CBSA Project Report

Mu Zhu, Hugh A. Chipman, Lu Xin, Nancy N. Zhang, R. Wayne Oldford[i]

December 2008

1. Background

We were asked by CBSA to investigate a prediction problem in 83 dimensions. The training set contained 15,279 observations, each labeled either "clean" or "dirty." The goal was to find good ranking algorithms so that "dirty" observations can be ranked ahead of the "clean" ones, and to test all algorithms on an independent test set containing 6,259 observations, which were *not* to be used to help develop the algorithm in any way.

Performance metric

This is a "rare target detection" problem, because most training data belong to the "clean" class and only a tiny fraction belong to the "dirty" class. CBSA used the area under the ROC curve (AUC) as the main performance metric to evaluate the effectiveness of various algorithms. This is a suitable — and, in fact, the most widely used — performance metric in the statistics community. We also focused on this metric, using the "ROCR" package in R to compute AUC, but we will also briefly describe a more direct performance metric below.

Benchmark

The learning algorithm produced internally by scientists at CBSA (simply the "CBSA algorithm" below) had an AUC performance of 0.7294 on the test set.

2. Results

We experimented with a number of learning algorithms. In principle, one would normally begin with simple algorithms and gradually move towards more complicated ones if simple methods were inadequate. However, we primarily focused on algorithms that our students were studying, such as the ones outlined in Zhu (2008). When relevant, tuning parameters were selected by two-fold cross validation on the training set alone, without using the test data set in any way. Once the tuning parameters were selected, the algorithm was re-trained using the entire training set before being applied to the test set. Table 1 summarizes our main results; details of our work are given in the Appendix.

Conditional results

Results shown in Table 1 are *conditional* on the fact that all algorithms were trained on a particular training set and evaluated on a particular test set, i.e., the ones given to us by CBSA, which use the last six months of data as the test set. If different training and test sets had been used — e.g., if the data were randomly separated into two halves, the results could very well have been different. In fact, there is often considerable variability associated with creating the "train-test" split. An algorithm may perform well with one "train-test" split and poorly with another. Tiger Woods is considered the best golfer in the world, not because he has won a few occasional tournaments, but because, over many tournaments, he has consistently demonstrated that he is hard to beat. Likewise, to find the best learning algorithm for any given data set, it is often necessary to repeat the same experiments many times, each time using a different "train-test" split. A large number of such repetitions are often needed in order to detect the typically small differences among competing algorithms, just as many tournaments are needed in order to identify the best golfer in the world. Results shown in Table 1 consists of just one such "tournament," and they may not be enough for us to draw a final conclusion about these algorithms' relative strengths and their respective suitability for this particular problem.

Table 1. Performance of various learning algorithms on the independent test set. The CBSA algorithm has a performance of 0.729 on the AUC scale.

Algorithm		*Best possible (AUC)*	*Regular*	
			AUC	*Top 500*
Ensemble approach				
Random Forest				
- R package (Figure 3)	(a)	0.652	-	-
- "quick and dirty" (Figure 4)		0.742	0.727	18
- Salford Systems (Table 4)	(b)	0.738	-	-
AdaBoost (Figure 5)		0.680	0.678	12
Kernel approach				
SVM – radial basis kernel (Figure 6)		0.734	0.720	19
LAGO – radial basis kernel (Figure 7)		0.695	0.683	13
Bayesian approach				
BART – default prior		-	0.676	-
Linear approach (18 predictors; Figure 8)	(c)			
Logit	(d)	-	0.728	20
SVM – linear kernel (Figure 9)		0.737	0.722	20
LAGO on the unit sphere (Figure 10)		0.723	0.720	20
Experimental approach	(c)			
Logit + trend heuristic (Figure 11)	(d)	-	0.741	21

(a) Cross validation results are omitted, and irrelevant given the poor "best possible" performance.
(b) A 3-day trial version of this commercial software was used; cross validation was not performed.
(c) These experiments were added as this report was being written.
(d) There are no tuning parameters for this method.

"Best possible" results

It is clear from Table 1 that the CBSA algorithm is quite competitive. We also asked the following question: what could the "best possible" performance be for each of these algorithms? To answer this question, we simply fitted, for each algorithm, a series of models on the entire training set using a variety of different tuning parameters, and examined the performance of these models on the test set. This allowed us to obtain a rough estimate of the "best possible" performance for the algorithm under consideration, one that we would have obtained if we had been able to find the ideal tuning parameters for the test set itself — notice that reporting this kind of result would normally be considered "cheating." Table 1 shows that the CBSA algorithm remains competitive against these "best possible" results.

A low-dimensional logit model

As part of our data exploration on the training set, we also found that 18 out of the 83 predictors were more important than others, and that linear classifiers trained using just those 18 predictors could perform surprisingly well (Appendix A.4). In particular, the first right singular direction (similar to the first principal component direction) of the training set was dominated by 18 predictors: X1, X2, X4, X12, X22, X23, X24, X25, X40, X42, X47, X51, X59, X61, X63, X72, X77, and X81 (see Figure 8). A standard logistic regression model fitted using just these predictors (simply the "low-dimensional logit model" below) performed quite well on the test set (see Table 1), suggesting that these predictors *may* span a potentially informative subspace! As previously forewarned, however, we are hesitant to jump to this conclusion without having repeated all the experiments with different training and test sets.

