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/

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 $(\boldsymbol{x}_i, y_i)_{i=1}^{l}$, where $\boldsymbol{x}_i \in R^n$ is the feature vector and $y_i \in \{1, -1\}$ is the class label. The j-th feature of $\boldsymbol{x}$ 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 $\boldsymbol{w}$ Assume that there is only one feature t with missing values and we add one feature so the final model looks like

$\hat{\boldsymbol{w}}^T = [\boldsymbol{w}^T \ \hat{w}_t]$. Then, the decision value of any instance $\boldsymbol{x}$ is

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

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:

$$\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_k} \sum_{y=1}^{k} \sum_{t=1}^{n} |w_{yt}| + C \log \sum_{i=1}^{l} \frac{e^{\boldsymbol{w}_{\bar{y}_i}^T \boldsymbol{x}_i}}{\sum_{y=1}^{k} e^{\boldsymbol{w}_y^T \boldsymbol{x}_i}}, \tag{1}$$

where k is the number of classes ($k = 4$ here), and $\bar{y}_i = \{1, 2, 3, 4\}$ indicates the transformed class of $\boldsymbol{x}_i$. We consider an l1-regularized solver here because it is suitable for noisy datasets.

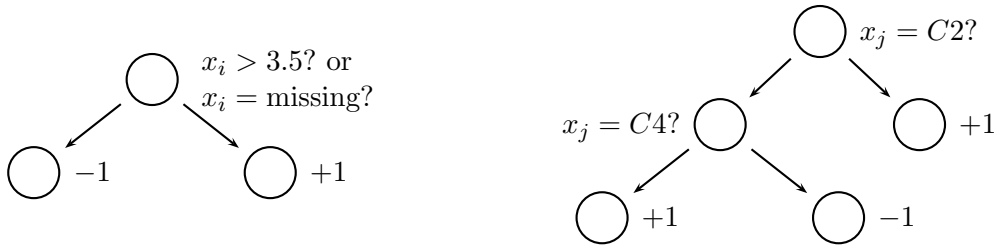
We choose the maximum entropy model since it delivers the probabilistic interpretation of data. To predict the probability of a test instance $\bar{\boldsymbol{x}}$, we use:

$$p(\bar{y}|\bar{\boldsymbol{x}}) = \frac{e^{\boldsymbol{w}_{\bar{y}}^T \bar{\boldsymbol{x}}}}{\sum_{y=1}^{k} e^{\boldsymbol{w}_y^T \bar{\boldsymbol{x}}}}. \tag{2}$$

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 AUCs 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.

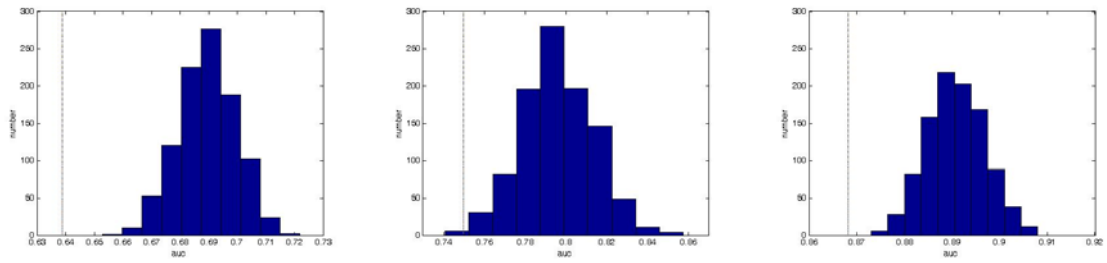
We design an experiment with RankLR to analyze the 10% test set:

1. We train a randomly selected subset of 40,000 samples and obtain the weight vector $\boldsymbol{w}$.
2. Use the $\boldsymbol{w}$ 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 $\boldsymbol{w}$ to calculate AUC(D_{val}).
4. Repeat step 3 for 1,000 times and obtain 1,000 values of AUC(D_{val}).

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

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



(a) Churn (b) Appetency (c) Upselling

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

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 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.

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

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

Base Classifier	Churn		Appetency		Upselling		
	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

similar to RankSVM (Herbrich et al., 2000) but replacing the SVM part by Logistic Regression. Let $\boldsymbol{x}_i$ denote an example with label $y_i = +1$ (positive) and $\boldsymbol{x}_j$ 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.

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

$$\text{where} \qquad E_{ij}(\boldsymbol{w}) = \log\Big(1 + \exp\Big(-\boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j)\Big)\Big).$$

We use stochastic gradient descent (Zhang, 2004) which updates the weight vector $\boldsymbol{w}$ by

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

After training, $\boldsymbol{w}^T\boldsymbol{x}$ 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

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

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

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. 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.

R. Herbrich, T. Graepel, and K. Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.

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

T. 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.