Analysis of the Perceptron and the LMS Neurons

Matthew A. Letter

October 7th 2014

Neural Networks 547

Perceptron

The perceptron is an algorithm for supervised classification of an input into one of several possible non-binary outputs. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.[1]

Java was used to code the perceptron and the jmathplot java API was used to plot the data into a presentable format automatically at the end of each test. The perceptron was given a set of learning data from two linearly separable classes. The testing data was made up of two non-linearly separable classes. The stop criterion for learning was when an epoch occurred without encountering any errors. The RMS (root mean squared) error calculated for the testing class was 0.9596462478022414 and the learning rate was (learning rate/ 1000). Below are three plots: before learning with learning data (Figure 1), after learning with learning data (Figure 2), and one with the PDR testing data (Figure 3).

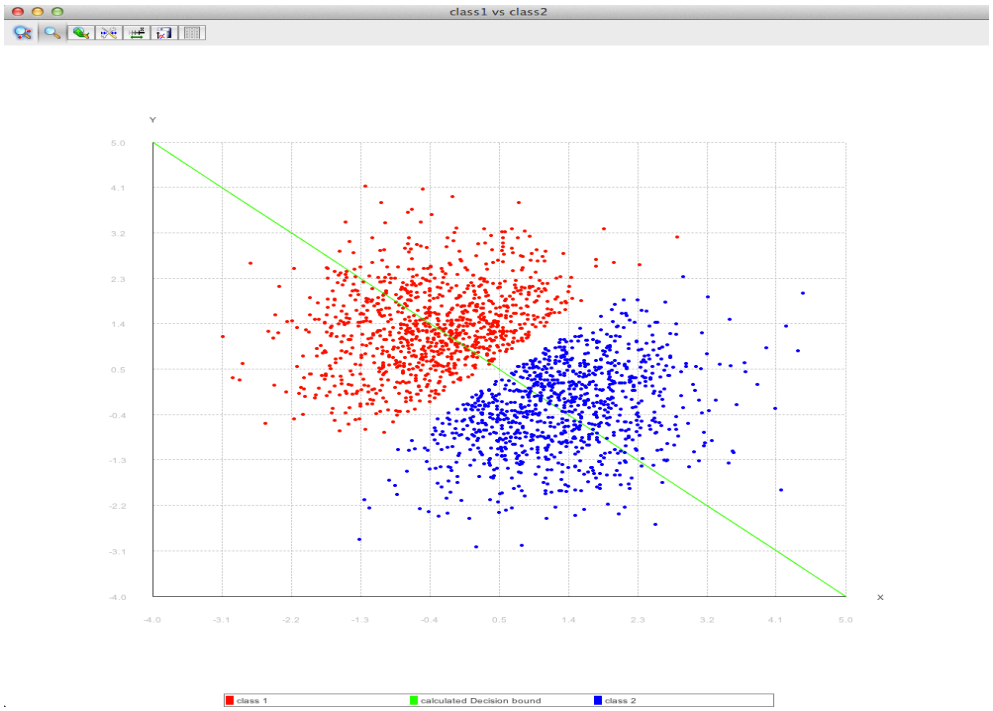

Figure 1: RMS error before learning with learning data.

