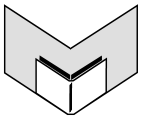
The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009)
Challenges in Machine Learning, Volume 3

The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009)

Challenges in Machine Learning, Volume 3

Gideon Dror, Marc Boullé, Isabelle Guyon,
Vincent Lemaire, and David Vogel, editors

Nicola Talbot, production editor



Microtome Publishing
Brookline, Massachusetts
www.mtome.com

The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009)
Challenges in Machine Learning, Volume 3

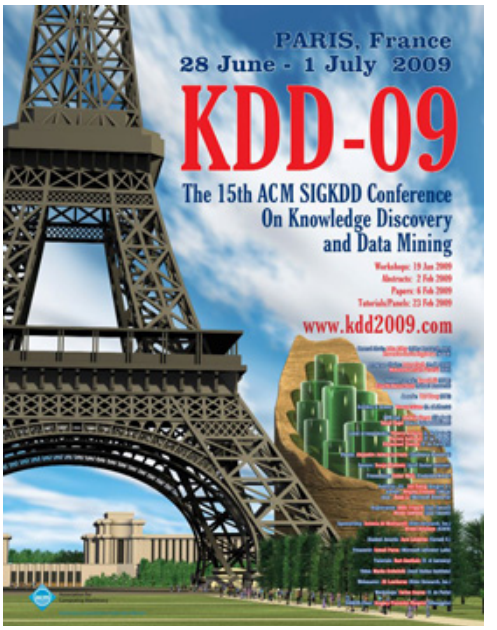
Gideon Dror, Marc Boullé, Isabelle Guyon,
Vincent Lemaire, and David Vogel, editors

Nicola Talbot, production editor

Collection copyright © 2011 Microtome Publishing, Brookline, Massachusetts, USA.
Copyright of individual articles remains with their respective authors.

ISBN-13: 978-0-9719777-3-0

KDD Cup 2009 Competition
⟨http://www.kddcup-orange.com/⟩



Foreword

Data mining (and knowledge discovery) is the art and science of finding useful and valid patterns in data.

Humans have a built-in ability to detect patterns. During our long evolution, those who were better able to figure out when an animal is dangerous, what food is edible, and so on were more successful in surviving and passing on their genes.

Unfortunately, humans have not evolved a good ability to separate valid patterns from spurious observations when dealing with large data. We tend to confuse coincidences with causality, and pay more attention to some coincidences then others. As a result, we are subject to superstitions (different in every culture), astrology, belief in miracle cures, climate change denials, and similar ill effects. Even I, a card-carrying data miner, succumb to consoling superstitions and jinxes, especially when the Red Sox play baseball in the World Series.

In the grand scale of things, the mission of knowledge discovery is to help scientists, biologists, doctors, politicians, intelligence agencies, climate change debaters, and all people to make correct inferences from data. Good data mining and knowledge discovery can have profound implications for medicine, politics, security, climate policy, and many other fields.

First, data miners need good examples of real world data on which to learn how to extract a few valid patterns from many false positives and avoid overfitting the data.

In the 1990s there were test datasets available in the UCI Machine Learning Repository, but most of them were quite small, toy examples. They did not have the complexity, large numbers of records, hundreds of attributes, noisy data, and "false predictors" (fields that "predict" target variables in the data, but after the fact, and cannot be used for real predictions).

There was a real need to have a real-world test set which was publicly available.

In 1997 as the head of KDD (Knowledge Discovery in Data) steering committee, I helped to start the first such test — the first KDD Cup, ably organized by Ismail Parsa. Parsa was able to get data from a charity on past responders to fund-raising (with appropriate anonymization and removal of target variable). The goal was to predict the most likely responders for a new campaign. The task was quite difficult since the response rate was only 1.4%, and there were over 300 variables, most of them quite useless.

The first KDD Cup had 45 participants of which only 16 turned in their predictions, and only a few of them had results significantly better than random. One of the winners was a large company that used a very complex method, while the second winner used a simple Naive Bayes algorithm that carefully avoided overfitting.

One submitted prediction from a well-known company was actually worse than random — probably as a result of poor data processing.

The following KDD Cups have covered such diverse topics as network intrusion detection, online retailer clickstream analysis, molecular biology, biomed documents, particle physics and protein homology prediction, search query categorization, medical image recognition, and Netflix movie recommendations.

The KDD Cup has achieved the status of a kind of World Championship of data mining and spawned many other data mining competitions, most notably a recent $1 million Netflix prize.

Having all the past data and results available (see KDD Cup Center www.sigkdd.org/kddcup/index.php) helped to raise the standards and significantly improved the average quality of predictions, compared to the first KDD Cup.

The 2009 KDD Cup centered around modeling the behaviour of telecom customers, including attrition, cross-sell, and up-sell. The data and prizes were provided by Orange, a French telecom company. One novel aspect was fast scoring — coming up with the results in 5 days of less.

The KDD Cup and associated KDD-09 workshop were ably organized by Isabelle Guyon, Vincent Lemaire, Marc Boullé, Gideon Dror, and David Vogel. Large prizes and effective advertising helped attract over 450 participants from 46 countries — the largest KDD Cup participation so far.

This book presents the papers from KDD Cup 2009. One of the principal technical conclusions from this KDD Cup (as well as from the Netflix prize) was that ensemble methods are very effective.

The data and the platform of the challenge remain available for research and educational purposes at http://www.kddcup-orange.com/.

The results and presentations in this book will undoubtedly help researchers and industry data miners to come up with better solutions under real constraints.

Gregory Piatetsky-Shapiro
www.kdnuggets.com/gps.html

Preface

The annual ACM SIGKDD conference on Knowledge Discovery and Data Mining (KDD) is dedicated to facilitating interactions between data mining researchers and practitioners from academia, industry, and government. The 15th edition took place in Paris, France, and hosted the KDD Cup competition, which attracted a very large number of participants, three times more than any KDD Cup in the past. The organizing team of the KDD Cup 2009 is pleased to welcome you to this edition of the proceeding of the workshop, where the challenge results were presented and analyzed.

We organized the KDD cup 2009 around a marketing problem with the goal of identifying data mining techniques capable of rapidly building predictive models and scoring new entries on a large database. Customer Relationship Management (CRM) is a key element of modern marketing strategies. The KDD Cup 2009 offered the opportunity to work on large marketing databases from the French Telecom company Orange to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling). The most practical way to build knowledge on customers in a CRM system is to produce scores. A score (the output of a model) is an evaluation for all target variables to explain (i.e., churn, appetency or up-selling). Tools producing scores provide quantifiable information on a given population. The score is computed using customer records represented by a number of variables or features. Scores are then used by the information system (IS), for example, to personalize the customer relationship. Building tens to hundreds CRM scores can be a key element in a marketing application. In this context, the automation of the data preparation and modeling steps of the data mining process is a challenging issue. The three CRM tasks of the KDD Cup 2009 related to the data donated by Orange encompass several scientifically or technically challenges: large datasets with 100,000 samples and 15,000 variables, noisy data, mixed types with numerical and categorical variables containing up to thousands of values, many missing values, unbalanced classes. In practice, scoring methods have to fulfill several objectives, such as full automation, effectiveness, time efficiency both for training and deployment, understandability of the models. Full automation is necessary in order to meet the increasing demand for building numerous scores. Effectiveness involves the accuracy of the scores, and has a direct impact on the marketing payoff: the better customers are targeted, the higher the response rate. The standard scientific accuracy indicator for scores is the area under the ROC curve (AUC), evaluated on test data. In practice, the marketing quality indicator is the payoff of a campaign which includes modeling costs, campaign costs (by mail, email, phone) and revenue related to the response. Training time efficiency allows to frequently update the scores, and involves modeling with train datasets up to hundred of thousands of samples and thousands of variables. Deployment time efficiency permits the scoring of tens of millions of customers in order to select the most responsive customers. Finally, the understandability of the models provides useful information to marketing teams. In a challenge, the tasks must be approachable and we chose to focus on effectiveness given training time constraints. The performance indicator is the AUC on test data. The participants had one month to get familiar with the data tables without the target values, and then fives days to submit their test results once the training target values were made available. The participant exploited a wide variety of preprocessing, feature selection, classification, model selection and ensemble

methods, providing a large and significant evaluation of the techniques effective for problems with large numbers of samples and variables, mixed types of variables, lots of missing values and unbalanced classes.

This volume gathers the material of the challenge on Fast Scoring on a Large Marketing Database organized for the conference on Knowledge Discovery and Data Mining, June 28, 2009 in Paris. The book contains a collection of papers first published in JMLR W&CP, including a paper summarizing the results of the challenge and contributions of the top ranking entrants. The book is complemented by a web site from which the datasets can be downloaded and post-challenge submissions can be made to benchmark new algorithms, see `http://www.kddcup-orange.com/`

December 2009

The KDD challenge team :

Gideon Dror
Academic College of Tel-Aviv-Yaffo, Tel Aviv, Israel
`gideon@mta.ac.il`

Marc Boullé
Orange Labs, Lannion, France
`marc.boulle@orange-ftgroup.com`

Isabelle Guyon
Clopinet, California, USA
`isabelle@clopinet.com`

Vincent Lemaire
Orange Labs, Lannion, France
`vincent.lemaire@orange-ftgroup.com`

David Vogel
Data Mining Solutions, Orlando, Florida, USA
`dvogel@dataminingsolutions.net`

Table of Contents

Papers published in JMLR W&CP

TABLE OF CONTENTS

Analysis of the KDD Cup 2009: Fast Scoring on a Large Orange Customer Database

Isabelle Guyon ISABELLE@CLOPINET.COM
Clopinet, Berkeley, CA 94798, USA

Vincent Lemaire VINCENT.LEMAIRE@ORANGE-FTGROUP.COM
Orange Labs, Lannion, 22300, France

Marc Boullé MARC.BOULLE@ORANGE-FTGROUP.COM
Orange Labs, Lannion, 22300, France

Gideon Dror GIDEON@MTA.AC.IL
Academic College of Tel-Aviv-Yaffo, Tel Aviv 61083, Israel

David Vogel DVOGEL@DATAMININGSOLUTIONS.NET
Data Mining Solutions, Orlando, Florida, USA

Editor: Neil Lawrence

Abstract

We organized the KDD cup 2009 around a marketing problem with the goal of identifying data mining techniques capable of rapidly building predictive models and scoring new entries on a large database. Customer Relationship Management (CRM) is a key element of modern marketing strategies. The KDD Cup 2009 offered the opportunity to work on large marketing databases from the French Telecom company Orange to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling). The challenge started on March 10, 2009 and ended on May 11, 2009. This challenge attracted over 450 participants from 46 countries. We attribute the popularity of the challenge to several factors: (1) A generic problem relevant to the Industry (a classification problem), but presenting a number of scientific and technical challenges of practical interest including: a large number of training examples (50,000) with a large number of missing values (about 60%) and a large number of features (15,000), unbalanced class proportions (fewer than 10% of the examples of the positive class), noisy data, presence of categorical variables with many different values. (2) Prizes (Orange offered 10,000 Euros in prizes). (3) A well designed protocol and web site (we benefitted from past experience). (4) An effective advertising campaign using mailings and a teleconference to answer potential participants questions. The results of the challenge were discussed at the KDD conference (June 28, 2009). The principal conclusions are that ensemble methods are very effective and that ensemble of decision trees offer off-the-shelf solutions to problems with large numbers of samples and attributes, mixed types of variables, and lots of missing values. The data and the platform of the challenge remain available for research and educational purposes at `http://www.kddcup-orange.com/`.

Keywords: challenge, classification, customer management, fast scoring

©2009 Isabelle Guyon, Vincent Lemaire, Marc Boullé, Gideon Dror, David Vogel

1. Introduction

Customer Relationship Management (CRM) is a key element of modern marketing strategies. The KDD Cup 2009 offered the opportunity to work on large marketing databases from the French Telecom company Orange to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling).

The most practical way to build knowledge on customers in a CRM system is to produce scores. A score (the output of a model) is an evaluation for all target variables to explain (*i.e.,* churn, appetency or up-selling). Tools producing scores provide quantifiable information on a given population. The score is computed using customer records represented by a number of variables or features. Scores are then used by the information system (IS), for example, to personalize the customer relationship. The rapid and robust detection of the most predictive variables can be a key factor in a marketing application. An industrial customer analysis platform developed at Orange Labs, capable of building predictive models for datasets having a very large number of input variables (thousands) and instances (hundreds of thousands), is currently in use by Orange marketing. A key requirement is the complete automation of the whole process. The system extracts a large number of features from a relational database, selects a subset of informative variables and instances, and efficiently builds in a few hours an accurate classifier. When the models are deployed, the platform exploits sophisticated indexing structures and parallelization in order to compute the scores of millions of customers, using the best representation.

The challenge was to beat the in-house system developed by Orange Labs. It was an opportunity for participants to prove that they could handle a very large database, including heterogeneous noisy data (numerical and categorical variables), and unbalanced class distributions. Time efficiency is often a crucial point. Therefore part of the competition was time-constrained to test the ability of the participants to deliver solutions quickly. The fast track of the challenge lasted five days only. To encourage participation, the slow track of the challenge allowed participants to continue working on the problem for an additional month. A smaller database was also provided to allow participants with limited computer resources to enter the challenge.

2. Background and motivations

This challenge uses important marketing problems to benchmark classification methods in a setting, which is typical of large-scale industrial applications. A large database was made available by the French Telecom company, Orange with tens of thousands of examples and variables. This dataset is unusual in that it has a large number of variables making the problem particularly challenging to many state-of-the-art machine learning algorithms. The challenge participants were provided with masked customer records and their goal was to predict whether a customer will switch provider (churn), buy the main service (appetency) and/or buy additional extras (up-selling), hence solving three binary classification problems. Churn is the propensity of customers to switch between service providers, appetency is the propensity of customers to buy a service, and up-selling is the success in selling additional good or services to make a sale more profitable. Although the technical difficulty of scaling up existing algorithms is the main emphasis of the challenge, the dataset proposed offers a variety of other difficulties: heterogeneous data (numerical and categorical variables), noisy data, unbalanced distributions of predictive variables, sparse target values (only 1 to 7 percent of the examples examples belong to the positive class) and many missing values.

3. Evaluation

There is value in a CRM system to evaluate the propensity of customers to buy. Therefore, tools producing scores are more usable that tools producing binary classification results. The participants were asked to provide a score (a discriminant value or a posterior probability $P(Y = 1|X)$), and they were judged by the area under the ROC curve (AUC). The AUC is the area under the curve plotting sensitivity *vs.* (1− specificity) when the threshold θ is varied (or equivalently the area under the curve plotting sensitivity *vs.* specificity). We call "sensitivity" the error rate of the positive class and "specificity" the error rate of the negative class. The AUC is a standard metric in classification. There are several ways of estimating error bars for the AUC. We used a simple heuristic, which gives us approximate error bars, and is fast and easy to implement: we find on the AUC curve the point corresponding to the largest balanced accuracy BAC = 0.5 (sensitivity + specificity). We then estimate the standard deviation of the BAC as:

$$\sigma = \frac{1}{2}\sqrt{\frac{p_+(1-p_+)}{m_+} + \frac{p_-(1-p_-)}{m_-}}\ , \tag{1}$$

where m_+ is the number of examples of the positive class, m_- is the number of examples of the negative class, and p_+ and p_- are the probabilities of error on examples of the positive and negative class, approximated by their empirical estimates, the sensitivity and the specificity (Guyon et al., 2006).

The fraction of positive/negative examples posed a challenge to the participants, yet it was sufficient to ensure robust prediction performances (as verified in the beta tests). The database consisted of 100,000 instances, split randomly into equally sized train and test sets:

- **Churn problem:** 7.3% positive instances (3672/50000 on train).
- **Appetency problem:** 1.8% positive instances (890/50000 on train).
- **Up-selling problem:** 7.4% positive instances (3682/50000 on train).

On-line feed-back on AUC performance was provided to the participants who made correctly formatted submissions, using only 10% of the test set. There was no limitation on the number of submissions, but only the last submission on the test set (for each task) was taken into account for the final ranking.

The score used for the final ranking was the average of the scores on the three tasks (churn, appetency, and up-selling).

4. Data

Orange (the French Telecom company) made available a large dataset of customer data, each consisting of:

- **Training :** 50,000 instances including 15,000 inputs variables, and the target value.
- **Test :** 50,000 instances including 15,000 inputs variables.

There were three binary target variables (corresponding to churn, appetency, and up-selling). The distribution within the training and test examples was the same (no violation of the i.i.d. assumption - independently and identically distributed). To encourage participation, an easier task was also built from a reshuffled version of the datasets with only 230 variables. Hence, two versions were made available ("small" with 230 variables, and "large" with 15,000 variables).

The participants could enter results on either or both versions, which corresponded to the same data entries, the 230 variables of the small version being just a subset of the 15,000 variables of the large version. Both training and test data were available from the start of the challenge, without the true target labels. For practice purposes, "toy" training labels were available together with the training data from the onset of the challenge in the fast track. The results on toy targets did not count for the final evaluation. The real training labels of the tasks "churn", "appetency", and "up-selling", were later made available for download, half-way through the challenge.

The database of the large challenge was provided in several chunks to be downloaded more easily and we provided several data mirrors to avoid data download congestion. The data were made publicly available through the website of the challenge `http://www.kddcup-orange.com/`, with no restriction of confidentiality. They are still available to download for benchmark purpose. To protect the privacy of the customers whose records were used, the data were anonymized by replacing actual text or labels by meaningless codes and not revealing the meaning of the variables.

EXTRACTION AND PREPARATION OF THE CHALLENGE DATA:

The Orange in-house customer analysis platform is devoted to industrializing the data mining process for marketing purpose. Its fully automated data processing machinery includes: data preparation, model building, and model deployment. The data preparation module was isolated and used to format data for the purpose of the challenge and facilitate the task of the participants. Orange customer data are initially available in a relational datamart under a star schema. The platform uses a feature construction language, dedicated to the marketing domain, to build tens of thousands of features to create a rich data representation space.

For the challenge, a datamart of about one million of customers was used, with about ten tables and hundreds of fields. The first step was to resample the dataset, to obtain 100,000 instances with less unbalanced target distributions. For practical reasons (the challenge participants had to download the data), the same data sample was used for the three marketing tasks. In a second step, the feature construction language was used to generate 20,000 features and obtain a tabular representation. After discarding constant features and removing customer identifiers, we narrowed down the feature set to 15,000 variables (including 260 categorical variables). In a third step, for privacy reasons, data was anonymized, discarding variables names, randomizing the order of the variables, multiplying each continuous variable by a random factor and recoding categorical variable with randomly generated category names. Finally, the data sample was split randomly into equally sized train and test sets. A random subset of 10% of the test set was designated to provide immediate performance feed-back.

5. Beta tests

The website of the challenge `http://www.kddcup-orange.com/` was thoroughly tested by the KDD cup chairs and volunteers. The datasets were downloaded and checked. Baseline methods were tried to verify the feasibility of the task. A Matlab® version of the data was made available and sample code were provided to format the results. A sample submission of random results was given as example and submitted to the website. The results of the Naïve Bayes method were also uploaded to the website to provide baseline results.

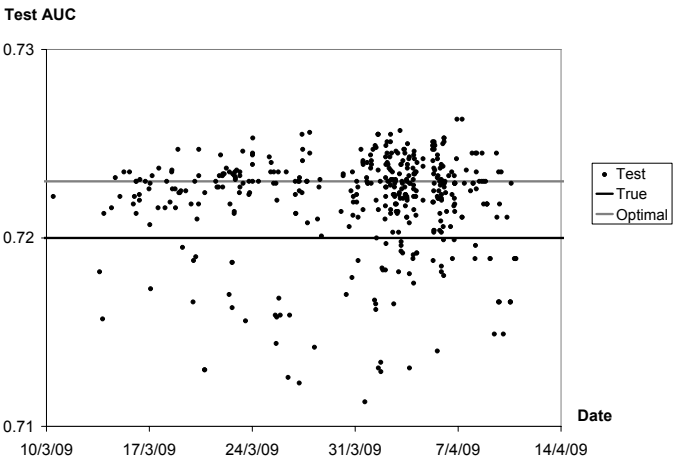


Figure 1: Toy problem test results.

TOY PROBLEM:

The Toy problem on the LARGE dataset consisted of one single predictive continuous variable (V5963) uniformly distributed on the interval $[0, 2.0]$. The target value was obtained by thresholding V5963 at 1.6 and adding 20% noise. Hence for 80% of the instances, lying in interval $[0, 1.6]$, the fraction of positive examples is 20%; for the remaining 20% lying in interval $]1.6, 2.0]$, the fraction of positive examples is 80%. The expected value of the AUC (called "true AUC") can easily be computed[1]. Its value is approximately 0.7206. Because of the variance in the sampling process, the AUC effectively computed using the optimal decision rule (called "optimal AUC") is 0.7196 for the training set and a 0.7230 for the test set. Interestingly, as shown in Figure 1, the optimal solution was outperformed by many participants, up to 0.7263. This illustrates the problem of multiple testing and shows how the best test performance overestimates both the expected value of the AUC and the performance of the optimal decision rule, increasingly with the number of challenge submissions.

BASIC NAÏVE BAYES CLASSIFIER:

The basic Naïve Bayes classifier (see *e.g.,* Mitchell, 1997) makes simple independence assumptions between features and votes among features with a voting score capturing the correlation of the feature to the target. No feature selection is performed and there are no hyper-parameters to adjust.

For the LARGE dataset, the overall score of the basic Naïve Bayes classifier is 0.6711, with the following results on the test set:

- **Churn problem :** AUC = 0.6468;

1. If we call T the total number of examples, the (expected value of) the total number of examples of the positive class P is the sum of the number of positive examples in the first and the second intervals, *i.e.,* $P = (0.2 \times 0.8 + 0.8 \times 0.2)\ T = 0.32\ T$. Similarly, the total number of negative examples is $N = (0.8 \times 0.8 + 0.2 \times 0.2)\ T = 0.68\ T$. If we use the optimal decision rule (a threshold on V5963 at 1.6) the number of true positive examples is the sum of the number of true positive examples in the two intervals, *i.e.,* $TP = 0 + (0.2 \times 0.8)\ T = 0.16\ T$. Similarly, the number of true negative examples is $TN = (0.8 \times 0.8)\ T = 0.64\ T$. Hence, the true positive rate is $TPR = TP/P = 0.16/0.32 = 0.5$ and the true negative rate is $TNR = TN/N = 0.64/0.68 \simeq 0.9412$. The balanced accuracy (or the AUC because $BAC = AUC$ in this case) is therefore: $BAC = 0.5\ (TPR + TNR) = 0.5\ (0.5 + 0.9412) = 0.7206$.

- **Appetency problem :** AUC = 0.6453;
- **Up-selling problem :** AUC=0.7211;

As per the rules of the challenge, the participants had to outperform the basic Naïve Bayes classifier to qualify for prizes.

ORANGE IN-HOUSE CLASSIFIER:

The Orange in-house classifier is an extension of the Naïve Bayes classifier, called "Selective Naïve Bayes classifier" (Boullé, 2007). It includes an optimized preprocessing, variable selection, and model averaging. It significantly outperforms the basic Naïve Bayes classifier performance, which was provided to the participants as baseline, and it is computationally efficient: The results were obtained after 3 hours using a standard laptop, considering the three tasks as three different problems. The models were obtained by applying the training process Khiops® only once since the system has no hyper-parameter to adjust. The results of the in-house system were not revealed until the end of the challenge. An implementation of the method is available as shareware from `http://www.khiops.com`; some participants downloaded it and used it.

The requirements placed on the in-house system are to obtain a high classification accuracy, under the following constraints:

- Fully automatic: absolutely no hyper-parameter setting, since hundred of models need to be trained each month.
- Fast to train: the three challenge marketing problems were trained in less than 3 hours on a mono-processor laptop with 2 Go RAM.
- Efficient after deployment: models need to process rapidly up to ten million instances.
- Interpretable: selected predictive variables must provide insight.

However, for the challenge, the participants were not placed under all these constraints for practical reasons: it would have been both too constraining for the participants and too difficult to enforce for the organizers. The challenge focused on maximizing accuracy under time constraints.

For the LARGE dataset, the overall score of the Orange in-house classifier is 0.8311, with the following results on the test dataset:

- **Churn problem :** AUC = 0.7435;
- **Appetency problem :** AUC = 0.8522;
- **Up-selling problem :** AUC=0.8975;

The challenge was to beat these results, but the minimum requirement to win prizes was only to outperform the basic Naïve Bayes classifier.

6. Challenge schedule and protocol

The key elements of our design were:

- To make available the training and test data three weeks before the start of the "fast challenge" to allow participants to download the large volume of data, read it and preprocess it without the training labels.

- To make available "toy" training labels during that period so participants could finalize their methodology and practice using the on-line submission system.
- To put participants under time pressure once the training labels were released (produce results in five days) to test their ability to produce results in a timely manner.
- To continue the challenge beyond this first milestone for another month (slow challenge) to give the opportunity to participants with less computational resources to enter the challenge.
- To provide a down-sized version of the dataset for the slow challenge providing an opportunity for participants with yet less computational resources to enter the challenge.
- To provide large prizes to encourage participation (10,000 Euros donated by Orange), without any strings attached (no legal constraint or commitment to release code or methods to download data or participate).

The competition rules are summarized below are inspired from previous challenges we organized (Clopinet):

1. **Conditions of participation:** Anybody who complied with the rules of the challenge (KDD cup 2009) was welcome to participate. Only the organizers listed on the *Credits* page were excluded from participating. The participants were not required to attend the KDD cup 2009 workshop and the workshop was open to anyone.
2. **Anonymity:** All entrants had to identify themselves by registering on the KDD cup 2009 website. However, they could elect to remain anonymous during the development period. To be eligible for prizes the had to publicly reveal their identity. Teams had to declare a team leader and register their members. No individual could be part of two or more teams.
3. **Data:** The datasets were available for download from the *Dataset* page to registered participants. The data were available in several archives to facilitate downloading.
4. **Challenge duration and tracks:** The challenge started March 10, 2009 and ended May 11, 2009. There were two challenge tracks:
   - **FAST (large) challenge:** Results submitted on the LARGE dataset within five days of the release of the real training labels counted towards the fast challenge.
   - **SLOW challenge:** Results on the small dataset and results on the large dataset not qualifying for the fast challenge, submitted before the KDD cup 2009 deadline May 11, 2009, counted toward the SLOW challenge.

   If more than one submission was made in either track and with either dataset, the last submission before the track deadline was taken into account to determine the ranking of participants and attribute the prizes.
5. **On-line feed-back:** During the challenge, the training set performances were available on the *Result* page as well as partial information on test set performances: The test set performances on the "toy problem" and performances on a fixed 10% subset of the test examples for the real tasks (churn, appetency and up-selling). After the challenge was over, the performances on the whole test set were calculated and substituted in the result tables.

6. **Submission method:** The method of submission was via the form on the *Submission* page, following a designated format. Results on the "toy problem" did not count as part of the competition. Multiple submissions were allowed, but limited to 5 submissions per day to avoid congestion. For the final entry in the slow track, the participants could submit results on either (or both) small and large datasets in the same archive.

7. **Evaluation and ranking:** For each entrant, only the last valid entry, as defined in the *Instructions* counted towards determining the winner in each track (fast and slow). We limited each participating person to a single final entry in each track. Valid entries had to include results on all three real tasks. Prizes could be attributed only to entries performing better than the baseline method (Naïve Bayes). The results of the baseline method were provided to the participants.

8. **Reproducibility:** Participation was not conditioned on delivering code nor publishing methods. However, we asked the participants to voluntarily fill out a fact sheet about their methods and contribute papers to the proceedings.

The full rules are available from the website of the challenge `http://www.kddcup-orange.com/`. The rules were designed to attract a large number of participants and were successful in that respect: Many participants did not participate in the fast challenge on the large dataset, but entered in the slow track, either on the small or the large dataset (or both). There was one minor design mistake: the small dataset was derived from the same data as the large one and, despite our efforts to disguise the identity of the features, it was possible for some entrants to match the features and entries in the small and large dataset. This provided a small advantage, *in the slow track only*, to the teams who did that data "unscrambling": they could get feed-back on 20% of the data rather than 10%.

The schedule of the challenge was as follows (Dates in 2009):

- **March 10** - Start of the FAST large challenge. Data tables without target values were made available for the large dataset. Toy training target values were made available for practice purpose. Objective: participants can download data, ask questions, finalize their methodology, try the submission process.

- **April 6** - Training target values were made available for the large dataset for the real problems (churn, appetency, and upselling). Feed-back: results on 10% of the test set available on-line when submissions are made.

- **April 10** - Deadline for the FAST large challenge. Submissions had to be received before midnight, time zone of the challenge web server.

- **April 11** - Data tables and training target values were made available for the small dataset. The challenge continued for the large dataset in the slow track.

- **May 11** - Deadline for the SLOW challenge (small and large datasets). Submissions had to be be received before midnight, time zone of the challenge web server.

7. Results

The 2009 KDD Cup attracted 1299 teams from 46 different countries. From those teams, 7865 valid entries were submitted by 453 different teams. The participation was more than three times greater than any KDD Cup in the past. Figure 2 represents the KDD Cup participation by year. A large participation was a key element to validate the results and for Orange to have a ranking of its in-house system; the challenge was very successful in that respect.

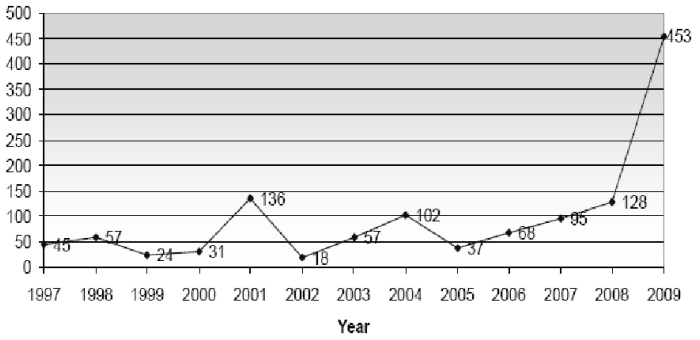


Figure 2: KDD Cup Participation by year (number of teams).

Table 1: **Winning entries.**

Prize	Team	Country	Fast track		Slow track	
			Rank	Score	Rank	Score
1	IBM Research	USA	1	0.8493	1	0.8521
2	ID Analytics	USA	2	0.8448	3	0.8479
3	David Slate & Peter Frey	USA	3	0.8443	8	0.8443
1	University of Melbourne	Australia	27	0.8250	2	0.8484
2	Financial Engineering Group, Inc.	Japan	4	0.8443	4	0.8477
3	National Taiwan University	Taiwan	20	0.8332	5	0.8461

7.1 Winners

The overall winner is the IBM Research team (IBM Research, 2009) who ranked first in both tracks. Six prizes were donated by Orange to top ranking participants in the fast and the slow tracks (see Table 1). As per the rules of the challenge, the same team could not earn two prizes. If the ranking of a team entitled it to two prizes, it received the best of the two and the next best ranking team received the other prize.

All the winning teams scored best on the large dataset (and most participants obtained better results on the large dataset then on the small dataset). IBM Research, ID Analytics, and National Taiwan University (NTU) "unscrambled" the small dataset. This may have provided an advantage only to NTU since "unscrambling" affected only the slow track and the two other teams won prizes in the fast track. We briefly comment on the results of the winners.

FAST TRACK:

- **IBM Research:** The winning entry (IBM Research, 2009) consisted of an ensemble of a wide variety of classifiers, following (Caruana and Niculescu-Mizil, 2004; Caruana et al., 2006). Effort was put into coding (most frequent values coded with binary features, missing values replaced by mean, extra features constructed, etc.)

- **ID Analytics, Inc.:** One of the only teams to use a wrapper feature selection strategy, following a filter (Xie et al., 2009). The classifier was built from the commercial TreeNet software by Salford Systems: an additive boosting decision tree technology. Bagging was also used to gain additional robustness.

- **David Slate & Peter Frey (Old dogs with new tricks):** After a simple preprocessing (consisting in grouping of modalities or discretizing) and filter feature selection, this team used ensembles of decision trees, similar to Random Forests (Breiman, 2001).

SLOW TRACK:

- **University of Melbourne:** This team used for feature selection a cross-validation method targeting the AUC and, for classification, boosting with classification trees and shrinkage, using a Bernoulli loss (Miller et al., 2009).

- **Financial Engineering Group, Inc.:** Few details were released by the team about their methods. They used grouping of modalities and a filter feature selection method using the AIC criterion (Akaike, 1973). Classification was based on gradient tree-classifier boosting (Friedman, 2000).

- **National Taiwan University:** The team averaged the performances of three classifiers (Lo et al., 2009): (1) The solution of the joint multiclass problem with an L1-regularized maximum entropy model. (2) AdaBoost with tree-based weak learners (Freund and Schapire, 1996). (3) Selective Naïve Bayes (Boullé, 2007), which is the in-house classifier of Orange (see Section 5).