3. Discussion

We now discuss a few points that we deem relevant and important.

A more direct performance metric

Given two observations, X and Z, let S_X and S_Z be the scores assigned to them by a certain algorithm. It can be proven that (see, e.g., Pepe 2003)

$$\text{AUC} = \text{Prob}(S_X > S_Z \mid X \text{ is "dirty" and } Z \text{ is "clean"}).$$

That is, AUC can be interpreted as the (conditional) probability that, given a pair of "dirty" and "clean" observations, the algorithm correctly ranks the "dirty" one ahead of the "clean" one. If we randomly rank these observations, this probability is 50%. This probabilistic interpretation of the AUC is important because it gives us the much needed intuition to judge just how good (or bad) an algorithm is if its AUC is 72%. In this regard, an even more direct performance metric is to simply count the number of "hits" among the n top-ranked items. For example, our "quick and dirty" random forest got 18 "hits" among the first 500 of its top-ranked items from the test set (see Table 1). At first glance, one might think that this kind of performance is not very good because an overwhelming majority of the top-ranked items are not "hits." However, on this particular test set, one would expect only about 5.35 "hits" if we selected 500 items at random —

about 1%. Therefore, the algorithm is doing 3+ times better than random selection. Put in this perspective, the performance is not as bad as it first appeared to be. Many find this direct performance metric to be more intuitive and easier to understand than the AUC.

Differences between training and test sets

There is overwhelming evidence that the test set seems to be quite different from the training set. Based on two-fold cross validation on the training set alone, we estimated that the AUC would be about 58–61% for most algorithms we investigated (see Figure 4, Figure 6, Figure 7, Figure 9, and Figure 10). When the algorithms were applied to the test set, however, their AUCs invariably increased by almost 10 percentage points, to about 68–72%! We found this highly surprising. Figure 1 shows the same 2-dimensional projection of both the training set and the test set, using the first two right singular directions estimated on the training set. The distributions of the majority class ("clean") appear to be fairly stable across the training and test sets, but the distributions of the rare class ("dirty") appear quite different. This is not entirely surprising — the rare class is expected to exhibit more sampling variations, just by virtue of being rare. However, in the current context, there exist other possibilities. As we were told, the test set consists of only marine containers arriving between July and December, whereas the training set includes containers arriving between January and June as well. We don't have the information to check whether this is what caused the distributional differences, but we can easily think of reasons why this could be so. If seasonal differences exist, then different prediction models should be built for different seasons. We think this direction is worthy of further pursuit. Another possibility is that the distributions are changing over time (more on this below), which would call for a different approach.

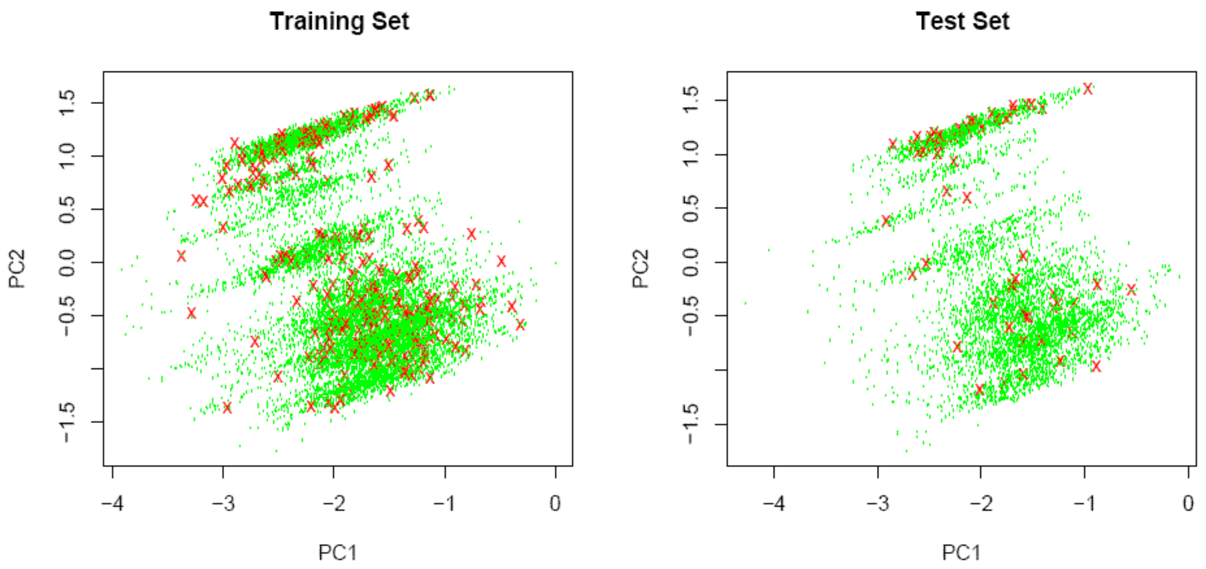

Figure 1. Visualization of training and test data using the first two right singular directions (similar to the first two principal components) of the training set. Green dots denote "clean" observations and red crosses denote "dirty" ones.

