Muhammad Ahmad

Learning

4/26/2018

For this project, I have implemented 2 learning algorithms: Decision tree and linear classifiers (perceptron and logistical). Most of the code is commented. Almost all of Ferguson's code was used. I mainly updated the methods in different classes. How to run the program is described in the ReadMe.

I will use this writeup to explain both forms of machine learning, my implementation and results.

Decision Tree:

"A decision tree represents a function DECISION TREE that takes as input a vector of attribute values and returns a "decision"—a single output value. The input and output values can be discrete or continuous." -AIMA.

For this project both the examples I did had discrete variables.

"A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the input attributes, Ai, and the branches from the node are labeled with the possible values of the attribute, A(i) =vk(i). Each leaf node in the tree specifies a value to be returned by the function." – AIMA

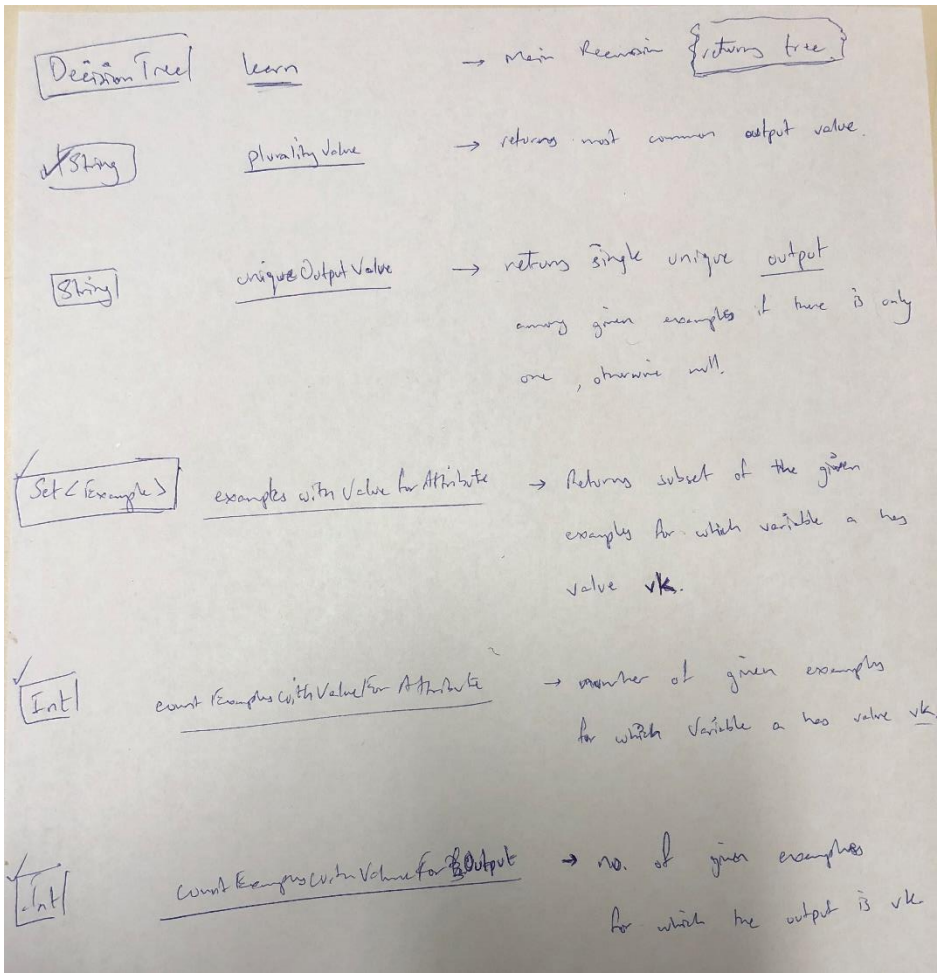
For this implementation of the decision tree algorithm I used AIMA's pseudo code:

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns a tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value v_k of A do
          exs ← {e : e ∈ examples and e.A = v_k}
          subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
      return tree
```

Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

To complement this algorithm, I had to add in a few new methods (with the help of Prof. Ferguson's comments):



Above is the picture of the methods I included. The first column represents the return type, second column is the name of the method and the last column describes the exact method.

My code looks like:

```
protected DecisionTree learn(Set<Example> examples, List<Variable> attributes, Set<Example> parent_examples) {
    if (examples.isEmpty()) {
        return new DecisionTree(pluralityValue(parent_examples));
    }
            int count = 0;
            // chooses the first examples value
            Iterator<Example> iterator = examples.iterator();
            String temp_classification = iterator.next().getOutputValue();
            count++;
            // compare it with all the others
            while(iterator.hasNext()) {
                if(iterator.next().getOutputValue().equals(temp_classification)) {
                    count++;
                }
            }
    if(count == examples.size()) {
        return new DecisionTree(temp_classification);
    }
    else if(attributes.isEmpty()) {
        return new DecisionTree(pluralityValue(examples));
    }
    else {
        Variable A = mostImportantVariable(attributes, examples);
        DecisionTree tree = new DecisionTree(A);

        for(String vk : A.getDomain()) {
            Set<Example> exs = examplesWithValueForAttribute(examples, A, vk);
            //List<Variable> new_attributes = new ArrayList<Variable>(attributes);

            attributes.remove(A);
            //Set<Example> new_examples = new ArraySet<Example>(examples);
            DecisionTree subtree = learn(exs, attributes, examples);
            //
            //
            tree.children.add(subtree);
        }
        return tree;
    }
}
```

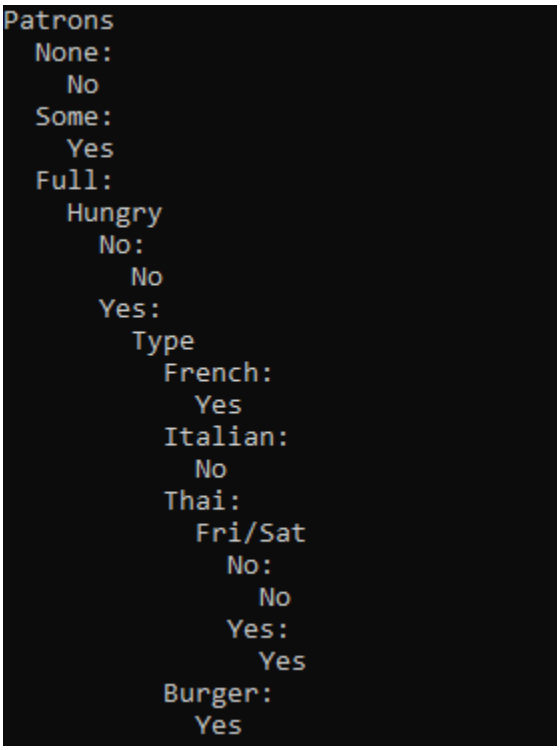
Explanation of the above code:

The code follows the pseudo code provided by the book. It is a recursive method. The first case is to return the parent if the examples are empty. If all examples fall under one category, we just make that one branch and put all the examples in that category. If we have exhausted all the attributes then we return a tree with the PV(examples). Otherwise we call the method again, making a subtree to our original tree with the most important variable (calculated using Prof. Ferguson's code). With all the examples who have the same value of the most important

attribute we put them in subtree and call the recursive method. We do this for all the vk values of the attribute A (the most important one).

Results:

After using the training data for the WillWait Problem, the program generated a decision tree which looks as follows:



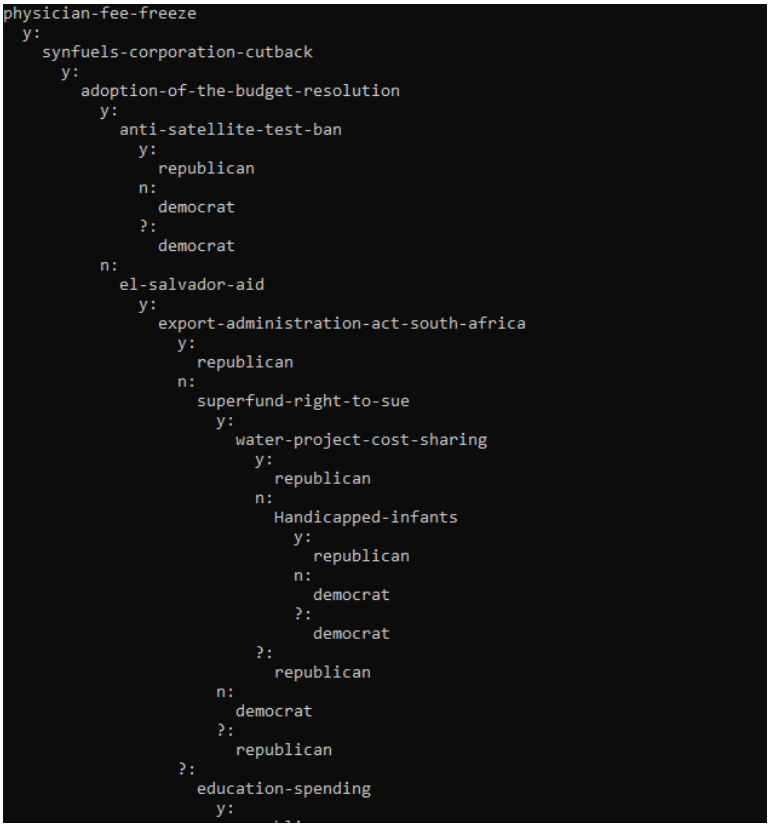
Upon testing the above tree with the same examples, the solution always came out to be correct i.e it was correct 12/12 times.