7.2 Performance statistics

We now turn to the statistical analysis of the results of the participants. The main statistics are summarized in Table 2.

In the figures of this section, we use the following color code:

1. **Black**: **Submissions received**.
2. **Blue**: **Overall best submissions received**. Referred to as $TestAUC^{**}$.
3. **Red**: **Baseline result**, obtained with the basic Naïve Bayes classifier or NB, provided by the organizers (see Section 5). The organizers consider that this result is easy to improve. They imposed that the participants would outperform this result to win prizes to avoid that a random submission would win a prize.
4. **Green**: **Orange system result**, obtained by the in-house Orange system with the Selective Naïve Bayes classifier or SNB (see Section 5).

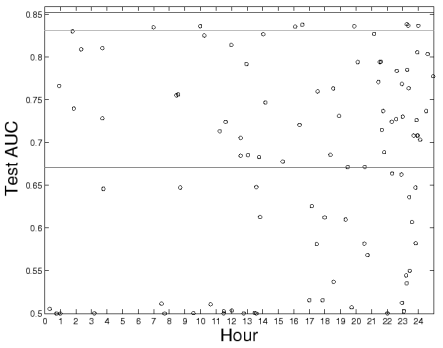
PROGRESS IN PERFORMANCE

Figure 3.a presents the results of the first day. A good result, better than the baseline result, is obtained **after one hour** and the in-house system is slightly outperformed **after seven hours**. The improvement during the first day of the competition, after the first 7 hours, is small: from 0.8347 to 0.8385.

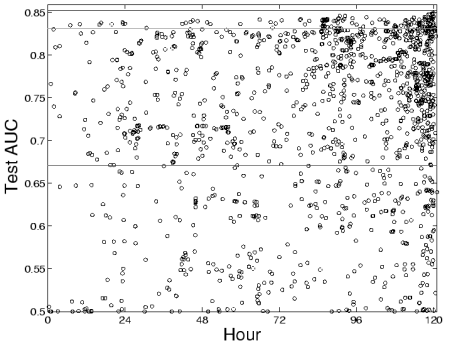
Figure 3.b presents the results over the first 5 days (FAST challenge). The performance progresses from 0.8385 to 0.8493. The rush of submissions before the deadline is clearly observable. Considering only the submission with Test AUC > 0.5 in the first 5 days, 30% of the submissions had worse results than the baseline (basic Naïve Bayes) and 91% had worse results than the in-house system (AUC=0.8311). **Only and 9% of the submissions had better results than the in-house system**.

Table 2: **Best results and baselines.** The first four lines show the best score $TAUC^*$ (averaged over the three tasks), over increasing periods of time $[0:t]$. For comparison we give the results of the basic Naïve Bayes classifier (NB) and the in-house Orange system (SNB). The best overall performance is $TAUC^{**} = TAUC^*(36d)$. The relative performance difference $\delta^* = (TAUC^{**} - TAUC)/TAUC^{**}$ is given in parenthesis (in percentage). The two last lines represent the relative performance difference $\delta = (TAUC^*(t) - TAUC)/TAUC^*(t)$ for the two reference results.

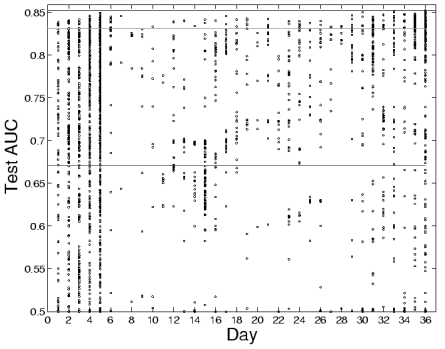
TAUC (δ^*%)	*TAUC** 12*h*	*TAUC** 24*h*	*TAUC** 5*d*	*TAUC***	*TAUC NB*	*TAUC SNB*
Churn	0.7467 (2.40)	0.7467 (2.40)	0.7611 (0.52)	0.7651 (0)	0.6468 (15.46)	0.7435 (2.82)
Appetency	0.8661 (2.17)	0.8714 (1.57)	0.8830 (0.26)	0.8853 (0)	0.6453 (27.11)	0.8522 (3.74)
Up-selling	0.9011 (0.89)	0.9011 (0.89)	0.9057 (0.38)	0.9092 (0)	0.7211 (20.69)	0.8975 (1.29)
Average	0.8380 (1.65)	0.8385 (1.60)	0.8493 (0.33)	0.8521 (0)	0.6711 (21.24)	0.8311 (2.46)
δ *NB* %	19.92	19.96	20.98	21.24	-	-
δ *SNB* %	0.82	0.88	2.14	2.46	-	-



(a) *Test AUC - hour ∈ [0:24]*



(b) *Test AUC - day ∈ [1:5]*



(c) *Test AUC - day ∈ [0:36]*

Figure 3: **Participant results over time.** "Test AUC" is the average AUC on the test set for the three problems. Each point represents an entry. The horizontal bars represent: basic Naïve Bayes, selective Naïve Bayes, and best participant entry.

Figures 3.a and 3.b show that good results were obtained already the first day and only small improvements were made later. These results, and an examination of the fact sheets of the challenge filled out by the participants, reveal that:

- There are available methods, which can process fast large databases using today's available hardware in both academia and industry.
- Several teams were capable of adapting their methods to meet the requirements of the challenge and reach quickly good performances, yet the bulk of the participants did not.
- The protocol of the challenge was well designed: the month given to download the data and play with the submission protocol (using the toy problem) allowed us to monitor progress in performance, not the time required to get ready for the challenge.

This last point was important for Orange to assess the time taken for generating state-of-the-art models, since speed of model generation is a key requirement in such applications. The fact that performances do not significantly improve after a few hours is further confirmed in Figure 3.c: very small improvements (from 0.8493 to 0.8521) were made after the 5^{th} day (SLOW challenge)[2].

Rapidity of model building

Figure 4.d gives a comparison between the submissions received and the best overall result over increasing periods of time: 12 hours, one day, 5 days, and 36 days. We compute the relative performance difference

$$\delta^* = (TestAUC^{**} - TestAUC)/TestAUC^{**} , \tag{2}$$

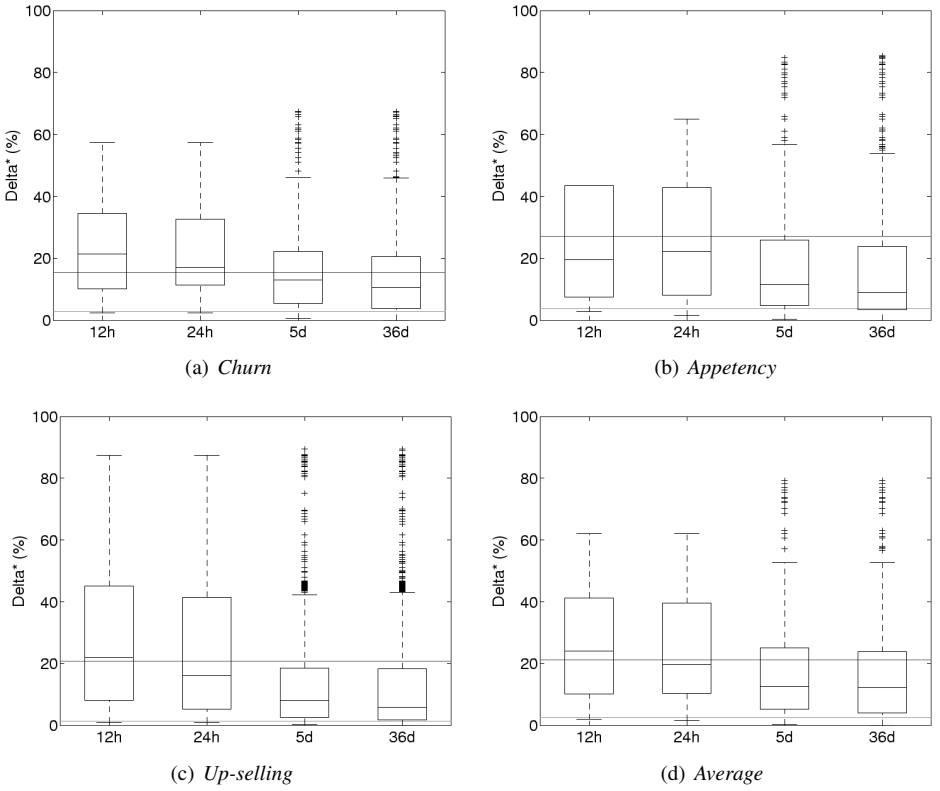
where $TestAUC^{**}$ is the best overall result. The values of δ^* for the best performing classifier in each interval and for the reference results are found in Table 2. The following observations can be made:

- there is a wide spread of results;
- the median result improves significantly over time, showing that it is worth continuing the challenge to give to participants an opportunity of learning how to solve the problem (the median beats the baseline on all tasks after 5 days but keeps improving);
- but the best results do not improve a lot after the first day;
- and the distribution after 5 days is not very different from that after 36 days.

Table 2 reveals that, at the end of the challenge, for the average score, the relative performance difference between the baseline model (basic Naïve Bayes) and the best model is over 20%, but only 2.46% for SNB. For the best ranking classifier, only 0.33% was gained between the fifth day (FAST challenge) and the last day of the challenge (SLOW challenge). After just one day, the best ranking classifier was only 1.60% away from the best result. The in-house system (selective Naïve Bayes) has a result less than 1% worse than the best model after one day ($\delta = (1 - 0.8311/0.8385) = 0.88\%$).

We conclude that the participants did very well in building models fast. Building competitive models is one day is definitely doable and the Orange in-house system is competitive, although it was rapidly beaten by the participants.

2. This improvement may be partly attributed to "unscrambling"; unscrambling was not possible during the fast track of the challenge (first 5 days).



(a) *Churn* (b) *Appetency*

(c) *Up-selling* (d) *Average*

Figure 4: **Performance improvement over time.** Delta* represents the relative difference in Test AUC compared to the overall best result $TestAUC^{**}$: $Delta^* = (TestAUC^{**} - TestAUC)/TestAUC^{**}$. On each box, the central mark is the median, the edges of the box are the 25^{th} and 75^{th} percentiles, the whiskers extend to the most extreme data points not considered to be outliers; the outliers are plotted individually as crosses.

INDIVIDUAL TASK DIFFICULTY

To assess the relative difficulty of the three tasks, we plotted the relative performance difference δ^* (Equation 2) for increasing periods of time, see Figure4.[a-c].

The *churn* task seems to be the most difficult one, if we consider that the performance at day one, 0.7467, only increases to 0.7651 by the end of the challenge (see Table 2 for other intermediate results). Figure 4.a shows that the median performance after one day is significantly worse than the baseline (Naïve Bayes), whereas for the other tasks the median was already beating the baseline after one day.

The *appetency* task is of intermediate difficulty. Its day one performance of 0.8714 increases to 0.8853 by the end of the challenge. Figure 4.b shows that, from day one, the median performance beats the baseline method (which performs relatively poorly on this task).

The *up-selling* task is the easiest one: the day one performance 0.9011, already very high, improves to 0.9092 (less that 1% relative difference). Figure 4.c shows that, by the end of the challenge, the median performance gets close to the best performance.

CORRELATION BETWEEN *TestAUC* AND *ValidAUC*:

The correlation between the results on the test set (100% of the test set), *TestAUC*, and the results on the validation set (10% of the test set used to give a feed back to the competitors), *ValidAUC*, is really good. This correlation (when keeping only AUC test results > 0.5 considering that test AUC < 0.5 are error submissions) is of 0.9960 ± 0.0005 (95% confidence interval) for the first 5 days and of is of 0.9959 ± 0.0003 for the 36 days. These values indicate that (i) the validation set was a good indicator for the online feedback; (ii) the competitors have not overfitted the validation set. The analysis of the correlation indicator task by task gives the same information, on the entire challenge (36 days) the correlation coefficient is for the Churn task: 0.9860 ± 0.001; for the Appetency task: 0.9875 ± 0.0008 and for the Up-selling task: 0.9974 ± 0.0002.

Several participants studied the performance estimation variance by splitting the training data multiple times into 90% for training and 10% for validation. The variance in the results that they obtained led them to use cross-validation to perform model selection rather than relying on the 10% feed-back. Cross-validation was used by all the top ranking participants. This may explain why the participants did not overfit the validation set.

We also asked the participants to return training set prediction results, hoping that we could do an analysis of overfitting by comparing training set and test set performances. However, because the training set results did not affect the ranking score, some participants did not return real prediction performances using their classifier, but returned either random results or the target labels. However, if we exclude extreme performances (random or perfect), we can observe that (i) a fraction of the models performing well on test data have a good correlation between training and test performances; (ii) there is a group of models performing well on test data and having an AUC on training examples significantly larger. Large margin models like SVMs (Boser et al., 1992) or boosting models (Freund and Schapire, 1996) behave in this way. Among the models performing poorly on test data, some clearly overfitted (had a large difference between training and test results).

7.3 Methods employed

We analyzed the information provided by the participants in the fact sheets. In Figures 5,6,7, and 8, we show histograms of the algorithms employed for preprocessing, feature selection, classifier, and model selection. We briefly comment on these statistics:

- **Preprocessing:** Few participants did not use any preprocessing. A large fraction of the participants replaced missing values by the mean or the median or a fixed value. Some added an additional feature coding for the presence of a missing value. This allows linear classifiers to automatically compute the missing value by selecting an appropriate weight. Decision tree users did not replace missing values. Rather, they relied on the usage of "surrogate variables": at each split in a dichotomous tree, if a variable has a missing value, it may be replaced by an alternative "surrogate" variable. Discretization was the second most used preprocessing. Its usefulness for this particular dataset is justified by the non-normality of the distribution of the variables and the existence of extreme values. The simple bining used by the winners of the slow track proved to be efficient. For categorical variables, grouping of under-represented categories proved to be useful to avoid overfitting. The winners of the fast and the slow track used similar strategies consisting in retaining the most populated categories and coarsely grouping the others in an unsupervised way. Simple normalizations were also used (like dividing by the mean). Principal Component Analysis (PCA) was seldom used and reported not to bring performance improvements.

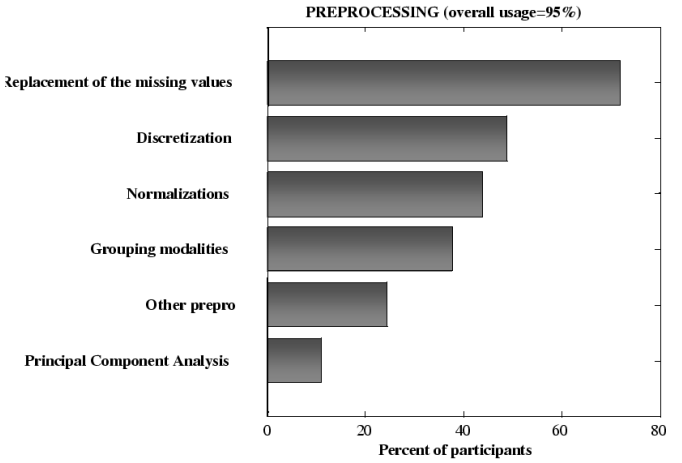


Figure 5: Preprocessing methods.

- **Feature selection:** Feature ranking and other filter methods were the most widely used feature selection methods. Most participants reported that wrapper methods overfitted the data. The winners of the slow track method used a simple technique based on cross-validation classification performance of single variables.

- **Classification algorithm:** Ensembles of decision trees were the most widely used classification method in this challenge. They proved to be particularly well adapted to the nature of the problem: large number of examples, mixed variable types, and lots of missing values. The second most widely used method was linear classifiers, and more particularly logistic regression (see *e.g.,* Hastie et al., 2000). Third came non-linear kernel methods (*e.g.,* Support Vector Machines, Boser et al. 1992). They suffered from higher computational requirements, so most participants gave up early on them and rather introduced non-linearities by building extra features.

- **Model selection:** The majority of the participants reported having used to some extent the on-line performance feed-back on 10% of the test set for model selection. However, the winners all declared that they quickly realized that due to variance in the data, this method was unreliable. Cross-validation (ten-fold or five-fold) has been the preferred way of selecting hyper-parameters and performing model selection. But model selection was to a large extent circumvented by the use of ensemble methods. Three ensemble methods have been mostly used: boosting (Freund and Schapire, 1996; Friedman, 2000), bagging (Breiman, 1996, 2001), and heterogeneous ensembles built by forward model selection (Caruana and Niculescu-Mizil, 2004; Caruana et al., 2006).

Surprisingly, less than 50% of the teams reported using regularization (Vapnik, 1998). Perhaps this is due to the fact that many ensembles of decision trees do not have explicit regularizers, the model averaging performing an implicit regularization. The wide majority of approaches were frequentist (non Bayesian). Little use was made of the unlabeled test examples for training and no performance gain was reported.

We also analyzed the fact sheets with respect to the software and hardware implementation (Figure 9):

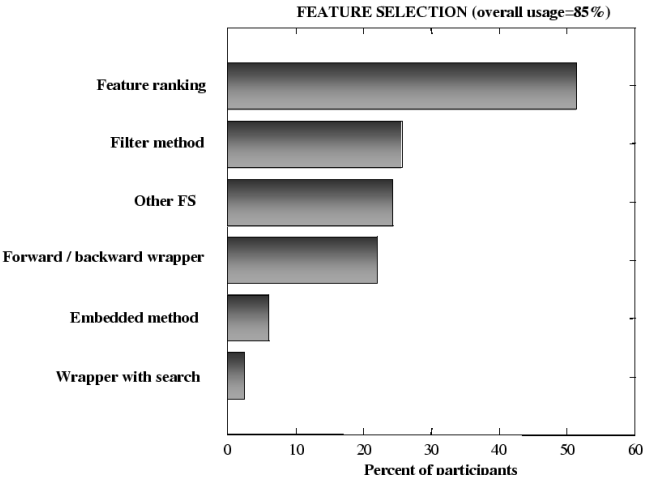


Figure 6: Feature selection methods.

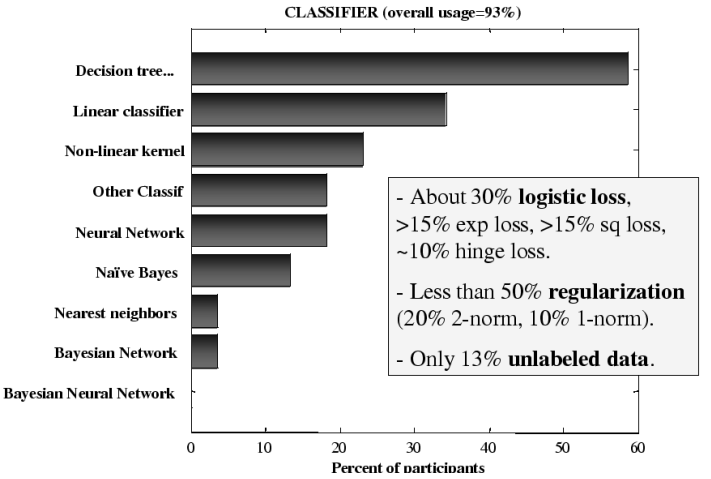


Figure 7: Classification algorithms.

- **Hardware:** While some teams used heavy computational apparatus, including multiple processors and lots of memory, the majority (including the winners of the slow track) used only laptops with less than 2 Gbytes of memory, sometimes running in parallel several models on different machines. Hence, even for the large dataset, it was possible to provide competitive solutions with inexpensive computer equipment. In fact, the in-house system of Orange computes its solution in less than three hours on a laptop.
- **Software:** Even though many groups used fast implementations written in C or C++, packages in Java (Weka) and libraries available in Matlab® or "R", presumably slow and memory inefficient, were also widely used. Users reported performing first feature selection to overcome speed and memory limitations. Windows was the most widely used operating system, closely followed by Linux and other Unix operating systems.

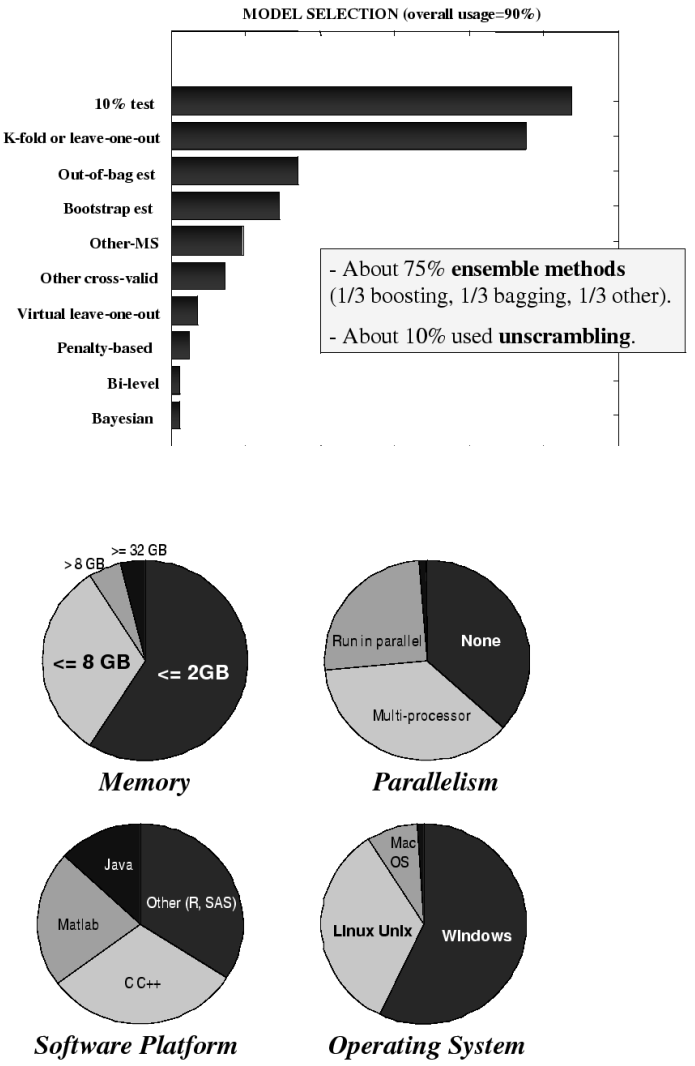
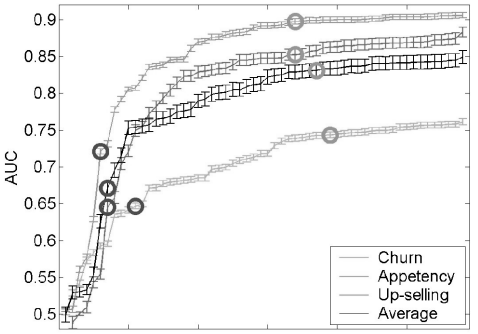


Figure 9: Implementation.

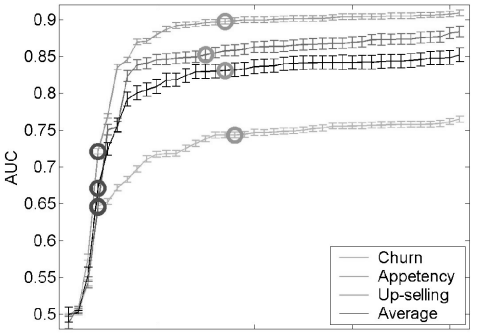
8. Significance Analysis

One of the aims of the KDD cup 2009 competition was to find whether there are data-mining methods which are significantly better than others. To this end we performed a significance analysis on the final results (last submission before the deadline, the one counting towards the final ranking and the selection of the prize winners) of both the SLOW and FAST track. Only final results reported on the large dataset were included in the analysis since we have realized that submissions based on the small dataset were considerably inferior.

To test whether the differences between the teams are statistically significant we followed a two step analysis that is specifically designed for multiple hypothesis testing when several independent task are involved (Demšar, 2006): First we used the Friedman test (Friedman, 1937), to examine the null hypothesis H_0, which states that the AUC values of the three tasks, (Churn, Appetency and Up-Selling) on a specific track (FAST or SLOW) are all drawn from a



(a) *FAST*



(b) *SLOW*

Figure 10: **Sorted final scores:** The sorted AUC values on the test set of each of the three tasks, together with the average of AUC on the three tasks. Only final submissions are included. (a) FAST track and (b) SLOW track. The baselines for the basic Naïve Bayes and selective Naïve Bayes are superposed on the corresponding tasks.

single distribution. The Friedman test is a non-parametric test, based on the average ranking of each team, where AUC values are ranked for each task separately. A simple test-statistic of the average ranks is sufficient to extract a p-value for H_0; In the case when H_0 is rejected, we use a two tailed Nemenyi test (Nemenyi, 1963) as a post-hoc analysis for identifying teams with significantly better or worse performances.

Not surprisingly, if one takes all final submissions, one finds that H_0 is rejected with high certainty (p-value $< 10^{-12}$). Indeed, significant differences are observed even when one inspects the average final AUCs (see Figure 10), as some submissions were not substantially better than random guess, with an AUC near 0.5. Of course, Figure 10 is much less informative than the significance testing procedure we adopt, which combines the precise scores on the three tasks, and not each one separately or their averages.

Trying to discriminate among the top performing teams is more subtle. When taking the best 20 submissions per track (ranked by the best average AUC) - the Friedman test still rejects

H_0 with p-values 0.015 and 0.001 for the FAST and SLOW tracks respectively. However, the Nemenyi tests on these reduced data are not able to identify significant differences between submissions, even with a significance level of $\alpha = 0.1$!

The fact that one does not see significant differences among the top performing submissions is not so surprising: during the period of the competition more and more teams have succeeded to cross the baseline, and the best submissions tended to accumulate in the tail of the distribution (bounded by the optimum) with no significant differences. This explains why the number of significant differences between the top 20 results decreases with time and number of submissions.

Even on a task by task basis, Figure 10 reveals that the performance of the top 50% AUC values lie on an almost horizontal line, indicating there are no significant differences among these submissions. This is especially marked for the SLOW track.

From an industrial point of view, this result is quite interesting. In an industrial setting many criteria have to be considered (performance, automation of the data mining process, training time, deployment time, etc.). But this significance testing shows using state-of the art techniques, one is unlikely to get significant improvement of performance even at the expense of a huge deterioration of the other criterions.

9. Conclusion

The results of the KDD cup 2009 exceeded our expectations in several ways. First we reached a very high level of participation: over three times as much as the most popular KDD cups so far. Second, the participants turned in good results very quickly: within 7 hours of the start of the FAST track challenge. The performances were only marginally improved in the rest of the challenge, showing the maturity of data mining techniques. Ensemble of decision trees offer off-the-shelf solutions to problems with large numbers of samples and attributes, mixed types of variables, and lots of missing values. Ensemble methods proved to be effective for winning, but single models are still preferred by many customers. Further work include matching the performances of the top ranking participants with single classifiers.

Acknowledgments

We are very grateful to the Orange company who donated the data, the computer servers, many hours of engineer time, and the challenge prizes. We would like to thank all the Orange team members, including Fabrice Clérot and Raphael Féraud. We also gratefully acknowledge the ACM SIGKDD and the Pascal2 European network of excellence (FP7-ICT-2007-1-216886) who supported the work of the KDD Cup 2009 co-chairs Isabelle Guyon and David Vogel and the web site development. The support of Google and Health Discovery Corporation allowed students to attend the workshop. We are very grateful to the technical support of MisterP and Pascal Gouzien. We also thank Gideon Dror for editing the proceedings.

References

H. Akaike. Information theory and an extension of the maximum likelihood principle. In B.N. Petrov and F. Csaki, editors, *2nd International Symposium on Information Theory*, pages 267–281. Akademia Kiado, Budapest, 1973.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, pages 144–152, 1992.

Marc Boullé. Compression-based averaging of Selective Naïve Bayes classifiers. *JMLR*, 8: 1659–1685, 2007. ISSN 1533-7928.

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

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

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. ISBN 1-58113-838-5.

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)*, December 2006. Full-length version available as Cornell Technical Report 2006-2045.

Clopinet. Challenges in machine learning. URL `http://clopinet.com/challenges`.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006. ISSN 1533-7928.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.

I. Guyon, A. Saffari, G. Dror, and J. Buhmann. Performance prediction challenge. In *IEEE/INNS conference IJCNN 2006*, Vancouver, Canada, July 16-21 2006.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Data Mining, Inference and Prediction*. Springer Verlag, 2000.

IBM Research. Winning the KDD cup orange challenge with ensemble selection. In *JMLR W&CP*, volume 7, KDD cup 2009, Paris, 2009.

Hung-Yi Lo et al. An ensemble of three classifiers for KDD cup 2009: Expanded linear model, heterogeneous boosting, and selective naïve Bayes. In *JMLR W&CP*, volume 7, KDD cup 2009, Paris, 2009.

Hugh Miller et al. Predicting customer behaviour: The University of Melbourne's KDD cup report. In *JMLR W&CP*, volume 7, KDD cup 2009, Paris, 2009.

T.M. Mitchell. *Machine Learning*. McGraw-Hill Co., Inc., New York, 1997.

P. B. Nemenyi. *Distribution-free multiple comparisons*. Doctoral dissertation, Princeton University, 1963.

V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, N.Y., 1998.

Jainjun Xie et al. A combination of boosting and bagging for KDD cup 2009 - fast scoring on a large database. In *JMLR W&CP*, volume 7, KDD cup 2009, Paris, 2009.

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.

©2009 IBM Research

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 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

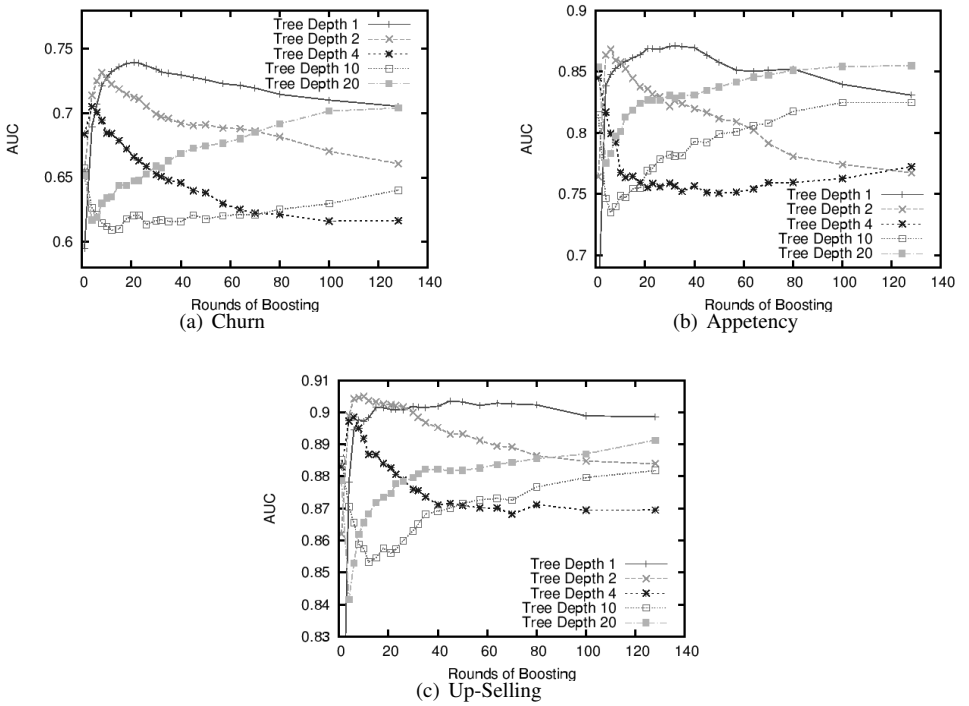


Figure 1: Performance obtained by boosting decision trees of various depths.

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

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.

and 0.9025 on up-selling (0.8443 average across the three problems). 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 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.

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

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.

JMLR Workshop and Conference Proceedings 7:35–43 KDD cup 2009

A Combination of Boosting and Bagging for KDD Cup 2009 — Fast Scoring on a Large Database

Jianjun Xie JXIE@IDANALYTICS.COM
Viktoria Rojkova VROJKOVA@IDANALYTICS.COM
Siddharth Pal SPAL@IDANALYTICS.COM
Stephen Coggeshall SCOGGESHALL@IDANALYTICS.COM
ID Analytics
15110 Avenue of Science
San Diego, CA, 92128, USA

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

Abstract

We present the ideas and methodologies that we used to address the KDD Cup 2009 challenge on rank-ordering the probability of churn, appetency and up-selling of wireless customers. We choose stochastic gradient boosting tree (TreeNet®) as our main classifier to handle this large unbalanced dataset. In order to further improve the robustness and accuracy of our results, we bag a series of boosted tree models together as our final submission. Through our exploration we conclude that the most critical factors to achieve our results are effective variable preprocessing and selection, proper imbalanced data handling as well as the combination of bagging and boosting techniques.

Keywords: KDD Cup, bagging, boosting, data mining, ensemble methods, imbalanced data

1. Introduction

The task of the KDD Cup 2009 is to build three Customer Relationship Management (CRM) models to predict three different wireless customer behaviors: 1) loss of interest toward current provider (churn), 2) propensity to purchase new products or services (appetency), and 3) tendency for upgrades or add-ons (up-selling).

