Implementation of Breiman's Random Forest Machine Learning Algorithm

Frederick Livingston

Abstract

This research provides tools for exploring Breiman's Random Forest algorithm. This paper will focus on the development, the verification, and the significance of variable importance.

Introduction

A classical machine learner is developed by collecting samples of data to represent the entire population. This data set is usually subdivided into two or more dataset. Part of the dataset set is commonly use for developing the machine learner, and the remaining data is use for evaluation. Often this data set is imbalanced; the data consists of only a very small minority of the data. Imbalanced machine learners tend to perform poorly with the classification of fraud detection, network intrusion, rare disease diagnosing, etc [1, 2]. This is due to imbalanced sampling during developing the machine learner. During the testing phase these rare cases are unseen during the training phase and are usually misclassified. Leo Breiman, a statistician from University of California at Berkeley, developed a machine learning algorithm to improve classification of diverse data using random sampling and attributes selection. This project involved the implementation of Breiman's random forest algorithm into Weka. Weka is a data mining software in development by The University of Waikato. Many features of the random forest algorithm have yet to be implemented into this software.

Background

The random forest machine learner, is a meta-learner; meaning consisting of many individual learners (trees). The random forest uses multiple random trees classifications to votes on an overall classification for the given set of inputs. In general in each individual machine learner vote is given equal weight. In Breiman's later work, this algorithm was modified to perform both un-weighted and weighted voting. The forest chooses the individual classification that contains the most votes. **Figure 1**. below is a visual representation of the un-weighted random forest algorithm.

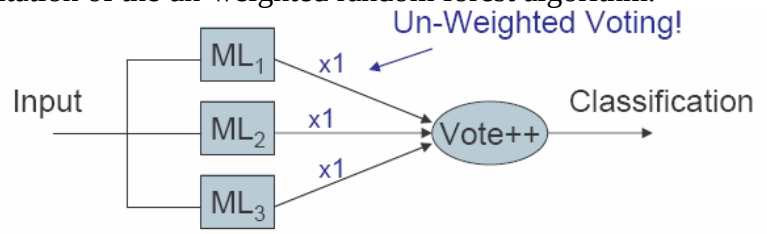

Figure 1. Meta Learners [3]

Individual random tree machine learners are grown in the following manner:

1. A data set [inbag] is formed by sampling with replacement members from the training set; this technique is often referred to as "bootstrapping". The number of examples in the [inbag] data set is equal to that of the training data set. This new data set may contain duplicate examples from the training set. Using the bootstrapping technique, usually one third of the training set data is not present in the [inbag]. This left over data is known as the out-of-bag data [oob].

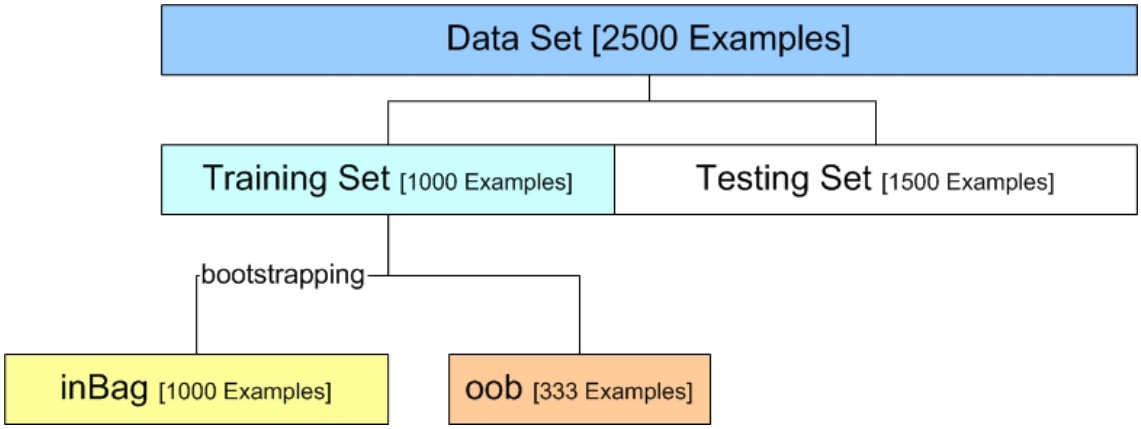

Figure 2. Sample with Replacing

2. A random number of attributes are chosen for each tree. These attributes form the nodes and leafs using standard tree building algorithms.

3. Each tree is grown to the fullest extent possible without pruning.

This process is repeated to develop multiple individual random trees learners. After the development of the tree, the out-of-bag examples are used to test the individual's trees as well as the entire forest. The average misclassification over all trees is known as the out-of-bag error estimate. This error estimate is useful for predicting the performance of the machine learner with out involving the test set example. This information could be found useful in determining the weights of the individual trees classification in the weighted random forest learner.

