CIS 430/530 Fall 2011 HW 2

Instructor: Ani Nenkova
TA: Alexander Shoulson

Released: September 30, 2011
Due: 11:59PM October 17, 2011

Overview

This assignment focuses on helping you get started with using NLTK and Python by performing simple text analysis. You will also begin to develop intuitions about different analytical techniques. The first two chapters of the NLTK book should suffice for this assignment. For information about how to get started with the NLTK tool on eniac, please see the instructions posted at:

`http://www.cis.upenn.edu/~cis530/hw_2011/generalinfo.pdf`

NOTE: All numbers in this document are for example purposes only and do not reflect actual results.

Deliverables (Total: 150 Points)

You will be asked to submit the code for the functions you have implemented as well as a brief writeup of your observations and results. Write ample comments in your code.

NOTE: If you do not have an account on Eniac, please contact the TA immediately.

Submitting your work

The code for your assignment should be placed in a single file called hw2_code_yourpennkey.py where yourpennkey is your Penn Key. The writeup should be in plain text format and named as hw2_writeup_yourpennkey.txt. Since my (Alex) pennkey is "shoulson", I would submit the following files: hw2_code_shoulson.py, hw2_writeup_shoulson.txt

To electronically submit homework, if you are not already working on Eniac, you need to place the file containing your solution on your SEAS account storage. One way to do this would be to use an SFTP client such as FileZilla or WinSCP.

Then connect via ssh to seas.upenn.edu and use the turnin command to submit your files for grading:

```
% turnin -c cis530 -p hw2 hw2_code_yourpennkey.py hw2_writeup_yourpennkey.txt
```

This should print out a confirmation message. If you are prompted for a section name, type 'ALL. You can run turnin multiple times before the deadline. Each time you run turnin, it overwrites your previous submission for that assignment. You can check that the homework was submitted successfully:

```
% turnin -c cis530 -v
```

This will show you the list of file(s) you have submitted.

Code Guidelines

You can use in-built NLTK and matplotlib modules whenever possible unless specified otherwise. However, only import the modules that you are actually using. For example:

```
from nltk import * # Bad!
from nltk import FreqDist, ConditionalFreqDist # Good!
```

You may write any extra helper functions that you think are necessary, but all the functions defined in this document should be present. Please add descriptive comments to all functions that you write. Do not do any preprocessing/filtering of data unless explicitly mentioned. For example, the words of a set of documents, refers to all tokens, even non-alphabetic ones.

Do not leave any code statements in the body of your file, with the exception of global variables (if you choose to use any). Place any test or execution code in your `if __name__ == '__main__':` block. The rest of your code should be in the functions you write for this assignment, or in helper functions.

Data

We are providing you with a set of news articles from the New York Times corpus. The documents provided to you are taken from four topics. These topics were manually assigned to the articles by editors at the New York Times. An article can be given more than one tag simultaneously. These topics are as follows:

- Finance
- Health
- Research
- Computers and the Internet

All the data has been cleaned of tags and sentence segmented. The data is located in the following directory:

`/home1/c/cis530/data-hw2/`

We have divided the above set of documents into training and test sets for this homework. These files are located in the following directories:

- *Training_set_large*: to use for training the classifier
- *Test_set*: the set of documents on which we will test the accuracy of classification
- *Training_set_small*: second training set with fewer documents than the first training set
- *Language_ model_set*: a set of documents on each topic on which you will create language models. These models will then be used to derive some features for topic classification and for a word prediction task.
- *stopwlist.txt*: a file with commonly used words. These words will be filtered from the documents when necessary

Each of the above training/test folders have subfolders for each topic, but the main folder as a whole constitutes the training or test set. In other words, the each set is the mixture of documents from all the subtopics.

Loading a Corpus Using NLTK

Here is an example of how to load files from a particular topic into memory. You should refer to NLTK book chapter 2 for more details about corpus-based functions in NLTK.

```
# Import the corpus reader
>>> from nltk.corpus import PlaintextCorpusReade
# Define the folder where the files are situated for a topic
>>> corpus_root = /home1/c/cis530/data-hw2/Training_set_large/Finance
# Read all files in that directory
>>> files_finance = PlaintextCorpusReader(corpus_root, '.*')
# List 3 files from this topic
>>> files_finance.fileids()[:3]
['2005_01_01_1638674.txt', '2005_01_01_1638675.txt', '2005_01_01_1638697.txt']
# Total number of files belonging to this topic
>>> len(files_finance.fileids())
2000
```

1 Problem 1: Topic Classification (100 points)

You will be training an automatic classifier to identify given a document, the topic it belongs to.