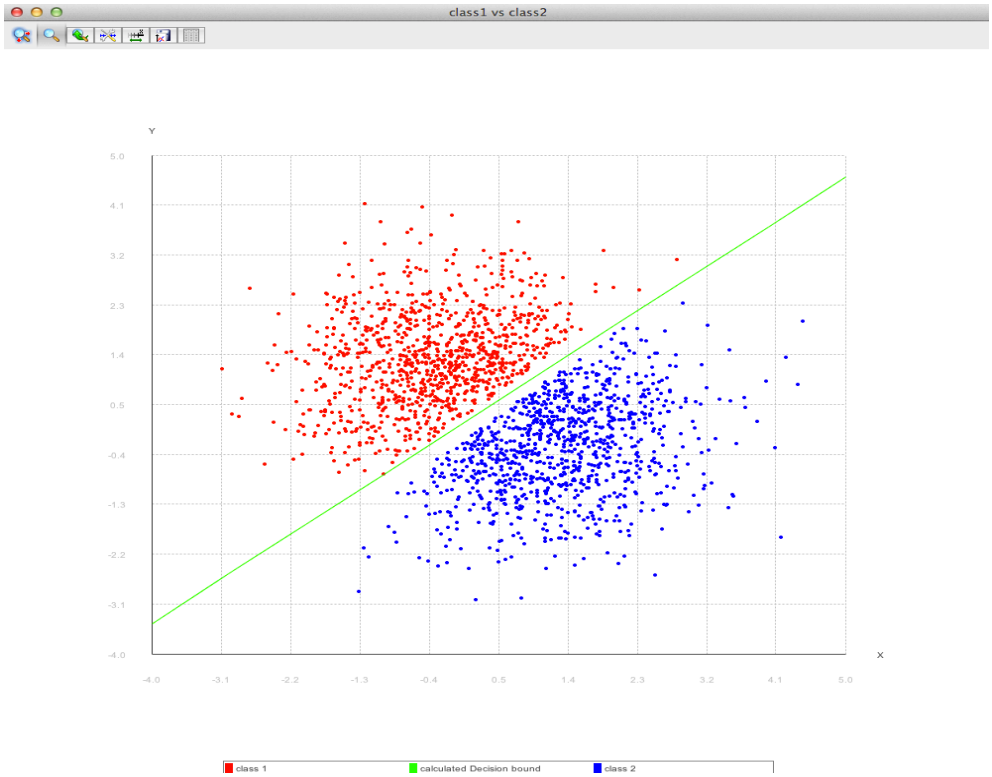

Figure 2: RMS error after learning with learning data.

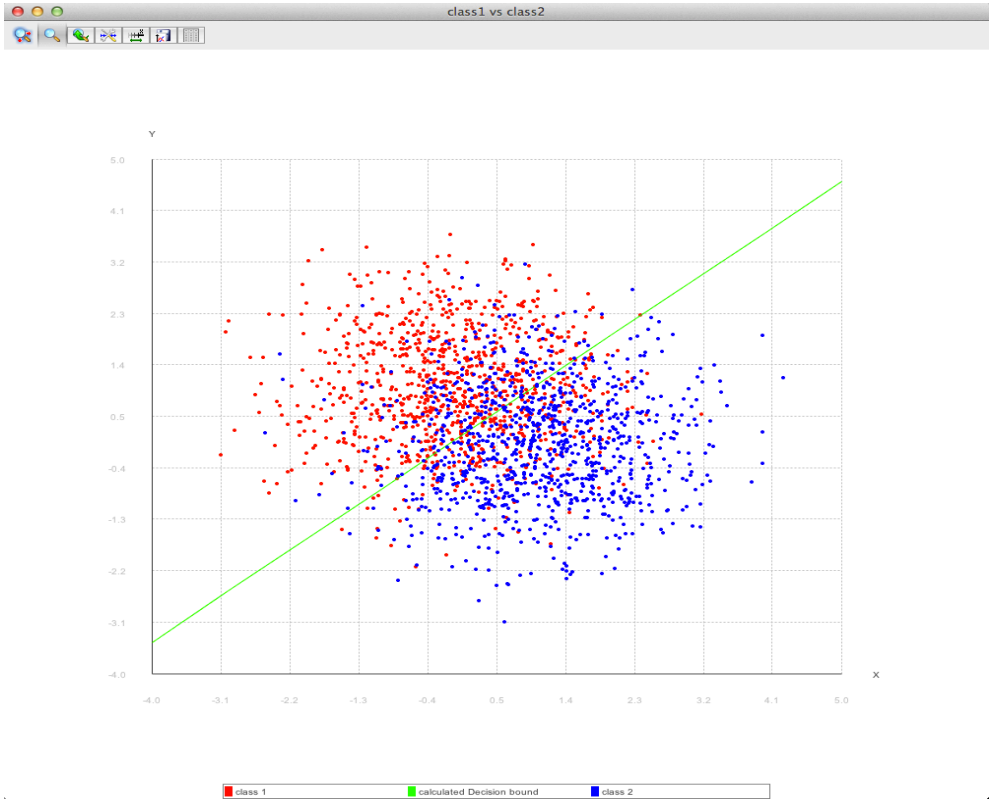

Figure 3: RMS error with PDR testing data.

Below are Tables 1-3 with changing starting values on weights, learning rate, and shuffling of the data respectively.

