Analysis of Supervised Learning Algorithms 1

Description of first classification problem 1

Classification using Command Line Interface – ml program suite 1

Classification using the weka tool : 3

A. Run the J48 Decision Tree Classifier Algorithm: 3

B. Run the Decision Tree Classifier Algorithm with Boosting : 4

C. Run the KNN Algorithm 5

D. Run the SVM Algorithm 6

E. Run the MultiLayerPerceptron (ANN) algo 7

Conclusion 8

Description of second classification problem 8

Classification using Command Line Interface – ml program suite 8

Classification using the weka tool : 8

A. Run the J48 Decision Tree Classifier Algorithm: 8

B. Performance improvement through Ensemble learning (by applying boosting algorithms) 9

C. Run the KNN Algorithm 10

D. Run the SVM Algorithm with PolyKernel and Normalized PolyKernel 11

E. Run the MLP Algorithm 12

Conclusion 12

# Analysis of Supervised Learning Algorithms

## Description of first classification problem

The Goal is to find the best supervised learning algorithm for classifying the credit rating based on the given training data.

Data source : <http://staff.informatics.buu.ac.th/~ureerat/321641/Weka/Data%20Sets/GermanCredit/>

Training Data : CreditRating-train.arff , Test data : CreditRating-test.arff

Description : CreditRating-description.pdf

The best algorithm can be used to predict the Credit Rating (good or bad) of a new customer.

Each instance in the training and test file contains the customer’s financial and personal profile data.

Why have we chosen this dataset ?

> The dataset is non-trivial and complex. It contains both categorical and numerical features. Data has lots variations in its

### Classification using Command Line Interface – ml program suite

I have created the program suite from scratch by leveraging weka api to provide an interactive command line tool for quick comparison of accuracy of algo.

Git repo : https://github.com/kaniska/machine\_learning\_library

> algo-analysis-cli/

> java -jar mlprograms-1.0.0.jar CreditBalance-train.arff CreditBalance-test.arff

Welcome to ML Programs Suite !

Which Algorithm do you want to run ?

Type 0 to exit.

Type 1 for Decision Tree - J48

Type 2 for KNN

Type 3 for SVM

Type 4 for MLP

Type 5 for Boosting Decision Tree

<===========================>

1

Apply J48 algo

What's the minimum number of instances per leaf ?

2

Type 0 - to exit

Type 1 - to run J48 (unpruned).

Type 2 - to run J48 (pruned).

Type 3 - to run J48 (reduced-error pruning)

Type 4 - to run J48 (folding on unpruned tree)

Type 5 - to run J48 (folding and pruning)

2

Apply J48 algo with simple pruning

Number of leaves: 22

Size of the tree: 31

Results

======

Correctly Classified Instances 499 86.7826 %

Incorrectly Classified Instances 76 13.2174 %

…………

---- Now apply the same classifier to the test data ---

Results

Correctly Classified Instances 0 0%

Incorrectly Classified Instances 25 100%

…………

Type 0 - to exit

Type 1 - to run J48 (unpruned).

Type 2 - to run J48 (pruned).

Type 3 - to run J48 (reduced-error pruning)

Type 4 - to run J48 (folding on unpruned tree)

Type 5 - to run J48 (folding and pruning)

===========================

3

Apply J4.8 algo with reduced-error pruning

Number of leaves: 17

Size of the tree: 25

Results

======

Correctly Classified Instances 485 84.3478 %

Incorrectly Classified Instances 90 15.6522 %

…………   
---- Now apply the same classifier to the test data ---

Results

Correctly Classified Instances 23 92%

Incorrectly Classified Instances 2 8%

………….

* That way we can compare the accuracy of an algo against both Training and Test samples.
* We can run different variations of the same algo (say J48) or select another algo from the main menu ( J48, KNN, ANN, MLP etc.).
* After running the program multiple times with CreditRating dataset on diff algo we clearly see that J48 tree with reduced-error pruning provides best accuracy for classifying the given test data.
* Similarly we can find the best classification algo for another interesting dataset with 3 output classes .
  + java -jar mlprograms-1.0.0.jar auto-mpg.arff auto-mpg-test.arff
* The command-line tool can be customized to allow user specify certain parameters like learning\_rate for MLP etc.

Next we shall run same algorithm with different variations using Weka as it also helps to create comparative charts.