Several aspects of data mining and statistical modeling have been addressed in this challenge:

- Handling a large dataset. The organizers provided 15000 variables, 14740 numerical and 260 categorical, to test the ability of participants in handling a large dataset. Although a downsized version with only 230 variables was made available in the second phase of the challenge, all the top teams also descrambled the variable mapping and used information from the large set.

©2009 Xie, Rojkova, Pal & Coggeshall

- Rapidity of model building. Participating teams were required to complete all three models in 5 days in order to win the fast track, which has a higher priority than the slow track.

- Variable preprocessing and selection. Variables were populated by unnormalized values. Missing entries and outliers are significant. Some categorical variables have a huge number of distinct entries. Effective variable preprocessing and selection is a must for any modeling algorithm to achieve the best results.

All three tasks are binary classification problems. There are several well established modeling algorithms available and suitable: Logistic Regression, Neural Networks, Decision Trees, SVM etc. Nowadays, ensemble learning schemes are widely used to enhance the overall performance of a single classifier by combining predictions from multiple classifiers (Breiman, 1996; Dietterich, 2000). In the family of decision tree classifiers, Random Forest uses bagging to combine many decision trees by bootstrap sampling (Breiman, 2001). TreeNet® uses the stochastic gradient boosting (Friedman, 1999a,b) which constructs additive regression trees by sequentially fitting a base learner to current pseudo-residuals by least squares at each iteration. The pseudo-residuals are the gradient of the loss functional being minimized, with respect to the model values at each training data point evaluated at the current step.

In this competition we use TreeNet® as our main classifier.The log-likelihood loss function has been chosen since all 3 tasks are binary classification problems. In order to further enhance the results, we combine bagging and boosting together. We bag a total of 5 boosted tree models for each task and take the average of all scores as the final prediction.

The results of each model are evaluated by area under receiver operating characteristic (ROC) curve, so called AUC. The AUC measurement does not require the models to produce the true probability of the predicted class as long as the model score can rank order the positive class and negative class effectively. This gives us more freedom of using sampling techniques to tackle the imbalance issue without worrying about converting the score back to a real probability.

We organize this paper as follows. Data analysis and preprocessing on training and testing datasets are described in Section 2. After establishing a gradient boosting tree as the main classifier, we proceed with variable selection and sampling the imbalanced data. These steps together with final bagging of different boosting decision tree models are described in Section 3. In Section 4 we explain our exploration on the small dataset. Finally, we summarize our final submissions and how they are compared with others in Section 5.

2. Data Analysis and Preprocessing

Data analysis is an important step of any data mining and modeling task. It helps for deep understanding the modeling task and in selecting the proper modeling technique. One illustrative example is KDD Cup 2008. The Patient ID was found to be a predictive variable: essentially a target leakage which was introduced into the training data by mistake. It was overlooked by everybody except for the winner (Perlich et al., 2008).

2.1 Histogram Analysis

The frequency distributions of all 15000 variables of the training and testing datasets are analyzed to establish the "equality" between training and testing samples. The binning for the histogram is performed around every discrete entry. Based on the histogram results we conclude that there is no substantial sampling bias between the training and testing data sets. In the

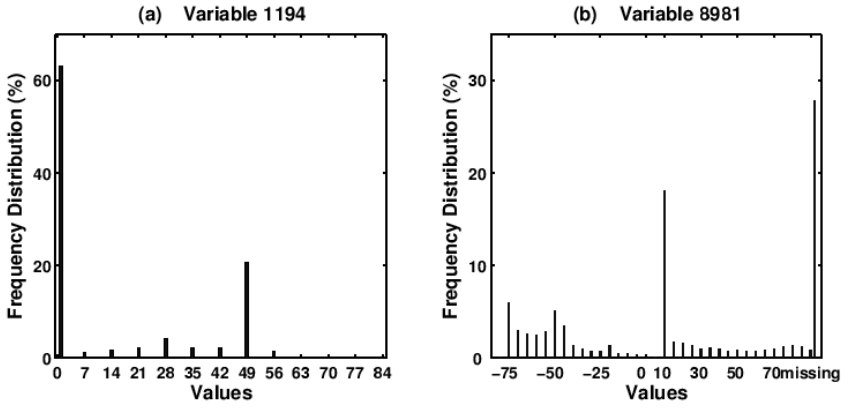


Figure 1: Example of histogram analysis: variable 1194 and variable 8981.

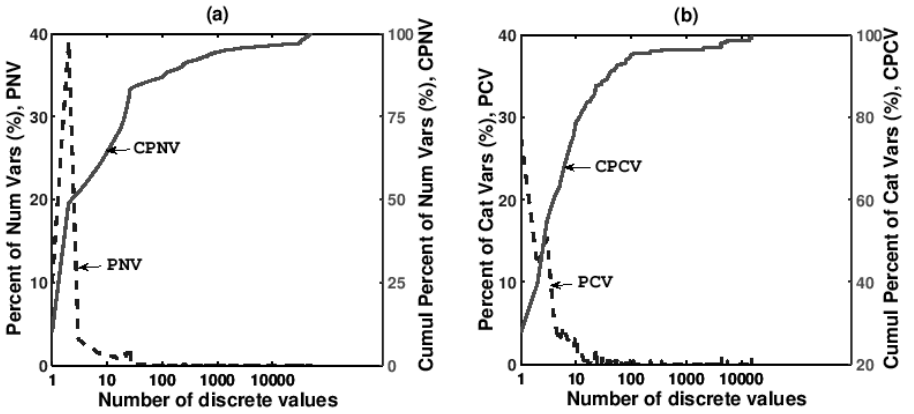


Figure 2: Distribution of discrete value frequencies in (a) numerical variable and (b) categorical variable.

slow track, we utilize histogram results of the large and small datasets to unscramble the small dataset. More details of the unscrambling process will be described in Section 4.1.

We find that most of the numerical variables have skewed distributions as shown in Figure 1. By checking the histograms of the variables, we discover that many of the numerical variables are populated by values that have a common factor, for example in Figure 1(a) the data values are all multiples of 7. This could be an indicator that these variables were artificially encoded.

Another observation is that many variables only have a few discrete values. For example, about 50% of all numerical variables have 1 or 2 discrete values as shown in Figure 2(a). Nearly 80% of all categorical variables have fewer than 10 categories, as shown in Figure 2(b). It can also be seen in Figure 2 that 12% of numerical variables and 28% of categorical variables are constant (only have a single entry). Furthermore, numerical values are heavily populated by 0s. We find that 80% of the numerical variables have more than 98% of their population filled by 0. These results suggest that a large number of variables can be removed since they are constant or close to constant.

Label frequencies for all three tasks are presented in Table 1. We can see that all of them are highly unbalanced. Appetency has extremely low positive rate. The imbalance of the class

Table 1: Histogram analysis on labels of churn, appetency and up-selling.

Churn	Appetency	Up-selling	Frequency	Percentage
-1	-1	-1	41756	83.51%
-1	-1	1	3682	7.36%
-1	1	-1	890	1.78%
1	-1	-1	3672	7.34%

distribution has to be taken into account in the model building step. There is no overlap between any pair of labels, they are exclusive to each other. This motivated us to incorporate the other 2 scores for a given model to improve the performance. However, we did not see significant improvement as described in Section 4.3.

2.2 Discretization and Grouping (Binning)

Even though most of numerical variables are populated by a limited number of discrete values, the population on each value differs significantly. We discretize 10 selected numerical variables that have strong correlation with the target. We consolidate the extremely low populated entries (having fewer than 200 examples) with their neighbors to smooth out the outliers. Similarly, we group some categorical variables which have a large number of entries (> 1000 distinct values) into 100 categories. This procedure of univariate groupings is frequently referred to as binning. Every category is replaced by a numerical risk factor (mean positive rate).

2.3 Missing Value Handling

A significant amount of variables are poorly populated, that is, for some variables many of the input values are missing. There are known techniques to approach the missing value problem which include mean substitution, multiple regression, maximum likelihood estimation, multiple imputation etc (Little and Rubin, 1987). In this work, we perform a simple substitution. We either replace them by a risk factor (for binned numerical variables and categorical variables) or treat them as a standalone entry ("missing" is simply another category of the input variable).

3. Modeling on Fast Track

3.1 Variable Selection

First, we removed 1531 constant variables and 5874 quasi-constant variables (where a single value occupies more than 99.98% population) based on our data analysis step. This left us a dataset with 7595 variables which is still a quite large number. We then went on with a multi-round wrapper approach. We first split the reduced training set into 3 chunks for each label and built 3 preliminary models for each task. The parameters used in TreeNet model were set as the following: learning rate = 0.02, number of nodes = 6, number of trees = 600. At every step TreeNet uses exhaustive search by trying all 7595 variables and split points to achieve the maximum reduction of impurity. Therefore, the tree construction process itself can be considered as a type of variable selection and the impurity reduction due to a split on a specific variable could indicate the relative importance of the variable in the tree model.

For a single decision tree a measure of variable importance can be calculated by (Breiman et al., 1984)

Table 2: Down-sampling rate for each modeling task.

Label	Sampling rate of negative records	Positive rate after sampling
Churn	70%	10.17%
Appetency	20%	8.31%
Up-selling	90%	8.12%

$$VI(x_i,T) = \sum_{t \in T} \triangle I(x_i,T),$$

where $\triangle I(x_i,T) = I(t) - p_L I(t_L) - p_R I(t_R)$ is the decrease in impurity to the actual or potential split on variable x_i at a node t of optimally pruned tree T. p_L and p_R are proportion of cases sent to the left or right by x_i.

The variables entered into the model based on their contribution to the impurity reduction at each split. We removed all variables with relative importance less than 2.0% for each model. We then merged all variables selected by these preliminary models for each task together as a pool. We got a total of 1720 variables for the next round selection. The final variable set was obtained by using 75% of all training data. The rest was reserved as testing. We kept removing variables that have the least importance until the model performance on the 25% test dataset started dropping. We narrowed down our final variable set to less than 300 for all 3 labels.

3.2 Down Sample Negative Population

As listed in Table 1, the distribution of labels for the 3 models are highly unbalanced. There are several popular ways to handle such an unbalanced dataset, for example, cost sensitive learning (Domingos, 1999), and a variety of sampling techniques (Van Hulse et al., 2007). Here we take the approach of down sampling the negative population. Table 2 lists the down sampling rate we used in our model. Since the model performance is measured by AUC which is the rank order of each record, the absolute value (scaling) of the score will not affect the results. Therefore, we directly used the raw score without any sampling correction.

3.3 Build the Best Boosting Tree Model via Cross Validation

We started building the best boosting tree models by adjusting the training parameters after variable selection. The factors we considered include the down sampling rate on the negative population, learning rate, number of trees, and minimum number of nodes. The model performance was evaluated by 5-fold cross validation. We also used the feedback on 10% test data as a reference. Not always was the 10% test feedback in agreement with the cross validation results. Upon checking the AUC results for each fold validation, we discovered large variations. Table 3 lists the AUC results for each fold and total of 5 folds for Upselling model. We can see that the variation among each fold can be as large as 0.02 which is big enough to drop your ranking by more than 20 in this very close competition. This makes us believe that the 10% feedback is not reliable to judge the performance of a model. We did not change our strategy even though we saw other team's results were better than ours based on the 10% test feedback.

Table 3: AUC results for Upsell model in 5-fold cross validation.

Fold Number	1	2	3	4	5	Total
AUC	0.8956	0.9112	0.9037	0.9182	0.9085	0.9071

3.4 Bagging the Selected Boosting Tree Models

After we determined the final variable set, learning rate, and other tree parameters we were convinced that the best strategy for improvement of model performance was consistent bagging. This is an essential part of our solution, since we take a down sampling approach on the negative class to improve the label balance, so as a result some of the records are never seen in the training set. By creating the training dataset 5 times using different random seeds, we built a total of 5 boosted tree models for each label. Our final model is a simple average of all these boosted tree models. We noticed after the competition that an ensemble of TreeNet classifiers won the 2003 Duke/NCR Teradata Churn model contest (Cardell et al., 2003). Ensembles of multiple TreeNet models usually outperform a single model.

4. Modeling on the Slow Track

We processed the small dataset in a similar way as we did for the large. We quickly realized that unscrambling the small dataset, mapping and combining it with the large might be the most beneficial strategy in the slow track. First, the rule of this year's KDD Cup competition requires the results on small dataset to compete with the large. Next, after experimenting in this direction we discovered that results on small dataset did not outperform the results on the large. Finally, a combination of small and large datasets gives us an additional 10% testing feedback on the same model.

4.1 Unscramble the Small Dataset

It turns out that unscrambling the small dataset is quite straightforward. There are two unscrambling steps: first is to unscramble the variable mapping, second is to unscramble the example order. Step 1 can be done by comparing the frequency distribution of each variable in the small and large datasets. This simple comparison maps 194 out of 230 variables in the small dataset to the large dataset. It is found that most of the numerical variables of the small dataset are simply scrambled as a constant fraction of the corresponding numerical variable in the large. The remaining variables typically have a one-to-many or many-to-many correspondence, which can be solved after Step 2.

In Step 2, we selected some mapped variables from the small and large and place them together as a key in the format of var_i, var_j, var_k ... var_n. We then cut them out from the small file in their original order. To ensure the uniqueness of the key enough variables have to be pulled. We create a file that has 2 fields, a sequence ID_{small} and a key. Then we created the same file using the mapped variables from the large data in the format of the sequence ID_{large} and key (consists of the corresponding variables from large dataset). The value is unscrambled so that the key will be same for both the large and small. Sort both files by key and then paste them together, and you get a map for the sequence IDs between the small and large data.

With this mapping table we converted one set of score files of the large dataset into the small data order and submitted for evaluation. Results on the 10% test feedback confirms that 1) there is a large variation of the AUC on the 10% test sample (we saw the same file geting 0.88 AUC on one 10% test sample and 0.79 AUC on another 10% test sample), and 2) the small

Table 4: AUC results of our final models on KDD Cup test dataset.Winner's final results are also listed for comparison.

Dataset	**Churn**		**Appetency**		**Up-selling**		**Scores**	
	10%	100%	10%	100%	10%	100%	10%	100%
Large (fast)	0.7333	0.7565	0.8705	0.8724	0.9308	0.9025	0.8354	0.8448
Large (slow)	0.7390	0.7614	0.8714	0.8761	0.9023	0.9061	0.8376	0.8479
Small (slow)	0.7612	0.7611	0.8544	0.8765	0.9155	0.9057	0.8437	0.8478
Winner's fast	-	0.7611	-	0.8830	-	0.9038	-	0.8493
Winner's slow	-	0.7651	-	0.8819	-	0.9092	-	0.8520

dataset does not have all the information needed to beat the large dataset. Therefore we stopped building models on the small dataset and focused on the large dataset only.

4.2 Combine Small and Large

We compared the variables selected from the small dataset and the large dataset, then added back several variables not present in the model on the large dataset and rebuilt the models on the large data. To improve the robustness of the model we binned all the top 10 numerical variables (if they were not discretized in the previous steps) for each task and then added them back to the variable list. We made 5 iterations of bagging on the final model. The final results are obtained by averaging the final score from the slow track and the score from the fast track.

4.3 Test of Using Scores from Other Models

As discussed in Section 2, the exclusive nature of the 3 tasks motivated us to incorporate scores from the other 2 tasks as variables in the 3rd task. We explored this strategy in the appetency model which has the most imbalanced distribution of target class. We did find a little improvement over the model without using other scores. However, this nested structure complicates the modeling process: update of the other models then requires the rebuilding of the underlying model.

5. Results and Discussions

Table 4 lists our final submissions for both the fast track and the slow track. The winner's results are also listed for comparison. Looking back to our submission history, we find that we did submit our best model as final, which confirms the correctness of our model improvement process. The difference between the 10% test and the 100% test is significant, which is in line with what we found in our cross validation process. The results on the small dataset are results of the model built on the large dataset bagged with one set of scores obtained from the real small data model. It actually decreases the overall performance from 0.8479 to 0.8478. In fact, all the top teams in the slow track descrambled the small dataset. In our view, the large dataset has more predictive information than the small.

We tried two other modeling techniques, logistic regression and SVM (Chang and Lin, 2001). Both of them require converting all categorical variables to proper numerical values. Logistic regression gives slightly worse results than boosted decision tress on same dataset we prepared. SVM can achieve similar accuracy but with much slower computing speed.

Comparing our results with the others in the contest, our overall performance ranked second in the fast track and third in the slow track. Our weakest model comparing with the winner was Appetency which has the most imbalanced class label. This might be an effect of our aggressive down-sampling and not enough number of bagging iterations.

In conclusion, we find through our practice that effective variable preprocessing and selection, proper imbalanced data handling and the combination of bagging and boosting are the important factors for achieving our results on the KDD Cup 2009 challenge.

Acknowledgments

We would like to thank the KDD Cup organizers for setting up this year's contest with a challenging problem, a nice website that participants can see online feedbacks and exchange ideas. Author J. Xie would like to thank his wife Rui Zhang for the support and encouragement she provided during her maternity leave for taking care of their baby girl Joann.

References

L. Breiman, Jerome H. Friedman, R.A. Olson, and C.J. Stone. *Classification and Regression Trees*. Wadsworth Int'l Group, Belmont, Calif., 1984.

Leo Breiman. Bagging predictors. In *Machine Learning*, volume 24, pages 123–140, 1996.

Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

N. Scott Cardell, Mikhail Golovnya, and Dan Steinberg. Churn modeling for mobile telecommunications, 2003. URL `http://www.salford-systems.com/doc/churnwinF08.pdf`.

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`.

Thomas G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *In Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164. ACM Press, 1999.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 1999a.

Jerome H. Friedman. Stochastic gradient boosting, 1999b. URL `http://www-stat.stanford.edu/~jhf/ftp/stobst.ps`.

R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, New York, 1987.

Claudia Perlich, Prem Melville, Yan Liu, Grzegorz Świrszcz, Richard Lawrence, and Saharon Rosset. Breast cancer identification: Kdd cup winner's report. *SIGKDD Explor. Newsl.*, 10 (2):39–42, 2008. ISSN 1931-0145.

Jason Van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. Experimental perspectives on learning from imbalanced data. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 935–942, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

JMLR Workshop and Conference Proceedings 7:45–55 KDD cup 2009

Predicting customer behaviour: The University of Melbourne's KDD Cup report

Hugh Miller H.MILLER@MS.UNIMELB.EDU.EDU
Department of Mathematics and Statistics
The University of Melbourne
Parkville, Victoria, 3010, Australia

Sandy Clarke SJCLARKE@UNIMELB.EDU.AU
Department of Mathematics and Statistics
The University of Melbourne
Parkville, Victoria, 3010, Australia

Stephen Lane S.LANE@MS.UNIMELB.EDU.AU
Department of Mathematics and Statistics
The University of Melbourne
Parkville, Victoria, 3010, Australia

Andrew Lonie ALONIE@UNIMELB.EDU.AU
Department of Information Systems
The University of Melbourne
Parkville, Victoria, 3010, Australia

David Lazaridis D.LAZARIDIS@PGRAD.UNIMELB.EDU.AU
Department of Mathematics and Statistics
The University of Melbourne
Parkville, Victoria, 3010, Australia

Slave Petrovski SLAVEP@UNIMELB.EDU.AU
Department of Medicine
The Royal Melbourne Hospital
The University of Melbourne
Parkville, Victoria, 3010, Australia

Owen Jones ODJONES@UNIMELB.EDU.AU
Department of Mathematics and Statistics
The University of Melbourne
Parkville, Victoria, 3010, Australia

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

Abstract

We discuss the challenges of the 2009 KDD Cup along with our ideas and methodologies for modelling the problem. The main stages included aggressive nonparametric feature selection, careful treatment of categorical variables and tuning a gradient boosting machine under Bernoulli loss with trees.

Keywords: KDD Cup 2009, nonparametric feature selection, generalised boosting machine, decision trees

©2009 Hugh Miller, Sandy Clarke, Stephen Lane, Andrew Lonie, David Lazaridis, Slave Petrovski and Owen Jones

1. Introduction

The KDD Cup 2009[1] was organised by Orange Labs, France. The data consisted of information about telecommunication customers, with 15,000 predictor variables. The competition involved producing binary classifiers for three types of consumer behaviour:

- churn, which is whether someone ceases to be a customer,
- appetency, being the propensity to buy a service or product, and
- upselling, where a more profitable or additional service is sold to a customer.

Competitors were provided with a training set of 50,000 observations, with an additional 50,000 in the test set, which was used by the organisers for model evaluation. The measure for predictive accuracy was the area under the ROC curve (AUC), which integrates sensitivity over all possible specificities of the model. The average of the AUC for the three different classification tasks was used to rank competitors. A reduced dataset of 230 variables was also available, which our team did not make use of for our primary entry.

The challenge had a fast component, with predictions for the test data due within 5 days of the full data being released, and a slow component, where predictions had to be submitted within 5 weeks. IBM Research produced the best model for both components, but as the competition rules stated that no team could win both parts, The University of Melbourne team won first prize in the slow component, having the second best model. Table 1 shows the final results for both IBM research and The University of Melbourne. Our model was based entirely on the large dataset, making no use of the other smaller dataset provided to competitors.

Table 1: Final model performance for IBM research and The University of Melbourne.

	Model			
Team	Churn	Appetency	Upselling	Average
IBM Research	0.7651	0.8819	0.9092	0.85206
Univ. Melbourne	0.7570	0.8836	0.9048	0.84847

The dataset provided for the KDD Cup 2009 is typical of many contemporary data-mining problems. There are a large number of observations, which enables many signals to be resolved through the noise, allowing complex models to be fit. There are also a large number of predictors, which is common since companies and other organisations are able to collect a large amount of information regarding customers. However many of these predictors will contain little or no useful information, so the ability to exclude redundant variables from a final analysis is important. Many of the predictors have missing values, some are continuous and some are categorical. Of the categorical predictors, some have a large number of levels with small exposure; that is, a small number of observations at that level. For the continuous variables, the distribution among the observations can have extreme values, or may take a small number of unique values. Further, there is potential for significant interaction between different predictors. Finally, the responses are often highly unbalanced; for instance only 7% of the upselling observations were labelled "1". All these factors need to be considered in order to produce a satisfactory model. Sections 2 to 4 detail the stages of our modelling for the KDD Cup, while Section 5 makes some comment on the computational resources used.

1. `www.kddcup-orange.com`

2. Feature selection

As mentioned in the introduction, many of the predictors were irrelevant for predictive purposes and thus needed to be excluded. In fact, some variables were absolutely redundant, having the same entry in all cells. Over 3,000, about 20%, of the variables had either entirely identical observations, or had fewer than 10 observations different to the baseline, so these were obvious candidates for removal.

For those features remaining, we assessed the individual predictive power with respect to the three responses (churn, appetency and upselling). To do this we split the data into two halves, one to make predictions and the other to measure the resulting AUC, so that the measure of predictor performance was directly related to the measure used in the competition. For categorical values, the proportion of positive responses for each level was used as the prediction that was applied to the second half of the data. For continuous variables we separated the observations into bins based on 1% quantiles and used the proportion of positive responses for each quantile bin as the prediction. In both cases missing values were treated as a separate level. An AUC score could then be calculated for each variable using the second half of the training data and the process was repeated to increase reliability

The above feature selection technique is very simple; it involves taking the mean of the responses for each level, and so amounts to a least squares fit on a single categorical variable against a 0-1 response, with the categories in the continuous case defined by quantiles. Despite its simplicity, it had a number of advantages:

- **Speed:** Computing means and quantiles is direct and efficient
- **Stability with respect to scale:** Extreme values for continuous variables do not skew predictions as they would in many models, especially linear models, and the results are invariant under any monotone transformation of the continuous variables. Therefore this is robust to unusual distribution patterns.
- **Comparability between continuous and categorical variables:** Predictive performance of the two types of variables is measured in a similar way and so they are directly comparable.
- **Accommodation of nonlinearities:** Since a mean is estimated for every quantile in the continuous case, nonlinear dependencies are just as likely to be detected as a linear pattern.

Naturally there were some drawbacks to this approach as well. For instance, by underemphasising linear patterns, any genuine linear or nearly linear patterns were less likely to be detected. Also, the choice of 1% for the quantiles was somewhat arbitrary, but judged to maintain a reasonable balance between shape flexibility and reliability. Figure 1 shows the quantile fit for the most important variable in the churn model, as recorded in Table 4, against the response. Although the fit does exhibit substantial noise when compared to the smoothed overlay, created using a local kernel regression fit, there remains a strong detectable signal and the noise is mitigated by testing on a separate portion of the data. It is also noteworthy that this variable exhibits significant nonlinearity.

This method of feature selection can be considered as a special case of generalised correlation as in Hall and Miller (2009). There the generalised correlation of the jth variable is the maximum correlation of the response with a nonlinear transformation of that variable:

$$\rho_j = \sup_{h \in \mathscr{H}} \operatorname{cor}\{h(X_j), Y\},$$

where $\mathscr{H}$ is the allowable set of nonlinear transformations. When this set has a finite basis then the choice of h is equivalent to the least squares fit under the basis of $\mathscr{H}$. In our case the

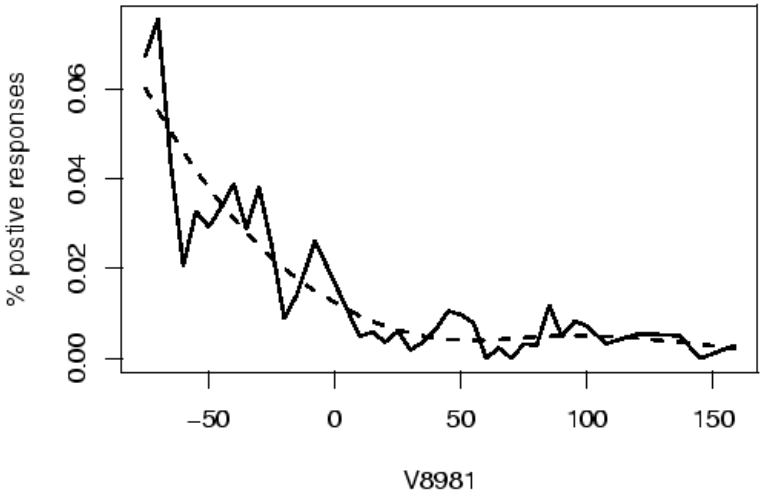


Figure 1: Quantile and smooth fits for variable V8981 against the churn response.

finite basis was the collection of quantile based indicator functions (in the continuous case), or indicator functions for each category (for categorical variables). Thus the feature selection may be thought of as maximising the nonlinear correlation between each variable and the response, making use of a large number of degrees of freedom, as permitted by the relatively large number of observations.

The above rankings were reasonably effective in capturing all the interesting behaviour for the churn and appetency models. However for the upselling model, spurious variables tended to appear high in the variable ranking. In this case, the list of top variables needed to be adjusted in the later, more sophisticated modelling stages to produce competitive results.

Figure 2 shows the sorted AUC scores for all the variables using the churn response. The plot is typical of the three different models, with the bulk of predictors having AUC close to 0.5, implying no relationship with the response. The dotted line represents our cutoff for admission into the boosted model of Section 4. The cutoff is reasonably aggressive, but there did not appear to be much gain in admitting more variables. Even if a more conservative cutoff was adopted, considering more than the top 2,000 variables for the final model appears to be unnecessary, so a substantial dimensionality reduction is possible and preferred.

We also compared the results of this feature selection with a F-score based feature selection method as described in Chen and Lin (2006). In general, agreement was good, although this alternative method suggested a small number of new variables to also include at the modelling stage.

3. Treatment of categorical variables with a large number of levels

Many of the categorical variables had a large number of levels—some even having over 10,000—and some of these ranked high in our feature selection. In many modelling contexts such an abundance of levels is undesirable, since it encourages overfitting on the small exposure levels. This is particularly true of decision trees, which we used for building our final models. Another

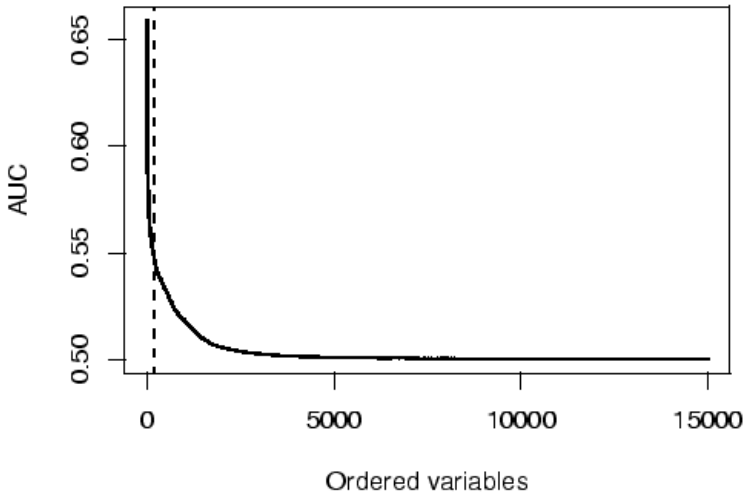


Figure 2: AUC scores for feature selection using churn response.

problem is that some levels appear in the training set but not the test set and vice versa. While some of the levels that had a large exposure were important, the other levels needed aggregation.

Our initial attempt to aggregate was to collapse the levels with less than 1,000 corresponding observations into 20 levels, with grouping based on the response. Thus levels with small exposure and a large proportion of positive responses were grouped together, while those with small exposure but lower proportion would be aggregated in a different level. This was the aggregation we used for the fast part of the challenge. Unfortunately this exacerbated the overfitting problem because we were artificially creating good predictors of the training set which depressed model performance on the test set, so an alternative means of aggregation was necessary.

To prevent this kind of overfitting, our second attempt at aggregation was completed independently of the response. If a categorical variable had more than 25 levels, we created a replacement variable by:

- keeping any levels that had at least 1000 observations worth of exposure,
- aggregating any levels with exposure between 500-999 into a new level,
- aggregating any levels with exposure between 250-499 into a new level, and
- aggregating any levels with exposure between 1-250 into a new level.

This removed the overfitting problem. It is not entirely clear whether the aggregating into three levels based on exposure did in fact provide any improvement compared to using a single level, although there is some supporting evidence. For instance some variables, such as V14788 and V14904, had only levels corresponding to the different exposures and were judged significant in some of our models. Also, Table 2 gives AUC scores on 5-fold cross-validated training set predictions for our final models using the exposure aggregation compared to a single aggregation. The churn and appetency models in particular seem to support the exposure based aggregation. While not conclusive, it is worth noting that if the differences in the table are representative then not including the exposure levels would have lowered the team's ranking.

Table 2: AUC scores comparing aggregation approaches for categorical variables

Model	Exposure-based aggregation	Single level	Difference
Churn	0.7493	0.7478	0.0015
Appetency	0.8790	0.8784	0.0006
Upselling	0.9062	0.9063	-0.0001

Another advantage of this approach compared to the initial attempt was that the processed categorical variables were the same across the three consumer behaviours.

4. Modelling with decision trees and boosting

The basic approach for constructing the final models involved the collection of shallow decision trees with boosting and shrinkage as in gradient boosting machines. Friedman (2001) serves as a primary reference for this approach; other literature on boosting includes Freund and Schapire (1997), Ridgeway (1999), Friedman et al. (2000) and Friedman (2002). Decision trees have been studied for many years, and include the work of Morgan and Sonquist (1963), Breiman et al. (1984) and Quinlan (1993). The basic principle is to fit a relatively simple tree-based model many times, each time focusing on the observations that are hardest to classify correctly by means of a weighting scheme. Bernoulli loss was used to compute the deviance, and the class weights were chosen so that the two classes had roughly equal weight. For example the churn model used a weight of 12 for the positive class, to better balance the trees.

Decision trees have a number of advantages which suited this year's KDD data, in particular. These are well-known, but worth restating here:

- Predictions are possible even when an important predictor has a missing value, through the use of surrogate variables.
- They are not affected by extreme values or strange distributions in continuous variables. In fact, they are invariant under monotone transformations of the predictors.
- They can easily handle both continuous and categorical variables.
- They can effectively model interactions between predictors.
- They allow for nonlinear dependencies.

Model validity was tested both by cross-validation and using the online feedback on the 10% test sample provided by the organisers. We aimed to build a model using about 200 predictors, partly for computational reasons and partly because adding extra predictors to our final subsets did not appear to noticeably improve performance. These variables were chosen on the basis of the feature selection ranking. However an important part of tuning the models involved discarding variables that did not appear useful in the model and adding some lower down the feature selection ranking. Here usefulness refers to the relative amount of deviance (Bernoulli loss) reduction each variable contributes to the model. Details of the variables used in each of the models are given in Appendix A. Model parameters for each of the fits are presented in Table 3. These were selected to maximise the AUC performance, using the test set feedback and cross-validation.