1.1 Coarse-Level Features (10 points)

Write a function get `coarse_level_features(dataset, output_file)` to compute some features based on the overall properties of each document. This function takes a top level directory name such as `'Training_set_large'` and computes the following set of features for each document in that set (over all categories).

- Number of tokens
- Number of types
- Number of content words (after removing the words contained in *stopwlist.txt*)
- Number of sentences
- Average sentence length
- Number of capitalized words (as a proxy for proper names)

The output file should be written in the following format for each line:

```
docid topic_name num_tokens num_types num_cont num_sents avg_slen num_caps
```

For example, the following command:

```
get_coarse_level_features('Training_set_small', 'Training_set_small.coarsefeatures')
```

Should generate the output file below in your working directory:

Training_set_small.coarsefeatures

```
Training set small/Health/2005_01_02_1639009.txt Health tok:878 typ:510 con:509
sen:67 len:13.10 cap:166
Training set small/Finance/2005_10_25_1712355.txt Finance tok:1250 typ:622 con:716
sen:87 len:14.36 cap:188
Training set small/Research/2005_01_11_1641065.txt Research tok:642 typ:327 con:352
sen:39 len:16.46 cap:67
Training set small/Research/2005_01_12_1641336.txt Research tok:480 typ:270 con:293
sen:23 len:20.86 cap:88
```

NOTE: Use the colon separator format above. (feature:value)
Each entry contains the path to a file and the topic name (Research, Finance) followed by the features listed above in order. The items on a line must be separated by spaces.

1.2 Part-of-Speech Features (20 points)

You will be using the NLTK maxent tagger. This is accessible using the `nltk.pos_tag()` function. If you get errors that the function is not available, type:

```
>>> nltk.download()
```

and choose the option, 'all packages used in the NLTK book'.

An example usage is provided below:

```
>>> text = [The, dog, ran, .]
>>> nltk.pos_tag(text)
[(The, DT), (dog, NN), (ran, VBD), (., .)]
```

First, we will prepare which features we would like to compute and we will compute these features for each document in the next step.

1.2.1 Data Preparation

Write a function `prepare_pos_features(Language_model_set, output_file)`. The first argument is the name of the language model set directory (which is *Language_model_set* in the data-hw2 directory). Tag the documents contained within this directory and find the following word classes over all the files taken together:

- Most frequent 200 nouns
- Most frequent 200 verbs
- Most frequent 200 adjectives
- Most frequent 100 adverbs
- Most frequent 100 prepositions

The tagger uses more than these five parts of speech. Merge these tags as follows:

- Nouns = NN, NNS, NP, NPS
- Verbs = VB, VBD, VBG, VBN, VBP, VBZ
- Adjectives = JJ, JJR, JJS
- Adverbs = RB, RBR, RBS

- Prepositions = IN

Create the output file in the following way. Add a prefix to each word indicating whether it was a frequent noun, verb etc. as follows:

- Noun = NN
- Verb = VV
- Adjective = ADJ
- Adverb = ADV
- Preposition = PREP

An example output file would look like this:

```
NNmarket
NNcancer
...
VVrising
...
PREPbelow
...
```

1.2.2 Computing Features

Write a function `get_pos_features(dataset, feature_set_file, output_file)`. It takes a top level directory such as `'Training_set_large'` and computes the following set of features for each document in that set (all subdirectories included). The `feature_set_file` is a list of words created by the function `prepare_pos_features()`.

For each word (together with part of speech) find out if the word appears in the given document with that part of speech. To know this, you will need to tag the current document as well. If it appears, record the number of times the word appears with the given part of speech.

Write the computed values to the `output_file` such that the features appear in the same order as the file `feature_set_file`. If a particular word-pos combination does not appear in the given document, that features value would be a zero.

An example output file may look like the following:

Training_set_small.posfeatures

```
Training_set_small/Health/2005_01_02_1639009.txt Health NNmarket:1 NNcancer:12 ...
VVrising:0 ... PREPbelow:7...
Training_set_small/Finance/2005_10_25_1712355.txt Finance NNmarket:6 NNcancer:0 ...
VVrising:5 ... PREPbelow:0...
Training_set_small/Research/2005_01_11_1641065.txt Research NNmarket:2 NNcancer:5 ...
VVrising:1 ... PREPbelow:1...
Training_set_small/Research/2005_01_12_1641336.txt Research NNmarket:0 NNcancer:2 ...
VVrising:0 ... PREPbelow:2...
```

NOTE: All lines will have the same number of features, the same number as the words in feature set file.