### Classification using the weka tool :

Version : weka-3-6-12 , Mac compatible Weka : weka-3-6-12-oracle-jvm.dmg

#### A. Run the J48 Decision Tree Classifier Algorithm:

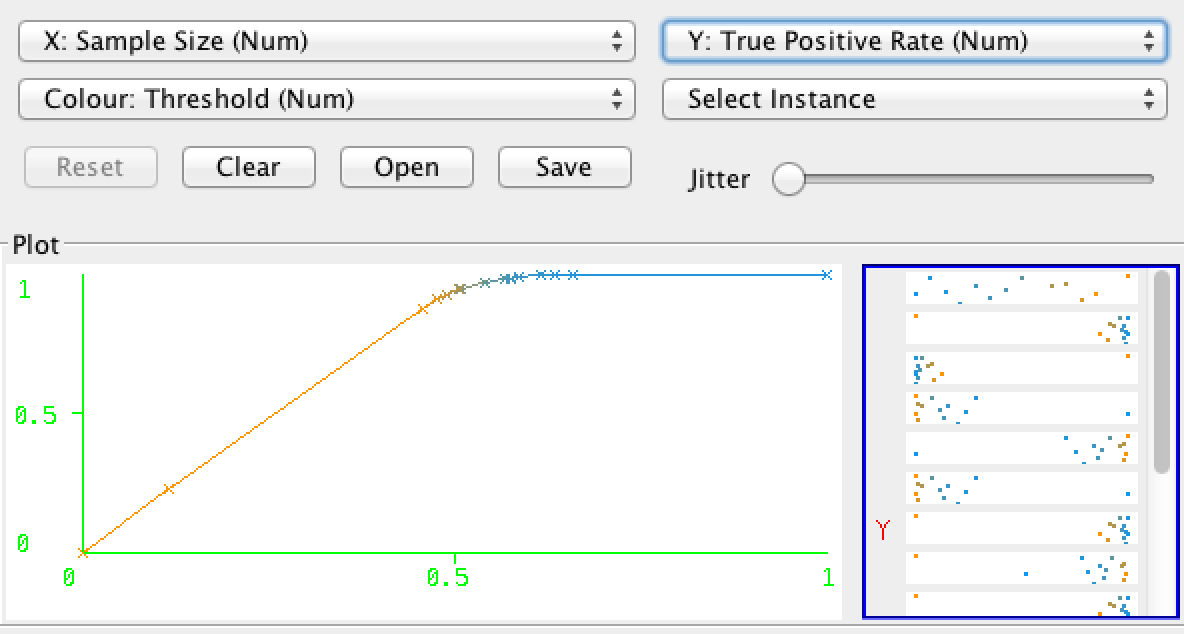
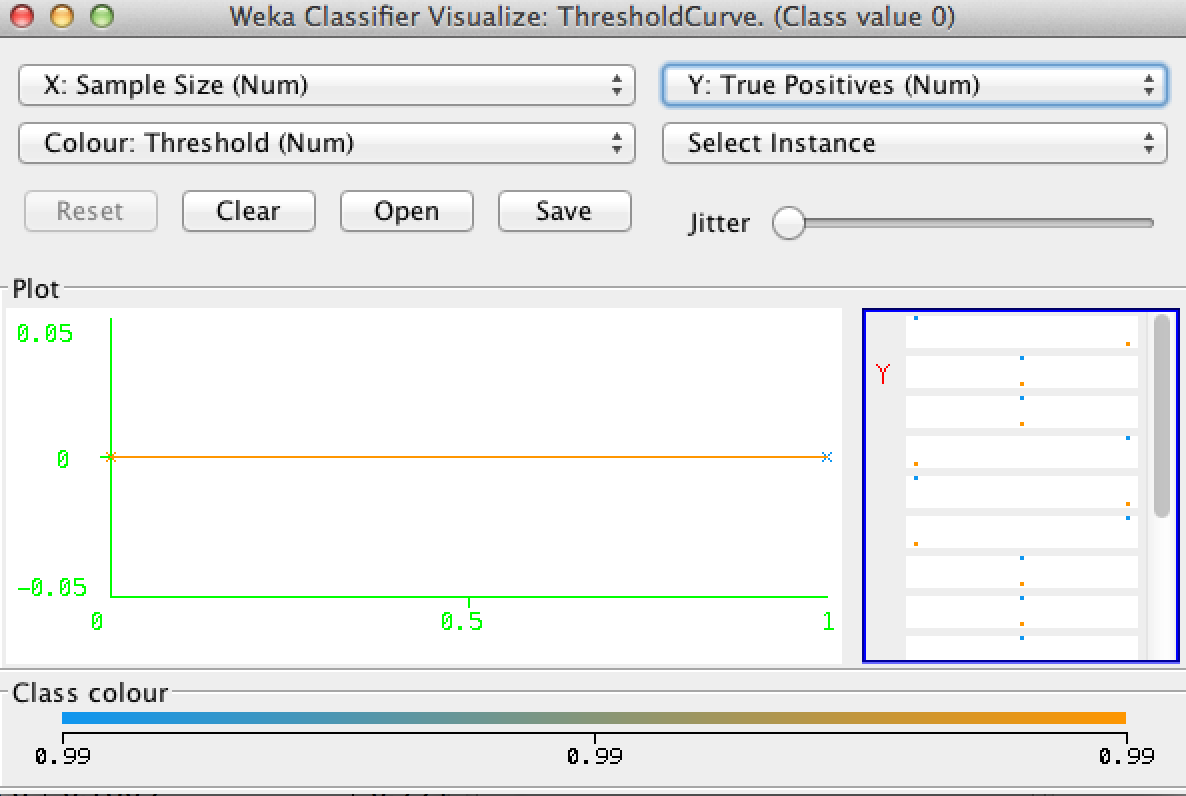
* Unpruned tree correctly classify 95.7% training samples, but fails to classify the test samples (0%).
* So clearly unpruned tree overfits the tree.
* Simple pruning reduces size of the tree but fails to classify the test samples correctly as shown in this table.

|  |  |  |
| --- | --- | --- |
|  | Performance of J48 tree with simple pruning | Performance of J48 tree with simple pruning against test samples |
| Number of Leaves | 22 | 22 |
| Size of the tree | 31 | 31 |
| Correctly Classified Instances | 86.78 % | 0% |
| Incorrectly Classified Instances | 13.22 % | 100% |

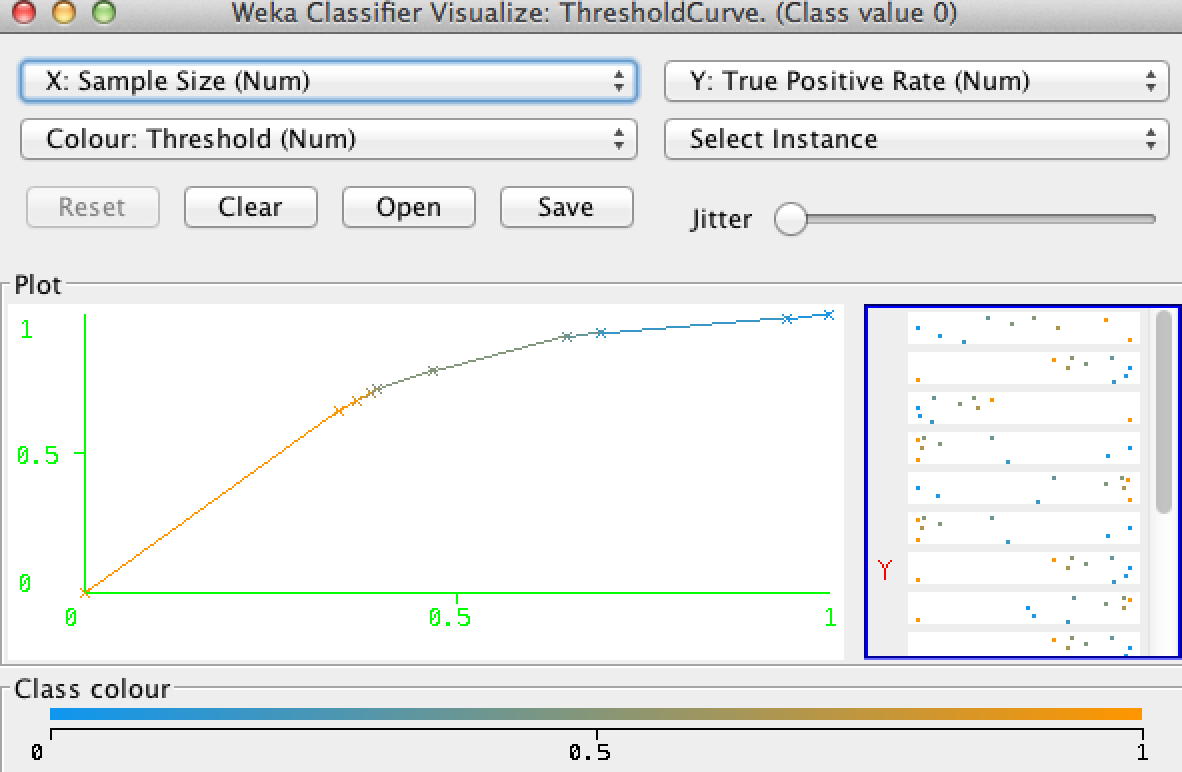
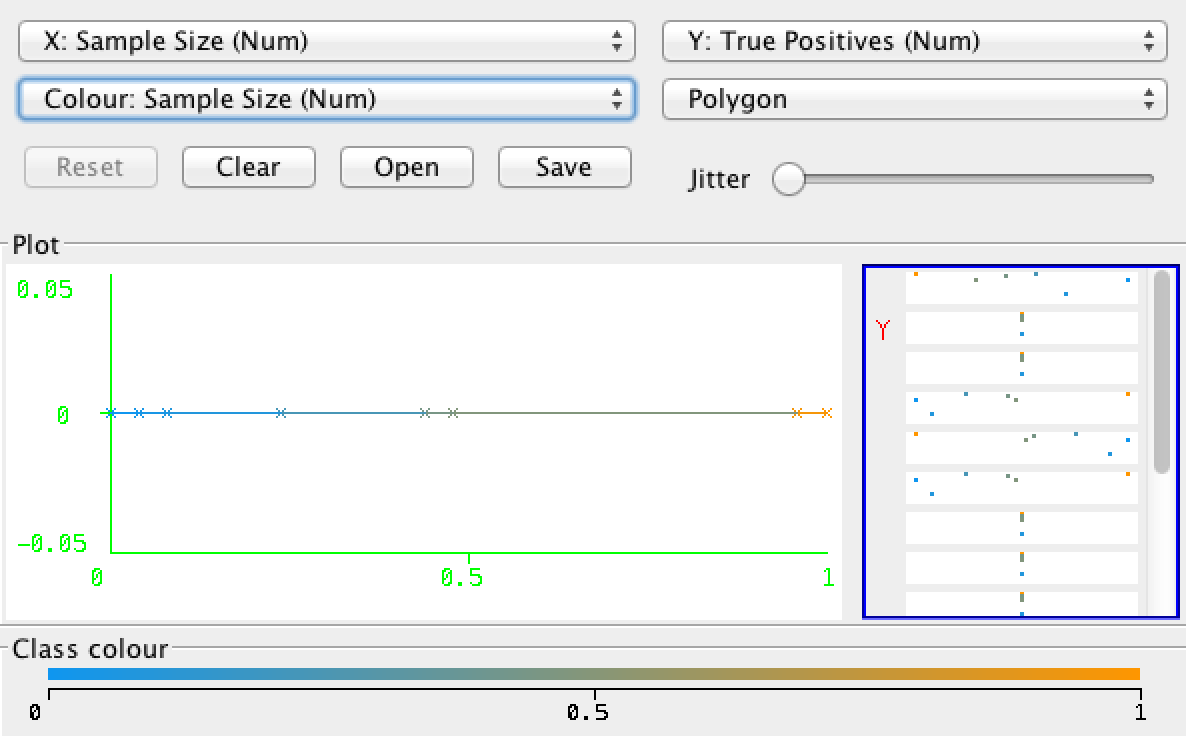
* **Applying reduced Error Pruning** on test samples lowers the complexity of the final classifier as well as offers **better predictive accuracy** by the reduction of overfitting and removal of sections of a classifier that may be based on noisy or erroneous data.
  + It lowers the memory footprint (from #of leaves 100 reduced to #of leaves 17, #size of tree reduced to 25 from 149).
  + It offers much better classification rate (92%)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Unpruned Tree performance | Unpruned Tree performance against Test samples | Performance with reduced-error pruning | Performance of reduced-error pruning against test samples | Performance of reduced-error pruning and Cross-validation (20 folds) | Performance of Unpruned Tree with Cross-validation (10 folds) | Performance with reduced-error pruning and 66% Split | Performance of Unpruned tree with 66% Split |
| Number of Leaves | 100 | 0 | 17 | 17 | 17 | 100 | 17 | 17 |
| Size of the tree | 149 | 25 | 25 | 25 | 25 | 149 | 25 | 25 |
| Correctly Classified Instances | 95.65 % | 0 % | 84.35 % | **92%** | 79.47% | 77.22% | 80% | 77.44% |
| Incorrectly Classified Instances | 4.35 % | 100 % | 15.65% | **8%** | 20.53% | 22.78% | 20% | 22.56% |
| Mean absolute error | 0.0654 | 0.9949 | 0.2053 | 0.2549 | 0.2545 | 0.2339 | 0.2304 | 0.2435 |
| Root mean squared error | 0.1809 | 0.9949 | 0.334 | 0.3403 | 0.4004 | 0.4508 | 0.4049 | 0.454 |
| Relative absolute error | 13.11% | 190.71 % | 41.14 % | 48.86% | 51.00 % | 46.87 % | 46.12% | 48.745% |

