

Spring 2014

ANALYZING BIG DATA WITH DECISION TREES

Lok Kei Leong
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Computer Sciences Commons](#)

Recommended Citation

Leong, Lok Kei, "ANALYZING BIG DATA WITH DECISION TREES" (2014). *Master's Projects*. 366.
DOI: <https://doi.org/10.31979/etd.9wrj-crra>
https://scholarworks.sjsu.edu/etd_projects/366

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

ANALYZING BIG DATA WITH DECISION TREES

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements of the Degree

Master of Science

by

Lok Kei Leong

Spring 2014

© 2014

Lok Kei Leong

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

ANALYZING BIG DATA WITH DECISION TREES

by

Lok Kei Leong

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2014

Dr. Chris Tseng

Department of Computer Science

Dr. Sami Khuri

Department of Computer Science

Mr. Frank Butt

Department of Computer Science

ABSTRACT

ANALYZING BIG DATA WITH DECISION TREES

by Lok Kei Leong

Machine learning is widely used for many current technologies. One of the fundamental machine learning methods is Decision Tree due to its fast learning tasks and consistent prediction results. In this project, we developed machine-learning programs to predict answers in an evaluation dataset after learning from the feature vectors in a provided training dataset. The programs were put to the test in two competitions, The Great Mind Challenge: Watson by IBM, which uses very large datasets, and The IARPA Trustworthiness Challenge by InnoCentive, which uses smaller datasets. This document proposed using Pruning, AdaBoost, RobustBoost, and a hybrid approach with Genetic Algorithm as methods of building decision trees. We developed the programs using Mathworks Matlab and compared the results. We observed that for large datasets, pruning has bad rates of prediction due to overfitting. AdaBoost yielded better rates of prediction but is easily affected by random noise. RobustBoost is able to avoid overfitting and random noise, which makes it the best rate of prediction for large datasets. For small datasets, Pruning, AdaBoost, and RobustBoost yielded the poor prediction rates. The hybrid Genetic Algorithm approach yielded the best prediction rates due to its ability to evolve until identifying the best feature vectors.

ACKNOWLEDGMENTS

I would like to express my gratitude to my project advisor, Dr. Chris Tseng, for his motivation, support, and for giving me opportunities to participate in challenges. His guidance helped me in research and solving problems.

I would like to thank Dr. Sami Khuri and Mr. Frank Butt for contributing as my committee members, and giving me their opinions and comments.

I would also like to thank the Computer Science department's professors and staff members that have been helping me during my study in San Jose State University.

Lastly, I would like to thank my family and friends who have helped me successfully completing this project with their thoughts and encouragement.

TABLE OF CONTENTS

1.0	Introduction.....	10
1.1	Problem Statement	11
2.0	Project Design.....	12
2.1	Decision Tree.....	14
2.2	Overfitting and Pruning.....	15
2.3	AdaBoost	20
2.4	RobustBoost	20
2.5	Feature Selection using Genetic Algorithm	22
2.5.1	Genetic Algorithm	23
2.5.2	Hybrid approach of Genetic Algorithm and Decision Tree	25
2.5.3	Fitness function and scoring judgment.....	26
3.0	Matlab for Challenges.....	27
3.1	Data Preparation for Matlab Code.....	27
4.0	Experimental Outcome and Analysis	29
4.1	The Great Mind Challenge	29
4.1.1	Pruning	29
4.1.2	RobustBoost	31
4.1.3	Feature Selection	34
4.2	The Trustworthiness Challenge.....	35
4.2.1	Pruning	35
4.2.2	RobustBoost	36

4.2.3 Feature Selection	38
5.0 Conclusions.....	40
REFERENCES	42

LIST OF FIGURES

Figure 1 Decision Tree of Discount	15
Figure 2 Decision Tree built from IBM Training Dataset.....	17
Figure 3 Pruned Decision Tree	19
Figure 4 Pruned Decision Tree Classification Rules.....	19
Figure 5 Example of weak classifiers in RobustBoost	22
Figure 6 Original dragons in our gene pool.....	24
Figure 7 two dragons with the most desired traits.....	25
Figure 8 Dragon traits after performing crossover	25
Figure 9 Dragon traits after mutation applied in addition to crossover.....	25
Figure 10 Genetic Algorithm with Decision Tree Hybrid approach.....	26
Figure 11 Pruning Errors in Great Mind Challenge	30
Figure 12 Weak Classifiers and Error Goal	32
Figure 13 Root Mean Square Errors in RobustBoost and AdaBoost	33
Figure 14 Pruning Errors in Trustworthiness Challenge	36
Figure 15 Boosting Algorithm Errors in Trustworthiness Challenge	37
Figure 16 Max and Min Fitness Score in each Iteration	39

LIST OF TABLE

Table 1 Pruning Root Mean Square Rate	30
Table 2 Great Mind Challenge ‘s Root Mean Square Errors in Boost Algorithm	33
Table 3 Pruning Error Rate in Trustworthiness Challenge	35
Table 4 The Trustworthiness Challenge’s Root Mean Square Errors in Boost Algorithm..	37
Table 5 Size of initial gene	39

1.0 Introduction

In 1959, Arthur Samuel defined machine learning as the “field of study that gives computers the ability to learn without being explicitly programmed”. Machine learning is a field of computer sciences that incorporates different algorithms to create a system capable of automatically predicting and taking actions based on data. This master's project involves using machine-learning algorithms to learn from a set of labeled training data and predict values in unseen datasets. This learning method is called supervised-learning. The master's project is divided into two main parts. In the first part, we will create different machine-learning algorithms to be tested in The Great Mind Challenge: Watson Edition (TGMC). The Great Mind Challenge is a series of software development competitions organized by IBM open to university students. The Watson Edition is designed specifically for students attending universities within the United States. The goal of this competition is to create an algorithm capable of analyzing a training dataset in order to predict the answers of an Evaluation dataset with the highest level of accuracy possible. This competition is inspired in the IBM Watson supercomputer, which is a system that was specifically designed to compete in the general knowledge quiz show Jeopardy! to answer questions formulated in natural language. Unlike the IBM Watson system, the algorithm being used for the Great Mind Challenge competition uses numeric datasets to produce True or False answers. In the second part of the project, we will create a machine-learning algorithm for the IARPA Trustworthiness Challenge. This challenge is another machine-learning competition organized by the crowdsourcing

company InnoCentive. The Trustworthiness Challenge is similar to The Great Mind Challenge: Watson Edition. But instead of analyzing the feature vectors to predict the answers as true or false, the IARPA Trustworthiness Challenge uses the feature vectors to label the answers as trustworthy or not. For this competition, the feature vectors represent different kinds of human behavior and the answer labels represents the level of trustworthiness for a person.

In the following sections, we will identify and work on solutions to the problems that The Great Mind Challenge: Watson Edition and the Trustworthiness Challenge present. We will use different algorithms, such as decision tree, pruning, Adaboost, Robustboost, and a hybrid approach of decision tree with Genetic Algorithm, and develop programs to test their performance in the IBM and InnoCentive challenges. Next, we will analyze and discuss the results of the programs in an attempt to identify which algorithms are more effective to create machine-learning systems for this kind of application.

