JMLR: Workshop and Conference Proceedings 7: 23-34 KDD CUP 2009

Winning the KDD Cup Orange Challenge with Ensemble Selection

Alexandru Niculescu-Mizil, Claudia Perlich, Grzegorz Swirszcz, Vikas Sindhwani, Yan Liu, Prem Melville, Dong Wang, Jing Xiao, Jianying Hu, Moninder Singh, Wei Xiong Shang, Yan Feng Zhu
IBM Research ANICULE@US.IBM.COM

Editor: Gideon Dror, Marc Boullé, Isabelle Guyon, Vincent Lemaire, David Vogel

Abstract

We describe our wining solution for the KDD Cup Orange Challenge.

1. Introduction and Task Description

The KDD Cup 2009 challenge was to predict, from customer data provided by the French Telecom company Orange, the propensity of customers to switch providers (churn), buy new products or services (appetency), or buy upgrades or add-ons (up-selling). The competition had two challenges: the Fast challenge and the Slow challenge. For the Fast challenge, after the targets on the training set were released, the participants had five days to submit predictions on the test set. For the Slow challenge, participants were given an additional month to submit their prediction.

The data set consisted of 100000 instances, split randomly into equally sized training and test sets. 15000 variables were made available for prediction, out of which 260 were categorical. Most of the categorical variables, and 333 of the continuous variables had missing values. To maintain the confidentiality of customers, all variables were scrambled. There was no description of what each variable measured.

For the Slow challenge, a reduced version of the data set was also provided, consisting of a subset of 230 variables, 40 of which were categorical. The small data set was scrambled differently than the large one, and the rows and columns were shuffled. Many participants, including ourselves, easily found the correspondence between instances in the small and large data sets. Uncovering this correspondence, however, provided us little benefit, if any.

Submissions were scored based on the Area Under the ROC Curve (AUC) performance, with the average AUC across the three tasks being used to rank the participants. Feedback was provided in terms of performance on a fixed 10% of the test set. While multiple submissions per team were allowed, the competition rules stated that only the last submission from the team leader will count towards the final ranking, so the participants had to face the burden of model selection. The slow challenge presented one additional twist in terms of evaluation. Participants were allowed to make two sets of submissions, one on the large and one on the small data set, and the best of the two was considered toward the final ranking.

Table 1: Our journey.

Classifier Type	Feat. Set	Chlng.	Churn	Appetency	Up-Selling	Average
Slow Challenge Submission (ES)	FS3	Slow	0.7651	0.8816	0.9091	0.8519
Ensemble Selection	FS3	Slow	0.7629	0.8805	0.9091	0.8509
Fast Challenge Submission (ES)	FS2	Fast	0.7611	0.8830	0.9038	0.8493
Ensemble Selection	FS2	Fast	0.7611	0.8793	0.9047	0.8484
Best Competitor, Slow		Slow	0.7570	0.8836	0.9048	0.8484
Ensemble Selection	FS1	Fast	0.7563	0.8771	0.9038	0.8457
Best Competitor, Fast		Fast	0.7565	0.8724	0.9056	0.8448
Best Single Classifier	FS3	Slow	0.7511	0.8794	0.9025	0.8443
Best Single Classifier	FS2	Fast	0.7475	0.8779	0.9000	0.8418
Best Single Classifier	FS1	Fast	0.7354	0.8779	0.9000	0.8378

As a final note, we want to emphasize that the results we present in this paper reflect the particular choices we have made and directions we have explored under the limited time of the competition. They are not a careful empirical study of the different methods we have used. So, while we will make a few comparative statements throughout the paper, we caution the reader against generalizing these results beyond the scope of this competition.

