CS240A - Project 2 Report **Omer Bar-or 303570454**

Important Note

All scripts and results described here are available online at:

http://school.omer.bar-or.org/CS240A/project2/

Project 2 Report

I split this project up into five parts (corresponding scripts are in parentheses):

1. Load the data into DB2 (load.sql).
2. Partition the data into a training set and a testing set and verticalize it: make a record in these sets for each column of each record in the raw data (partition.sql).
3. Create a Naive Bayesian Classifier based on the training set (nbc.sql).
4. Use the Naive Bayesian Classifier to predict classifiers in the testing set and see how well it does (test.sql).
5. Boost the Naive Bayesian Classifier until our predictions for the testing set stop improving (boost.sql).

In fact, one can simply run db2 on these five scripts in order to run all parts of the project.
I created versions of these scripts for two datasets: letter and mushrooms. The scripts are almost identical in both cases, except for the load.sql and partition.sql, which depend on the raw data. The main differences between the two sets of scripts are: a) I discretized the data for the letter dataset, and b) the partition dataset contained unknown values for one of the columns; I decided to ignore them, which means that, when computing P(a,c)/P(c) for some attribute value a and classifier value c, a null value for attribute A affects neither P(a,c) nor P(c); if an unknown value appeared in a testset, its probability was ignored (P(NULL|c) = 1 for all c). After running the above five scripts, the program could predict the correct letter (out of 26 possibilities) in 41.3% of the testset, and it could predict whether a mushroom is poisonous (out of 2 possibilites) in 86.9% of the testset. For the letter dataset, the NBC was boosted 6 times before boosting stopped helping, while it was boosted 10 times for mushrooms. The overall percentage of success is given below:

letter dataset:

Run	Success Rate
1	0.3878243512
2	0.4009980039
3	0.4049900199
4	0.4133732534
5	0.4161676646
6	0.4129740518

mushrooms dataset:

Run	Success Rate
1	0.8414634146
2	0.8448780487
3	0.8560975609
4	0.8609756097
5	0.8629268292
6	0.8653658536
7	0.8658536585
8	0.8668292682
9	0.8687804878
10	0.8687804878

Boosting

steps 1-4 above (everything except for boosting) proved fairly straight-forward, though the test script contains a somewhat complex query, with four WITH statements. But, as the above tables show, the initial predictions before any boosting are fairly high: 38.8% for the letter dataset and 84.1% for the mushroom dataset.

Boosting proved difficult because, initially, each iteration made predictions on the testset worse, not better. And, it did so for both datasets. I tested to make sure that my scripts were doing what I expected them to do by implementing some aspects of the queries in Perl, and these Perl scripts returned the same probabilities and chose the same misclassified records in the training set to boost. I attempted several variations on boosting a misclassified record: I duplicated the record; I gave each record a weight an increased it; I gave each record a weight and doubled it. In each case, boosting lowered the effectiveness of the classifier. So, instead of boosting misclassified records, I attempted to boost *correctly* classified records. It is this version of boosting that leads to the results above.

There are several possible causes for this result. The most obvious and most likely is that my

implementation of boosting is incorrect. It is unlikely that correctly classified records were being chosen incorrectly, because the choice was done by a script similar to test.sql, which (as noted above) gives good results before boosting. I explored two other possibilities. The first possibility is that my data was not fit to be boosted. Indeed, our textbook claims that something special should be done when an NBC has less than 50% success, which is precisely what happens with the letter dataset. It was for this reason that I originally started working with the mushrooms dataset, which is easier to predict as there are only two values for the classifier. But, even for the mushrooms dataset, for which the initial NBC had above 80% success, traditional boosting continued to lower success, while my version increased it.

The second possibility I explored is that my testing script was returning more than one classifier value as having the maximum probability (i.e., at least two values have the same maximum probability). This would explain the strong results of the testing script, which counts the number of times that a *correct* prediction is found, as well as the poor results from boosting, which increases the weight of a record based on whether an *incorrect* prediction is found. If the same record has two predictions, then it would be counted as correct for the testing set, but marked as incorrect (needing boosting) for the training set.

To test this hypothesis, I created four boosters: a) boost all records that are misclassified (traditional boosting), b) boost all records that are not misclassified, c) boost all records that are correctly classified, d) boost all records that are not correctly classified. I then ran these four boosters nine times on the same training set and testing set (for the mushrooms dataset). The results (shown below) reveal two things. First, it reveals that exactly the same success rates are found for (a) and (d) as well as for (b) and (c). This means that there is no distinction between "IN misclassified" and "NOT IN correctly_classified": i.e., that there are no duplicates. Second, the data reveals once again that boosting correctly classified records was beneficial while boosting incorrectly classified records was detrimental. As to why this should be the case, especially given how established the boosting algorithm is, is a mystery to me.

Booster a: IN misclassified

Run	Success Rate
1	.8570024570
2	.8491400491
3	.8054054054
4	.6879606879
5	.6009828009
6	.4776412776
7	.3921375921
8	.3513513513
9	.3267813267

Booster b: NOT IN misclassified

Run	Success Rate
1	.8570024570
2	.8599508599
3	.8619164619
4	.8633906633
5	.8668304668
6	.8702702702
7	.8722358722
8	.8732186732
9	.8771498771

Booster c: IN correctly classified

Run	Success Rate
1	.8570024570
2	.8599508599
3	.8619164619
4	.8633906633
5	.8668304668
6	.8702702702
7	.8722358722
8	.8732186732
9	.8771498771

Booster d: NOT IN correctly classified

Run	Success Rate
1	.8570024570
2	.8491400491
3	.8054054054
4	.6879606879
5	.6009828009
6	.4776412776
7	.3921375921
8	.3513513513
9	.3267813267