1.1 Problem Statement

The Great Mind Challenge and the Trustworthiness Challenge can be divided in two phases: the testing phase and the evaluation phase. For the first phase of each competition, both IBM and InnoCentive released two CSV files with different datasets. The first CSV files contained the Training datasets. These datasets contained data fields with Question ID, Problem ID, multiple Feature Vectors, and Answers. The second CSV files contained the Evaluation datasets. These datasets contained the same fields as the first CSV files, but the Answers columns were left unanswered. The purpose of the algorithms was to analyze and learn from the numeric patterns of the Feature

Vectors fields provided in the Training dataset, and then, use this information as a reference to predict the data in the Answers fields in the Evaluation datasets and label them as True or False, or Trust or Don't Trust. For the first phase, the challenge participants are allowed to submit the Evaluation datasets with the predicted answers to the IBM or InnoCentive's websites for unlimited verifications. These verifications allow the participants to fine tune their algorithm. For the second phase of the competitions, the evaluation phase, both IBM and InnoCentive provide a new dataset with the answers field blanked. In this phase, participants are only allowed to submit this Evaluation datasets with predicted answers once to get the final judgment for the competitions. The Great Mind Challenge's training dataset contained approximately 2,400 data rows and 321 field columns. The first two columns were the Question ID and Problem ID, respectively. The next 319 columns were Feature Vectors, and the final column contained the Answer. The Trustworthiness Challenge's training dataset contained approximately 430 data rows and 115 field columns. The first four columns were the Question ID and Section ID fields, followed by one Answer column for the trustworthiness conditions. The remaining 109 columns contained the Feature Vectors. The main objective of the two challenges was to predict as many right answers as possible in the evaluation phase, and the participant with the highest amount of correct answers became the winner of the challenge.

2.0 Project Design

For the development and implementation of the programs, we looked into different

machine-learning algorithms that were designed for data classification problems. Among the most notorious algorithms considered for the competitions, we investigated Neural Networks, Data Clustering, Bayesian Networks, Decision Trees, Support-Vector Networks, Genetic Algorithms, and others. Most of those algorithms have the potential to produce good predictions for classification problems. However, some of them were unsuitable for the challenges, since they were not very efficient at predicting large amounts of generic undefined feature data provided. As a result of the investigation, it was decided that the best way of efficiently predicting the answers for the Evaluation datasets was to use the Decision Tree algorithm. A decision tree is a simple straightforward algorithm. It uses a white box process, which can be easy to debug. Moreover, a decision tree can handle missing data, as well as making changes to the structure of the tree using boosting or bagging techniques. This allows decision trees to be used for supervised and unsupervised learning. However, when dealing with very large datasets like the ones used in The Great Mind Challenge and The Trustworthiness Challenge, the Decision Tree algorithm could get overwhelmed by an infinite amount of potential outcomes. More importantly, having so many potential outcomes would reduce the probability of finding the correct answer and would require far more processing power from the computer system running the programs. To reduce the size of the trees and increase the precision of their predictions, we implemented other algorithms into our decision tree. Amongst the algorithms implemented, we have Pruning, AdaBoost, RobustBoost, and a hybrid approach with Genetic Algorithm, which will be discussed in more detail in the following sections.

2.1 Decision Tree

A decision tree is a decision-making technique that is commonly used by making a graphical representation of the possible consequences of a number of given cases. It is called a decision tree since the graph used to represent the ramifications of the possible consequences, resemble the branches of a tree. Because of that, a decision tree can be used as a predictive model in a machine learning application. Kotsiantis has a formal definition of a decision tree as a predictive model; “Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. Instances are classified starting at the root node and sorted based on their feature values” (Kotsiantis, 2007). Decision tree algorithms also have classification models, such as Iterative Dichotomiser 3 (ID3), C4.5, and Classification And Regression Tree (CART). For both The Great Mind Challenge and the Trustworthiness Challenge, we have decided to use the C4.5 classification model to create all decision trees. The C4.5 algorithm uses a set of training data and the concept of information entropy, a measure of the uncertainty in a random variable (Ihara, 1993), to build the decision tree. Because of its common application in classification tasks, C4.5 is usually defined as a statistical classifier. An example of a C4.5 decision tree is given in Figure 1, where we can see a decision tree created to predict if a movie theater customer is eligible to get a ticket discount based on different data features. The data features in this example would be the ages of the customers, and if they are currently students enrolled in a high school or university. Once the decision tree has been built, the algorithm would be able to classify the customers as senior citizens, students, both or none of the two, and then

decide if the customer qualifies for the discount. The example in Figure 1 is just a simplified version of a decision tree using C4.5 since this algorithm can be used in far more complex situations, such as the datasets used in The Great Mind Challenge and The Trustworthiness Challenge.

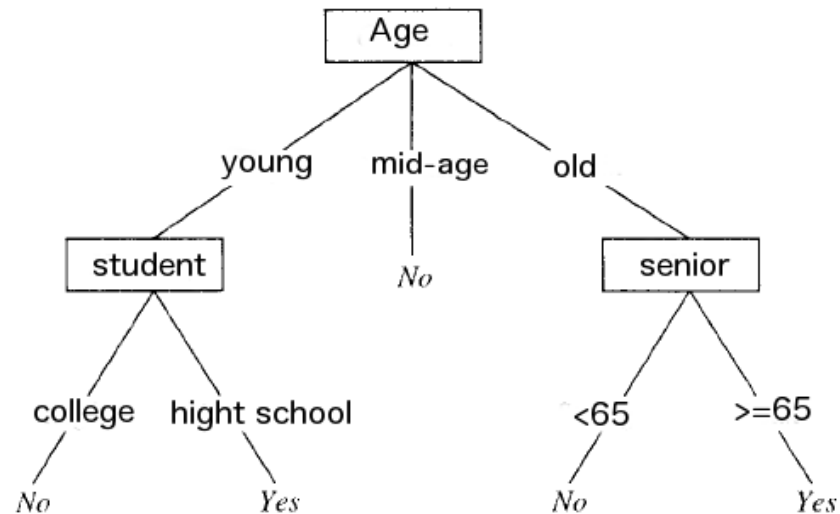


Figure 1 Decision Tree of Discount

The following, is a pseudo-code for building C4.5 decision tree algorithm from (Kotsiantis, 2007):

- 1 Check for base cases
- 2 For each attribute x
 - Find the normalized information gain ratio from splitting on x
- 3 Let the highest normalized information gain be a_best
- 4 Create a decision *node* that splits on a_best
- 5 Recursive on the sub lists obtained by splitting on a_best , and add those nodes as children of *node*

2.2 Overfitting and Pruning

When an algorithm tries to build a decision tree, oftentimes it overfits its training

data. To explain the definition of overfitting, Kotsiantis states that for “a decision tree, or any learned hypothesis h , is said to overfit training data if another hypothesis h' exists that has a larger error than h when tested on the training data, but a smaller error than h when tested on the entire dataset” (Kotsiantis, 2007). For The Great Mind Challenge and The Trustworthiness Challenge, we were given very large sets of data. When the decision tree was built, a lot of branches that were only associated with very few specific cases appeared. Those branches could have confused the decision tree data predictions by creating several potential answers. Because of that, building an entire decision tree utilizing every single value in the dataset may not have helped predicting the best answers for our Evaluation dataset accurately. In Figure 2, we can observe the decision tree built by our algorithm using the Training dataset provided for The Great Mind Challenge. In the figure, we can observe that the decision tree algorithm alone created a very large tree with lots of branches, which represent hundreds of potential answers for our prediction.

