CSE P546 Data Mining - Spring 2007 - Project 1
Clickstream Mining
Jeremy Calvert

Filtering

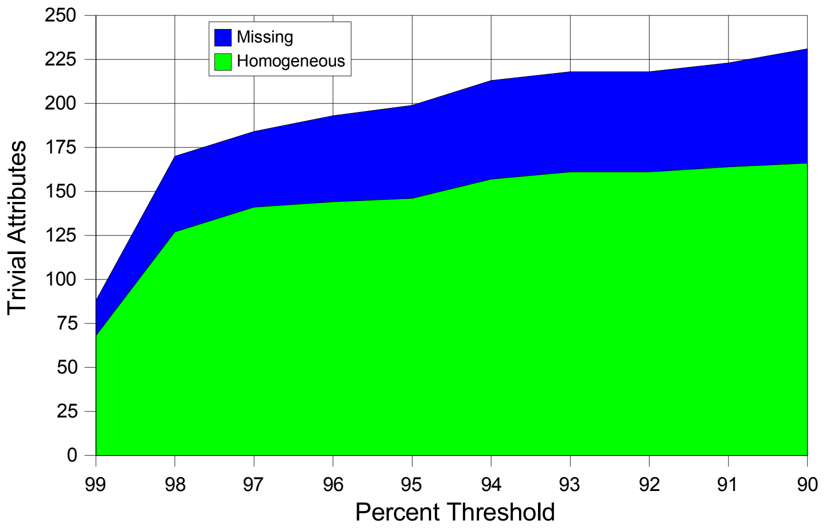
Initially, I loaded the data into Weka's Explorer and immediately noticed that for many of the attributes, almost all of the relations had "NULL" or missing values for said attribute. We'll refer to such an attribute value as "trivial". It seems intuitive that while such attributes will add cost to generation of predictive models, they will add little value to the end result. Implicit in this assertion is that trivially valued attributes have high relation coincidence. In other words, there is a small subset of relations where there are few non-trivial attribute values. This seems to be a safe assumption, based on the fact that the attributes are determined via customer survey, where either it was conducted or it wasn't, or via software instrumentation, where either the software was in place or it wasn't.

Next, I began to remove these attributes by hand. This quickly proved tedious, and was all the more so to do consistently between the training and testing data sets. Looking through the available filters, I found one called RemoveUseless, which seemed to do something along the lines of what I wanted. It removes attributes that are different for "almost all" of the relations. This is worth doing, and eliminates useless attributes such as "Session ID" which are distinct in "almost every" relation, but I also wanted to remove attributes that are the same or missing for "almost all" relations.

So, I wrote a new filter, based on RemoveUseless, which I called "RemoveTrivial". Like "RemoveUseless" it's parameterized by a user-provided percentage *p*. It removes the following additional classes of attributes:

1. Attributes where *p* percent of the relations are missing that attribute.
2. Attributes where *p* percent of the relations have the same value for that attribute (typically "NULL" for nominal attributes and 0 for numerical attributes).
3. Numeric attributes that have (*p* * *number_relations*) distinct value ("RemoveUseless" only does this for nominal attributes).

I scripted a run of this on the training set for a spectrum of values of *p*, yielding the following relationship between *p* and the number of attributes deemed nontrivial.



In order for Weka to accept a separate testing set, the attributes have to be identical to those found in the training set. So RemoveTrivial produces a list of trivial attribute indexes which can be removed via the "Remove" filter from the test data.

Model Runs

With our trimmed down training set, it's possible to run many more modeling algorithms and compare results. As instructed, we start with the J48 classification algorithm. First, we run the same algorithm with and with out the trimmed data, to verify that there is no substantial loss in accuracy. We do this for `weka.classifiers.trees.J48 -S -C 0.25 -B -M 2` using a 66% split of the full training set into training and testing. For the full data set, we get an accuracy of 80.3067%, while for the trimmed data set, we get 80.3405%. So we actually obtain a gain in accuracy while the model requires approximately 1/3 as much CPU time to generate, amounting to a 1.3 hour long process execution time rather than four. We should also note, however, that the degenerate ZeroR model that predicts "false" as the class for every relation has an accuracy of 80.4825% on the training data subset, and takes no time whatsoever to generate!