Distributional changes over time

As we were told, the training set consists of data from 2006 and the first half of 2007, whereas the test set consists of data from the second half of 2007. Though individual time stamps were not available, the training set was sent to us in three separate parts. We conjectured that these records might have been organized chronologically. Figure 2 contains the same type of plots as Figure 1, except that the three parts of the training set were plotted separately. If our conjecture held, then "training set – part 1" would correspond roughly to the first half of 2006, "training set – part 2" to the second half of 2006, and "training set – part 3" to the first half of 2007. Figure 2 would then tell a very interesting story. Let us divide this two-dimensional space roughly into two regions: top and bottom. In the top region, the distributions appear relatively stationary over time. In the bottom region, however, the prevalence of "dirty" containers appears to be decreasing steadily from 2006 to 2007! Clearly, for test data in the bottom region, ignoring a "trend" like this would lead us to overestimate their probabilities of being "dirty."

A simple "trend heuristic"

To take this "trend" into account, we implemented a very simple heuristic (Figure 11 and Appendix A.5), and it improved the performance of our "low-dimensional logit model" on the test set from AUC = 0.728 to AUC = 0.741.

Kernels for binary data

For kernel-based algorithms (i.e., SVM and LAGO), we used the radial basis kernel function. It is arguably not the most suitable kernel function for binary predictors. It is of considerable research interest to ask what the "right" kernel function is for binary data. We will be investigating this direction during the next phase. The reason why it has not been pursued in the current phase is because it is not easy to experiment with non-standard kernel functions using only the standard implementations of these kernel algorithms.

Prediction contest

In the winter semester of 2009, we will be offering a graduate course on "Kernels and Ensembles," and we are considering using these data as a course project in the form of a prediction contest. The plan is to challenge the students to each come up with the best prediction algorithm possible. We think we might gain some more insights by opening this problem to a group of fresh-minded students. We also feel that we have not experimented enough with the more basic but often powerful learning algorithms. For example, it was interesting to see that we could do quite well with just logistic regression plus variable selection, and would like to find out whether simple strategies like this can hold up against repeated trials.

4. Summary

We experimented with a variety of learning algorithms. Conditional on the given training and test sets, our experiments confirmed that the CBSA algorithm was quite competitive. Our experiments also led to a number of interesting findings and conjectures: (i) a simple, low-dimensional logit model performed quite well on the test set; (ii) the current training set and test set are different; (iii) data distributions may be changing over time; and, (iv) we may be able to model the distributional changes and improve our predictions.