The final models suggest that there are some significant interactions between predictors in the models, most strikingly between continuous and categorical variables. Figure 3 shows one example of this, plotting the partial dependence between the two most important variables in the appetency model, V9045 and two levels of V14990. Note that this is not the change in the

Table 3: Model parameters for boosted tree models

	Model		
	Churn	Appetency	Upselling
Number of variables	198	196	201
Class weight	12	20	12
Shrinkage parameter	0.01	0.01	0.01
Number of trees	1300	1300	3000
Tree depth	5	3	5

response excluding the effect of all the other variables, but rather integrating over them. The different behaviour in the continuous variable for the different levels is visible.

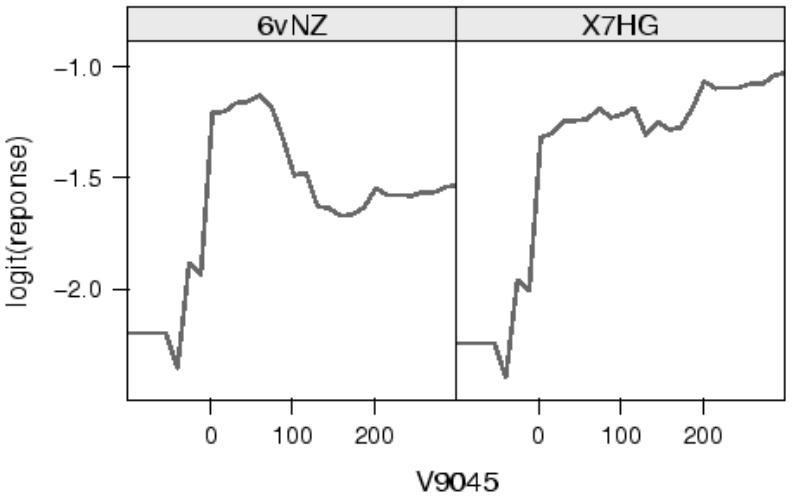


Figure 3: Partial dependence plots in the appetency model for variables V9045 and two levels of V14990. The different shapes, particularly for higher values of V9045, suggest interactions are present.

5. Computational details

The analysis and modelling work was performed almost entirely in the free open source program R.[2] We say "almost", because the original data chunks were too large to be read into R with our limited hardware, so it was first read into SAS[3] and exported in batches of 200 variables, each of which could then be read into and then deleted from R.

All modelling was conducted on individual desktop and laptop computers; the computer that did the most modelling was a mass-market laptop running Windows XP with 2Gb of RAM, a 2.66GHz Intel Core 2 Duo processor and a 120Gb hard drive. The feature selection and

2. http://cran.r-project.org/
3. http://www.sas.com/

categorical collapsing was programmed ourselves, while the boosted decision tree used the "gbm" package, also freely downloadable[2].

The feature selection stage took a few hours of computing time for each response, while the boosted decision tree models typically took just over an hour to fit, depending on the number of trees and variables involved. This demonstrates that a linux cluster is not necessary to produce strong predictive results, although the authors suspect it would help; in our case, it would have enabled more comprehensive investigation of the effect of choices in category collapsing and feature selection. Interested readers are encouraged to contact the first author regarding any questions of coding or computation, or with any suggestions and comments.

References

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, Belmont, 1984.

Y. W. Chen and C. J. Lin. Combining svms with various feature selection strategies. In *Feature extraction, foundations and applications*. Springer-Verlag, 2006.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38 (4):367–378, 2002.

J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–374, 2000.

P. Hall and H. Miller. Using generalised correlation to effect variable selection in very high dimensional problems. *Journal of Computation and Graphical Statistics (to appear)*, 2009.

J. N. Morgan and J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58:415–434, 1963.

R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, 1993.

G. Ridgeway. The state of boosting. *Computing Science and Statistics*, 31:172–181, 1999.

Appendix A. Tables relating to final models

Table 4: Relative influence of top 20 variables in final models

Rank	Churn		Appetency		Upselling	
	Name	Rel. Inf.	Name	Rel. Inf.	Name	Rel. Inf.
1	V8981	20.13	V9045	23.78	V9045	45.52
2	V14990	10.25	V8032	13.56	V14990	7.86
3	V10533	4.65	V14995	10.79	V8981	5.32
4	V14970	4.60	V14990	6.07	V12507	4.96
5	V5331	2.36	V5826	3.72	V6808	4.65
6	V14995	2.19	V8981	3.23	V1194	2.58
7	V14822	2.10	V10256	3.03	V14970	2.16
8	V9045	2.00	V12641	2.72	V14871	1.33
9	V2570	2.00	V14772	1.72	V1782	1.15
10	V14923	1.88	V14939	1.68	V10256	1.05
11	V14765	1.19	V14867	1.62	V5026	0.96
12	V14904	1.14	V14970	1.42	V8032	0.91
13	V5702	1.13	V11781	1.14	V14786	0.81
14	V11047	1.12	V14871	0.89	V7476	0.62
15	V14778	0.97	V14788	0.86	V11781	0.59
16	V14795	0.90	V13379	0.81	V14795	0.57
17	V990	0.90	V5216	0.71	V6255	0.57
18	V12580	0.86	V14795	0.70	V5216	0.50
19	V9075	0.86	V11315	0.66	V2591	0.50
20	V647	0.85	V12702	0.62	V12641	0.46

Table 5: Variables used in final models

Churn			Appetency			Upselling		
V47	V5216	V10447	V28	V5723	V11315	V28	V5216	V10136
V173	V5245	V10513	V83	V5808	V11322	V169	V5405	V10256
V384	V5277	V10533	V134	V5826	V11392	V173	V5462	V10402
V559	V5331	V10557	V182	V5873	V11396	V182	V5521	V10443
V621	V5360	V10589	V193	V5899	V11642	V213	V5576	V10521
V635	V5365	V10687	V282	V6003	V11777	V542	V5632	V10538
V647	V5559	V10808	V647	V6016	V11781	V559	V5723	V10687
V698	V5613	V10985	V698	V6238	V11916	V749	V5815	V11051
V706	V5666	V11047	V855	V6310	V12058	V941	V5826	V11083
V724	V5702	V11068	V941	V6424	V12102	V959	V5840	V11092
V749	V5723	V11172	V959	V6468	V12147	V975	V5985	V11115
V843	V5808	V11247	V1026	V6503	V12252	V1004	V6032	V11135
V941	V5820	V11315	V1075	V6565	V12264	V1045	V6228	V11160
V953	V5833	V11322	V1204	V6620	V12321	V1194	V6246	V11196
V990	V5895	V11392	V1275	V6659	V12483	V1362	V6255	V11277
V1036	V5982	V11480	V1476	V6735	V12507	V1376	V6503	V11315
V1095	V6016	V11671	V1514	V6751	V12517	V1596	V6514	V11369
V1227	V6049	V11731	V1543	V6812	V12548	V1623	V6565	V11566
V1254	V6255	V11985	V1596	V6825	V12638	V1782	V6637	V11781
V1392	V6310	V12199	V1969	V7004	V12641	V1853	V6735	V11832
V1428	V6468	V12200	V2120	V7055	V12670	V1925	V6778	V11859
V1501	V6534	V12264	V2157	V7180	V12702	V2095	V6808	V12011
V1565	V6551	V12370	V2284	V7212	V12747	V2120	V6837	V12058
V1604	V6636	V12381	V2334	V7335	V12840	V2157	V6892	V12147
V1996	V6653	V12580	V2352	V7356	V12884	V2249	V6894	V12199
V2284	V6722	V12702	V2413	V7575	V13084	V2321	V7004	V12221
V2315	V7071	V12840	V2418	V7579	V13104	V2434	V7014	V12264
V2370	V7146	V12993	V2453	V7651	V13362	V2453	V7029	V12507
V2450	V7212	V13008	V2531	V7653	V13379	V2531	V7055	V12539
V2453	V7229	V13038	V2544	V7904	V13492	V2591	V7230	V12548
V2456	V7425	V13053	V2591	V7950	V13653	V2849	V7308	V12641
V2570	V7500	V13153	V2715	V7960	V13871	V2852	V7476	V12702
V2773	V7511	V13210	V2822	V8003	V13952	V2890	V7485	V12884
V2822	V7670	V13350	V2849	V8032	V14221	V2892	V7521	V12952
V2852	V7706	V13571	V2852	V8343	V14246	V2985	V7522	V13038
V2961	V7758	V13572	V2966	V8458	V14334	V3128	V7575	V13135
V3080	V7817	V13573	V3000	V8591	V14344	V3219	V7579	V13153
V3104	V7964	V13644	V3128	V8619	V14362	V3305	V7631	V13162
V3264	V8032	V13663	V3130	V8787	V14374	V3487	V7737	V13287
V3305	V8181	V13714	V3199	V8936	V14377	V3558	V7874	V13362
V3339	V8375	V13849	V3202	V8981	V14517	V3568	V7987	V13379
V3439	V8484	V14087	V3219	V9001	V14643	V3711	V8032	V13467
V3508	V8605	V14187	V3249	V9045	V14696	V3962	V8070	V13469
V3515	V8621	V14226	V3305	V9248	V14721	V3999	V8122	V13592
V3624	V8709	V14274	V3339	V9311	V14732	V4048	V8181	V13653
V3719	V8717	V14334	V3704	V9408	V14772	V4075	V8338	V13705
V3759	V8854	V14359	V3719	V9409	V14786	V4221	V8458	V13727
V3766	V8863	V14429	V3759	V9655	V14788	V4316	V8505	V13952
V3886	V8981	V14487	V3863	V9671	V14795	V4566	V8561	V14015
V3905	V9001	V14502	V4186	V9704	V14834	V4585	V8591	V14138
V3972	V9037	V14765	V4248	V10032	V14846	V4614	V8619	V14157
V4028	V9045	V14778	V4340	V10130	V14867	V4659	V8833	V14170
V4088	V9075	V14788	V4347	V10212	V14871	V4665	V8981	V14362
V4098	V9342	V14791	V4585	V10256	V14878	V4686	V9045	V14721
V4218	V9375	V14795	V4590	V10333	V14923	V4735	V9051	V14773
V4389	V9408	V14822	V4614	V10343	V14928	V4802	V9069	V14778
V4393	V9498	V14846	V4665	V10405	V14939	V4856	V9230	V14786
V4563	V9536	V14871	V4902	V10415	V14970	V4996	V9294	V14795
V4669	V9608	V14904	V4957	V10443	V14974	V5021	V9311	V14862
V4735	V9616	V14906	V5026	V10450	V14980	V5026	V9386	V14871
V4856	V9686	V14923	V5065	V10521	V14990	V5053	V9409	V14890
V4986	V9704	V14970	V5185	V10589	V14995	V5065	V9431	V14928
V5025	V9711	V14990	V5213	V10594		V5097	V9574	V14946
V5026	V9799	V14995	V5216	V10739		V5138	V9658	V14965
V5031	V10073		V5405	V10843		V5144	V9708	V14970
V5166	V10183		V5462	V11196		V5182	V9797	V14990
V5170	V10256		V5554	V11247		V5213	V10097	V14995

JMLR Workshop and Conference Proceedings 7:57–64 KDD cup 2009

An Ensemble of Three Classifiers for KDD Cup 2009: Expanded Linear Model, Heterogeneous Boosting, and Selective Naïve Bayes

Hung-Yi Lo, Kai-Wei Chang, Shang-Tse Chen, Tsung-Hsien Chiang, Chun-Sung Ferng, Cho-Jui Hsieh, Yi-Kuang Ko, Tsung-Ting Kuo, Hung-Che Lai, Ken-Yi Lin, Chia-Hsuan Wang, Hsiang-Fu Yu, Chih-Jen Lin, Hsuan-Tien Lin, Shou-de Lin {D96023, B92084, B95100, B93009, B95108, B92085, B93038, D97944007, R97028, R97117, B94B02009, B93107, CJLIN, HTLIN, SDLIN}@CSIE.NTU.EDU.TW

Department of Computer Science and Information Engineering, National Taiwan University Taipei 106, Taiwan

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

Abstract

This paper describes our ensemble of three classifiers for the KDD Cup 2009 challenge. First, we transform the three binary classification tasks into a joint multi-class classification problem, and solve an l1-regularized maximum entropy model under the LIBLINEAR framework. Second, we propose a heterogeneous base learner, which is capable of handling different types of features and missing values, and use AdaBoost to improve the base learner. Finally, we adopt a selective naïve Bayes classifier that automatically groups categorical features and discretizes numerical ones. The parameters are tuned using cross-validation results rather than the 10% test results on the competition website. Based on the observation that the three positive labels are exclusive, we conduct a post-processing step using the linear SVM to jointly adjust the prediction scores of each classifier on the three tasks. Then, we average these prediction scores with careful validation to get the final outputs. Our final average AUC on the whole test set is 0.8461, which ranks third place in the slow track of KDD Cup 2009.

Keywords: Heterogeneous large dataset, regularized maximum entropy model, AdaBoost, selective naïve Bayes

1. Introduction

The KDD Cup 2009 challenge[1] aims at developing a predictive model for the customer relationship management. It consists of three tasks: predicting the churn, appetency and up-selling characteristics of each customer. The dataset is provided by Orange, a French telecom company. There are two versions of the dataset: the large version contains 15,000 feature variables and the small version contains only 230. In both versions, there is a training set and a test set, each containing 50,000 examples. The performance of the predictions is evaluated according to the area under the ROC curve (AUC). Below we describe some important properties of the dataset:

1. http://www.kddcup-orange.com/

©2009 Lo et al.

1. Many of the features, in particular for the large version, are irrelevant or redundant and there is a significant amount of missing values.

2. The datasets are heterogeneous. In other words, they contain both numerical and categorical features. The number of distinct categories varies sharply on different categorical features. It can range from two to more than ten thousand.

This paper describes our solutions to handle the above challenges. Our approach combines three classifiers. The first classifier transforms the three binary classification tasks into a single multi-class classification problem, and solves an l1-regularized maximum entropy model by coordinate decent under the LIBLINEAR framework. We feed the classifier with pre-processed features that are constructed by adding categorical indicators and missing indicators and using both linear and log scaling strategies. The second classifier couples the AdaBoost algorithm with a heterogeneous base leaner that trains with the original dataset without pre-processing. The third classifier is a selective naïve Bayes classifier that automatically groups categorical features and discretizes numerical ones. We also conduct post-processing to adjust the prediction scores across the three tasks. Our final model is composed by averaging the predictions from the three classifiers.

The next section describes details of our methods. In Section 3, we show the experimental settings and results. Then, we discuss some other observations and ideas in Section 4, and conclude in Section 5.

2. Method

In this section, we discuss our feature pre-processing method, three main classifiers, and the post-processing strategy. We first define the following notations. The training set is $(x_i, y_i)_{i=1}^{l}$, where $x_i \in R^n$ is the feature vector and $y_i \in \{1, -1\}$ is the class label. The j-th feature of x is denoted by x_j.

2.1 Feature Pre-processing

Some learning models such as boosting can directly deal with heterogeneous data and missing values. However, some models such as the maximum entropy model assume no missing values in the data. With a proper pre-processing approach, it is possible to improve not only the test performance but also the training speed. This section introduces how we handle heterogeneous data and missing values for the maximum entropy model. The other two classifiers deal with those issues in different manners, which will be described in the corresponding sections.

2.1.1 FEATURE WITH MISSING VALUES

In the training data, 371 numerical and 243 categorical features contain at least one missing value. All missing values are filled with zero while we add 371+243 binary features to indicate the missing status. That is, we add a 0/1 indicator for any feature with at least one missing value. Experiments show that this strategy improves the test performances considerably. Consider the linear model with the weight vector w Assume that there is only one feature t with missing values and we add one feature so the final model looks like $\hat{w}^T = [w^T \ \hat{w}_t]$. Then, the decision value of any instance x is

$$\hat{w}^T x = \begin{cases} w^T x + \hat{w}_t, & \text{if } x_t \text{ is missing,} \\ w^T x, & \text{otherwise.} \end{cases}$$

This is equivalent to setting $x_t = \frac{\hat{w}_t}{w_t}$ when x_t is missing. In other words, the indicator allows the linear model to fill in the missing values automatically.

2.1.2 CATEGORICAL FEATURE

We transform each categorical feature to several binary ones. A binary feature corresponds to a possible value of the categorical feature. This setting induces a large number (111, 670) of additional features. Furthermore, for numerical features with less than five values, we treat them as if they are categorical and also add binary indicators for them.

2.1.3 SCALING

We observe that in the training data, features are in quite different ranges. More specifically, some feature values are extremely large, which not only cause numerical difficulties but also hurt the classification performance. To tackle this problem, we scale both the training and test instances. After several experiments, we decide to use both log scaling (scale each value by the logarithm function) and linear scaling (scale each feature to the range $[0,1]$ linearly) for the numerical features. Both types of scaled features are included for training and testing.

2.2 Classification

2.2.1 REGULARIZED MAXIMUM ENTROPY MODEL

In the training data, we observe that the positive labels (i.e., $y_i = 1$) for the three tasks (churn, appetency, and upselling) are exclusive. That is, the label of each example falls into the following four classes:

	Label of churn	Label of appetency	Label of upselling
class 1	1	0	0
class 2	0	1	0
class 3	0	0	1
class 4	0	0	0

This observation inspires us to transform the three independent binary classification tasks into a joint multi-class classification problem. We then propose the following multi-class l1-regularized maximum entropy model:

$$\min_{w_1,\ldots,w_k} \sum_{y=1}^{k}\sum_{t=1}^{n} |w_{yt}| + C\log\sum_{i=1}^{l} \frac{e^{w_{\bar{y}_i}^T x_i}}{\sum_{y=1}^{k} e^{w_y^T x_i}}, \tag{1}$$

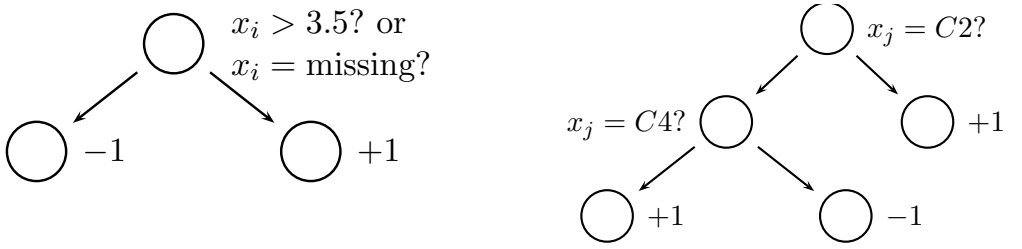
where k is the number of classes ($k = 4$ here), and $\bar{y}_i = \{1,2,3,4\}$ indicates the transformed class of x_i. We consider an l1-regularized solver here because it is suitable for noisy datasets. We choose the maximum entropy model since it delivers the probabilistic interpretation of data. To predict the probability of a test instance $\bar{x}$, we use:

$$p(\bar{y}|\bar{x}) = \frac{e^{w_{\bar{y}}^T \bar{x}}}{\sum_{y=1}^{k} e^{w_y^T \bar{x}}}. \tag{2}$$

Several techniques are available to solve (1). We use a coordinate descent method under the LIBLINEAR framework (Fan et al., 2008).

2.2.2 ADABOOST WITH A HETEROGENEOUS BASE LEARNER

We design a heterogeneous base learner and couple it with AdaBoost (Freund and Schapire, 1997). The advantage of this method is that it does not need to make any assumption on the values of the missing features. Our proposed base learner combines two different base learners: a categorical tree classifier and a numerical tree classifier. Figure 1 shows an example of the numerical and the categorical tree classifiers.



(a) Numerical tree classifier

(b) Categorical tree classifier

Figure 1: Example of numerical and categorical tree classifiers

Our AdaBoost model utilizes the base learner to form a more accurate predictive model by calling the learner repeatedly on different distributions over the training instances. At each iteration, our model searches for all features and selects one (numerical or categorical) feature to grow a (numerical or categorical) tree. During training, the numerical base learner tries to identify a threshold of the numerical feature, and the nodes in the categorical tree is branched according to if a certain category is matched or not. The missing value is also treated as a category. Furthermore, we have observed that some of the categorical features contain lots of distinct categories. For example, both the $14,784$-th and $14,868$-th features in the large version contain 15,416 distinct categories. The tree classifier trained on these highly-branching features tends to overfit the data. Thus, we decide to regularize the model complexity of the categorical tree by setting a parameter B that limits the maximal height of the tree.

Our classifiers select features inherently in the manner of wrapper methods. Nevertheless, to speed up the training of AdaBoost, we calculate the F-scores of the original features to pre-select a feature subset from the large version as the actual training inputs.

2.2.3 SELECTIVE NAÏVE BAYES

Another classifier we exploit is the selective naïve Bayes (SNB) (Boullé, 2007). There are three steps in SNB. First, the Minimum Optimized Description Length criterion is applied to perform numerical feature value discretization and categorical feature value grouping (Hue and Boullé, 2007). Second, a heuristic search with the MODL criterion is utilized to select a subset of features. Finally, a naïve Bayes classifier is used to classify the examples.

2.3 Post-processing

As discussed in Subsection 2.2.1, each instance is associated with at most one positive label among the three tasks (churn, appetency or upselling). Therefore, we decide to add a post-processing procedure to adjust the classifier's prediction scores if an instance receives high scores for more than one task. We exploit a linear support vector machine for the post-processing. For each classifier, we use the normalized log-transformed prediction scores, the

entropy of the score (i.e., assuming a multi-class problem with four outcomes) and the predicted rank within each task as the features. We then build a post-processing model with the actual label as the target for each task. The adjusted scores are used to form the final ensemble.

3. Experiments

We discuss model selection and present experimental results.

3.1 Model Selection and Analysis of the 10% Testing Data

Machine learning classifiers are sometimes sensitive to parameters. We select them using cross-validation on the training set. However, prediction results on the 10% test data are worse than cross-validation results. For example, Table 1 shows that the test AUC is drop notably on two classifiers (LIBLINEAR and RankLR, which will be described in Section 4.1). As we can submit various models to the competition website and improve the testing result, we must decide if fitting the 10% test data is beneficial or not.

We design an experiment with RankLR to analyze the 10% test set:

1. We train a randomly selected subset of 40,000 samples and obtain the weight vector w.
2. Use the w to predict the scores of the test set and submit them to the website to get AUC(D_{test}). Here D_{test} denotes the test set.
3. Randomly select 5,000 examples denoted as the set (D_{val}) from the remaining 10,000 samples for validation and use the same w to calculate AUC(D_{val}).
4. Repeat step 3 for 1,000 times and obtain 1,000 values of AUC(D_{val}).

Table 1: AUC Results on the validation set and the website

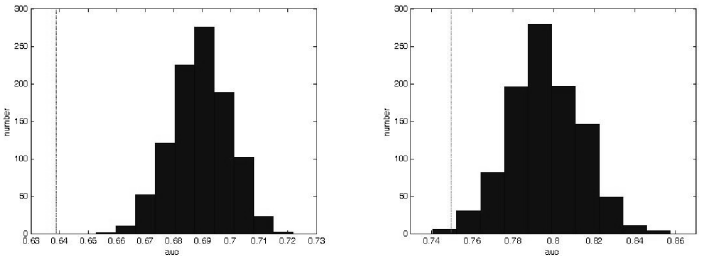
	Churn		Appetency		Upselling	
	on validation	on website	on validation	on website	on validation	on website
RankLR	0.7174	0.6553	0.8367	0.7562	0.8977	0.8753
LIBLINEAR	0.7539	0.6970	0.8706	0.8442	0.9033	0.8735

Figure 2 shows the histogram for the distribution of AUC(D_{val}) of each task. The vertical axis indicates the number of times that AUC(D_{val}) falls into each interval. The website results AUC(D_{test}) of churn, appetency and upselling are the dashed lines in the graph and are 0.6388, 0.7498 and 0.8682, respectively. We see that most AUC(D_{val}) values are higher than AUC(D_{test}). Thus the 10% test set might be quite different from the rest 90%. Therefore, we decide not to overfit the 10% test set but use parameters from the cross-validation procedure. Interestingly, we have learned that such a difference is not as significant for the other classifiers we have tried, and eventually we did not include RankLR and LIBLINEAR in the final ensemble.

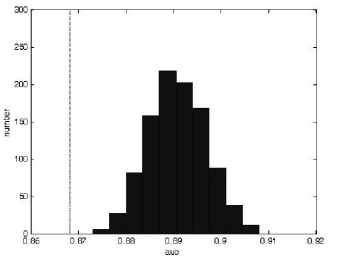
Parameters used in the post-processing stage are also chosen by cross-validation.

3.2 Experimental Results

Our experimental results are summarized in Table 2. We observe that cross-validation AUC on the training data is similar to the result on the test data, which is released after the competition ends. Each classifier performs differently on the three tasks and none is apparently superior to



(a) Churn (b) Appetency



(c) Upselling

Figure 2: Histogram for the distribution of AUC(D_{val})

the others. We combine the best two models for each task (according to the cross-validation results) by averaging the rank of their prediction scores as the merged model shown in Table 2. We also observe that the performance on the appetency task improves significantly after post-processing. The merged model with post-processing produces our best result and is selected as the third place in the slow track of KDD Cup 2009.

Table 2: Performance (AUC) of single classifiers and the merged model using cross-validation and the test set for the three tasks.

	Churn		Appetency		Upselling		
Base Classifier	CV	Test	CV	Test	CV	Test	Score
Maximum Entropy	0.7326	0.7428	0.8669	0.8786	0.9001	0.8975	0.8396
Heterogeneous AdaBoost	0.7350	0.7395	0.8660	0.8623	0.9030	0.9021	0.8347
Selective Naïve Bayes	0.7375	0.7428	0.8560	0.8529	0.8593	0.8564	0.8174
Merged Model (w./o. PP)		0.7557		0.8671		0.9036	0.8421
Merged Model (w. PP)		0.7558		0.8789		0.9036	0.8461
The winner of the slow track		0.7570		0.8836		0.9048	0.8484

4. Discussion

In this section, we discuss some other ideas and interesting findings.

4.1 RankLR

It is known that maximizing AUC is equivalent to maximizing the pair-wise ranking accuracy (Cortes and Mohri, 2003). We tried Rank-Logistic-Regression (RankLR) which is similar to RankSVM (Herbrich et al., 2000) but replacing the SVM part by Logistic Regression. Let x_i denote an example with label $y_i = +1$ (positive) and x_j denote an example with label $y_j = -1$ (negative). Assume that there are N_+ positive examples and N_- negative ones. We solve the following optimization problem.

$$\min_w \qquad E(w) = \frac{1}{N_+ \times N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} E_{ij}(w)$$

$$\text{where} \qquad E_{ij}(w) = \log\left(1 + \exp\left(-w^T(x_i - x_j)\right)\right).$$

We use stochastic gradient descent (Zhang, 2004) which updates the weight vector w by

$$w \longleftarrow w - \eta \nabla E_{ij}(w), \text{ where } \eta \text{ is the learning rate.}$$

After training, $w^T x$ is taken as the scoring function. The results, however, are at best similar to the maximum entropy model. Hence, we do not include RankLR in the final ensemble.

4.2 Discovery of Missing Patterns

We find that the training and test instances can be categorized into 85 groups according to the pattern of their missing numerical features. All instances in a group share the same missing numerical features. The number of instances varies from group to group, and the largest group contains $13,097$ training instances and $13,352$ test instances. We can then train and test each group separately.

Based on this finding, we build a tree-based composite classifier with several group-specific classifiers. The composite classifier delegates each test instance to its group, and uses the group-specific classifier to make the prediction. Unfortunately, the performance of this composite classifier is not significantly better than other classifiers we have tried, and thus this classifier is not included in the final ensemble.

5. Conclusions

We combine three diverse classifiers and exploit their specialties to carefully design steps that deal with heterogeneous and partially missing data. We couple categorical and missing feature expansion with the linear model to fill in the missing values and to choose the numerical meaning of the categorical values automatically. We combine the heterogeneous information using AdaBoost with separate categorical and numerical decision trees that handles the missing values directly. We compress the numerical and the categorical features and discover their probabilistic relations with the selective naïve Bayes. Furthermore, the observation about the connections between different tasks are used both in model design and in post-processing. Finally, overfitting is carefully prevented during the individual training and the post-processing steps using cross-validation.

References

Marc Boullé. Compression-based averaging of selective naive bayes classifiers. *Journal of Machine Learning Research*, 8:1659–1685, 2007.

Corinna Cortes and Mehryar Mohri. Auc optimization vs. error rate minimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

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.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 1997.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.

Carine Hue and Marc Boullé. A new probabilistic approach in rank regression with optimal bayesian partitioning. *Journal of Machine Learning Research*, 8:2727–2754, 2007.

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning (ICML)*, 2004.

KDD Cup 2009 @ Budapest: feature partitioning and boosting

Miklós Kurucz Dávid Siklósi István Bíró Péter Csizsek
Zsolt Fekete Róbert Iwatt Tamás Kiss Adrienn Szabó
{MKURUCZ, SDAVID, IBIRO, CSIZSEK, ZSFEKETE, RIWATT, KISSTOM, ASZABO}@ILAB.SZTAKI.HU
Computer and Automation Research Institute of the Hungarian Academy of Sciences

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

Abstract

We describe the method used in our final submission to KDD Cup 2009 as well as a selection of promising directions that are generally believed to work well but did not justify our expectations. Our final method consists of a combination of a LogitBoost and an ADTree classifier with a feature selection method that, as shaped by the experiments we have conducted, have turned out to be very different from those described in some well-cited surveys. Some methods that failed include distance, information and dependence measures for feature selection as well as combination of classifiers over a partitioned feature set. As another main lesson learned, alternating decision trees and LogitBoost outperformed most classifiers for most feature subsets of the KDD Cup 2009 data.

Keywords: Feature selection, classifier combination, LogitBoost, Alternating Decision Trees.

1. Introduction

The KDD Cup 2009 task targeted for the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling). For 50,000 anonymous customers a small data set of 230 and a large of 15,000 features was provided; in this paper we describe our various successful and failed attempts mostly over the large data set.

Telephone customer behavior analysis appears less frequently in publications of the data mining community. Some exceptions include machine learning methods for churn prediction on real data (evolutionary algorithm, Au et al. (2003); classifier combination by linear regression, Wei and Chiu (2002); Naive Bayes, Nath and Behara (2003); graph stacking, Csalogány et al. (2007)). The area is explored in more depth in marketing research including small sample survey results of Kim and Yoon (2004) and rule extraction and behavior understanding over a small 21-feature data by Ultsch (2002). Of closest interest, Neslin et al. (2006) present the overview of a churn classification tournament very similar to the KDD Cup 2009 task but with emphasis also on managerial meaningfulness and model staying power.

The difference in the KDD Cup 2009 large data set compared to typical classification problems is the abundance of features. For this end we had prepared feature selection and partitioning methods before the training label release. By the lessons we have learned from Web

©2009 Kurucz, Siklósi et al

Spam Challenges (Siklósi et al. (2008)), we had expected that feature partitioning and classifier combination would perform better than global classifiers.

Due to the large number of features it was also clear that feature selection methods are required. Our best performing methods have turned out to be very different from those described in some well-cited surveys as e.g. by Dash and Liu (1997): feature evaluation could only be used for a weak pre-selection while wrapper methods failed due to slow convergence and overfitting. Our final feature selection method is the union of the best features selected by LogitBoost of Friedman et al. (2000) over the feature partition that we have originally devised for classifier combination.

Our classifier implementation choice was mostly determined by the possibilities of the machine learning toolkit Weka of Witten and Frank (2005) that has wide variety in logistic regression, decision tree, neural nets mostly considered applicable for churn prediction described e.g. in Neslin et al. (2006). In addition we tested the SVM implementation of Chang and Lin (2001) considered most powerful for several classification tasks as well as Latent Dirichlet Allocation, a dimensionality reduction and generative modeling approach by Blei et al. (2003) that is believed to work well for skewed features. In our experiments the ultimate method turned out to be the combination of LogitBoost Friedman et al. (2000) and ADTrees of Freund and Mason (1999).

Next we describe our method and experimental results in detail including some partial results that appeared promising but did not perform as expected. In Section 2.1 we describe the way we partitioned features by their global properties. Then we describe successful and unsuccessful attempts first for feature selection (Sections 2.2–2.3), then for classifier ensemble construction along with the detailed AUC evaluation in Sections 2.4–2.5.

2. Experiments and Results

We describe our experiments, but successful and unsuccessful, over the large data set of KDD Cup 2009. The sole exception of a small data set experiment is one described in Section 2.2. As the key step of our final result, we have managed to select a powerful small feature subset by a LogitBoost based method described in Section 2.3.

An important ingredient of our method consists of an expert partitioning of the feature set (Section 2.1). While in our initial plans described in Section 2.5, the purpose of this partitioning was to train separate models and combine them, this approach was outperformed by our final submissions in Section 2.4. The partitioning however proved useful for our feature selection procedure in Section 2.3.

For classification we applied Weka implementations with intensive parameter search. In order to avoid overfitting for the online feedback from the 10% test set, we typically we tested performance over a random 10% **heldout set** set aside for method combination and another 10% **validation set** internal testing.