```
[No, 0-10, Some, Yes, Yes, $$$, No, No, French, Yes] -> Yes     Yes
[No, 30-60, Full, Yes, Yes, $, No, No, Thai, No] -> No  No
[Yes, 0-10, Some, No, No, $, No, No, Burger, No] -> Yes Yes
[No, 10-30, Full, Yes, Yes, $, Yes, Yes, Thai, No] -> Yes       Yes
[No, >60, Full, Yes, No, $$$, Yes, No, French, Yes] -> No       No
[Yes, 0-10, Some, No, Yes, $$, No, Yes, Italian, Yes] -> Yes    Yes
[Yes, 0-10, None, No, No, $, No, Yes, Burger, No] -> No No
[No, 0-10, Some, No, Yes, $$, No, Yes, Thai, Yes] -> Yes        Yes
[Yes, >60, Full, No, No, $, Yes, Yes, Burger, No] -> No No
[Yes, 10-30, Full, Yes, Yes, $$$, Yes, No, Italian, Yes] -> No  No
[No, 0-10, None, No, No, $, No, No, Thai, No] -> No     No
[Yes, 30-60, Full, Yes, Yes, $, Yes, No, Burger, No] -> Yes     Yes
correct: 12/12 (100.00)%
```

This shows the programs accuracy in generating and testing the decision tree algorithm.

After this I tried to run my program with the house votes problems, which had way more variables and the tree was going to be more complex than it already was.

House vote problem tree looks something like this:



```
physician-fee-freeze
  y:
    synfuels-corporation-cutback
      y:
        adoption-of-the-budget-resolution
          y:
            anti-satellite-test-ban
              y:
                republican
              n:
                democrat
              ?:
                democrat
          n:
            el-salvador-aid
              y:
                export-administration-act-south-africa
                  y:
                    republican
                  n:
                    superfund-right-to-sue
                      y:
                        water-project-cost-sharing
                          y:
                            republican
                          n:
                            Handicapped-infants
                              y:
                                republican
                              n:
                                democrat
                              ?:
                                democrat
                          ?:
                            republican
                    n:
                      democrat
                    ?:
                      republican
                ?:
                  education-spending
                    y:
```

```
                    y:
                      republican
                    n:
                      democrat
                    ?:
                      republican
              n:
                democrat
              ?:
                republican
          ?:
            democrat
      n:
        duty-free-exports
          y:
            immigration
              y:
                republican
              n:
                aid-to-nicaraguan-contras
                  y:
                    republican
                  n:
                    mx-missile
                      y:
                        democrat
                      n:
                        religious-groups-in-schools
                          y:
                            crime
                              y:
                                republican
                              n:
                                republican
                              ?:
                                republican
                          n:
                            republican
                          ?:
                            republican
                    ?:
                      democrat
                ?:
                  republican
```

After testing the data on the examples were given to generate the tree, it came out to be correct 98.39% of the time i.e out of the 435 examples 428 of them were correct. In order to achieve 100% we can possibly provide more training data as with the number of attributes this problem had, we need a big training data to ensure that the results are 100% correct. Also for some of the examples we did not have an explicit yes or no answer, instead it was just a '?'. And this increased the domain of each attribute making it hard for the program to generate a tree that is 100% of the time correct.