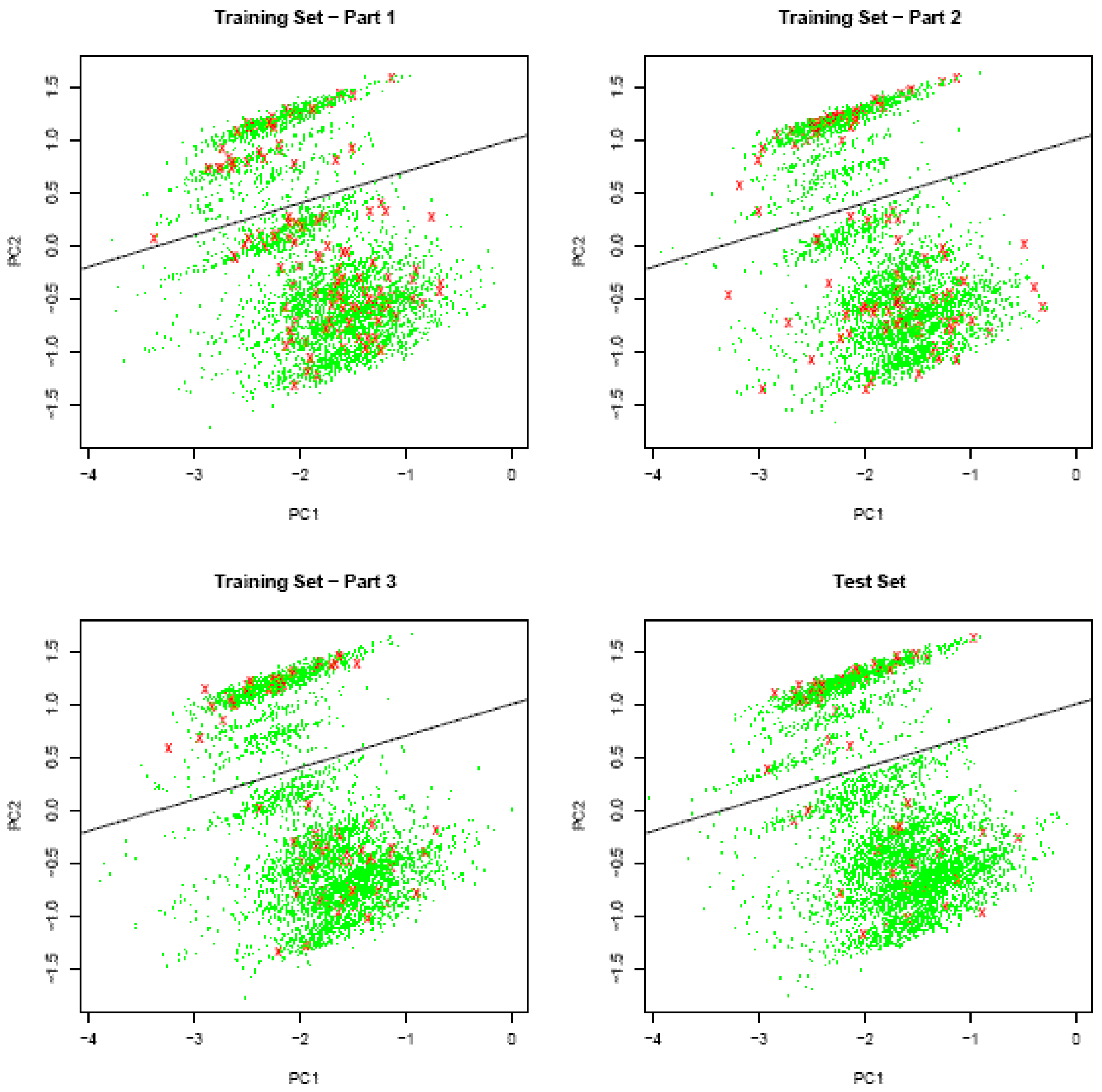

Figure 2. Same two-dimensional visualization as Figure 1, except that the three parts of the training set are now plotted separately. The diagonal line is a manually-added boundary to separate the space into two regions, top and bottom. The prevalence of "dirty" observations appears to be decreasing in the bottom region.

Appendix

This appendix provides more details about our work. However, if a learning algorithm is standard (i.e., it is well documented in the literature, and not invented, assembled, or modified for this particular project), we will not describe it in much detail here. The “clean:dirty” ratio on the training set was about 49:1, so training observations labeled “dirty” were weighted 49 times more heavily than those labeled “clean” when running all algorithms *except* LAGO, which automatically weights the rare class more heavily.

A.1. Ensemble approach (Lu Xin, Mu Zhu)

According to the project description we received from CBSA, the CBSA algorithm is essentially a hand-crafted learning ensemble, consisting of 7 hand-picked decision trees from a pool of over a thousand. Ensemble methods are indeed very “hot” in the machine-learning community; their strong predictive power is also well supported by much empirical evidence. Since CBSA started by taking an ensemble approach, we decided to start our investigation by experimenting with more standard learning ensembles built in a more systematic fashion, such as random forest (Breiman 2001) and AdaBoost (Freund and Schapire 1996).

Random forest in R

We started by using the “randomForest” package in R. Contrary to our initial expectations, its performance turned out to be quite poor (Figure 3). We investigated and discovered that the R implementation of random forest contained a bug. We asked the forest to average its individual trees’ probability estimates (rather than their final binary votes), but it appeared that this was the same as averaging the votes. A bug such as this would give rise to a significant loss of “resolution,” which would explain the poor performance on the AUC scale. A fine “resolution” is not as critical for classification, but it is absolutely essential for ranking.

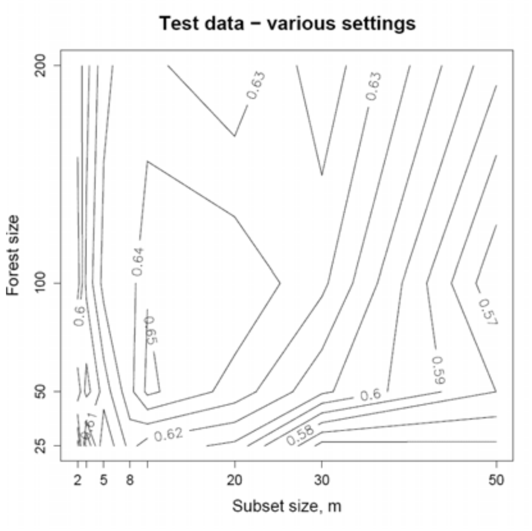

Figure 3. Results from the R implementation of random forest. The “best possible” performance on the test set is about AUC = 0.652.

A “quick and dirty” random forest

In view of the aforementioned difficulty, we proceeded to develop a “quick and dirty” version of random forest on our own (Table 2); the original random forest algorithm by Breiman is described in Table 3 for comparison. To fit each individual tree in our forest, we used the “rpart” package in R. The main difference between our “quick and dirty” version and Breiman’s original version of random forest is as follows:

- In our “quick and dirty” random forest, a random subset of predictors is selected before building each tree. As a result, every split inside a single tree is optimized over the same subset of predictors (but different trees use different subsets).

- In Breiman’s original random forest, a random subset of predictors is selected not before building each tree but before making each split. As a result, even the splits inside a single tree are optimized over different subsets of predictors.

Notice that, while Breiman’s random forest allows each individual tree to grow to its maximal complexity, our “quick and dirty” random forest seems to be sensitive to the complexity of its individual trees, which is controlled by the “cp” parameter in “rpart” (see Figure 4).

