CIS 430/530 Fall 2012 HW 3

Instructor: Ani Nenkova
TA: Kai Hong, Jessy Li, Achal Shah

Released: November 3, 2012
Due: 11:59PM November 16, 2011

Overview

This assignment focuses on sentiment analysis, name entity recognition, part of speech tagging, and classification using SVMs.

Deliverables

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 or if you cannot access Biglab, please contact the TA immediately. Large jobs on Eniac will likely get terminated, so make sure you can access **Biglab** and run your code there. Also, make sure you **START EARLY**! Running **name entity recognition** and **part of speech tagging** takes time, so you want to make sure that you get to it as soon as you can!

Submitting your work

The code for your assignment should be placed in a single file called hw3_code_yourpennkey.py where yourpennkey is your Penn Key. Since my (Achal) pennkey is "achals", I would submit the following files: hw3_code_achals.py

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 hw3 hw3_code_yourpennkey.py scores.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.

Data

In previous homework, you have seen some of the announcements/situations about a company. Actually, as you can imagine, each announcement can lead to a change of condition in the stock market. In this homework, rather than giving you raw text for each announcement, we will give you two .csv files which include the headline, situation about a company's announcement as well as the change of stock price. **Your task is to predict the changing of stock price 2 days after announcement date as accurate as you can.** To simply the task, we just need to predict whether the stock prize raises or drops. Thus there're only two kinds of conditions which could happen.

The data you will be using comes from two locations from the CSV file: the column **situation** which includes the news at that day, and the column **xret2** which includes the stock price's change 2 days after the news entry, For this field, which is essentially what we will be classifying on, consider a positive change to be of class 1, and a negative change to be of class -1. You can only use the raw text from **situation** to extract your features. All of the following features afterwards are extracted from this entry.

The data you will be using is located at `/home1/c/cis530/hw3/`. This folder contains two files, `training.csv` and `testing.csv`, in which you will be using **situation** and **xret2** in training to predict the **xret2** in testing, learning from **situation** in testing data. We hope you can have a deeper understanding of NLP-techniques as well as comparing various kinds of features of predicting the status of stock price from this task.

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. Unless specified otherwise, all the functions you are asked to write should be top level functions that are not part of a class.

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.

****Ambiguity is a very common problem in NLP. For this homework, we will only show you how the answer should LOOK LIKE instead of giving you the answer ran on a smaller corpus. Your code will be both automatically and manually graded, so be sure to put comments and justify the way you solve a problem, especially whenever you feel there can be an ambiguity. Try to write clean code as well.**

Overview

This assignment focuses on sentiment analysis and classification using SVMs. You will again get the chance to become familiar with several resources: two subjectivity lexicons, an SVM classifier and will run a POS tagger. The textual data, as in the previous homeworks, consists of CIQ Key Development news summaries about various companies. We have split the companies into two classes: those that had excess return in the two days after the news were published that was *higher* than that for the two days before the news and those for which it was *lower*. The task is to predict in which group a company will fall, based of features derived from a single news item. There are many simplifications, but the general goal is to predict what features from the news may predict company performance

1 Problem: Lexical features (5 points)

For the task of classification, one category of the features we want to use is lexical features for all the documents.

1.1 Extracting the top words having more than 5 occurrences (2 points)

Write a function `extract_top_words(csv_file)` that takes in the path for the data directory and returns a list that contains the words from all the news entries, that have more than 5 occurrences in the entire corpus.

This list is what we will now compare the individual news entires to, as follows.

1.2 Creating lexical features per news entry (3 points)

Write a function, `map_entry(entry, top_words)` that takes in a news entry, and the list generated from the previous function, and returns a list of the same size, with each column containing the number of the times the corresponding word appeared in the text entry. If the word does not appear in the text entry, the value should be 0.

2 Problem: Sentiment dictionary features (20 points)

We want to see whether the use of hand-curated sentiment word lists can be used to help predicting the performance of the company. For this purpose, we will use two wordlists: The MPQA lexicon, and the General Inquierer lexicon.

The MPQA lexicon is a lexicon of words that have been tagged with their perceived subjective polarity, which can be either **negative**, **positive** or **neutral**. It can be found at:

```
/project/cis/nlp/data/corpora/mpqa-lexicon/subjclueslen1-HLTEMNLP05.tff
```

A readme file is also available in the same directory, and explains the layout of the corpora.
The General Inuquirer tagsetis a lexicon of words that have been tagged with their percieved subjective poolarity, which can be either negative or positive. There are also various other tags that the tagset provides. More information can be found at: `http://www.wjh.harvard.edu/~inquirer/homecat.htm` The file containing words annotated with this tag set can be found at:

```
/project/cis/nlp/data/corpora/inquirerTags.txt
```