Variable Importance

An important feature of Breiman's algorithm is the variable importance calculation. This algorithm analyzes each attribute and reveals the importance of the attribute in predicting the correct classification of the random forest machine learner. The user then could filter out unnecessary attributes which would save time during data collecting and experimental run time. This algorithm first computes untouched correct count, the number of correct classification using the out-of-bag data as its test set. The values of the attributes are then randomly permuted in the out-of-bag examples. This new data set is then tested for correct classification. The average of this number over all trees in the forest is the raw importance score for the particular attribute. This algorithm is explained in more details in the implementation section.

$$raw\_importance_{variable[m]} = \frac{untouched\_count - variable_m count}{number\_of\_trees}$$

Eqn. 1: raw importance calculation

A low raw importance score, a score near zero, indicates a poor relationship between given attribute and correct classification. A positive importance score indicates that the given variable is important for correct classification

Implementation of Variable Importance into Weka

The implementation of the variable importance required the modification of *weka.classifiers.meta.Bagging.java* and *weka.classifiers.trees.RandomForest.java*. To leave the original code intact, two new classes where created called *BaggingExt* and *RandomForestExt*. Individual trees are created with the call to the *weka.classifiers.meta.Bagging.buildClassifier (Instances data)* method, where *Instances data* is the training dataset. This method builds the individual trees and stores them in the object *m_Classifiers[j]*, where *j* is the index of the tree. This method also stores the out-of-bag examples in the array of instances *oobData[j]*. Each individual tree has its own set of out-of-bag examples which can be obtain using by the index *j*.

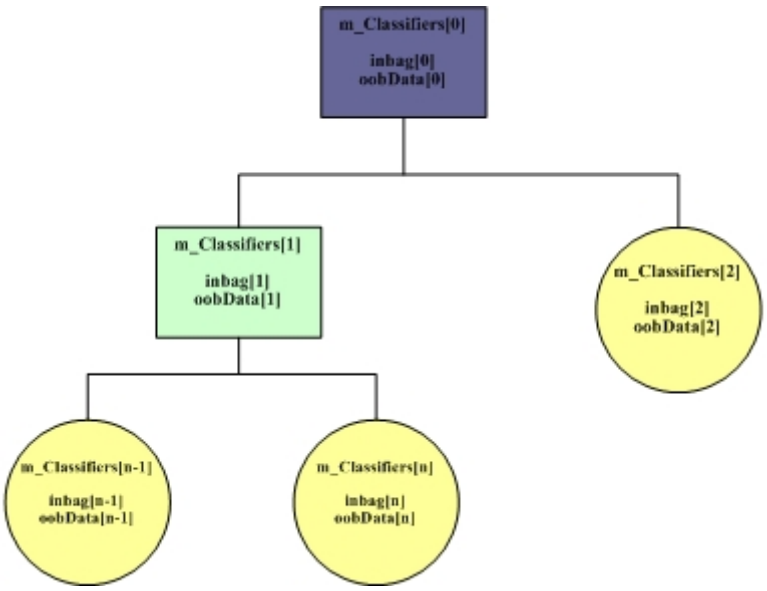

Figure 3: Tree Objects

Storage of the out-of-bag examples is an important step for computing variable importance. Variable importance is performed in *weka.classifiers.meta.Bagging.calcImportance()*. The first step in calculation the variable importance is to obtain a correct classification count of the entire forest using out-bag-examples as the test sets. This value is stored in the variable *correctSum*. After the computation of the *correctSum*, each attribute in the dataset is created with a permuted value for the given attribute. For example: The classical Weather.arff dataset has four input attributes (outlook, temperature, humidity, and windy) and one output attribute play. Let pretend a node in the forest (m_Classifier[j]) has the out-of-bag dataset in Table1 column 1. Permuting the attribute, outlook, would have the result

located in column 2 of Table1. This attribute is permuted throughout all nodes in the entire forest.

Table 1.

Original Out-of-bag Instance	Modified Out-of-Bag Instance
@data sunny,hot,high,FALSE,no rainy,mild,high,FALSE,yes rainy,cool,normal,TRUE,no sunny,cool,normal,FALSE,yes sunny,mild,normal,TRUE,yes	@data overcast,hot,high,FALSE,no sunny,mild,high,FALSE,yes sunny,cool,normal,TRUE,no rainy,cool,normal,FALSE,yes sunny,mild,normal,TRUE,yes

If the attribute is nominal, the value is randomly picked from the defined selection of values in the arff file. If the attribute is numeric or real the value is selected using the output from the random number generator. This newly modified example is then executed down the tree for classification. The modified out-of-bag correct classification count is stored in the variable *rawScore*. If the variable is important the correct classification count should vary from the original count, *correctSum*. This procedure is repeated for every attribute in the training set.