2. Our Story

Our overall strategy was to address this challenge using Ensemble Selection (Caruana and Niculescu-Mizil, 2004). In a nutshell Ensemble Selection is an overproduce-and-select ensemble building method that is designed to generate high performing ensembles from large, heterogeneous libraries of classifiers. Ensemble Selection has several desirable properties that made it a good fit for this challenge. First, it has been proven to be a robust ensemble building technique that yields excellent performance. Second, the generated ensembles can be optimized to any easily computable performance metric, including AUC. Third, it allows for loose coordination of the team members, as everyone can independently train classifiers using their preferred techniques, and add those classifiers to the library. And fourth, it is an anytime method, in the sense that when the time to make predictions comes, an ensemble can be generated very fast using whatever classifiers made it into the library at that time.

Our results are summarized in Table 1. The first column indicates the classifier type (the best individual classifier we have trained, an ensemble generated by Ensemble Selection, or the submission of our competitors). The second column indicates what feature set was used (FS1 indicates the set of features provided, after some standard preprocessing summarized in Section 2.1; FS2 indicates the use of additional features created to capture some non-linearity in the data, as described in Section 2.2.3; FS3 indicates the use of even more additional features described in Section 2.3.1). The next columns show the test set AUC on the three problems, and the average AUC. Entries are ordered by average AUC.

In the following sections we will present our work in close to chronological order to motivate our choices as we went along.

2.1 Preprocessing, Cleaning and Experimental Setup

Since the feature values were available prior to the targets, we spent our initial time on some fairly standard preprocessing. The data set posed a number of challenges here: many features, missing values, and categorical variables with a huge number of possible values.

Missing categorical values are typically less of a problem as they can be considered just a separate value. Missing numeric values on the other hand are more concerning. We followed

a standard approach of imputing missing values by the mean of the feature. We considered that the missingness itself might predictive and added, for each of the 333 variables with missing values, an additional indicator variable indicating missingness. Another advantage of this approach is that some class of models (e.g. linear) can now estimate the optimal constant to replace the missing value with, rather than relying on the means.

Most of the learning algorithms we were planning to use do not handle categorical variables, so we needed to recode them. This was done in a standard way, by generating indicator variables for the different values a categorical attribute could take. The only slightly non-standard decision was to limit ourselves to encoding only the 10 most common values of each categorical attribute, rather than all the values, in order to avoid an explosion in the number of features from variables with a huge vocabulary.

Finally, the features were normalized by dividing by their range, and the data was cleaned by eliminating all the features that were either constant on the training set, or were duplicates of other features. In the end we were left with 13436 features.

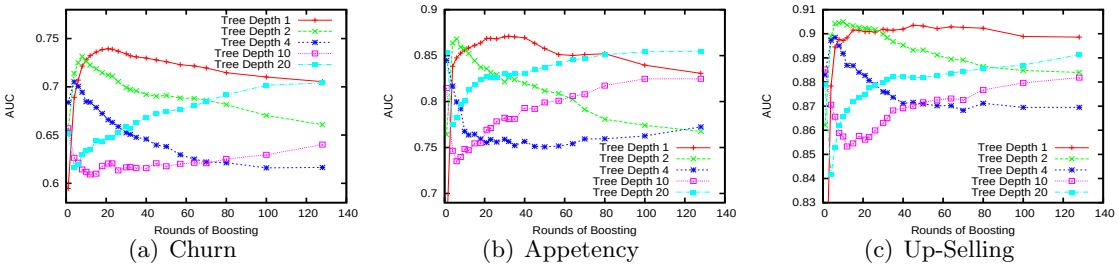
To evaluate the performance of the classifiers, and build ensembles via Ensemble Selection, we adopted a 10-fold cross-validation approach. While we would have liked to perform all the 10 iterations of the cross-validation, considering, in turn, each fold as a validation fold, this was unrealistic in the allotted time. Ultimately we only finished two iterations for the Fast challenge, giving us a total of 10000 validation instances, and four iterations for the Slow challenge, for a total of 20000 validation instances. To make predictions on the test set, given that we now had a version of a classifier for each fold (two for the Fast challenge and four for the Slow one), we averaged the predictions of the corresponding models. This has a bagging like effect that should lead to a performance boost.