The heldout and validation sets were fixed once for the entire experiment. These test typically performed 2-3% better than the on-line feedback but more or less kept the relative order of the predictors. Note that our final submission consisted of two classifiers combined by taking the average score. In this case the final training was performed over the entire training set, including both the heldout and the validation set.

In our experiments we managed to avoid overfitting for the feedback on the 10% test set: except for a single task (appetency with the difference in the number of LogitBoost iterations) among all of our submissions, the relative order over the 10% and 100% test set was identical. Up to now however we are not able to explain why the relative order compared to other teams

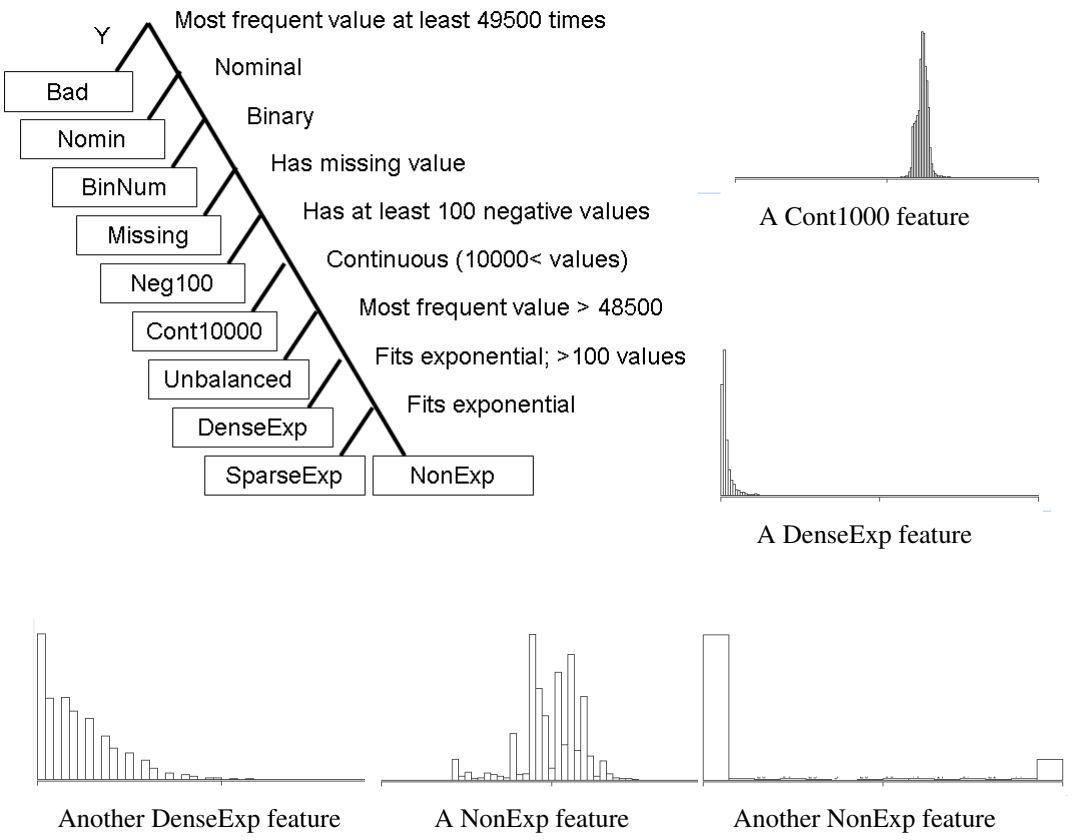


Figure 1: The feature partitioning tree obtained by investigating elementary properties, with some sample histograms. Over the tree the "yes" branch is always on the left side.

have changed; note that over 10% of the large test set our submission performed best with an AUC score 0.8457 while our final result is behind the winner by 0.0065.

We note that we did not apply feature preprocessing and did not try to capture the interaction effects between variables. While some internal tests involved cross-validation, due to time constraints we used our predefined heldout and validation sets since we observed no inconsistencies in their predicted performance.

We ran our tests on standalone multicore machines with more than 32GB RAM and a condor driven cluster of some older dual-core machines. We run in parallel different algorithms on different machines.

In what follows, in Section 2.1 we describe the expert feature partitioning. In Section 2.2 we briefly describe unsuccessful attempts for feature selection based on several methods generally considered effective in the literature as e.g. in the survey of Dash and Liu (1997). Our successful feature selection procedure is based on the expert partitioning and LogitBoost and is described in Section 2.3. The final submissions based on the combination of LogitBoost and ADTrees is found in Section 2.4. Finally in Section 2.5, we describe some unsuccessful attempts and in particular the most promising one consisting of a large classifier ensemble based on our expert feature partition. For reference, the parameters used in our procedures are summarized in Table 3 at the end of the paper.

2.1 Feature exploration and partitioning

As first step before test set release, we explored feature properties and partitioned the data. The partition is based on observable properties of the distributions that we have identified in the attempt of understanding the actual source of the features. Some sample histograms for a few less obvious classes along with the selection steps are found in Fig. 1. Our motivation of partitioning stems from our Web spam classifier of Siklósi et al. (2008) where for example content and link based features are best classified separately with the results combined by e.g. random forest. Although classification along the same lines as it will be described in Section 2.5 is outperformed by other methods, we used this partitioning for feature selection as described in Section 2.3.

The rules for defining the feature partition as also seen in Fig. 1 are the following.

Bad: the most frequent or missing value has frequency at least 49500 (10500).

Nomin: nominal with at least 500 non-missing values that is not Bad (269).

BinNum: numeric binary feature that is not Bad (1190).

Missing: numeric with missing values that is not BinNum or Bad (330).

Neg100: numeric with at least 100 negative values that is not Missing or Bad (85).

Cont10000: numeric with continuous range (at least 10,000 distinct values) that is not Missing or Neg100 (503).

Unbalanced: numeric with the most frequent value appearing at least 48500 times that is neither Bad, Missing nor Neg100 (540).

DenseExp: numeric with good fit to the exponential distribution that has more than 100 distinct values and neither Bad, Neg100, Cont10000, Missing or Unbalanced (530).

SparseExp: numeric with good fit to the exponential distribution that has at most 100 distinct values and neither Bad, Neg100, Cont10000, Missing or Unbalanced (445).

NonExp: numeric that is not Bad, Neg100, Cont10000, Missing, Unbalanced, DenseExp and SparseExp (587).

2.2 Feature selection

Our first feature selection attempt relied on well known feature evaluation methods such as Information Gain, Gain Ratio, and Chi Squared Probe. These methods turned out to be useful only as a pre-selection of still a relative large number of features and we have completely dropped this approach from consideration. Information Gain and Chi Squared Probe tends to overscore features with many unique values while Gain Ratio that normalizes scores proportional to the number of unique values tends to overscore features with few unique values. We observed the following problems with this selection:

- Non-predictive features were selected in a high number.
- The threshold to drop features was generally hard to decide and justify.
- These methods tended to select highly correlated features.

Our second attempt was the classifier-driven approach. One possibility is to start a random walk in the feature space, starting from a preselected feature set, and at each step choose a feature to add or drop. The evaluation of every feature set can be made by a given classifier.

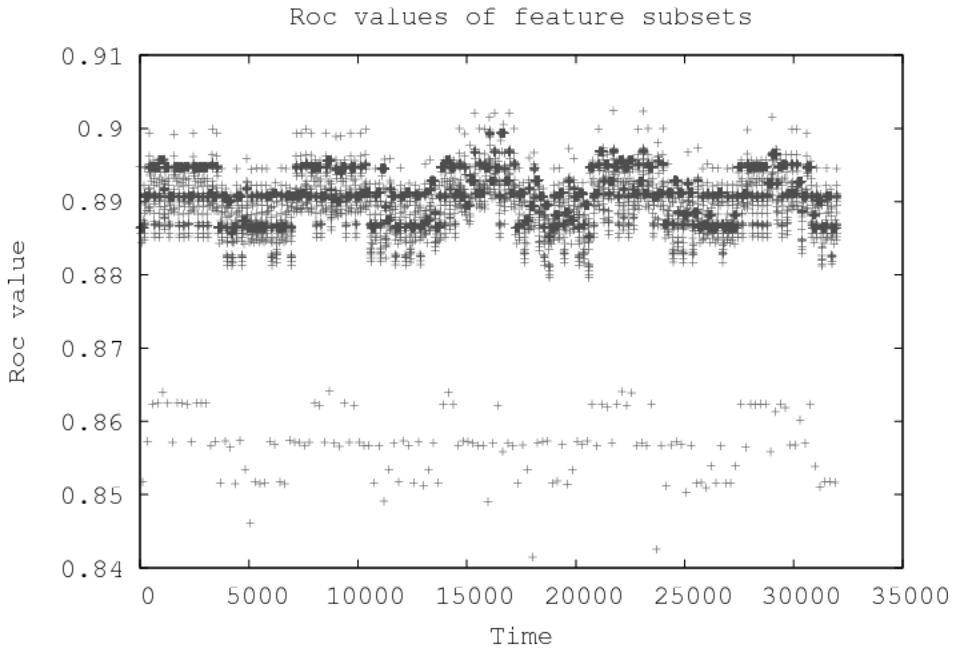


Figure 2: A sample 35,000 feature subset performance selected by our random walk wrapper method biased towards better AUC over the small data set.

Although this method ran efficiently in our parallel environment, it still took a long time to find a generally good feature set. In addition in our experiments the method was prone to overfitting: the difference between the exceptionally best and overall good feature subsets diminished when we switched between our heldout and validation sets. A sample run over 170 features of the small data set selected by feature evaluation is shown in Fig. 2.

2.3 The final feature selection procedures

For our final submissions, LogitBoost as feature selection proved to be the most effective method. For each partition of features from Section 2.1, after preselection by feature evaluation we run LogitBoost of Friedman et al. (2000) with Decision Stump base classifier. This composite classifier chooses only a few (in our case 10–60) features to build a model. We run LogitBoost for all partitions of Section 2.1 with the following parameters:

- We used 60 iterations of boosting;
- We used bagging over 90% of the data with 10 iterations;
- We selected the best 10 features; the actual figure varies as LogitBoost tends to select the best features more than once but bagging yields multiple sets;
- For the largest partition BinNum we selected 40 features instead of 10.

We created our feature set from the union of those selected over the partition. After this procedure we were left with 75 features for churn, 90 for appetency and 65 for upselling. These figures reflect the hardness of classifying appetency: in this case there were no real best features

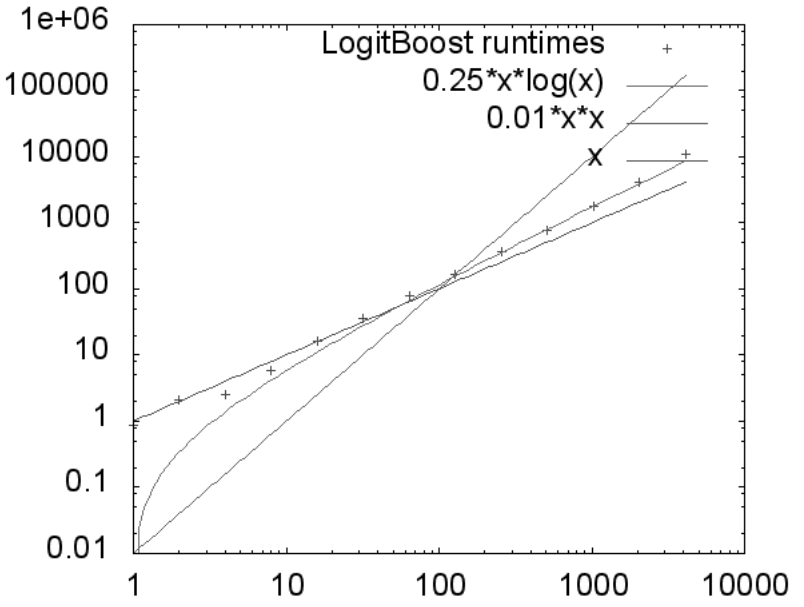


Figure 3: The measured running time of the LogitBoost implementation of Weka with decision stump as base classifier, 60 iterations, as the function of the number of features.

and LogitBoost selected the same feature multiple times less often than for the other two tasks. We remark that these final features still contained 3-4 features with both zero IG and χ^2 and these could have been removed with no performance difference.

One rationale for running feature selection separately for each partition is that the running time grew superlinearly with the number of features, as seen in Fig. 3. A linear fit to the log-log plot of the Figure produces a slope of 1.2911. Even without point before the leftmost, which is probably an outlier, the slope is 1.1338.

Another reason for the expert partitioning of the features is that we could even achieve marginal improvement (see Table 2 in Section 2.4) over random partitioning. For comparison we also tested a random partition of about 250-300 features in each set that could fit into single lower capacity machines of our cluster. Since the number of sets in the partition was large, we had to iterate selection until only a smaller number of approximately 200 features remained. Note that this method was also much slower since we could not count on an appropriate grouping of the features and we had to keep larger candidate subsets in intermediate steps. For this reason we did not even compute results for all tasks with this method.

We found that the two methods find almost the same features. We observed that the features with great imbalance are generally non-predictive: they were only exceptionally selected into the final feature sets.

We also identified some problems with our LogitBoost-based feature selection method:

- Some discarded features could have been useful for other classifiers—this is a common problem for classifier driven feature selection.
- Non-predictive features can still be selected, although unlikely.

Table 1: The AUC value of selected final methods over the test set. LogitBoost with and without ADTree is trained over the entire training label set and by using the features selected within each of the partitions of Fig. 1. The last entry consists of the combination of individual classifiers over this same partition as described in Section 2.5.

	churn	appetency	upselling	score
Slow Track Winner (U Melbourne)	0.7570	0.8836	0.9048	0.8484
LogitBoost + ADTree by partition (final)	0.7567	0.8736	0.9065	0.8456
Fast Track Winner (IBM Research)	0.7611	0.8830	0.9038	0.8493
LogitBoost by partition (final fast)	0.7496	0.8683	0.9042	0.8407
Combination LogitBoost	0.7409	0.8561	0.8894	0.8288

- Uneven distribution of predictive features in partitions may cause dropping some of them, although unlikely.

2.4 The final classifier ensemble

After the feature selection procedure of Section 2.3, over less than a hundred features, we used LogitBoost of Freund and Mason (1999) with decision stump as base classifier for the fast track and additionally ADTree classifiers of Friedman et al. (2000) for the slow track. Both classifiers were used with 10 iterations of bagging over 90% of the data points. Cost matrices improved performance for appetency only where the optimal false negative to false positive cost ratio was 30. The actual values were tuned over our predefined 10% heldout set. For a summary of parameters, see also Table 3.

We summarize our results for the KDD Cup large test set, both 10% and the whole, in Table 1. Classifiers over the best features selected over our expert partition performed best. The combination of LogitBoost and ADTree improved performance. Best combination turned out to be the plain average that constitutes our final submission of AUC 0.8456 ranking sixth in the competition.

2.5 Combination over the feature partition

In this section we describe our unsuccessful attempt of defining a classifier ensemble over our expert partition as well as the behavior of the individual feature subsets. We summarize the results for our 10% heldout and validation sets in Table 2. Note that over these sets the final ensemble method (top row of Table 2) outperformed LogitBoost on this set, but LogitBoost trained over the entire set performed better for the Cup test set as seen in Table 1.

For all feature subsets we have tested a variety of Weka classifiers, and libsvm with various kernels. The top part of Table 2 shows performance by using all features while the bottom part by using given subsets only. No classifier has managed to outperform LogitBoost for any of the feature subsets nor did they yield improvement in any combination.

Algorithm 1 Classifier ensemble method.

for all feature subsets **do**
  train models on training - heldout - validation (80%)
  tune parameters on the heldout set
  compute log-odds for the entire training set*
end for
combine over the validation set

Table 2: The AUC value of different classification methods over the whole set (top) and certain feature subsets (bottom) for the three subtasks.

	churn		appetency		upselling	
	heldout	valid	heldout	valid	heldout	valid
Ensemble combin. by LogitBoost		0.7667		0.8537		0.9100
LogitBoost by expert selection	0.7557	0.7649	0.8668	0.8509	0.9122	0.9099
LogitBoost random selection	0.7540	0.7612			0.9064	0.9069
Ensemble log-odds by LogitBoost		0.7583		0.8361		0.9026
Linear SVM	0.6764	0.7026	0.8028	0.7987	0.8845	0.8760
Ensemble combin. by BayesNet		0.7012		0.7905		0.8057
LDA with BayesNet	0.6008	0.6246	0.5995	0.6278	0.7598	0.7539
Churn predictor			0.5995			
Missing w/ LogitBoost	0.7232	0.7318	0.8394	0.8217	0.8855	0.8931
NonExp w/ AdaBoost	0.7188	0.7359	0.8551	0.8332	0.8835	0.8815
Nominal w/ LogitBoost	0.6657	0.6696	0.8385	0.7868	0.7623	0.7649
Cont10000 w/ LogitBoost	0.6465	0.6631	0.6564	0.6712	0.7419	0.7474
BinNum	0.6369	0.6187	0.7204	0.7233	0.8016	0.8126
DenseExp w/ LogitBoost	0.6294	0.6473	0.6398	0.6591	0.7251	0.7391
NonExp w/ Bayes	0.6230	0.6531	0.5870	0.6393	0.7330	0.7224

We describe the best performing ensemble combination method in Algorithm 1. The optional line marked with $*$ corresponds to the log-odds based combination proposed by Lynam et al. (2006). The results correspond to the first and fourth rows of Table 2 that show a surprisingly deteriorated performance of log-odds. In these cases the heldout set was used for combination and hence results for the validation set are given. Ensemble combination with BayesNet performed much worse.

In comparison to the ensemble methods, the second and third rows of Table 2 show a single LogitBoost classifier applied to the features of the two different successful feature selection methods (Section 2.3). Note that in the final submission, we trained LogitBoost over the entire test set that changed the relative order of the two methods for the final submissions in Table 1.

Among the individual feature subsets surprisingly those with missing values performed best. In our guesses these may include responses to questionnaire or call center operators with predictive value stronger than generated features based on service usage. These were followed by the "regular" numeric features with non-exponential distribution, the only exception in that here AdaBoost performed better than LogitBoost. Other classifiers performed much worse for all subsets.

Global classifiers such as BayesNet of linear SVM performed poor. Latent Dirichlet Allocation for dimensionality reduction as in Bíró et al. (2009) performed surprisingly weak as well.

The only successful feature evaluation method, in the sense of Section 2.2, turned out AUC itself. In this run we passed 95 features with best individual AUC to LogitBoost.

As a final unsuccessful trial we experimented with using training data from one task to improve prediction for the other. A simplest example is the application of the churn predictor for appetency in Table 1. The rationale is that a user who churns will not buy upgrades and vice verse. We have also observed that positive labels were disjoint for the three tasks. We tested two methods but could not improve our results:

Table 3: The AUC value of different classification methods over the whole set (top) and certain feature subsets (bottom) for the three subtasks.

parameter	value
iterations, LogitBoost	60
bagging data fraction	90%
bagging iterations	10
final feature set size, churn	75
final feature set size, appetency	90
final feature set size, upselling	65
FN/FP cost ratio, appetency	30

Classifier combination: We used the results of several final and partial classifiers across tasks in combination. Note that the AUC of one classifier for another task never reached even close to 0.6 and hence failure is no surprise.

Case weighting: If we classify appetency, those who churn are "more negative" than those who just do not buy new services. For a decision stump classifier we may introduce three classes and use a cost matrix with penalties higher for classifying churned customers positive for appetency than non-churned negatives. In this way decision stump acts as regression for the outcome 1 for appetency, 0 for no appetency, and a larger negative value for churn.

Conclusion

In our experiments we observe a clear gain of two classifiers, LogitBoost with decision stump and ADTrees. LogitBoost also performed well for feature selection. We used a feature partitioning method based on statistical properties. The combination of the classifiers over this partition performed close to best (in some cases even better on our heldout sets) and thus we believe that a partition relying also on the meaning of the features (e.g. traffic, sociodemographic or neighborhood based) may outperform the blind anonymous classifiers of the Cup participants. We also expect the call graph extracted from the call detail record can boost performance via the graph stacking framework as e.g. in Csalogány et al. (2007).

Acknowledgments

This work was supported by the EU FP7 project LiWA—Living Web Archives and by grant OTKA NK 72845.

References

Wai-Ho Au, Keith C. C. Chan, and Xin Yao. A novel evolutionary data mining algorithm with applications to churn prediction. *IEEE Trans. Evolutionary Computation*, 7(6):532–545, 2003.

István Bíró, Dávid Siklósi, Jácint Szabó, and András A. Benczúr. Linked latent dirichlet allocation in web spam filtering. In *AIRWeb '09: Proceedings of the 5th international workshop on Adversarial information retrieval on the web*. ACM Press, 2009.

D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(5):993–1022, 2003.

C.C. Chang and C.J. Lin. LIBSVM: a library for support vector machines, 2001.

Károly Csalogány, A.A. Benczúr, D. Siklósi, and L. Lukács. Semi-Supervised Learning: A Comparative Study for Web Spam and Telephone User Churn. In *Graph Labeling Workshop in conjunction with ECML/PKDD 2007*, 2007.

M. Dash and H. Liu. Feature selection for classification. *Intelligent data analysis*, 1(3):131–156, 1997.

Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *In Machine Learning: Proceedings of the Sixteenth International Conference*, 1999.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of statistics*, pages 337–374, 2000.

H.S. Kim and C.H. Yoon. Determinants of subscriber churn and customer loyalty in the Korean mobile telephony market. *Telecommunications Policy*, 28(9-10):751–765, 2004.

T.R. Lynam, G.V. Cormack, and D.R. Cheriton. On-line spam filter fusion. *Proc. of the 29th international ACM SIGIR conference on Research and development in information retrieval*, pages 123–130, 2006.

S.V. Nath and R.S. Behara. Customer churn analysis in the wireless industry: A data mining approach. *Proceedings of the 34th Annual Meeting of the Decision Sciences Institute*, 2003.

S.A. Neslin, S. Gupta, W. Kamakura, J. Lu, and C.H. Mason. Defection detection: Measuring and understanding the predictive accuracy of customer churn models. *Journal of Marketing Research*, 43(2):204–211, 2006.

Dávid Siklósi, András A.Benczúr, István Bíró, Zsolt Fekete, Miklós Kurucz, Attila Pereszlényi, Simon Rácz, Adrienn Szabó, and Jácint Szabó. Web Spam Hunting @ Budapest. In *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2008.

A. Ultsch. Emergent self-organising feature maps used for prediction and prevention of churn in mobile phone markets. *Journal of Targeting, Measurement and Analysis for Marketing*, 10(4):314–324, 2002.

Chih-Ping Wei and I-Tang Chiu. Turning telecommunications call details to churn prediction: a data mining approach. *Expert Syst. Appl.*, 23(2):103–112, 2002.

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005. ISBN 0120884070.

JMLR Workshop and Conference Proceedings 7:77–88 KDD cup 2009

Logistic Model Trees with AUC Split Criterion for the KDD Cup 2009 Small Challenge

Patrick Doetsch PATRICK.DOETSCH@RWTH-AACHEN.DE
Christian Buck CHRISTIAN.BUCK@RWTH-AACHEN.DE
Pavlo Golik PAVLO.GOLIK@RWTH-AACHEN.DE
Niklas Hoppe NIKLAS.HOPPE@RWTH-AACHEN.DE
Michael Kramp MICHAEL.KRAMP@RWTH-AACHEN.DE
Johannes Laudenberg JOHANNES.LAUDENBERG@RWTH-AACHEN.DE
Christian Oberdörfer CHRISTIAN.OBERDOERFER@RWTH-AACHEN.DE
Pascal Steingrube PASCAL.STEINGRUBE@RWTH-AACHEN.DE
Jens Forster JENS.FORSTER@RWTH-AACHEN.DE
Arne Mauser ARNE.MAUSER@RWTH-AACHEN.DE

Lehrstuhl für Informatik 6
RWTH Aachen University, Ahornstr. 55, 52056 Aachen, Germany
Human Language Technology and Pattern Recognition

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

Abstract

In this work, we describe our approach to the "Small Challenge" of the KDD cup 2009, a classification task with incomplete data. Preprocessing, feature extraction and model selection are documented in detail. We suggest a criterion based on the number of missing values to select a suitable imputation method for each feature. Logistic Model Trees (LMT) are extended with a split criterion optimizing the Area under the ROC Curve (AUC), which was the requested evaluation criterion. By stacking boosted decision stumps and LMT we achieved the best result for the "Small Challenge" without making use of additional data from other feature sets, resulting in an AUC score of 0.8081. We also present results of an AUC optimizing model combination that scored only slightly worse with an AUC score of 0.8074.

Keywords: KDD cup, Area Under Curve, Logistic Regression, Boosting, Decision Trees, Model Combination, Missing Values, Imbalanced Data

©2009 P. Doetsch, C. Buck, P. Golik, N. Hoppe, M. Kramp, J. Laudenberg, C. Oberdörfer, P. Steingrube, J. Forster, A. Mauser

1. Introduction

In the following we describe our approach to the "Small Challenge" of the KDD cup 2009. The task was to predict customer behavior for a mobile phone network provider. While exploring several classification and regression methods, we focused on tree-based models. Our most successful approaches were a variant of Logistic Model Trees (Landwehr et al. (2005)) and boosted decision stumps (Schapire and Singer (2000)).

Our final submission used these classifiers in combination and was ranked 35th in the slow challenge and was the best solution that did not use data from other data sets to aid classification ("unscrambling").

Our team was formed within a student Data Mining lab course at the Chair for Computer Science 6 of RWTH Aachen University. Eight students took part in the lab.

This paper is organized as follows: We briefly describe the task of the KDD cup 2009 in Section 2. In Section 3 we explain our preprocessing steps. In Section 4 the classification methods are described, while the model combination techniques are described in Section 5. Experimental results are given in Section 6. Finally, conclusions are drawn in Section 7.

2. Task

The task of the KDD cup 2009 was to improve marketing strategies in the context of customer relationship management using data provided by the French telecom company Orange. Three binary target variables had to be predicted: Churn (propensity of customers to switch providers), Appetency (propensity of customers to buy new products or services) and Up-selling (propensity of customers to buy upgrades or add-ons). The challenge was split into a fast and a slow track. The fast track required predictions to be submitted after 5 days, the slow track lasted for one month. For the slow track there were two separate data sets with different number of features: a small data set with 230 features (190 numerical and 40 categorical) and a large data set with 15,000 features. Both sets contained 50,000 labeled train instances and 50,000 unlabeled test instances. We participated in the slow track and used only the small data set.

Neither the feature names nor the values allowed any intuitive interpretation of the data. In 211 out of 230 features values were missing and 18 features contained no values at all. Since the target classes were disjoint in the training data, we reformulated the classification task as a four-class problem. Experimental results have shown an average improvement of about 0.015 AUC score compared to performing three binary predictions separately. The class distribution of the data was imbalanced. Only 16.5% of the training instances were labeled positive in one of the three classes and 1.8% of the data belonged to the class Appetency.

Evaluation The Area Under the Curve (AUC) was the evaluation criterion of the predicted ranking. According to Bradley (1997), this ranking measure is defined as the area under the curve obtained by plotting the specificity against the sensitivity. Sensitivity is the percentage of correctly classified positives, whereas specificity is the percentage of correctly classified negatives.

The organizers of the KDD cup provided an opportunity to upload the predicted rankings during the cup to check performance on 10% of the test data. For internal evaluation and parameter tuning we split the training data as described in the following section.

3. Preprocessing

3.1 Cross-validation

The full training set consisting of 50,000 data samples was split into a holdout set of 10,000 samples and a training set of 40,000 samples for internal evaluation. For training and optimization we used a 5-fold cross validation. Each fold consisted of 32,000 training and 8,000 test samples. The AUC score on the test part of the individual folds was averaged and used as evaluation criterion. We used the cross validation data to optimize the parameters of the algorithms and the holdout set to evaluate this parametrization on unseen data.

3.2 Missing Values

A large number of values was missing in the data. 18 features contained no values at all and for five features only one value was observed. These features were discarded. Among the remaining 207 features only 19 were fully observed. Features with less than 5% of missing values were handled by inserting the mean or mode for numerical features or the most frequent value for categorical features. We divided the remaining features into three categories and handled each category separately.

Selection Criterion To select an adequate method to address the missing values for the features with different fractions of missing values, we introduced the following measure: Given a set of features $f_i, \ldots, f_I$, let f_i be a feature with missing values and M_{f_i} the number of distinct feature values in f_i. Further assume that there are N_{f_i} observations in our data set where the value of this feature is not missing. Then we define the Missing Value Ratio (MVR) as $N_f/(M_f \cdot (I-1))$, where I is the total number of features. The value of $\mathrm{MVR}(f_i)$ gives the average number of samples per possible target value and feature available to predict missing values in f_i. A small value of $\mathrm{MVR}(f_i)$ indicates that it will be difficult to predict the feature from the given data. We use the MVR to divide features into three categories. 37 features with $\mathrm{MVR}(f) \geq 3$ belong to category A, 25 features with $1 \leq \mathrm{MVR}(f) < 3$ belong to category B and the remaining 126 features belong to category C. These thresholds were chosen based on initial experiments and experience from other tasks. Most features are contained in the category C. These features cannot be addressed by simple imputation techniques, so we tried different algorithms to estimate the values as described below.

Imputation Methods Missing values can be treated as a prediction problem, so we formulated the search for the missing values for each category as classification problem or as regression problem. All samples where the corresponding feature is not missing were used as training data to impute the values of this feature in the remaining samples. The algorithms we used to solve this classification or regression task were all parametrized. To optimize these parameters empirically, we created a development set for each feature. This set contained about 5% of the data where the particular feature is not missing.
Category A features provided a sufficiently large number of occurrences per value and we classified their missing values with support vector machines using an RBF kernel and cost penalties between 1 and 2. Category B and C were imputed by a regression tree implemented in MATLAB. We declared all features with less than ten distinct feature values to be categorical. The implementation used an entropy split criterion and the minimum number of observations for splitting a node was set to 200.

For category C features, we additionally used the full information maximum likelihood method (FIML, Myrtveit et al. (2001)), where the missing values are replaced by estimating

a normal distribution. To choose between different imputation methods for each feature from category C we used ranking methods described in Subsection 3.3.

Feature Extraction Beside the features created by the imputation methods described above, we produced additional features, such as flags for missing values or features describing the length of the textual values. Class-specific thresholding of features yielded useful binary indicators. For instance, the value of -30 in feature 126 supported Up-selling. While the features improved the performance of log-linear models and neural networks by up to 10% relative, tree induction methods were able to find such relations by themselves. We produced 5,318 new features in this way. Since most of our classifiers only support numerical input, categorical ones had to be binarized, resulting in more than 10,000 features in total.

3.3 Feature Selection

In order to build feature sets with a high predictive potential we analyzed correlation matrices, performed forward selection and backward elimination of features, and implemented two ranking algorithms to create class-specific feature sets.

For one, we used a ranking based on information gain ratio (Quinlan (1986)) in order to find most relevant features. In addition, we used the likelihood-ratio test (Duda et al. (2000)) as a ranking criterion. We only looked at features with a class-dependent support of more than 100 within the first cross-validation fold. The value of -30 in feature 126 mentioned above given the Up-selling class label had the highest likelihood-ratio. Although this approach led to good binary features, the tree-based learning algorithms described in Section 4 were able to extract those indicators automatically.

With these selection techniques we produced a feature set consisting of the 206 features from the original set described above with imputed missing values and additionally a selection of highly ranked features. In the following this set will be denoted by X_O, while the same features without imputation of missing values will be denoted by X_M.

4. Modeling

Given the setup described in Section 3 we evaluated the performance of various standard classifiers. This included parameter optimization as well as the selection of an appropriate set of features and preprocessing steps for each classifier. We briefly describe the used classifiers in this section.

4.1 MLP

One of our first experiments on the KDD cup 2009 task was done using multilayer perceptrons (MLP) implemented in the Netlab (Nabney (2001)) toolbox with GNU Octave. The MLPs provided one input node for each feature, one layer of hidden nodes and four output nodes, one for each class. Using the logistic activation function in the nodes, the MLPs were trained with different numbers of hidden nodes and training iterations. Due to the nature of the nonconvex optimization problem and random initialization of the weights, 10 runs with identical parameter sets were made in order to find different local optima. Subsequently, the outputs of these runs were averaged class-wise to construct the final outputs.

4.2 SVM

Several methods to incorporate the AUC as an optimization criterion in SVMs have been proposed with implementations available online. We used SVMperf (Joachims (2005)), an implementation that can optimize a number of multivariate performance measures including AUC. Due to the large amount of generated features all experiments were done using a linear kernel.