Using Variable Importance in Weka

The newly added features of the Random Forest can be access by selecting the ***RandomForestExt*** Classifier from the Weka Explorer. The results from the variable importance calculation are appended to the classifier output.

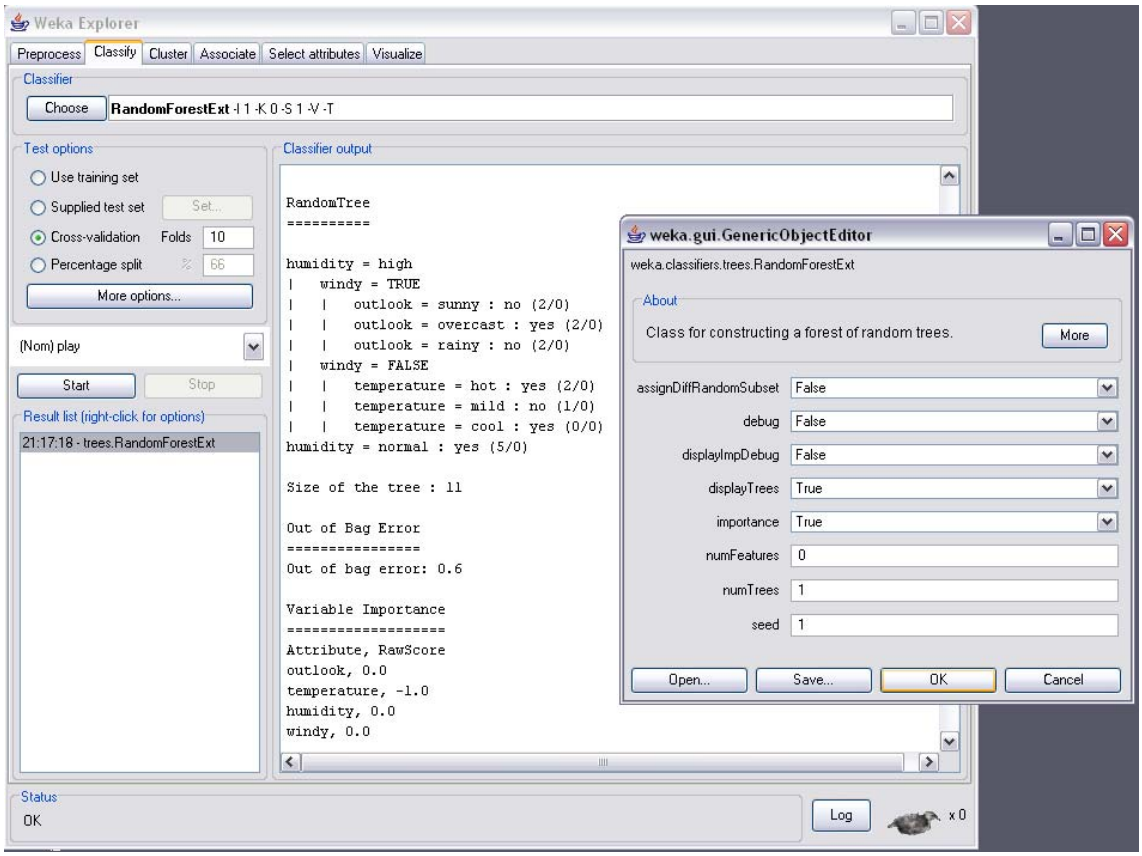

Figure 3. Weka Screenshot

Figure 3. display the screenshot for the extend version of the random forest algorithm. The extended version is executed similar to the original version with a few modifications in the parameters. The newly added parameters consist of displayImpDebug, displayTrees, and importance.

- displayImpDebug, Controls whether or not to display variable importance debugging information. Setting this value True will display information similar to the items in Table1. (Note: Increase required memory)

- displayTrees, Controls whether or not to display the individual trees

- importance, Controls whether or not to calculate and display variable important

Verification of Variable Importance

Validation of the variable importance implementation was performed by comparing the results of the Weka's output to the results from Breiman's random forest paper of 2001. Breiman's paper included two example calculations of variable importance. In the first example, variable importance was computed using the Diabetes data test, with 1000 trees. **Figure 4.** displays the output from Breiman's paper and **Figure 5.** displays the output from Weka. Comparing the two outputs there is similar behavior in the variable importance. In both figures the second attribute is significantly more important to the correct classification.