In order to avoid overfitting the test set of the leader board, we decided not to rely on the feedback on the 10% for anything but sanity checks and final guidance in picking an ensemble building strategy.

2.2 The Fast Challenge

2.2.1 Many Different Classifiers

The first step in building an ensemble classifier via Ensemble Selection is to generate a library of base classifiers. To this end, we trained classifiers using using a range of learning methods, parameter settings and feature sets. We looked for learning algorithms that were efficient enough to handle a data set of this size in the allotted time, while still producing high performing models. Guided in part by the results of (Caruana et al., 2008), we generated classifier libraries using random forests (Breiman, 2001) and boosted trees (Schapire, 2001) trained using the FEST package (Caruana et al., 2008), logistic regression trained using the BBR package (Genkin et al., 2007), SVMs trained using SVMPerf (Joachims, 2005), LibLinear (Fan et al., 2008) and LibSVM (Chang and Lin, 2001), decision trees, TANs and Naïve Bayes trained using Weka (Witten and Frank, 2005), Sparse Network of Winnows trained using the SNoW package (Carlson et al., 1999), and k-NN, regularized least squares regression and co-clustering (Sindhwani et al., 2008) trained using in house code. We also trained some of these learning algorithms on several reduced feature sets obtained through PCA and through feature selection using filter methods based on Pearson correlation and mutual information. For a complete description of the trained classifiers see



(a) Churn (b) Appetency (c) Up-Selling

Figure 1: Performance obtained by boosting decision trees of various depths.

Appendix A. To make all base models "talk the same language" we applied post training calibration using Platt Scaling (Niculescu-Mizil and Caruana, 2005). For classifiers that make predictions between 0 and 1 we also put the uncalibrated classifiers in the library.

Little or no attempt was made to optimize the performance of the individual models; all models, no matter their performance, were added to the model library. The expectation is that some of the models will yield good performance, either in isolation or in combination with other models. In total, the classifier libraries were composed of 500-1000 individual models for each of the three problems problem.

Judging from the two folds of internal cross-validation we performed up to this point, the best individual classifiers on churn were boosted trees followed by regularized logistic regression, and random forests. On appetency, the best single method was random forests, followed by boosted trees and logistic regression, while on up-selling, boosted trees were best, followed by random forests and logistic regression. At this point, using the best single classifiers, as deemed by the internal cross-validation, yielded a test set AUC of 0.7354 for churn, 0.8779 for appetency, and 0.9000 on up-selling, for an average AUC of 0.8378 (last line in Table 1). This was lower than the AUC obtained by many of the competing teams.

Interestingly, on all three problems, boosting clearly overfit with more rounds of boosting (see Figure 1), yielding peak performance after only about 10-20 rounds and decreasing significantly after that (except for boosted stumps on the up-selling problem). Also boosting shallow trees performed better than boosting deeper trees. One trend, that we actually did not notice during the competition, is that boosting deeper trees (more than 10 levels) has a dip in performance during early stages of boosting, but recovers later. It is conceivable that higher performance could have been obtained by boosting deeper trees for longer.

For the linear models, we tried optimizing four different loss functions: hinge loss and AUC, with L_2 regularized SVMs, squared error with L_2 regularized least squares, and cross-entropy (log-loss, log-likelihood), with logistic regression using both L_1 and L_2 regularization. As expected, optimizing hinge loss yielded significantly lower AUC scores than directly optimizing AUC. What was less expected was that optimizing cross-entropy with L_2 regularization did as well as, or even slightly better than directly optimizing AUC. Using L_1 regularization with logistic regression further improved the AUC performance and using feature selection on top provided yet another slight performance boost.

One worry one might have would be that training all these base level models would be very resource demanding. Training all the base level models for all three problems took little more than one day per cross-validation fold on a cluster of nine dual Opteron nodes (2 GHz) with 3Gb memory. So, while training all these models is by no means cheap, the computational load can be easily handled with fairly modest resources.