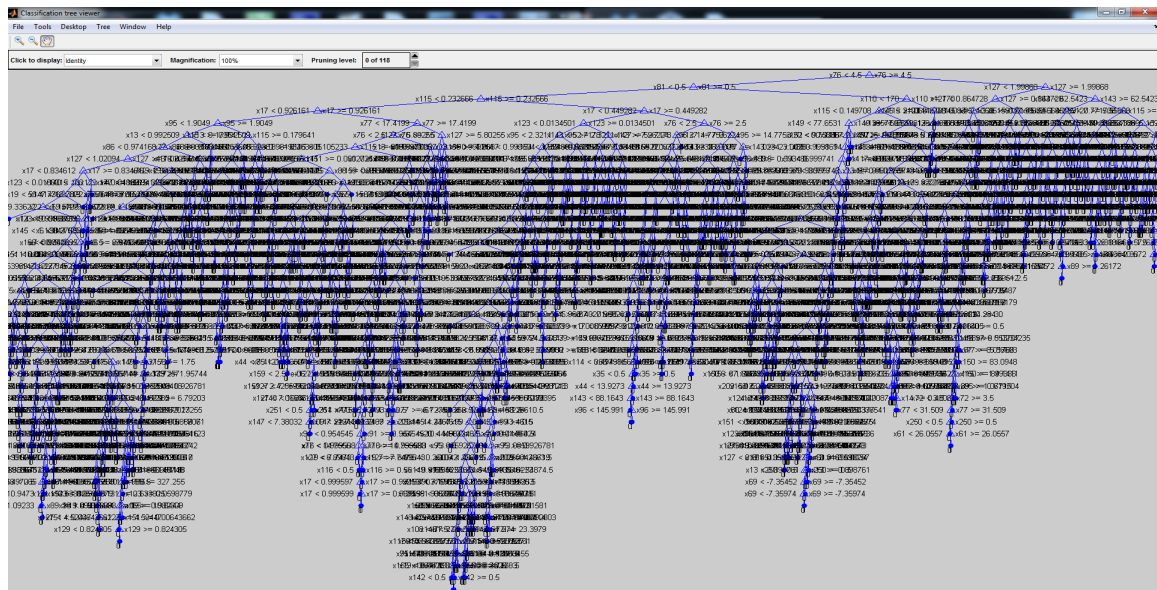


Figure 2 Decision Tree built from IBM Training Dataset

As part of the research, we have looked into five different ways of avoiding overfitting the training data for a decision tree. The first method is to stop the training algorithm before it is able to produce a fully developed decision tree. To do so, a threshold is set up to limit the amount of branches being built in the tree. The threshold would control the size of the decision tree and therefore, the amount of potential answers would be limited. However, the main problem with this method is that some relevant answers might be excluded from the tree since the threshold doesn't discriminate. Because of that, this method was not used for our algorithm. The second method is to prune the branches carrying answers with the least probability of being correct. If two decision trees perform a prediction with the same level of accuracy, the one with the least amount of branches would be preferred. The pruning process can be applied before the decision tree is built or after. When pruning is applied before the

decision tree is built, it is called pre-pruning. When pruning is applied after the decision tree is built, it is called post-pruning. To optimize the decision tree for both The Great Mind Challenge and The Trustworthiness Challenge, we decide to use post-pruning algorithms. After our algorithm created the decision trees using the Training datasets, it had to calculate the best amount of pruning to be applied to the trees based on classification error. To calculate the classification error for each level of pruning, the algorithm used 10-fold cross validation. The level that came up with the lowest rate of classification error was then set as the optimal level. Once the optimal level was defined, the algorithm was ready to prune the decision tree. In Figure 3, we can observe a pruned version of one of the decision trees created for The Great Mind Challenge. Compared to the un-pruned decision tree shown in Figure 2, the new decision tree is much more smaller with a maximum length of six branches and maximum width of four branches. We can observe the classification rules for the pruned tree in Figure 4.

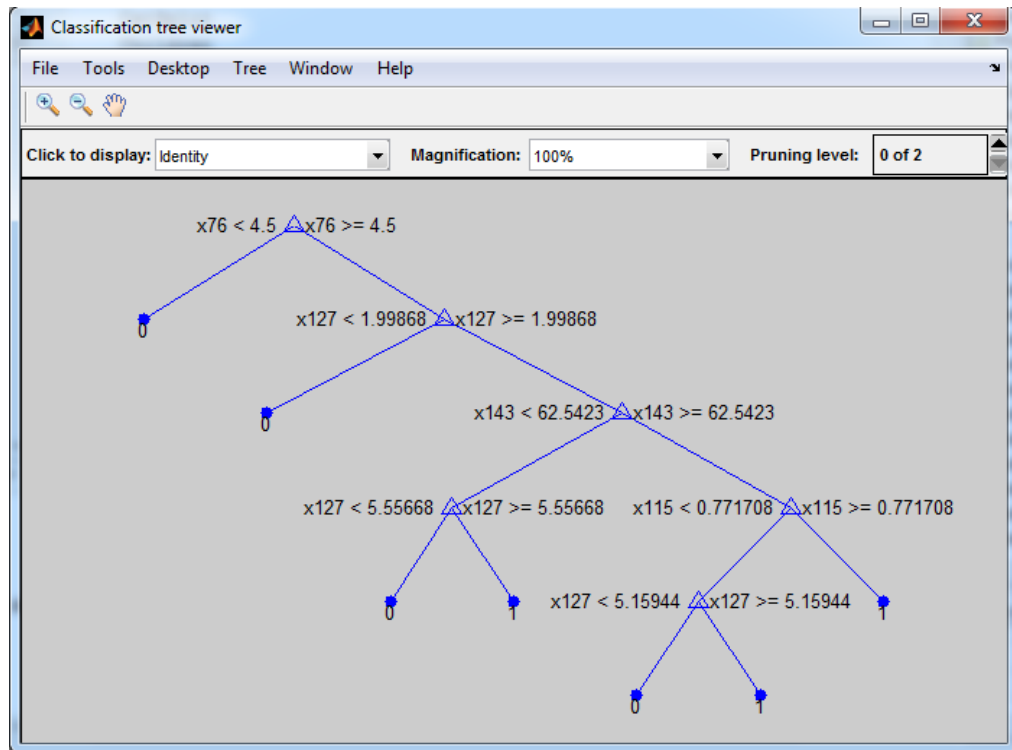


Figure 3 Pruned Decision Tree

```
>> view(PrunedLearningTree)

Decision tree for classification
1  if x76<4.5 then node 2 elseif x76>=4.5 then node 3 else 0
2  class = 0
3  if x127<1.99868 then node 4 elseif x127>=1.99868 then node 5 else 0
4  class = 0
5  if x143<62.5423 then node 6 elseif x143>=62.5423 then node 7 else 1
6  if x127<5.55668 then node 8 elseif x127>=5.55668 then node 9 else 0
7  if x115<0.771708 then node 10 elseif x115>=0.771708 then node 11 else 1
8  class = 0
9  class = 1
10 if x127<5.15944 then node 12 elseif x127>=5.15944 then node 13 else 1
11 class = 1
12 class = 0
13 class = 1
```

Figure 4 Pruned Decision Tree Classification Rules

2.3 AdaBoost

The third method used to attenuate the effects of overfitting is known as AdaBoost, the short form for Adaptive Boosting, is a boosting algorithm created by Yoav Freund and Robert Schapire. The algorithm combines multiple weak classifiers to create one single strong classifier by using multiple weighted samples in training stages. As a result, the system is capable of focusing in learning from the most difficult examples instead of combining classifiers that have equal weight. The AdaBoost algorithm improves the prediction progressively depending on the time spent learning and the number of weak classifiers being used. One disadvantage for AdaBoost is that it gives too much weight to outliers or data that is irrelevant. Therefore, if the dataset where AdaBoost is being applied has lots of noisy data, the algorithm could produce incorrect predictions. Nevertheless, applying AdaBoost is a good way of avoiding training data overfits for a decision tree if the amount of noise is low.