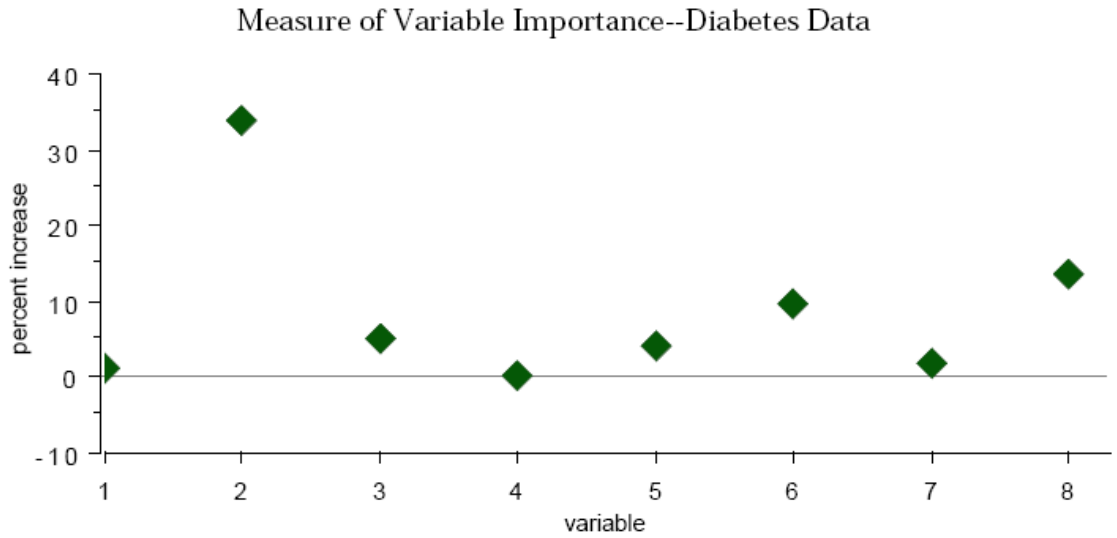

Figure 4. Breiman's Diabetes Output [1]

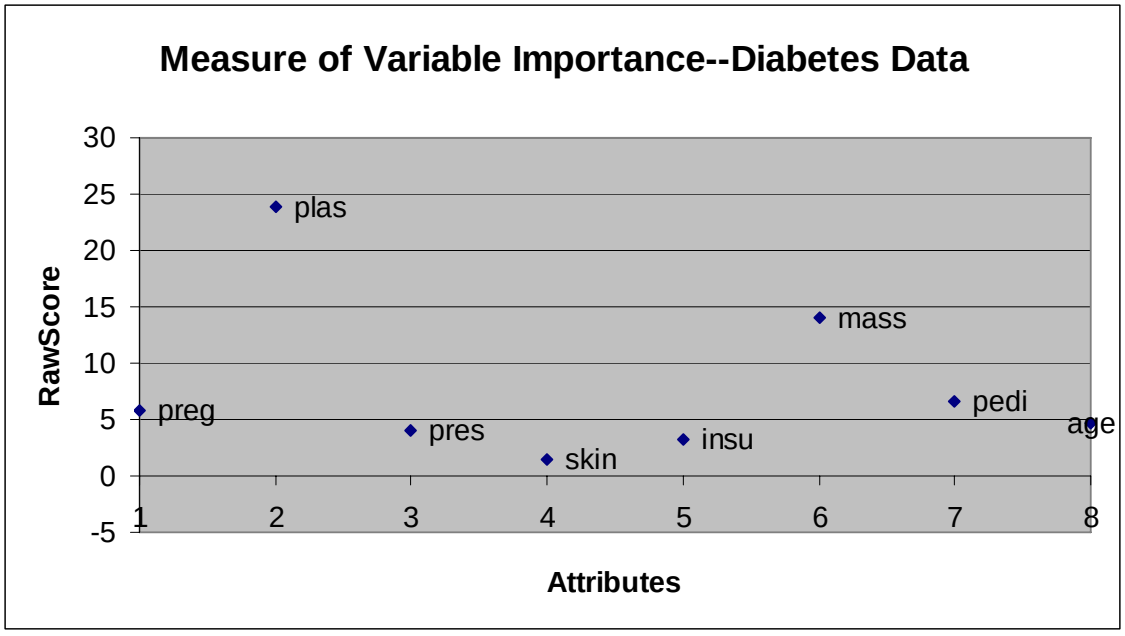

Figure 5. Weka's Diabetes Output

The experiments where repeated using the Votes dataset. **Figure 6.** shows Breiman's output and **Figure 7.** shows Weka's outputs for this experiment. Again the variables importance in both figure have the same trends.