Table 2. Our "quick and dirty" random forest algorithm; d = total number of predictors.

```
1) Choose a subset size, $m<d$, and a complexity parameter, $c$.
2) For $b = 1$ to $B$
   a) Draw a bootstrap sample of the data, call it $D^{*b}$.
   b) Randomly select a subset of $m<d$ predictors, call it $S_b$.
   c) Using $D^{*b}$, build a tree, $f_b$, using only predictors in the subset, $S_b$, and the complexity parameter,
      $c$.
   End For
3) For a new observation, each tree in the forest, $f_b$, will produce an estimated probability (that it is
   “dirty”). Average these probabilities over all trees in the forest to rank the new observation.
```

Table 3. Breiman's original random forest algorithm; d = total number of predictors.

```
1) Choose a subset size, $m<d$.
2) For $b = 1$ to $B$
   a) Draw a bootstrap sample of the data, call it $D^{*b}$.
   b) Using $D^{*b}$, build a tree of maximal complexity, $f_b$, as follows: for each split $j$, (i) form a subset,
      $S_{bj}$, by randomly selecting $m<d$ predictors; (ii) choose the best split by a greedy search on the
      set $S_{bj}$, rather than over all possible predictors.
   End For
3) For a new observation, each tree in the forest, $f_b$, will produce an estimated probability (that it is
   “dirty”). Average these probabilities over all trees in the forest to rank the new observation.
```

A commercial random forest

Our “quick and dirty” random forest improved upon the R implementation, but the amount of improvement remained small. Concerned that our “quick and dirty” approach might have been too crude, we obtained a *3-day trial version* of a commercial random forest software distributed by Salford Systems (www.salfordsystems.com). Due to the limited-time access to this software, we skipped cross validation and proceeded directly to estimate the “best possible” result on the test set. However, the commercial implementation did not significantly outperform our “quick and dirty” approach (Table 4).

AdaBoost

Next, we experimented with a different kind of learning ensemble. Instead of building up an ensemble using an *iid* stochastic mechanism, e.g., random forest, we can also build up an ensemble sequentially, allowing each subsequent member to address the weakness of the previous one. This is what AdaBoost does. We implemented AdaBoost ourselves in R, following Zhu (2008; Table 1) and using trees as our base learners, which were fitted with the “rpart” package in R. Instead of using the “cp” parameter in “rpart,” we controlled the complexity of our trees by limiting their “depths.” This was computationally more efficient. As it turned out, AdaBoost did not perform as well as random forest (Figure 5).

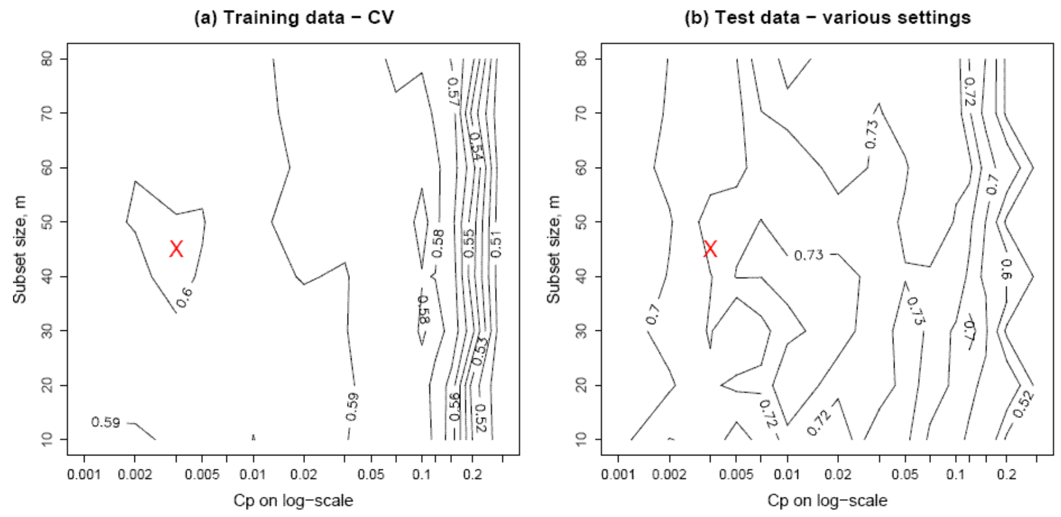

Figure 4. Results from our "quick and dirty" random forest, with a forest size of 100 trees. The "X" marks the tuning parameters selected by cross validation. The "best possible" performance on the test set is about AUC = 0.742.

Table 4. Results from a commercial random forest software distributed by Salford Systems (3-day trial version), with a forest size of 500 trees (default).

Subset size, *m*	AUC
3	0.728
5	0.721
8	0.730
10	0.728
20	0.734
30	0.738
40	0.738
55	0.737
75	0.732

A.2. Kernel approach (Nancy Zhang, Mu Zhu)

We also experimented with another class of learning algorithms, known as kernel methods, e.g., the support vector machine (Figure 6) and LAGO (Figure 7). For more details about these algorithms, see Zhu (2008). We used the radial basis kernel to run both SVM and LAGO.