* Performance graph for unpruned tree against Training data and Test data respectively.

* Performance graph for reduced-error pruning against Training data and Test data respectively.

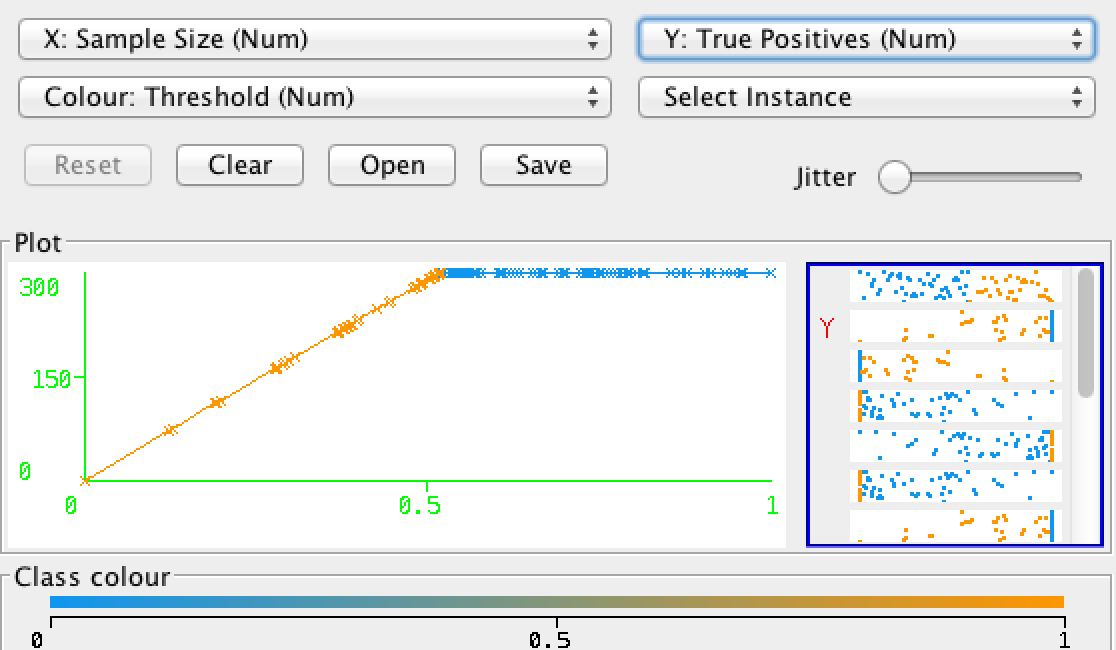
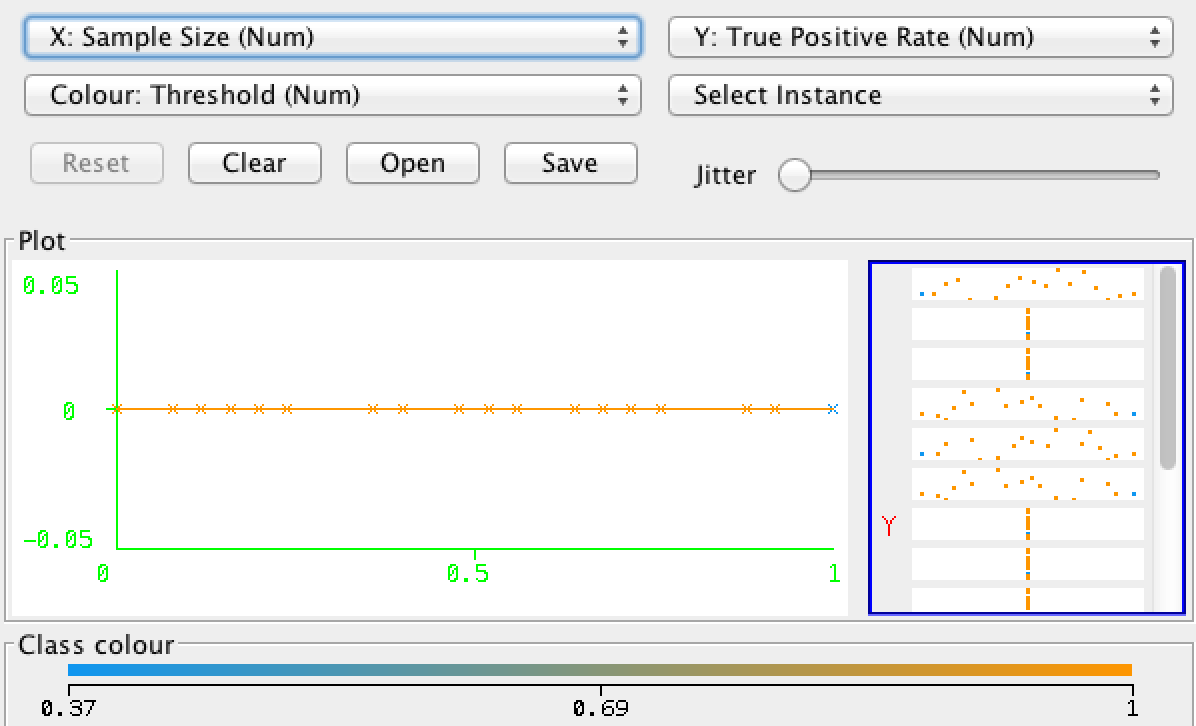
 