W0	W1	W2	# Of Epochs	Learning rate
-100	-100	-100	17191	0.25
-1	-1	-1	173	0.25
0	0	0	4	0.25
1	1	1	255	0.25
100	100	100	25460	0.25

Table 1: The learning rate is low to exaggerate the effects of the weight changes. The weights converge much faster when they are initially set to 0. All data is taken at the time error reached 0.

W0	W1	W2	# Of Epochs	Learning rate
100	100	100	6365	1
100	100	100	637	10
100	100	100	64	100

Table 2: The weights are set to 100 based off figure one to exaggerate changes. As the learning rate gets large the weight converge much faster. All data is taken at the time error reached 0.

W0	W1	W2	# Of Epochs	Learning rate	Was Shuffled
0	0	0	4	0.25	False
0	0	0	2	0.25	True

Table 3: Halved the number of epochs by shuffling the input learning data.

In Graphs 4 and 5, a generalization graph was produced for 100 neurons.

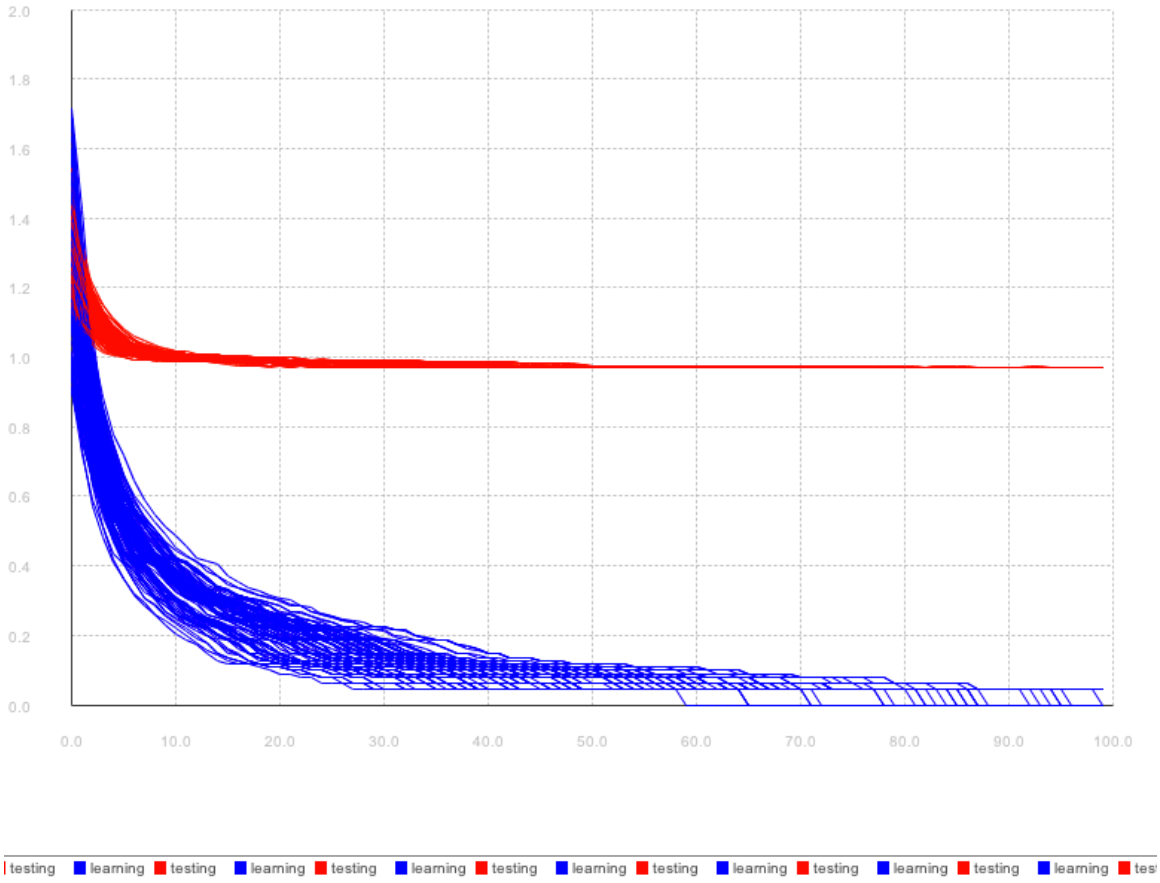

Figure 4: Y-axis is the RMS error, x-axis are the number of epochs. Red is the error for testing data and blue is the error for the learning data. Random weight sets were chosen for each neuron, leaving all other variables to a constant initially.

