MaxStack: Robust Classification with an Ensemble Classifier

Alen Lukic
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
alukic@andrew.cmu.edu

Poorna Omprakash
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
pompraka@andrew.cmu.edu

ABSTRACT

When a machine learning method is used by someone with no special expertise in the field, the method's robustness is an important consideration; reasonable performance with minimal tuning for a given problem is a highly desirable property. In this paper, we utilize a classifier's error rate relative to a Naive Bayes approach as a quantifiable measure of robustness, and describe a particular ensemble learning method which we call MaxStack, which combines the FunctionalTree, LogitBoost, Dagging, AdaBoost, LogisticRegression and BayesNet Weka classifiers, that is robust according to this measure. In general, we observed that making use of this ensemble classifier resulted in a significant robustness improvement. Without MaxStack, the best performance obtained was an average error of 0.826 relative to Naive Bayes. Using MaxStack resulted in an average error ratio of 0.618.

1. INTRODUCTION

Machine learning is generally a complicated endeavor. There is no formal way of determining which learning method works most well for a given problem. Singleton classifiers work well for data sets with specific properties and less well for others. For example, logistic regression works well only if the data is linearly separable. As such, there is no way to select a singleton classifier a priori which is guaranteed to work well on a given dataset.

Juxtaposed with this reality is that fact that, in many settings, the availability of a ubiquitous classifier which requires little or no tuning and works well on given data is highly desirable. Machine learning methods are used in numerous domains and often by non-experts who do not necessarily possess the training required to select a robust learner for their problem. A robust learner, as discussed in this paper, refers to a machine learning technique which (1) produces minimal classification error for a given problem and (2) performs well for data sets with differing properties which generally limit the performance of certain singleton classifiers.

Given these requirements, we look to ensemble methods. On a high level, the goal of ensemble learning is to achieve better performance through the combination of several constituent algorithms than any of the constituent algorithms could achieve individually. In this paper, we explore several ensemble methods and compare their performance against that of tuned singleton classifier learners on a set of Weka-formatted data sets, including a vanilla voted committee algorithm and a batch stacking method we call MaxStack.

2. BACKGROUND

Our proposed ensemble method extends a baseline Vote ensemble method. The baseline method makes use of models from the following classifiers to make predictions:

2.1 LogitBoost

LogitBoost is a boosting algorithm that casts the AdaBoost algorithm into a statistical framework. The LogitBoost algorithm can be derived by considering the AdaBoost algorithm as a generalized additive model and then applying the cost functional of logistic regression.
The LogitBoost algorithm minimizes the logistic loss:

$$\sum_i log(1 + exp(-y_i f(x_i)))$$

Here, f is an additive model of the form:
$f = \sum_t \alpha_t \ h_t$

LogitBoost was originally formulated by formulated by Jerome Friedman, Trevor Hastie, and Robert Tibshirani.[1]
In Weka, LogitBoost is implemented by the LogitBoost() class.
This class performs the additive logistic regression mentioned above. It performs classification using a regression scheme as the base learner, and can handle multi-class problems.
The various Weka options available for use with LogitBoost are:
-Q
Use resampling instead of reweighting for boosting.
-P <percent>
Percentage of weight mass to base training on. (default 100, reduce to around 90 speed up)
-F <num>
Number of folds for internal cross-validation. (default 0 –

no cross-validation)
-R <num>
Number of runs for internal cross-validation. (default 1)
-L <num>
Threshold on the improvement of the likelihood. (default -Double.MAXVALUE)
-H <num> Shrinkage parameter. (default 1)
-S <num>
Random number seed. (default 1)
-I <num>
Number of iterations. (default 10)
-D
If set, classifier is run in debug mode and may output additional info to the console
-W
Full name of base classifier.

Here, the various options can be set to vary different parameters such as number of folds of cross validation, the number of iterations, etc.