2.1 Grabbing the MPQA lexicon (4 points)

Write a function, `get_mpqa_lexicon(lexicon_path)` that should access the MPQA corpus file above, and returns a dictionary which maps a word to a tuple, the first element of the tuple being the word type for the word, and the second element being the sentiment of the word. The dictionary should look something like:

```
>>> mpqa['best']
('strongsubj', 'positive')
```

2.2 Feature vector from MPQA lexicon (6 points)

Write a function, `get_mpqa_features(text, dictionary)`, that given a text and the dictionary produced from the previous function, should produce a tuple of three elements, the first element being the number of words with positive polarity in the text, and the second element being the number of words with negative polarity, and the third element being the words with neutral polarity.

```
>>> mpqa_dict = get_mpqa_lexicon(path)
>>> get_mpqa_features(text, mpqa_dict)
(10, 1, 0)
```

Write a function, `get_mpqa_features_wordtype(text, dictionary)`, that given a text and the dictionary produced from the previous function, should produce a tuple of six elements, the first three element being the number of words of strong subject type, with positive, negative and neutral polarity, the other three elements being the same counts, but for weak subject type words.

```
>>> mpqa_dict = get_mpqa_lexicon(path)
>>> get_mpqa_features_wordtype(text, mpqa_dict)
(4, 1, 0, 6, 0, 0)
```

This function results in a tuple for each text entry to the function, representing their mapping into the MPQA sentiement dimensions.

2.3 Grabbing the General Inquirer lexicon (4 points)

Write a function, `get_geninq_lexicon(lexicon_path)` that should access the corpus file above, and return a dictionary consisting of words associated with a tuple containing two elements, the first associating the word with either it's positive (GI called it **Positiv**) or negative (**Negativ**) polarity, the second element associating the word with it's **Strong** or **Weak** polarity, similar to the previous the MPQA dictionary.

2.4 Feature vector from General Inquirer lexicon (6 points)

Write a function, `get_geninq_features(text, dictionary)`, that given a text and the dictionary produced from the previous function, should produce a tuple of two elements, the first element being the number of words with positive polarity in the text, and the second element being the number of words with negative polarity.

```
>>> geninq_dict = get_geninq_lexicon(path)
>>> get_geninq_features(text, geninq_dict)
(6, 1)
```

Write a function, `get_geninq_features_strength(text, dictionary)`, that given a text and the dictionary produced from the previous function, should produce a tuple of four elements, the first two element being the number of words of strong subject type, with positive and negative polarity, the other two elements being the same counts, but for weak subject type words.

```
>>> geninq_dict = get_geninq_lexicon(path)
>>> get_geninq_features(text, geninq_dict)
(4, 0, 2, 1)
```

This function results in a tuple for each text entry to the function, representing their mapping into the General Inquirer sentiement dimensions.

3 Problem: Named Entites (25 points)

Name Entity Recognition (NER) is a way which locates atomic elements (words or phrases) into predefined categories such as *Person Name*, *Companies*, *Locations*, *Date*, *Money*, etc... As you can imagine, the occurrence of name entity type in corpus might be an important indication about text status.

Thus we want to use the presence of various name entities as another feature. For this problem and the problem afterwards, we will be relying on the Stanford CoreNLP. This toolkit can perform a number of NLP tasks such as Part Of Speech tagging, Parsing, Lemmatization, Named Entity recognition, etc. More information can be found at `http://nlp.stanford.edu/software/corenlp.shtml`

We aim to extract Named Entity information and Part of Speech information for all the news entries that we have, and use them as features in the classification task. To obtain this data from CoreNLP, we need to first format the data that we have in a way that it can be sent to the toolkit as input.

3.1 Cleaning and getting the data data

Refer the introduction to understand the layout of the CSV file, and write a the function to obtain all the news entries for the CSV file. Write these entries into a text file, with one news entry per line, and all the news entries seperated by newline characters. Make sure that the entires do not have any new line characters contained in them. This is the format that coreNLP package will use as input.

3.2 Running CoreNLP (10 points)

Details regarding how to run the CoreNLP toolkit can be found at: `http://nlp.stanford.edu/software/corenlp.shtml#Usage`. But basically, an parameter file has to be created with the necessary parameters, telling the toolkit what annotations are needed for the input text that is to be given.

First create a .properties file containing the following annotations:

```
annotators = tokenize, ssplit, pos, lemma, ner, parse
```

Then run a command like so:

```
java -cp stanford-corenlp-YYYY-MM-DD.jar:stanford-corenlp-YYYY-MM-DD-models.jar:xom.jar:joda-time.jar
-Xmx3g edu.stanford.nlp.pipeline.StanfordCoreNLP
 [ -props <YOUR CONFIGURATION FILE> ] -file <YOUR INPUT FILE>
```

This will create an output xml file which contains all the necessary annotations. A sample output file can be found here: `http://nlp.stanford.edu/software/corenlp_output.html`

Write a function, `run_corenlp(input_file, output_file)`, that takes in a input file name and an ouput file name, and runs the command by adjusting the above template accordingly.

