A-011

Mining Rebate Systems

Skip Jordan*

1 Introduction

The Data Mining Cup[1] is an annual, international, *student competition* in data mining. This year, 688 participants of 159 universities from 40 countries considered the application of data mining to optimizing rebate coupons. The task was to mine customer coupon redemption history in order to identify potential coupon users and the type of coupon these customers are likely to respond to.

We introduce the task, discuss some statistical properties of the data, describe our implementation of instance-based classification, and report on our techniques searching for better-performing parameters. Our submission ranked #47 and was the highest scoring non-European submission. We also provide some ideas for improving our performance.

2 Background

Rebate systems and couponing are important tools for retailers. Although issuing coupons incurs a cost, if additional revenue is generated and customers retained or increased, the expenditure is likely to be justified. The widespread use of membership cards allows retailers to make individualized decisions at checkout regarding which coupon to issue a customer. Using past customer history, it is possible to apply data mining techniques to estimate which coupon will optimize the benefit to the retailer.

Training data was provided regarding a trial run of a new coupon, which was available in two varieties, A and B. During the trial run, 50,000 customers were given both coupons. The data includes the coupon redeemed by each: neither coupon (N), A or B. Cases in which both A and B were redeemed were discarded before the training data was produced. For each of these 50,000 customers, the training data includes a 22-tuple consisting of their ID number, the number of times that they redeemed each of past coupons 1 to 20, and the result of the trial (N, A or B). There were 3,307 redemptions of A, 8,668 of B and 38,026 of neither.

An additional 50,000 lines of test data was provided with the results of the trial stripped (21-tuples, ID numbers and the number of redemptions for each of the same 20 past coupons). Issuing no coupon (N) results in no cost to the retailer. Issuing an unredeemed coupon costs 1 unit. Issuing a coupon A that is redeemed gives a benefit of 3, while a redeemed B gives a benefit of 6. Thus, denoting the size of the group assigned X that actually redeemed Y in the trial run as $|XY|$, the goal was to provide a set of 50,000 assignments for the test data maximizing

$$C := 3|AA|+6|BB|-(|AB|+|AN|+|BA|+|BN|). \quad (1)$$

Each participant submitted a set of assignments for the test set to be scored by the organizers.

3 Implementation

We examined the 20-tuples of coupon redemption history in the training data and noticed that if a 20-tuple is present, then it is usually present multiple times. More precisely, 48,639 (97.3%) lines of the test data occur exactly at least once in the training data. As an extreme example, one tuple of coupon redemption history occurs 6,134 times (12.3%) in the test data and 6,062 times in the training data.

Given this information, and making the assumption that information is not encoded in ID numbers, we consider classifying based on exact duplicates. For example, the 20-tuple consisting of zeros (customers who did not redeem any past coupons) is present 844 times in the training data. Although we might expect these customers to also ignore the new coupon, only 1 of them did so. In this group, 626 redeemed A and 217 redeemed B. This provides an estimate on a probability distribution for the 20-tuple of all zeros. We wish to maximize (1), and so in this case assign A to all 795 test elements that are zero vectors.

Our algorithm, given an input vector v of past customer history to which we wish to assign a coupon, is essentially:

```
for (d=0 to infinity)
  Find all u with distance(u,v) at most d.
  If the number of such u is < THRESHOLD,
    increment d and repeat the loop.
  For each possible assignment (A,B or N),
    compute C over these u.
  Return the assignment maximizing C.
```

If two (or more) assignments result in exactly the same expected value of C, we compare three tie-breaking methods: *conservative*, where N is chosen in ties involving it, and A chosen in A-B ties, *aggressive*, where B is chosen in ties involving it and A chosen in A-N ties, and *lazy*, where we increment d and postpone the assignment.

