CS 334: Homework #4

Submission Instructions: Your submission should consist of two steps (detailed in Homeworks #1, #2). If *either of the two steps are late, then your assignment is late.*

Dataset Description: For this homework, you will be building models to perform spam detection. The new dataset `spamAssassin.data` is a subset of the SpamAssassin Public Corpus (see `https://spamassassin.apache.org/old/publiccorpus/`). It contains emails that were tagged as either spam or not spam. Each line contains an email, with the label in the front (1 denoting a spam, while 0 denoting not spam).

Here is a sample email that contains a URL, an email address (at the end), numbers and dollar amounts.

```
> Anyone knows how much it costs to host a web portal ?
> Well, it depends on how many visitors youre expecting. This can be anywhere
from less than 10 bucks a month to a couple of $100. You should checkout
http://www.rackspace.com/ or perhaps Amazon EC2 if youre running something big...

To unsubscribe yourself from this mailing list,
send an email to: groupnameunsubscribe@egroups.com
```

We have already implemented the following email preprocessing steps: lower-casing; removal of HTML tags; normalization of URLs, e-mail addresses, and numbers. In addition, words have been reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Finally, we removed all non-words and punctuation. The result of these preprocessing steps on the same email is shown below:

```
anyon know how much it cost to host a web portal well it depend on how mani visitor
your expect thi can be anywher from less than number buck a month to a coupl of
dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run someth
big to unsubscrib yourself from thi mail list send an email to emailaddr
```

1. **Preprocessing + Model Assessment** (7+7+7+9 =30 points): For this homework, you will extract features from each e-mail and construct several datasets. You will also implement the training and test data split for model assessment in your experiments. The template code for this problem is found in `q1.py`. Note that you will need to modify and fill out the `main` function so that you can take in the raw dataset and store the different extracted datasets. In other words, if we run `q1.py`, we should be able to get the datasets you use for problem 2 and 3.

   (a) (Code) Implement train-test split as your model assessment strategy in the function `model_assessment`. It should return a training set and test set with type `pandas.DataFrame`. The training set and test set should each contain two columns `y` and `text`, where `y` contains the binary label, and `text` is the text in each email as a list of strings. Make sure to randomly shuffle the dataset before splitting, which means each call to your function should not generate identical outputs. Report how you determined the sizes of the training and test set. Your function should return a Python tuple (i.e., multiple values). For example, a function with `return item1, item2` would return a tuple with item1 first and item2 second.

(b) (Code) Build a vocabulary map (or dictionary) that corresponds to the number of emails that each word appears in the function `build_vocab_map` where the input is a pandas dataframe consistent with the previous part (i.e., 2 columns with `y` and `text`). Identify all the words that appear in *at least 30 e-mails* in the input pandas dataframe. Your function should return a tuple (i.e., 2 objects) where the first object is a dictionary with the word as the key and the number of times it appears in the corpus as the value and the second object is a list of words that occurs over 30 times.

(c) (Code) Construction of Binary Dataset: For each e-mail, transform it into a feature vector where the ith entry, x_i, is 1 if the ith word in the vocabulary occurs in the email, or 0 otherwise. You should only use the words that you identified in (b) for improving scalability. This should be done in the function `construct_binary`. You need to construct both a binary training set and a binary test set, which should be done by invoking the `construct_binary` function twice in the main method. Note that the output vector should be a 2-d `numpy.array`.

(d) (Code) Construction of Count Dataset: For each e-mail, transform it into a feature vector where the ith entry, x_i, represents the number of times the ith word appears in the email. You should only use the words that you identified in (b) for improving scalability. This should be done in the function `construct_count`. You need to construct both a count training set and a count test set, which should be done by invoking the `construct_count` function twice in the main method. Note that the output vector should be a 2-d `numpy.array`.

2. **(3+3+5+5+6+3+6+8+6 = 45 pts) Spam Detection via Perceptron** For this problem, you will implement the Perceptron algorithm and apply it to the problem of e-mail spam classification. You *ARE NOT* allowed to use any existing toolbox / implementation. You'll be comparing the two different feature representations as well as the number of epochs through the data.

(a) (Code) Implement the perceptron algorithm in `perceptron.py`. Note that you must update the `w` variable in your class with the appropriate weights. `w` must be a numpy 1-d array (i.e., np.array([1,2,3,4]) is an example of a 1d array, failure to follow this convention may result in failed autograder test cases. For the train/predict, you should assume any dummy padding to set a non-zero threshold will be done prior to calling train and predict. You will implement the following functions:

i. `calc_mistakes`: Helper function that calculates the number of mistakes.

ii. `transform_y`: Given an input dataframe from above, transform y to -1 and 1.

iii. `predict`: Predict the class label based on the input data. For the boundary case of $\mathbf{w} \cdot \mathbf{x} = 0$, predict the +1 class.

iv. `sample_update`: Given a single sample, x and y, calculate the updated weights based on the sample. The function should return a tuple where the first value is the updated weight (w) and the second value is whether or not there was a mistake for this sample prior to the weight update (0 for no mistake, 1 for a mistake).

v. `train`: Train the model that uses the training examples provided to the function that returns a dictionary where the key is the pass (epoch) through the data and the value is the number of updates (mistakes). After each epoch, you should *shuffle* your data (this will improve generalizability). Note that the epochs should start

from 1 (1 pass through the data) and exit if there are no mistakes, or the maximum number of epochs is reached, whichever comes first.

vi. `main`: Update the function so that you can take in one of your datasets (e.g., binary) from problem 1, pad it with a dummy feature to get a non-zero feature, and report the number of mistakes on the test dataset.

(b) (Code) Implement `tune_perceptron` which will use k-fold cross validation (you can choose k) as the model selection strategy and find the optimal number of epochs for an input dataset (e.g., the binary or the count created in question 1). Your code should return the optimal epoch numbers.

(c) (Written) For each of the two datasets created in question 1 (binary and count), use `tune_perceptron` to find the optimal number of epochs. Then train a perceptron model using the optimal epoch number, and report the number of mistakes on training set and test set respectively for each of the two datasets (binary and count).

(d) (Written) Using the vocabulary list together with the parameters learned in question (c), output the 15 words with the most positive weights, and the 15 words with the most negative weights for the binary and then count dataset.

3. **(15+10=25 pts) Spam Detection using Naive Bayes and Logistic Regression**

For this problem, you will compare the spam detection against Naive Bayes and Logistic Regression using `sklearn` modules. You should write your own Python script to do this.

(a) (Written) Train the appropriate Naive Bayes algorithm against each of the two datasets and assess the performance. (Hint: You may consider MultinomialNB or BernoulliNB for the two datasets.) How does your Naive Bayes perform in terms of the number of mistakes?

(b) (Written) Train a logistic regression model against each of the two constructed datasets. How does logistic regression perform in terms of the number of mistakes?