```
[y, y, y, n, y, y, y, y, n, y, y, n, n, y, n, n] -> democrat    democrat
[y, y, y, n, y, y, y, y, ?, ?, y, n, n, y, y, n] -> democrat    democrat
[y, y, y, n, y, n, n, y, n, n, n, n, ?, y, y, n] -> democrat    democrat
[y, n, y, n, y, n, y, y, n, n, n, y, n, y, y, n] -> democrat    democrat
[n, n, y, y, y, y, n, n, y, y, n, n, y, y, y, y] -> republican  republican
[?, n, y, n, y, n, y, y, y, n, ?, ?, n, y, y, n] -> democrat    democrat
[y, n, y, n, y, ?, y, y, n, n, n, y, ?, y, y, y] -> democrat    democrat
[y, n, n, y, n, y, y, n, y, y, n, n, y, y, y, y] -> republican  republican
[y, n, y, n, y, n, y, n, n, n, n, n, n, y, y, n] -> democrat    democrat
[n, y, n, y, n, y, n, n, y, y, ?, n, y, y, n, y] -> republican  republican
[n, n, ?, y, ?, y, ?, n, y, y, n, n, y, y, ?, y] -> republican  republican
[n, n, n, y, n, y, n, ?, y, y, y, n, y, n, y, y] -> republican  republican
correct: 428/435 (98.39)%
C:\Users\muham\Desktop\CSC_242\Project_4\Project_4_extra_dt\src>
```

//Done for extra credit

Linear classifiers (perceptron and logistical):

My results for the classifiers match exactly to the results in the book AIMA fig 18.16 and 18.18

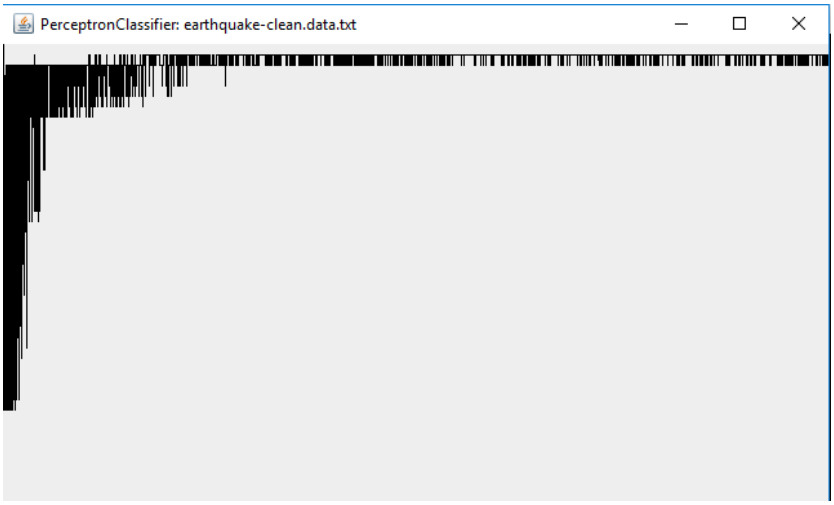
RESULTS: (EXPLANATION AFTER RESULTS)

Perceptron Classifier:

Earthquake clean

Steps: 1000

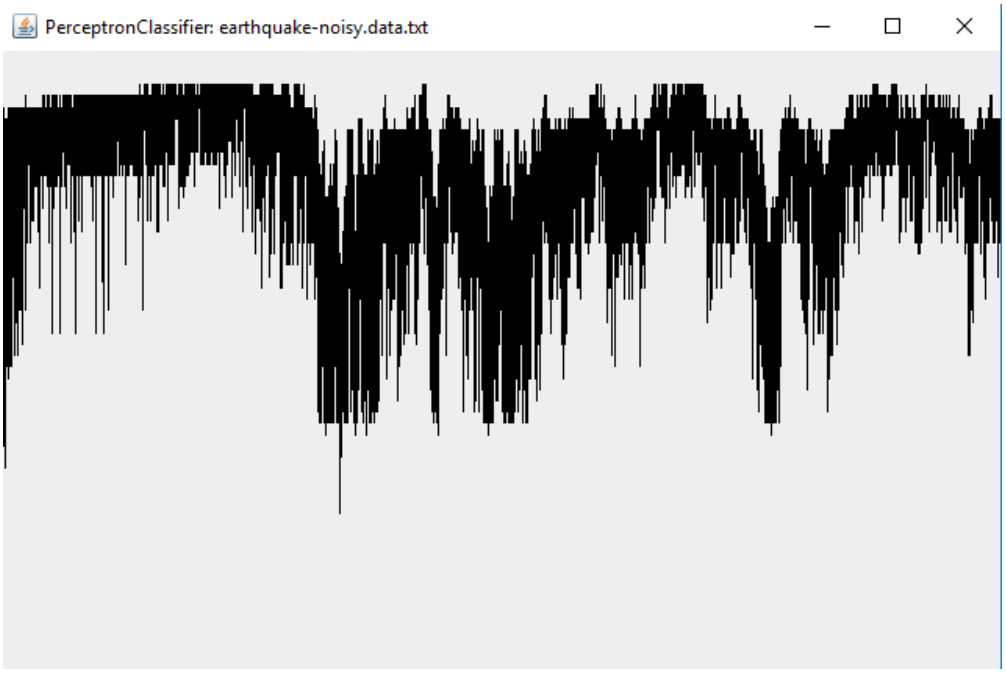
Alpha: 0



Earthquake noisy

Steps: 1000

Alpha: 0

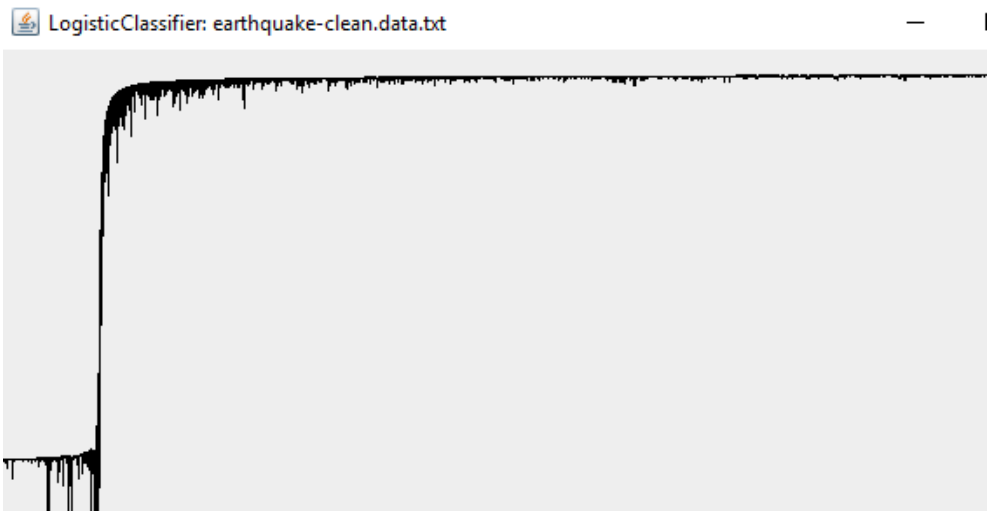


Logistic Classifier:

Earthquake clean

Steps: 1000