2.4 RobustBoost

Implementing the AdaBoost algorithm in our decision tree allowed our program to reduce the size of its decision tree and improve the prediction results. However, we still needed to decrease the overfitting effect coming from noisy data due to the size of the datasets evaluated for both IBM and InnoCentive's challenges. To do so, we found another boosting algorithm called RobustBoost (Freund, 2009). RobustBoost works in a similar manner to AdaBoost. Nevertheless, RobustBoost was designed to be more resistant to the effects of random data noise and imbalanced data in comparison to AdaBoost. To decrease the effect from outliers, RobustBoost uses a classification margin

threshold, which limits how much the decision tree can grow within the training dataset in order to minimize the number of training samples being created for the training dataset. Also, to minimize the cost functions, RobustBoost normalizes the relevance weight of each vector. This normalization process can reduce the effects from outliers when creating decision trees. Therefore, RobustBoost is able to perform better average classifications with more accuracy.

The pseudo-code for RobustBoost algorithm by Freund is shown below:

1. The algorithm starts at $t = 0$.
2. At every step, Robust Boost solves an optimization problem to find a positive step in time Δt and a corresponding positive change in the average margin for training data Δm .
3. RobustBoost stops the training and exits if at least one of these three conditions is true:
 - Time $t \geq 1$.
 - RobustBoost cannot find a solution to the optimization problem with positive updates Δt and Δm .
 - RobustBoost grows as many learners as requested.

RobustBoost is a self-terminating algorithm. It will end the learning process as soon as the time is greater or equal to one. If the error goal is set to a number that is too small, then RobustBoost will not terminate the process. Setting the right value for the error goal is done by searching for the minimal value of error rates for which the algorithm terminates within a reasonable number of iterations. Figure 5 shows one of the weak classifiers created using Robustboost for The Great Mind Challenge dataset. After running our program, we determined that using 1820 weak classifiers and a 0.1 error goal gives the best prediction result for The Great Mind Challenge.

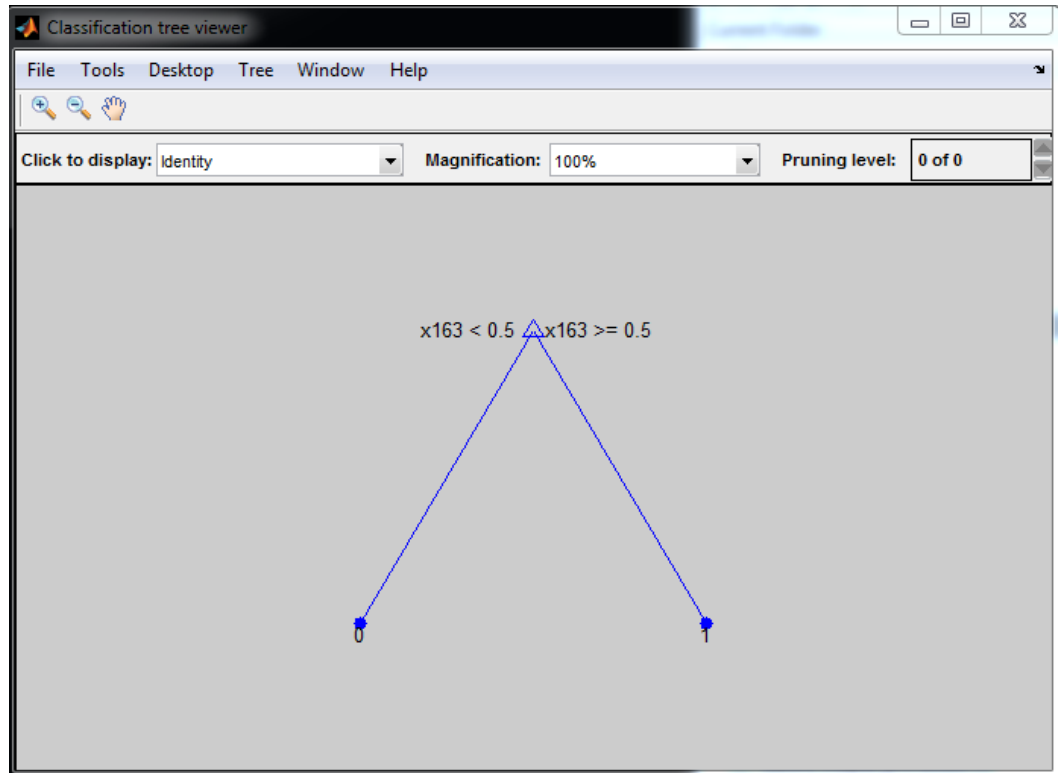


Figure 5 Example of weak classifiers in RobustBoost

2.5 Feature Selection using Genetic Algorithm

The fifth method for preventing building a large decision tree is by selecting the most important feature vectors from the dataset and building a smaller decision tree. The datasets provided by The Great Mind Challenge and The Trustworthiness Challenge have a huge amount of feature vectors and not all of the features may be useful for decision making. Therefore if we can eliminate the false feature vectors, we could have a better prediction. One of the ways of eliminating the false feature vectors is by using Genetic Algorithm. Stein et al. have done similar research on feature selection using Genetic Algorithm, in our project, we follow Stein et al's method and apply on the challenges.

2.5.1 Genetic Algorithm

Genetic Algorithm (GA) is a searching algorithm designed to mimic the biological process of evolution by natural selection. Forrest compares genetic algorithms and natural selection, “Genetic algorithms are loosely based on ideas from population genetics; they feature population genotypes, an individual’s genetic material, stored in memory, differential reproduction of these genotypes, and variations that are created by processes analogous to the biological processes of mutation and crossover” (Forrest, 1993). A Genetic Algorithm starts with a large population of potential solutions to a problem. The potential solutions evolve towards even better solutions. Each potential solution has a set of properties called hypothesis, which can mutate or alter. Usually the initial hypothesis is randomly generated and is evaluated through fitness functions. If the hypothesis has a higher fitness score it will be selected in the next generation of potential solutions. The next generation of solution is generated by the best two hypothesis with crossover or mutation processes. The hypothesis will continue changing until it either fulfills the fitness function requirements or exceeds the maximum number of generations.

A crossover is a way of exchanging and combining two separate hypotheses. Genetic Algorithm chooses a random point in a hypothesis and swaps and combines the first half of the first hypotheses and the second half to the other hypotheses. One possible downside on using crossover is that it could take several generations of evolution before it generates good types of hypotheses. A mutation is a process that randomly adds or deletes data from the hypothesis to give it more variety. An example is given in Figure 6. Figure 6 shows three types of dragons with their different genetic traits or

characteristics. Our goal is to create a dragon which genetic traits make it able to fly, has strong eyesight, and has strong teeth. A fitness function identifies the dragons with the most number of desired traits and eliminates from the gene pool the dragon that doesn't have enough desired traits, as can be observed in Figure 7. To create our dragon, we can perform a crossover, find a random point in the genes, split the gene in half, and swap the first half with the first gene and the second half with the second gene. This would mix and combine the characteristics of the two genes. The process would be repeated until we end up with a dragon with the desired characteristics, as shown in Figure 8. Now, let's assume that in addition to the traits mentioned before, we also want to include the ability to swim trait to our new dragon. Since we already eliminated the third original dragon from our process, its traits are not available to our new dragon's gene pool anymore. Thus, we are unable to add this new trait to our dragon-using crossover. To solve this problem, we can apply mutation to the dragon creation process. This process would add random traits to our new dragon until we obtain the perfect individual, as shown in Figure 9.