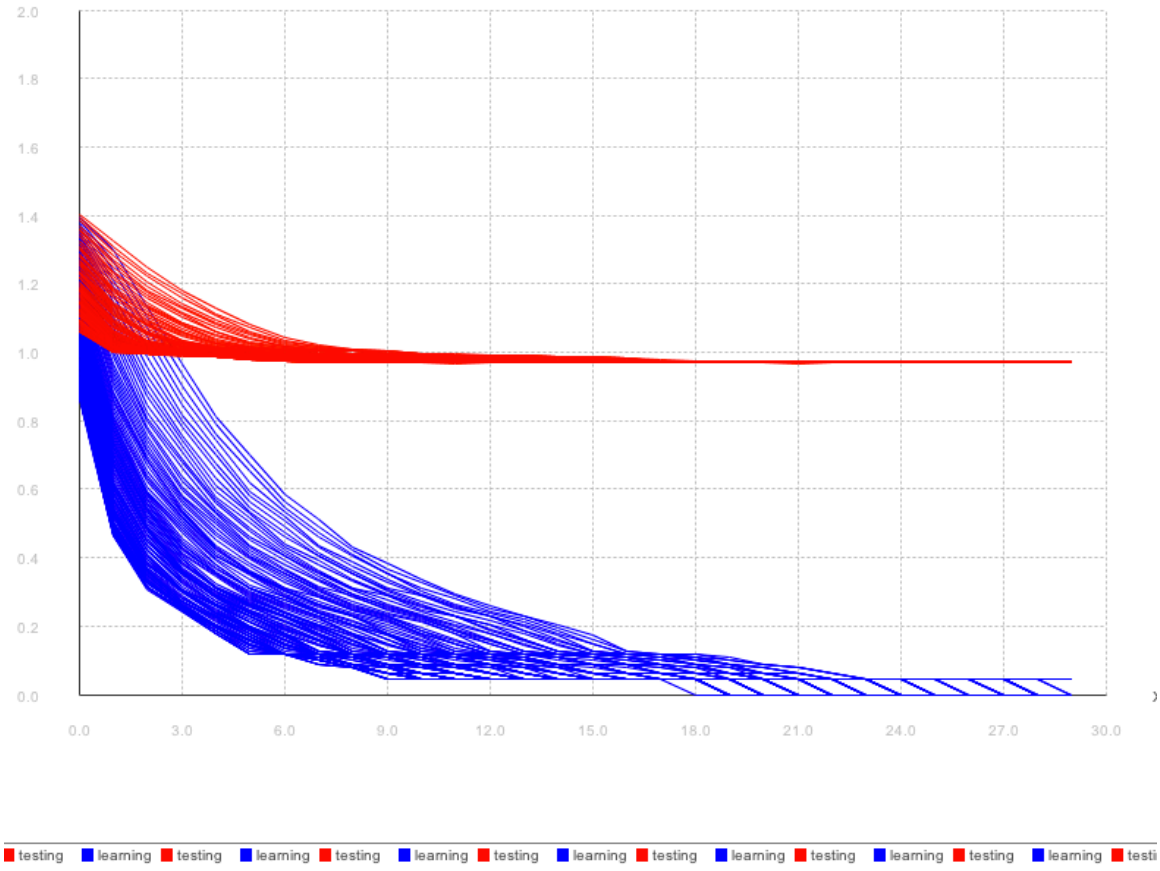

Figure 5: This increments the learning rate by 0.25 over 100 different runs. The lowest learning rate converges the slowest where as the highest learning rate (closest to y-axis) converges much faster.

Perceptron Results and Conclusions

There are many things to be gleaned from the above data. From the results presented above it is observed that a learning rate increase is positively correlated with the convergence rate. Starting the weights close to 0 makes the error rate converge faster and randomly shuffling the input learning data makes the error converge faster.

Based on the results, when choosing start parameters for a fast converging perceptron, a high learning rate with an initial weight set close to 0 along with random shuffling of the inputs is the correct way to make the perceptron's error rate to converge the fastest.

