CS470 Introduction to Artificial Intelligence Fall 2015

Homework 6 – Natural Language Processing

TA Zae Myung Kim (zaemyung@kaist.ac.kr)
TA Jonghwan Hyeon (hyeon0145@kaist.ac.kr)

Date assigned: Mon. 23 November 2015
Date due: Wed. 2 December 2015

Submit one zip file containing your report and program codes via KLMS (naming format of the zip file: <student#>_<name>.zip, for example "20150724_john.zip"). Late submissions will not be accepted. (Write your name and student ID in your report.) This homework should be done individually and written in English.

1. Cocke–Younger–Kasami (CYK) parsing (40 points)

Consider the following grammar:

S → NP VP
VP → VBG NNS
VP → VBZ VP
VP → VBZ NP
NP → DT NN
NP → JJ NNS
DT → **an**
NN → **actress**
VBZ → **likes**
VBG → **playing**
JJ → **playing**
NNS → **kids**

Questions

A. Draw all the parse trees in this grammar for the sentence: **"an actress likes playing kids"**

B. Show all the table entries that would be made by a (non-probabilistic) CYK parser on this sentence.

2. Linguistic Ambiguities (20 points)

Consider the following sentence (from *The New York Times*, July 28, 2008):

> Banks struggling to recover from multibillion-dollar loans on real estate are curtailing loans to American businesses, depriving even healthy companies of money for expansion and hiring.

Questions

A. Which of the words in this sentence are lexically ambiguous?

B. Find two cases of syntactic ambiguity in this sentence (there are more than two.)

C. Give an instance of metaphor in this sentence.

D. Can you find semantic ambiguity?

3. Classifying Spams (40 points)

In this problem, we are going to implement an SMS spam classifier using naïve Bayes algorithm. You can refresh your memory on naïve Bayes classifier at: (http://en.wikipedia.org/wiki/Naive_Bayes_classifier)

Our dataset is organized as follows. The training set (*train*) contains 3861 ham messages and 597 spam messages. The test set (*test*) contains 966 ham messages and 150 spam messages. Each line contains the class label (either "spam" or "ham") followed by "|" and the SMS message as shown below:

```
spam|FreeMsg>FAV XMAS TONES!Reply REAL
```

Rather than using the raw input stream, we will perform two text preprocessing techniques.

1) Stop words removal

Stop words refer to words that are very commonly used in a given language. This makes them meaningless for any kind of classification. For example, in English you have words such as "the", "to", "you", "he", "only", "if", "it" that you can safely strip out from the text. A file containing the list of stop words are included in the zip file.

2) Stemming

Also, because our classifier cannot understand the context of words in a sentence, it would treat two words differently even though they share the same morphological root. For example, consider the word "running". You would want to treat the word same as the

word “run”, which is the morphological root of the word. You would also want to treat “studying” and “studies” as “study”.

This process is called stemming and there are various algorithms for it. In this project, we utilize a simple rule-based stemming algorithm called Porter Stemming.

Refer to http://9ol.es/porter_js_demo.html for an online demo. The algorithm is already implemented in NLTK toolkit: http://www.nltk.org/howto/stem.html Use this library when performing stemming.

3) Representing feature words

Due to high dimensionality of bag-of-words model and relatively short length of SMS messages, we need to devise an effective set of features. In this assignment, we closely look at the spam messages, and observe the characteristics that are prevalent in the spam data, but not really so in the ham messages.

URLs: The spam messages tend to contain a link to a website. Rather than having each separate website link as a distinct word, we can represent any link by a special word, say, `URL`. We can do this by using regular expressions: http://regexone.com/

In python, the `re` package does the job: https://docs.python.org/2/library/re.html

For matching URLs, take a look at https://mathiasbynens.be/demo/url-regex and choose one you think is good.

Phone numbers: Another notable feature is the presence of phone numbers. We can observe that spam messages tend to contain phone numbers more than ham messages. Devise a regex for capturing phone numbers and replace them by `PHONENO`.

Feel free to come up with additional features if you think this would help the classification accuracy.

4) Limit the size of vocabulary

Once the features are defined, we represent the corresponding words in the training set with these features. We then limit our vocabulary size to 10000, i.e., we take the top 10000 words (*after stopword filtering*).

Replace words that are not in the vocabulary by `UNKNOWN`.

Questions

A. Define regular expressions for capturing URLs and phone numbers, and replace such occurrences in the dataset with corresponding word tokens.

B. Implement **def filter_stopwords**(string, list of stopwords) that takes an SMS message and a list of stopwords, and returns a list of words with the stopwords removed. Note that you need to tokenize the string first using the same toolkit: http://www.nltk.org/api/nltk.tokenize.html

C. Limit the size of vocabulary to 10000, and replace words outside the vocabulary by UNKNOWN token.

D. Implement **def do_stemming**(string) that takes the list of words and returns a list of stemmed words using the Porter stemmer of NLTK toolkit.

E. Implement **def train**(list of strings, list of strings) that takes the list of preprocessed SMS messages with corresponding list of class labels, i.e., spam or ham, and constructs naïve Bayes classifier.

F. Implement **def classify**(string) that takes a new SMS message and returns its predicted class label.

G. Report your precision, recall, and f1-measure on the test set.

https://en.wikipedia.org/wiki/Precision_and_recall

Note that you must **not** use the test set when training the classifier. **You need to implement the naïve Bayes classifier by yourself – do not use an external library for this.**
Annotate your codes with detailed comments, clearly illustrating how each task above is implemented.