#### B. Run the Decision Tree Classifier Algorithm with Boosting :

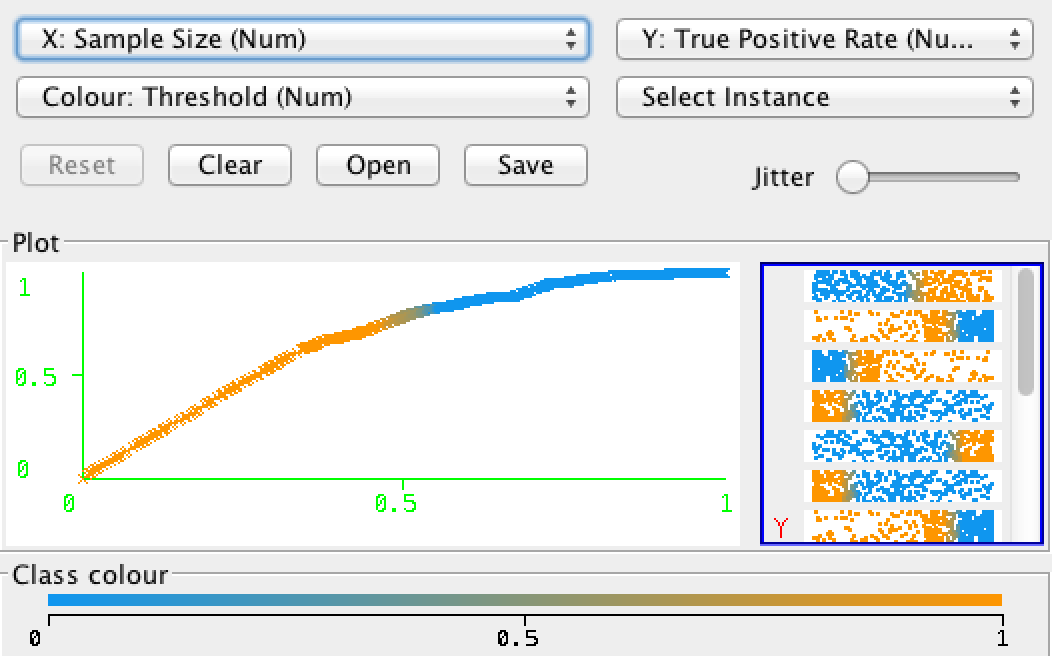
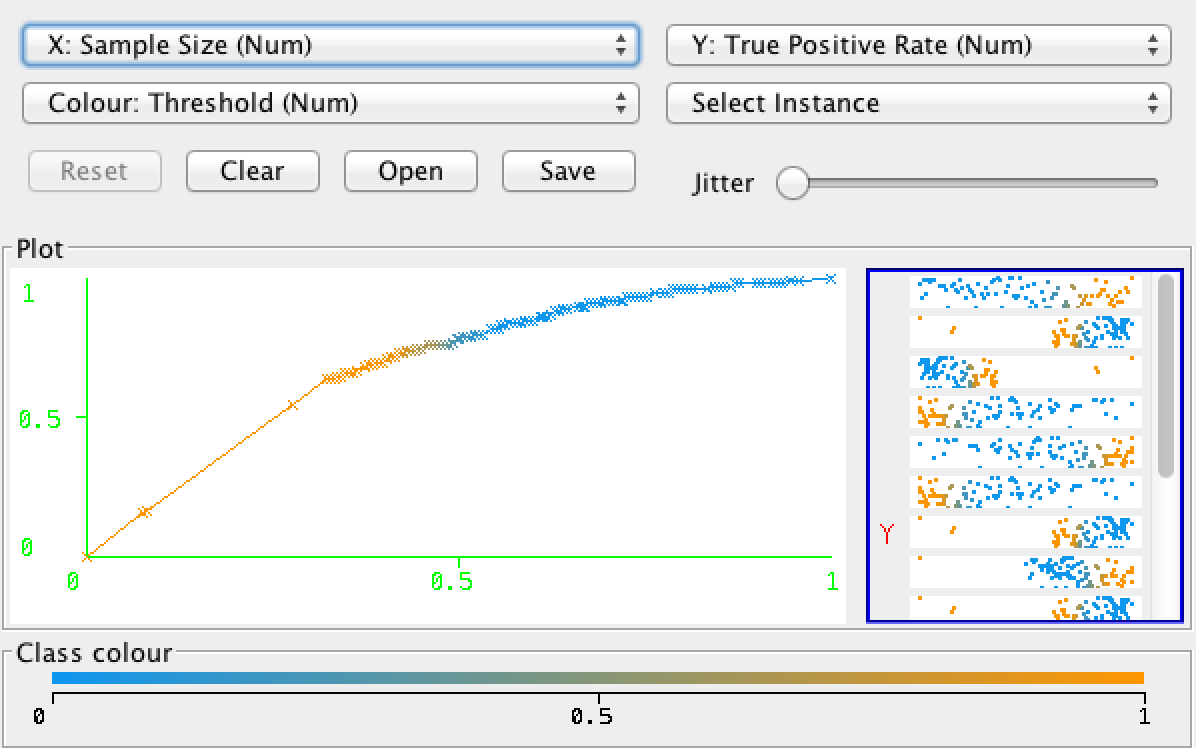
* MultiBoostAB and LogitBoost do not offer good performance over the test samples.
* AdaBoostM1 with reduced error pruning offers accuracy 80% which is less than the accuracy of reduced-error pruning algo without boosting (92%).
  + applying boosting on reduced-error pruning algo does not decrease the size of the tree !

