Rupali Vohra (rv5rr)

CS 4710: Classifier (Homework 4)

Tuesday, April 28, 2015

**Summary**

The implementation I chose for my classifier involved use of the Naïve Bayes method. This allowed me to easily account for both discrete and continuous variables; a simple proportion was used for discrete features, while the assumption of a Gaussian distribution for continuous variables allowed me to predict outcomes based on means and variances found from the training set.

**Approach**

Though a bit messy, the best way I could think of to keep track of all the various features, whether it was continuous or discrete, and what the outcome was with that given feature was to have multiple maps that kept track of that information. Since I knew I would not be deleting anything from the map, I decided that HashMaps would be the most efficient way to solve the problem.

I ended up creating 9 HashMaps. One of these (featureOptions) keeps track of each feature, and maps it to an array of all the options available for that feature. This was used to populate other HashMaps that are used to keep track of how many times each of these options appears. Two such HashMaps (frequencyMapGreater and frequencyMapLess) were used to keep track of the number of times each option occurred, depending on whether the outcome was “>50K” or “<=50K,” respectively. Similarly, I created two HashMaps (valueMapGreater and valueMapLess) to keep track of the values that appeared for each continuous variable, depending on whether the outcome was “>50K” or “<=50K,” respectively. The maps for discrete variables were used to calculate proportions for probabilities, and the maps for continuous variables were used to calculate the means and variances for each feature, which was then used to calculate probabilities. The probabilities calculated from the discrete variables were stored in two HashMaps (featureProbGreater and featureProbLess), which mapped probabilities for which the classification was “>50K” in the former and outcomes where the classification was “<=50K” in the latter. Similarly, probabilities for continuous variables were tracked in two HashMaps: numericValsGreater, which stored data for when the outcome was “>50K,” and numericValsLess, which stored data for when the outcome was “<=50K.”

All of the HashMaps were populated with data from the training set. The section of the training set I actually used while testing varied (more information given below). This variation did not, obviously, affect my general approach to the problem of making predictions. I did so by following the formula: . In other words, I calculated the probability that each outcome of each feature in the given test case would occur if the classification were “>50K,” and compared that value to the probability that that each feature’s outcome would occur if the classification were “<=50K,” and selected the larger probability as my classification. In the event that these two probabilities tied, I decided to return an outcome based on the decision of a random number generator.

There is a stipulation in the Naïve Bayes approach for classification that mentions the conditional independence for all features in the data set. This is, as stated in the slides, a fairly large assumption, and one that does not hold for the given training set. Specifically, the education\_num field is clearly highly correlated with the education\_level field, and probably derived from that field. Since I created my Classifier to work without accounting for this at first, I decided it would be interesting to see the difference such an account would make in estimations. The results of this experiment are detailed below.

**Testing**

In order to speed up the testing process, I created a testing file called TestCaseGenerator.java. This class asks the user to provide how many of the 1499 training set cases the Classifier should use as part of the training set, and uses the rest as test cases. Then, in order to vary the location of the test data within the training set, the code also prompts the user to specify a section of the training set to use as the test cases, and uses the rest of the training set to train the Classifier. This class then creates the test and training files from the original training set (census.train), creates a solution file from what would have been the last column in the test data, runs the classifier’s prediction, and then compares the output of the classifier to that of the solution. The code prints out the success rate of the classifier.[[1]](#footnote-1)

For the following test data, the following format is used to describe tests: “[cardinality of the training set] training set, [cardinality of the test set] test set; test section is [bucket of the training set from which the test set was taken]”. I also include the mean accuracy for each bucket size.

As previously stated, I was curious to see how including features that essentially measured the same thing factored into the prediction, so I decided to keep the education\_num field for my first few tests. All of the following tests include this feature, until the point where I note the change in my code.

