Project Milestone 3

Machine Learning Fall 2013

Team Name: JZL
Team Members: Vineet Joshi, David Zhou, Luigi Leung
Team Member Andrew IDs: vineetj, dzhou, lleung

1. TUNING THE FT CLASSIFIER

1.1

FT (Functional Tree) is a supervised learning algorithm where a complex problem is divided into simpler problems as a decision tree where each node of the tree could be a logistic regression function to classify the data. It can deal with multi-class target variables as well as missing values. (Normal name: Functional Trees, Weka class name: FT)

Reference:
Gama, Joao. "Functional Trees." *Machine Learning*. 55.3 (2004): 219-250. Print.

1.2

The parameter we experimented for FT with are -M (Minimum number of instances at which a node can be split) and -F (Functional Tree type).

-M: If we increase the minimum number of instances at which a node can be split, the runtime is about the same (around 5 seconds on a laptop). FT seems to be rather sensitive to the minimum number of instances at which a node can be split. After experiments, we fixed this parameter at 12, which seems to achieve the lowest average error rate for the 12 datasets. With a runtime of around 5 seconds, we can afford to set this parameter to be give us the best result. Over-fitting is possible with this goal in achieving the lowest error rate, but after testing on all 12 datasets, we can safely rule out this possibility.

-F: The functional tree type parameter has three choices and can be FT, FTInner or FTLeaves. FT is takes into consideration of the linear combination of both inner nodes and leaves, while the other two are a simplified versions restricting to linear combinations of inner nodes or leaves. The runtime of the all three variations of this parameter is similar (around 5 seconds on a laptop). Theoretically, FTInner and FTLeaves should run faster due to their simplier nature. According to Gama, the author of this Weka algorithm and article on Functional Trees, the performance of the three variations are similar with the full FT model having marginal advantages but the performance is dependent on the dataset.

The runtime for training this classifier was ~5 seconds for all benchmarks.

1.3

The final classifier learner was constructed by fixing -F first to find the best -M in steps from 10 to 20 and then, fixing the best -M to find the best -F for all three variations. The optimal settings for FT that we found is “-M 12 -F 2”. On the 12 datasets, this setting has an average of 20.3% error rate and performs better than Naive Bayes with an error ratio (FT_error/NB_error) of 0.681.

2. TUNING THE PART CLASSIFIER

2.1

PART (Partial Decision Trees) builds a partial decision tree in each iteration, and turns the best leaf in an iteration into a rule using a separate-and-conquer strategy for building a rule, removing instances that it covers, and continue until no instances are left. (Normal name: Partial Decision Tree, Weka class name: PART)

Reference:
Eibe Frank, Ian H. Witten: Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning, 144-151, 1998

2.2

-C (Confidence threshold for pruning): Defines the level of confidence necessary for the algorithm to proceed through the decision tree without backtracking. Varying this parameter by lowering confidence threshold should decrease runtime by allowing the algorithm to proceed more boldly through the decision tree, and should do better against overfitting by considering training data more loosely, and should have no effect on classifier size relative to making a very stringent confidence threshold. In our benchmarks, the confidence level is varied from default in few training sets with no gains in robustness. As a consequence, this variable was unchanged in most of the models.

-M (minimum number of objects per leaf): Defines the minimum number of objects generated at each leaf when choosing between rules at a node. Increasing minimum number of objects per leaf by varying this parameter increases learning time, but should do worse against overfitting and have no obvious effect on reducing classifier size, because highly precise rules fit closely to the training data and does not increase the number of rules added to the tree, unless each rule removes fewer instances per tree. For the most part, varying this parameter did not prove optimally robust for most benchmarks.

-R (whether to use reduced error pruning): Reduced error pruning sets training data aside to determine whether to drop the tail of a rule and uses a stopping criterion to prune a rule as a heuristic. According to literature, it decreases learning time and classifier size by reducing the size of rule sets by pruning tails. By leaving out training data for this purpose, it also does better against overfitting by using less training data and reducing the risk of incorporating too much training data into the model. Varying this parameter improved error rates in half of the benchmarks as assessed by cross-validation, with unreliable improvements in performance compared to not using error pruning. No value of R was fixed.

The learning time for this classifier was within one second for all benchmarks.

