Johnny Polimeni Collaborators: Luke Gerstner, Isaac Wong

Project 4

Jpolimen

CSC 242

Project 4 Writeup

**Classes:**

The classes that we used from Professor Ferguson for decision tree learning were AbstractDecisionTreeLearner, DecisionTree, Domain, Example, Problem, Variable, YesNoDomain, VectorOps,WillWaitProblem, and ArraySet. The classes we made were DecisionTreeLearner. In addition to the will wait problem we also implemented 8 other data sets that function with our decision tree. We also implemented 9 different data sets. The Will Wait data was given to us and the others came from http://archive.ics.uci.edu.

**Decision Tree Learning:**

Decision tree leaning is a type of machine learning that takes in attribute values and returns a decision. The decisions are made by following a series of tests. The tests happen at each node of the tree. A decision tree is created by using the decision tree learning algorithm. This is a greedy divide and conquer algorithm that works by testing the most important attributes first. This is done in the AbstractDecisionTree Class in the mostImportantVariable method. When creating the tree it is important to start because it reduces the size of your tree. By reducing the size of your tree you’re able to process data quicker and more efficiently. We can see in the analysis section how important it is to lay out your tree correctly. In our DecisionTreeLearner class we implement the DecisionTreeLearning Algorithm. The algorithm takes in a set of examples, a list of variables, and a set of parent examples. This algorithm starts by identifying the most important variable then from there a tree is created and, the rest of the variables are put into a new list and the process is done again until the tree is complete. After the tree is completed we test it on some data. The data is processed at each node of the tree until a leaf node is found, which returns a result. Reasons for incorrect answers are due to overfitting, which is when the tree memorizes the training data and doesn’t generalize very well to new data.

**Examples:**

[Alternate, Bar, Fri/Sat, Hungry, Patrons, Price, Raining, Reservation, Type, WaitEstimate] -> WillWait

[No, Yes, 0-10, $$$, Some, Yes, French, No, No, Yes] -> Yes

[No, Yes, 30-60, $, Full, Yes, Thai, No, No, No] -> No

[No, No, 0-10, $, Some, No, Burger, No, Yes, No] -> Yes

[Yes, Yes, 10-30, $, Full, Yes, Thai, Yes, No, No] -> Yes

[Yes, No, >60, $$$, Full, Yes, French, No, No, Yes] -> No

[No, Yes, 0-10, $$, Some, No, Italian, Yes, Yes, Yes] -> Yes

[No, No, 0-10, $, None, No, Burger, Yes, Yes, No] -> No

[No, Yes, 0-10, $$, Some, No, Thai, Yes, No, Yes] -> Yes

[Yes, No, >60, $, Full, No, Burger, Yes, Yes, No] -> No

[Yes, Yes, 10-30, $$$, Full, Yes, Italian, No, Yes, Yes] -> No

[No, No, 0-10, $, None, No, Thai, No, No, No] -> No

[Yes, Yes, 30-60, $, Full, Yes, Burger, No, Yes, No] -> Yes

WaitEstimate

0-10:

Alternate

No:

Bar

No:

Fri/Sat

No:

Hungry

No:

No

Yes:

Yes

Yes:

Yes

Yes:

Fri/Sat

No:

Hungry

No:

Patrons

None:

No

Some:

Yes

Full:

Yes

Yes:

Yes

Yes:

Yes

Yes:

Yes

10-30:

Alternate

No:

Yes

Yes:

Bar

No:

Yes

Yes:

No

30-60:

Bar

No:

No

Yes:

Yes

>60:

No

[No, Yes, 0-10, $$$, Some, Yes, French, No, No, Yes] -> Yes

Yes

[No, Yes, 30-60, $, Full, Yes, Thai, No, No, No] -> No

No

[No, No, 0-10, $, Some, No, Burger, No, Yes, No] -> Yes

Yes

[Yes, Yes, 10-30, $, Full, Yes, Thai, Yes, No, No] -> Yes

Yes

[Yes, No, >60, $$$, Full, Yes, French, No, No, Yes] -> No

No

[No, Yes, 0-10, $$, Some, No, Italian, Yes, Yes, Yes] -> Yes

Yes

[No, No, 0-10, $, None, No, Burger, Yes, Yes, No] -> No

No

[No, Yes, 0-10, $$, Some, No, Thai, Yes, No, Yes] -> Yes

Yes

[Yes, No, >60, $, Full, No, Burger, Yes, Yes, No] -> No

No

[Yes, Yes, 10-30, $$$, Full, Yes, Italian, No, Yes, Yes] -> No

No

[No, No, 0-10, $, None, No, Thai, No, No, No] -> No

No

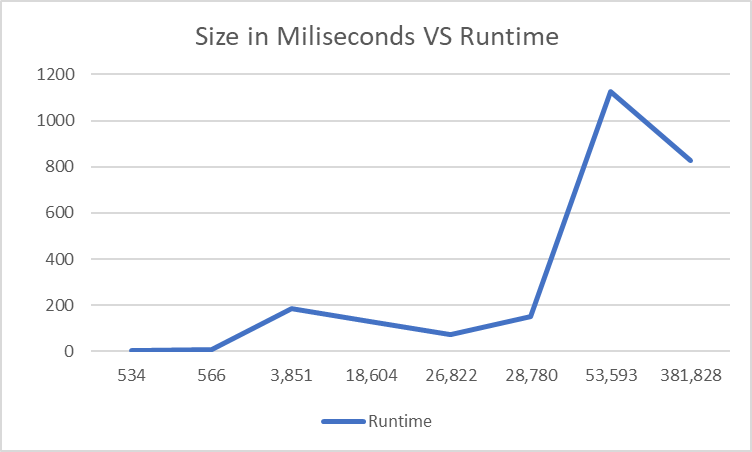
[Yes, Yes, 30-60, $, Full, Yes, Burger, No, Yes, No] -> Yes Yes

correct: 12/12 (100.00)%

**Analysis:**

**Graph 1:**

This Graph is the size of data in bites VS Runtime in milliseconds. As we can see the runtime increases as the size of the file increases. However the runtime for the poker data set is actually faster than the mushroom data set even though the poker data is larger. A reason this might happen is the decision tree for poker may be better than the mushroom tree. This shows the importance of creating efficient trees.



**Graph 2:**

This graph displays the runtime verses the size of the domain of the data. The domain is the number of variables in the data. We can see that the runtime of the program increases as the number of variables increases. This may be because as the number of variables in the data increases, the size of the tree will also increase. This makes the tree harder to search and it takes more time to find a solution to the data. This data increases very rapidly as the domain increases, and slightly resembles an exponential relationship.

**Extra Credit:**

**Linear Classifiers:**

The second algorithm we implemented was the linear classifiers. We used all of the classes that Professor Ferguson gave us, which include; Example, LearningRateSchedule, LinearClassifier, ClassifierDisplay, XYPlotCanvas, Data, and PerceptronClassifierTest. Linear classifiers are used to demonstrate classification a regression. Linear Classifiers take in data and with that data they classify the data into two classes. The decision boundary separates the two classes. With linear classification a line represents the boundary, hence the name. After the decision boundary is created, weights are created to minimize the loss. To do this we used the perceptron learning rule. If the perceptron rule is working correctly, and the data converges, then as the number of updated weights increase the accuracy of the classifier will increase. With our linear classifier, we also printed out a graph of proportions correct vs the number of weight updates. The graph we got matches ones that were in the book and in lecture.