*Graduate School of Information Science and Technology, Hokkaido University, email: skip@ist.hokudai.ac.jp

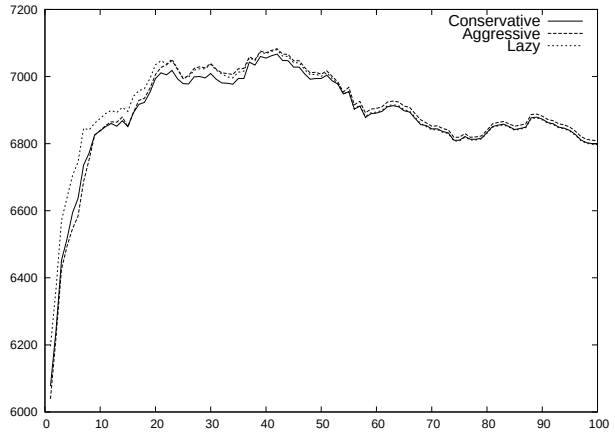
[1] http://www.data-mining-cup.com/

Next we discuss our search for a suitable threshold and distance function.

4 Threshold

Our implementation refuses to classify instances for which an insufficient number of sufficiently similar training examples exist. In these cases, it increases the distance which it considers acceptable and tries again. The idea is to avoid giving excessive influence to individual training points. A threshold of one effectively disables the parameter.

The following is a plot of scores on one random 10-fold cross-validation for each of the threshold tie-breaking strategies at thresholds from 1 to 100.



We chose a threshold value of 21 and a conservative strategy. Using this threshold seemed to consistently give a 1,000 point gain. It is close to the maximum, and we preferred to use the lower threshold.

5 Distance measure

We considered Euclidean distance, squared Euclidean distance, Manhattan distance (1-norm), and the number of differing attributes to determine the distance between two vectors of coupon redemption history. The latter two are in some senses quite intuitive: we can imagine a customer was sick on a given day, accidentally lost a coupon, or perhaps there was an error in data entry. Two coupon histories that differ on only a single attribute seem likely to be correlated. Using the Manhattan distance is a slight refinement of the idea, a customer who uses a given coupon twice is likely closer to someone who uses it once than to someone who doesn't use it. Multiple redemptions of a given coupon are rare in the provided data, but examples do exist (in the training set, 721 (1.4%) customers redeemed a coupon twice and 16 (0.03%) redeemed a coupon three times).

That the Euclidean distance and squared Euclidean distance have different effects is a result of using a fixed discrete increment in the distance at which we consider a training point similar.

The four distance measures performed fairly similarly, however for every 10-fold cross-validation run that we analyzed, the Manhattan distance slightly outperformed the others and squared Euclidean distance outperformed Euclidean distance. The results are summarized in Table 1. "Split 1" refers to using the first 45,000 lines of training data for training and last 5,000 for test. "Split 2" refers to using one random 45,000:5,000 split. The "10-fold" column refers to the arithmetic mean of scores over at least ten 10-fold cross-validation runs. All scores use a threshold of 21 and are computed using C, see (1).

Classifier	Split 1	Split 2	10-fold
$Manhattan$	647	812	7131
$Euclid^2$	646	812	7046
$Euclid$	643	819	6966
$NumberD$	641	808	7066

Table 1: Comparison of distance measures

6 Weighting function

We then considered a weighting w on the attributes used when computing the distance measures. We would then define squared weighted Euclidean distance for example, as

$$dist(a,b) = \sum_{i=1}^{20} w_i(a_i - b_i)^2,$$

where the w_i are real numbers. The unweighted variant sets all $w_i = 1.0$.

To find a suitable set of weights within time constraints, we first restricted our attention to the top representative measures from the two types: Manhattan distance and squared Euclidean distance. We used the following weight-finding algorithm.

```
Perform two separate random 10-fold splits.
Set oldscore=bestscore=-infinity.
for (j=1 to 40)
   Do 10-fold cross-validation runs on each.
   Set newscore to be the sum of these scores.
1: If newscore>oldscore,
      set oldscore=newscore, oldweights=w.
      If newscore>bestscore,
         set bestscore=newscore, bestweight=w.
   If oldscore>newscore,
      set w=oldweights.
   Randomly choose a weight to modify.
   Randomly choose a non-zero v, -0.25<v<0.25.
   Add v to the weight, use new weights as w.
Repeat the entire algorithm indefinitely.
```

We also changed line 1 to allow non-beneficial changes with very low probability in order to hopefully move away from local optima. Thus, we tune our weights to optimize the mean of two random 10-fold cross-validation splits, frequently changing these splits to avoid excessive changes in response to "bad" splits.