	Can fly	Toothless	Good eyesight	Green
	Can run	Strong teeth	Bad eyesight	Green
	Can swim	Toothless	Bad eyesight	Red

Figure 6 Original dragons in our gene pool



	Can fly	Toothless	Good eyesight	Green
	Can run	Strong teeth	Bad eyesight	Green

Figure 7 two dragons with the most desired traits


	Can fly	Strong teeth	Good eyesight	Green
---	---------	--------------	---------------	-------

Figure 8 Dragon traits after performing crossover

	Can fly	Strong teeth	Good eyesight	Green	Can swim
---	---------	--------------	---------------	-------	----------

Figure 9 Dragon traits after mutation applied in addition to crossover

2.5.2 Hybrid approach of Genetic Algorithm and Decision Tree

Genetic Algorithm is known for optimization in large datasets (Mitchell, 1996). Because of this conception, we believe Generic Algorithm can help our program finding the most meaningful feature vectors in the Great Mind Challenge and the Trustworthiness Challenge datasets. In our prediction program, Genetic Algorithm is applied before the decision tree is built. To perform our predictions, we divided the training datasets into 70% for training and 30% for testing, and then we create 20 distance genes. From each gene, the program randomly generates a potential solution population based on the data obtained from the training dataset. Next, the Genetic Algorithm selects features vectors from the dataset and creates a decision tree. The decision tree then attempts to predict the answers in the testing dataset and calculates the prediction score. The prediction score is then compared with the fitness function to

identify the genes that have the two highest scores. Once we identified the top genes in our gene pool, we perform a crossover. If the prediction scores of the new genes are better, Genetic Algorithm replaces the two genes that have the lowest score with the new better genes. If the prediction scores reach a tie, which means the highest score and the lowest score end up being the same, then the algorithm randomly selects a gene and perform mutation. The mutation gives our gene pool more variety and therefore, better prediction scoring genes could be created. The entire process is repeated over and over until our program obtains an optimal prediction score to generate the final decision tree or reaches the maximum number of evolved generations. Once the final decision tree is created, the program is ready to be applied to the Evaluation dataset. Figure 10 shows the processes of feature selection using Genetic Algorithm.

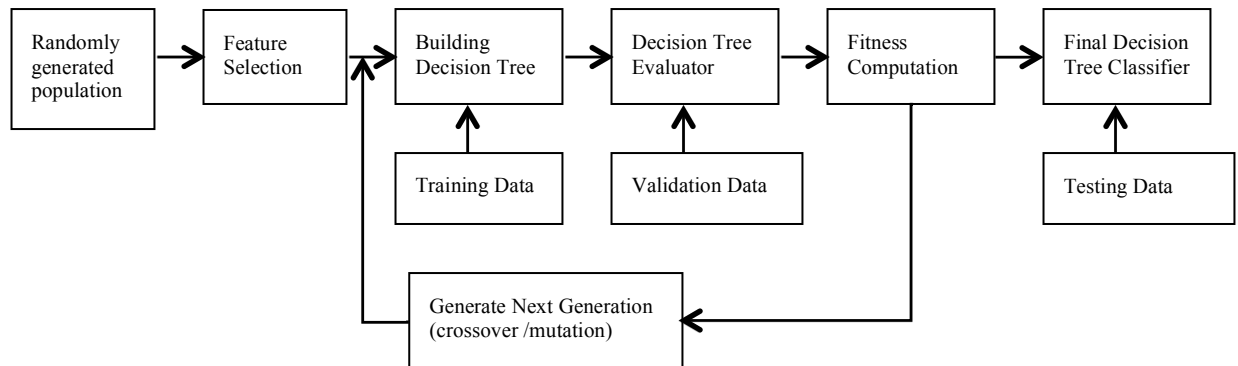


Figure 10 Genetic Algorithm with Decision Tree Hybrid approach

2.5.3 Fitness function and scoring judgment

The fitness function in Genetic Algorithm follows the scoring method used by The Great Mind Challenge. For The Great Mind Challenge scoring method, if the answers in the prediction and the answer key are both true, one point is added. If the prediction answer is true, but the answer in the answer key is false, one point is deducted. If the

prediction answer is false, and the answer in the answer key is either true or false, we do not deduct or add any points.

3.0 Matlab for Challenges

For this project, we chose to use Matlab, a numerical computing environment. Matlab has a friendly user interface, as well as easy access to virtualization and a wide range of toolboxes capable of executing pruning and boosting algorithms. In order to run our algorithms, we required a computer system with MathWorks Matlab with the Optimization Toolbox set installed. For this project, we used Matlab version R2013b in Windows 7. The program was installed in a PC with an Intel i5-2500k CPU clocked at 4.2GHz and 16GB of RAM, which provided enough computer resources to execute our algorithm.

3.1 Data Preparation for Matlab Code

Before analyzing the Training dataset, we need to modify the raw data files provided by IBM and InnoCentive in order to build the decision trees properly. In The Great Mind Challenge, the answers field in the Training dataset displays a string value of either “True” or “False”. We converted the “True” values to 1 and the “False” values to 0 to allow Matlab to recognize the answers as binary Boolean outputs. The size of the Training datasets weighed approximately 390MB, while the Evaluation datasets weighed around 84MB. Both the Training and Evaluation datasets provided for The Great Mind Challenge are relatively big compared to the Trustworthiness Challenge, and because of this, our computer system was able to execute our machine-learning program without

using up all the computer resources in our system. For the Trustworthiness Challenge, two of columns in the training dataset need to be converted. The B-ALS column displays a string value of either “High” ,”Medium” or “Low”. This column contains the risk of trusting a in the dataset person. For our algorithm, we converted the “High” values to 1, the “Medium” values to 0.5, and the “Low” values to 0, in order to give them a numeric representation for within the program. The second column that needs to be converted is the answers field, which displays answers as “Exact amount promised.”, “More than promised.” , “Promise not fulfillable.”, or “Less than promised.”. According to The Trustworthiness Challenge guidelines, those conditions are ultimately used to label the people in the dataset as trustworthy and untrustworthy. Therefore, we converted the answer values “Exact amount promised.” and “More than promised.” to a 1, and the answer values “Promise not fulfillable.”, or “Less than promised.” to a 0. These two numbers were used as numeric representations of trustworthy and untrustworthy, respectively.

4.0 Experimental Outcome and Analysis

In this section, we are comparing the error rates produced by the Pruning, AdaBoost, RobustBoost, and the hybrid decision tree algorithms that were applied to The Great Mind Challenge and The Trustworthiness Challenge. Since as part of both competitions, we are not able to obtain the answer key for the Evaluation datasets in both challenges, we decided to use the training datasets to perform the tests on the prediction accuracy for each algorithm. For these tests, we used 70% of the dataset for training and 30% for evaluation. Next, we compared the results the newly evaluated dataset with the already known answers.

4.1 The Great Mind Challenge

4.1.1 Pruning

In order to evaluate the efficiency of 10-fold cross-validation as a way of finding the most optimal level of pruning, we tested different levels of pruning in the decision tree. From testing results shown in Table 1, as well as in Figure 11, we can observe that a pruning level of 80 yields the smallest error rate. We can also notice that after we increased the level of pruning, the root mean square error also started decreasing until the decision tree reached the maximum level of pruning. Any pruning after we applied reach maximum level will not work because the algorithm would start removing potential answers with high probability of occurrence. In the decision tree created for The Great Mind Challenge's Training dataset, the maximum level of pruning was 85, and applying any higher level of pruning resulted in an error in our program. Additionally, the optimal

pruning level in a decision tree is not a fixed numeric value. The levels of pruning are dependent on the structure of the decision tree.