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Performance of AdaBoostM1 with simple pruning against training samples. | Performance of AdaBoostM1 with simple pruning against test samples. | Performance of AdaBoostM1 with reduced-error pruning and Cross-validation. | Performance of AdaBoostM1 with reduced-error pruning with 66% Split |
| Number of Leaves | 575 | 47 | 47 | 47 |
| Size of the tree | 0 | 80 | 80 | 80 |
| Correctly Classified Instances | 100 % | 8% | 79.13% | 79.49% |
| Incorrectly Classified Instances | 0 % | 92% | 20.87% | 20.52% |
| Mean absolute error | 0 | 0.9492 | 0.2085 | 0.2081 |
| Root mean squared error | 0 | 0.9643 | 0.4365 | 0.4313 |
| Relative absolute error | 0.0014 % | 190.7113 % | 41.7695 % | 41.6601 % |

* Performance Graph of applying AdaBoostM1 on simple pruning against training samples and test samples.

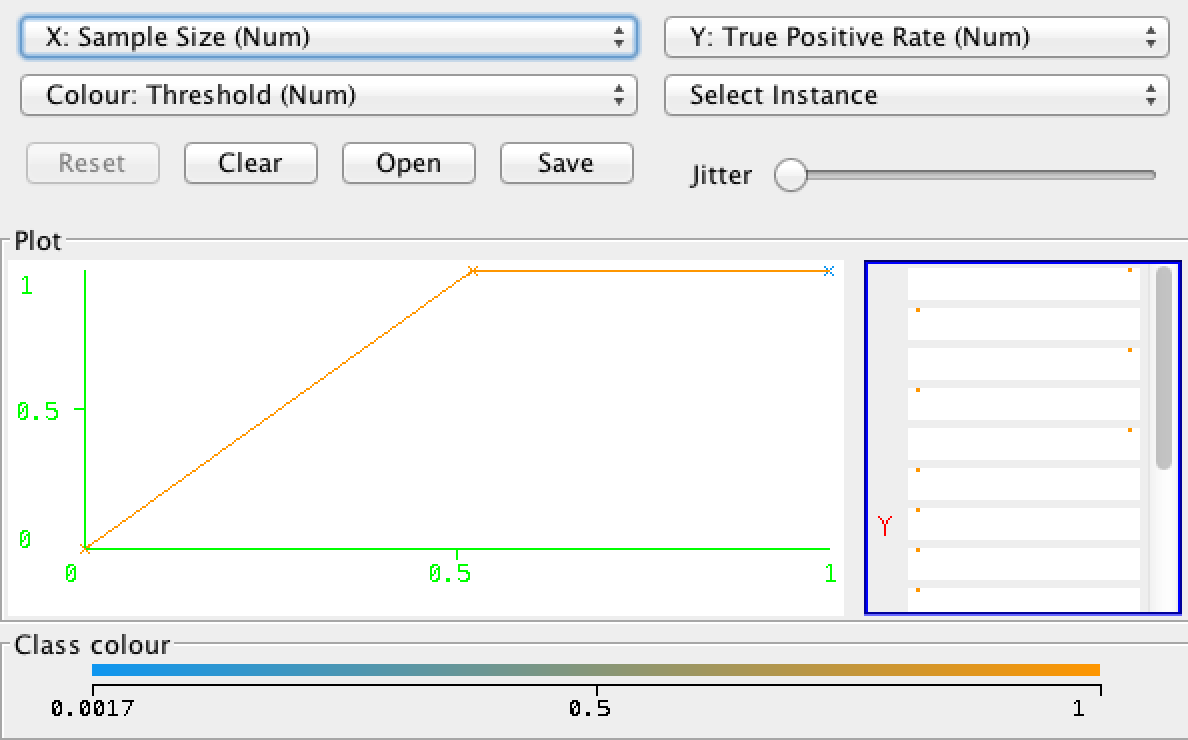
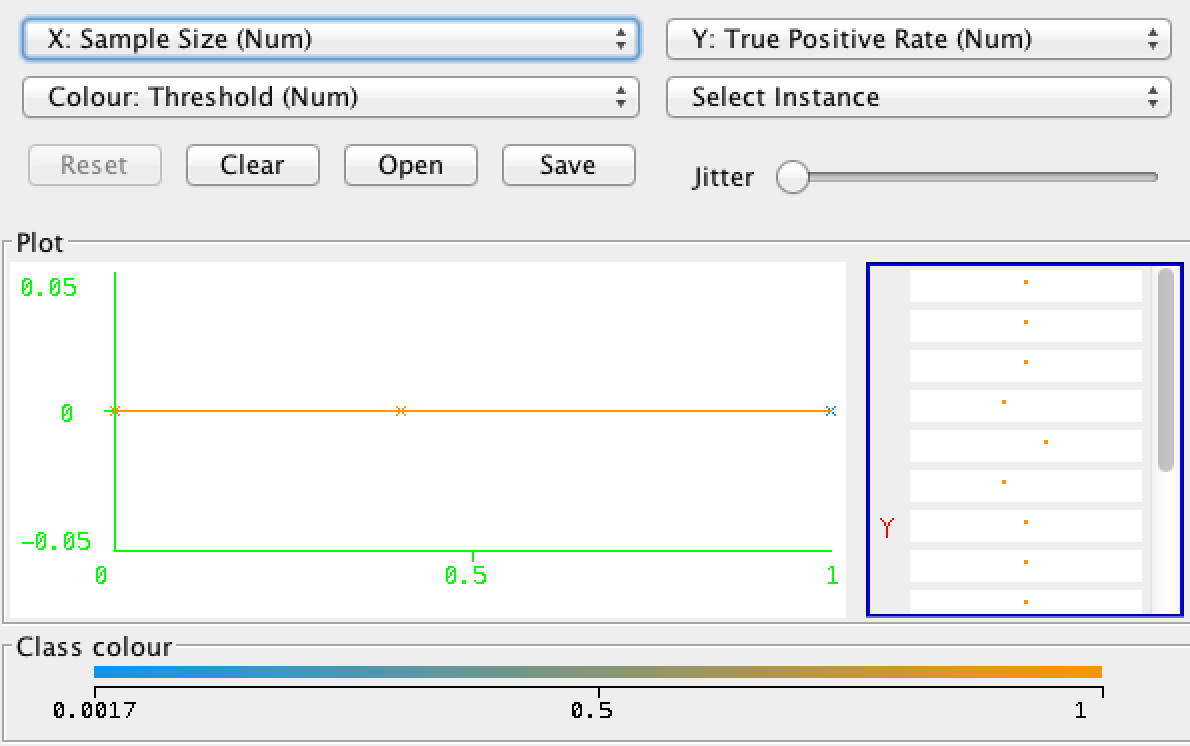
* Performance Graph of applying AdaBoostM1 with reduced-error pruning against Training and Test samples

