EECS 349 Machine Learning Homework Xtra

WHAT TO HAND IN

You are to submit the following things for this homework:

1. A **SINGLE PDF** document containing answers to the homework questions.
2. The **WELL COMMENTED** Python source code for all software you write.

HOW TO HAND IT IN

To submit your assignment:

1. Compress all of the files specified into a .zip file.
2. Name the file in the following manner, firstname_lastname_hwX.zip. For example, Bryan_Pardo_hwX.zip.
3. Submit this .zip file via Canvas.

DUE DATE: as specified in the course calendar.

How we will grade your code

Is it in the required format? Both functions described later (makedictionary and spamsport) must be in the SAME .py file called *spellchecker.py*. That is, in *spellchecker.py*, there should be a def makedictionary, and a def spamsort. This is so that when the testing script is run, it will just import your functions from the assumed existing file *spellchecker.py*, and run them to see if they work. **Solutions that don't adhere to this will not be accepted!**

Does it work at all? That is, can I run my testing script on it without a crash? Does it produce the output files where and how it is supposed to. Are they named correctly? Do files get moved to the correct directories?

Does it produce correct output? In other words, are the spams in the spam folder and the hams in the ham folder? Did your dictionary contain the right probabilities after I ran it on a training corpus?

Is it well-commented? This part matters a lot if you fail at working or producing correct output. Does each function explain what it expects as input and what it will produce as output? Are the internal steps commented?

How we will grade your answers

Are they clear and concise? Were all the questions answered? Did the ideas make sense? Has the data been presented in a way that is easy to understand, with labeled tables or figures and explanations thereof? **A screen dump showing 5 pages of output to the command line is neither clear nor concise.**

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.

EECS 349 Machine Learning Homework Xtra

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.

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

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

In order 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).

PYTHON FILE FORMAT (IMPORTANT READ THIS)

Both functions described below (makedictionary and spamsport) must be in the SAME .py file called *spellchecker.py*. That is, in *spellchecker.py*, there should be a def makedictionary, and a def spamsort. This is so that when the testing script is run, it will just import your functions from the assumed existing file *spellchecker.py*, and run them to see if they work. Solutions that don't adhere to this will not be accepted!

EECS 349 Machine Learning Homework Xtra

THE DICTIONARY BUILDER (5 points)

You will need to write a Python function that builds a dictionary of words from a set of spam and ham files, along with the associated probability of occurrence of each word.

The dictionary builder should accept the parameters shown below

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

The parameter `spam_directory` is a character string containing the path to a directory containing ONLY spam files. Each file in this directory will be an ASCII text file that is spam. The parameter `ham_directory` is a character string containing the path to a directory containing ONLY ham files. Each file in this directory will be an ASCII file containing a ham document. Once the dictionary is created, it should save the results to a file named `dictionary_filename`. This file should be IN THE CURRENT DIRECTORY. Repeat:This file should be **IN THE CURRENT DIRECTORY**.

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 is calculated by counting how many documents 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.*

The dictionary file should consist of three **space-delimited** elements per line. The first element is a word. The second element is the probability of observing that word **at least once** in a spam document. The third element is the probability of observing the word **at least once** in a ham document.

Lines in the dictionary MUST BE IN ALPHABETICAL ORDER. **Assume A-Z is equivalent to a-z (i.e. capitalization does not matter) and that the digits 1-9 precede A-Z in alphabetical order.**

Here is an example dictionary format

```
[word]      [P(word|spam]     [P(word|ham)]
```

Here is an example dictionary of four keywords.

```
Bob          0.003       0.4
break        0.001       0.02
Cl3an        0.7         0.003
Supercalafragalisticexpealidocious  0.01  0.0
```

While you must ignore capitalization, you will still have to make a lot of choices in designing your dictionary maker. Will you treat "feature" and "features" as different? What about "Bob" and "B0b"? Will you ignore the text in the headers of emails? These design choices are up to you, but you must justify them in the question-answering portion of this lab.

THE SPAM CLASSIFIER (5 points)

You will need to write 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_filename, spam_prior_probability):
```

Here, `mail_directory, spam_directory, ham_directory, dictionary_filename` are all character strings. The variable `spam_prior_probability` is a real value. 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 parameter `dictionary_filename` specifies the name and path of a dictionary file in the same format as that output from the ***`makedictionary`*** function. The parameter `spam_prior_probability` specifies the base probability that any given document is spam.

EECS 349 Machine Learning Homework Xtra

Your program must classify each document in the mail directory as either spam or ham. Then, it must move it to `spam_directory` if it is spam. It must move the file to the `ham_directory` if it is ham (not spam). When done, the `mail_directory` should be empty, since all the files will have been moved to either `spam_directory` or `ham_directory.`

QUESTION TO ANSWER (5 points total, 1 point per question)

You will need to test the performance of your system. This means creating a training dataset and a testing dataset. You will train the spam filter on the training set and then evaluate results on the testing set.

1) **Design choices:**

A. Describe how you decided to build your dictionary. How did you decide to deal with non-alphanumeric characters? How about plurals? What other issues did you build your system to handle? Is every word from the training corpus in your dictionary or did you remove some words or limit the dictionary size? Why?

B. Describe how you decided to build your spam sorter. How did you calculate the probabilities? Did you end up using equation (6) or (7)? If you used (7), give a proof it would return same results as (6), if underflow were not an issue.

2) **Datasets:** In your experiments, 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?

3) **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?

4) **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).

5) **Future Work:** How might you try to redesign the system to improve performance in the future? Would you take into account higher-level features (e.g. sentence structure, part-of-speech tagging) when categorizing as spam? Would you exclude portions of the mail as meaningless noise? Why? Be specific. Saying only "I will use part-of-speech information" is not a meaningful response. I want a description of what improvements you would make, why you think they would help and how you would implement that.