Trees
Next I tried variations on the parameters in the J48 classifier. With subtree raising and reduced error pruning turned on, and binary splits turned off, the accuracy (on the separate training set) drops to 73.656%, although the model generation takes 10 minutes rather than 100 minutes. With counts at leaves smoothed with Lapace, reduced error pruning, and binary splits on, the accuracy is 73.612%. These are both below the accuracy of ZeroR on the testing set, 74.832%.

Bayes
The naïve Bayes classifier with the default settings performed pretty dismally. It achieved an accuracy of 26.3409% on the training data, and oddly somewhat better, at 32.168% on the test data. Next we tried it with the kernel estimator enabled and the accuracy jumped up to 73.1419%. Still this is woefully far below the accuracy of the degenerate predictor.

Rules
There are a wide array of rules based classifiers provided by Weka, so I wrote a shell class to try each of them on the trimmed training subset. ZeroR has, unfortunately, been the most accurate rule, at 74.832% accuracy, indicating an utter failure to find any meaningful correlation of relation to class. Ridor turned out to be equivalent to ZeroR. Prism reports that it can only handle nominal attributes. PART achieves a 71.432% accuracy, OneR achieves 74.704%. NNGE gets 66.068%, Jrip: 74.292%, Decision table: 73.86%, Conjunctive rule is equivalent to ZeroR, and M5Rules require numeric class value.

Others
Having failed thus far to improve on ZeroR, I cast a wider net. Since classifiers are run, in the library code, by passing an instance `c` of a Classifier to `Evaluation.`*`evaluateModel`*`(c, argv)`, where `argv` is the list of command-line parameters, why not try *all* of the classifiers? So I wrote a small class to find all the Classifiers (via reflection) and deal them out to a set of machines. Some of the models are not applicable, because they can only handle nominal attributes, or they could only handle numeric attributes, or nominal or numeric classes. Unfortunately none of the models outperformed ZeroR, though they were all run with their default parameter settings. Some models weren't applicable to the type of relations in our dataset and others caused OOM crashes even with 5GB of physical memory at their disposal.

Analysis
My approach has been to regard this as an exercise to find a model on the provided training data to predict classes in the provided test data. Since we ultimately have to find a model on the test data, it doesn't seem to make sense to try and improve on the training data from the raw click session data, because this and the test data would be "apples and oranges".

As such, the provided data seems to be very sparse. Given more time, perhaps it would make sense to focus on the 2% of relations that have more than 90 non-trivial attribute values. One could come up with a model that predicts false if the relation is deemed to be trivial (meaning most of its attribute values are trivial), and then apply a model derived from the 2% of relations. This could potentially increase the accuracy by up to 2% (achieving 76.832% accuracy). According to the organizer's report:

> For Question 1 (Given a set of page views will the visitor view another page on the site or will the visitor leave), the accuracy values ranged from 77.06% to 59.56% with a mean of 73.14%. The difference between the top two performers was only 0.10%, which translates into 50 sessions. In fact, the difference in accuracy of the top five participants was statistically insignificant (a 95% confidence interval corresponds to ±=0.37%).

Unless I'm mistaken, this says that the ZeroR predictor outperformed the mean. This also indicates that predicting false for all the relations with mostly trivial attribute values, and finding a moderately successful predictor for relations with mostly non-trivial attributes could have an accuracy within the confidence interval of the top performer from the KDD cup.

Conclusion

A major lesson from this exercise seems to be that a primary component of mining real world data involves dealing with sparse attributes. Had the majority of attributes, particularly the ostensibly useful ones (system generated, such as "Num _____ Views") been present in even 10% of the relations, higher success rates may have been much easier. As the data was, this seems to have been more an exercise in data filtering and scrubbing than analyzing.