4.3 LMT

We used our own implementation of the Logistic Model Tree (LMT) (Landwehr et al. (2005)), which is available for free download on the website of our chair[1]. In this algorithm a tree is grown similar to the C4.5 approach where each node estimates a LogitBoost model (Friedman et al. (1998)) on the assigned data, i.e. it performs an iterative training of additive logistic regression models. At each split, the logistic regressions of the parent node are passed to the child nodes. The final model in the leaf nodes accumulates all parent models and creates probability estimates for each class. After growing the whole tree, pruning was applied to simplify the model and to increase the generalization capability.

The LogitBoost algorithm creates an additive model of least-squares fits to the given data for each class c, which has the following form:

$$L_c(x) = \beta_0 + \sum_{i=1}^{F} \beta_i x_i$$

where F is the number of features and β_i is the coefficient of the ith component in the observation vector x. The posterior probabilities in the leaf nodes can then be computed by the method of linear logistic regression (Landwehr et al. (2005)):

$$p(c|x) = \frac{\exp\left(L_c(x)\right)}{\sum_{c'=1}^{C} \exp\left(L_{c'}(x)\right)}$$

where C is the number of classes and the least-squares fits $L_c(x)$ are transformed in a way that $\sum_{c=1}^{C} L_c(x) = 0$.

For the task of the KDD cup 2009 we modified the algorithm to use AUC as split criterion. Ferri et al. (2003) introduced the AUC split criterion for decision trees, where each leaf is labeled by one class. Each possible labeling corresponds to one point on the curve obtained by plotting the specificity against the sensitivity. The search for a split point then corresponds to finding an optimal labeling of the resulting leaves. This can be done efficiently, as shown by Ferri et al. (2002): Let N_l^c be the number of training examples in leaf l assigned to class c. For each pair of classes c_i and c_j, $i \neq j$, we define the local accuracy (LA) in leaf l as:

$$\mathrm{LA}_l(c_i, c_j) = \frac{N_l^{c_i}}{N_l^{c_i} + N_l^{c_j}}$$

Given the two classes c_i, c_j and a possible split point S with the corresponding leaves l_{left} and l_{right}, we calculate the LA for each leaf and sort them by this value in decreasing order. According to Ferri et al. (2003), this ordering creates the path of labelings on the curve obtained by plotting the specificity against the sensitivity. Finally the metric defined by Ferri et al. (2002) is used to compute the AUC:

$$\mathrm{AUC}_S(c_i, c_j) = \frac{1}{2QR}\left(N_1^{c_i} N_1^{c_j} + N_2^{c_i}(2N_1^{c_j} + N_2^{c_j})\right) \tag{1}$$

1. http://www-i6.informatik.rwth-aachen.de/web/Teaching/LabCourses/DMC/lmt.html

where $Q = N_1^{c_i} + N_2^{c_i}$ and $R = N_1^{c_j} + N_2^{c_j}$. We select the split point that maximizes Eq. (1) averaged over all class pairs. Our implementation of the LMT supported only numerical features, so each split always generated two child nodes and there were no leaves which did not contain any observation. Therefore, Eq. (1) considers only the two distinct labelings of these two child nodes, while Ferri et al. (2002) give a general form of this equation.

To increase the generalization ability of the model, pruning was applied to the fully grown tree. We used a two fold cross validation within the tree to calculate the pruning values for each node. The pruning procedure originally calculated the pruning value for a node based on the tree error of its subtrees. For this purpose the tree is grown using one fold of the internal cross validation. Then the corresponding test set is evaluated and each node saves the number of samples misclassified by its LogitBoost model as local error E_L. If v is a node of the LMT and v_{left} and v_{right} are its child nodes, the tree error E_T of v is $E_L(v)$ if v is a leaf and $E_T(v_{\text{left}}) + E_T(v_{\text{right}})$ otherwise.

Finally, the pruning values are calculated by the ratio of the local error and the tree error in a node. This was modified for the AUC criterion, but we did not observe an improvement using this pruning procedure. Unpruned trees with about 7,500 observations in each leaf also led to comparable results to pruned trees, as shown in Section 6. The computational complexity of building a logistic regression model is quadratic in the number of features F. The complexity of building an LMT is $O(NF^2d + v^2)$, where N denotes the number of training samples, d is the depth of the tree, and v the number of nodes in the unpruned tree. Note that the internal K-fold cross-validation generates a large constant factor in the complexity, such that training unpruned trees is about K times faster.

4.4 Boosted Decision Stumps

Since boosting and tree induction methods performed well on the given task, we also made experiments with boosted decision stumps. For this purpose we used BoosTexter, which was developed for text categorization and is a particular implementation of the AdaBoost algorithm (Freund and Schapire (1999)) with decision stumps as weak learners. In our experiments we used the Adaboost.MH version of the algorithm. For categorical features, the weak learners pose simple "questions" whether the given feature value occurs in the sample or not. As in the LMT, numerical features are thresholded by a split point. Missing values are ignored in this procedure. For each split point we compute weights W_+^{lc} and W_-^{lc} for each class c and leaf l generated by that split, where $+$ and $-$ indicate whether the observations in leaf l belong to class c or not according to these weights. As usual, the computation is based on the distribution estimated by the boosting procedure (Eq. (4) in Schapire and Singer (2000)). Given these weights, we select the split point minimizing $2\sum_l \sum_{c=1}^{C} \sqrt{W_+^{lc} W_-^{lc}}$, where C is the number of classes. For details and other versions of the AdaBoost algorithm implemented in BoosTexter, see Schapire and Singer (2000).

The algorithm creates a ranked scoring of the classes for a given test sample. The absolute value of this score can be interpreted as unnormalized "confidence" measure of the prediction. A negative score indicates that the sample does not belong to the specific class.

In our experiments we used a reimplementation called icsiboost[2]. The implementation is able to handle categorical features, numerical features, and missing values in the weak learners as described above. The AUC evaluation could directly be applied to the class scores produced by the implementation, but for the combination methods described in Section 5 we normalized them to obtain posterior probabilities. Therefore, a sigmoid function is fit to the resulting scores. Let m be the number of iterations performed during training and $\text{score}(x,c)$ the output of the

2. http://code.google.com/p/icsiboost

classification procedure for the test sample x and class c. Then the posterior probability defined by Niculescu-Mizil and Caruana (2005) is given by

$$p(c|x) = \left(1 + \exp\left(-2m \cdot \text{score}(x,c)\right)\right)^{-1}$$

Boosted decision stumps performed best on all 206 features described in Section 3 with no further preprocessing and a selection of binary features (see Section 6 for details). After about 25 iterations the AUC converged, while the test error still decreased. A direct optimization on AUC was not implemented. The algorithm can be implemented with a time-per-round complexity of $O(NC)$, so it is linear in the number of classes C and training samples N.

5. Combinations

We combined the approaches described in Section 4 to improve the overall result. Simple accumulation of multiple predictions showed no improvements in performance compared to the results obtained by boosted decision stumps. Thus we implemented two other combination methods, which are presented in this section.

Rank combination In order to aggregate different models, we transformed their output into a new ranking $r(c|x)$. We first assumed that the predictions obtained from a certain classifier k for an observation x and class c could be interpreted as posterior probabilities $p_k(c|x)$. While this is not true for e.g. SVMs, it enabled us to compute the average of posteriors as a baseline for classifier combination.

The posteriors can also be interpreted as votes for a relative ranking. $p_k(c|x) > p_k(c|y)$ would indicate one vote for x to be ranked higher than y. By counting all votes and normalizing for each classifier we can derive a total ranking by accumulating all votes for each observation x:

$$r_{\text{vote}}(c|x) = \frac{1}{|K|} \sum_k \frac{1}{Z_k} \sum_y \delta\left(p_k(c|x) - p_k(c|y)\right)$$

where Z_k is a normalization constant so that $\max_x 1/Z_k \sum_y \delta\left(p_k(c|x) - p_k(c|y)\right) = 1$ and $\delta(x)$ is 1 if $x > 0$ and 0 otherwise. The normalization is needed to give equal influence to each classifier and not to penalize predictions with few distinct values. Both of the methods described above can be extended to a weighted linear combination by using a weight w_k for each classifier k. We implemented a general gradient descent and the downhill simplex method by Nelder and Mead (1965) to optimize the weights w_k with respect to the AUC evaluation criterion.

Stacking As further combination approach we used the outputs of one classifier as features for another classifier (Smyth and Wolpert (1999)). We created stacking features from boosted decision stumps described in Section 4. With these features we were able to improve the raw results of boosted decision stumps with an LMT, which is presented in the next section. In the following, the feature set X_O extended by these additional stacking features will be denoted as X_S.

6. Results

All results were averaged over all folds of our 5-fold cross validation. The feature sets were optimized for each classifier. Table 4 shows the comparison of their performance on a common feature set.

Table 1: AUC scores of boosted decision stumps on the X_M set

Features	Missing values	Appetency	Churn	Up-selling
original	imputation methods	0.7928	0.6389	0.8401
original	internal handling	0.8013	0.6758	0.8503
original + binary	imputation methods	0.7913	0.6452	0.8425
original + binary	internal handling	**0.8024**	**0.6945**	**0.8538**

Table 2: AUC scores of Logistic Model Trees on the X_O set

Features	Configuration	Appetency	Churn	Up-selling	Average
original	entropy	0.7578	0.6857	0.7565	0.7333
	entropy + pruning	0.7595	**0.6894**	0.7555	0.7348
	AUC	0.7564	0.6834	**0.8443**	0.7614
	AUC + pruning	**0.7844**	0.6803	0.8417	**0.7688**

MLP As described in Section 4, one of our first attempts were MLPs. The number of hidden nodes and number of iterations maximizing the average AUC for all classes and folds were found by grid search. The highest AUC score for Appetency was achieved with 100, for Churn with 300, and for Up-selling with 500 hidden nodes. This behavior indicates different complexities for the classification on each separate class. The best result with an AUC score of 0.78 was achieved on the data set X_O with a selection of binary features. Therefore, MLPs were discarded as a single approach for the KDD cup 2009 task.

Boosted decision stumps As starting point for the best of our final submissions we used boosted decision stumps, described in Section 4. In the experiments presented in Table 1 we declared all features with less than 10 distinct values as categorical and performed 25 boosting iterations. In our first attempt we did not apply any missing value imputation methods, since the algorithm is able to handle them internally. We repeated the experiment with the same feature set, but missing values were imputed by the techniques described in Section 3. Table 1 shows the class-specific results. Furthermore, we found binary features as described in Section 3 which were able to improve the resulting AUC.

Table 1 shows that the internal handling of missing values outperforms our imputation method. As described in Section 4, the missing values are simply ignored in the split point search of the weak learners. So our imputed values often belong to the wrong partitions obtained by these splits. The binary features mainly improved the results of Churn and Up-selling. Since these features indicate particular values of numerical features, they were not considered by the decision stumps.

LMT Based on the predictions by boosted decision stumps we generated stacking features as described in Section 5 and appended them to the X_M set used in the boosted decision stumps to create the X_S set. In our experiments, the LMT was the only classifier that could benefit from the stacking features. All experiments in Table 2 and 3 were done using 50 LogitBoost iterations and splitting only was applied to nodes with at least 7,500 observations. The Tables 2 and 3 compare the entropy split criterion to the AUC split criterion described in Section 4 and show the effect of pruning in this task. To show the benefit of the stacked feature set X_S, we additionally applied the algorithm to the original features X_O.

Notice that pruning had a low impact on the resulting AUC when applied to the X_O feature set. Many tree induction methods were suitable to the given problem. Since the stacked fea-

Table 3: AUC scores of Logistic Model Trees on the X_S set

Features	Configuration	Appetency	Churn	Up-selling	Average
stacked	entropy	0.7096	0.6730	0.8090	0.7305
	entropy + pruning	0.7950	0.7037	0.8457	0.7815
	AUC	0.7652	0.6938	0.8349	0.7646
	AUC + pruning	**0.8050**	**0.7037**	**0.8557**	**0.7881**

Table 4: Comparison of classifiers on a common feature set of 399 numerical features, sorted by averaged AUC score

Classifier	Appetency	Churn	Up-selling	Score
Boosted decision stumps	0.8172	**0.7254**	**0.8488**	**0.7971**
LMT	**0.8176**	0.7161	0.8450	0.7929
MLP	0.8175	0.7049	0.7741	0.7655
SVMPerf	0.8102	0.7000	0.7495	0.7532

tures were better than any other subset of the X_O features, the tree growing procedure tend to split only on the X_S features in the first levels. Therefore, pruning indeed increased the generalization performance of the model and led to improved results. Trees with an entropy based split criterion resulted in degenerated trees, while trees with the AUC split criterion produced balanced tree structures. Especially on Up-selling, these balanced tree structures yielded an improvement. While on the X_O data set the difference to the entropy criterion was rather small, a benefit could be observed when applied to the X_S feature set.

In the presented experiments the feature sets were selected separately for each classifier. To isolate the classification performance from the effect of the feature selection, we conducted experiments on a common feature set. Table 4 shows the results on this data.

Combination Methods Finally, we combined results from different models, which produced our second best result. In the presented experiments we combined the predictions of four classifiers. In particular, boosted decision stumps, an MLP, an SVM, and an LMT were passed to the algorithm. The results obtained by the individual classifiers are shown in Table 4. With the Downhill-Simplex method an average AUC of 0.8018 was achieved on the common feature set, which is an improvement of +0.0047 compared to the boosted decision stumps. The weighted posteriors described in Section 5 were able to improve the result of boosted decision stumps by +0.0054 with an average AUC score of 0.8025. Further results are presented in Table 5.

Table 5: Combination of boosted decision stumps, MLP, SVM and LMT on a common feature set of 399 numerical features, sorted by averaged AUC score

Combination method	Appetency	Churn	Up-selling	Score
combined posteriors	**0.8263**	0.7283	0.8355	0.7967
combined votes	0.8246	0.7285	0.8344	0.7958
weighted posteriors	0.8242	0.7325	0.8507	**0.8025**
weighted votes	0.8225	**0.7331**	**0.8516**	0.8024
Downhill-Simplex	0.8256	0.7306	0.8493	0.8018

Table 6: Winners of the KDD cup 2009

Rank	Team Name	Appetency	Churn	Up-selling	Score
Fast track					
1	IBM Research	0.8830	0.7611	0.9038	0.8493
2	ID Analytics, Inc	0.8724	0.7565	0.9056	0.8448
3	David Slate, Peter W. Frey	0.8740	0.7541	0.9050	0.8443
Slow track					
1	IBM Research	0.8819	0.7651	0.9092	0.8521
2	Uni Melb	0.8836	0.7570	0.9048	0.8484
3	ID Analytics, Inc	0.8761	0.7614	0.9061	0.8479
35	RWTH Aachen	0.8268	0.7359	0.8615	0.8081

7. Conclusions

As part of a Data Mining lab course at the Chair of Computer Science 6 of RWTH Aachen University, we participated in the "Small Challenge" of the KDD cup 2009. The task was the prediction of three aspects of customer behavior: Churn, Appetency and Up-Selling. In our submission, we restricted ourselves to the provided feature set and did not use additional data from other tracks.

Given the large amount of missing values in the data, techniques for handling missing values were important in this task. While imputation methods are helpful and can improve results, our experiments showed that tree-based methods are more capable of handling or ignoring missing values. Our most successful method was a Logistic Model Tree with AUC as split criterion using predictions from boosted decision stumps as features. With the final AUC score of 0.8081, this was the best submission for the "Small Challenge" of the KDD cup 2009 that did not use additional data from other feature sets. It was ranked 35th among 89 submissions for this track. A second approach using an AUC-optimized weighted linear combination of several rankings scored slightly worse with an average AUC of 0.8074. The final results of the competition are summarized in Table 6.

8. Acknowledgments

The authors want to thank Thomas Deselaers, for his inspiration and helpful advice.

References

A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000. ISBN 0471056693.

C. Ferri, P. A. Flach, and J. Hernandez-Orallo. Learning decision trees using the area under the ROC curve. In *Proceedings of the 19th International Conference on Machine Learning*, pages 139–146. Morgan Kaufmann, 2002.

C. Ferri, P. A. Flach, and J. Hernandez-Orallo. Improving the auc of probabilistic estimation trees. In *Proceedings of the 14th European Conference on Machine Learning*, pages 121–132. Springer, 2003.

Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998.

T. Joachims. A support vector method for multivariate performance measures. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 377–384, New York, NY, USA, 2005. ACM.

N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

I. Myrtveit, E. Stensrud, and U. H. Olsson. Analyzing data sets with missing data: An empirical evaluation of imputation methods and likelihood-based methods. *IEEE Transactions on Software Engineering*, 27(11):999–1013, 2001.

I. T. Nabney. *NETLAB: Algorithms for Pattern Recognition*. Springer, 2001.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

A. Niculescu-Mizil and R. Caruana. Obtaining calibrated probabilities from boosting. In *Proc. 21st Conference on Uncertainty in Artificial Intelligence (UAI '05)*. AUAI Press, 2005.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

P. Smyth and D. Wolpert. Linearly combining density estimators via stacking. *Machine Learning*, 36(1-2):59–83, 1999.

JMLR Workshop and Conference Proceedings 7:89–100 KDD cup 2009

Classification of Imbalanced Marketing Data with Balanced Random Sets

Vladimir Nikulin V.NIKULIN@UQ.EDU.AU
Department of Mathematics, University of Queensland, Australia

Geoffrey J. McLachlan GJM@MATHS.UQ.EDU.AU
Department of Mathematics, University of Queensland, Australia

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

Abstract

With imbalanced data a classifier built using all of the data has the tendency to ignore the minority class. To overcome this problem, we propose to use an ensemble classifier constructed on the basis of a large number of relatively small and balanced subsets, where representatives from both patterns are to be selected randomly. As an outcome, the system produces the matrix of linear regression coefficients whose rows represent the random subsets and the columns represent the features. Based on this matrix, we make an assessment of how stable the influence of a particular feature is. It is proposed to keep in the model only features with stable influence. The final model represents an average of the base-learners, which is not necessarily a linear regression. Proper data pre-processing is very important for the effectiveness of the whole system, and it is proposed to reduce the original data to the most simple binary sparse format, which is particularly convenient for the construction of decision trees. As a result, any particular feature will be represented by several binary variables or bins, which are absolutely equivalent in terms of data structure. This property is very important and may be used for feature selection. The proposed method exploits not only contributions of particular variables to the base-learners, but also the diversity of such contributions. Test results against KDD-2009 competition datasets are presented.

Keywords: ensemble classifier, gradient-based optimisation, boosting, random forests, decision trees, matrix factorisation

1. Introduction

Ensemble (including voting and averaged) classifiers are learning algorithms that construct a set of many individual classifiers (called base-learners) and combine them to classify test data points by sample average. It is now well-known that ensembles are often much more accurate than the base-learners that make them up (Biau et al., 2007). The tree ensemble called "random forests" (RF) was introduced in (Breiman, 2001) and represents an example of a successful classifier. In another example, the bagged version of the support vector machine (SVM) (Zhang et al., 2007) is very important because direct application of the SVM to the whole data set may not be possible. In the case of the SVM, the dimension of the kernel matrix is equal to the sample size, which thus needs to be limited.

©2009 Vladimir Nikulin and Geoffrey J. McLachlan

Our approach was motivated by (Breiman, 1996), and represents a compromise between two major considerations. On the one hand, we would like to deal with balanced data. On the other hand, we wish to exploit all available information. We consider a large number n of balanced subsets of available data where any single subset includes two parts (a) nearly all 'positive' instances (minority) and (b) randomly selected 'negative' instances. The method of balanced random sets (RS) is general and may be used in conjunction with different base-learners.

Regularised linear regression (RLR) represents the most simple example of a decision function. Combined with quadratic loss function it has an essential advantage: using a gradient-based search procedure we can optimise the value of the step size. Consequently, we will observe a rapid decline in the target function.

By definition, regression coefficients may be regarded as natural measurements of influence of the corresponding features. In our case we have n vectors of regression coefficients, and we can use them to investigate the stability of the particular coefficients.

Proper feature selection (FS) may reduce overfitting significantly. We remove features with unstable coefficients, and recompute the classifiers. Note that stability of the coefficients may be measured using different methods. For example, we can apply the t-statistic given by the ratio of the mean to the standard deviation (Nikulin, 2008).

Matrix factorisation, an unsupervised learning method, is widely used to study the structure of the data when no specific response variable is specified. In principle, it would be better to describe the data in terms of a small number of meta-features, derived as a result of matrix factorisation, which could reduce noise while still capturing the essential features of the data. In addition, latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items.

Note that the methods for non-negative matrix factorisation (NMF) which were introduced in (Lee and Seung, 2001) are valid only under the condition that all the elements of all input and output matrices are non-negative. In Section 3.4 we formulate a general method for matrix factorisation, which is significantly faster compared with NMF. Note also that some interesting and relevant ideas for the stochastic gradient descent algorithm were motivated by methods used in the well-known Netflix Cup; see, for example, (Paterek, 2007).

The proposed approach is flexible. We do not expect that a single specification will work optimally on all conceivable applications and, therefore, an opportunity of tuning and tailoring the algorithm is important.

Results which were obtained during the KDD-2009 Data Mining Competition are presented.

2. Task Description

The KDD Cup 2009[1] offered the opportunity to work on large marketing databases from the French Telecom company Orange to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling).

Churn (Wikipedia definition) is a measure of the number of individuals or items moving into or out of a collection over a specific period of time. The term is used in many contexts, but is, most widely, applied in business with respect to a contractual customer base. For instance, it is an important factor for any business with a subscriber-based service model, including mobile telephone networks and pay TV operators. The term is also used to refer to participant turnover in peer-to-peer networks. Appetency is the propensity to buy a service or a product. Up-selling (Wikipedia definition) is a sales technique whereby a salesman attempts to have the

1. http://www.kddcup-orange.com/

customer purchase more expensive items, upgrades, or other add-ons in an attempt to make a more profitable sale. Up-selling usually involves marketing more profitable services or products, but up-selling can also be simply exposing the customer to other options that he or she may not have considered previously. Up-selling can imply selling something additional, or selling something that is more profitable or, otherwise, preferable for the seller instead of the original sale.

Customer Relationship Management (CRM) is a key element of modern marketing strategies. The most practical way in a CRM system to build knowledge on customer is to produce scores. The score (the output of a model) is computed using input variables which describe instances. Scores are then used by the information system (IS), for example, to personalize the customer relationship. There is also an industrial customer analysis platform able to build prediction models with a very large number of input variables (known as explanatory variables or features).

Generally, all features may be divided into two main parts: primary (collected directly from the customer) and secondary, which may be computed as a functions of primary features or may be extracted from other databases according to the primary features. Usually, the number of primary features is rather small (from 10 to 100). On the other hand, the number of secondary features may be huge (up to a few thousands). In most cases, proper design of the secondary features requires deep understanding of the most important primary features.

The rapid and robust detection of the features that have most contributed to the output prediction can be a key factor in a marketing applications. Time efficiency is often a crucial point, because marketing patterns have a dynamic nature and in a few days time it will be necessary to recompute parameters of the model using fresh data. Therefore, part of the competition was to test the ability of the participants to deliver solutions quickly.

3. Main Models

Let $\mathbf{X} = (\mathbf{x}_t, y_t), t = 1, \ldots, n$, be a training sample of observations where $\mathbf{x}_t \in \mathbb{R}^\ell$ is ℓ-dimensional vector of features, and y_t is binary label: $y_t \in \{-1, 1\}$. Boldface letters denote vector-columns, whose components are labelled using a normal typeface.

In supervised classification algorithms, a classifier is trained with all the labelled training data and used to predict the class labels of unseen test data. In other words, the label y_t may be hidden, and the task is to estimate it using vector of features. Let us consider the most simple linear decision function

$$u_t = u(\mathbf{x}_t) = \sum_{j=1}^{\ell} w_j \cdot x_{tj} + b,$$

where w_i are weight coefficients and b is a bias term.

We used AUC as an evaluation criterion, where AUC is the area under the receiver operating curve. By definition, ROC is a graphical plot of true positive rates against false positive rates.

3.1 RLR Model

Let us consider the most basic quadratic minimization model with the following target function:

$$L(\mathbf{w}) = \Omega(\phi, n, \mathbf{w}) + \sum_{t=1}^{n} \xi_t \cdot (y_t - u_t)^2, \tag{1}$$

where $\Omega(\phi, n, \mathbf{w}) = \phi \cdot n \cdot \|\mathbf{w}\|^2$ is a regularization term with ridge parameter ϕ; the ξ_t are non-negative weight coefficients.

Remark 1 *The aim of the regularization term with parameter ϕ is to reduce the difference between training and test results. Value of ϕ may be optimized using cross-validation; see Mol et al. (2009) for more details.*

3.1.1 GRADIENT-BASED OPTIMISATION

The direction of the steepest decent is defined by the gradient vector

$$g(\mathbf{w}) = \{g_j(\mathbf{w}), j = 1, \ldots, \ell\},$$

where

$$g_j(\mathbf{w}) = \frac{\partial L(\mathbf{w})}{\partial w_j} = 2\phi \cdot n \cdot w_j - 2\sum_{t=1}^{n} x_{tj}\xi_t (y_t - u_t).$$

Initial values of the linear coefficients w_i and the bias parameter b may be arbitrary. Then, we recompute the coefficients by

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \delta_k \cdot g(\mathbf{w}^{(k)}),\ b^{(k+1)} = b^{(k)} + \frac{1}{n}\sum_{t=1}^{n} \xi_t \cdot (y_t - u_t),$$

where k is the iteration number. Minimizing (1), we find the size of the step according to the formula

$$\delta = \frac{L_1 - L_2 - \phi \cdot n \sum_{j=1}^{\ell} w_j g_j}{\sum_{t=1}^{n} \xi_t s_t^2 + \phi \cdot n \sum_{j=1}^{\ell} g_j^2},$$

where

$$L_1 = \sum_{t=1}^{n} \xi_t s_t y_t, \quad L_2 = \sum_{t=1}^{n} \xi_t s_t u_t, \quad s_t = \sum_{j=1}^{\ell} x_{tj} g_j.$$

3.2 Random Sets

According to the proposed method, we consider large number of classifiers, where any particular classifier is based on a relatively balanced subset with randomly selected (without replacement) 'positive' and 'negative' instances. The final decision function was calculated as the sample average of the single decision functions or base-learners.

Definition 2 *We refer to the above subsets as random sets $RS(\alpha, \beta, m)$, where α is the number of positive cases, β is the number of negative cases, and m is the total number of random sets.*

This model includes two very important regulation parameters: m and $q = \frac{\alpha}{\beta} \leq 1$, where q is the proportion of positive to negative cases. In practice, m must be big enough, and q can not be too small.

We consider m subsets of $\mathbf{X}$ with α positive and $\beta = k \cdot \alpha$ negative data-instances, where $k \geq 1, q = \frac{1}{k}$. Using gradient-based optimization (see Section 3.1.1), we can compute the matrix of linear regression coefficients: $W = \{w_{ij}, i = 1, \ldots, m, j = 0, \ldots, \ell\}$.

3.3 Mean-Variance Filtering

The mean-variance filtering (MVF) technique was introduced in (Nikulin, 2006), and can be used to reduce overfitting. Using the following ratios, we can measure stability of the contributions of the particular features by

$$r_j = \frac{|\mu_j|}{\lambda_j},\ j = 1, \ldots, \ell, \tag{2}$$

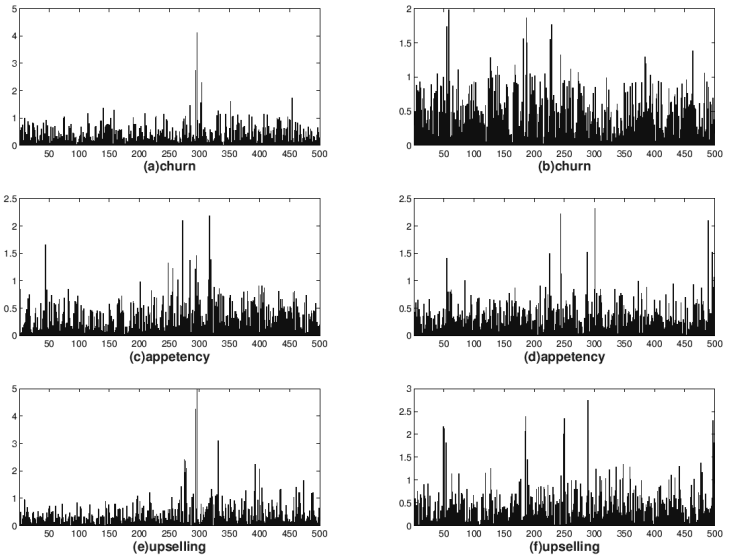


Figure 1: MVF, ratios (2): (a-b) Churn, (c-d) Appetency and (e-f) Upselling. In order to improve visibility, we displayed two fragments (out of 9586) with 500 ratios each.

where μ_j and λ_j are the mean and standard deviation corresponding to the j-column of the matrix W.

A low value of r_j indicates that the influence of the jth binary secondary feature is not stable. We conducted feature selection according to the condition: $r_j \geq \gamma > 0$. The sum of the r_j corresponding to the original feature f will give us a stability rating for the feature f.

3.4 Learning from the Test Data with Matrix Factorisation

Unlike classification and regression, matrix decomposition requires no response variable and thus falls into the category of unsupervised learning methods. Using this fact as a motivation, we can merge training and test datasets into one matrix $\mathbf{X}^+$. There are possible differences between training and test sets, and we can expect that the matrix factorisation will be able to smooth such differences.

In this section, we describe the procedure for undertaking the matrix factorisation,

$$\mathbf{X}^+ \sim \mathbf{AB}, \tag{3}$$

where matrices $\mathbf{A} = \{a_{if}, i = 1,\ldots,2n, f = 1,\ldots,k,\}$ and $\mathbf{B} = \{b_{fj}, f = 1,\ldots,k, j = 1,\ldots,\ell\}$. After the factorisation, the first n rows of the matrix $\mathbf{A}$ will used for training and the last n rows will be used for testing.

The matrix factorisation represents a gradient-based optimisation process with the objective to minimise the following squared loss function,

$$L(\mathbf{A},\mathbf{B}) = \frac{1}{2n \cdot \ell} \sum_{i=1}^{2n} \sum_{j=1}^{\ell} E_{ij}^2, \tag{4}$$

Algorithm 1 Matrix factorisation.

1: Input: $\mathbf{X}^{+}$ - dataset.
2: Select ν - number of global iterations; k - number of factors; $\lambda > 0$ - learning rate, $0 < \tau < 1$ - correction rate, L_S - initial value of the target function.
3: Initial matrices $\mathbf{A}$ and $\mathbf{B}$ may be generated randomly.
4: Global cycle: repeat ν times the following steps 5 - 17:
5: samples-cycle: for $i = 1$ to $2n$ repeat steps 6 - 17:
6: features-cycle: for $j = 1$ to ℓ repeat steps 7 - 17:
7: compute prediction $S = \sum_{f=1}^{k} a_{if} b_{fj}$;
8: compute error of prediction: $E = x_{ij} - S$;
9: internal factors-cycle: for $f = 1$ to k repeat steps 10 - 17:
10: compute $\alpha = a_{if} b_{fj}$;
11: update $a_{if} \Leftarrow a_{if} + \lambda \cdot E \cdot b_{fj}$ (see (5a));
12: $E \Leftarrow E + \alpha - a_{if} b_{fj}$;
13: compute $\alpha = a_{if} b_{fj}$;
14: update $b_{fj} \Leftarrow b_{fj} + \lambda \cdot E \cdot a_{if}$ (see (5b));
15: $E \Leftarrow E + \alpha - a_{if} b_{fj}$;
16: compute $L = L(\mathbf{A}, \mathbf{B})$;
17: $L_S = L$ if $L < L_S$, and $\lambda \Leftarrow \lambda \cdot \tau$, otherwise.
18: Output: $\mathbf{A}$ and $\mathbf{B}$ - matrices of latent factors.

where $E_{ij} = x_{ij} - \sum_{f=1}^{k} a_{if} b_{fj}$.

The above target function (4) includes in total $k(2n + \ell)$ regulation parameters and may be unstable if we minimise it without taking into account the mutual dependence between elements of the matrices $\mathbf{A}$ and $\mathbf{B}$.

As a solution to the problem, we can cycle through all the differences E_{ij}, minimising them as a function of the particular parameters which are involved in the definition of E_{ij}. Compared to usual gradient-based optimisation, in our optimisation model we are dealing with two sets of parameters, and we therefore mix uniformly updates of these parameters, because the latter are dependent.

The following partial derivatives are necessary for Algorithm 1:

$$\frac{\partial E_{ij}^2}{\partial a_{if}} = -2E_{ij} b_{fj}, \tag{5a}$$

$$\frac{\partial E_{ij}^2}{\partial b_{fj}} = -2E_{ij} a_{if}. \tag{5b}$$

Similarly, as in Section 3.1, we can optimise here the value of the step-size. However, taking into account the complexity of the model, it will be better to maintain fixed and small values of the step size or learning rate. In all our experiments, we conducted matrix factorisation with the above Algorithm 1 using 300 global iterations with the following regulation parameters: $\lambda = 0.01$ - initial learning rate, $\xi = 0.75$ -correction rate. We conducted experiments with Algorithm 1 against the datasets in a binary format, and Figure 2 illustrates convergence of the algorithm.