1.3 Language Model Features (20 points)

Here you will compute two types of features based on language modelslikelihood and perplexity of a document. Using the subfolders one for each topic in Language model set, create a bigram language model for each topic. **Use Laplace smoothing.**

Write a function `get_lm_features(dataset, output_file)` which computes the following set of features for each document:

- Log probability under the Finance bigram model
- Log probability under the Health bigram model
- Log probability under the Research bigram model
- Log probability under the Computers_and_the_Internet bigram model
- Log perplexity under the Finance bigram model
- Log perplexity under the Health bigram model
- Log perplexity under the Research bigram model
- Log perplexity under the Computers_and_the_Internet bigram model

Sample output:
Training_set_small.lmfeatures

```
Training_set_small/Finance/2005_01_02_1639022.txt Finance finprob:-2133.4 hlprob:-2112.3
resprob:-2148.4 coprob:-2127.3 finper:1233.4 hlper:2112.3 resper:2148.4 coper:2127.3
Training_set_small/Finance/2005_01_02_1639028.txt Finance finprob:-13015.0 hlprob:-12982.2
resprob:-13145.2 coprob:-13087.8 finper:13015.0 htper:12982.2 resper:13145.2 coper:13087.8
Training_set_small/Health/2005_01_18_1642925.txt Health finprob:-6577.7 hlprob:-6397.8
resprob:-6479.4 coprob:-6566.4 finper:6577.7 hlper:6397.8 resper:6479.4 coper:6566.4
Training_set_small/Health/2005_01_18_1643011.txt Health finprob:-14045.6 hlprob:-14195.6
resprob:-14387.3 coprob:-14411.9 finper:14045.6 hlper:14195.6 resper:14387.3 coper:14411.9
```

1.4 Combining Features

Write a utility function `get_feature_file(directory_name, features_to_use, output_file)` to combine sets of features before training and classification. Parameter `directory_name` will be one of `'Training_set_large`, `'Training_set_small` or `Test_set`. Parameter `features_to_use` is a list of strings specifying which sets of features should combined for the output file. The possible options are `'coarse'`, `'pos'`, or `'lm'`. For example, if `features_to_use = ['pos', 'lm']`, then the part of speech and language model features should be combined for each document and written to `output_file`. The format for output file is the same as the previous sections with document name, topic name and then the list of features separated by spaces, only that the features from the specified categories would be written together for each document.

Example usage:

```
>>> get_feature_file(Training_set_large, [pos, lm], Train_large_poslm_features)
```

You can now use this function whenever you want to combine two sets of features for training and testing a classifier.

NOTE: Before classifying, make sure that your training and test set features match. For example, if you want to combine language model and pos features, use this function to combine the features for both training and test sets. Then train and test using the function below.

1.5 Classifying Topics (20 points)

Use the NLTK class `nltk.NaiveBayesClassifier` and train a classifier to classify a given document into one of the 4 topics. Please read over the examples in Chapter 6 of the NLTK book.

Write a function `get_NB_classifier(training_examples)` which returns an instance to a Naive Bayes classifier trained on a given set of documents. The `training_examples` parameter is a file formatted according to the output of your feature-collecting functions, as above.

Write a function `classify_documents(test_examples, model, classifier_output)`. The `test_examples` parameter is, again, a file with features and `model` is a Naive Bayes trained model such as one returned by the `get_NB_classifier` function. Parameter `classifier_output` specifies a file where you should write the label returned by the classifier for each document in test examples. The format of classifier output is:

```
document_id true_topic predicted_topic
```

Where `true_topic` is the actual topic of the document, and `predicted_topic` is the label returned by the classifier. An example usage is as follows:

```
>>> poslm_model = get_NB_classifier('Train_large_poslm_features')
>>> classify_documents('Test_poslm_features', poslm_model, 'Test_poslm_features.pred')
```

Test_poslm_features.pred

```
Test_set/Finance/2006_05_26 1764479.txt Finance Computers and the Internet
Test_set/Finance/2006_05_27 1764608.txt Finance Finance
Test_set/Research/2006_05_27 1764610.txt Finance Health
Test_set/Health/2006_05_27 1764613.txt Finance Health
```

1.6 [WRITEUP] Accuracy of Topic Classification (30 Points)

Based on the classification results for the 4 topics, answer the following. The test set is always the documents in *Test_set*.

- What is the accuracy of classification for the topics? Combine all three sets of features (coarse, pos and lm), train on *Training_set_large*. Write down the confusion matrix for the classification. (Refer to section 6.3 in the NLTK book for an example.)

