Nick Chapman

Data Mining Project 1

February 18, 2016

COSC-285

Project status: Complete

Relevant information not given beforehand:

- Entropy based discretization will take longer to implement than anything else

Project Design:

The system is composed of two main parts: the ARFF processor and the Naïve Bayes Classifer. The ARFF processor handles all actions related to loading data and preprocessing it. For example, when a new ARFF processor is initialized it is given an ARFF file to load, from which it will immediately read the file headers and extract the raw data. Then through 2 main methods the ARFF processor can fill missing values and convert continuous data into categorical data.

To fill in missing data points two strategies are undertaken. For continuous variables the mean value of the records class is calculated and substituted. For categorical values the mode value of that attribute is substituted. In the event that there is more than one mode, a selection is made at random.

To convert the continuous data to categorical, the ARFF processor uses entropy based discretization. It computes the initial entropy of the data set and then seeks to find a partition that will result in the lowest entropy. Each partition is then partitioned again until the entropy threshold is reached. The entropy threshold for this project is set at 0.001, which seems to maximize the classifier accuracy for me. It is important to note that in the discretization strategy it ***does not*** check whether every possible split is viable. Doing so took far too long so I devised an algorithm that would produce reasonably sized steps to take. The formula utilizes logarithms to determine how many points it should actually check at. This is quite useful since it means that for a data set of 10 billion records it won't take 10 Billion^2 or even 10 Billion^3 time. Rather it will get it in a fraction of that.

An aside on calculating splits:

When the function ArffProcessor.get_splits() runs, it recursively calls itself and at every level it utilizes ArffProcessor.find_best_split() which determines where the ideal split would be AND whether or not the split is worth it based on the entropy threshold.

The find_best_split() function ues the entropy() function repeatedly to determine which possible split point will reduce the entropy in the bin the most.

When a split is determined to be worth it the get_splits() function creates a utils.NumericalDataBin object which is simply a range object with the lower bound always being inclusive and the upper bound being exclusive.

When all of the splits are calculated the data bins are sorted and the bottom and top bin have their min and max replaced with -∞ and ∞ respectively. This is so that all data that is sent to the classifier fits into at least one of the bins.

NB: Special care is taken to stitch the ranges of the data bins together so that there are no numbers which don't belong to a bin.

Once the data is processed it is passed to the Naïve Bayes Classifier. The classifier takes an ARFF classifier as its input to build its classification model. The classifier simply calculates all of the probabilities necessary for Bayes' theorem and saves them all. Whenever there is a change in the ARFF data, it becomes necessary to rebuild the model as the model is no longer accurate. However, building the model is actually quite fast so recalculating isn't that intensive.

In order to classify a record, the classifier simply runs Naïve Bayes using the model it has just built and then chooses the best of the calculated probabilities.

Validating the model is done in the main program, which performs a 10 fold cross-validation using randomly selected training and testing data. The testing data is chosen at random from the ARFF data and then when the run is complete it is put back so that new data can be selected. For each run of the system a confusion matrix is generated and then micro and macro analysis is performed. The analysis functions are outsourced to the utils.py file since they don't fit nicely into the flow of main.

Results:

See the attached `Table 1.txt` and `Weka Results.txt`.

Analysis:

From a quick look at Table 1, we can see that the micro analyltics are not terribly useful since they all work out to be the same. This makes sense though since the recall and the precision are inverses of each other. Then the F1 formula returns the same value again because the numerator and denominator cancel out to yield just the precision or the recall, which is equal to the recall or the precision!

The macro statistics are at least a little bit more interesting and varied. We see that the precision hovers in the high 70s and the recall in the low 80s. Both numbers are appear to be pretty good. When comparing against Weka's data we see that it is approximately equal.

However, the Weka categorical precision has values that are vastly different from our own. For example the Weka precision for the >50K class is actually quite low.

On overall accuracy of the model, we compare quite well with Weka. Weka yields an 84% accuracy and our classifier produces an 83% accuracy. This leads one to believe that our Naïve Bayes Classifier is actually doing quite a good job.

When building a data mining model, we are always competing against the worst case scenario of simply guessing what something should be. Compared to simply guessing, our model does approximately 60% better than flipping a coin which is a drastic improvement.