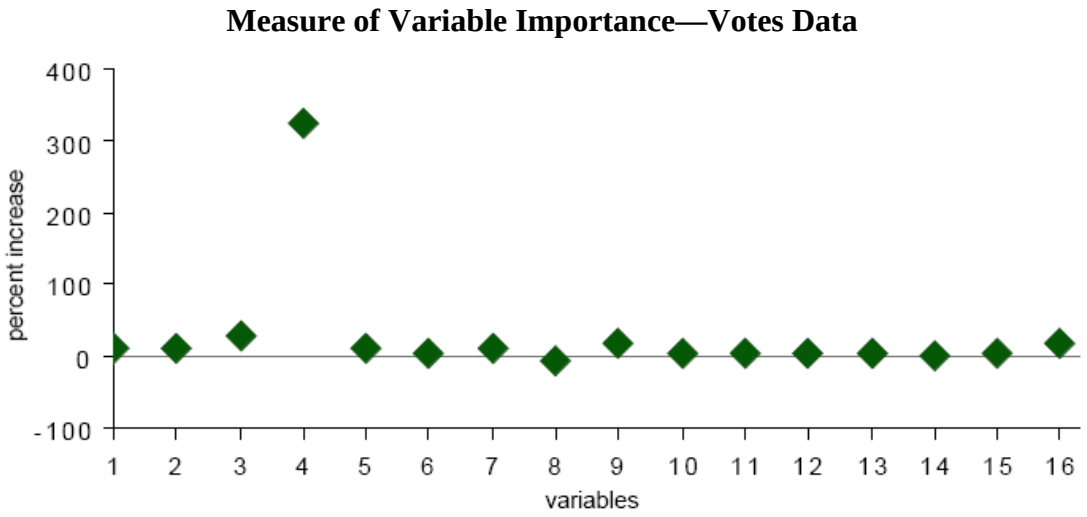

Figure 6. Breiman's Votes output [1]

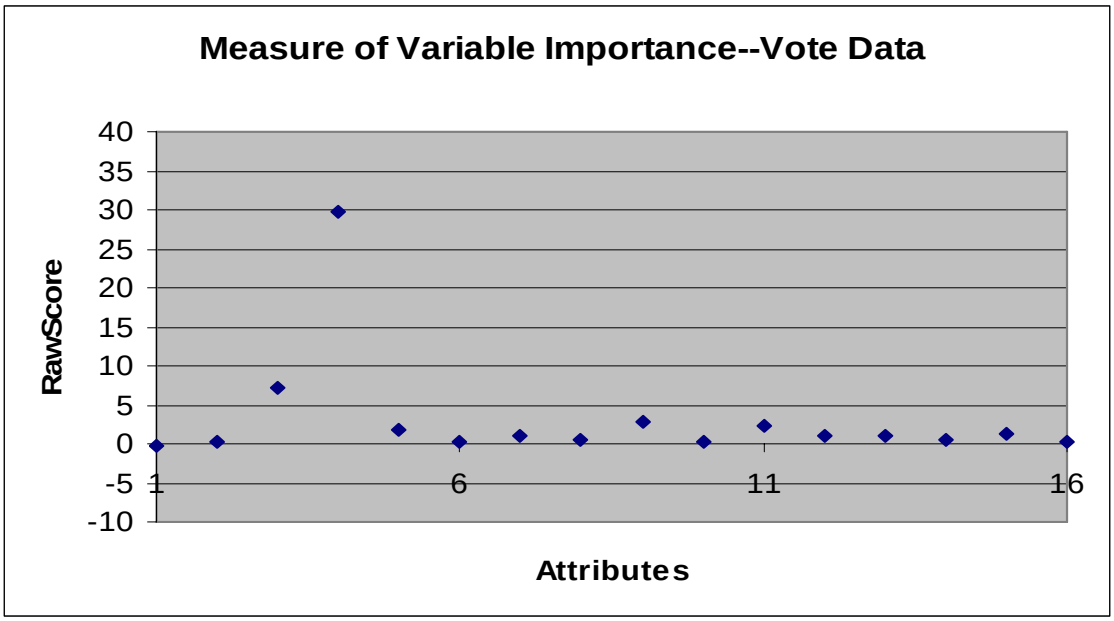

Figure 7. Weka's Votes output

Although the Weka and Breiman's results are very similar; the validation of the software at this point is somewhat weak. One potential problem is that the graphs are comparing variable importance rawscore (see **Eqn 1.**) to percent increase. Although there may be some kind a relationship between the two; these comparisons are not ideal. Another potential problem is lack of verification examples.

