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, the prediction of three aspects of customer behavior for a telecommunications service provider. Our most successful method was a Logistic Model Tree with AUC as split criterion using predictions from boosted decision stumps as features. This was the best submission for the "Small Challenge" that did not use additional data from other feature sets. A second approach using an AUC-optimized weighted linear combination of several rankings scored slightly worse with an average AUC of 0.8074. From the given 230 features, we extracted additional binary features and imputed missing values using SVMs and Decision Trees.

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

1. Introduction

In this work, we describe our approach to the "Small Challenge" of the KDD cup 2009, the prediction of customer behavior for a mobile phone network provider. While exploring several classification and regression methods, we focused on tree-based models. The 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 was organized as a student Data Mining lab course at the Chair for Computer Science 6 of RWTH Aachen University. Eight students took part in the lab. The remainder of the 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, the 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 exists a small data set of 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.

The feature names allowed no intuitive interpretation of the values and categorical features were obfuscated. In 211 out of 230 features values were missing, 18 features contained no values at all. Since no training example was a member of more than one class we assumed the target classes to be disjoint and 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, since 16.5% of the training instances belonged to one of the three classes of interest. Only 1.8% of the data was labeled with the class Appetency.

2.1 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 also 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 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.

3.2 Missing Values

A large number of values was missing from 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. To impute the missing values of the 188 features we categorized them by a selection criterion. Using this criterion, each feature was handled by a method according to the category it belongs to. We handled features where less than 5% of the values were missing also by straight forward imputation methods, like mean or mode imputation for numerical features or inserting the most frequent value for categorical features.

Selection Criterion To select an adequate method to address the missing values for an individual feature, we introduced the following measure: Let f be a feature with missing values and M_f the number of distinct feature values in f. Further assume that there are N_f samples where the value of this feature is not missing. Then we define the Missing Value Ratio (MVR) as $N_f/(M_f F)$, where F is the total number of features minus one, as we leave out f.

If we formulate the imputation of the missing values as a new learning task, MVR(f) gives the average number of samples per possible target value and feature available for f. We used the MVR to partition the features into three categories. 37 features with MVR(f) ≥ 3 belong to category A, 25 features with $1 \leq$ MVR(f) < 3 belong to category B and the remaining 126 features belong to category C. Most features are contained in the category C. These features can not be addressed by simple imputation techniques, so we tried different algorithms to estimate the values as described below.

Imputation Methods Category A features provided a sufficiently large number of occurrences per value and we classified them with support vector machines using an RBF kernel and low cost penalties. These parameters were tuned by splitting off about 5% of the set. Category B and C were imputed by a regression tree implemented in MATLAB. We declared all features with less than ten feature values to be categorical. The implementation uses an entropy splitting criterion and we chose the minimum number of observations for splitting a node to be 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. Based on the ranking methods described in Subsection 3.4 we selected the best imputation method for these features.

3.3 Feature Generation

Beside the features created by the imputation methods described above, we produced additional features, like 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. The large amount of those features already gave rise to focus our attention to partitioning algorithms.

We produced 5,318 new features in this way. We had to binarize the categorical ones, since most of our classifiers only support numerical input, and got more than 10,000 features in total.

3.4 Feature Selection

Many features generated by the methods described above were redundant and not all of them were useful for our classifiers. In order to build feature sets with a high prediction 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 (first introduced by Quinlan (1986)) in order to find most relevant features. The ratio of feature f for class c having M distinct values $f_1, ..., f_M$ with support $|f_i|$ is defined as:

$$\mathrm{R}(f,c) = \frac{\mathrm{H}(f,c) - \sum_{i=1}^{M} \frac{|f_i|}{M} \cdot \mathrm{H}(f_i,c)}{-\sum_{i=1}^{M} \frac{|f_i|}{M} \cdot \log_2\left(\frac{|f_i|}{M}\right)}$$

where $H(x,c)$ denotes the entropy for the distribution of class c.

Secondly, we used the likelihood-ratio as a ranking criterion. In order to avoid noise 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 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 high ranked features. In the following we will denote this set by X_O . 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 classifiers. This included parameter optimization as well as the selection of an appropriate set of features and preprocessing steps for each classifier.

4.1 MLP

One of our first experiments on the KDD cup 2009 task were done using multilayer perceptrons (MLP) implemented in the Netlab (Nabney (2001)) toolbox with GNU Octave. All categorical attributes were enumerated by occurrence in the whole data set and all features were normalized to $[0, 1]$. 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 random initialization of the weights, ten runs with identical parameter sets were made. 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 high number of generated features all experiments were done using a linear kernel.

4.3 LMT

We implemented a 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 fits a LogitBoost model (Friedman et al. (1998)) to the assigned data, i.e. it performs an iterative fitting 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 reduce the tree size and to increase the generalization of the model.