Table 1 Pruning Root Mean Square Rate

Pruning Level	Root Mean Square Error
0	0.1363
10	0.1355
20	0.1329
30	0.1272
40	0.1183
50	0.1124
60	0.1071
70	0.1049
80	0.1038
81	0.1040
82	0.1042
83	0.1040
84	0.1049
85	0.1104

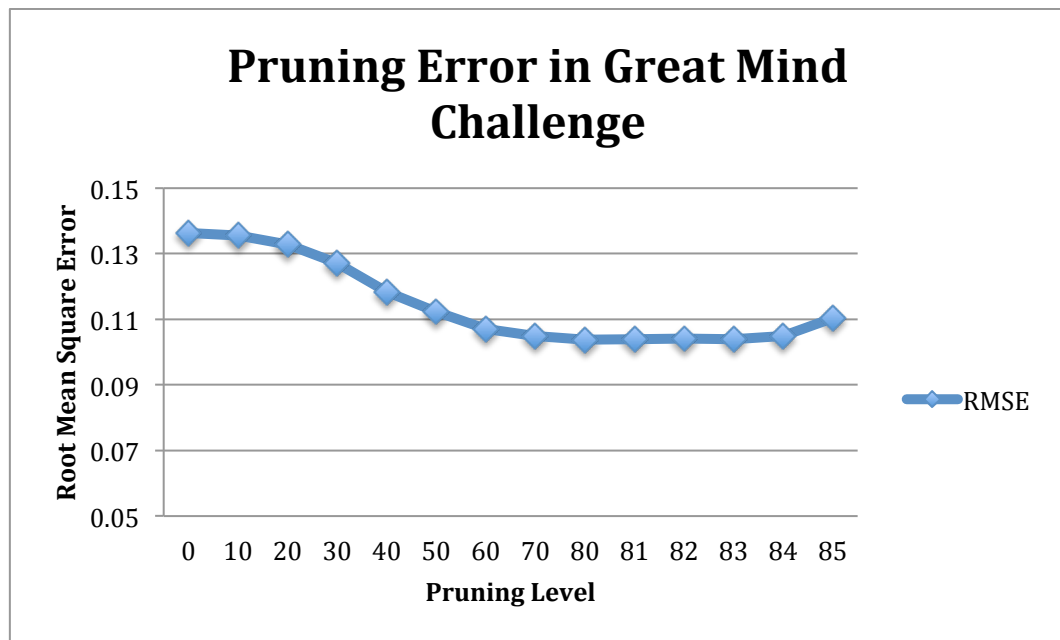


Figure 11 Pruning Errors in Great Mind Challenge

4.1.2 RobustBoost

The Matlab RobustBoost function has four parameters that allow the adjustment of the prediction accuracy: number of weak classifiers, RobustErrorGoal, RobustMaxMargin, and RobustMarginSigma. The RobustErrorGoal parameter is the target classification error, ranging from 0 to 1. The RobustMaxMargin parameter is the maximum classification margin in a training set. The margin minimizes the number of observations in the training set and acts as the bottleneck for classification margins. The RobustMarginSigma parameter represents the variation of the output value. This parameter is used for classification margins in the training set, and only allows positive numeric values. For The Great Mind Challenge, we set the RobustErrorGoal parameter to 0.01, the RobustMaxMargin parameter to 0, and the RobustMarginSigma parameter to 0.01. In order to test the effect of the number weak classifiers used for RobustBoost, we tested the algorithm with up to 1820 weak classifiers. In Figure 12, we can observe that as we get a higher amount of weak classifiers involved with the training, the error rate gets closer to the error goal.

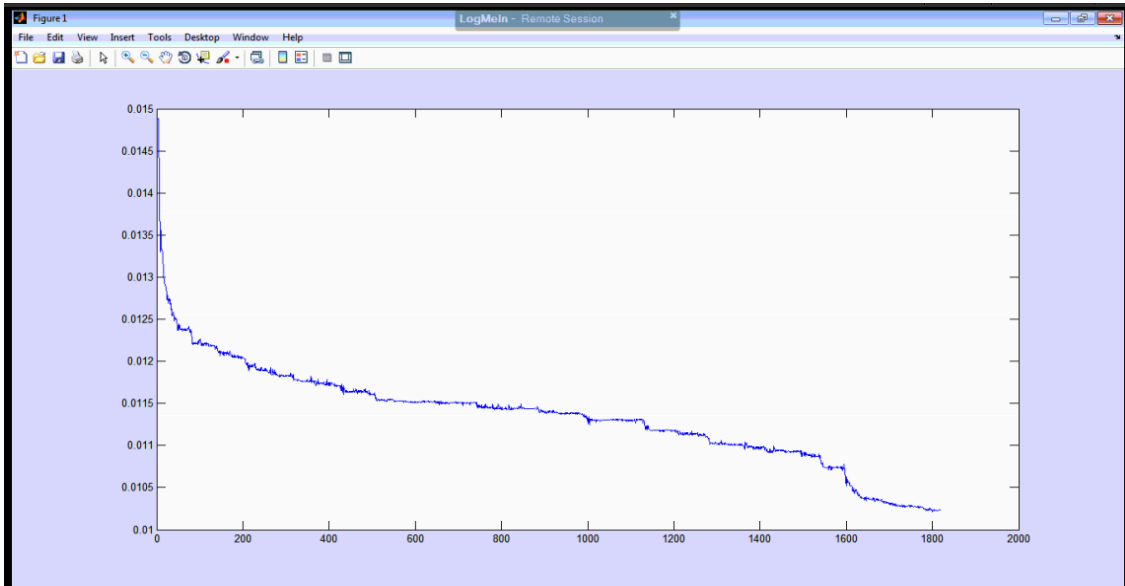


Figure 12 Weak Classifiers and Error Goal

From the results in Table 2, we can observe that the root mean square error obtained after applying RobustBoost is smaller than the root mean square error obtained after using the pruning and AdaBoost algorithms. Figure 13 shows that as the number of weak classifier increases, the root mean square error decreases. Nevertheless, the biggest challenge of using RobustBoost is to be able to find the right amount of weak classifiers, since having too many weak classifiers would require more time for training and could also increase the probability of predicting bad results. Figure 13 also shows the RobustBoost root mean square error fluctuating higher and lower. For this experiment, the best number of weak classifier was found to be 250. In Figure 13 we can also observe the results from AdaBoost, which ended up having a much higher error rate than RobustBoost. AdaBoost also showed the same unstable behavior as RobustBoost.

Table 2 Great Mind Challenge ‘s Root Mean Square Errors in Boost Algorithm

Number of Weak Classifiers	Root Mean Square Error AdaBoost	Root Mean Square Error RobustBoost
25	0.1077	0.1026
50	0.1037	0.1016
100	0.1034	0.1011
150	0.1027	0.1005
250	0.1021	0.0994
500	0.1019	0.0997
750	0.1021	0.1001
1000	0.1026	0.0999
1250	0.1027	0.1000
1500	0.1025	0.0997
1750	0.1026	0.0997
1820	0.1025	0.0998

