{1}{N} \sum_{c_i,c_j} N_{c_i,c_j} \log N \\
&= \frac{1}{N} \sum_{c_i,c_j} N_{c_i,c_j} \log \frac{N_{c_i,c_j}}{N_{c_i} \cdot N_{c_j}} + \log N
\end{aligned} \tag{3}$$

For (2), $\sum_{w_i} N_{w_i} \log N_{w_i}$ is fixed with respect to a specific class mapping. For (3), $\frac{1}{N}$ and $\log N$ are constant. So we can say that maximizing the log likelihood of the bigram class model (i.e. equation (2)), is the same as maximizing mutual information of adjacent class pairs in Brown clustering (i.e. equation (3)).

2. Word Embeddings (Theory)

   a. As J.R.Firth said, *You shall know a word by the company it keeps*, the distributional hypothesis is that words that occur in the same contexts tend to have similar meanings.

      - For Skip-Gram model, we can use a single word to determine the words surrounding it (context), which illustrates that the word is related tightly with its context as in distributional hypothesis.
      - For CBOW model, we are going to fill the missing word given its context. This is exactly what Firth said.

   b. The actual data is all words that appear in the context of the target word. The noise distribution is the distribution of all other words in the corpus that do not appear in the context of the target word. We can just randomly choose words in the corpus with its distribution, if it doesn't appear in the context of the target word, we can use it.

c. i. For more frequent words, we have larger $f(w_i)$, and then larger $P(w_i)$, which means these words are more likely to be discarded. For infrequent words, they will have a lower probability to be discarded. So this actually reduce the impact on more frequent words for our model in ranking. This makes sense because we don't just want good embeddings for frequent words, but for infrequent words as well.

ii. Some really high frequent words are removed by subsampling, so the target word may be able to reach further words than before. So the size of the context window is increased.

3. Word Embeddings (Practice)
We can write the following codes to make our life easier

```
import sys
import math

WORD_DIC = {
    'the': [ 0.131, 0.001, 0.023, 0.918, 0.991, 0.912, 0.787,
       0.675, 0.787, 0.987 ],
    'cat': [ 0.911, 0.891, 0.912, 0.016, 0.099, 0.189, 0.777,
       0.776, 0.853, 0.992 ],
    'for': [ 0.112, 0.009, 0.032, 0.819, 0.971, 0.932, 0.788,
       0.677, 0.777, 0.988 ],
    'data': [ 0.954, 0.919, 0.881, 0.812, 0.901, 0.990, 0.012,
       0.002, 0.014, 0.909 ],
    'mouse': [ 0.912, 0.881, 0.922, 0.019, 0.100, 0.199, 0.011,
       0.003, 0.016, 0.898 ],
    'it': [ 0.142, 0.010, 0.026, 0.820, 0.917, 0.923, 0.781, 0.611,
        0.722, 0.977 ],
    'dog': [ 0.922, 0.882, 0.931, 0.011, 0.101, 0.193, 0.769,
       0.762, 0.841, 0.989 ],
    'also': [ 0.121, 0.004, 0.021, 0.919, 0.981, 0.917, 0.790,
       0.617, 0.712, 0.969 ],
    'computer': [ 0.912, 0.923, 0.899, 0.853, 0.910, 0.991, 0.022,
       0.010, 0.016, 0.912 ]
}

def dot_prod(vector1, vector2):
    temp_sum = 0
    for i in range(len(vector1)):
        temp_sum += vector1[i] * vector2[i]
    return temp_sum

def cos_similarity(word, magnitudes, query_word):
    dot_product = dot_prod(WORD_DIC[word], WORD_DIC[query_word])
    return dot_product / (magnitudes[word] * magnitudes[query_word])

if __name__ == '__main__':
    magnitudes = {}
    for word, vector in WORD_DIC.items():
        temp_sum = 0
        for v in vector:
            temp_sum += v * v
        magnitudes[word] = math.sqrt(temp_sum)

    query_word = sys.argv[1]

    similarities = []
    for word, vector in WORD_DIC.items():
        if word == query_word:
            continue
        similarities.append((word, cos_similarity(word, magnitudes, query_word)))

    similarities = sorted(similarities, key=lambda x: x[1], reverse=True)

    for similarity in similarities:
        print(similarity)
```

a. By running the script with `'cat'`, we can have

```
('dog', 0.9999071932103957)
('mouse', 0.8069059656464488)
('data', 0.6604214298514217)
('computer', 0.6577737671141277)
('for', 0.6064200638550592)
('it', 0.606230088862793)
('the', 0.5969642596224372)
('also', 0.5849669075267356)
```

The three nearest neighbors are `'dog'`, `'mouse'` and `'data'`

b. By running the script with `'computer'`, we can have

```
('data', 0.9996640177196193)
('mouse', 0.816147502773864)
('also', 0.6642668333372335)
('it', 0.663583374559236)
('dog', 0.6619753729437896)
('cat', 0.6577737671141277)
('the', 0.655148259644331)
('for', 0.6517125735184346)
```

The three nearest neighbors are `'data'`, `'mouse'` and `'also'`

c. By running the script with `'the'`, we can have

```
('also', 0.9993256668326264)
('for', 0.9990916713931316)
('it', 0.9988488550590425)
('computer', 0.655148259644331)
('data', 0.6473562342495666)
('cat', 0.5969642596224372)
('dog', 0.5933226317859459)
('mouse', 0.32060085114877945)
```

The three nearest neighbors are `'also'`, `'for'` and `'it'`

d. I will cluster the words as followings

Semantics	Words
Animals	cat, mouse, dog
Information Tech	data, mouse, computer
Non-object	the, for, it, also

If we use KNN with $k = 3$, we can have

Words	1st Neighbor	2nd Neighbor	3rd Neighbor
the	also	for	it
cat	dog	mouse	data
for	it	the	also
data	computer	mouse	dog
mouse	data	computer	dog
it	for	also	the
dog	cat	mouse	data
also	the	it	for
computer	data	mouse	also

According to k-means, the groups are: (the, for, it, also), (computer, data), (cat, dog, mouse).
So we can find that this clustering correspond to the clustering that can be obtained from the vector space from kNN.

e. Hard clustering methods can only cluster a specific word into a single cluster, so if a word has more than one meaning in different context (e.g. mouse in this example), it may not be able to reflect the truth behind the scene. Fuzzy clustering can give each cluster a probability for a word, which address this problem like word embedding does.