CS112: Modeling Uncertainty in Information Systems
Homework 4

Due Wednesday, May 30, 2pm (submitted via courseweb)

For this assignment, you will implement a naive Bayes classifier to distinguish between two kinds of strings. The program will take as input 2 files of training strings (one for each of the two types of strings) and 2 files of test strings. The file that a test string is in indicates the TRUE class for that test string. The true classes will only be used to evaluate the accuracy of your classifier.

You will be writing the naive Bayes classifier itself. This should take in training strings and their class labels, and then, when given a test string, output the posterior probability for each class and the MAP class for that string. Your program will also report the classification accuracy over all the test strings.

The assignment consists of four parts, each of which is described below. You will document your results from parts 2, 3, and 4 in a brief write-up that will be turned in along with your code.

The C++ files and data files needed for this assignment are available on Piazza attached to the note titled "Code for Homework 4." The files are in an archive called `naive_bayes.tgz`. To uncompress this in linux, you can use the command "`tar xvfz naive_bayes.tgz`". There is a Makefile included; to compile the code, simply type "make" from the command line. There is only one .cpp file so compilation is trivial. From the command line, it would be "`g++ text_classifier.cpp`".

Part I: Implement the Classifier

Complete the provided code to create a classifier that uses 26 features: the presence of each letter (A to Z, ignoring the distinction between upper and lower case) in a string. You must implement the classes `Feature` and `NaiveBayesClassifier` in the files `feature.hpp` and `naive_bayes_classifier.hpp`. You can then run the classifier using `text_classifier.cpp` (already implemented). To implement `naive_bayes_classifier.hpp`, you need to fill in the following:

- `NaiveBayesClassifier()` - This is the constructor. You should set up the private member variable `vector<Feature> m_features` here.
- `addTrainingExample(int featurePresence, int classNumber)` - Adds a training example to each of the features within the NaiveBayesClassifier using `Feature.addTrainingExample(int,int)`. This should also modify the counts for computing the prior probability.
- `getPriorProbability(int classNumber)` - Returns the prior probability of the specified class, $\mathbf{P}(C = classNumber)$, calculated using a maximum likelihood estimate from the training data.
- `getLikelihood(int classNumber, string s)` - Returns the likelihood, that is, the probability $\mathbf{P}(F_1 = f_1, F_2 = f_2, \cdots, F_n = f_n | C = classNumber)$ of the feature vector corresponding to the string given the specified class.

- `classify(string s)` - Returns the MAP class for (i.e., classifies) the specified string, using both `getPriorProbability(···)` and `getLikelihood(···)`.
- `getPosteriorProbability(int classNumber, string s)` - Returns the posterior probability, $\mathbf{P}(C = classNumber | F_1 = f_1, F_2 = f_2, \cdots, F_n = f_n)$, of the specified class given the specified string, using `getPriorProbability(···)`, `getLikelihood(···)`, and the Total Probability Theorem.

You will also fill in three functions in the `feature` class:

- `isFeaturePresent(string s)` - Determines whether or not the feature occurs in the string.
- `addTrainingExample(int featurePresence, int classNumber)` - Updates the counts that will be used to compute the probability of the feature given a class.
- `getProbOfFeatureGivenClass(int featurePresence, int classNumber)` - Returns the probability that the feature is present in a string from the given class, calculated using a maximum likelihood estimate with smoothing (adding one to each feature count in order to avoid getting zero or infinite probabilities).

You must not change the signatures of any of the functions within `naive_bayes_classifier.hpp` or `feature.hpp`. You may add functions to these files if you like.

You do not need to add anything to your write-up for this part.

Part II: Classify Cities

Using the data files in `dat/cities`, use your naive Bayes classifier to classify city names into 2 classes. You will perform 3 experiments:

(a.) US vs. Russia: use usCities100.txt and russiaCities100.txt as training data, and usCitiesNext50.txt and russiaCitiesNext50.txt as test data

(b.) Russia vs. other: use russiaCities100.txt and otherCities100.txt for training, and russiaCitiesNext50.txt and otherCitiesNext50.txt for testing

(c.) US vs. other: use usCities100.txt and otherCities100.txt as training data, and usCitiesNext50.txt and otherCitiesNext50.txt as test data.

The command line format for these experiments is:

```
./textClassifier class1-trainfile class2-trainfile class1-testfile
class2-testfile
```

Run the compiled text classifier program with no arguments for additional instructions.

There is a file within `naive_bayes.tgz` called `accuracy.txt`. This contains the answer for part a. Make sure that your accuracy scores agree with the ones in that file.

In your write-up, report the accuracy of the classifier for parts b and c.