AdaBoost is a boosting algorithm that is closely related to LogitBoost. It can be used in conjunction with many other learning algorithms to improve their performance. The classifiers that AdaBoost uses can be weak, implying that there is a high error rate. However, the final model will be improved as long as the performance of the classifiers is such that the error rate is smaller than 0.5 for binary classification.
AdaBoost runs a series of rounds t = 1,...,T. For each round, it generates and calls a new weak classifier. For each call, a distribution of weights D_t is updated. This indicates the importance of examples in the data set for the classification. On each round, the weights of each incorrectly classified example are increased, and the weights of each correctly classified example are decreased. In this way, the new classifier focuses on the examples which have not been correctly classified so far.
AdaBoost was formulated by formulated by Yoav Freund and Robert Schapire.[2]

2.2 FunctionalTree

Functional Trees[3] are a type of classification trees that could have logistic regression functions at the inner nodes and/or leaves. The algorithm can deal with binary and multi-class target variables, numeric and nominal attributes and missing values.
Functional Trees use combinations of attributes at decision nodes, leaf nodes, or both nodes and leaves in regression and classification tree learning. Functional Trees use a framework that combines a univariate decision tree with a linear function by means of constructive induction. Decision trees derived from this framework can use decision nodes with multivariate tests, and leaf nodes that make predictions using linear functions. Multivariate decision nodes are built when growing the tree, while functional leaves are built when pruning the tree.

Functional Trees in Weka are implemented by the FT() class. There are multiple options availale for use with Functional Trees in Weka. They are:
-B
Binary splits (convert nominal attributes to binary ones)
-P
Use error on probabilities instead of misclassification error for stopping criterion of LogitBoost.
-I <numIterations>
Set fixed number of iterations for LogitBoost (instead of using cross-validation)
-F <modelType>
Set Funtional Tree type to be generate: 0 for FT, 1 for FTLeaves and 2 for FTInner
-M <numInstances>
Set minimum number of instances at which a node can be split (default 15) -W <beta>
Set beta for weight trimming for LogitBoost. Set to 0 (default) for no weight trimming.
-A
The AIC is used to choose the best iteration.

These options enable us to tune the Functional Tree classifier. We can control the number of iterations, whether we wish to have logistic regression at the inner nodes and/or leaves etc.

2.3 Logistic Model Trees

A Logistic Model Tree[4] (LMT) is a classification model that combines Logistic Regression and Decision Tree Learning. The basic LMT induction algorithm uses cross-validation to find a number of LogitBoost iterations that does not overfit the training data.
LMT extends the idea of using Linear Regression and Logistic Regression with trees. The linear regression variant of the tree has linear regression model at the leaves of the tree. The logistic regression variant (as seen with LogitBoost above) produces a Logistic Regression model at every node in the tree. The node is then split using the C4.5 criterion. Each LogitBoost invocation is started from the results in the parent node. Finally, the tree is pruned.
In addition to the above functionality, the LMT Algorithm also performs cross-validation to determine the number of LogitBoost iterations such that the training data is not overfit.
Thus, we can say that a Logistic Model Tree is a classification tree with logistic regression functions at the leaves. The algorithm can deal with binary and multi-class target variables, numeric and nominal attributes and missing values.
In Weka, Logistic Model Trees are implemented using the MLT() class.
There are various options that can be used with Logistic Model Trees in Weka. These options can be used to tune the classifier by varying different parameters to suit our requirements. In Weka, the different options that can be used with Logistic Model Trees are:
-B
Binary splits (convert nominal attributes to binary ones)
-R
Split on residuals instead of class values
-C
Use cross-validation for boosting at all nodes (i.e., disable heuristic)
-P
Use error on probabilities instead of misclassification error for stopping criterion of LogitBoost.

-I <numIterations>
Set fixed number of iterations for LogitBoost (instead of using cross-validation)
-M <numInstances>
Set minimum number of instances at which a node can be split (default 15)
-W <beta>
Set beta for weight trimming for LogitBoost. Set to 0 (default) for no weight trimming.
-A
The AIC is used to choose the best iteration.

These options enable us to control parameters such as the number of iterations, and whether or not to use cross validation. We can also control the minimum number of instances at which a node can be split.