**Tests 1-3**: 500 training set, 999 test set

Average classifier accuracy: 78.79%

**Test 1:** 500 training set, 999 test set; test section is 0

Classifier accuracy: 78.20%

**Test 2**: 500 training set, 999 test set; test section is 1

Classifier accuracy: 76.80%

**Test 3**: 500 training set, 999 test set; test section is 2

Classifier accuracy: 78.96%

**Tests 4-7**: 400 training set, 1099 test set

Average classifier accuracy: 78.84%

**Test 4:** 400 training set, 1099 test set; test section is 0

Classifier accuracy: 78.25%

**Test 5**: 400 training set, 1099 test set; test section is 1

Classifier accuracy: 79.25%

**Test 6**: 400 training set, 1099 test set; test section is 2

Classifier accuracy: 74.50%

**Test 7:** 400 training set, 1099 test set; test section is 3

Classifier accuracy: 79.93%

**Tests 8-12**: 300 training set, 1199 test set

Average classifier accuracy: 78.79%

**Test 8**: 300 training set, 1199 test set; test section is 0

Classifier accuracy: 76.67%

**Test 9**: 300 training set, 1199 test set; test section is 1

Classifier accuracy: 80.00%

**Test 10:** 300 training set, 1199 test set; test section is 2

Classifier accuracy: 76.67%

**Test 11**: 300 training set, 1199 test set; test section is 3

Classifier accuracy: 78.00%

**Test 12**: 300 training set, 1199 test set; test section is 4

Classifier accuracy: 79.93%

**Tests 8-12**: 200 training set, 1299 test set

Average classifier accuracy: 79.42%

**Test 13:** 200 training set, 1299 test set; test section is 0

Classifier accuracy: 77.50%

**Test 14:** 200 training set, 1299 test set; test section is 1

Classifier accuracy: 80.00%

**Test 15:** 200 training set, 1299 test set; test section is 2

Classifier accuracy: 80.50%

**Test 16:** 200 training set, 1299 test set; test section is 3

Classifier accuracy: 81.00%

**Test 17:** 200 training set, 1299 test set; test section is 4

Classifier accuracy: 71.50%

**Test 18:** 200 training set, 1299 test set; test section is 5

Classifier accuracy: 80.50%

**Test 19:** 200 training set, 1299 test set; test section is 6

Classifier accuracy: 78.50%

**Test 20:** 200 training set, 1299 test set; test section is 7

Classifier accuracy: 86.87%

**Test 21:** 100 training set, 1399 test set; test section is 0

Classifier accuracy: 78.00%

**Test 22:** 100 training set, 1399 test set; test section is 1

Classifier accuracy: 78.00%

**Test 23:** 100 training set, 1399 test set; test section is 2

Classifier accuracy: 77.00%

**Test 24:** 100 training set, 1399 test set; test section is 3

Classifier accuracy: 82.00%

**Test 25:** 100 training set, 1399 test set; test section is 4

Classifier accuracy: 81.00%

**Test 26:** 100 training set, 1399 test set; test section is 5

Classifier accuracy: 81.00%

**Test 27:** 100 training set, 1399 test set; test section is 6

Classifier accuracy: 85.00%

**Test 28:** 100 training set, 1399 test set; test section is 7

Classifier accuracy: 78.00%

**Test 29:** 100 training set, 1399 test set; test section is 1

Classifier accuracy: 69.00%

**Test 30:** 100 training set, 1399 test set; test section is 2

Classifier accuracy: 74.00%

**Test 31:** 100 training set, 1399 test set; test section is 3

Classifier accuracy: 76.00%

**Test 32:** 100 training set, 1399 test set; test section is 4

Classifier accuracy: 86.00%

**Test 33:** 100 training set, 1399 test set; test section is 5

Classifier accuracy: 82.00%

**Test 34:** 100 training set, 1399 test set; test section is 6

Classifier accuracy: 75.00%

**Test 35:** 100 training set, 1399 test set; test section is 7

Classifier accuracy: 86.87%

Further tests were conducted, but including them here in such a format did not seem useful, so instead I calculated a mean success rate across all of the buckets in each of the following training-set sizes: 50, 25, 10. I also included the maximum and minimum success rates across the buckets in each category.

**Test 36**: 50 training set, 1449 test set

Classifier accuracy: 78.79%

Max accurate: 90.00%

Min accurate: 68.00%

**Test 37:** 25 training set, 1474 test set

Classifier accuracy: 78.72%

Max accurate: 96.00%

Min accurate: 64.00%

**Test 38**: 10 training set, 1489 test set

Classifier accuracy: 78.72%

Max accurate: 100.00%

Min accurate: 40.00%

1. I have included this testing file with my source code, but since the assignment instructions specifically said not to include any other output, the testing file will not work. It relies on the Classifier saving output to a file called “mysol,” the code for which has been commented out for submission. [↑](#footnote-ref-1)