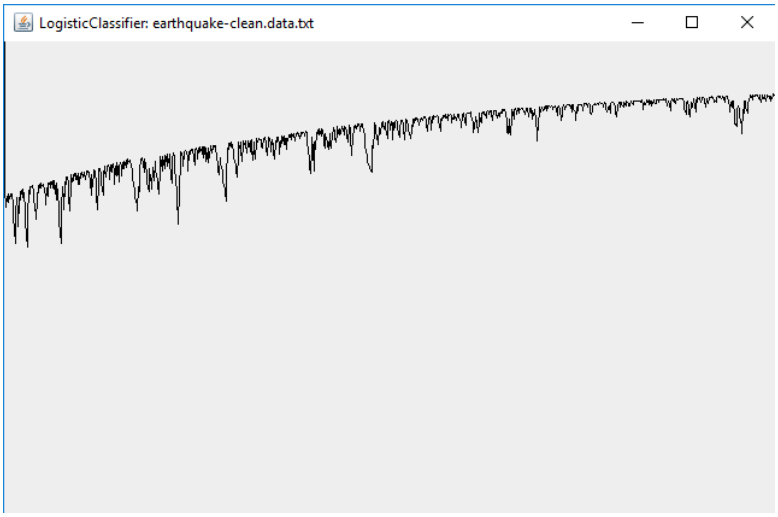
Alpha: 0



Earthquake clean

Steps: 1000

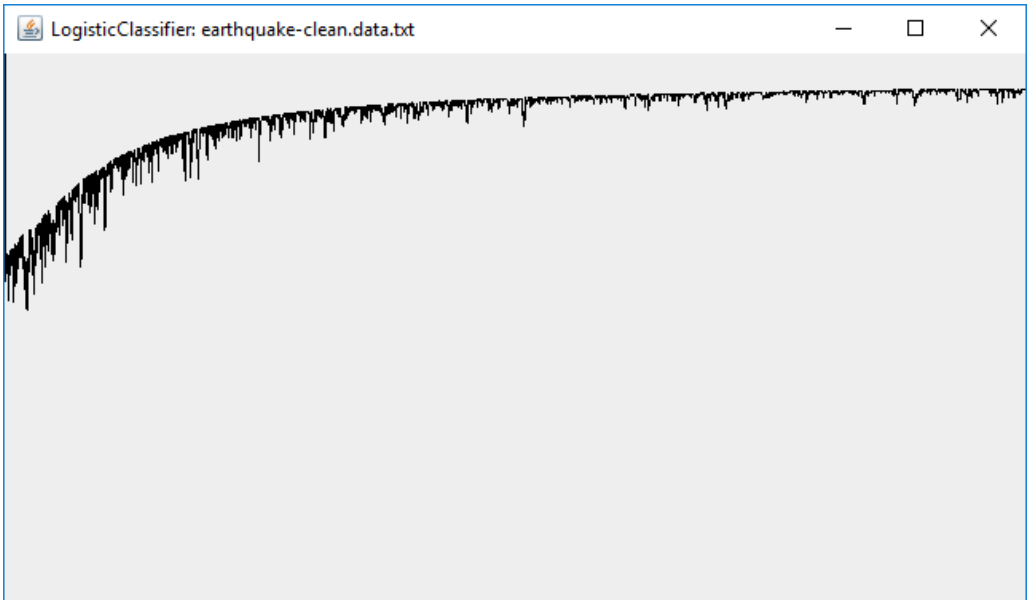
Alpha: 0.05



Earthquake clean

Steps: 5000

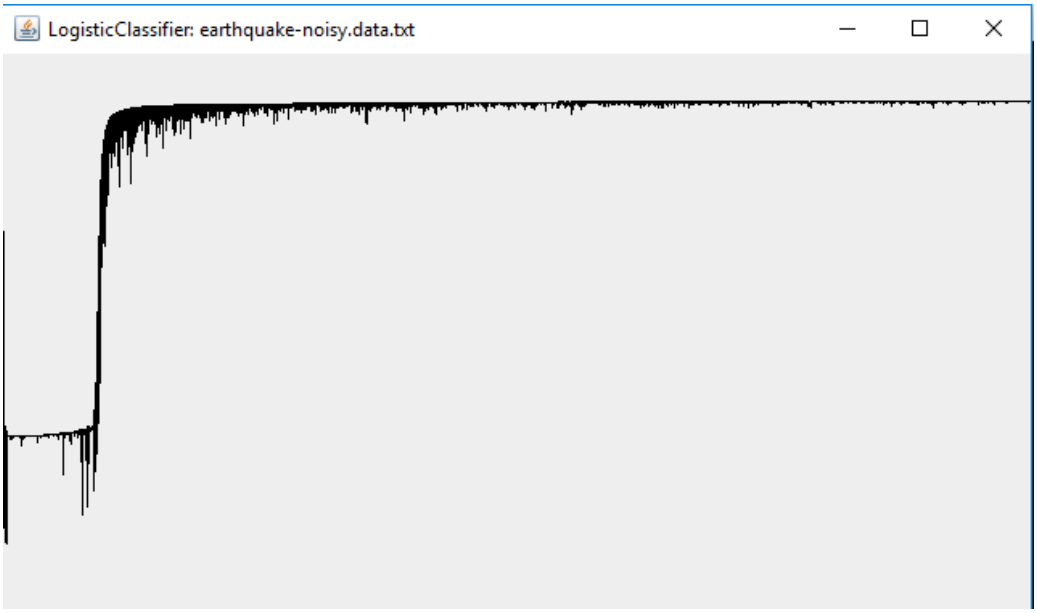
Alpha: 0



Earthquake noisy

Steps: 1000

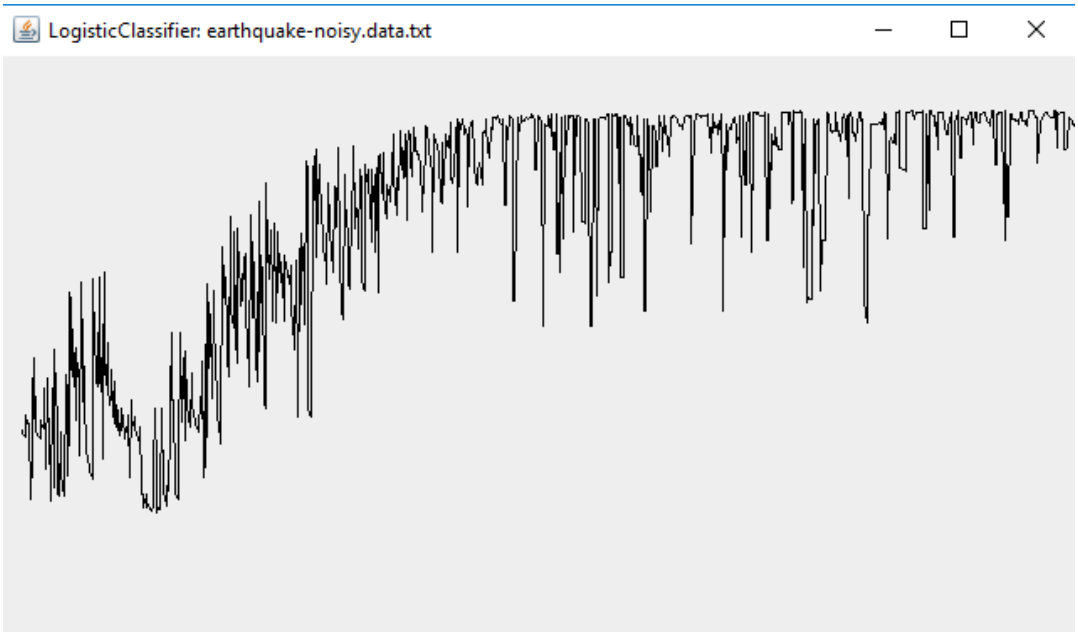
Alpha: 0



Earthquake noisy

Steps: 1000

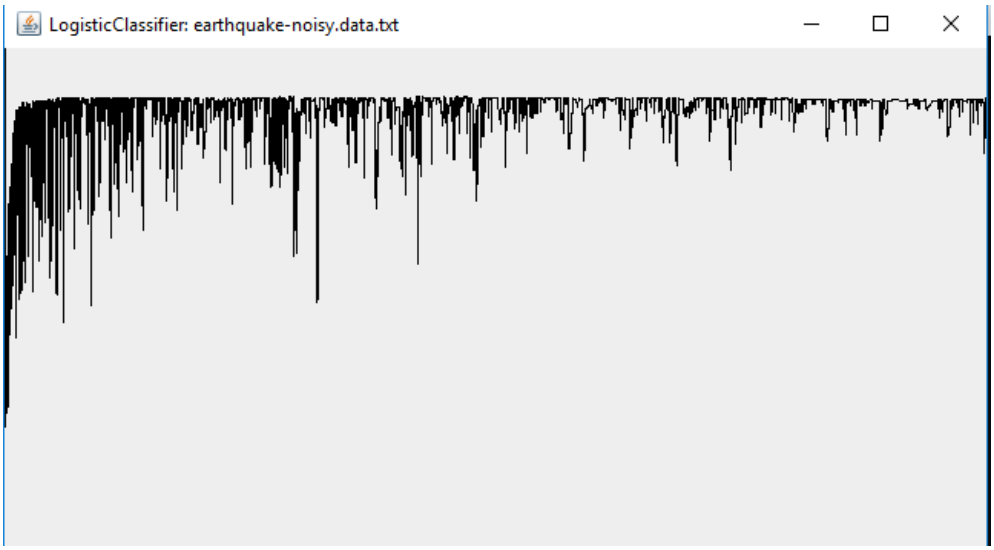
Alpha: 0.5

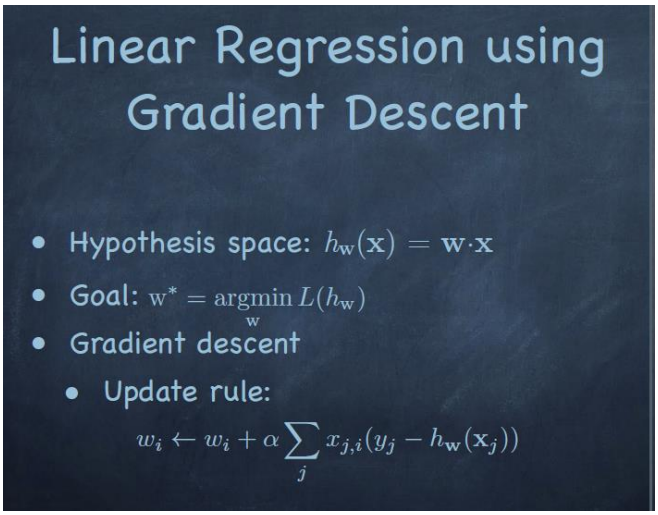


Earthquake noisy

Steps: 10000

Alpha: 0.5





For the perceptron classifier the update rule was used and for the logistical one similar update rule was used but with a logistical function instead of the threshold function. With the soft threshold (logistical) the program trained itself quicker and came to normal, as the loss was not as much as the perceptron classifier. This is because the perceptron classifier uses hard threshold i.e it is 1 if equal to or greater than 0, otherwise 0. With the hard threshold some data points would not exactly fit, which may lead to some discrepancies. If alpha is not 0, it decreases over time with 1000/(1000+t) – this was already done by Professor Ferguson in his code. An important point to note is that as we increase the number of steps, the graph smooths out as illustrated and explained by fig 18.16 in AIMA. With the same scale and more iterations, the graph gets straight, it also depends on the alpha the user chooses.

The update method for the perceptron classifier with hard threshold:

```java
/**
 * A PerceptronClassifier uses the perceptron learning rule
 * (AIMA Eq. 18.7): w_i \leftarrow w_i+\alpha(y-h_w(x)) \times x_i
 */
public void update(double[] x, double y, double alpha) {
    // Must be implemented by you
    for(int i = 0; i<x.length; i++) {
        double sum = 0.0;
        for(int j = 0; j<x.length; j++) {
            sum += x[j]*weights[j];
        }
        weights[i] = weights[i] + alpha*(y - threshold(sum))*x[i];

    }
}

/**
 * A PerceptronClassifier uses a hard 0/1 threshold.
 */
public double threshold(double z) {
    // Must be implemented by you
    if(z >= 0) {
        return 1;
    }
    else {
        return 0;
    }
}
```

The update method for the logistic classifier with soft threshold:

```java
/**
 * A LogisticClassifier uses the logistic update rule
 * (AIMA Eq. 18.8): w_i \leftarrow w_i+\alpha(y-h_w(x)) \times h_w(x)(1-h_w(x)) \times x_i
 */
public void update(double[] x, double y, double alpha) {
    // Must be implemented by you
    for(int i = 0; i<x.length; i++) {
        double sum = 0.0;
        for(int j = 0; j<x.length; j++) {
            sum += x[j]*weights[j];
        }
        weights[i] = weights[i] + (alpha*(y - threshold(sum))*(threshold(sum))*(1 - threshold(sum))*x[i]);
    }
}

/**
 * A LogisticClassifier uses a 0/1 sigmoid threshold at z=0.
 */
public double threshold(double z) {
    // Must be implemented by you
    z = 1.0/(1.0 + Math.exp(-z));
    return z;
}
```

THANK YOU!

References:

The AIMA book was used for the project and the writeup

Professor Fergusons slides were used for this writeup (Particularly Lecture 3.3 and Lecture 3.4)