2.2.2 Ensemble Selection

Once a classifier library is generated, Ensemble Selection builds an ensemble by selecting from the library the subset of classifiers that yield the best performance on the target optimization metric (AUC in our case). Models are selected for inclusion in the ensemble using greedy forward stepwise classifier selection. The performance of adding a potential model to the ensemble is estimated using a *hillclimbing* set containing data not used to train the base classifiers. At each step ensemble selection adds to the ensemble the classifier in the library that maximizes the performance of the ensemble on this held-aside hillclimbing data. Classifiers can be added to the ensemble multiple times, allowing for a crude weighting of the different classifiers in the ensemble.

When there are a large number of base classifiers to select from, the chances of overfitting increase dramatically. Caruana and Niculescu-Mizil (2004) describe two methods to combat overfitting. The first is to initialize the ensemble with a set of N classifiers that have the best uni-model performance on the hillclimbing set. The second performs classifier bagging—multiple ensembles are built from random subsets of classifiers, and then averaged together. The aim of the classifier bagging is to increase performance by reducing the variance of the forward stepwise selection process.

We built an AUC optimized ensemble classifier for each of the three problems using Ensemble Selection. Following (Caruana et al., 2006), we combined the two validation folds from our internal cross-validation and used them as the Ensemble Selection hillclimbing set. Both overfitting prevention techniques described above were used. The test set performance of the ensemble models was 0.7563 on churn, 0.8771 on appetency and 0.9038 on up-selling, for an average AUC of 0.8457. This performance was already better than that of the other competitors on the Fast challenge. It is notable that this performance was obtained through fairly standard techniques without much need for human expertise or intervention, or tailoring to the particular problems addressed in this competition. One can easily imagine all these techniques being incorporated in a general purpose push-button application.

At the time of the competition, however, we did not know that we had the best performance. In fact, on the 10% of the test set that was used for feedback our performance was below that of other participants.

2.2.3 More Features

We had one more day to push forward. This was when we realised that there were notable discrepancies between measuring the quality of individual variables via mutual information with the targets, and via rank correlation with the targets. Some of the features with the highest mutual information (calculated by binning numeric variables into 20 bins) had quite a poor rank correlation. This was most likely due to some form of non-monotonic dependence, so, while these features were very predictive, linear models could not take advantage of them. Our solution was to construct new features to allow expressing non-linear relationships in a linear model. To this extent we explored two approaches:

- **Binning:** As explained earlier, we observe higher predictive performances in terms of mutual information when binning was used. So the obvious solution was to include, for each such feature, 20 additional binary features corresponding to the 20 bins. However, it is unlikely that the equal size binning is optimal.

- **Decision Tree:** The second approach was to use a decision tree to identify the optimal splitting points. We recode each feature by training a decision tree of limited depth (2,3 or 4) using that feature alone, and let the tree directly predict the target. The probabilistic predictions of this decision tree were used as an additional feature, that now was linearly (or at least monotonically) correlated with the target.

The addition of these new features had a significant impact on the performance of linear models, with L_1 regularized logistic regression becoming the best model on churn and improving the test set performance of the best base level churn model by 0.0121 to 0.7475. It also had a positive impact on the performance of the ensembles build by Ensemble Selection for all three problems, resulting in a test set performance of 0.7611 on churn, 0.8793 on appetency, and 0.9047 on up-selling.(See entries using FS2 in Table 1.)

2.2.4 Submission for the Fast Challenge

Before the final submission for the Fast challenge, we analysed in more detail the ensembles built by Ensemble Selection. We realized that on appetency, after the initialization phase (where models with high uni-model performance were added to the ensemble), the first model Ensemble Selection was adding was some poor performing decision tree. We were worried that this indicates that Ensemble Selection was actually overfitting the hillclimb set. So, for appetency, we decided to use the ensemble model generated right after the initialization phase, containing only the six best models (as measured on the hillclimb set), and not continue with the forward stepwise classifier selection. The results on the 10% of the test set were also in accord with this hypothesis. In hindsight, it turned out to be the right decision, as it significantly improved our performance on the test set.[1]