3.3 Collecting the company Entities (5 points)

Write a function called `extract_company_entities(data_dir, xml_file)`, which takes as input the path to the data directory containing the news XML file and returns the number of companies-entities that are listed in the text of the file.

3.4 Collecting the people Entities (5 points)

Write a function called `extract_people_entities(data_dir, xml_file)`, which takes as input the path to the data directory containing the news XML file and returns the number of people-entities that are listed in the text of the file.

3.5 Collecting the location Entities (5 points)

Write a function called `extract_location_entities(data_dir, xml_file)`, which takes as input the path to the data directory containing the news XML file and returns the number of people-entities that are listed in the text of the file.

3.6 Extra Credit: Entity class of your choice (5 points extra credit)

Write a function called `extract_custom_entities(data_dir, xml_file)`, which takes as input the path to the data directory containing the news XML file and returns the number of entities of your choice that are listed in the text of the file. The entities that are counted here can be any collection of entities of your choice, but you are required to justify this choice of entities. Write your justification as a comment in your code for this function.

Make sure that you run the coreNLP toolkit on BigLab, as it consumes too much memory to be run successfully on eniac. And once again, START EARLY! This may take time!

4 Problem: Part of Speech (25 Points)

Another set of features we are going to use to help in the classification task are Part of Speech tags, specifically the adjectives used in the news entries.

4.1 Extracting Adjectives (10 points)

Write a function `extract_adjectives(xml_file)`, which takes output of the Stanford toolkit, and for each news entry extracts a list of all the adjectives that appear in the all the text entires, that have more than 5 occurrences.

```
>>> adj_list = extract_adjectives(xml_file)
>>> adj_list
['good', ...]
```

4.2 Mapping Adjectives (15 points)

Write a function `map_adjectives(news_entry, adj_list)`, which takes in as input a news entry, and the list of adjectives from the previous section, and returns a list of the same dimension, with a 1 in the locations where the corresponding adjective is present in the news entry, and a 0 if not present.

5 Problem: SVM Classification (25 points)

5.1 Training Data

Using the features that we have extracted from the previous steps, we will now see if we can identify how well companies perform. For this task we will be using the LIBSVM toolkit, which uses Support Vector Machines for classification and regression, amongst other things More details about the package can be found at:

```
http://www.csie.ntu.edu.tw/~cjlin/libsvm/
```

To use the model, we must first format our data into a file with the format necessary for LIBSVM to use it. This format is:

```
[Label] [Index1]:[Feature1] [Index2]:[Feature2] ...
[Label] [Index1]:[Feature1] [Index2]:[Feature2] ...
```

Essentially, each line represents a single instance of the training set. All lines are separated by newline characters ("\n"), and class of each instance is represented as a label, followed by feature index and it's corresponding feature value as shown. In case you may forgot, we're using 1 for the rasing of stock price, -1 for the dropping of stock price.

5.2 Getting training data (10 points)

Write another function, `process_corpus_(data)` that uses the data that is obtained from the previous step, and the three features extractors, using the features extracting functions we have written in sections 1-4. We want to create the features vector associated with each news entry using the following features: Only Lexical features, only Sentiment word features, only Named Entity Features, only Part of Speech features, and all these features combined.

For each of the features under consideration, after this file has been created, we should scale the values so that they all lie in the same range, between 0 and 1. We can do this using the svm-scale program on the command line.

```
svm-scale -s scaling_parameters train_data > scaled_train_data
```

You can now used the scaled training data to train a classification model, using the svm-train command. Running this command on any train data file titled 'train-data' will result in a model file called 'train-data.model'. This model can then be used to predict what the stock price should be for the instances in the test set; whether they exceed expectations or are below expectations.

5.3 Running SVM and making the model.

The model file can be created using the training data and the svm-train program as so:

```
svm-train [options] training_set_file [model_file]
```

Where you can find out more about the options on the linked website. Since we are performing classification, it is sufficient to use just the default options that svm-train works with and not specify any of your own.

5.4 Testing on held out news entries, and final results (15 points)

Now that you have obtained the model using svm-train, we can use the model file obtained to test on held out test data. Use the functions written previously to create a feature file for the test data similar to the training file, and obtain predictions for the test news entries. Train and test a classifier based on lexical features you extracted in problem 1.2 sentiment features you extracted in problem 2.2 and 2.4, named entities obtained in section 3, adjective features you extracted in 4.2 , and all the combined using a representation consisting of the concatenation of the above. Then, use the same functions to create the same kind of features for the testing data, and produce the predictions for the entries in this file, and compare them to the actual outcomes.

Using these predictions, we want to calculate Precision, Recall, and F-Score for the test set. We want to calculate these scores for each of the 4 features individually, as well as a combination of all four features. Write a file called "scores.txt" with the three values for each feature on each line separated by tabs, and lines separated by newline characters, in the order of:

```
lexical
sentiment
adjectives
named entities
combined
```