LMS

The delta rule is a gradient descent-learning rule for updating the weights of the inputs to artificial neurons in single-layer neural network. It is a special case of the more general backpropagation algorithm. The delta rule is derived by attempting to minimize the error in the output of the neural network through gradient descent. Therefore it is expected that the RMS error would approach an asymptote, and not 0 like the case of the perceptron.[2]

Java was used to code the LMS and the jmathplot java API (a common graphing java tool) was used to plot the data into a presentable format automatically at the end of each test. The LMS was given a set of learning data from two non-linearly separable classes. The testing data was made up of two non-linearly separable classes, as well. The stop criterion for learning was when an epoch occurred with an error difference from the previous of less then 0.0001. The RMS error calculated for the testing class was 0.9974943583775273 and the learning rate was (learning rate/ 1000). Below are three plots: LMS equation line before training with associated learning data (Figure 6), after learning with learning data (Figure 7), and one with the PDR testing data (Figure 8).

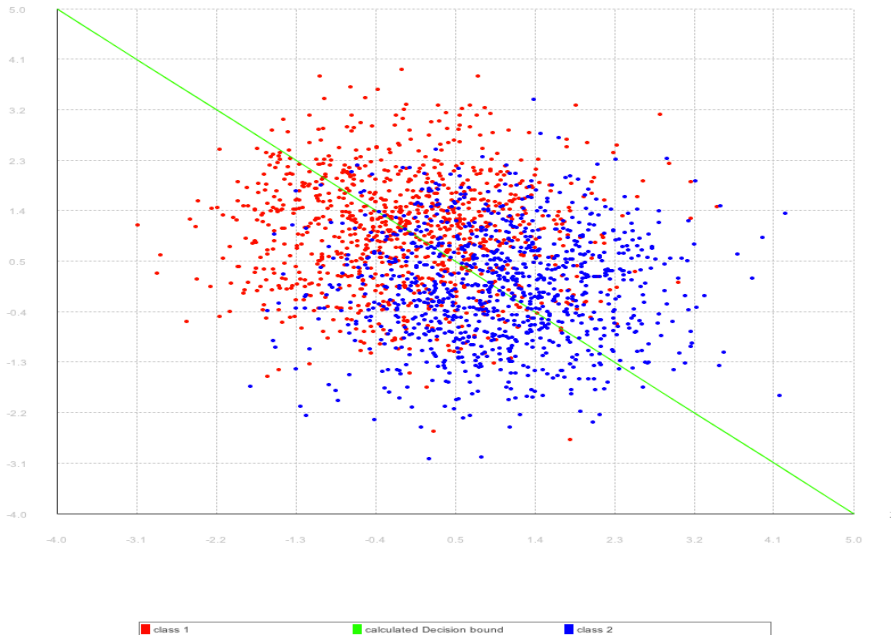

Figure 6: LMS equation line before training with associated learning data.