We have also investigated whether classifier bagging was necessary, by running Ensemble Selection with this option turned off. We noted that, on the 10% of the test set we received feedback on, classifier bagging provided no benefit on churn and up-selling, and was only increasing performance on appetency (which was consistent with our hypothesis that Ensemble Selection overfit on this problem). Being also guided by the results in (Caruana et al., 2006), which stated that, once the hillclimbing set is large enough, classifier bagging is unnecessary, we decided to use ensembles built without classifier bagging as our final submissions for churn and up-selling. In hindsight, this was not a good decision, as the test set performance on up-selling was slightly worse than if we were to use classifier bagging.

2.3 Slow Challenge

For the slow challenge, we first increased the hillclimbing/validation set to 20000 instances by training on two extra folds, bringing us to four corss-validation folds.

Encouraged by the improvements we obtained in the last day of the Fast challenge, the main thrust of our efforts was towards creating new and better features. The addition of these features, described below, in combination with the move from two to four folds, yielded an increase in test set performance for the best individual models to 0.7511 on churn, 0.8794 on appetency and 0.9025 on up-selling (0.8443 average across the three problems).

1. This was not the case for the other two problems, churn and up-selling, where the forward stepwise classifier selection improved the performance significantly.

The performance of the Ensemble Selection built ensembles rose to 0.7629 for churn, 0.8805 for appetency, and 0.9091 for up-selling (0.8509 average).

2.3.1 Even More Features

Explicit Feature Construction: For a number of features with typical characteristics, we were able to isolate the signal directly: The positive rate of churn for all rows with 0 value was up to twice the positive rate for all other numeric values. This happened for a number of numeric features that overall seemed to be close to normally distributed, but, under close inspection, showed certain regularities, such as frequency spikes for certain values. The effect is not overly strong; typically only a few thousand examples have a zero value and a zero indicator for a single such numeric feature only has an AUC of up to 0.515. However, counting the number of times an example had a zero within one of these numeric features had an validation AUC of 0.62.

Features From Tree Induction: We extended the decision tree based recoding approach to pairs of attributes in order to get two way non-additive interactions between pairs of variables. To this end, for each pair of attributes, we trained a decision tree of limited depth (3,4) to predict the target from only the two attributes. We then used the predictions of the tree as an extra feature. We only used the constructed features that outperformed, by a significant margin, both individual attributes.

Co-clustering: We have also tried a new feature generation approach. When looking at missing values, we noticed that they were missing for groups of features at once. That is, for every instance, the values for all the features in the group were either all missing or all present. Inspired by this observation, we extend the idea to other categorical/numerical values. For example, suppose that features f_1, f_2, and f_3 take values a, b, and c respectively across instances i_1, i_2, i_3, and i_4. We can then generate a feature, that takes 1 on i_1, i_2, i_3, and i_4 and 0 on all other instances. The problem of identifying subsets of features/instances with this property, is known as the constant bi-clusters problem in the bio-informatics domain. We ran a fast probabilistic bi-clustering algorithm (Xiao et al., 2008) to identify promising bi-clusters which we then used to generate new indicator features.

2.3.2 Small Data Set

We quickly trained all our models on the small data set, but the internal cross-validation results suggested that the performance obtained from the small data set was significantly worse than what we obtained from the large one. So we decided not to pursue the small data set any more, and focused our attention on the large data set. Nevertheless, we did unscramble the small data set for two main reasons: to get feedback on about 20% of the test data instead of only 10%, and to be able to make two distinct submissions using models trained on the better performing large data set (per competition rules, the best of the two submissions would count towards the Slow challenge ranking). In the end, however, it turned out that unscrambling the small set did not give us any significant advantage as the feedback on the 20% of the data was barely used, and the two submissions we ended up making were very similar to each other.

