CIS 430/530 Fall 2012 HW 3

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

Released: November 3, 2012
Amended: November 13, 2012
Due: 11:59PM November 19, 2012

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 CoreNLP. The textual data, as in the previous homeworks, consists of CIQ Key Development news summaries about various companies. There are many simplifications, but the general goal is to predict what features from the news may predict company performance.

Here we'd like to apologize for any confusions, bugs or ambiguities in first version of the homework. We are using new data and have changed almost all questions in this year's homework. We also appreciate all the discussions and questions posted on Piazza. You guys have been really helpful and amazing.

The main changings happened in Problem 3 and Problem 5. For problem 3, we understand that you're finding it difficult to manipulate a tremendously large-size xml file to handle the features. Thus we provide clean data on each news event, and you can handle them one by one. For problem 5, we'll estimate on full dataset as well as a small set where noise has been reduced (See Section 5.4.2 for Details). For those we've already ran problem 3, you don't need to re-run the experiment, as the features have already been extracted.

As for the other problems, the changes are mostly in descriptions. We've also provided examples to make the input/output clear. The credits given for each problems have also been changed a little bit.

Deliverables

You will be asked to submit the code for the functions you have implemented. Also write ample comments in your code.

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_adv.txt
scores.txt 21733.xml scaled_named_entities.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 following things: (1) **Changing of stock price 2 days after announcement date.** (2) **Trend of stock price in extreme conditions, as specified in Problem 5**. To simplify 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 four columns from the original CSV file: the column **situation** which includes the news at that day, and the columns **xret0**, **xret2**, **xret5** which includes the stock price's change within same day, 2 days after the news entry and 5 days after the news entry announcement. For the return value, 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 use the processed text that we have provided to extract your features. Note that for some instances, the actual price changes may not be available. You should ignore these instances while training.

We have also included the cleaned text version of training data. These should be helpful, especially to resolve the problem of mapping sentences in Problem 3.

The data you will be using is located at `/home1/c/cis530/hw3/`. This folder contains two files, `training.csv` and `testing.csv`; as well as two folders: `data/training` and `data/testing` which includes the text version for each news entry. 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.

1 Problem: Lexical features (10 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 (5 points)

Write a function `extract_top_words(csv_file)` or `extract_top_words(directory)` that takes in the path for the CSV file, or the path to the data directory containing the all the cleaned files, and returns a list that contains the words from all the news entries **in training**, that have more than 5 occurrences (> 5) in the entire corpus. Lower case all words.

```
>>> extract_top_words(csv_file) (or extract_top_words(directory))
['new', 'york', 'starbucks' ... ]
```

1.2 Creating lexical features per news entry (5 points)

Write a function, `map_entry(string, top_words)` that takes in a news(situation) as a string, and the list generated from the previous function, and returns a list of the same size as top_words, 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.

```
>>> map_entry(string, top_words)
[1, 0, 3, .... ]
```

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 Inquirer 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.

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 list of tuples. This is aimed at handling the condition which one word has different meanings. For each tuple, the first element of the tuple being the word type for the word, and the second element being the sentiment of the word. If one word has multiple polarities and emotions, we should return all of them.

The dictionary should look something like:

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

The word **mean** is both *strong negative* and *weak neutral*, thus the output is like:

```
>>> mpqa['mean']
[('strongsubj', 'negative'), ('weaksubj', 'neutral')]
```

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. If one word has multiple polarities, we should +1 to each specific meaning. The word **mean** is both negative and neutral, and with example below:

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

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. We also take the word **mean** as example here:

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

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

2.3 Grabbing the General Inquirer lexicon (4 points)

The General Inquirer tagset is a lexicon of words that have been tagged with their perceived subjective polarity, 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`.

Write a function, `get_geninq_lexicon(lexicon_path)` that should access the corpus file above, and returns a dictionary which the value is **a list of four numbers**: (1) Positive polarity or not? (GI called it **Pstv**), (2) Negative (**Ngtv**) polarity or not?, (3) Strong (**Strng**) polarity or not? (4) Weak (**Weak**) polarity or not? This setting is because one word exactly corresponded to different tags, which is different from **MPQA**, thus it's possible for the return value to be $[0, 0, 0, 0]$ or $[1, 1, 1, 1]$. Example below:

```
>>> gi = get_geninq_lexicon(lexicon_path)
>>> gi["make"] = (1, 1, 1, 0)
>>> gi["malady"] = (0, 1, 0, 1)
```

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. Ways of dealing with word with multiple meanings is the same as 2.2.

```
>>> 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, producing 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 weak positive and weak negative.

```
>>> 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 sentiment dimensions.

3 Problem: Named Entities (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 for 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 at extracting 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.

3.1 The data

There are two options for you to find and use the data. The first is still using the large **.csv** file to generate the **.xml** you want, as described in previous version of homework. Or you can work with raw text file for this homework. The raw texts are located at `hw3/data`, in directories called `training` and `testing`. In each of these directories, each news entry is represented as a single file in `all_files` directory. In the directories, you should find the following files: a `price_mapping.out` file that maps a single news entry ID to the share price changes, a `text_mapping.out` file that maps a single news entry ID to the text of the news entry, and a directory called `all_files` which contains a single file for each news entry. The file name for these files is the same as the ID assigned to the news entry.

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`. Basically, we need to specify a list of annotators, the list of input files, and the output directory.