Scores of weighted algorithms using a threshold of 21 appear in Table 2. The columns have the same meaning as above. Scores here use weights derived randomly using the above algorithm for at least 24 hours.

Classifier	Split 1	Split 2	10-fold
$wManhattan$	717	800	7267
$wEuclid^2$	671	774	7278

Table 2: Comparison of weighted distance measures

The difference in the last column is not statistically significant, and in most additional cases we observed, Manhattan distance wins by 2-3 points. Weighting seems more important than the precise choice of distance measure.

We continued tuning weights until shortly before the deadline, performing additional tests intended to prevent regression.

7 Combining models

We used various techniques to combine our models and classification schemes from Weka[1]. Our first technique, $\langle X, Y \rangle$, classifies inputs using classifier X, unless X's output is N, in which case it classifies unconditionally as Y. The notation can be nested. The idea was to use a very accurate classifier as X and a more aggressive classifier as Y, hopefully minimizing A-B confusion. However, this method of combination was consistently beaten by the next method.

We next considered a simple plurality vote, denoted here with hard brackets, which can also be nested. Tied votes are broken aggressively, choosing B first and then A. The results of various combinations of classifiers are given in Table 3. New classifier names represent the corresponding classifier in Weka, using *default settings* except for cost-sensitive classification per C. All Weka scores would presumably improve with individual attention, however we investigated hundreds of combinations. Due to technical details of our implementation and time constraints, the 10-fold column here represents one random split, not an average over multiple splits. Combinations involving the same classifier use different weights for each instance. We abbreviate wManhattan as "wMan," wEuclid^2 as "wEuc^2", and NumberD as "NumD".

Classifier	Split 1	Split 2	10-fold
[[wMan, wMan], BayesNet]	730	819	7335
[wMan, wMan]	724	813	7271
[wMan, BayesNet]	719	800	7329
[wMan, wEuc^2], BayesNet]	713	789	7362
[wEuc^2, BayesNet]	687	796	7334
[NumD, BayesNet]	675	811	7166

Table 3: Comparison of classifier combinations

Upon further examining the output of these classifiers, we noticed that the set of assignments generated by our classifiers changed *drastically* (over 1,000 lines) in comparison to the score improvement (30-40 points) when combined with BayesNet.

8 Results

We chose to submit what we denote as [wManhattan, wManhattan] with the sets of tuned weights separated by approximately 24 hours of tuning. These were the last pair of weights we were able to test to our satisfaction, and were derived after approximately a week of CPU time. Due to a relative lack of data and the comparatively large changes for comparatively small (but present) gain of combining with BayesNet with our techniques, we were unwilling to take what we perceived as a risk. The choice of wManhattan over wEuclid^2 was primarily intuition; we were able to imagine a real meaning in the Manhattan distance in this particular case.

Our final score was 7,140, somewhat lower than the average of our score on 10-fold cross-validations, but within the range of our scores. This difference is possibly due to the test set containing more N than the training set or over-training weights. The winning score was 7,890.[2]

9 Future improvements

There were a number of improvements that we were unable to implement and test for regressions due to the time constraints of the contest. In the future we plan to tune weights in a more systematic fashion (possibly with hill-climbing), which would drastically decrease tuning time. Additionally, although our classifier estimates a probability distribution for input vectors, it does not give these distributions when it votes. That is, a classifier that feels 80% confident in its classification should be treated differently than one that only feels 4% confident.

Our meta-process was to choose first a threshold, then a distance measure and weighting function, and finally a combination thereof. It would be better to choose these simultaneously, that is, the threshold may depend on the weighting function. This search should also be further automated. Although our search algorithms produced large tables of scores, decisions about parameters were made manually.

Acknowledgements

The author wishes to thank his supervisor, Prof. Thomas Zeugmann, for his encouragement and guidance in this project.

References

[1] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 2005.

[2] DMC 2007 Student Results, www.data-mining-cup.com/2007/Wettbewerb/Preise/.