Examples:

# of Weight Proportion

Updates Correct

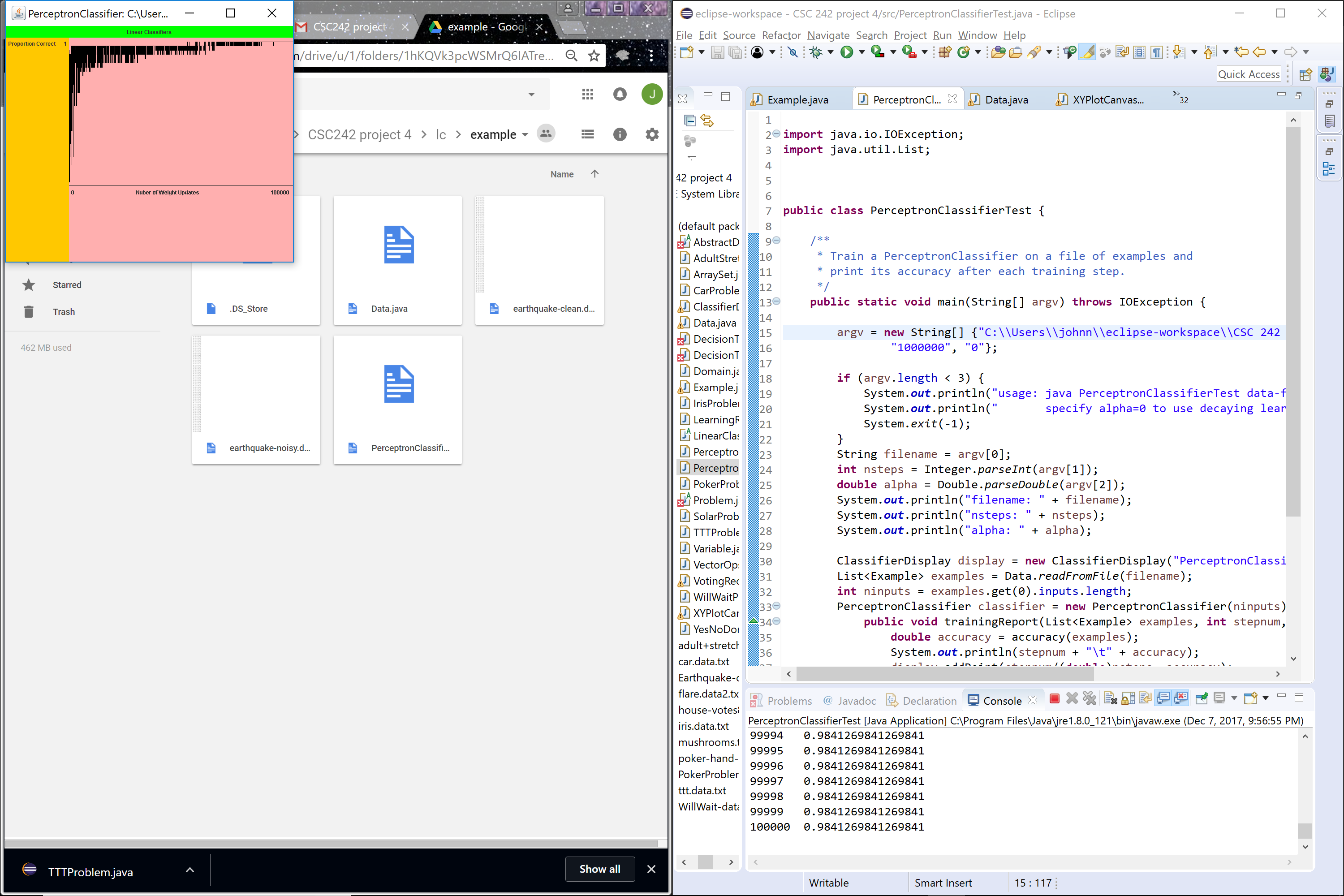
1 0.4603174603174603

100 0.5396825396825397

1000 0.9682539682539683

10000 0.9682539682539683

100000 0.9841269841269841



**Improvements:**

Our Decision Tree algorithm works pretty well but there are some other aspects that we could implement to improve the overall quality of our code we could modify our work to work for all types of data sets. Our code did not work for all types of data. Finding a way to get that to work would vastly improve our code. Additionally, if we had time we definitely would have implemented neural networks.