Natural Language Processing (CS 6120/4120)

Assignment 1

Language Modeling, Sequence Modeling - POS Tagging (HMM), Naive Bayes for Text Categorization, Word Sense Disambiguation

Instructor: Professor Lu Wang

Due Date: 6pm, October 9^{th}, 2017

For the programming questions, you can use Java, Python or C/C++. You also need to include a README file with detailed instructions on how to run the code and commands to facilitate understanding for TAs. Failure to provide a README will result in points deduction. No need to print out the code while submitting the hard copy, just turn it in on Blackboard.

1 Naive Bayes for Text Categorization (15 points)

Given the following short movie reviews, each labeled with a genre (class):

1. fun, couple, love, love : **comedy**
2. fast, furious, shoot : **action**
3. couple, fly, fast, fun, fun : **comedy**
4. furious, shoot, shoot, fun : **action**
5. fly, fast, shoot, love : **action**

And a new document D: fast, couple, shoot, fly

Compute the most likely class for D. Assume a naive Bayes classifier and use Laplace smoothing (add one) for the likelihoods.

2 Word Sense Disambiguation (10 points)

Given a paragraph below:
But we refuse to believe that the bank of justice is bankrupt. We refuse to believe that there are insufficient funds in the great vaults of opportunity of this nation. So we have come to cash this check - a check that will give us upon demand the riches of freedom and the security of justice.

2.1 Using WordNet (`http://wordnetweb.princeton.edu/perl/webwn`), determine total number of senses for each open class word in each sentence. (5 points)

2.2 Using WordNet (`http://wordnetweb.princeton.edu/perl/webwn`), determine how many distinct combinations of senses are there for each sentence. (5 points)

3 Language Modeling (35 points)

There are 3 files namely, testset.csv, trainset.csv and devset.csv, collected from Switchboard Corpus (`http://compprag.christopherpotts.net/swda.html`). Implement trigram language model with unknown word handling (replace words of frequency less than 5 as UNK).

1. [**10 points**] Write code for add-1 smoothing, compute perplexity on test set.
2. [**10 points**] Write code for simple interpolation, compute perplexity on test set.
3. [**10 points**] Write code to generate sentences from above 1 and 2. Print out 20 sentences for each model.
4. [**5 points**] Compare the two smoothing methods and analyze which one is better and why.

Please refer to Switchboard Corpus (`http://compprag.christopherpotts.net/swda.html`) for data format. Here, trainset.csv has 70% of the total data, and devset.csv and testset.csv each has 15%.

The Switchboard Corpus is a telephone speech corpus containing conversations between people. It has the fields namely, filename, basename, conversationId, numberoflines in conversation, act tags, conversing person ID, utterance indices, subutterance and the text (conversation) itself.

Steps to be taken for each file:

1. You can use nltk to extract the utterances from the files or use your own scripts.
2. Extract the text field from each line of each csv file.
3. Remove all the unnecessary symbols leaving behind meaningful words.
4. Lower case all the words.

5. Keep a count of each word appearing in the corpus which helps you generalize words which appear less than 5 times as "UNK".

6. Calculate the probability using add-1 (for part-1) and simple interpolation (for part-2) and then compute perplexity.

4 POS Tagging - HMM (40 points)

Implement part-of-speech tagger using bigram HMM for given corpus. You have been given a dataset in English from Universal Dependencies (`http://universaldependencies.org/#en`).

Refer this link (`http://universaldependencies.org/en/pos/all.html`) to see all possible tags in the corpus. Though it is helpful to understand actual data files, we have provided some preprocessed files for ease of development as described below.

1. **train.counts:** Frequency counts from the training file. It has two types of counts:

   (a) Emission counts, in the format
   <count> WORDTAG <tag> <word>

   (b) n-gram counts (where N is 1, 2 or 3), in the format
   <count> <N-gram> <tag 1> ... <tag N>

2. **test.words:** Simplified version of the original test file, with one word per line and blank lines separating sentence.

3. **test.tags:** Simplified version of the original test file, with word and the true tag on each line and blank lines separating sentences.
   <word> <tag>

For POS tagging, you will use HMM decoding - given an observation sequence of n words w_1^n and a set of hidden states (POS tags) t_1^n, choose the most probable sequence of tags. Following equation gives an intuition behind Viterbi algorithm for decoding (eq. **10.9** in book - `http://web.stanford.edu/~jurafsky/slp3/10.pdf`).

$$\hat{t}_1^n = \underset{t_1^n}{argmax}\, P(t_1^n, w_1^n) \approx \underset{t_1^n}{argmax} \prod_{i=1}^{n} \overbrace{P(w_i, t_i)}^{emission} \overbrace{P(t_i, t_{i-1})}^{transition}$$

1. **[10 points]** Emission probability is the probability of a given word (e.g. teams) being associated with a given tag (e.g. NOUN). Calculate emission probabilities of the training set using following equation.

$$P(w_i, t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

2. **[10 points]** Transition probability is the probability of a tag given its previous tag. Calculate transition probabilities of the training set using following equation.

$$P(t_i, t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

3. **[20 points]** Derive the most probable tag sequence using Viterbi algorithm (pseudocode can be found in book (`http://web.stanford.edu/~jurafsky/slp3/10.pdf`) under **figure 10.8**.). Store output tags derived using Viterbi algorithm for each word in test in a separate file in same format as test.tags (<**word**> <**tag**>). You can use eval_tagger.py to compute accuracy of your output against actual tags.

- You need to submit your code, README.txt file explaining how to run your code, and the output file with predictions in given format.
- Read Chapter 10 of text-book from `http://web.stanford.edu/~jurafsky/slp3/10.pdf` for your reference.