Random Forest Software Improvements

To properly verify the new Weka's implementation the classifier outputs should be compared to the output of Breiman's code. The complete random forest algorithm is implemented in open source Fortran 77 code. To execute his code one must hard code the classifier parameters such as number of trees, data sets names, and etc. Then the user must compile the code using an obsolete compiler, and format the data sets in a manner such that it can be read by the program. Another disadvantage to his code is that Fortran 77 performs static memory allocation; meaning memory for the program is reserved at the beginning of execution. This has an advantage for speed but could cause problems with large data sets on computers with limit random-access-memory. Fortran 77 also lacks features such as command line arguments and pointers. To make Breiman's work more user-friendly modifications/add-on were investigated.

To convert data set from and to Weka's format (arff extension) to Breiman's format a java application was developed call arffConverter. This program can be executed manually with the following command.

java –cp RandomForest.jar arffConverter *[input file] [output file] [mode]*

[input file], the absolute or relative path to the input file
[output file], the absolute or relative path to the where the new file will be stored
[mode], an integer that is either a one or zero.

If the mode is zero, the *[input file]* is converted from the Breiman's format to Weka's file format. If the mode is one, the *[input file]* is converted from the Weka's format to Breiman's file format.

Breiman's file format contains only numerical data. Strings are given integers labels beginning with the index of 1 and incremented once for every string instances and the commas are replaced by spaces. **Table 2.** displays an example of this conversion.

Notes: Breiman's code does not except attributes values of zero. A solution to this is to fudge all values of zero by some small number (ex. 0 become 0.001). This fudging is not implemented in this software and must be done manually. Most dataset do not need this adjustment.

Weka's arff Format	Breiman's Format
sunny,85,85,FALSE,no	1 85 85 1 1
sunny,80,90,TRUE,no	1 80 90 2 1
overcast,83,86,FALSE,yes	2 83 86 1 2
rainy,70,96,FALSE,yes	3 70 96 1 2
rainy,68,80,FALSE,yes	3 68 80 1 2

Table 2. Sample Conversion

The next improvement involves modification of the Fortran 77 code. Instead of the user hard coding parameters and then compiling the program, it would be ideal to past the parameters into the program. Fortran 77 has very little capability of performing this task; since all arrays sizes are static and are determine during when compiling. One solution is to initialize the arrays to a very large size. Although this is a valid solution it is very memory intensive. The best solution is use dynamic arrays where the size of the array can be set during runtime. This feature was made available starting with Fortran 90. Fortran 90 is almost backward compatible with Fortran 77 except for a few small syntax changes. These formatting changes, as well as the modification from static to dynamic arrays, where made to Breiman's code and compiled as random_foreset_win32.exe and random_forest_linux.bin. This executable reads in the experiment parameters from params.dat. **Table 3.** displays the configuration file (params.dat) for the Weather dataset. More details about the configuration can be found in Breiman's Manual.

4 14 2 1 0 0 1	Line 1: Describe data
4 1 10 10 1 0 0 0 4351	Line 2: Set run parameters
0 0 0 0	Line 3: Set importance options
0 5	Line 4: Set proximity computations
0 0 0	Line 5: Set options based on proximities
-999 0 0	Line 6: Replace missing values
0	Line 7: Visualization
0 0 0 0	Line 8: Saving a forest
0 0 0 0	Line 9: Running new data down a saved forest
dataset.train dataset.test	Line 10: datasets files

Table 3. Configuration File

To execute this program, create a text file named params.dat containing the proper configuration data. Create dataset by hand or using the arffConverter. Then run the appropriate random_forest executable at the command prompt. The output is written the standard output. The program also created several files that save different parameters. The program will generate a runtime error if the files already exist. **These files must be deleted before re-execution.**

The RandomForest java application was developed to automate all of the above process and provide a GUI to the random forest algorithm. **Figure 8.** provides a screenshot of the application.