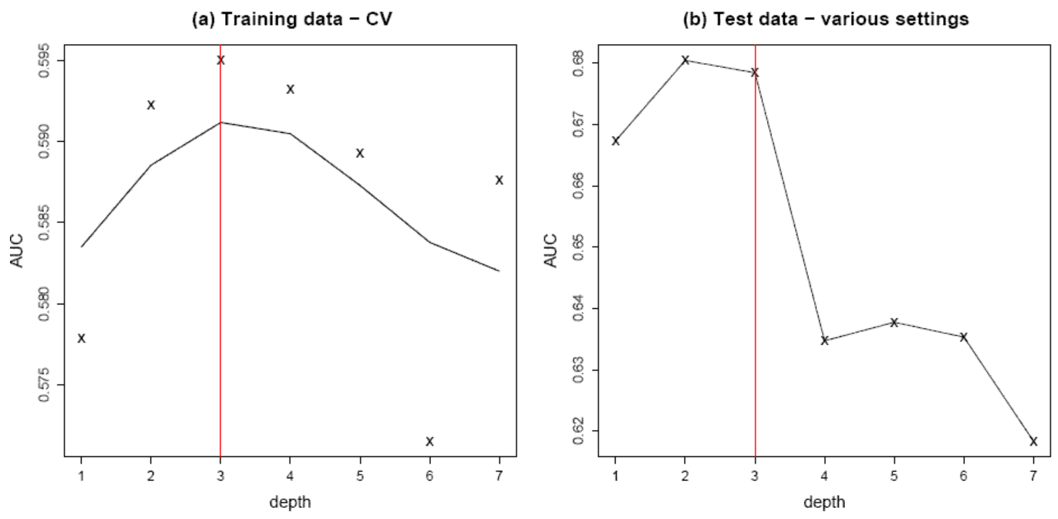

Figure 5. Results from AdaBoost, with 100 trees built sequentially. The vertical line marks the complexity parameter selected by cross validation for each individual tree. The "best possible" performance on the test set is about AUC = 0.680.

Low-information predictors

As soon as we started to experiment with kernel methods, we discovered that, for three predictors (X29, X30, and X36), all training observations have their values equal to zero. Therefore, the training set does not contain any information about these three predictors; consequently, these predictors were not used. The reason why this artifact was not discovered during the "ensemble" phase (see Section A.1) of our project is because, if a predictor contains no information, the tree method will simply ignore that predictor automatically and not make any split using that predictor. Since useless predictors tend to "confuse" kernel-based algorithms, this raised the question of whether other "low-information" predictors should be discarded as well. For example, we computed the first right singular direction using the training data. As expected, this direction was dominated by only a few predictors. We repeated some of our experiments using these predictors only, but the final results were almost identical, i.e., AUC ~ 0.72 – 0.74 for random forest, AUC ~ 0.72 – 0.73 for SVM, and AUC ~ 0.68 – 0.69 for LAGO.

A.3. Bayesian approach (Hugh Chipman)

Next, we ran a Bayesian ensemble algorithm called BART (Chipman, George, and McCulloch, 2008). Using the default prior distribution for its various parameters, BART produced a result of AUC = 0.676 on the test set. Potentially, this result could be improved further by a more suitable choice of the prior distribution, but we didn't have the resources to pursue it at this stage.

A.4. Linear approach (Mu Zhu)

As previously mentioned (Section A.2), the first right singular direction on the training set was dominated by only a few predictors (Figure 8). Out of pure curiosity, we fitted a standard logistic regression model on the training set using just these predictors, applied the model to the test set, and obtained an AUC of 0.728. In view of Figure 2, we found it remarkable that a *linear* classifier could do so well, so we experimented with two other linear algorithms using the same subset of predictors: SVM with a linear kernel (Figure 9), and LAGO on the unit sphere (Figure 10); see Laflamme-Sanders and Zhu (2008). The final results were comparable. However, logistic regression using predictors selected by a stepwise optimization of the Akaike information criterion (AIC) did not yield as good a result.

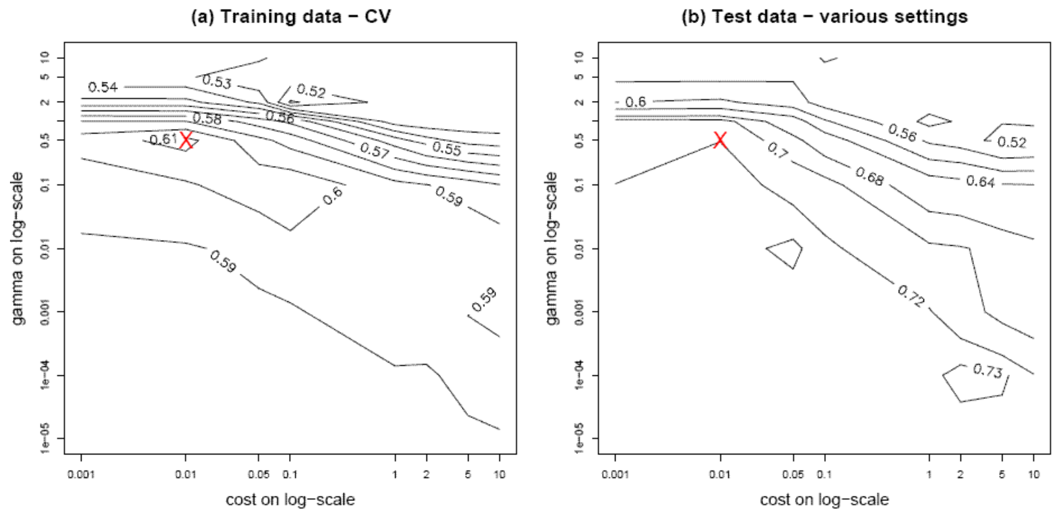

Figure 6. Results from SVM, using the "e1071" package in R. The "X" marks the tuning parameters selected by cross validation. The "best possible" performance on the test set is about AUC = 0.734.