2.4 Logistic Regression

Logistic Regression is a classification model that is closely related to the two aforementioned classification algorithms, i.e. Functional Trees and Logistic Model Trees.
Logistic Regression is used to predict a class label, given a set of features. That is, it is used to estimate empirical values of the parameters in a qualitative response model. The probabilities describing the possible outcomes of a single trial are modeled as a function of the predictor variables, using a logistic function. It is also easy to train LR classifiers using standard constrained or unconstrained optimization techniques. Furthermore, incorporating a regularization parameter will ensure that LR is robust to noise.
Thus, we can say that Logistic Regression is an approach to learning functions of the form $f : X \rightarrow Y$, or $P(Y|X)$ in the case where Y is discrete-valued, and $X = \{X_1 ... X_n\}$ is any vector containing discrete or continuous variables.
Logistic Regression assumes a parametric form for the distribution P(Y|X), then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where Y is boolean is:

$$P(Y = 1|X) = \frac{1}{1 + exp(w_0 + \sum_{i1}^{n} w_i X_i)}$$

Furthermore, $P(Y = 0|X) = 1 - P(Y = 1|X)$
In the ensemble method proposed in this paper, we do not directly make use of Logistic Regression. However, LR forms the basis for the two aforementioned classifiers, and is thus an important part of the ensemble classifier proposed.

2.5 Vote

Vote[5] is a ensemble classifier that combines multiple classifiers to predict labels. The Vote meta classifier works as follows:
Each of the classifiers used in the Vote meta classifier predicts a nominal class label for a test sample. The label that was predicted the most is then selected as the output of the vote classifier. For the baseline algorithm, the Vote classifier we used combined Functional Tree, LogitBoost and Logistic Model Trees. We obtained nominal class labels for each of the datasets for the FT, LB and LMT classifiers. We then selected the label that was predicted the most often to be the output of the vote classifier.

In Weka, the Vote classifier is implemented using the Vote() class. There are multiple options available for use with the Vote classifier. The options we used to tune the Vote meta classifier are:
-S <num>
Random number seed. (default 1)
-B <classifier specification>
Full class name of classifier to include, followed by scheme options. May be specified multiple times. (default: "weka.classifiers.rul
-D
If set, classifier is run in debug mode and may output additional info to the console
-P <path to serialized classifier>
Full path to serialized classifier to include. May be specified multiple times to include multiple serialized classifiers. Note: it does not make sense to use pre-built classifiers in a cross-validation.
-R <AVG|PROD|MAJ|MIN|MAX|MED>
The combination rule to use (default: AVG)

Here, we can tune various parameters, such as the classifiers to be included, the metric for combining results from different classifiers, etc.

3. PROPOSED LEARNING METHOD: MAXSTACK

We developed a straightforward stacking method we call MaxStack. MaxStack simply evaluates the test instances using each of the aforementioned trained models individually. The combiner algorithm is simple: the predictions produced by the model whose error rate relative to Naive Bayes are returned as MaxStack's final predictions.

Prior to developing MaxStack, we first tried selecting amongst the singleton algorithms which performed most well on average. However, due to the shortcomings of singleton algorithms already discussed, the average error ratio was relatively high (see the M2 result). Next, we instead attempted to use a single ensemble method: dagging. This actually ended up producing even worse results than using the singleton classifier (see M3 results). Given thes results, we shifted toward the use of a voted ensemble classifier. This classifier used two tree algorithms (logistic model tree and functional decision tree) and one ensemble method (logistic boosting) as part of its ensemble. Here we saw a notable performance boost over M2 (see M4 results). A fundamental shortcoming of the voted committee is that if only a specific classifier works well on a particular data set, then the performance of the voted committee will be poor, because a majority of the other classifiers are likely to produce the incorrect label for a test instance. From this observation we created MaxStack. MaxStack is structurally similar to a voted committee, but it selects only the classifier which produces the minimum training error in order to classify a corresponding test example.

Each of the component algorithms utilizies 10-fold cross validation in order to ensure that the trained model does not overfit the training data. Hence, the predictions which minimize the error on the training set after cross-validation are likely to correspond to the model which will perform most well on a blind test set.

In order to increase the likelihood that a good model is produced for the training data, MaxStack utilizes several different types of classification algorithms in its ensemble, including Bayesian approaches (Bayesian network), functional algorithms (logistic regression), tree-based algorithms (functional decision trees and random forest), and other ensemble methods (logistic boosting, AdaBoost, bagging, and dagging). This makes MaxStack a general-purpose algorithm which has a higher likelihood of performing well on any given training data than other classifiers which perform well for only certain kinds of data.

