EECS 349 Machine Learning Homework 5

How to submit your homework

1. Create a **PDF document** containing answers to the homework questions. Show your reasoning in your answers.
2. Include source code for the program you write.
3. Compress all of the files specified into a .zip file.
4. Name the file in the following manner, *firstname_lastname_hw5.zip*.
3. Submit this .zip file via Canvas by the date specified on Canvas.

THE PROBLEM

Spam is e-mail that is both unsolicited by the recipient and sent in substantively identical form to many recipients. Spam is dealt with on the receiving end by automated spam filter programs that attempt to classify each message as either "spam" (undesired mass email) or "ham" (mail the user would like to receive). A Naïve Bayes Spam Filter uses a Naive Bayes classifier to identify spam email. Many modern mail programs implement Bayesian spam filtering. Examples include SpamAssassin and SpamBayes. In this lab, you will create a spam filter based on Naïve Bayes classification.

THE MATH OF A NAÏVE BAYES CLASSIFIER

A Naïve Bayesian classifier takes objects described by a feature vector and classifies each object based on the features. Let V be the set of possible classifications. In this case, that will be "spam" or "ham."

$$V = \{spam, ham\} \tag{1}$$

Let a_1 through a_n be Boolean values. We will represent each mail document by these n Boolean values. Here, a_i will be "true" if the ith word in our dictionary is present in the document, and a_i will otherwise be false.

$$a_i \in \text{Dictionary} \tag{2}$$

If we have a dictionary consisting of the following set of words:
{"feature", "ham", "impossible","zebra"}, then the first paragraph of this section would be characterized by a vector of four Boolean values (one for each word).

$paragraph1 : a_1 = T \wedge a_2 = T \wedge a_3 = F \wedge a_4 = F$

We write the probability of paragraph1 as: $P(a_1 = T \wedge a_2 = T \wedge a_3 = F \wedge a_4 = F)$

When we don't know the values for the attrubutes yet, we write...

$P(a_1 \wedge a_2 \wedge a_3 \wedge a_4)$ ...but we don't mean the probability these variables are TRUE, we mean the probability these variables take the observed values in the text we're encoding.

A Bayesian classifier would find the classification for the spam that produces the maximal probability for the following equation. This is the Maximum A Posteriori (MAP) classification.

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j \mid a_1 \wedge a_2 \ldots \wedge a_n) \tag{3}$$

Now…to estimate values for our probabilities, it is going to be more convenient to flip these probabilities around, so that we can talk about the probability of each attribute, given that something is spam. We can do this by applying Bayes rule.

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1 \wedge a_2 \ldots \wedge a_n \mid v_j) P(v_j)}{P(a_1 \wedge a_2 \ldots \wedge a_n)} \tag{4}$$

Since we are comparing a single text over the set of labels V in this equation, we can remove the divisor (which represents the text we've encoded), since it will be the same, no matter what label v_j is set to.

$$v_{MAP} = \arg\max_{v_j \in V} P(a_1 \wedge a_2 \ldots \wedge a_n \mid v_j) P(v_j) \tag{5}$$

A Naïve Bayes classifier is "naïve" because it assumes the likelihood of each feature being present is completely independent of whether any other feature is present.

$$\begin{aligned} v_{NB} &= \arg\max_{v_j \in V} P(a_1 \mid v_j) P(a_2 \mid v_j) \ldots\ldots P(a_n \mid v_j) P(v_j) \\ &= \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i \mid v_j) \end{aligned} \tag{6}$$

Now, given this mathematical framework, building a spam classifier is fairly straightforward. Each attribute a_i is a word drawn from a dictionary. To classify a document, generate a feature vector for that document that specifies which words from the dictionary are present and which are absent. *Note about this encoding: repeating a word 100 times in a document won't cause the answer to "Is word X present in document Y ?" to be any more true than if the word is there 1 time. This is true BOTH when you build your probabilities in the dictionary AND when you encode an email to see if it spam.*

Then, for each attribute (such as the word "halo"), look up the probability of that attribute being present (or absent), given a particular categorization ("spam" or "ham") and multiply them together. Don't forget, of course, to determine the base probability of a particular category (how likely is it for any given document to be spam?). Voila, the categorization that produces the highest probability is the category selected for the document.

Oh…one other thing. Your dictionary will have thousands of words in it. If you multiply thousands of words together, what happens to the probabilities? Can you represent them? You might consider using equation (7), instead of (6), when implementing your code.

$$v_{NB} = \arg\max_{v_j \in V} \left(\log(P(v_j)) + \sum_i \log\left(P(a_i \mid v_j)\right) \right) \tag{7}$$

A TRAINING CORPUS

To train your system, we suggest you use the spam corpus used by the people who built the Spam Assassin program. To get this spam corpus, go to the following link: http://spamassassin.apache.org/publiccorpus/

You should pick a spam file and a ham file for you to test and train your system. Two that seem to work fairly well are *20030228_spam.tar.bz2, 20021010_easy_ham.tar.bz2*, but you don't HAVE to use those files in your experiments. You can unzip these files using the standard unix utility tar (with the options xjf) or from windows using WinRAR (which can be found here: http://www.rarlab.com/download.htm).