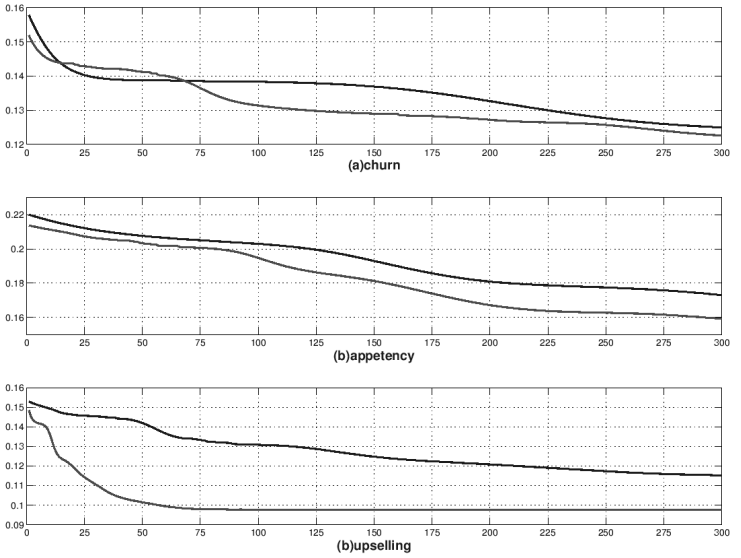


Figure 2: Convergence of Algorithm 1: (a) Churn, $\ell = 477$; (b) Appetency, $\ell = 532$; and (c) Upselling, $\ell = 385$, where blue lines correspond to $k = 8$ and red lines correspond to $k = 20$.

4. Experiments

4.1 Pre-processing Technique

The sizes of the training and test datasets are the same and are equal to 50000. There are 14740 numerical and 260 categorical features in the large dataset. The training data are very imbalanced. The numbers of positive cases were 3672 (Churn), 890 (Appetency) and 3682 (Upselling) out of a total number of 50000.

Firstly, we conducted a basic checking of the categorical data. We detected 72 variables with only one value. In addition, we removed 74 variables, where the number of missing variables was greater than 49500. The number of the remaining variables was 184. As a next step, we considered all possible values for the latter 184 variables, and found that 1342 values occurred frequently enough to be considered as independent binary variables (otherwise, values were removed from any further consideration).

An effective way to link information contained in numerical and categorical variables is to transfer the numerical variables to binary format (as it was before in the application to the categorical variables). We used a technique similar to that used before in converting the categorical variables to binary format. We removed all variables with numbers of missing and zeros greater than 49500. The number of the remaining variables was 3245. Next, we split the range of values for any particular variable into 1000 subintervals, and computed the numbers of occurrences for any subinterval. These numbers were considered later as weights of the bins.

Then, we split all subintervals for the particular variable into 10 consecutive bins with approximately the same size (in terms of weights). In many cases, where weights of some subintervals were too big, the numbers of bins were smaller than 10.

Table 1: Training and test in terms of AUC with averaged score 0.8059 (initial results); 0.8373 (fast and slow tracks results); 0.8407 (best results); 0.851 (post-challenge results). The column FS indicates the number of variables, which were used in the model, where by $\star$ we indicate the number of binary variables

Status	Data	Method	Train	Test	FS
Initial	Churn	RLR	0.8554	0.7015	9586*
Initial	Appetency	LinearSVM	0.9253	0.8344	9586*
Initial	Upselling	RLR	0.9519	0.8819	9586*
Initial	Toy	RLR	0.7630	0.7190	645*
Fast	Churn	LogitBoost	0.7504	0.7415	39
Fast/Best	Appetency	BinaryRF	0.9092	0.8692	145*
Fast	Upselling	LogitBoost	0.9226	0.9012	28
Slow/Best	Churn	LogitBoost	0.7666	0.7484	41
Slow	Appetency	LogitBoost	0.9345	0.8597	33
Slow	Upselling	LogitBoost	0.9226	0.904	54
Best	Upselling	LogitBoost	0.9208	0.9044	44
Best	Toy	Special	0.7354	**0.7253**	1 (N5963)
Post	Churn	Ensemble	0.8772	**0.7618**	278
Post	Appetency	Ensemble	0.9586	**0.8835**	348
Post	Upselling	Ensemble	0.9628	**0.9077**	285

Finally, we got a totally binary dataset in a sparse format with 13594 variables. After secondary trimming, we were left with 9586 binary features.

Remark 3 *It is a well-known fact that the exact correspondence between small and large datasets may be found. We managed to find such a correspondence as some other teams (it was rather a response to the findings of other teams). However, we can not report any significant progress (in the sense of the absolute scores), which was done by the help of this additional information.*

4.2 Results

The initial experiments, which were conducted against the vector of toy labels, were interesting and helpful for further studies. The system clearly detected all binary variables, which are secondary to the only one important original variable *N5963* (see Table 1). The definition of the transformation function between two known variables is a rather technical issue, which may be solved easily using two steps procedure: (1) sorting according to the explanatory variable, and (2) smoothing using sample averages in order to reduce noise.

As a first step (after labels for the Churn, Appetency and Upselling were released), we applied regularised linear regression model as described in Section 3.1. The number of the random sets was 100 and the ridge parameter was $\phi = 0.01$. In order to form any random set, we used about 90% of positive cases and $k = 1$. That is, any random set contains equal number of positive and negative instances. Note that in the case of Appetency, we considered initially the use of the SVM with a linear kernel; see the first 3 lines of Table 1.

Further, we employed mean-variance filtering, and reduced the number of features to 145 for Appetency, 151 for Churn, and 68 for Upselling (see Figure 1).

The participants received feedback against 10% of the test dataset (named leaderboard). In addition, we used cross-validation (CV) with up to 20 folds. Any CV-fold was formed under the strict condition that relation of the patterns is exactly the same as in the training dataset. Based on our CV-experiments, we observed a close relationship between the leaderboard and CV-results.

Remark 4 *After publication of the final results, we found that the relationship between the leaderboard and test results is also tight. It appears that in this particular case an "excessive" submissions against the leaderboard dataset did not lead to overfitting of the model. This prior knowledge may be very helpful for the second (slow) part of the competition.*

The binary (sparse) format gives a significant advantage in the sense of computational speed. But, it is not very important for the R-based packages in difference to the memory allocation. Accordingly, we returned to the original variables by replacing the binary features by their sequential indices (within the group corresponding to the particular original feature) before loading the new datasets into the R-environment.

We used mainly in our experiments five models RLR, LinearSVM, BinaryRF, LogitBoost and RF, where the last two models were implemented in R, the other models were written in C. For example, the following settings were used for the BinaryRF (Appetency case, see Table 1): (1) decision trees with up to 14 levels; (2) the number of features were selected randomly out of the range between 12 and 17 for any particular split; (3) the splitting process was stopped if improvement was less than 0.1% or number of data in the node was less than 100; 4) number of RS was 100 and number of trees for any RS was 400 (that means, the total number of trees was 40000).

5. Post-challenge Submissions

Firstly, we decided to rebuild completely all the databases. It was an extension of the previous databases based on the MVF feature selections ratings. Also, we took into account ratings from the KDD-preprint of the University of Melbourne team - winner of the slow track (see Table 3). We were working in two mutually dependent directions: binary databases for our own software binaryRF, and the corresponding integer databases of indexes for R packages named randomForest, ADA and GBM (for the last one we would like to thank the University of Melbourne team again, as they reported this package in their KDD-preprint).

Table 2: Selected pure results

Data	Model	Test	Weight	Data	Model	Test	Weight
Appetency	BinaryRF	0.8784	10	Churn	GBM	0.7613	-
Appetency	GBM	0.878	9	Upselling	GBM	0.9072	20
Appetency	ADA	0.8742	3	Upselling	ADA	0.9071	19

5.1 Ensemble Constructor

Using results of Table 2 separately we can achieve the score of 0.8489. Now, we shall describe a general framework how using ensemble technique we can increase the score to 0.851 based on the results of the above Table 2 and without any additional information. Note that the particular solutions, which were produced using different software may have approximately the same AUC, but they are very different in the structural sense and we can not link them directly.

Table 3: Results of the winners

Track	Team	Churn	Appetency	Upselling	Score
Fast	IBM Research	0.7611	0.883	0.9038	0.8493
Slow	Uni. Melb.	0.7542	0.8836	0.9047	0.8475
Overall	-	0.7651	0.8836	0.9092	0.8526

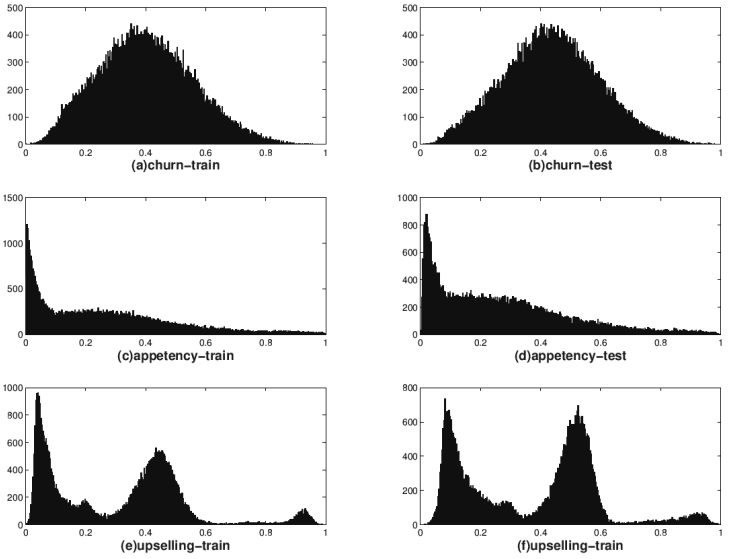


Figure 3: Histograms with 300 bins illustrating the structural similarities between training and test trajectories, where the right columns corresponds to the score 0.8509: (a-b) Churn, (c-d) Appetency and (e-f) Upselling.

Suppose we have k solutions, which may be represented by the squared matrix $\mathbf{T}$ with k columns. By $\mathbf{S}$ we shall denote the matrix, which was derived from the original matrix of solutions $\mathbf{T}$ by sorting on any column in an increasing order. In order to define the ensemble constructor we also need a matrix $\mathbf{R}$ of indices defining an exact correspondence between the elements of $\mathbf{T}$ and $\mathbf{S}$. More precisely, $s_{r_{ij}j} = t_{ij}$.

Let us denote by β_j the quality measurement of the solution with index $j = 1,\ldots,k$. In order to simplify the following notation and without loss of generality we shall assume that (a) solution i is better comparing with solution j if $\beta_i > \beta_j$, (b) the top solution corresponds to the index 1, that means $\beta_1 \geq \beta_j, j = 2,\ldots,k$. Then, we can accept solution N1 as a base solution, and shall adjust remaining $k-1$ solutions according to the rule: $q_{ij} = s_{r_{ij}1}, i = 1,\ldots,n, j = 2,\ldots,k$. Note that the first column in the matrix $\mathbf{Q}$ coincides with the first column of the original matrix $\mathbf{T}$ (base solution).

We shall define vector-column of the weight coefficients $w_j = \psi(\beta_j), \sum_{j=1}^{k} w_j = 1$, where ψ is an increasing function. In line with the proposed method, the ensemble solution will be computed according to the formula: $f = \mathbf{QW}$.

We can report a significant progress with above technique in application to the Appetency case. Our ensemble solution was constructed using three pure solutions as it is displayed in Table 2, where the weight coefficients are shown without normalisation. Also, we achieved some modest progress in application to the Upselling case. However, in the case of Churn we were not able to improve GBM-solution using above technique.

On the other hand, we were trying to exploit mutual dependences between different cases. For example, higher combined score in relation to the Appetency and Upselling cases most likely indicates lower score in application to the Churn case:

$$new.score_C = old.score_C - c_1 score_A - c_2 score_U, \tag{6}$$

where c_1 and c_2 are non-negative constants. In fact, using above method (6) we managed to achieve some small improvement only in application to the Churn case. Possibly, some interesting results may be achieved using multinomial logistic regression (MLR) (Bohning, 1992), as this model explore a hidden dependencies between labels. In our case we have four different labels: 1) Churn, 2) Appetency, 3) Upselling and 4) Other. In the case of MLR, we shall be dealing with the 3ℓ-dimensional Hessian matrix of second derivatives. We can expect that the matrix factorisation, as it is described in Section 3.4, will be effective to reduce original dimensionality ℓ to k - number of the latent factors.

It is interesting to note that the histograms displayed on Figure 3 illustrate remarkable similarity between left (training) and right (test) columns. The structure of the histograms corresponding to the Upselling case is far from simple and it will be very important to find explanations in the terms of the particular features.

6. Concluding Remarks

The main philosophy of our method may be described as follows. We can not apply fairly complex modelling systems to the original huge and noisy database, which contains more than 90% of useless information. So we conducted the FS step with three very simple and reliable methods, namely RS, RLR, and MVF. As an outcome, we produced significantly smaller datasets, which are able to be used as an input for more advanced studies.

In general terms, we have found that our results are satisfactory, particularly, for the most important fast track. Based on the results of our post-challenge submission, we can conclude that significant progress may be achieved using more advanced pre-processing and feature selection techniques. Also, it will be a good idea not to rely completely on any particular model or software, and conduct cross-validation experiments with several different models. In this way, the performance of the final solution might be improved using various ensemble techniques.

References

G. Biau, L. Devroye, and G. Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9:2015–2033, 2007.

D. Bohning. Multinomial logistic regression algorithm. *Ann. Inst. Statist. Math.*, 44(1):197–200, 1992.

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

L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

D. Lee and H. Seung. Algorithms for non-negative matrix factorisation. NIPS, 2001.

C. Mol, S. Mosci, M. Traskine, and A. Verri. A regularised method for selecting nested groups of relevant genes from microarray data. *Journal of Computational Biology*, 16(5):677–690, 2009.

V. Nikulin. Learning with mean-variance filtering, SVM and gradient-based optimization. In *International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21*, pages 4195–4202. IEEE, 2006.

V. Nikulin. Classification of imbalanced data with random sets and mean-variance filtering. *International Journal of Data Warehousing and Mining*, 4(2):63–78, 2008.

A. Paterek. Improving regularised singular value decomposition for collaborative filtering. pages 39–42. KDD, San Jose, 2007.

B. Zhang, T. Pham, and Y. Zhang. Bagging support vector machine for classification of SELDI-ToF mass spectra of ovarian cancer serum samples. In *LNAI*, volume 4830, pages 820–826. Springer, 2007.

JMLR Workshop and Conference Proceedings 7:101–109 KDD cup 2009

Application of Additive Groves Ensemble with Multiple Counts Feature Evaluation to KDD Cup'09 Small Data Set

Daria Sorokina DARIA@CS.CMU.EDU
School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213

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

Abstract

This paper describes a field trial for a recently developed ensemble called Additive Groves on KDD Cup'09 competition. Additive Groves were applied to three tasks provided at the competition using the "small" data set. On one of the three tasks, appetency, we achieved the best result among participants who similarly worked with the small dataset only. Postcompetition analysis showed that less successfull result on another task, churn, was partially due to insufficient preprocessing of nominal attributes.

Code for Additive Groves is publicly available as a part of TreeExtra package. Another part of this package provides an important preprocessing technique also used for this competition entry, feature evaluation through bagging with multiple counts.

Keywords: Additive Groves, feature evaluation, KDD Cup 09.

1. Introduction

Additive Groves is a powerful ensemble of regression trees (Sorokina et al. (2007)). It is originally developed for regression problems and is shown to outperform state-of-the-art methods on medium-size regression data sets. The data sets for the KDD Cup competition presented new challenges for the algorithm due to the following differences from the data sets used in the original paper:

- Binary classification problems
- ROC performance metric
- Large size of the train set (50000 data points)
- High-dimensional data (260 features in the small, 15000 in the large data set)
- Presence of nominal features with extremely high arity

Binary classification and ROC. Several modifications were attempted to tailor original regression algorithm to classification problem and ROC performance metric. However, it turned out that binary classification combined with ROC is a favorable setting for the original Additive Groves. In the end, only minor changes were applied to create the classification version of the algorithm.

©2009 Daria Sorokina

Large high-dimensional data set. The size of data was too large to be processed by Additive Groves in reasonable time. Therefore, a feature selection technique had to be applied. From our experience, "white-box" feature evaluation techniques based on analyzing a structure of bagged trees provide good sets of features for tree-based ensembles in general and therefore are useful as a preprocessing step for Additive Groves. We used "multiple counts", a simple technique described in (Caruana et al. (2006)).

Nominal features. We have converted each nominal feature into a single integer feature, with different integer values corresponding to different nominal values. We discovered later that such preprocessing was not sufficient to prevent overfitting in at least one of the three competition tasks.

In the end, Additive Groves proved its applicability to large classification problems by producing the absolute best score on one of the three tasks in the "small" challenge.

2. Data preprocessing

There were two versions of the data set at the competition, "large" and "small". Both data sets contain 50000 data points in the train set. The "large" data set includes 15000 features and due to the lack of computing resources was not used in this entry. The "small" data set consists of 260 features: 190 numerical and 40 nominal. Labels were provided for three tasks on the same data: churn, appetency and upselling.

2.1 Nominal features

Our implementation of decision trees does not have a built-in technique to deal with nominal (also called categorical or non-numerical) features. We had to convert them to binary or integer features during the data preprocessing stage. As some nominal features contained thousands of values, each nominal feature was converted into a single integer feature, where each integer value corresponded to a single nominal value. The values were ordered by the number of data points taking on this value. See Section 6 on useful improvements for this type of preprocessing.

2.2 Initial parameter tuning

Our feature selection technique described in the next section requires training an ensemble of bagged trees. Bagging is a well-known ensemble method introduced by Breiman (1996). It creates a set of diverse models by sampling from the training set, and then decreases variance by averaging the predictions of these models. Decision tree is a common choice for the model in the bagging ensemble.

We would like to calculate feature scores from the version of ensemble that produces the best performance. Bagged trees have an important parameter: size of tree. Our implementation controls the size of the tree through an external parameter α. α defines the percentage of training data in a leaf, so in some sense it is reverse to the size of the tree. We have tried the following set of values of α: 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0. See (Sorokina et al. (2007)) for the table of approximate correspondence between different values of α and the number of nodes in the resulting trees.

Another parameter is the number of trees in the ensemble. We have opted for a fixed value of 100 trees.

In the competition framework we have a limited labeled data set, so we have to decide on another important parameter: train/validation set ratio. Our data contains 50000 data points. We have considered train sets of $1,2,5,10,20,30,40,45,48$ and 49 thousands of data points.

Table 1: Parameters producing the best result for bagged trees

task	α	train set size
churn	0.1	48 000
appetency	0.05	45 000
upselling	0.05	40 000

Table 1 shows the combination of parameters that produced the best performance on validation set for bagged trees on each task.

2.3 Feature evaluation: multiple counts technique

The simple idea behind the "white-box" feature importance evaluation is the following: let us run a fast tree-based ensemble - bagging - and see which features are actually used by the trees. We discuss a number of such techniques in our earlier work (Caruana et al. (2006)). For this entry we used one of the methods producing good results - multiple counts. We define a score of a tree node as the number of the data points in the train set passing through this node during training. Then, the score of the feature is defined as the sum of the scores of the internal tree nodes that use this feature. For convenience the score is scaled by the number of data points in the train set as well as the number of trees. *Multiple* in the name of this method refers to the fact that a data point can be counted several times towards the same score if the same feature is used several times along one branch. For example, the tree on Fig. 1 will produce scores of 1.6, 0.8 and 0.2 for features *A*, *B* and *C* respectively.

For every task we created ensembles of bagged trees using the whole train set and the best values of α from Table 1. These ensembles were used to calculate the multiple counting scores. In each case, top 20 features were selected to compose a data set for applying Additive Groves. The list of selected features and their scores is shown in Table 2.

3. Additive Groves

3.1 Regression ensemble

Additive Groves is a tree ensemble algorithm based on regression trees, additive models and bagging and is capable of both fitting additive structure of the problem and modelling nonlinear components with large trees at the same time. Combination of these properties makes it superior

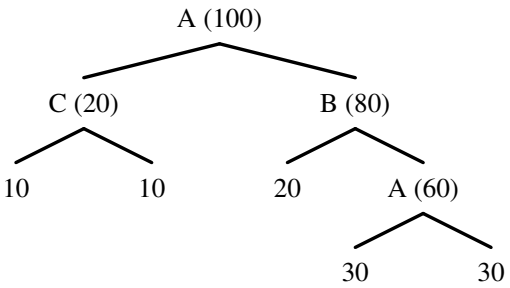


Figure 1: Sample decision tree splits the data using features A, B and C. Node labels show the percentage of the train set data points passing through each node.

Table 2: Top 20 features and their importance scores

churn		**appetency**		**upselling**	
Var126	1.22	Var126	1.32	Var126	1.57
Var29	1.00	Var29	1.00	Var28	1.36
Var130	0.99	Var130	0.99	Var130	0.99
Var201	0.98	Var201	0.94	Var201	0.97
Var90	0.85	Var90	0.85	Var29	0.96
Var192	0.72	Var218	0.76	Var211	0.84
Var138	0.64	Var217	0.67	Var90	0.83
Var113	0.52	Var138	0.64	Var217	0.70
Var74	0.44	Var125	0.53	Var138	0.59
Var189	0.37	Var199	0.48	Var73	0.42
Var205	0.29	Var211	0.47	Var113	0.27
Var73	0.28	Var202	0.46	Var94	0.24
Var211	0.25	Var28	0.44	Var216	0.23
Var199	0.23	Var193	0.39	Var214	0.22
Var212	0.21	Var73	0.34	Var6	0.21
Var217	0.11	Var94	0.34	Var202	0.20
Var2	0.11	Var57	0.33	Var192	0.20
Var13	0.11	Var192	0.31	Var197	0.19
Var218	0.09	Var200	0.30	Var125	0.19
Var81	0.09	Var189	0.28	Var57	0.18

in performance to other existing tree ensemble methods like bagging, boosting and Random Forests.

Additive Groves consists of bagged additive models, where every element of an additive model is a tree. A single grove is trained similar to an additive model: each tree is trained on the residuals of the sum of the predictions of the other trees. Trees are discarded and retrained in turn until the overall predictions converge to a stable function. In addition, each grove of a non-trivial size is iteratively built from smaller models: either a tree is added to a grove with fewer trees or the size of trees is increased in a grove with smaller trees; which of the two steps happen in each particular case is defined by comparing models on out-of-bag data.

A size of a single grove (single additive model) is defined by two parameters: the size of tree and the number of trees. A single grove consisting of large trees can and will overfit heavily to the training set, therefore bagging is applied on top in order to reduce variance.

Algorithm 1 gives pseudo-code for training a grid of groves of different sizes during one bagging iteration. We refer the reader to the paper where Additive Groves were introduced (Sorokina et al. (2007)) for more detailed description of the algorithm.

3.2 Modification for classification problems

In Additive Groves, models of different sizes are naturally produced during the training phase. These models are evaluated on the validation set in order to determine the combination of parameters producing the best model. The only modification we ended up using for the classification problems is the use of ROC metric insted of original RMSE (root mean squared error) during this final model evaluation. We still use RMSE for fitting single trees and building additive models.

Algorithm 1 Training Additive Groves

// $\text{Tree}_i^{(\alpha_j,n)}$ denotes an i^{th} tree in the grove with the parameters (α_j, n)
function TrainGrove(α,N,trainSet)
(
$\alpha_0 = 0.5, \alpha_1 = 0.2, \alpha_2 = 0.1, \ldots, \alpha_{\max} = \alpha$
for $j = 0$ **to** max **do**
  for $n = 1$ **to** N **do**

    for $i = 1$ **to** $n-1$ **do**
      $\text{Tree}_{\text{attempt1},i} = \text{Tree}_i^{(\alpha_j,n-1)}$
    end for
    $\text{Tree}_{\text{attempt1},n} = 0$
    Converge(α_j,n,train,$\text{Tree}_{\text{attempt1},1}, \ldots, \text{Tree}_{\text{attempt1},n}$)

    if $j > 0$ **then**
      for $i = 1$ **to** n **do**
        $\text{Tree}_{\text{attempt2},i} = \text{Tree}_i^{(\alpha_{j-1},n)}$
      end for
      Converge(α_j,n,train,$\text{Tree}_{\text{attempt2},1}, \ldots, \text{Tree}_{\text{attempt2},n}$)
    end if

    winner = Compare $\sum_i \text{Tree}_{\text{attempt1},i}$ and $\sum_i \text{Tree}_{\text{attempt2},i}$ on out-of-bag data
    for $i = 1$ **to** n **do**
      $\text{Tree}_i^{(\alpha_j,n)} = \text{Tree}_{\text{winner},i}$
    end for
  end for
end for
)
function Converge(α,N,{$\mathbf{x}$,y},$\text{Tree}_1^{(\alpha,N)}, \ldots, \text{Tree}_N^{(\alpha,N)}$)
(
repeat
  for $i = 1$ **to** N **do**
    newTrainSet $= \{\mathbf{x}, y - \sum_{k \neq i} \text{Tree}_k^{(\alpha,N)}(\mathbf{x})\}$
    $\text{Tree}_i^{(\alpha,N)} = \text{TrainTree}(\alpha, \text{newTrainSet})$
  end for
until (change from the last iteration is small)
)

Several modifications were tried in an attempt to tune Additive Groves better for classification problems evaluated by ROC metric. None of them demonstrated any improvements and eventually these modifications were discarded. They include:

- Using ROC instead of RMSE when comparing different models of the same size during training
- Clipping predictions of each additive model at 0 and 1[1] while fitting models

1. Predictions greater than 1 become 1, predictions less than 0 become 0

- Clipping predictions of the whole bagged ensemble at 0 and 1 after the training is completed

The last two modifications were attempted because additive model-based algorithms sometimes can predict a value greater than 1 or smaller than 0, which does not make much sense in binary $0-1$ classification. Such prediction will generate a negative residual during training even if it is technically correct (that is, actual true values are 1 or 0 respectively). Some performance metrics, for example RMSE, will count such predictions as errors. However, ROC does not have this problem, because it does not evaluate the actual prediction numbers. ROC is interested only in the ordering of predictions, and it does not matter if the true positive prediction is greater than 1 as long as it is larger than predictions for negative cases. In this sense ROC is a very convenient metric for evaluating models such as Additive Groves on classification problems.

4. Final models

Additive Groves have 3 crucial parameters: α (controls size of tree), N (number of trees in each grove) and number of bagging iterations. The last parameter is conservative - the more bagging iterations the better, while for the other two parameters there usually exist some optimal level of complexity.

In order to evaluate the models and to find the optimal values of α and N, we reused the train/validation splits of data from the earlier experiments (Table 1). Table 3 provides best performance of Additive Groves on our validation sets and parameters of correspondent models.

Table 3: Performance on validation set

task	α	N	# of bagged models	ROC on validation set
churn	0.05	6	140	0.762
appetency	0.2	5	100	0.832
upselling	0.05	4	100	0.860

We have trained final models using the whole labeled data set, (α, N) values from the Table 3 and 1000 bagging iterations. It is worth noting that our implementation allows to train the models much faster once the desired values of N and α are fixed.

5. Competition results

KDD Cup'09 competition consisted of several separate challenges: fast, slow(large) and slow(small). More, there were two types of submission for small challenge: with unscrambling and without. Submissions of the first type made use of both large and small data set and therefore are not comparable to the submissions that used the small data set only.

Additive Groves competed in the small challenge without unscrambling. Table 4 shows 47 entries from the official results table (Guyon (2009)) that fall into this category.

The good news is that Additive Groves defeated every other entry on the appetency task with the score of 0.8311. The second best score in this category is 0.8268. This result shows that Additive Groves can be as superior for classification as it is for regression at least on some tasks.

On the upselling task Additive Groves got the score of 0.8605. The best result was 0.8644.

Unfortunately, the result for churn was much worse - Additive Groves achieved ROC of 0.7135 while the best result was 0.7406. With more than half competitors ahead, it was clear

Table 4: Competition results for small data set without unscrambling

team	churn	appetency	upselling	average
creon	0.7359	0.8268	0.8615	**0.80808**
LosKallos	0.7398	0.8204	0.8621	0.80745
FEG_BOSS	**0.7406**	0.8149	0.8621	0.80583
Lenca	0.7348	0.8175	0.8629	0.80506
M	0.7319	0.8153	**0.8644**	0.80384
FEG ATeam	0.7325	0.8160	0.8610	0.80313
pavel	0.7358	0.8130	0.8591	0.80266
Additive Groves	0.7135	**0.8311**	0.8605	0.80171
nikhop	0.7359	0.8098	0.8589	0.80153
mi	0.7365	0.8090	0.8569	0.80079
java.lang.OutOfMemoryError	0.7360	0.8090	0.8572	0.80075
Lajkonik	0.7323	0.8073	0.8600	0.79986
FEG CTeam	0.7321	0.8062	0.8596	0.79930
Celine Theeuws	0.7230	0.8147	0.8584	0.79871
zlm	0.7232	0.8175	0.8544	0.79835
FEG B	0.7354	0.8031	0.8544	0.79760
CSN	0.7282	0.8051	0.8594	0.79757
TonyM	0.7249	0.7996	0.8596	0.79471
Sundance	0.7244	0.8172	0.8400	0.79387
Miner12	0.7264	0.7973	0.8484	0.79072
FEG D TEAM	0.7219	0.8077	0.8422	0.79060
DKW	0.7153	0.8015	0.8547	0.79050
idg	0.7146	0.8041	0.8507	0.78980
homehome	0.7176	0.8062	0.8416	0.78848
Mai Dang	0.7167	0.8099	0.8372	0.78795
parramining.blogspot.com	0.7134	0.8056	0.8438	0.78756
bob	0.7053	0.8052	0.8520	0.78751
muckl	0.7239	0.8195	0.8180	0.78714
C.A.Wang	0.7067	0.8043	0.8502	0.78707
KDD@PT	0.7081	0.7989	0.8528	0.78660
decaff	0.7120	0.7916	0.8498	0.78446
StatConsulting	0.7137	0.7605	0.8501	0.77477
Dr. Bunsen Honeydew	0.7170	0.8052	0.7954	0.77254
Raymond Falk	0.6905	0.7744	0.8465	0.77045
vodafone	0.7258	0.6915	0.8582	0.75853
K2	0.7078	0.7670	0.7931	0.75600
Claminer	0.6665	0.7785	0.8199	0.75499
rw	0.7257	0.6928	0.8369	0.75178
Persistent	0.6416	0.7167	0.7370	0.69843
Louis Duclos-Gosselin	0.6168	0.7571	0.6792	0.68434
Chy	0.6027	0.7201	0.6936	0.67214
Abo-Ali	0.6249	0.6425	0.7218	0.66305
sduzx	0.6057	0.6465	0.6167	0.62295
MT	0.5494	0.5378	0.6873	0.59149
Shiraz University	0.5077	0.5047	0.7000	0.57082
Klimma	0.5283	0.5231	0.5909	0.54745
thes	0.5016	0.4993	0.4982	0.49969

that we failed to extract some important information from the data in this case. In the next section we discuss one of the possible reasons.

6. Post-competition improvements

We have reconsidered several aspects of preprocessing of nominal values after the competition. First, we modified the way of ordering the values while converting a nominal feature to a numerical. During the competition we have been ordering values by the number of data points taking on this value. Later we have adopted a potentially more useful technique: ordering the values by the ratio of positive responses among the data points taking on each value (Hastie et al., 2001, chapter 9.2.4).

Second, in order to understand the reasons for failing on churn problem, the author has contacted Hugh Miller, the leader of Uni Melb team. This team achieved much better results on

churn in the small challenge. We learned that they also used 20 features, however, the exact set of features they were using did not help to improve the results in our case. Further, we learned that they used a more elaborate preprocessing of the nominal features.

"For those categorical variables with more than 25 categories, levels with fewer than 100 instances in the training data were aggregated into a small category, those with 100-500 instances were aggregated into a medium category, and those with 500-1000 instances aggregated into a large category." - Uni Melb factsheet

Such preprocessing decreases the chance for the trees to overfit by making too many splits on a nominal feature.

We have incorporated the techniques mentioned above into preprocessing and repeated experiments for churn with Additive Groves on the improved version of the data. This helped to boost our best performance on the official test set from 0.7135 to 0.7157.

Therefore, it seems that at least a part of our problem with churn task was caused by insufficient data preprocessing.

7. TreeExtra package

Both Additive Groves and bagged trees with multiple counts feature evaluation are available as parts of TreeExtra package. TreeExtra is a set of command line tools written in C++/STL. You can find binaries, code and manuals at

`www.cs.cmu.edu/~daria/TreeExtra.htm`

Acknowledgments

The author would like to acknowledge her collaborators who helped to develop algorithms described in this paper: Rich Caruana, Mirek Riedewald and Art Munson.