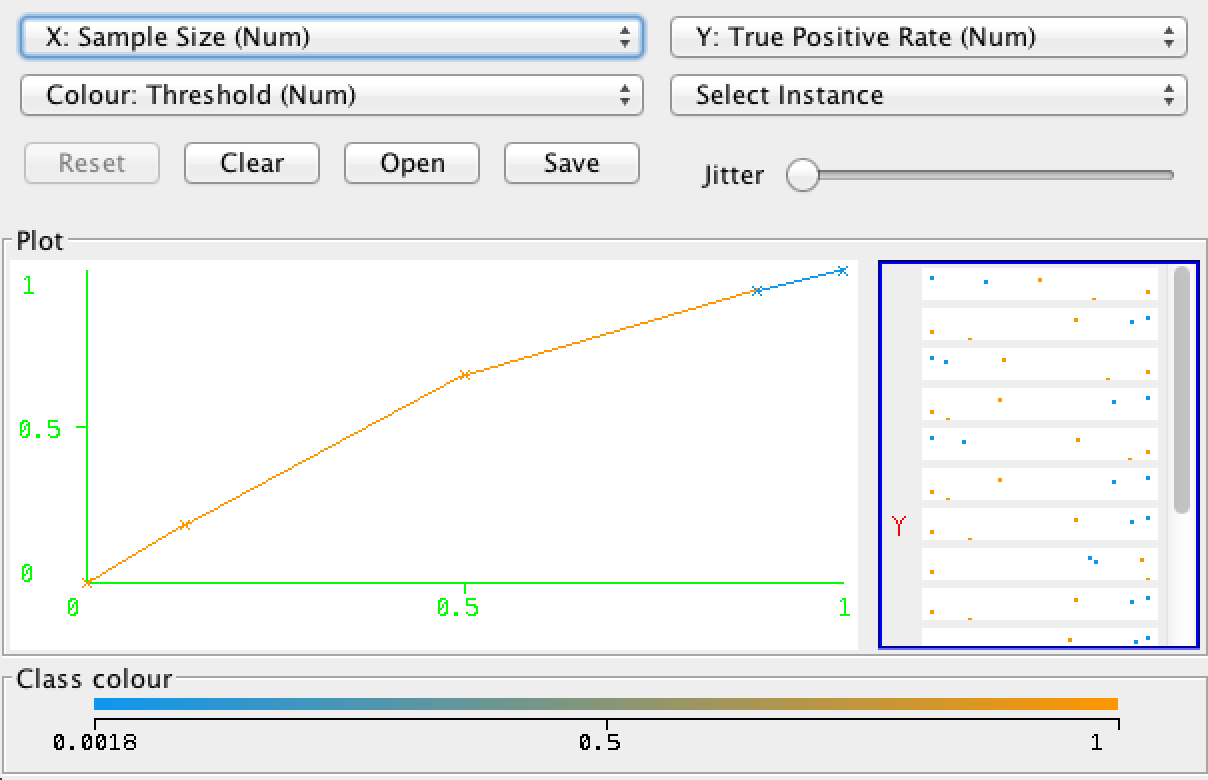
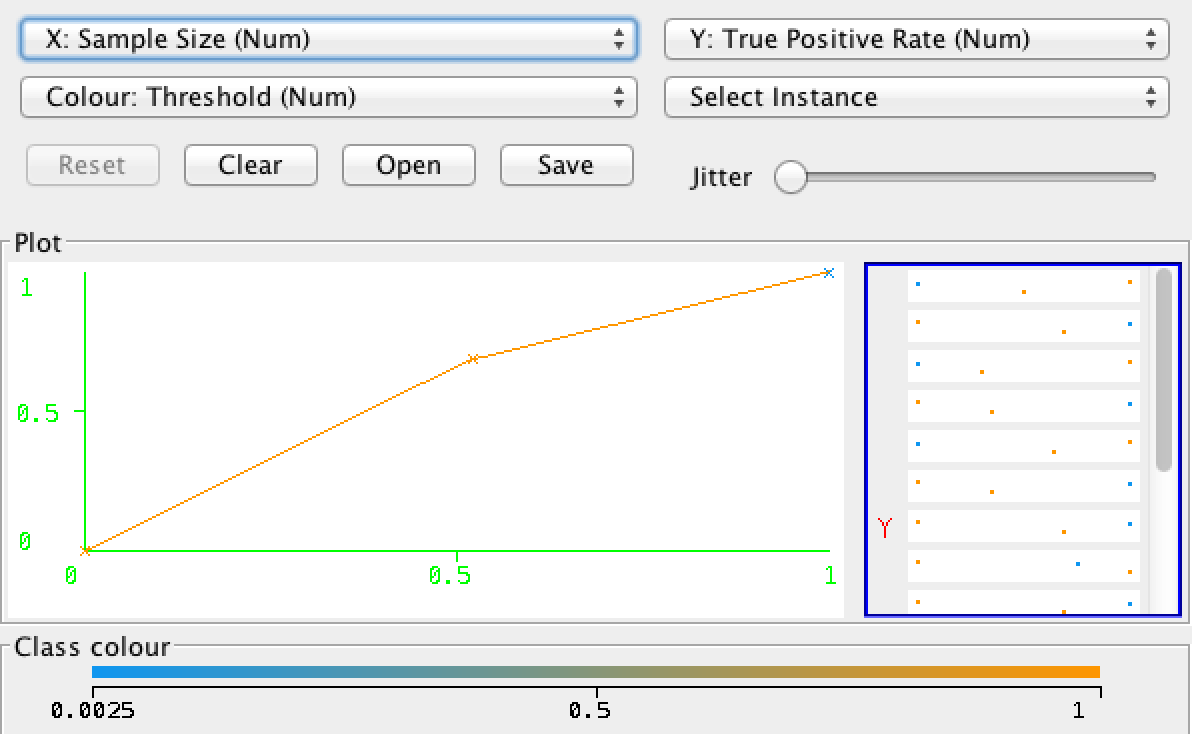
 

#### C. Run the KNN Algorithm

* So far reduced-error pruning methods (92%) still offers better performance than KNN algo (68%) (assuming test samples have been classified)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Performance of IBk algo with k=1 | Performance of IBk algo with k=1 against test samples | Performance of IBk algo with k=1 and 20 folds | Performance of IBk algo with k=1 and 80% split | Performance of IBk algo with k=3 | Performance of IBk algo with k=3 against test samples | Performance of IBk algo with k=3 and 80 folds | Performance of IBk algo with k=3 and 70% split |
| Correctly Classified Instances | 100 % | 60 % | 67.3043 % | 66.8605 % | 85.74% | 68 % | 69.74 % | 69.7674 % |
| Incorrectly Classified Instances | 0 % | 40% | 32.6957 % | 33.1395 % | 14.26% | 32% | 30.26 % | 30.2326 % |
| Mean absolute error | 0.0017% | 0.4003 | 0.3276 | 0.3322 | 0.227 | 0.3868 | 0.3544 | 0.3452 |
| Root mean squared error | 0.0017% | 0.6314 | 0.5708 | 0.5743 | 0.3275 | 0.5205 | 0.4674 | 0.4531 |

#### D. Run the SVM Algorithm

* accuracy rate is only 64% against test samples even with AdaBoostM1 (Normalized PolyKernel)
* so looks like SVM is not the suitable algo for this type of data.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Performance of SVM algo (with PolyKernel) against Training samples | Performance of SVM algo (with PolyKernel) against test samples | Performance of SVM algo with 70% split (with PolyKernel) | Performance of SVM algo with 10 folds (with PolyKernel) | Performance of SVM algo with Normalized PolyKernel | Performance of SVM algo with Normalized PolyKernel against test samples | Performance of SVM algo ( Normalized PolyKernel) with 100 folds | Performance of SVM algo with AdaBoostM1 (Normalized PolyKernel) – Test Data |
| Total size | 575 | 25 | **575** | **575** | 575 | 25 | **575** | **25** |
| Correctly Classified Instances | 85.565% | 52 % | **84.3023 %** | **81.0435 %** | 90.4348 % | 64% | **80.5217 %** | **64 %** |
| Incorrectly Classified Instances | 14.435% | 48% | 15.6977 % | 18.9565 % | 9.5652 % | 36% | 19.4783 % | 36 % |
| Mean absolute error | 0.1443 | 0.48 | 0.157 | 0.1896 | 0.0957 | 0.36 | 0.1948 | 0.3952 |
| Root mean squared error | 0.3799 | 0.6928 | 0.3962 | 0.4354 | 0.3093 | 0.6 | 0.4413 | 0.5802 |
| Relative absolute error | 28.9241% | 92.0133 % | 31.4161 % | 37.984 % | 19.1665 % | 69.01% | 39.03 % | 75.76 % |
| Time taken | 0.15 seconds | 0.08 seconds | 0.07 seconds | 0.08 seconds | 0.17 seconds | 0.16 seconds | 0.16 seconds | 4.56 seconds |

* I have omitted the performance graphs for the sake of brevity !
* They can be always generated from Weka ~ ‘sample size(x)’ vs ‘true positive rate(y)’ .