2.3.3 Submission for Slow Challenge

Finally, we again analyzed the ensembles produced by Ensemble Selection in more detail, and noticed some strange behaviour with the ensemble initialization phase. Because the model libraries contained a large number of high performing, but very similar logistic regression classifiers, in the initialization phase, Ensemble Selection was adding all these classifiers to the ensemble, essentially overemphasizing the logistic regression models. Given that we also had a larger hillclimb set, overfitting was less of a concern, so we decided to turn off the ensemble initialization. With the initialization turned off, we gained, on average, another 0.001 in test set AUC, for a final performance of 0.7651 on churn, 0.8816 on appetency, and 0.9091 on up-selling (0.8519 average AUC).

3. Conclusions

Our winning solution for the 2009 KDD Cup Orange Challenge was to use Ensemble Selection to generate and ensemble model from a large library of 500-1000 base classifiers for each problem. While it is hard to give a definite answer, we believe that our success was mainly due to three factors. The first factor was the exploration of a large variety of learning methods. As the results show, different learning methods were best for different problems. The second factor was the use of an ensemble building technique that is able to take advantage of the large variety of base classifiers without overfitting. The third factor was the creation of additional features capturing non-linearities and other helpful signals in the data.

Acknowledgements

We thank Nikos Karampatziakis and Ainur Yessenalina for help with the FEST package and helpful discussions, Art Munson for Platt Scaling code, and Ramesh Natarajan for helpful discussions. We are also grateful to the many people who made their code available on-line. Finally, we thank the 2009 KDD Cup organizers for a fun and exciting competition.

References

Michael W. Berry. Svdpack: A fortran-77 software library for the sparse singular value decomposition. Technical report, Knoxville, TN, USA, 1992.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Andrew J. Carlson, Chad M. Cumby, Jeff L. Rosen, and Dan Roth. Snow user guide. Technical report, 1999.

Rich Caruana and Alexandru Niculescu-Mizil. Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*, 2004.

Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. Getting the most out of ensemble selection. In *Proceedings of the 6th International Conference on Data Mining (ICDM '06)*, 2006.

Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Alexander Genkin, David D. Lewis, and David Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49:291–304(14), August 2007.

Thorsten Joachims. A support vector method for multivariate performance measures. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 2005.

George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132, 1995.

Nicholas Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *COLT '91: Proceedings of the fourth annual workshop on Computational Learning Theory*, pages 147–156, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1-55860-213-5.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, 2005.

R. M. Rifkin. *Everything Old is new again: A fresh Look at Historical Approaches to Machine Learning*. PhD thesis, MIT, 2002.

Dan Roth. Learning to resolve natural language ambiguities: a unified approach. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pages 806–813, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7.

R. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2001.

Vikas Sindhwani and Prem Melville. Document-word co-regularization for semi-supervised sentiment analysis. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 1025–1030, Washington, DC, USA, 2008. IEEE Computer Society.

Vikas Sindhwani, Jianying Hu, and Aleksandra Mojsilovic. Regularized co-clustering with dual supervision. In *Advances in Neural Information Processing Systems 21*, 2008.

J. Weston, A. Elisseeff, G. Bakir, and F. Sinz. The spider machine learning toolbox., 2005. URL `http://www.kyb.tuebingen.mpg.de/bs/people/spider/`.

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, second edition, 2005.

Jing Xiao, Lusheng Wang, Xiaowen Liu, and Tao Jiang. An efficient voting algorithm for finding additive biclusters with random background. *Journal of Computational Biology*, 15(10), 2008.

Appendix A. Base Level Models and Other Things We Tried

Random Forests: We trained random forests using the FEST package (Caruana et al., 2008). We varied the number of features considered at each split from $0.5 \cdot \sqrt{\#features}$ to $64 \cdot \sqrt{\#features}$ by factors of 2. For the smaller numbers we trained random forests of 500 trees, and went down to 300 trees as training became more computationally expensive at higher numbers of considered features. Since the code supported example weighting, we used a weight of 0.1 for the negative class on all problems, to account for class imbalance. We did not try to vary this parameter, or to run without example weighting. Random forests worked well, especially on appetency, where they had the best performance.