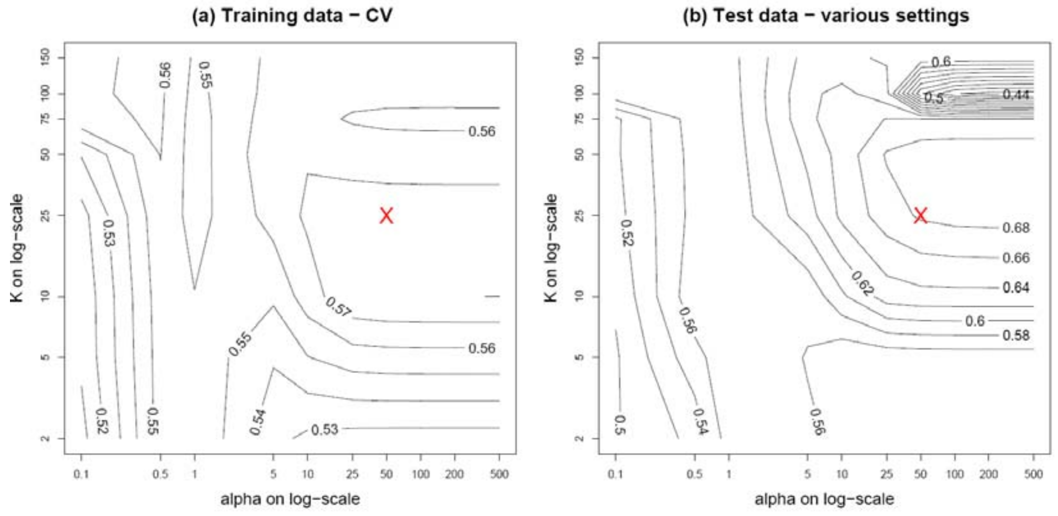

Figure 7. Results from LAGO, using the "lago" package in R. The "X" marks the tuning parameters selected by cross validation. The "best possible" performance on the test set is about AUC = 0.695.

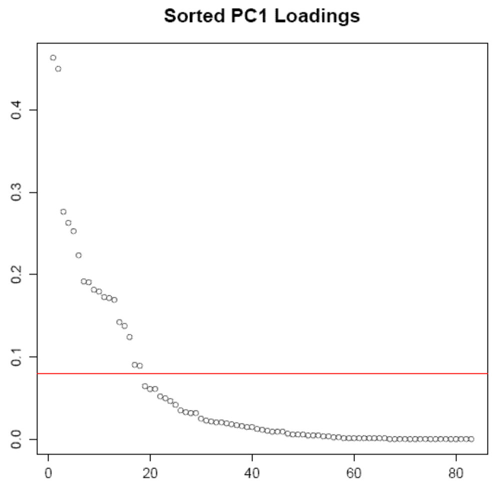

Figure 8. Sorted absolute loadings for the first right singular direction. Linear classifiers fitted using the top 18 predictors lying above the horizontal line are found to perform quite well on the test set.

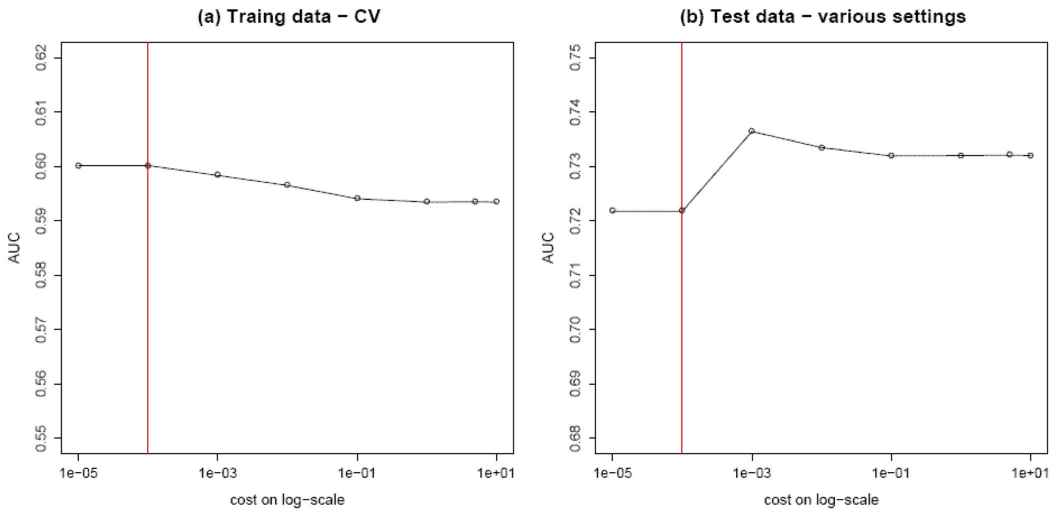

Figure 9. Results from running SVM using the linear kernel. The vertical line marks the tuning parameter selected by cross validation. The “best possible” performance on the test set is about AUC = 0.737.