References

Leo Breiman. Bagging Predictors. *Machine Learning*, 24:123–140, 1996.

Rich Caruana, Mohamed Elhawary, Art Munson, Mirek Riedewald, Daria Sorokina, Daniel Fink, Wesley M. Hochachka, and Steve Kelling. Mining citizen science data to predict prevalence of wild bird species. In *Proceedings of ACM KDD*, 2006.

I. Guyon. KDD Cup results. http://www.kddcup-orange.com/winners.php?page=slow, 2009.

T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

Daria Sorokina, Rich Caruana, and Mirek Riedewald. Additive Groves of Regression Trees. In *Proceedings of ECML*, 2007.

JMLR Workshop and Conference Proceedings 7:111–122 KDD cup 2009

Accelerating AdaBoost using UCB

Róbert Busa-Fekete BUSAROBI@GMAIL.COM
LAL, University of Paris-Sud, CNRS
Orsay, 91898, France
Research Group on Artificial Intelligence of the
Hungarian Academy of Sciences and University of Szeged
Aradi vértanúk tere 1., H-6720 Szeged, Hungary

Balázs Kégl BALAZS.KEGL@GMAIL.COM
LAL/LRI, University of Paris-Sud, CNRS
Orsay, 91898, France

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

Abstract

This paper explores how multi-armed bandits (MABs) can be applied to accelerate AdaBoost. AdaBoost constructs a strong classifier in a stepwise fashion by adding simple base classifiers to a pool and using their weighted "vote" to determine the final classification. We model this stepwise base classifier selection as a sequential decision problem, and optimize it with MABs. Each arm represents a subset of the base classifier set. The MAB gradually learns the "utility" of the subsets, and selects one of the subsets in each iteration. ADABOOST then searches only this subset instead of optimizing the base classifier over the whole space. The reward is defined as a function of the accuracy of the base classifier. We investigate how the well-known UCB algorithm can be applied in the case of boosted stumps, trees, and products of base classifiers. The *KDD Cup 2009* was a large-scale learning task with a limited training time, thus this challenge offered us a good opportunity to test the utility of our approach. During the challenge our best results came in the *Up-selling* task where our model was within 1% of the best AUC rate. After more thorough post-challenge validation the algorithm performed as well as the best challenge submission on the small data set in two of the three tasks.

Keywords: AdaBoost, Multi-Armed Bandit Problem, Upper Confidence Bound

1. Introduction

ADABOOST (Freund and Schapire, 1997) is one of the best off-the-shelf learning methods developed in the last decade. It constructs a classifier in a stepwise fashion by adding simple classifiers (called *base classifiers*) to a pool, and using their weighted "vote" to determine the final classification. The simplest base learner used in practice is the *decision stump*, a one-decision two-leaf decision tree. Learning a decision stump means to select a feature and a threshold, so the running time of ADABOOST with stumps is proportional to the number of data points n, the number of attributes d, and the number of boosting iterations T. When *trees* (Quinlan, 1993) or *products* (Kégl and Busa-Fekete, 2009) are constructed over the set of stumps, the computational cost is multiplied by an additional factor of the number of tree levels N or the number

©2009 Róbert Busa-Fekete and Balázs Kégl

of terms m. Although the running time is linear in each of these factors, the algorithm can be prohibitively slow if the data size n and/or the number of features d is large.

There are essentially two ways to accelerate ADABOOST in this setting: one can either limit the number of data points n used to train the base learners, or one can cut the search space by using only a subset of the d features. Although both approaches increase the number of iterations T needed for convergence, the net computational time can still be significantly decreased. The former approach has a basic version when the base learner is not trained on the whole weighted sample, rather on a small subset selected randomly using the weights as a discrete probability distribution (Freund and Schapire, 1997). A recently proposed algorithm of the same kind is FILTERBOOST (Bradley and Schapire, 2008), which assumes that an oracle can produce an unlimited number of labeled examples, one at a time. In each boosting iteration, the oracle generates sample points that the base learner can either accept or reject, and then the base learner is trained on a small set of accepted points. The latter approach was proposed by (Escudero et al., 2000) which introduces several feature selection and ranking methods used to accelerate ADABOOST. In particular, the LAZYBOOST algorithm chooses a fixed-size random subset of the features in each boosting iteration, and trains the base learner using only this subset. This technique was successfully applied to face recognition where the number of features can be extremely large (Viola and Jones, 2004).

In this paper we aim to improve the latter approach by "aiding" the random feature selection. It is intuitively clear that certain features are more important than others for classification. In specific applications the utility of features can be assessed a-priori (e.g., on images of characters, we know that background pixels close to the image borders are less informative than pixels in the middle of the images), however, our aim here is to *learn* the importance of features by evaluating their empirical performance during the boosting iterations. Our proposed method is similar in spirit to the feature extraction technique described recently by (Borisov et al., 2006; Tuv et al., 2009). The objective of their method is to use tree-based ensembles for feature selection whereas our goal is more restrictive: we simply want to accelerate ADABOOST. To avoid harming the generalization ability of ADABOOST it is important to keep a high level of base learner diversity, which is the reason why we opted for using *multi-armed bandits* (MAB) that are known to manage the exploration-exploitation trade-off very well.

MAB techniques have recently gained great visibility due to their successful applications in real life, for example, in the game of GO (Gelly and Silver, 2008). In the classical bandit problem the decision maker can select an arm at each discrete time step (Auer et al., 2002b). Selecting an arm results in a random reward, and the goal of the decision maker is to maximize the expected sum of the rewards received. Our basic idea is to partition the base classifier space into subsets and use MABs to learn the utility of the subsets. In each iteration, the bandit algorithm selects an optimal subset, then the base learner finds the best base classifier in the subset and returns a reward based on the accuracy of this optimal base classifier. By reducing the search space of the base learner, we can expect a significant decrease of the complete running time of ADABOOST. We use the UCB algorithm (Auer et al., 2002a) by assigning each feature to a subset. In the case of trees and products we use UCB by considering each tree or product as a sequence of decisions, and using the same partitioning as with decision stumps at each inner node.

The paper is organized as follows. First we describe the ADABOOST.MH algorithm and the necessary notations in Section 2. Section 3 contains our main contribution of using MABs for accelerating the selection of base classifiers. In Section 4 we present experiments conducted during the development period of the competition, our competition results, and some post-challenge analysis. Closing discussions are in Section 5.

ADABOOST.MH($\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}$, BASE$(\cdot,\cdot,\cdot), T$)

1 **for** $t \leftarrow 1$ **to** T

2 $\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot) \leftarrow$ BASE$\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$

3 $\quad$ **for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K

4 $\quad\quad w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \dfrac{\exp\big(-h_\ell^{(t)}(\mathbf{x}_i) y_{i,\ell}\big)}{\sum_{i'=1}^{n}\sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} \exp\big(-h_{\ell'}^{(t)}(\mathbf{x}_{i'}) y_{i',\ell'}\big)}$

5 **return** $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\cdot)$

Figure 1: The pseudocode of the ADABOOST.MH algorithm. $\mathbf{X}$ is the observation matrix, $\mathbf{Y}$ is the label matrix, $\mathbf{W}^{(1)}$ is the initial weight matrix, BASE$(\cdot,\cdot,\cdot)$ is the base learner algorithm, and T is the number of iterations. $\alpha^{(t)}$ is the base coefficient, $\mathbf{v}^{(t)}$ is the vote vector, $\varphi^{(t)}(\cdot)$ is the scalar base classifier, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) classifier.

2. ADABOOST.MH

For the formal description let $\mathbf{X} = (x_1, \ldots, x_n)$ be the $n \times d$ *observation matrix*, where $x_i^{(j)}$ are the elements of the d-dimensional observation vectors $x_i \in \mathbb{R}^d$. We are also given a *label matrix* $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ of dimension $n \times K$ where $\mathbf{y}_i \in \{+1, -1\}^K$. In *multi-class* classification one and only one of the elements of $\mathbf{y}_i$ is $+1$, whereas in *multi-label* (or *multi-task*) classification $\mathbf{y}_i$ is arbitrary, meaning that the observation x_i can belong to several classes at the same time. In the former case we will denote the index of the correct class by $\ell(x_i)$.

The goal of the ADABOOST.MH algorithm ((Schapire and Singer, 1999), Figure 1) is to return a vector-valued classifier $\mathbf{f} : \mathscr{X} \to \mathbb{R}^K$ with a small *Hamming loss*

$$R_H\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \mathbb{I}\Big\{\operatorname{sign}\big(f_\ell^{(T)}(x_i)\big) \neq y_{i,\ell}\Big\}^1$$

by minimizing its upper bound (the exponential margin loss)

$$R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\big(-f_\ell^{(T)}(x_i) y_{i,\ell}\big), \tag{1}$$

where $f_\ell(x_i)$ is the ℓth element of $\mathbf{f}(x_i)$. The user-defined weights $\mathbf{W}^{(1)} = \big[w_{i,\ell}^{(1)}\big]$ are usually set either uniformly to $w_{i,\ell}^{(1)} = 1/(nK)$, or, in the case of multi-class classification, to

$$w_{i,\ell}^{(1)} = \begin{cases} \frac{1}{2n} & \text{if } \ell = \ell(x_i) \text{ (i.e., if } y_{i,\ell} = 1), \\ \frac{1}{2n(K-1)} & \text{otherwise (i.e., if } y_{i,\ell} = -1) \end{cases} \tag{2}$$

to create K well-balanced one-against-all classification problems. ADABOOST.MH builds the final classifier $\mathbf{f}$ as a sum of *base classifiers* $\mathbf{h}^{(t)} : \mathscr{X} \to \mathbb{R}^K$ returned by a *base learner* algorithm BASE$(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)})$ in each iteration t. In general, the base learner should seek to minimize the *base objective*

$$E\big(\mathbf{h}, \mathbf{W}^{(t)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp\big(-h_\ell(x_i) y_{i,\ell}\big). \tag{3}$$

1. The indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

Using the weight update formula of line 4 (Figure 1), it can be shown that

$$R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \prod_{t=1}^{T} E\big(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\big), \tag{4}$$

so minimizing (3) in each iteration is equivalent to minimizing (1) in an iterative greedy fashion. By obtaining the multi-class prediction

$$\widehat{\ell}(x) = \arg\max_{\ell} f_{\ell}^{(T)}(x),$$

it can also be proven that the "traditional" multi-class loss (or *one-error*)

$$R\big(\mathbf{f}^{(T)}\big) = \sum_{i=1}^{n} \mathbb{I}\Big\{\ell(x_i) \neq \widehat{\ell}(x_i)\Big\} \tag{5}$$

has an upper bound $KR_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big)$ if the weights are initialized uniformly, and $\sqrt{K-1}R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big)$ with the multi-class initialization (2). This justifies the minimization of (1).

2.1 Learning the base classifier

In this paper we use *discrete* ADABOOST.MH in which the vector-valued base classifier $\mathbf{h}(x)$ is represented as

$$\mathbf{h}(x) = \alpha \mathbf{v} \varphi(x),$$

where $\alpha \in \mathbb{R}^+$ is the *base coefficient*, $\mathbf{v} \in \{+1,-1\}^K$ is the *vote vector*, and $\varphi(x) : \mathbb{R}^d \to \{+1,-1\}$ is a *scalar* base classifier. It can be shown that for minimizing (3), one has to choose φ that maximizes the *edge*

$$\gamma = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} v_{\ell} \varphi(x_i) y_{i,\ell}, \tag{6}$$

using the votes

$$v_{\ell} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\big\{\varphi(x_i) = y_{i,\ell}\big\} > \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\big\{\varphi(x_i) \neq y_{i,\ell}\big\}, \\ -1 & \text{otherwise,} \end{cases} \quad \ell = 1, \ldots, K. \tag{7}$$

The optimal coefficient is then

$$\alpha = \frac{1}{2} \ln \frac{1+\gamma}{1-\gamma}.$$

It is also well known that the base objective (3) can be expressed as

$$E\big(\mathbf{h}, \mathbf{W}\big) = \sqrt{(1+\gamma)(1-\gamma)} = \sqrt{1-\gamma^2}. \tag{8}$$

The simplest scalar base learner used in practice is the *decision stump*, a one-decision two-leaf decision tree of the form

$$\varphi_{j,b}(x) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise,} \end{cases}$$

where j is the index of the selected feature and b is the decision threshold. If the features are pre-ordered before the first boosting iteration, a decision stump maximizing the edge (6) can be found very efficiently in $\Theta(ndK)$ time (making the total running time $\Theta\big(nd(\log n + KT)\big)$).

Although boosting decision stumps often yields satisfactory results, state-of-the-art performance of ADABOOST is usually achieved by using decision trees as base learners. In this paper we use an "in-house" implementation that calls the decision stump optimizer as a subroutine. The learner is similar to Quinlan's C4.5 algorithm (Quinlan, 1993), except that we use the edge improvement (instead of Quinlan's entropy-based criterion) to select the next node to split and the threshold b. The base learner has one hyperparameter, the number of the leaves N, which also shows up as a linear factor in the running time.

We also test our approach using a recently proposed base learner that seems to outperform boosted trees (Kégl and Busa-Fekete, 2009). The goal of this learner is to optimize products of simple base learners of the form

$$\mathbf{h}(\cdot) = \alpha \prod_{j=1}^{m} \mathbf{v}_j \varphi_j(\cdot), \tag{9}$$

where the vote vectors $\mathbf{v}_j$ are multiplied element-wise. The learner also calls the decision stump optimizer as a subroutine but in an iterative rather than a recursive fashion. The hyperparameter m, again, appears as a linear factor in the total running time.

3. Using multi-armed bandits to reduce the search space

In this section we will first describe the MAB framework and next we show how bandit algorithm UCB can be used to accelerate the base learning step in ADABOOST.

3.1 Multi-armed bandits

In the classical bandit problem there are M arms that the decision maker can select at discrete time steps. Selecting arm j in iteration t results in a random reward $r_j^{(t)} \in [0,1]$ whose (unknown) distribution depends on j. The goal of the decision maker is to maximize the expected sum of the rewards received. Intuitively, the decision maker's policy has to balance between using arms with large past rewards (exploitation) and trying arms that have not been tested enough times (exploration). The UCB algorithm (Auer et al., 2002a) manages this trade-off by choosing the arm that maximizes the sum of the average reward

$$\bar{r}_j^{(t)} = \frac{1}{T_j^{(t)}} \sum_{t'=1}^{t} \mathbb{I}\{\text{arm } j \text{ is selected}\} r_j^{(t')}$$

and a confidence interval term

$$c_j^{(t)} = \sqrt{\frac{2 \ln t}{T_j^{(t)}}},$$

where $T_j^{(t)}$ is the number of times when arm j has been selected up to iteration t. To avoid the singularity at $T_j^{(t)} = 0$, the algorithm starts by selecting each arm once. We use a generalized version, denoted by UCB(k), in which the best k arms are selected for evaluation, and the one that maximizes the actual reward $r_j^{(t)}$ is finally chosen.

3.2 The application of UCB(k) for accelerating ADABOOST

The general idea is to partition the base classifier space into (not necessarily disjunct) subsets and use MABs to learn the utility of the subsets. In each iteration, the bandit algorithm selects

an optimal subset (or, in the case of UCB(k), a union of subsets). The base learner then finds the best base classifier in the subset, and returns a reward based on this optimal base learner. By reducing the search space of the base learner, we can expect a significant decrease of the complete running time of ADABOOST.

The upper bound (4) together with (8) suggest the use of $-\frac{1}{2}\log(1-\gamma^2)$ for the reward. In practice we found that

$$r_j^{(t)} = 1 - \sqrt{1-\gamma^2}$$

works as well as the logarithmic reward; it was not surprising since the two are almost identical in the lower range of the $[0,1]$ interval where the majority of the edges are. The latter choice has another advantage of always being in the $[0,1]$ interval which is a formal requirement in MABs.

The actual partitioning of the base classifier set depends on the particular base learner. In the case of decision stumps, the most natural choice for UCB is to assign each feature to a subset, i.e., *jth* subset is $\{\varphi_{j,b}(x) : b \in \mathbb{R}\}$. In principle, we could also further partition the threshold space but that would not lead to further savings in the linear computational time since, because of the changing weights $w_{i,\ell}$, all data points and labels would have to be visited anyway. On the other hand, subsets that contain more than one feature can be efficiently handled by UCB(k).

In the case of trees and products we use UCB by considering each tree or product as a sequence of decisions, and using the same partitioning as with decision stumps at each inner node. In this setup we lose the information in the dependence of the decisions on each other *within* a tree or a product.

4. Experiments

4.1 Data set description and data preparation

In *KDD Cup 2009* we were provided with two data sets referred as *Small* and *Large*. These two data sets differed only in the number of features they consisted of. In the *Small* data set there were 190 numerical and 40 categorical features and the *Large* data consisted of 14740 numerical and 260 categorical features for a total of $d = 15000$. Both data sets contained the same 50000 training and 50000 test instances. Each instance had three different labels corresponding to the three tasks of *Churn*, *Appetency*, and *Up-selling*. About 65.4% (2%) of the values were missing in the *Small* (*Large*); we treated them by using an out-of-range value (i.e., we set all missing value to ∞).

Since the three tasks used the same instances, we experimented with both a *single-task* and a *multi-task* approach. In the former, the three classifiers were trained completely separately, whereas in the latter we trained one classifier with a three-element binary label.

First, we trained all of our models using the large feature set, only deleting features with one singular value. We also investigated the utility of the features. Using the info-gain based feature ranker of WEKA package (Witten and Frank, 2005), we found that only a relatively small number of features have positive score for any of task. In the single-task setup we used only those features which had positive score for the given task, and in the multi-task case we used those features which had positive score for at least one of the three tasks. The numbers of remaining features after feature selection are shown in Table 1. We also performed experiments where we applied PCA, but this do not results improvement in performance.

All of the three KDD tasks were very imbalanced in size of classes: all three label sets contained only a small number of positive labels compared to the size of the whole data set. In order to handle this imbalance problem we tried a few initial weighting scheme beside the uniform weighting described in Eq. 2. We found that the best-performing weighting scheme

Table 1: The number of features after applying feature selection.

	single-task			*multi-task*
	Churn	*Appetency*	*Up-selling*	
Small	51	45	65	71
Large	2839	2546	4123	5543

was when both classes received half of the total weight, which meant that the instances from the positive class had higher initial weights than the instances form the negative class.

4.1.1 VALIDATION

During the challenge we validated only the number of iterations using a $60\% - 40\%$ simple validation on the training set. Figure 2 shows the AUC curves for the three tasks. Due to the time limit in the Fast Track we did not validate the number of tree leaves N and number of product terms m. We set the $N = 8$ and $m = 3$ based on the former experiments using our program package (Kégl and Busa-Fekete, 2009). The only remaining hyperparameter we had to choose was the number of best arms to be evaluated in the case of UCB(k). We set k to 50. We also carried out some experiments with a lower value ($k = 20$), but we found that this only slightly influenced the results.

Table 2 shows our official results. The *Churn* and *Up-selling* tasks were evaluated by ROC analysis and the *Appetency* task was evaluated using Balanced Accuracy[2].

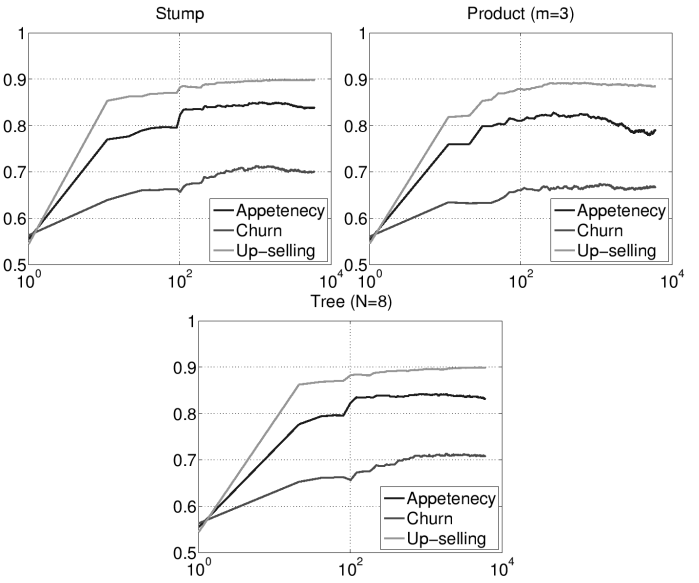


Figure 2: The ROC values vs. number of iterations on the validation set using ADABOOST.MH with Stump, Product, and Tree base learners.

As a post-challenge work we carried out a more comprehensive validation. We used the same $60\% - 40\%$ validation scheme but this time we validated all the parameters in a wide range: $2 \leq N \leq 32$, $2 \leq m \leq 10$, $T \leq 10000$. Similarly to other teams (Miller et al., 2009;

2. http://www.kddcup-orange.com/

Table 2: The validation and evaluation results in Fast Track. The bold face values indicate our final submission.

learner \ data set	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Validation	Evaluation	Validation	Evaluation	Validation	Evaluation
STUMP	0.7424	0.6833	0.7051	0.7359	0.8938	**0.8917**
PRODCT	0.6702	0.6377	0.6999	0.6398	0.8888	0.8665
TREE	0.7088	**0.6819**	0.7424	**0.7216**	0.8956	0.8891
BEST	–	0.7611	–	0.883	–	0.9038

IBM Research, 2009), we found that, relatively to other benchmarks, smaller trees and products worked better on this challenge. In the case of products, it turned out that the optimal number of terms m is in only a few times more than two (see in Tables 3 and 5). The optimal number of iterations was chosen based on the maximum of the ROC values on the validation set calculated in each iteration. In the multi-task case we used the average of the ROC values of the three tasks. Both in the single- and multi-task setup, the ROC values can have a large fluctuation from one iteration to another so we smoothed the learning curves using a moving average filtering with a relative window size of 20%. In general, the optimal numbers of iterations are also relatively small compared to the parameters of the experiments of (Kégl and Busa-Fekete, 2009).

4.2 Combination technique

We also tried to combine the three (stump, product, tree) learners. In case of *Appetency* we applied a simple majority voting. In the tasks where scores were required, we used the discriminant output $\mathbf{f}(x)$ rescaled into $[0,1]$ of the trained models as posterior probabilities and we simply multiplied them.

4.3 Performance evaluation

4.3.1 LARGE DATA SET

Our official results in Fast Track on the *Large* data set using multi-task approach are shown in Table 2. Without the full knowledge of the validation we found that the multi-task approach using feature selection achieves better than single-task one.

Table 3 shows our post-challenge results and the validated parameters. The first four blocks (single/multi-task, feature selection on/off) are followed by results obtained using the combined models. We tried combining only the multi-task or only the single-task classifiers, then we combined all of them. The results revealed a few general trends. The single-task approach was superior in solving the *Churn* task whereas the *Appetency* task seemed to prefer the multi-task approach. In the *Up-selling* problem both were competitive, and compared to the winning results we scored the best on this task.

4.3.2 SMALL DATA SET

In the Slow Track we concentrated only on the *Small* data set. During the challenge our best results were obtained by single-task models using feature selection. Table 4 shows our official submission results and Table 5 shows our post-challenge results together with the validated parameters. We found that the multi-task approach did not work very well; our explanation for this is that the three tasks had different complexities and they needed very different number of iterations, so a single shared stopping time harmed the results. On the other hand, the

Table 3: The post-challenge results for *Large* data set with the validated parameters.

	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Parameters	Evaluation	Parameters	Evaluation	Parameters	Evaluation
MULTI-TASK						
STUMP	$T = 1671$	0.6808		0.7176		0.8844
PRODUCT	$T = 1664$ $m = 2$	0.6877		**0.7388**		0.8817
TREE	$T = 4090$ $N = 3$	0.6918		0.7158		0.8870
MULTI-TASK/FEATURE SELECTION						
STUMP	$T = 369$	0.6802		0.7514		0.8804
PRODUCT	$T = 946$ $m = 2$	0.6749		0.6406		0.8818
TREE	$T = 1501$ $N = 4$	0.6871		0.6165		0.8822
SINGLE-TASK						
STUMP	$T = 1528$	0.7002	$T = 682$	0.5403	$T = 431$	0.7685
PRODUCT	$T = 448$ $m = 2$	0.6873	$T = 401$ $m = 2$	0.5726	$T = 344$ $m = 4$	0.7558
TREE	$T = 1362$ $N = 2$	0.6350	$T = 241$ $N = 5$	0.6719	$T = 1201$ $N = 2$	0.7784
SINGLE-TASK/FEATURE SELECTION						
STUMP	$T = 427$	0.7052	$T = 255$	0.5981	$T = 2260$	0.6449
PRODUCT	$T = 348$ $m = 2$	0.6887	$T = 9177$ $m = 4$	0.6875	$T = 410$ $m = 2$	0.8840
TREE	$T = 296$ $N = 2$	0.71	$T = 2514$ $N = 3$	0.6836	$T = 296$ $N = 2$	0.8762
COMBINED CLASSIFIERS						
MULTI-TASK		0.7197		0.7362		0.8920
SINGLE-TASK		0.7126		0.6312		0.8832
ALL		**0.7245**		0.7013		**0.8944**
WINNERS						
BEST/Fast[a]	–	0.7611	–	0.8830	–	0.9038
BEST/Slow[b]	–	0.7570	–	0.8836	–	0.9048

a. IBM Research (IBM Research, 2009)
b. University of Melbourne (Miller et al., 2009)

single task approach worked very well in these experiments. In fact, the combined single-task post-challenge classifiers outperformed the best official results (among teams that did not used unscrambling).

Table 4: The validation and evaluation results in the Slow Track. The bold face values indicate our final submission.

learner \ data set	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Validation	Evaluation	Validation	Evaluation	Validation	Evaluation
STUMP	0.716862	**0.7258**	0.6712	0.7243	0.85747	**0.8582**
PRODCT	0.692894	0.6816	0.78055	**0.6915**	0.842176	0.8512
TREE	0.695429	0.7158	0.778751	0.6174	0.840568	0.8549
BEST	–	0.7375	–	0.8245	–	0.8620

Table 5: The post-challenge results for *Small* data set with the validated parameters.

	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Parameters	Evaluation	Parameters	Evaluation	Parameters	Evaluation
MULTI-TASK						
STUMP	$T = 600$	0.7046		0.7045		0.8542
PRODUCT	$T = 150$ $m = 2$	0.7128		0.6677		0.8500
TREE	$T = 600$ $N = 2$	0.7255		0.7214		0.8578
MULTI-TASK/FEATURE SELECTION						
STUMP	$T = 740$	0.6273		0.5000		0.6872
PRODUCT	$T = 191$ $m = 2$	0.6188		0.5000		0.6805
TREE	$T = 194$ $N = 3$	0.6223		0.5006		0.6842
SINGLE-TASK						
STUMP	$T = 465$	0.7303	$T = 20$	0.5000	$T = 250$	0.8590
PRODUCT	$T = 191$ $m = 2$	0.7210	$T = 10$ $m = 2$	0.6135	$T = 174$ $m = 2$	0.8543
TREE	$T = 200$ $N = 2$	0.7278	$T = 100$ $N = 3$	0.6494	$T = 185$ $N = 2$	0.8571
SINGLE-TASK/FEATURE SELECTION						
STUMP	$T = 300$	0.7353	$T = 20$	0.5162	$T = 540$	0.8612
PRODUCT	$T = 154$ $m = 2$	0.7237	$T = 10$ $m = 4$	0.5318	$T = 345$ $m = 2$	0.8551
TREE	$T = 188$ $N = 2$	0.7317	$T = 191$ $N = 2$	**0.7207**	$T = 203$ $N = 2$	0.8606
COMBINED CLASSIFIERS						
MUTLI-TASK		0.7040		0.6600		0.8241
SINGLE-TASK		**0.7369**		0.6033		**0.8630**
ALL		0.7316		0.5918		0.8538
WINNERS						
BEST[a]	–	0.7375	–	0.8245	–	0.8620

a. University of Melbourne (Miller et al., 2009)

4.4 Time complexity

Since out goal was to accelerate ADABOOST, we show in Table 6 the time (in minutes) needed for training and testing for 8000 iterations. The numbers indicate that all the models can be trained and tested in less than 10 hours on the *Large* database. For the *Small* data set the whole training time is typically less than an hour.

Table 6: Training and testing running times (in minutes) on the *Large* data set. The number of iterations is $T = 8000$.

Base learner	STUMP	PRODUCT	TREE
Time requirements	274	456	384

5. Discussion and Conclusions

The goal of this paper was to accelerate ADABOOST using multi-armed bandits. Recently, machine learning applications have become the center of interest in which millions of training examples and thousands of features are not uncommon. In this scenario, fast optimization becomes more important than the asymptotic statistical optimality (Bottou and Bousquet, 2008). From this point of view, our approach has solved the tasks well because it reduced greatly the computational complexity of the learning phase. In the official competition our approach achieved competitive results only on the *Up-selling* task. Based on our post-challenge analysis it seems that on small data set we could also have been competitive also on the *Churn* task, but since there was a confusion during the competition (people merged the small and large data sets), we decided to concentrate on the Large set. Our post-challenge results nevertheless confirmed that ADABOOST is among the best generic classification methods on large, real-world data sets.

References

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002a.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

A. Borisov, A. Eruhimov, and E. Tuv. *Tree-Based Ensembles with Dynamic Soft Feature Selection*, volume 207, pages 359–374. Springer, 2006.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20, pages 161–168, 2008.

J.K. Bradley and R.E. Schapire. FilterBoost: Regression and classification on large datasets. In *Advances in Neural Information Processing Systems*, volume 20. The MIT Press, 2008.

G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. In *Proceedings of the 11th European Conference on Machine Learning*, pages 129–141, 2000.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

S. Gelly and D. Silver. Achieving master level play in 9 x 9 computer go. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1537–1540, 2008.

IBM Research. Winning the KDD Cup Orange Challenge with ensemble selection. In *this volume*, pages 21–32, 2009.

B. Kégl and R. Busa-Fekete. Boosting products of base classifiers. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

H. Miller, S. Clarke, Lane S., A. D. Lazaridis, Petrovski S., and Jones O. Predicting customer behaviour: The University of Melbourne's KDD Cup report. In *this volume*, pages 0–0, 2009.

J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. *Journal of Machine Learning Research*, 10:1341–1366, 2009.

P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

Appendix I

KDD Challenge Fact Sheets

Links to the fact sheets :

- Winning the KDD Cup Orange Challenge with Ensemble Selection — IBM Research — URL : http://www.kddcup-orange.com/factsheet.php?id=66
- *A Combination of Boosting and Bagging for KDD Cup 2009 — Fast Scoring on a Large Database* — Xie, Rojkova, Pal, Coggeshall — URL : http://www.kddcup-orange.com/factsheet.php?id=58
- *Predicting customer behaviour: The University of Melbourne's KDD Cup report* — Hugh Miller, Sandy Clarke, Stephen Lane, Andrew Lonie, David Lazaridis, Slave Petrovski, Owen Jones — URL : http://www.kddcup-orange.com/factsheet.php?id=21
- *An Ensemble of Three Classifiers for KDD Cup 2009: Expanded Linear Model, Heterogeneous Boosting, and Selective Naive Bayes* — Hung-Yi Lo, Kai-Wei Chang, Shang-Tse Chen, Tsung-Hsien Chiang, Chun-Sung Ferng, Cho-Jui Hsieh, Yi-Kuang Ko, Tsung-Ting Kuo, Hung-Che Lai, Ken-Yi Lin, Chia-Hsuan Wang, Hsiang-Fu Yu, Chih-Jen Lin, Hsuan-Tien Lin, Shou-de Lin — URL : http://www.kddcup-orange.com/factsheet.php?id=78
- *KDD Cup 2009 @ Budapest: feature partitioning and boosting* — Miklós Kurucz, Dávid Siklósi, István Bíró, Péter Csizsek, Zsolt Fekete, Róbert Iwatt, Tamás Kiss, Adrienn Szabó — URL : http://www.kddcup-orange.com/factsheet.php?id=63
- *Logistic Model Trees with AUC Split Criterion for the KDD Cup 2009 Small Challenge* — P. Doetsch, C. Buck, P. Golik, N. Hoppe, M. Kramp, J. Laudenberg, C. Oberdörfer, P. Steingrube, J. Forster, A. Mauser — URL : http://www.kddcup-orange.com/factsheet.php?id=86
- *Classification of Imbalanced Marketing Data with Balanced Random Sets* — Vladimir Nikulin, Geoffrey J. McLachlan — URL : http://www.kddcup-orange.com/factsheet.php?id=95
- *Application of Additive Groves Ensemble with Multiple Counts Feature Evaluation to KDD Cup'09 Small Data Set* — Daria Sorokina — URL : http://www.kddcup-orange.com/factsheet.php?id=7
- *Accelerating AdaBoost using UCB* Róbert Busa-Fekete, Balázs Kégl — URL : http://www.kddcup-orange.com/factsheet.php?id=79