EECS 349 Machine Learning Homework 5

STARTER CODE

We have provided you a starter framework for your work. In a file called *spamfilter.py*, is a a `def makedictionary`, a `def spamsort` and a `main` that you can call from the command line. These functions are the skeleton for what you need to write. They don't do the full job, though. The first function is the one to make a dictionary.

```
def makedictionary( spam_directory, ham_directory,
dictionary_filename):
```

The parameter `spam_directory` is a character string containing the path to a directory that should contain only ASCII text files that are spam. The parameter `ham_directory` is a character string containing the path to a directory containing only ham ASCII text files.

The start code will read in each ASCII file, separate it into blank-delineated strings and add each unique string (a word) to a python dictionary. Associated with each word in this dictionary are two probabilities:

- The probability of observing that word **at least once** in a spam document.
- The probability of observing the word **at least once** in a ham document.

`makedictionary` returns to the calling function two things: the dictionary (`words)` and the prior probability of spam, based on the training data (`spam_prior_probability).` It also saves the dictionary to an ASCII file named `dictionary_filename`, placed in the current directory. NOTE THAT THE STARTER CODE DOESN'T ACTUALLY CALCULATE THE PROBABILITIES AND JUST INITIALIZES THEM ALL TO 0.

`spamsort` is a Python function that classifies each document in a directory containing ASCII text documents as either spam (undesired junk mail) or ham (desired document).

```
def spamsort( mail_directory, spam_directory, ham_directory,
dictionary, spam_prior_probability):
```

The parameter `mail_directory` specifies the path to a directory where every file is an ASCII text file to be classified as spam or not. The parameters `spam_directory,ham_directory` specify the destination directories for the mail. The parameter `dictionary` specifies the dictionary constructed by makedictionary. The parameter `spam_prior_probability` specifies the base probability that any given document is spam.

The function classifies each document in the mail directory as either spam or ham. Then, it copies the file to `spam_directory` if it is spam. It copies the file to the `ham_directory` if it is ham (not spam). When done, the number of mails in the `mail_directory` should be exactly the same as the sum of the number of mails in `spam_directory` and `ham_directory`, since all the files will have been copied to either `spam_directory` or `ham_directory.`

`NOTE: THE STARTER CODE CLASSIFIES THINGS BY ASSIGNING THE MOST PROBABLE CLASS, ACCORDING TO PRIOR PROBABILITY. YOUR WILL NEED TO MAKE SPAMSORT INTO A NAÏVE BAYES CLASSIFIER.`

PROBLEMS TO SOLVE

1) (2 points) The starter code, as given to you, assigns a probability of 0 to every word. You need to replace the line that assigns a 0 to every probability with a reasonable estimate using the approach described in "The Math of a Naïve Bayes Classifier". For each word in the dictionary, your system should determine the probability of observing that word in a spam document and the probability of observing that word in a ham document. *Note: the probability of observing a word in a document of a particular class is calculated by counting how many documents in that class contain one or more instances of that word. If I have 100 documents, where 99 of them have no instances of "smurf" and 1 document has 100 instances of "smurf," the probability of smurf is 0.01, NOT 1.0.*

EECS 349 Machine Learning Homework 5

2) (3 points) The starter code simply uses prior probability to classify files as spam or ham. You need to replace the code that always decides to use the most likely class with code that uses the probabilities in the dictionary to implement a Naïve Bayes classifier as described in "The Math of a Naïve Bayes Classifier".

3) (2 points) Would equation (6) and (7) always return the same results, if there were no issues with underflow or precision? Give a proof to support your answer. What happens if there is underflow? Do they still give the same result? Why or why not?

4) (3 points) Design and run an experiment to compare your spam filter to one using prior probability.

A) **Datasets:** In your experiment, what data set did you use to build your dictionary? What data set did you use to test the resulting spam filter? Were they the same? Why did you choose the data sets you chose? How many files are in your data? What is the percentage of spam in the data? Is your data representative of the kinds of spam you might see in the real world? Why or why not?

B) **Your Testing Methodology:** Explain how you tested the spam filter to see if it worked better than just using the prior probability of spam learned from your training set. Did you do cross validation? If so, how many folds? If not, why not? What is your error measure?

C) **Experimental Results:** Does your spam filter do better than just using the prior probability of spam in your data set? Give evidence to back up your assertion. This may include well-labeled graphs, tables and statistical tests. The idea is to convince a reader that you really have established that your system works (or doesn't).