#### E. Run the MultiLayerPerceptron (ANN) algo

* Even though MLP offers great accuracy rate over training samples (99.8%) but correctly classifies only 60% test instances.
* Performance improves a bit (64% accuracy rate against test data) at a slower learning rate (0.2)
* **It takes longer time ( 5.4 s) to build the model.**
* Even with a stopping condition like validationThreshhold , trainingTime, validationSetSize accuracy rate does not improve.
* Interestingly if we decrease momentum , lower the learning rate and increase hidden layers then performance improves. For example :
  + If learning rate = 0.12 , momentum = 0.01 , #hidden layers = 6, then accuracy is 72%
  + Also it builds the model in 0.19 second as the number of hidden layers have increased!
* Overall, MLP is not able to converge to a solution for this particular training set due to
  + either unbiased estimators but probably with high variance (due to over-fitting) ;
  + or a rigid model in contrast , leading to small variance but high bias.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Performance of MLP algo (learning rate=0.3) | Performance of MLP algo against test samples (learning rate=0.3) | Performance of MLP algo with 10 folds (learning rate=0.3) | Performance of MLP algo with 66% split (learning rate=0.3) | Performance of MLP algo (learning rate=0.2) | Performance of MLP algo against test samples (learning rate=0.2) | Performance of MLP algo with 10 folds (learning rate=0.2) | Performance of MLP algo with 66% split (learning rate=0.2) |
| Total size | 575 | 25 | **575** | **575** | 575 | 25 | **575** | **575** |
| Correctly Classified Instances | 99.8261 % | 60% | **78.6087 %** | **77.9487 %** | 99.4783% | 64% | **77.3913%** | **80 %** |
| Incorrectly Classified Instances | 0.1739 % | 40% | 21.3913 % | 22.0513 % | 0.5217% | 36% | 22.6087% | 20% |
| Mean absolute error | 0.0059 | 0.4158 | 0.2257 | 0.2265 | 0.0105 | 0.3504 | 0.2297 | 0.2171 |
| Root mean squared error | 0.0423 | 0.5805 | 0.4308 | 0.4315 | 0.0727 | 0.5309 | 0.4326 | 0.4136 |
| Relative absolute error | 1.1723 % | 79.7042 % | 45.2297 % | 45.3454 % | 2.0957 % | 67.1713 % | 46.028 % | 43.4568 % |
| Time taken | 5.51 s | 5.47 s | 5.49 s | 5.49 s | 5.47 seconds | 5.49 seconds | 5.45 seconds | 5.49 seconds |

#### Conclusion

* J48 with reduced-error pruning is the best algorithm to classify the CreditRating data.
* It effectively removes noise and over-fitting.

## Description of second classification problem

The Goal is to find the best classification algorithm by analyzing the given training data.

The dataset contains information on free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. The data was captured using a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. Each instance contains 34 real values.

Training Data : ionosphere\_train.arff, Test data : ionosphere\_test1.arff , ionosphere\_test2.arff

Citation : <https://archive.ics.uci.edu/ml/datasets/Ionosphere>

### Classification using Command Line Interface – ml program suite

Refer to the previous classification problem , to find the usage of this CLI tool.

> algo-analysis-cli/

> java -jar mlprograms-1.0.0.jar ionosphere\_train.arff ionosphere\_test1.arff

### Classification using the weka tool :

#### A. Run the J48 Decision Tree Classifier Algorithm:

We want a model (decision tree) that generalizes beyond the instances that have been provided at "training time" to new unseen examples.

* Looks like there is not much noise in the data as even unpruned tree classifies test samples with an accuracy 88.89%
* Applying reduced-error pruning on J48 with certain combination improves the performance and lowers memory footprint.
  + After experimenting with numFolds , we find that numFolds=5 offers a higher accuracy of 91.67% (it alters between 88.9 and 91.44 with different folds ) for test data ionosphere\_test1.arff with a much reduced tree size ( 5 leaves, size 9) .
  + Similarly it provides accuracy 91.43 for the second test samples (ionosphere\_test1.arff) when we change numFolds to 5.

\*\* One fold is used for pruning, the rest for growing the tree.

* If we do not use ‘reducedErrorPruning’ and increase cross-validation folds, overall accuracy against test dataset is low : correctly Classified Instances ~ 72.33 % and also size of tree is larger : number of leaves = 17, size of the tree = 31.
  + Actually, unless the signal:noise ratio is high or dataset is really huge , Cross-validation does not yield precise result.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Unpruned Tree performance against Training Samples | Unpruned Tree performance against Test samples | Performance with simple pruning against Training Samples | Performance of simple pruning against Test samples | Performance of reduced-error pruning against Training Samples | Performance of reduced-error pruning ( numFolds=3) against Test samples | Performance of reduced-error pruning ( numFolds=5) against Test samples |
| Number of Leaves | 15 | 15 | 15 | 15 | 8 | 8 | 5 |
| Size of the tree | 29 | 29 | 29 | 29 | 15 | 15 | 9 |
| Correctly Classified Instances | 99.048 % | 88.89 % | 99.048 % | 88.89 % | 93.33 % | 88.89 % | 91.43% |
| Incorrectly Classified Instances | 0.0184 | 11.11% | 0.0184 | 11.11% | 6.67% | 11.11% | 11.11% |
| Mean absolute error | 0.1155 | 0.1155 | 0.1155 | 0.1155 | 0.0877 | 0.0938 | 0.0938 |
| Root mean squared error | 0.0959 | 0.3273 | 0.0959 | 0.3273 | 0.2266 | 0.2414 | 0.2414 |
| Relative absolute error | 3.9933 % | 25.061% | 3.9933 % | 25.061% | 19.043 % | 20.337 % | 20.337 % |
| Time taken | 0.01 seconds | 0.01 seconds | 0.01 seconds | 0.01 seconds | 0.01 seconds | 0.01 seconds | 0.01 seconds |