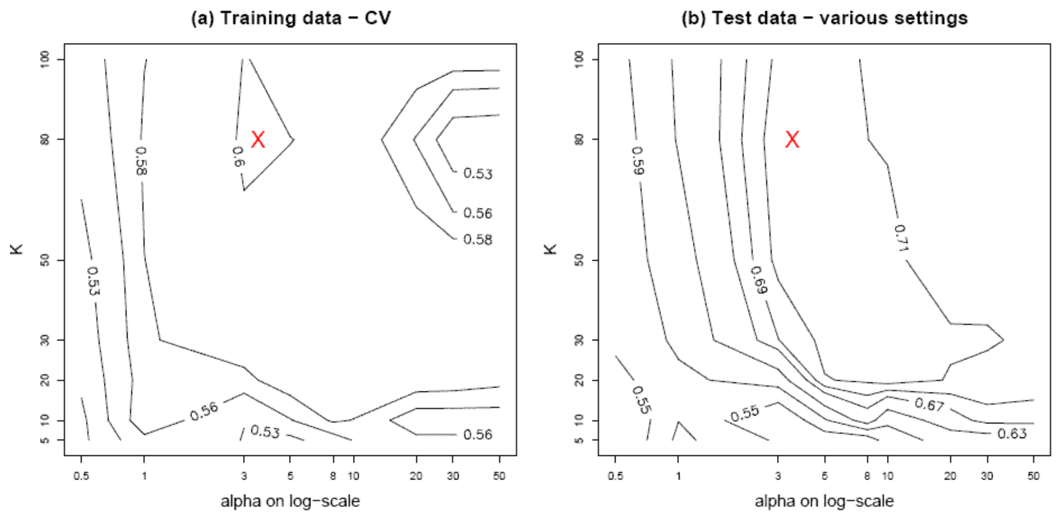

Figure 10. Results from running LAGO on the unit sphere. The "X" marks the tuning parameters selected by cross validation. The "best possible" performance on the test set is about AUC = 0.723.

A.5. Experimental approach (Mu Zhu, Wayne Oldford)

The training set was delivered to us in three parts. Having noticed some differences between the training and test sets in a two-dimensional projection (Figure 1; simply the "2D-space" below), we conjectured that the three parts of the training set might have been in chronological order. For each part, we estimated the prevalence of "dirty" observations in the bottom region of this 2D-space (see Figure 2). We then fitted a simple linear regression line to model the decreasing trend. Using this regression line, we predict that, on the test set, the prevalence in this region would be about 1/3 of the prevalence in the same region on the training set (Figure 11). To incorporate this information, we applied a simple "trend heuristic" to our baseline predictions, e.g., those produced by the low-dimensional logit model: for test observations lying in the bottom region of the 2D-space, we discounted their baseline predictions by a factor of about 1/3. This increased the AUC on the test set from 0.728 to 0.741.

References

Breiman, L. (2001), "Random Forests," *Machine Learning*, **45**, 5 – 32.

Chipman, H. A., George, E. I., McCulloch, R. E. (2008), "BART: Bayesian Additive Regression Trees," arXiv:0806.3286.

Freund, Y., and Schapire, R. (1996), "Experiments With a New Boosting Algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference*, San Francisco: Morgan Kauffman, 148 – 156.

Laflamme-Sanders, A., Zhu, M. (2008), "LAGO on the unit sphere," *Neural Networks*, **21**, 1220 – 1223.

Pepe, M. S. (2003), *The Statistical Evaluation of Medical Tests for Classification and Prediction*, Oxford University Press..

Zhu, M. (2008), "Kernels and ensembles: Perspectives on statistical learning," *The American Statistician*, **62**, 97 – 109.

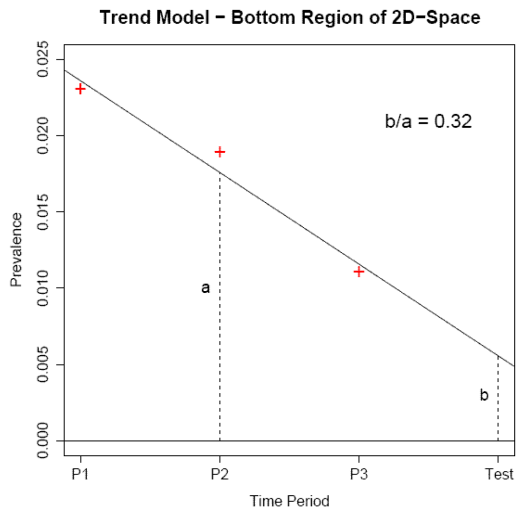

Figure 11. A simple "trend heuristic." Using hypothesized chronological information on the training set, it is estimated that, on the test set, the prevalence of "dirty" observations in the bottom region of the 2D-space (see Figure 2) will be about 1/3 of the prevalence in the same region on the training set. Therefore, this heuristic mandates that baseline predictions for test observations lying in this region be discounted by a factor of about 1/3.

[i] Zhu is Associate Professor and Oldford is Professor and Director of Computational Mathematics, University of Waterloo. Chipman is Professor and Canada Research Chair, Acadia University. Xin is MMath student and Zhang is BMath student, University of Waterloo. Other than Zhu and Oldford, all contributing authors are listed in alphabetical order. Xin is the only one who received salary support from the CBSA-Waterloo contract.