Note that the code reports accuracy, i.e., the fraction of correctly-classified city names, but also another metric, mean squared error. Accuracy only looks at whether the probability produced was above or below 0.5. For mean squared error, we see how far the probability estimate was from the correct answer. If the classifier reports $P = .9$ and the answer is 1, the error is $(1 - .9)^2$. If the answer is 0, the answer is $(.9 - 0)^2$. The average of those (squared) distances is what's reported. Smaller is better.

Part III: Classify Tweets

Twitter provides a search ability which supports filtering recent "tweets" by subject, language, and time (among other options). In the `dat/tweets` directory, you'll find some training and test sets of tweets organized according to language and topic. Using the "hasselhoff" data sets, classify English vs. German words on the topic of Hasselhoff. Report the accuracy in your write-up.

We can check how well the classifier performs on the training data by running the classifier with the same pair of training data files given to the classifier as both the training input and the test input. Try this for the "hasselhoff" data sets. Report the accuracy in your write-up and answer the following questions. Does the classifier have higher accuracy when tested on the training data or when tested on the fresh test data? Why might this be the case?

We could also classify tweets by topic. Using the "republican" and "democrat" data sets, classify English tweets into 2 classes (republican and democrat). Report the accuracy in your write-up.

Part IV: Choose Two Extensions

Pick any two of the following tasks:

1. See if you can improve the performance of your classifier by adding additional features. For example, you could add 2-letter combinations, whether a string starts or ends with a given letter, other regular expressions, whether a string has 3 or more vowels, the length of a string, or other features that might be helpful. Write up what you tried and report the accuracy on the data sets from parts 2 and 3. Did the accuracy improve with these new features? (It's ok if it does not!) Comment about why this might be the case.
2. Code up a "correct" 3-way classifier. Compare its performance to the one that's provided in `text_classifier.cpp` (which simply runs all three 1-class-vs-the-rest classifiers and picks the one with the highest probability). Briefly describe what you did in the write-up, and report the accuracy of both.
3. Using the original set of features, generate a plot showing how the accuracy on the test set varies as a function of the amount of training data given.
4. Think of another task this classifier could do, get data for it, and try it out. Describe the data you chose in the write-up, and report the accuracy of your classifier on this data. Submit the data files.

Extra Credit!!!

Instead of completing two tasks from part IV, you may complete 3 or 4 tasks for extra credit. If you complete all of the requirements for 3 tasks, 1 percentage point will be added to your final class average. If you complete all of the requirements for 4 tasks, 2 percentage points will be added to your final class average.

A note about compilation

We will be testing your code for part I with testing scripts on the Seasnet Linux machines. You may develop your code on any platform you want, but it must compile on one of the Seasnet Linux machines. You can log onto ugrad.seas.ucla.edu via SSH to access the SEASnet Linux machines. You may also work in the Linux lab which is 4405 BH if you choose.

Submission Instructions

You must submit the following through the Assignments/Submit section of Courseweb:

- The full code for Part I which is `feature.hpp` and `naive_bayes_classifier.hpp`.

  IMPORTANT: You must not change the signatures of any of the functions within `naive_bayes_classifier.hpp` or `feature.hpp`. You may add functions to these files if you would like.

- The full code for Part IV. The files you turn in depend on which parts you do:

  1. You can use the same `naive_bayes_classifier.hpp` and `feature.hpp` skeleton code from part I for this part. If you choose to do this part, you must include your source code. Before turning the files in, rename them to `feature_41.hpp` and `naive_bayes_classifier_41.hpp`. Make sure your writeup is clear in describing what you did. (Total: 2 files)
  2. You can again use the same `naive_bayes_classifier.hpp` and `feature.hpp` skeleton code from part I for this part. You must include your source code. For this part, you will also have to modify `text_classifier.cpp`. Before turning the files in, rename them to `text_classifier_42.cpp`, `feature_42.hpp` and `naive_bayes_classifier_42.hpp`. (Total: 3 files)
  3. Your plot should be turned in as either pdf, ps, png or jpeg. Call it `plot_43.pdf` (or .ps, .png, .jpeg) (Total: 1 file)
  4. Your files should be called `class1Training.txt`, `class2Training.txt`, `class1Testing.txt` and `class2Testing.txt` (Total: 4 files)

- A write-up of your results, which must include answers to all of the questions above, as well as a description of your extensions. Your write-up should be submitted as a single PDF document called `writeup.pdf`.

All of these parts must be submitted via courseweb by the deadline, Wednesday, May 30, 2pm. **No late assignments will be accepted.**

The usual academic integrity policy applies for this assignment. You may discuss the assignment with anyone you like, but your code and write-up must be your own work.