  Which topics did you expect that can be classified more accurately? Which topics are frequently confused with other topics? Do the results conform to your intuitions?

- Report the number of documents from each topic present in the training and test sets. Which is the larger category, which is the smallest? What effect do you think this distribution will have on the classification accuracy? Do you observe this difference in your accuracy results?

- What are the accuracies using each feature set (coarse, lm, pos) individually? Use two different combinations of 2 feature sets that you think will be most beneficial. For example, you may choose to combine language model and coarse features. Why do you expect the combination to be better than individual features? Do the results conform to your hyptheses? Which combination worked best?

- What is the accuracy of classification based on the smaller training set *Training set small*? What is the effect on accuracy after reducing the training set size? Explain your findings.

For each of the following classification tasks, list 5 features that you think will be helpful. You need not worry about how to implement these but be specific in your answer. For each feature, explain in one sentence why you expect it to be useful for the respective task.

- Spam email vs. genuine email

- Separating positive vs. negative reviews for products on Amazon
- Classifying a document as easy or difficult to read

2 Problem 2: Word Prediction (50 points)

Using the language model class you had implemented in the previous homework, train bigram models for the documents from each topic. Use the document sets provided in *Language_model_set* and Laplace method for smoothing. For this task, you will be given sentences with blanks in between and a set of choices for words to fill in. You should predict which word fills a particular blank based on a given language model.

An example sentence:

`Stocks <blank> this morning.`
i) rise ii) walked iii) discovered iv) plunged

2.1 Fit of a Word (15 Points)

Write a function `get_fit_for_word(sentence, word, langmodel)`. Parameter `sentence` is a sentence string such as the one in our example above and word is one of the choices provided for that sentence. Using the bigram language model supplied in `langmodel`, find the log probability of the sentence after inserting the word in the given blank. This log probability will be the return value for this function. The token `<blank>` will be used to represent the blank in a sentence, so your program could check for this position.

Example usage:

```
>>> sent = 'Stocks <blank> this morning.'
>>> get_fit_for_word(sent, 'walked', 'Health')
-92.394
```

2.2 Best Fit for a Sentence (15 points)

Here, write a function `get_bestfit_topic(sentence, wordlist, topic)`. This function returns one of the words from wordlist. This word is such that when it is inserted into the sentence, the language model for the specified topic assigns the highest probability for the given sentence compared to the other words in wordlist. wordlist is the list of choices provided for insertion.

Example usage:

```
>>> sent = 'Stocks <blank> this morning.'
>>> possible_words = ['rise', 'walked', 'discovered', 'plunged']
>>> get_bestfit_topic(sent, possible_words, 'Health')
'rise'
```

2.3 [WRITEUP] Automatic Language Prediction (20 Points)

Here are a few sentences with some blanks and a set of words which could probably fill in this space. You will now assess the ability of language models from our 4 topics to predict the right word to fill in.

`[a] Stocks <blank> this morning.`
i) rise ii) walked iii) discovered iv) plunged

`[b] Stocks plunged this morning, despite a cut in interest <blank> by the Federal Reserve.`

i) patients ii) rates iii) researchers iv) levels

`[c] Stocks plunged this morning, despite a cut in interest rates by the <blank> Reserve.`
i) bank ii) university iii) Federal iv) Internet

`[d] Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began <blank> for the first time.`
i) trading ii) wondering iii) recovering iv) hiring

`[e] Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last Tuesdays <blank> attacks.`
i) heart ii) terrorist iii) doctor iv) alien

`[f] Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last Tuesdays terrorist attacks.`

- Which of the four topics language models do you think will be best in predicting the blanks for the above sentences `[a]` to `[e]`? Explain.
- Use the `get_bestfit_topic(sentence, wordlist, topic)` function developed for problem 2, to identify the word that would be chosen to fill the blank under each topics bigram language model. For example, you may find that the `'Health'` language model chooses option 1 (gives highest probability to the sentence after this word is inserted compared to other words) and `'Research'` chooses option 2. Since there are four topics, for each of the sentences `[a]` to `[e]`, write down which word is predicted to be the best fit according to each topic.
- Do the results conform with your intuitions about the best predicting topic? What is the accuracy of prediction by the topic you thought would be best? What is the accuracy of prediction using other topics language models? Is there a model which has a clear advantage over others?
- Which topic is worst at predicting the blanks?
- Compute the log probability of the sentence `[f]` using the language models from each of the 4 topics. Which topic assigns the highest probability to the complete sentence? Is this result the one you would expect?