We want to specify a list of files on which to run and extract Part of Speech and Named Entity information. So, we can first create a file that lists all the files on which to run like so:

```
ls all_files > fileList
```

Be aware that the above command will just enter the filenames into fileList, and the path may have to be prepended. Then, we Run a command like so to create the xml files with the necessary information:

```
java -cp stanford-corenlp-2012-07-09.jar:stanford-corenlp-2012-07-06-models.jar:xom.jar:joda-time.jar
-Xmx3g edu.stanford.nlp.pipeline.StanfordCoreNLP
-annotators tokenize,ssplit,pos,lemma,ner -filelist fileList
-outputDirectory <OUTPUT DIRECTORY PATH>
```

This command feeds CoreNLP with a list of training files, and outputs the xml files into the specfied directory. You will have to run this twice, once for training data, once for testing data (only the file list paths and the output directory are different). A sample output file can be found here: `http://nlp.stanford.edu/software/corenlp_output.html`

For this section of the homework, please hand in the xml file of 21733 of the training set. (If you've already ran on the large .xml as specified in previous version of homework, you don't need to run all the things again. However, you need to generate the testing data for file 21733, specifically.)

3.3 Collecting the Entities (15 points)

We will now extract entities for each file. Write a function called `extract_named_entities(xml_files_path)`, which takes as input the path to the data directory containing the news XML files and returns a list, each element of the list will also be a list like the following: $[A_i, B_i, C_i]$. (Or you can run on one big xml file as specified in previous homework, which function looks like `extract_named_entities(xml_file)`, return value being the same).

It means for article i, A_i is the number of companies mentioned in this article, B_i is the number of people names in this article, and C_i is the number of locations mentioned in this article.

Note that since the file name sorted from small to large corresponds to each entry in the csv file, construct the array so that the i-th entry corresponds to the i-th entry in the csv file as well. Ascending order means: 2 is smaller than 10, according to the number they represent.

```
>>>extract_named_entities(xml_files_path) or extract_named_entities(xml_file)
[[10, 5, 3] ...]
```

Hint: Look for "Organizations", "Person", "Location".

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

Write a function called `extract_custom_entities(text_path, 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 (15 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 (8 points)

Since the ambiguity of previous homework description, you could have two ways of handling this problem.

Case 1: Reading the full xml_file. In this case, you need to write a function `extract_adjectives(xml_file)`, which takes output of the Stanford toolkit, and return a list of all the adjectives that appear in all the the text entries which have occurrence more than 5 times.

Case 2: You can also take the directory of clean files as input. In this case, you need to run CoreNLP or Stanford-NER separately for each file (like in question 3, CoreNLP can batch-process). In fact if you have batch-processed the files in question 3, then the XML files generated should already have Part of Speech information. However, the return value is the same: a list of all the adjectives that appear in all the the text entries which have occurrence more than 5 times.

Notes: Please do this on **Training** data only.

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

4.2 Mapping Adjectives (7 points)

Write a function `map_adjectives(string, adj_list)`, which takes the input piece of news(situation) as a string, and the list of adjectives from the previous section, and returns a list of the same dimension, with **1** in the locations where the corresponding adjective is present in the news entry, and a 0 if the adjective is not present. You can also have your first parameter as the filename if you feel it's more convenient.

```
>>> map_adjectives(string, adj_list) or map_adjectives(filename, adj_list)
[1, 0, 1, .... ]
```

5 Problem: SVM Classification (30 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 change 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_dir, features)` which uses the data provided, and the functions you have written in sections 1-4. We want to create the feature vector associated with each news entry using the following feature settings: Only Lexical features, only Sentiment word features, only Named Entity Features, only Part of Speech features, and all these features combined. The `features` parameter should be an integer, and its value should specify which features should be part of the vector, and which shouldn't. The values should be as follows:

```
1 -> Only Lexical
2 -> Only Sentiment Words
3 -> Only Named entity features.
4 -> Only Part of Speech features.
5 -> All 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 or below expectations after k days.

We will check this problem according to your submission of the **scaled training file using just the Named Entities** (**scaled_named_entities.txt**) as features.

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]
```

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 (20 points)

Now that you have obtained the model using Svm-train, we can use the model file generated 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 from Problem 1, Sentiment features from Problem 2, Named Entities from Problem 3, Adjectives from Problem 4, 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.

5.4.1 Full Set Performance (10 Points)

Using these predictions, we want to calculate Precision, Recall, and F-Score for the test set on **Both** *Positive* and *Negative* samples. 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 six values (Pos-P, Pos-R, Pos-F, Neg-P, Neg-R, Neg-F) on each line separated by tabs, and lines separated by new-line characters, in the

order of *lexical, sentiment, adjectives, named entities, combined.*

The result file should look like this:

```
0.6 0.4 0.48 0.571 0.75 0.648  #####: (separated by \t) - Lexical
      ... ...                  #####: - Sentiment
      ... ...                  #####: - Adjectives from Pos-tagging
      ... ...                  #####: - NER
      ... ...                  #####: - Combined
```

5.4.2 Extreme Set Performance (10 Points)

You may already find that the accuracy of features running on full set is not adorable. This is because just taking the returning value on **one day** is not a very good idea. Further more, stock price may change due to various reasons, while the announcement/news event is only one of them. One way to overcome this problem is to extract more powerful features, or doing feature selection based on some statistical approaches. For us, we decide to play with the reduced Dataset, say, only considering the news events at **extreme conditions**. In this way, we give a definition of **extreme conditions** and split our data into three groups with different labels, as following shows:

$$\begin{cases} -1, & ret0 < -2,\ ret2 < -2,\ ret5 < -2; \\ 0, & \text{Others}; \\ +1, & ret0 > +2,\ ret2 > +2,\ ret5 > +2 \end{cases} \tag{1}$$

For the news event with Label 0, we just throw them away from both training and testing data, only preserving the events with Label $+1$ or with Label -1, and no more than 10% should have left. (And the new label to train a model and predict are also decided not just on $xret2$, but on all of the three: $xret0$, $xret2$ and $xret5$.) Again reporting the performance running on the five feature sets in the format described above, and submit a file called `scores_adv.txt`.