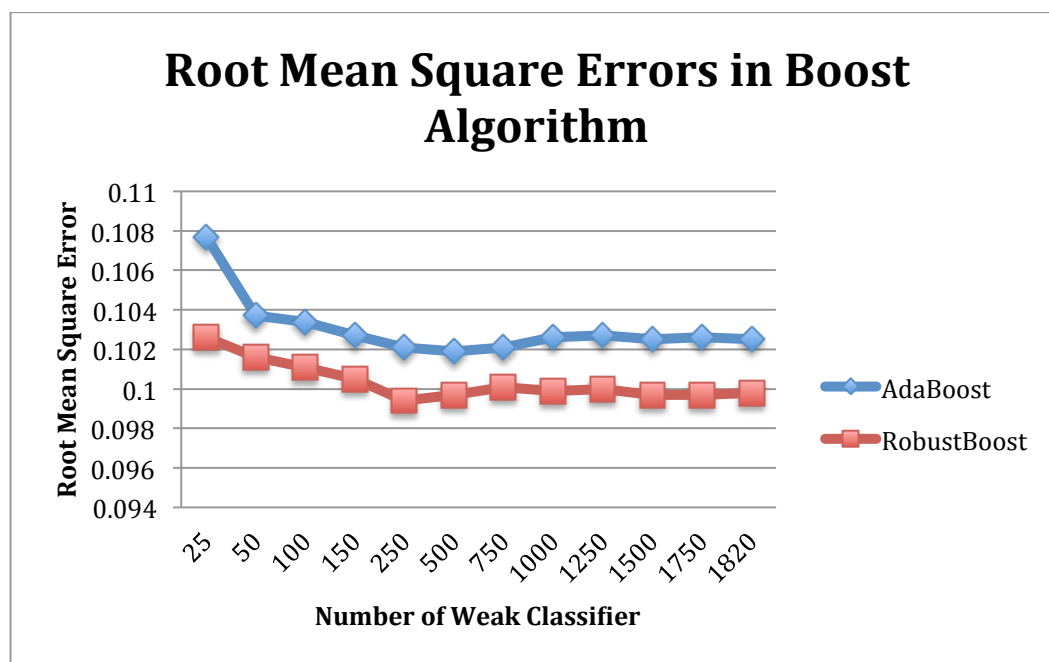


Figure 13 Root Mean Square Errors in RobustBoost and AdaBoost

4.1.3 Feature Selection

In the Great Mind Challenge training dataset, we noticed that 72 feature vectors have the same value throughout the entire dataset. Therefore, those feature vectors can be removed in order to reduce the number of features used for building the decision tree. Besides having repeated values, some features vectors may also have values with no effect on the decision making process. To deal with these feature vectors, we tried to apply the hybrid approach to select the most useful features used for building the decision tree. However, due to the large size of the datasets used in the competition, building decision trees on each iteration required an excessive amount of system resources. As a result, the entire program took several hours and even more than a full day to run. Because of this, we were not able to find the optimal features to build the decision tree using this specific algorithm due to its impracticality. We also attempted to run the program using Amazon's Elastic Computing Cloud (EC2) as a system resource, which offers a 32-core Xeon E5-2680 v2 processor running at 3.2 GHz and 60 GB of RAM memories. But since the algorithms are applied using Matlab tools optimized for single-core processing, running the programs in EC2 actually took longer than our local system. Therefore, we consider that using the hybrid approach to eliminate low relevance features vectors is not suitable for The Great Mind Challenge datasets.

4.2 The Trustworthiness Challenge

4.2.1 Pruning

We used the 10-fold cross-validation to find the best level for pruning. As a result, the best level of pruning in the Trustworthiness Challenge's decision tree was found at the eighth level, which yielded a root mean square error of 0.5261. Although the eighth level gives the smallest root mean square error, the decision tree predicted every answer as 'Trustworthy'. This occurred because the Trustworthiness Challenge dataset is much smaller than The Great Mind Challenge's dataset. Therefore, pruning a relatively small decision tree is not a suitable method for the Trustworthiness Challenge since the pruning could end up making the rate of prediction worse.

Table 3 Pruning Error Rate in Trustworthiness Challenge

Pruning Level	Root Mean Square Error
0	0.5901
1	0.6124
2	0.6268
3	0.6268
4	0.6268
5	0.6124
6	0.6196
7	0.5825
8	0.5261

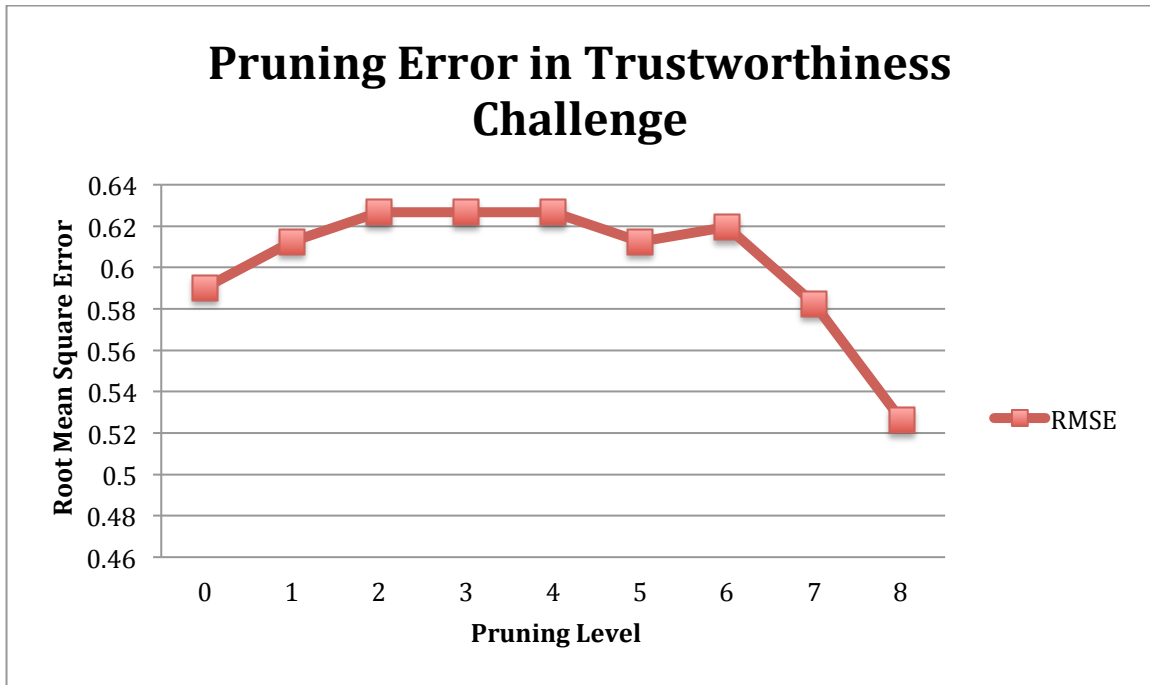


Figure 14 Pruning Errors in Trustworthiness Challenge

4.2.2 RobustBoost

Since The Trustworthiness Challenge's dataset has a similar structure to the dataset provided by The Great Mind Challenge, it was initially thought that RobustBoost and AdaBoost would improve our rates of prediction. However, as observed in the results in Table 4 the root mean square error for the predictions doesn't seem to see an impact after we apply the algorithms. We do not see a lot of improvement either after increasing the amount of weak classifiers. Therefore, RobustBoost and AdaBoost are not a suitable method for the Trustworthiness Challenge.

Table 4 The Trustworthiness Challenge's Root Mean Square Errors in Boost Algorithm

Number of Weak Classifiers	Root Error Rate AdaBoost	Root Error Rate RobustBoost
25	0.6409	0.6478
50	0.5976	0.5901
100	0.5825	0.5669
150	0.5510	0.5748
250	0.5825	0.5825
500	0.5669	0.5669
600	0.5748	0.5590
650	0.5825	0.5590

