Natural Language System Lab 1 Report

10406141 Wenchang Liu

1. Task 1: POS Tagging

(a) Using NLTK pos tagging tools to tag all the tokens

First step is to use NLTK pos tagging tools to generate ground truth pos tags for later calculations.

(b) Count necessary statistics and Calculate likelihood

We want to know the best sequence of tags(including race/NN or race/VB) that corresponds to the observation sequence of words.

In order to decide which is most probable, we need to use Bayes Rule and its independence assumption to calculate the likelihood for race/NN and race/VB and then compare them. For comparison, we can remove the common components of the likelihood formula and the final formula would be: Likelihood(race is NN) = P(race|NN) * P(NN|DT) * P(IN|NN), Likelihood(race is VB) = P(race|VB) * P(VB|DT) * P(IN|VB). And in order to get those probabilities we need to count how many times “race” is NN, NN after DT, IN after NN, number of NN tags, number of DT tags and the same for VB as well. (For word likelihood probabilities e.g. P(race|NN) = count(race tagged as NN) / count(NN tags), for tag transition probabilities e.g. P(NN|DT) = count(NN occurs after DT) / count(DT tags))

After we get all the statistics, we can calculate the probabilities, and then multiply them to get the likelihood for NN and VB.

(c) Compare likelihood and Draw conclusion

The result shows that the likelihood for (race is NN) is larger so that we can draw the conclusion that race/NN is more likely.

2. Task 2: Distributional Semantics

(a) Building the Target Words Vocabulary Words Matrix

First we need to create an array that contains all the target words, and also read all the text under the corpus folder, I removed all symbols and numbers and using a stopword list from GitHub, and then split each document by space for tokenization, then I create an empty 2d array and initiate all elements to 0, for each target word, search through all the corpus text, if the target word exists in the document or context window, I go through every word (except for that target word, I only increase 1 instead of number of occurrences in that case) of the document or context window and find the element (matrix[target_word_index][vocab_word_index]) and increase its value by the times that the target word occurs in that context. Finally, the initial 2d array is what we need.

The cluster algorithm I chose is from scikit-learn Kmeans, it will allow you to define the number of clusters and return a list of labels (showing which cluster the target word is in) for each input target word. Here we cluster 50 target words into 50 clusters, each word will be in a different cluster for sure.

(b) Evaluate Results

The sklearn k-means algorithm will return a list containing 100 elements corresponding to 50 target words and its reversed words, and the value of each element is the cluster label for each target word, so ideally, there will be 50 clusters and each cluster contains only 1 pair of a word and its reverse. Therefore we use a loop i from 0 to 49, for each loop, we take elements in the ith cluster, because I put the reversed target word next to the original one, so their indexes should be consecutive numbers, so I check if it contains 2 consecutive values, if so, we regard this cluster as correct, we calculate accuracy by using (number of correct clusters / 50). The average accuracy for cluster trained on corpusB is around 40%, we can see that many pairs are clustered into the same cluster which means the algorithm cannot distinguish those words, I think it may because each document contains similar words, so some of the words usually co-occur together or co-occur together with similar vocabulary words, so that their vectors will be similar, which will make the cluster algorithm have difficulty to distinguish them.

(c) Analyse Variations

I experimented with several variations including:

(i) Different context window sizes compared with (b)

The context window means that we only define a target word's context by other words in its small context instead of the whole document. Using smaller context window instead of whole documents can be better because other target words are less likely to occur in one target word's small context while it is more likely to happen when using whole documents, and this can make the algorithm distinguish different target words better.

I use the whole document as a context in (b), now we try to make small context windows that contain target words, and experimented from size 5 to size 40, the result shows that the algorithm reaches higher accuracy as the context window size increases and reach maximum accuracy around 72% with context window size around 35. I think this may because larger context windows allow taking more context into consideration while if the context window sizes are small, and if the small context consists of several common words, then the different target words are likely to be cluster together because the algorithm cannot distinguish them, so basically, larger context windows allow more error-tolerance for the words that next to the target words.

(ii) Different type of features (no stemming vs. stemming)

Using stemming can reduce the feature dimension sizes, however, not much differences in terms of accuracy compared with (b), the accuracy is still around 30% - 40%. Although the context of a target word and its reverse may share more similar context after stemming, the target word is more likely to share similar context with other target words which may cause more than 1 pair to be clustered into the same cluster. I think that is why stemming is not necessarily a better option.

(iii) Different training data (corpusB vs. corpusC vs. corpusBC)

The accuracy for using corpusC and combination of corpusBC are much higher than using corpusB, the quality of training data indeed affects much. I think if in a corpus each target word has quite distinguishable contexts and for each time a target word appears, its context is similar, then the cluster on this corpus will have a higher accuracy. It seems that corpusC is of better quality of corpusB in terms of the standard above, and the combination is somewhere between the middle of the two corpus.

(iv) Different context window sizes using corpusC and corpusBC

In addition, I tried different sizes of context windows on corpusC and corpusBC, and they can reach very decent accuracy when context windows sizes can be more than 10, for corpusC, the context of each target words is more distinguishable, so we do not need a very large window to do the clustering, while the combination of corpusB and corpusC is affected by corpusB so it is slightly less accurate than corpusC but still better than corpusB. For all the corpus, using a good context window instead of whole documents will achieve better results. The maximum accuracy for corpusC is about 96% while the maximum accuracy for the combination is about 92%.

(v) Other features

Even if we consider small context window, we are still using the idea of “Bag of Words”, BOW has the limitation that we do not take the order of the sequence into consideration, so I think a new feature could be using multi-gram instead of single words, so if different order of the words have different meanings, we can also take those information into consideration when doing the clustering, which will make the target words more distinguishable.