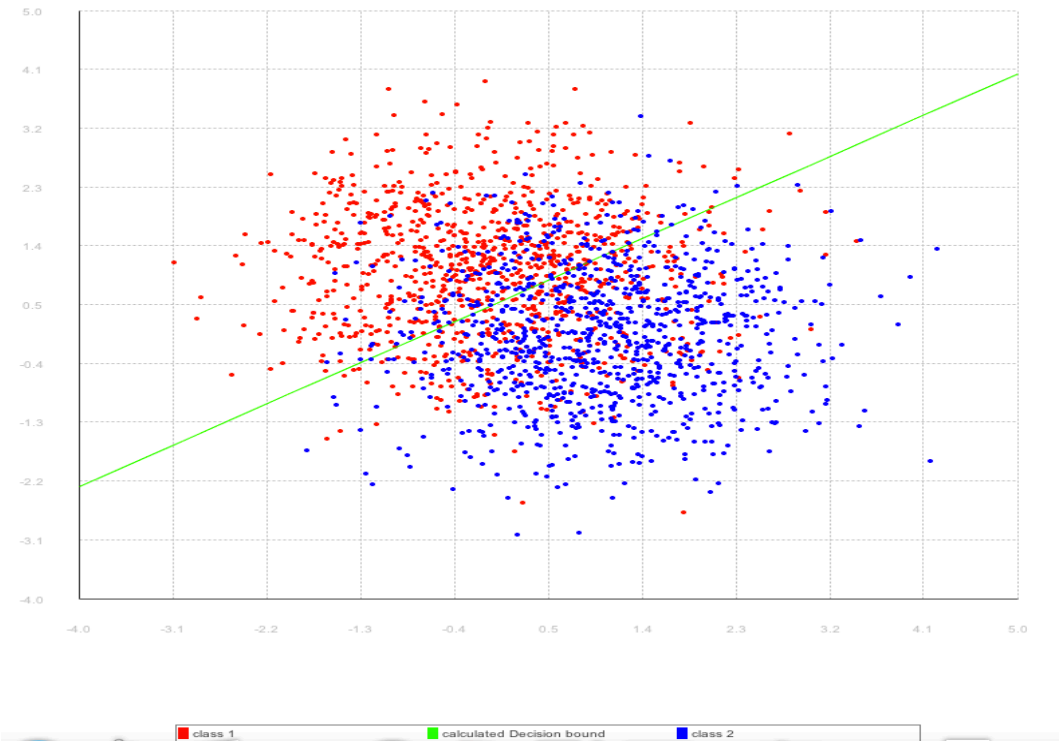

Figure 7: LMS equation line after training with associated learning data.

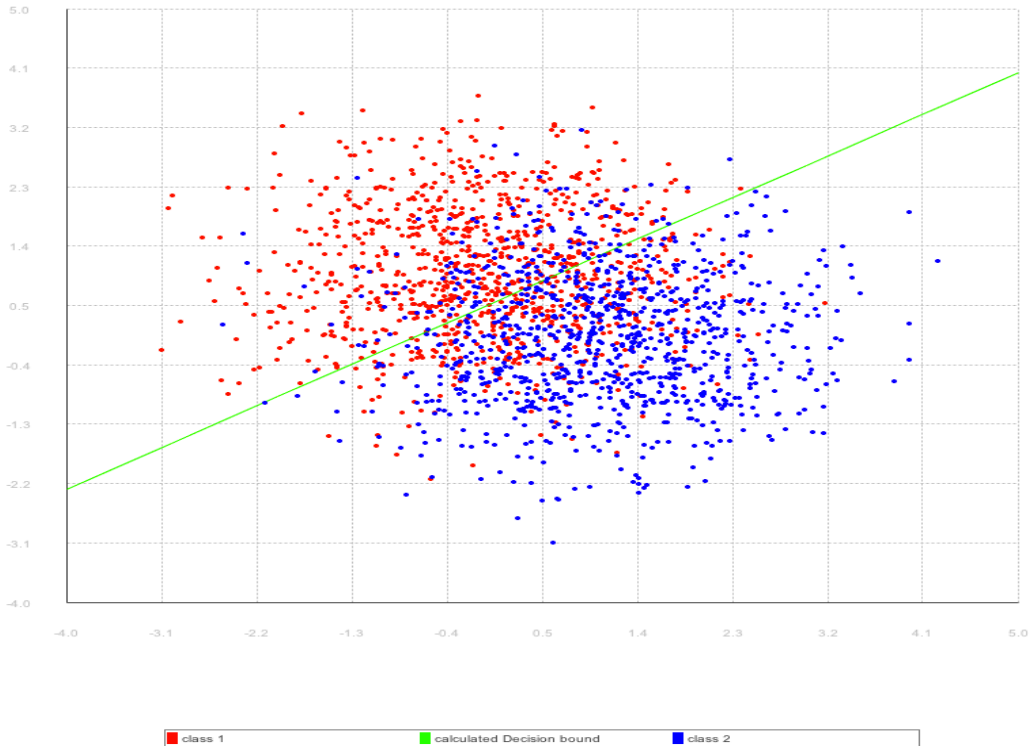

Figure 8: LMS equation line after training with testing data.

Tables 4-6 have varying stating weights (Table 4), learning rate (Table 5), and shuffling of the data (Table 6).

W0	W1	W2	# Of Epochs	Learning rate
-100	-100	-100	broken	0.25
-1	-1	-1	18	0.25
0	0	0	2	0.25
1	1	1	17	0.25
100	100	100	broken	0.25

Table 4: The weights converge much faster when they are initially set to 0 and they generally converge must faster than the perceptron. All data is taken when the error change was minimized to 0.0001. Large start weights appear to have broken the LMS neuron.