We take several measures to increase both the classification performance and speed of MaxStack. As mentioned, cross-validation is applied to each component algorithm in order to ameliorate the effects of overfitting. Furthermore, we utilizie Weka's built-in parameter selection capabilities to produce optimal tuning parameters for each component algorithm given the training data. Finally, we take advantage of the inherent parallelizability of MaxStack. Since the constitutent algorithms do not depend on each other, we train each algorithm and classify the test data in parallel. Although this does not reduce the asymptotic time complexity of the algorithm, in practice this optimization is highly useful.

4. EXPERIMENTAL RESULTS

The following are the list of datasets used in the experiments:

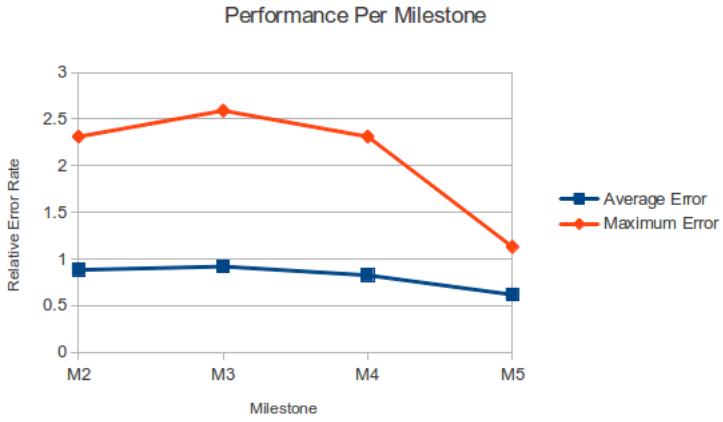
Datasets: Development and Tuning

Name	# Train	# Test	# Features
Anneal	646	339	38
Audiology	226	149	226
Autos	168	98	398
Balance-Scale	429	215	625
Breast-Cancer	207	108	569
Colic	275	148	368
Credit-a	483	248	690
Diabetes	529	267	8
Glass	158	86	9
Heart-c	222	119	13
Hepatitis	129	76	19
Hypothyroid	2219	1245	27
Ionosphere	235	156	34
Labor	38	19	16
Lymph	99	48	18
Mushroom	5443	2690	22
Segment	1547	762	19
Sonar	138	68	60
Soybean	458	227	35
Splice	2139	1053	61
Vehicle	567	279	18
Vote	291	144	16
Vowel	663	327	13
Zoo	68	33	17

The table below shows the performance of the tested classification technique at each milestone on the randomized set of 24 data sets provided in M5. The same data sets are used to allow for direct performance comparison.

Performance Per Milestone

Milestone	Average Error	Maximum Error
M2	0.883	2.312
M3	0.920	2.590
M4	0.826	2.312
M5	**0.618**	**1.130**



5. CONCLUSIONS

MaxStack significantly outperforms all of the other approaches, with a 33.6% average error rate reduction and a 104.6% maximum error rate reduction. Also notable is the fact a tuned voted committee outperforms a single-best singleton classifier by 6.9%. The results confirm our intuition that a well-tuned ensemble classifier generally works more well than a well-tuned singleton classifier, and that a voted ensemble which dynamically selects the single best-performing constituent learner for a data set performs significantly more well than a voted ensemble which averages classification results from its constituent learners.

6. REFERENCES

[1] J. Friedman, T. Hastie and R. Tibshirani. Additive Logistic Regression: a Statistical View of Boosting. *http://www-stat.stanford.edu/~jhf/ftp/boost.ps, 1998.*

[2] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. *In Proceedings of Thirteenth International Conference on Machine Learning, 1996.*

[3] J. Gama. Functional Trees. *Machine Learning, Vol. 55, 2004.*

[4] N. Landwehr, M. Hall and E. Frank. Logistic model trees. *Machine Learning, Vol. 95, 2004.*

[5] J. Kittler, M. Hatef, R. P.W. Duin and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence, No. 3 Vol. 95, 1998.*