CS 334: Homework #4

Submission Instructions: The homework is due on Nov 4th at 11:59 PM ET. Make sure your program uses Python 3.7. 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. **Feature Extraction + Model Selection** (5+5+6+7+9=32 points): For this homework, you will extract features from each e-mail and classify it as spam. You will also be designing the methodology for 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 of the program so that you can take in the dataset and store the different datasets that reflect both your model assessment strategy and your preprocessing. In other words, if we run `q1.py`, we should be able to get the datasets you use for the remainder of the problem.

   (a) Define your model assessment strategy that you will use for this dataset. You must justify your methodology. Implement your methodology in the function `model_assessment`.

   (b) Build a vocabulary map (or dictionary) that corresponds to the number of emails that each word appears in. Identify all the words that appear in *at least 30 e-mails* in your training dataset. This should be done in the function `build_vocab_map`.

   (c) 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`.

(d) 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`.

(e) Construction of TF-IDF Dataset: For each email, you will construct the term frequency-inverse document frequency (TF-IDF) to capture how important a word is to the corpus. The term frequency of a word is the number of times the word appears in the document. The inverse document frequency refers to thee logarithmically scaled inverse fraction of the documents that contains the word. You can use the `TfidfTransformer` or `TfidfVectorizer` in `sklearn.feature_extraction.text` to calculate the TF-IDF representation using your features in (c). This should be done in the function `construct_tfidf`.

2. **(30+9+5 = 44 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 three different feature representations as well as the number of epochs through the data.

   (a) Implement the perceptron algorithm in `perceptron.py` This means you will implement the following four functions:

   - `predict`: Predict the prediction based on the input data. For the corner case of $\mathbf{w} \cdot \mathbf{x} = 0$, predict the $+1$ class.
   - `calc_mistakes`: Calculate the number of mistakes.
   - `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). Your algorithm should exit if there are no mistakes, or the maximum number of epochs, whichever comes first.
   - `main`: Update the function so that you can take in one of your datasets (e.g., binary) from problem 1 and report the number of mistakes on the test dataset.

   (b) For each of the three datasets above, train the perceptron using your training set. How many mistakes are made before the algorithm terminates? What is your estimated predictive error? What should be the optimal number of epochs?

   (c) Using the vocabulary list together with the parameters learned in the previous question, output the 15 words with the most positive weights with your most promising perceptron. What are they? Which 15 words have the most negative weights?

3. **(15+9=24 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) Train the appropriate Naive Bayes algorithm against each of the 3 datasets and assess the performance. You may discover that you need to use a different class for at least two of the datasets. How does your Naive Bayes perform in terms of the number of mistakes?

   (b) Train a logistic regression model against each of the 3 constructed datasets. How does logistic regression perform in terms of the number of mistakes it makes?