Tuning the AODE Classifier

Background

The Aggregated One-Dependence Estimator (AODE)[1] is a Naive-Bayes-like classifier in the sense that the conditional probabilities for each class are computed using Bayes' Law. Unlike Naive-Bayes which makes a strong assumption about conditional independence, AODE makes the slightly weaker assumption that each feature is conditionally dependence on *one* other feature. “Parent” features are selected for each feature to make a One-Dependence model, and several of these models are aggregated together to make a final model.

Tuning

The main parameter for AODE is whether or not to use M-estimates for smoothing. If so, a choice about the weight needs to be made. We tested each dataset with and without M-estimates, then varied the weight for the M-estimate from 1 to 5. Typically, AODE gives higher robustness values when using M-estimates with a weight of 1. However, there are a few data sets that respond poorly to using M-estimates, regardless of the weight. If the performance of a model using AODE without M-estimates was better than baseline Naive-Bayes, using an M-estimator tended to improve the performance. If AODE performed worse than Naive-Bayes (perhaps in situations where the conditional independence assumption is actually met) then adding an M-estimator decreased performance.

Another parameter is a frequency limit for the parent features. When set to larger values, this prevents rare feature values from becoming a super parent. Through cross-validated parameter selection, we found that the default value of 1 was the best setting for all but one datasets.

Final Settings

AODE only works with nominal values. To use the classifier on datasets with numeric attributes, we first apply a normalization filter to the numeric data and then discretize with equal-width discretization using optimal number of bins for each dataset using the leave-one-out estimate of estimated entropy. This method of discretization gave better results than using equal-frequency binning.

1 G. Webb, J. Boughton & Z. Wang (2005). Not So Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning, 58*, 5-24.

Tuning the MultilayerPerceptron Classifier

Background

MultilayerPerceptron is Weka's implementation of a neural network. The network weights are learned by Backpropogation. Neural networks were one of the first research areas in machine learning.

Tuning

One varied parameter was the number of epochs. This parameter controls the number of times the weights are adjusted during training. We determined robustness values for each dataset with 50, 100, 250, 500, 750, and 1000 epochs. We found that more epochs did not necessarily mean better performance, most likely due to overfitting.

To combat this, we experimented with using a validation set. Training would stop when the error rate on the validation set increased after E consecutive epochs. We experimented with holding out 20% of our training data as a validation set and trying E=1,5,10,15,20,30,50 for 2000 epochs. We used internal cross-validation via a CVParameterSelection to find the best threshold for each classifier.

The most important parameter of this classifier is the structure of the hidden layers. Due to the complexity involved in networks with more than 1 hidden layer, we limited ourselves to networks with one hidden layer of various sizes. We experimented with the default Weka settings of the number of hidden nodes: the number of attributes, the number of classes, and the sum and mean of these two values. We manually found the best configuration for each dataset. We found that some datasets were very sensitive to this parameter with robustness values fluctuating by more than .2. On the other hand, some datasets were insensitive to this parameter with the error rate remaining the same regardless of the network configuration. We conclude that this is a parameter to be set on a per-dataset basis.

Final Settings

We apply a normalization and discretization filter across all numeric attributes across all datasets before training.

Tuning the LogitBoost Classifier

Background

The LogitBoost algorithm was created by Friedman, Hastie, and Tibshirani in their seminal paper “Additive Logistic Regression: a Statistical View of Boosting.” It is a variant of AdaBoost where we maximize the log-likelihood and weight instances based on a logit function obtained by regression.

Tuning

We have the option to resample the training set during boosting rather than reweighting instances. That is, instead of using all instances weighted by varying amounts, we only use a subset of instances chosen from a distribution. We found that using resampling increases the error rate in all cases.

When determining the reweighting regression function, we have the opportunity to perform an internal cross-validation. We found that on certain datasets, doing so yields better results while other datasets do better without this internal validation. We find the best parameter on a per-dataset level by using a cross validated parameter selection wrapper.

We can also change a “shrinkage” parameter, which reduces the weight of each generation. The algorithm was found to be sensitive to this parameter. After experimentation, we set the shrinkage to the default value of 1 for best results.

Perhaps the most important feature is the number of iterations of boosting (ie, how many classifiers will vote in the final classification). Setting this to be low reduces overfitting since less training data is used repeatedly. As expected, we found that LogitBoost is sensitive to this parameter. We use a cross-validated parameter selection wrapper to discover the optimal number of iterations to minimize overfitting while maximizing performance.

Final Settings

When selecting the optimal combination of internal folds and iterations using a cross-validated parameter selection wrapper, we found that typically the error rate was reduced by not doing internal validation and instead setting the number of iterations appropriately. Therefore, in the final model, we only optimize the number of iterations, allowing that to reduce overfitting instead of internal validation.

We apply a Normalization filter to all datasets prior to the parameter selection. Certain models perform better when the numeric parameters are discretized, so we apply a standard discretization filter when it improves the performance on the cross-validated training set.

Summary

Among our three classifiers, LogitBoost proved to be the most robust. Already the most robust of the three out-of-the-box, we were able to improve the average robustness from .732 to .655 while keeping the maximum robustness constant at 1. We were able to improve AODE slightly from .95 to .89 but its (however weakened) conditional independence assumption keeps it from being significantly better than a Naive Bayes baseline. MultiLayerPerceptron was naturally robust with the default values. The best hidden layer configuration obtained by cross-validation on the training set was not always the best one for the training set. It seems that the network configuration itself is sensitive to overfitting. When the optimal network configuration was combined with the optimal validation threshold, our results were worse than an untuned instanced of the classifier.