Boosted Decision Trees: The FEST package was also used to train boosted decision trees. We boosted decision trees 1, 2, 3, 4, 5, 7, 10 and 20 levels deep. We varied the number of rounds of boosting between 1 and 128. As with random forests, we used a weight of 0.1 on the negative examples. Boosted decision trees had the best performance on the up-selling problem, and good performance on the other problems as well. Boosting, however, clearly overfit on all three problems, with the best performance being obtained after less than 20 rounds. Also, boosting shallower trees performed better than boosting deeper trees, although there is a chance that if we had boosted even deeper trees for longer we would have obtained better performance. (See Figure 1).

Regularized Logistic Regression: For logistic regression we used the BBR package (Genkin et al., 2007). We used both L_1 and L_2 regularization, varying the regularization parameter from 10^{-3} to 100 by factors of 10. We also used the feature selection capability implemented in the BBR package and selected subsets of 100, 200, 500, 1000, 2000 and 5000 features using Pearson's Correlation. L_1 regularization worked better than L_2 on all problems. Feature selection provided another slight improvement in performance, made the results less sensitive to the regularization parameter, and reduced the gap between L_1 and L_2 regularization. Logistic regression also was a well performing technique, providing the top performance on the churn problem after the addition of the extra features meant to capture non-linearities in the data.

SVM: We trained linear SVMs using the SVMPerf (Joachims, 2005), LibLinear (Fan et al., 2008) and LibSVM (Chang and Lin, 2001) packages.[2] The regularization parameter was varied from 10^{-5} to 100 by factors of 10. Besides training regular SVMs that optimize hinge loss, we also directly optimized AUC with SVMPerf. Directly optimizing AUC did indeed yield significantly better AUC performance than optimizing hinge loss. It is interesting, however, that optimizing to cross-entropy (log-likelihood, log-loss) via logistic regression not only outperformed optimizing to hinge loss, but also slightly outperformed optimizing AUC directly.

We also tried training kernel SVMs using the LaSVM package (Bordes et al., 2005) but we were unable to get them to perform well so we abandoned them early on.

Regularized Least Squares: We trained linear Regularized Least Squares Classifiers (RLSC) that often perform competitively with linear Support Vector Machines and other regularized risk minimization methods on classification tasks (see e.g., (Rifkin, 2002)). The

2. SVMs proved to be a popular method in our team, with multiple team members training them using different packages.

training was done with efficient sparse matrix computations in Matlab using the conjugate gradient algorithm (with a tolerance of 1e-4 and 500 maximum iterations). To handle class imbalance, our loss terms measured squared error over positive and negative examples separately. We then conducted feature selection using mutual information between the features and the class variable. Mutual information (MI) was computed by discretizing continuous features into 20 bins. Our MI implementation was adapted from the Spider machine learning toolbox (Weston et al., 2005). We generated models with 1000 to 2000 features that showed improved performance.

Naive Bayes: We used a Naïve Bayes classifier with kernel density estimation for continuous variables (John and Langley, 1995). In our pilot experiments we found that kernel density estimation worked better than alternative supervised and unsupervised approaches to deal with the continuous variables in the data. We used an online implementation of the Naïve Bayes classifier, which makes it highly scalable. Only one pass over the data is required, and the classifier can be quickly updated one instance at a time. This makes it possible to process the full data set with minimal memory usage.

Tree Augmented Naive Bayes: We used the WEKA package (Witten and Frank, 2005) to train Tree Augmented Naive Bayes (TAN) models using the 50, 100 and 200 attributes with the highest information gain. The TAN models were learned using an entropy score to measure network quality, and applying a Markov blanket correction to the learnt structure. Additional improvement was obtained through bagging. Using the top 100 or 200 attributes, did not result in any improvement over those learned using just the top 50 attributes.