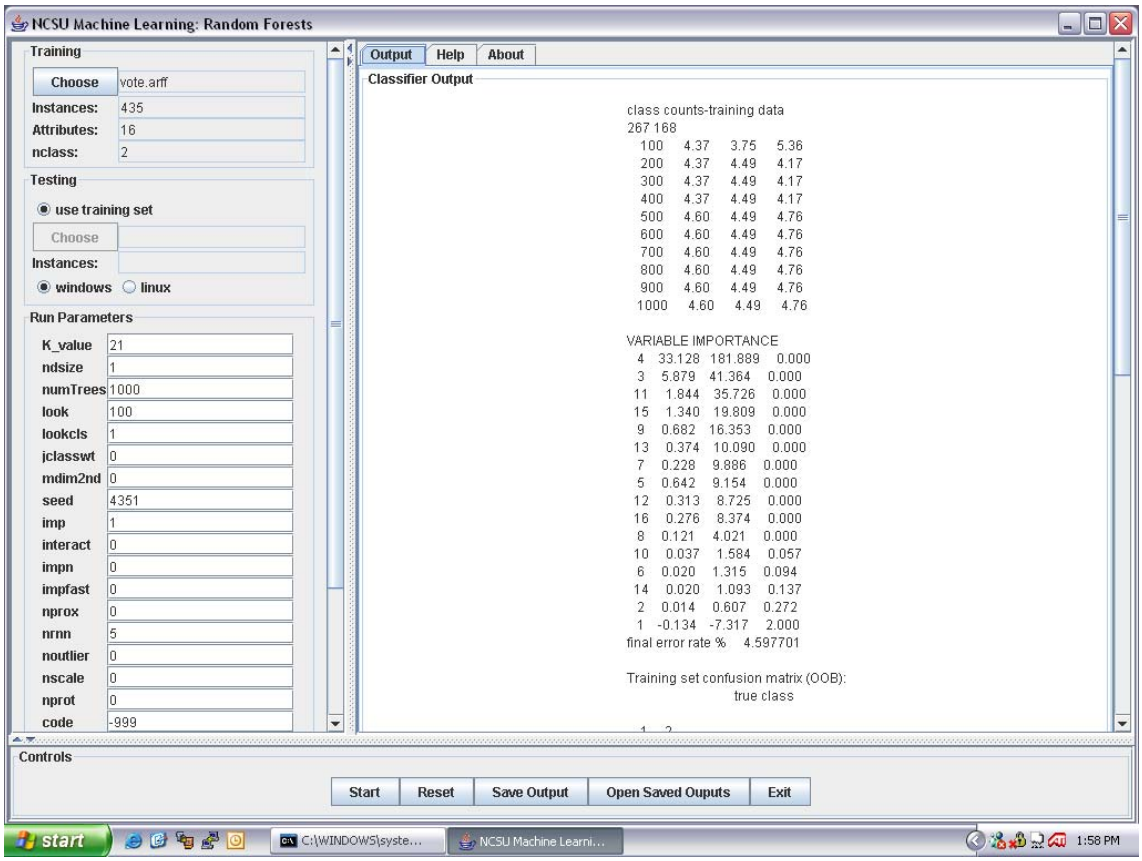

Figure 8. Random Forest GUI

The following command is used to start the application.

java -Xmx256m -jar RandomForest.jar

Small enchantments where made to RAFT, a random forest visualization tool, for easier operation. In order to use the RAFT software the following parameters must be set in the random forest application:

impn=1, imp=1, nscale=3 and nprox=1

To start RAFT use the following command:

java -mx300m -cp visad.jar;ij.jar;raft.jar;. Raft –norescale

Improved Verification

These new tools where then use to repeat the verification experiments. **Table 4.** compares the variable importance from Weka and Breiman. Although the raw score varies, the prediction of the two most important attributes is the same.

Attributes	Weka RawScore	Breiman RawScore
preg	5.877	2.103
plas	23.835	8.122
pres	4.037	0.286
skin	1.497	0.367
insu	3.182	0.457
mass	14.067	3.617
pedi	6.641	0.859
age	4.675	2.674
oob error%	30.69	23.697918

Table 4. Verification Diabetes Dataset

This experiment was repeated for the Votes dataset. The result from this experiment verifies the correct implementation of the Weka's code. In **Table 5.** Weka's output and Breiman's output is very similar.

Attributes	Weka's RawScore	Breiman's RawScore
handicapped-infants	-0.19	-0.134
water-project-cost-sharing	0.135	0.014
adoption-of-the-budget-resolution	7.281	5.879
physician-fee-freeze	29.706	33.128
el-salvador-aid	1.921	0.642
religious-groups-in-schools	0.146	0.02
anti-satellite-test-ban	1.011	0.228
aid-to-nicaraguan-contras	0.486	0.121
mx-missile	2.871	0.682
immigration	0.136	0.037
synfuels-corporation-cutback	2.232	1.844
education-spending	1.126	0.313
superfund-right-to-sue	0.902	0.374
crime	0.611	0.02
duty-free-exports	1.407	1.34
export-administration-act-south-africa	0.137	0.276
oob error%	6.52	4.5977

Table 5. Verification Votes Dataset

Understanding the Importance Variable Importance

This section examines the variable importance results from the previous section in more detail. **Figure 9.** displays a visual representation of the variable importance of the

Diabetes dataset. The second and the six attributes have higher importance than the other attributes.

