Natural Language System Lab 2 Report

10406141 Wenchang Liu

1. Task 1: Named-entity recognition

(a) Using NLTK NER classifier

First we load the corpus and pos tagging the corpus like we did in lab1, then we can use ne_chunk() to get NER results. The result is a tree, we can convert it to a list by using tree2conlltags so that we can know the position of ORGANIZATION terms in the original document(so we can compare results with stanford NER). NLTK classified 895 ORGANIZATION terms in total and 223 terms remain if duplicates are removed.

(b) Using Stanford NER classifier

We still load the corpus like (a), then we can use stanford NER to get the results. Stanford NER cannot automatically combine words into phrases, so I combine them myself, I think if the words are all classified as ORGANIZATION and the positions in the original documents are consecutive, then I can safely say they belong to the same phrase. Stanford NER generated more precise results but less entities are recognized, it classified 302 ORGANIZATION terms in total and 72 terms remain if duplicates are removed.

(c) Compare results of ORGANIZATION class

- ➔ Which tool seems better in getting the bounders of named entities right?
  It seems that NLTK can recognize more entities, but in terms of quality, Stanford NER is better, especially in terms of bounders. For example, in NLTK NER, you can see cases when there are no words after 'of' (e.g. 'FAILURE OF', 'RELATION OF', 'Author of'), which are not making sense, while Stanford NER can even handle long phrases very well, e.g. 'Board of Engineers and Naval Commissioners', 'Permanent Court of International Justice', and 'Department of Commerce and Labor'.
- ➔ How many exact matches?
  Because we record each ORGANIZATION term's position in the original document(start and end index), so that we can compare whether NLTK and Stanford recognize a ORGANIZATION term at the same position(i.e. both start and end index are matched).
  There are 227 exact matches out of 302(Stanford recognized 302 ORGANIZATION terms in total).
- ➔ How many partial overlaps?
  Similar to the exact matches, this time we add an 'else if' condition as following: if a term from Stanford NER has start index a and end index b, and a term from NLTK NER has start index c and end index d, and if a > d or c > b, then the two terms are not overlapping, so we negate the condition to "if not(a>d or c>b)".
  There are 36 partial overlap cases out of 302(Stanford recognized 302 ORGANIZATION terms in total). And there are more than 87% of cases in which the two NER agreed totally or partially.

2. Task 2: Sentiment analysis of movie reviews

(a) Collect more adjectives by bootstrapping

- ➔ Patterns and Additional Patterns
  The patterns in my case are relying on pos tags and polarity signals. Firstly I use patterns like X and New_Word, X but New_Word, so when I encounter a seed word, I will look the next word, if the next word is ‘and’(or ‘but’) then I will check the next word, if the next word is also an adjective, then I will assign this adjective the same(or different) polarity as the seed word. After we went through this process across the whole corpus, we can get new adjectives with polarity. Apart from ‘X and Y’ and ‘X but Y’, I also implement additional patterns like ‘X, Y and Z’, ‘X but adv. Y’, ‘X rather Y’, ‘X rather than Y’. After the first iteration of bootstrapping(use the original seeds), there are in total 66 negative words and 94 positive words. We can run for more iterations to generate more adjectives by regarding the new generated adjectives as the seeds(I ran the second iteration and there are in total 102 negative words and 189 positive words), however, there can be more errors and it is not suitable for the manual evaluation later.
- ➔ Resolved Conflicts
  First, I assume that the seeds are the ground truth, so any cases that conflict with the seeds’ polarity will be assigned the polarity same as the seeds. I put all the adjectives with polarity into a dict(key is the word, value is polarity), each time a word is assigned the same polarity with a positive seed, the value will +1, and each time the word is assigned the same polarity with a negative seed, the value will -1. Therefore, in the end of iteration, we can iterate the dict and see whether each word has the polarity value bigger than 0 and assign the final polarity to that word(ignore the word if it is still 0).
- ➔ Results Evaluation
  Since the patterns I use are relatively conservative, the overall correctness is good and there are not many errors. There are some rare cases where I got the polarity wrong because I did not take the whole sentence into consideration.
  For negative words, ‘sanguine’ and ‘affable’ should be positive words, they were assigned wrong polarity, for ‘sanguine’, the review says “between sappy and sanguine”, but I only check for pattern “X and Y” which is the opposite, and for ‘affable’, I assume in pattern ‘X but adv. Y’ the adv. will not negate Y, however, the review says “affable but undernourished romantic”. For positive words, ‘sad’, ‘unsatisfying’, were assigned wrong polarity, for ‘sad’, the reviewer just use ‘funny and sad’ to represent two polarity of life, it is a rare case, for ‘unsatisfying’, the review says ‘creepy but ultimately unsatisfying’ for a thriller movie so creepy is actually a positive word here.

(b) Movie reviews binary classification

- ➔ Baseline
  Firstly, load all positive and negative words in mpqa into a dict(key is the word, value is positive or negative), then we read each line from corpus2 and count the number of positive and negative words, we assign polarity to the review to the one with more counts. The overall accuracy for the baseline classifier is around 51.3%.
- ➔ Simple BoW
  I use CountVectorizer from scikit-learn on all the lines of reviews directly to build review-words matrix, there are 18330 features for the simple BoW model, then I use 5 fold cross validation and both logistic regression and naive bayes classifier to do the classification, the overall accuracy for logistic regression classifier is 76.5% while 77.6% for naive bayes.
- ➔ Extra features and modifications
  I have tried 4 approaches to improve the accuracy, the final accuracy for logistic regression is 77.4% and 78.8% for naive bayes:
  1) Use tf-idf word embedding instead of word counts (even worse)
     We can simply do this by using TfidfVectorizer instead of CountVectorizer, however, the accuracy gets even worse, it may because some important polarity words occur very often, and tf-idf will reduce the significance of them.
  2) Deal with negation (not using this eventually)
     If there is a negation(e.g. n't, no, never), we add a prefix 'NOT_' to all words afterwards, however, the accuracy did not get improved(almost the same but takes longer time, may because some words after negation should not be negated)
  3) Add features of how many words comes from mpqa lexicon
     We use mpqa data to help the classifier by adding extra features telling the classifier how many positive words and negative words the review has according to mpqa data, the accuracy improved a little bit(around 0.5% for naive bayes).
  4) Remove numbers from feature sets
     There are features which are pure numbers in the BoW feature set initially, which are not helpful for sentiment judgement, and after removing them, there are about 18156 features left. The accuracy improved 0.5% for naive bayes and 1% for logistic regression.
- ➔ Comparison
  The baseline merely gets 51.3% accuracy which is almost random, I think it may because it ignores all the contexts. When using the simple BoW model, it takes all words into consideration, so the classifier can understand some of the semantics(i.e. different combinations of words represent different semantics), and the accuracies improve significantly to more than 75%. After we added lexicon features and removed irrelevant features, the classifiers got further improvement about 1%. However, the improvement is still limited, because even if we add some features, most of the features come from BoW. I think we need to consider sequences of words or some novel approaches like doc2vec if we want to achieve significant improvement in terms of accuracy.