Sparse Network of Winnows: We used the Sparse Network of Winnows (SNoW) learning architecture introduced in (Roth, 1998). SNoW builds a sparse network of linear functions, and is specifically designed for domains with a large number of features that may be unknown a priori. It has been applied successfully to a variety of large-scale natural language and visual processing tasks. The key strength of SNoW comes from exploiting the fact that using the Winnow update rule the number of examples needed to learn a linear function grows linearly with the number of *relevant* features and only logarithmically with the total number of features (Littlestone, 1991; Kivinen and Warmuth, 1995). Winnow is known to learn any linear threshold function efficiently and is resilient to various kinds of noise. We trained SNoW with 50 learning iterations of Perceptron and Winnow, and combined the output of the two.

k-NN: With respect to k-NN we wrote C++ code optimized for fast working with sparse representations. To compute distances we used weighted Euclidian distance for continuous variables, and Hamming distance for categoricals. For the slow challenge we replaced the Hamming distance by the inverse of the second power of the frequency of the value of the feature. In the version used for the fast challenge we weighted the features by their mutual information with labels (3 different weight sets, one for each problem). For the slow challenge we used instead the AUC score obtained by using k-NN with only this column as an input. Those weights turned out to be better than mutual information, especially in case of churn. The final prediction was a distance-weighted median of the neighbors. The optimal number of neighbors turned out to be rather low, between 100 and 500.

Co-Clustering: We trained models using the graph-based co-clustering technique in (Sindhwani et al., 2008). The motivation behind trying this technique is two fold: 1) since the

features are extremely high dimensional, co-clustering methods could benefit from implicit dimensionality reduction, and 2) because these techniques naturally make use of un-labeled data in the clustering process, they could potentially take advantage of the test data that are available during training.

We performed column-wise shifting and rescaling of the data to ensure all feature values are non-negative (a requirement of bi-partite graph based co-clustering methods), and raw-normalization so that each row adds up to one. These two preprocessing steps proved critical to ensure a decent outcome from the graph-based co-clustering methods. We also experimented with column normalization, which did not seem to help in this case.

Since the original method described in (Sindhwani et al., 2008) is not scalable to the size of the data, we experimented with two scalable versions of the algorithm. We first used a transductive version of the algorithm which had been previously applied to document-word co-regularization (equation 6 in (Sindhwani and Melville, 2008)). We varied the parameters μ (weight of the graph-based co-clustering regularizer) and p (order of the Laplacian operator). We found that, for these problems, the results were not very sensitive to μ in the range of 0.01 to 100, and p in the range of 1 to 10.

The transductive version, however, produced inferior results to a well tuned supervised classifier. We thus decided to try another variation of the original algorithm which combines transductive learning with a linear model (equation 8 in (Sindhwani and Melville, 2008), with the last term dropped since there are no feature labels). While this lead to much more competitive results, the graph-based co-clustering techniques still failed to produce the benefits we had hoped for.

Missing Value Imputation: We have also tried imputing the missing values in the hope it will provide an additional benefit. For each of the numerical features with missing values, we trained a model to predict the respective feature from the rest of the attributes. To this end we used two learning techniques: least squares and k-NN. For k-NN we used k=500 and calculated distances between instances using only the top 1000 features ranked by their respective linear SVM weights. Both sets of imputed values seemed to yield only small improvements in performance, if any. In the end neither of them was extensively used.

PCA: Given the large dimension of the feature space, we explore reducing the dimension using principle component anlaysis (PCA). We use the package SVDpack (Berry, 1992) to reduce the dimension to 500 and run logistic regression, SVM and k-NN classifiers. The performance of these models, however, was mediocre at best.

Multi-Task Learning: We also explore the idea of multi-task learning, in which we assume the three tasks are not independent (which seems true for this particular application). One easy way to make use of the dependencies between tasks is to use the labels (or predicted labels) from other tasks as additional features and train another layer of classifiers. We explore this idea and find that using the true labels from other tasks are able to improve the performance by 3-5%, but there is no improvement when using the predicted labels.