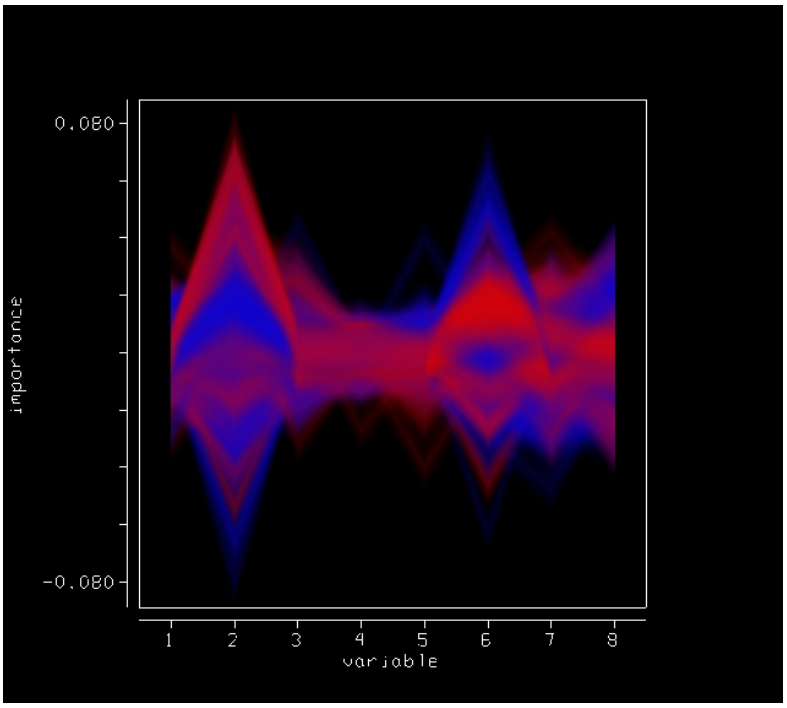

Figure 9. RAFT generated results from Diabetes dataset

Using 1000 trees, the out-of-bag error is 24.61%. I wanted to see how the two most important variables affect the out-of-bag error. The experiment was re-run using only the two most important variables, plas and mass. The incorrectly classification increased slightly to 27.99%. The experiment was re-run using all other attributes except for plas and mass. In this experiment I was expecting a large increase in incorrect classification. Although the error rates increase it was much smaller than expected. These experiments where repeat using fewer trees to see if the trends remain constants. The results from these experiments are in **Table 6.** This information could be useful in developing future learners for this particular dataset. One could gather and train the learners using only the plas and mass attributes and receive similar results with all the attributes. This could reduce computation time since the dataset size is reduced.

Trial	Correctly Classified Instances	Incorrectly Classified Instances
1000 Trees		
all attributes	75.39%	24.61%
plas,mass	72.01%	27.99%
all others	66.15%	33.85%
100 Trees		
all attributes	75.99%	24.09%
plas,mass	72.01%	27.99%
all others	66.28%	33.72%
10 Trees		
all attributes	73.96%	26.04%
plas,mass	71.48%	28.52%
all others	66.54%	33.46%

Table 6. Classification Error –Diabetes dataset

Conclusion

The initial goal of this project was to fully implement Breiman's random forest algorithm into Weka. Due to the difficulty of his algorithm and the complexity of Weka, only the variable importance was implemented into Weka, but alternative programs were created. The RandomForest java application allows full access to the Breiman's algorithm and is compatible with the Weka's datasets. The source code and the executables for this project can be obtained at the links listed below.

RandomForest GUI
http://s112088960.onlinehome.us/ml2005/Zip%20of%20OTHER%20ournal%20Paper%20Files%20NOT%20JOURNAL%20PAPER/RandomForest.zip

Weka with Variable Importance Add-ons
http://s112088960.onlinehome.us/ml2005/Zip%20of%20OTHER%20Journal%20Paper%20Files%20NOT%20JOURNAL%20PAPER/Weka_Distro_FredLivingston.zip

RAFT Visualization Tools
http://s112088960.onlinehome.us/ml2005/Zip%20of%20OTHER%20Journal%20Paper%20Files%20NOT%20JOURNAL%20PAPER/RAFT_Distro_Livingston.zip

Future Work

Below is a list of possible tasks that could be found beneficial:

- Implementing even more features into Weka and verifying those features.
- Modify code to deal with non-nominal classifier attributes such as reals and numerics
- Interfacing RAFT into Weka

Acknowledgement

I would like to acknowledge the reviewers of my journal paper titled "Implementing Breiman's Random Forest Algorithm into Weka". Their feedback was helpful in approving the quality of this paper.

References

[1] Breiman, Leo
Random Forests. January 2001

[2] Breiman, Leo and Adele Cutler
Random Forests
URL: http://www.stat.berkeley.edu/users/breiman/RandomForests/cc_papers.htm

[3] White, Mark
ECE591Q-Machine Learning – Lecture slides, Fall 2005

[4] Witten, Ian and Frank, Eibe
"Data Mining, Practical Machine Learning Tools and Techniques"