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.

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.

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

Table 1: Parameters producing the best result for bagged 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 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

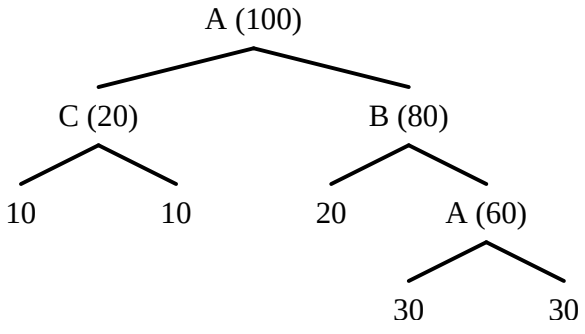

Figure 1: Sample decision tree splits the data using features A, B and C. Node labels show the percentage of the train set data points passing through each node.

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

Table 2: Top 20 features and their importance scores

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 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.

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**
        $\text{Tree}_{\text{attempt1},i} = \text{Tree}_i^{(\alpha_j,n-1)}$
      $\text{Tree}_{\text{attempt1},n} = 0$
      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**
          $\text{Tree}_{\text{attempt2},i} = \text{Tree}_i^{(\alpha_{j-1},n)}$
        Converge(α_j,n,train,$\text{Tree}_{\text{attempt2},1}, \ldots, \text{Tree}_{\text{attempt2},n}$)

      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**
        $\text{Tree}_i^{(\alpha_j,n)} = \text{Tree}_{\text{winner},i}$

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 = $\{\mathbf{x}, y - \sum_{k \neq i} \text{Tree}_k^{(\alpha,N)}(\mathbf{x})\}$
      $\text{Tree}_i^{(\alpha,N)} = \text{TrainTree}(\alpha, \text{newTrainSet})$
  until (change from the last iteration is small)

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.

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
- 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.

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

Table 3: Performance on validation set

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.

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

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

Table 4: Competition results for small data set without unscrambling

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 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 0.7161.

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.