#### B. Performance improvement through Ensemble learning (by applying boosting algorithms)

* When we apply the AdaBoostM1 ( where new models are influenced by performance of previously built ones) on J48 tree (pruned using reduced-error pruning method) against ionosphere\_test1.arff , we find that 97.22 % instances are correctly classified. Similarly 94.29% data are correctly classified for second test set.
* But if we used simple pruning (NOT reduced error pruning) then accuracy is 100% for both the test sets.
* That means , boosting trees created using reduced-error pruning slightly over-prunes the tree !
* We observe similar behavior when we apply MultiBoostAB on pruned trees against same test data.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Performance of AdaBoostM1 on pruned tree (reduced-error pruning) against training data | Performance of AdaBoostM1 on pruned tree against ionosphere\_test1.arff | Performance of AdaBoostM1 on pruned tree against ionosphere\_test2.arff | Performance of MultiBoostAB on pruned tree ((reduced-error pruning) against training samples. | Performance of MultiBoostAB on pruned tree on pruned tree against ionosphere\_test1.arff | Performance of MultiBoostAB on pruned tree against ionosphere\_test2.arff |
| Number of Leaves | 8 | 8 | 8 | 8 | 8 | 8 |
| Size of the tree | 15 | 15 | 15 | 15 | 15 | 15 |
| Correctly Classified Instances | 97.14% | 97.22% | 94.29% | 97.143% | 97.22% | 97.143% |
| Incorrectly Classified Instances | 2.86% | 2.78% | 5.71% | 2.857% | 2.78% | 2.86% |
| Mean absolute error | 0.0577 | 0.0613 | 0.0896 | 0.0334 | 0.0333 | 0.0333 |
| Root mean squared error | 0.1697 | 0.1806 | 0.2284 | 0.1726 | 0.1696 | 0.1708 |
| Relative absolute error | 12.528 % | 13.3054 % | 19.3093 % | 7.2617 % | 7.2228 % | 7.1782 % |
| Time taken | 0.03 s | 0.03 s | 0.03 s | 0. 07 s | 0.07 s | 0.07 s |

#### C. Run the KNN Algorithm

* When 1-NN classifier is tested on training data, accuracy will be 100% since the "nearest neighbor" will be the point itself (except ties).
* The test result is not very consistent.
  + With cross-validation and k=3, accuracy is 83.3% for test dataset1 and 100% for test dataset2.
  + With k=3 and without cross-validation, accuracy for test dataset 1 ~ 94.28% and for test dataset2 ~ 80.56%
  + Such inconsistencies may arise due to unbalanced data distribution problem which is not handled properly by simple KNN.
    - Ref : <http://inside.mines.edu/~huawang/Papers/Conference/2010aaai_mlknn.pdf>
* So far the best performing algo is Performance of AdaBoostM1 on pruned tree with accuracy as 100% and 97.22 respectively.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Performance of IBk algo with k=1 against training samples | Performance of IBk algo with k=1 against test samples (ionosphere\_test2.arff) | Performance of IBk algo with k=1 against test samples (ionosphere\_test1.arff) | Performance of IBk algo with k=3 against training samples | Performance of IBk algo with k=3 against test samples (ionosphere\_test2.arff) | Performance of IBk algo with k=3 against test samples (ionosphere\_test1.arff) | Performance of IBk algo with k=3 , crossValidate=true against training samples | Performance of IBk algo with k=3 , crossValidate=true against test samples (ionosphere\_test1.arff) | Performance of IBk algo with k=3 , crossValidate=true against test samples (ionosphere\_test2.arff) |
| Correctly Classified Instances | 100 % | 100% | **83.33%** | **91.4286 %** | **94.28%** | **80.56%** | **98.73%** | **83.33%** | **100%** |
| Incorrectly Classified Instances | 0 % | 0% | 16.67% | 8.5714 % | 5.72% | 19.44% | 1.27% | 16.67% | 0% |
| Mean absolute error | 0.0031 | 0.0032 | 0.1686 | 0.0895 | 0.0676 | 0.1951 | 0.0617 | 0.1816 | 0.0301 |
| Root mean squared error | 0.0031 | 0.0032 | 0.407 | 0.2202 | 0.1869 | 0.3806 | 0.1737 | 0.3814 | 0.1195 |
| Time taken to build the model | 0 s | 0 s | 0 s | 0 s | 0 s | 0 s | 0 s | 0 s | 0 s |

#### D. Run the SVM Algorithm with PolyKernel and Normalized PolyKernel

* SMO algo is very much resilient to over-fitting .
* Looks like Normalized PolyKernel performs better on test datasets than Polykernel as it normalizes some sparseness and feature vectors and finds a better model.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Performance of SVM algo with PolyKernel | Performance of SVM algo with PolyKernel against Test Samples | Performance of AdaBoostingM1 on SVM with PolyKernel | Performance of AdaBoostingM1 on SVM with PolyKernel against Test Samples | Performance of SVM algo with Normalized PolyKernel | Performance of SVM algo with Normalized PolyKernel against Test Samples | Performance of AdaBoostingM1 on SVM with Normalized PolyKernel | Performance of AdaBoostingM1 on SVM with Normalized PolyKernel against Test Samples |
| Correctly Classified Instances | 92.06% | 97.14% | 93.33% | 94.29% | 96.19% | 97.15 | 94.286 % | 97.14% |
| Incorrectly Classified Instances | 7.9% | 2.85% | 6.67% | 5.71% | 3.81% | 2.85 | 5.7143 % | 2.86% |
| Mean absolute error | 0.0794 | 0.169 | 0.0753 | 0.0468 | 0.0381 | 0.0286 | 0.091 | 0.0791 |
| Root mean squared error | 17.2406 % | 6.1589 % | 0.2087 | 0.1318 | 0.1952 | 0.169 | 0.1955 | 0.1536 |
| Relative absolute error | 58.737% | 34.9721 % | 16.3527 % | 10.0803 % | 8.2755 % | 6.1589 % | 19.7598 % | 17.0462 % |
| time | 0.02 s | 0.10 s | 0.11 s | 0.12 s | 0.03 s | 0.03 s | 0.98 s | 0.98 s |

#### E. Run the MLP Algorithm

* MLP provides best result for both the test data sets. MLP is particularly useful when there has large number of features.
* Usually MLP offers best performance as it uses a non-linear sigmoid transfer function with backpropagation. MLP iteratively minimizes the error using steepest descent.
* MLP is slower to learn nonlinear functions with complex local structures due to global nature of functional approximations. During gradient descent, the plateaus in which the parameter is trapped in the process of learning, take long time to get rid of them.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Performance of MLP algo against Training data | Performance of MLP algo against Test Sample Set1 | Performance of MLP algo against Test Sample Set2 |
| Correctly Classified Instances | 99.68% | 100 % | 100 % |
| Incorrectly Classified Instances | 0.32% | 0% | 0% |
| Mean absolute error | 0.0105 | 0.0092 | 0.0064 |
| Root mean squared error | 0.0551 | 0.0182 | 0.0116 |
| Relative absolute error | 2.283% | 1.9799 % | 1.3729 % |
| Time taken | 1.45 s | 1.47 s | 1.44 s |

#### Conclusion

* We have applied the Supervised Learning algo against the same datasets.
* MLP provides 100% accuracy but takes 1.45 s (avg), whereas SVM (with Normalized PolyKernel) offers 97.14% accuracy taking only 0.10 s to build the model.
* AdaBoostM1 on pruned tree against ionosphere\_test1.arff correctly classifies 97.22% instances in 0.03 s.
* KNN Algo with 3 neighbors offer accuracy of 91.43% and 94.28% against test samples.
* If we are okay with the time taken to build models by MLP (~1.45 s), then MLP stands as the winner.