2.3

The final classifier learner was constructed by fixing -M and -R first to find the best -C in 5 steps from 0.15 to 0.35, and then, fixing the best -C and -R to find the best -M in steps from 1 to 4, and then, fixing the best -C and -M to find the best -R for its two variations. The optimal settings for PART that we found is “-C 0.3 -M 2”. On the 12 datasets, this setting has an average of 21.2% error rate and performs better than Naive Bayes with an error ratio (FT_error/NB_error) of 0.747.

3. TUNING THE SMO CLASSIFIER

3.1

Sequential Minimal Optimization (SMO) algorithm is used for training a support vector classifier. In this implementation all missing values are replaced globally and nominal attributes are transformed into binary ones. All attributes are also normalized by default. (Normal name: Sequential Minimal Optimization, Weka class name: SMO)

Reference:
J. Platt: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, 1998.

3.2

-P (The epsilon for round-off error): Epsilon gives an upper bound on the relative error due to rounding in floating point arithmetic.

-L (The tolerance parameter): While training the SMO classifier, an iterative process is used to optimize the support vector function. This optimisation is generally controlled by using a tolerance parameter. Training is terminated when the gradient of the optimized function is less than or equal to tolerance.

-M (whether to fit logistic models to SVM outputs or not)

In this approach, for each pair of class values, we use either the training data or an internal cross-validation on the training data and create a new data set that contains one of the input attributes along with a binary class attribute. This resulting two column data set is then used as training data for logistic regression. *

* http://list.waikato.ac.nz/pipermail/wekalist/2011-October/053668.html

The runtime for this classifier was 10-12 seconds for all benchmarks.

3.3

The final classifier learner was constructed by fixing -L and -M first to find the best -P in 10 steps of from 1.0e-14 to 1.0e-10, and then, fixing the best -P and -M to find the best -L in 10 steps from 1.0e-6 to 1.0e-2, and then fixing the best -P and best -L to find the best -M by comparing to two -M options. The optimal settings for SMO that we found is “-P 1.0e-14 -L 1.0e-4”. On the 12 datasets, this setting has an average of 21.4% error rate and performs better than Naive Bayes with an error ratio (FT_error/NB_error) of 0.833.

4. TUNING THE DAGGING CLASSIFIER

4.1

Dagging is a meta classifier that creates disjoint sample data that are processed from a base classifier (we used the default base classifier). Each result from the sampled data is then processed through a vote meta classifier. Dagging is similar to bagging where bagging randomly sample with replacement while dagging randomly samples without replacement. (Normal name: Dagging, Weka class name: Dagging)

Reference:
Ting, K. M., Witten, I. H.: Stacking Bagged and Dagged Models. In: Fourteenth international Conference on Machine Learning, San Francisco, CA, 367-375, 1997.

4.2

Similar to tuning the SMO classifier,
-P (The epsilon for round-off error): Epsilon gives an upper bound on the relative error due to rounding in floating point arithmetic.

-L (The tolerance parameter): While training the SMO classifier, an iterative process is used to optimize the support vector function. This optimisation is generally controlled by using a tolerance parameter. Training is terminated when the gradient of the optimized function is less than or equal to tolerance.

-M (whether to fit logistic models to SVM outputs or not)

In this approach, for each pair of class values, we use either the training data or an internal cross-validation on the training data and create a new data set that contains one of the input attributes along with a binary class attribute. This resulting two column data set is then used as training data for logistic regression. *
* http://list.waikato.ac.nz/pipermail/wekalist/2011-October/053668.html

The runtime for this classifier was 12-14 seconds for all benchmarks.

4.3

The final classifier learner was constructed by fixing -L and -M first to find the best -P in 10 steps of from 1.0e-14 to 1.0e-10, and then, fixing the best -P and -M to find the best -L in 10 steps from 1.0e-6 to 1.0e-2, and then fixing the best -P and best -L to find the best -M by comparing to two -M options. The optimal settings for Dagging that we found is “-P 1.0e-14 -L 1.0e-6”. On the 12 datasets, this setting has an average of 24.8% error rate and performs better than Naive Bayes with an error ratio (FT_error/NB_error) of 0.984.

5. SUMMARY

Among the classifiers we tested, FT (Functional Trees) was the most robust by tuning each of two parameters M (minimum number of instances at which a node can be split) and F (choice of functional tree type) using cross validation. Tuning this classifier produced marked improvements in performance in nearly all of the benchmarks. 10 to 19 minimum number of instances at each node and the tree types FT and FTLeaves were tuned to our 12 benchmarks.