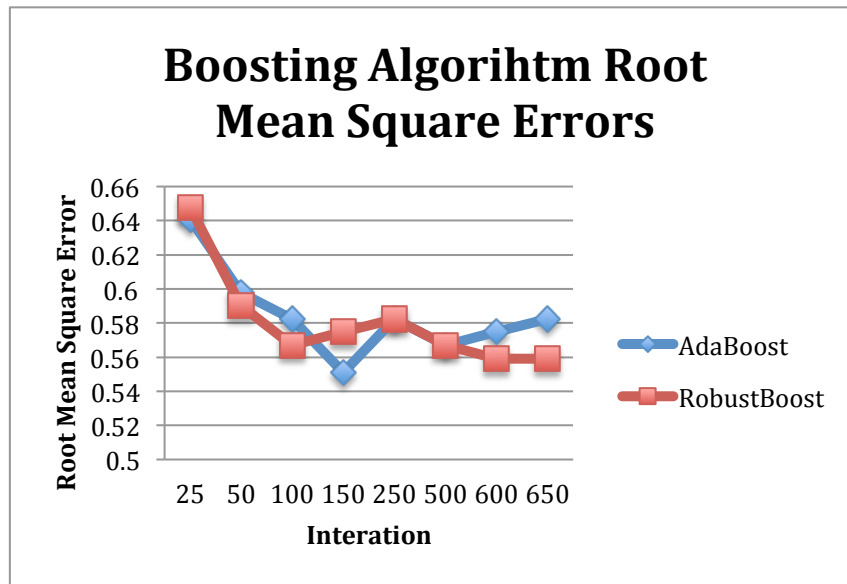


Figure 15 Boosting Algorithm Errors in Trustworthiness Challenge

4.2.3 Feature Selection

In the Trustworthiness Challenge, we created 50 genes for the Genetic Algorithm feature selection process. As the results in Table 5 demonstrate, we experimented with different sizes of initial gene pools and found out that a gene that has around 50 features yields the lowest root mean square error. The original dataset has a total of 109 features, and after applying Genetic Algorithm; our program selected the 50 features that were most useful for making predictions. Figure 16 shows the score obtained by the fitness function during the training process. The figure shows the level of improvement in each iteration. The blue line in the graph represents the highest score on each iteration. The red line represents the lowest score. We observe that after applying many crossovers, the maximum and the minimum scores end up being the same. At this point we apply mutation to randomly add or delete features that could potentially improve the score. If the mutation is not able to improve the score, the genetic algorithm process will stop and return the optimal features. Also, from Table 5, we can observe that using all of the features for predicting yields the highest root mean square error. Therefore, selecting fewer features can potentially improve the prediction rates.

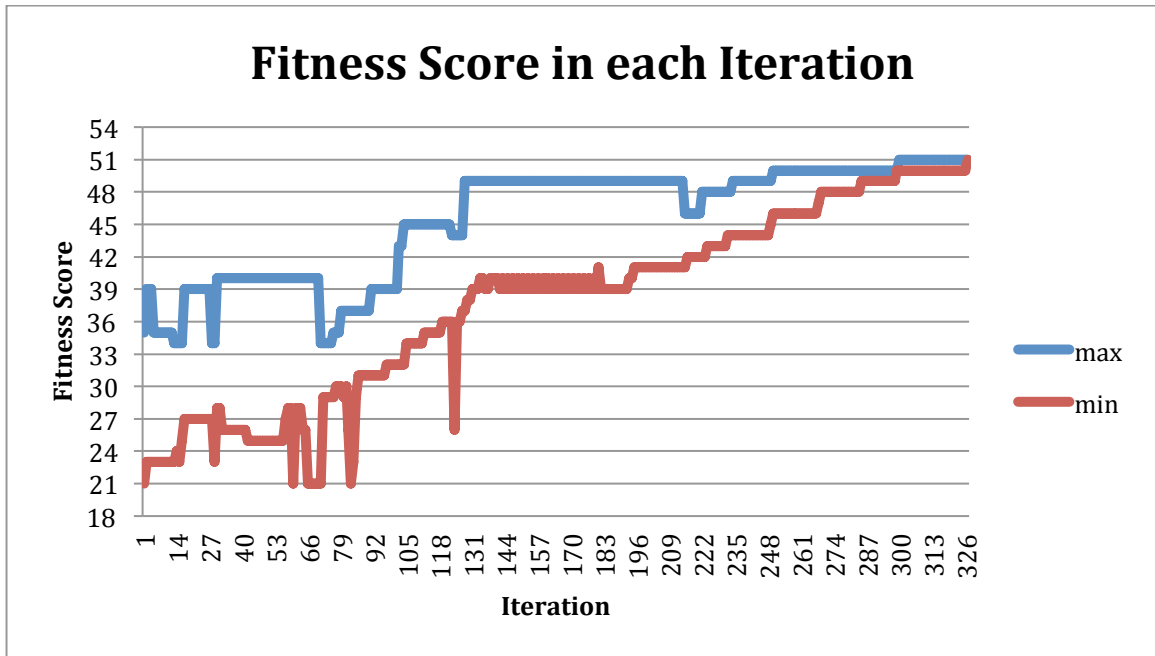


Figure 16 Max and Min Fitness Score in each Iteration

Table 5 Size of initial gene

Numbers of features pick initial	Number of features picked after GA	RMSE
109	109	0.5901
100	103	0.5261
70	79	0.5000
50	63	0.4629
30	52	0.4725

5.0 Conclusions

For The Great Mind Challenge and The Trustworthiness Challenge, we proposed creating a supervised learning program using decision trees with different algorithms: Pruning, AdaBoost, RobustBoost, and a Genetic Algorithm Hybrid. Out of the four algorithms, RobustBoost produced the best rate of prediction in the Great Mind Challenge, while the Decision Tree with Genetic Algorithm hybrid produced the best rate of prediction in the Trustworthiness Challenge. Using Adaboost was inefficient for this type of datasets due to the susceptibility to data noise while Pruning was very limiting and was unable to discern between weak and strong classifiers.

For The Great Mind Challenge, the RobustBoost approach performed better than the other algorithms due to its ability of removing noisy data. In order to obtain good rates of prediction, our training program identified and analyzed weak classifiers. While a larger number of weak classifiers improved our prediction results, it also made the execution time much longer. Because of this, defining the right amount of weak classifiers was crucial in order to run the program efficiently. The decision tree with Genetic Algorithm hybrid approach was not used for the Great Mind Challenge due to its inefficiency. This was caused by the large size of the datasets provided by IBM, which required several hours or days to run for each iteration of our training program. Nevertheless, the hybrid approach proved to be very effective at identifying the best feature vectors in smaller datasets. This allowed us to build optimal decision trees for the datasets in the Trustworthiness Challenge. On the other hand, RobustBoost was unable to find enough

weak classifiers in these datasets due to the small amount of data available for the training process.

From these results, we can conclude that the RobustBoost algorithm can provide the best approach if we are dealing with very large datasets with several feature vectors available for training. For smaller the datasets, the decision tree with Genetic Algorithm hybrid approach proved to produce the best predictions rates due to its ability of improving its results after each program iteration.

REFERENCES

- Experiments with a new Boosting Algorithm
<http://cseweb.ucsd.edu/classes/fa01/cse291/AdaBoost.pdf>
- Forrest, Stephanie (1993, Aug. 13) Genetic Algorithms: Principles of Natural Selection Applied to Computation, Vol. 261, No. 5123. p. 872-878
- Freund, Yoav (2009, May 13). A more robust boosting algorithm. Retrieved May 13, 2009 from <http://arxiv.org/pdf/0905.2138.pdf>
- Freund, Yoav (2009, June). Drifting Games, Boosting and Online Learning. Retrieved June 2009 from http://dev.videolectures.net/icml09_freund_dgb/
- Gary Stein, Bing Chen, Annie S. Wu, and Kien A. Hua. (2005). Decision tree classifier for network intrusion detection with GA-based feature selection. In Proceedings of the 43rd annual Southeast regional conference - Volume 2 (ACM-SE 43), Vol. 2. ACM, New York, NY, USA, 136-141. DOI=10.1145/1167253.1167288 from <http://doi.acm.org/10.1145/1167253.1167288>
- Ihara, Shunsuke (1993). Information theory for continuous systems. World Scientific. p. 2. ISBN 978-981-02-0985-8.
- Kotsiantis, S. B. (2007, July 16). Supervised Machine Learning: A Review of Classification Techniques. Retrieved July 16, 2007, from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.9683&rep=rep1&type=pdf>.
- Mohri, M., & Rostamizadeh, A. (2012). Learning scenarios. Foundations of machine learning (p. 7). Cambridge, MA: MIT Press.
- MathWorks filensemble <http://www.mathworks.com/help/stats/fitensemble.html>
- The Great Mind Challenge Watson Edition 2013 Official site
<https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=116b5889-5469-4114-971c-21cc9c5dd859>