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

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

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

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:

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

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. 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 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 probabili-

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

ties $p_k(c|x)$. 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:

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

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 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.

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.

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: AUC scores of boosted decision stumps on the X_M 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**

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

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

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

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

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 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.

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

Table 6: Winners of the KDD cup 2009

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.