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

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

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

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) intoduced 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. We then can use the metric defined by Ferri et al. (2002) to compute the AUC:

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

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. Eq. (1) therefore 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 defined as:

$$E_T(v) = \begin{cases} E_L(v) & \text{if } v \text{ is a leaf} \\ E_T(v_{\text{left}}) + E_T(v_{\text{right}}) & \text{otherwise} \end{cases}$$

Finally, the pruning values are calculated by the ratio of the local error and the tree error in a node. This was also modified for the AUC criterion, but we did not observe an improvement using this pruning procedure. In the present task, 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 $O(NF^2d + k^2)$, where N denotes the number of training samples, d is the depth of the tree and k the number of nodes in the unpruned tree. Note that the internal cross-validation generates a large constant factor in the complexity, such that unpruned trees without this factor are much faster in training.

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 learner. For categorical features, the weak learners form 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, this computation is based on the distribution estimated by the boosting procedure. Given these weights, we select the split point minimizing the objective function

$$Z = 2 \sum_l \sum_{c=1}^{C} \sqrt{W_+^{lc} W_-^{lc}}$$

where C is the number of classes. For details and particular 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 decision tree 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 classification procedure for the test sample x and class c. Then we obtain the posterior probability as defined by Niculescu-Mizil and Caruana (2005):

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

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

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 interpolation techniques like accumulation gave no performance gain compared to the results obtained by boosted decision stumps. We therefore implemented two more complex methods to combine several classifiers, which are presented in this section.

Rank combination We implemented a number of simple methods to combine the output of several classifiers $k \in K$ 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.

$$r_{\text{avg}}(c|x) = p_{\text{avg}}(c|x) = \frac{1}{|K|} \sum_k p_k(c|x)$$

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

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

where Z_k is a normalization constant so that $\max_x 1/Z_k \sum_y \delta\big(p_k(c|x) - p_k(c|y)\big) = 1$ 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 combination by replacing $p_k(c|x)$ with $p_{w_k,k}(c|x) = w_k p(c|x)$. We implemented a gradient descent method to optimize the weights w_k with respect to the AUC criterion.

Downhill-Simplex Beside the rank combination we also used the Downhill-Simplex method proposed by Nelder and Mead (1965), which is a particular instance of the Simplex method and can be used for nonlinear optimization problems (Press et al. (1992)). As shown in Section 6, this method was able to achieve further improvements over the results given by the combination of boosted decision stumps with an LMT.

Stacking One general purpose technique for combining several approaches is to use 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,

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 1: Boosted decision stumps on the X_M set including additional binary features shown as AUC

which is presented in the next section. In the following, the feature set X_O including 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 at the end of this section shows the comparison of their performance on a common feature set.

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 exhaustive search. The highest AUC scores for Appetency were achieved with 100 hidden nodes, for Churn with 300 hidden nodes and for Up-selling with 500 hidden nodes. This behavior indicates different complexities for the classification on each separate class. First experiments with imputed missing values yielded average AUC scores of 0.76. After adding a binary feature set to 1 if the value of feature 126 was -30, the average AUC increased to 0.78. Additional tweaking of the parameters did not outperform the result of 0.78. 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 given results we declared all features with less than 10 distinct as categorical and performed 25 boosting iterations. In our first experiment we did not apply any missing value handling, 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.

The table shows that the internal handling of missing values outperforms our imputed values. 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, the decision stumps could not find them by threshold split points.

LMT Based on the predictions by boosted decision stumps we generated new features as described in Section 5 and appended them to the X_M set used in the boosted decision

Features	Configuration	Appetency	Churn	Up-selling	Score
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**

Table 2: Logistic Model Tree on the X_O set presented as AUC

Features	Configuration	Appetency	Churn	Up-selling	Score
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 3: Logistic Model Tree on the X_S set presented as AUC

stumps to create the X_S set. Although many learning algorithms yielded good results on the X_S set, the only one which was able to generate a further improvement was the LMT. The training converged after few iterations. 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 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. This observation is consistent with the general success of partitioning algorithms in the KDD cup 2009. Many tree induction methods were suitable to the given problem. Since the stacked features 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 given 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 submission. 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 originally obtained results 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

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

Table 4: Comparison of classifiers 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 5: Combination of boosted decision stumps, MLP, SVM and LMT on a common feature set of 399 numerical features, sorted by averaged AUC score

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.

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 for a mobile phone carrier. 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 show that tree-based methods are more capable of handling or ignoring incomplete data. As additional preprocessing, we generated binary features for categorical values.

Our most successful method was a Logistic Model Tree with AUC as split criterion using predictions from boosted decision stumps as features. This was the best submission for the "Small Challenge" of the KDD cup 2009 that did not use additional data from other feature sets. With an average AUC score of 0.8081, it was ranked 35th among all 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.

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.

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.

W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

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.