W0	W1	W2	# Of Epochs	Learning rate
1	1	1	17	.25
1	1	1	11	.5
1	1	1	9	1

Table 5: The weights are set to 1 (based on Table 4) to emphasize the changes. As the learning rate gets large the weight converges faster. All data is taken at the time the error reached 0.

W0	W1	W2	# Of Epochs	Learning rate	Was Shuffled
1	1	1	17	0.25	False
1	1	1	18	0.25	True

Table 6: Shuffling does not appear to change the number of epochs to reach the convergence criteria.

In Graphs 9 and 10, a generalization graph was produced for 100 neurons.

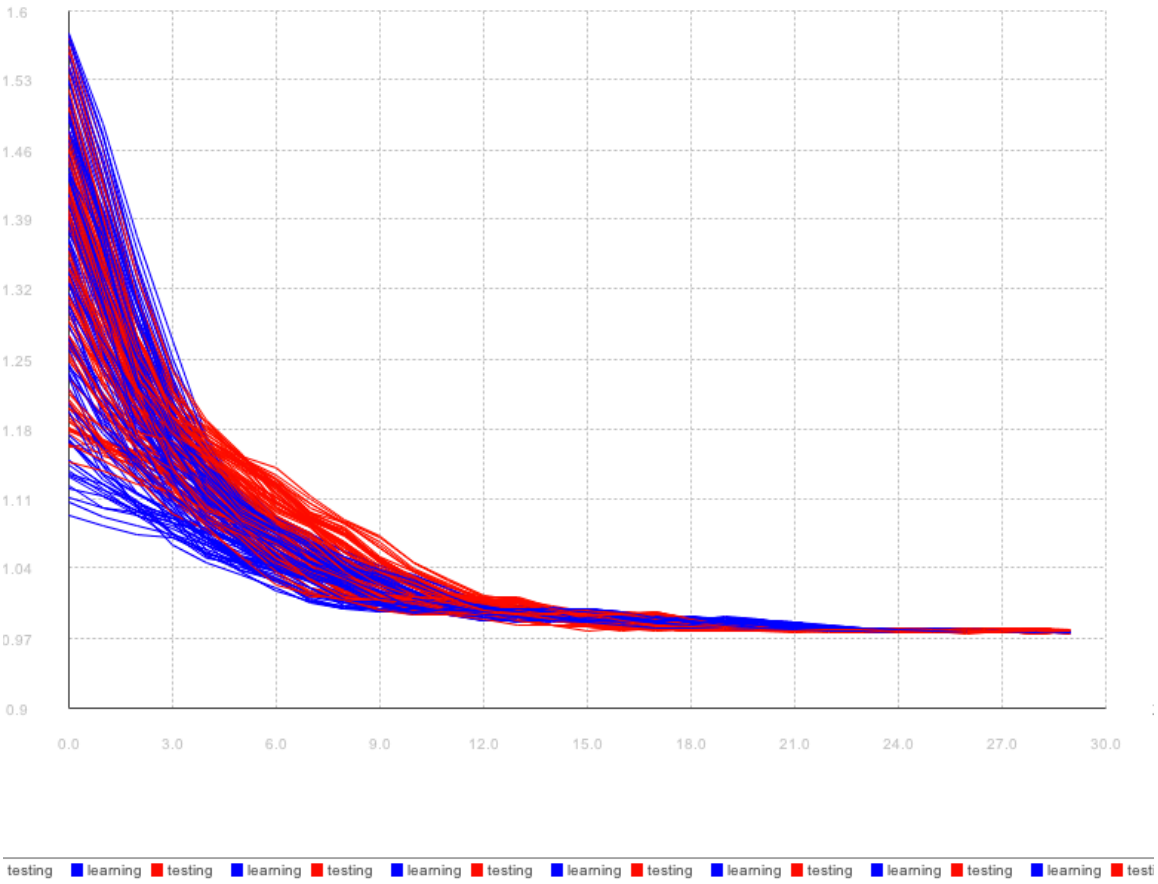

Figure 9: RMS error vs. number of epochs. The error for testing data (red) and the error for the learning data (blue) were taken after every epoch. The start weights were randomly chosen from 0-1.

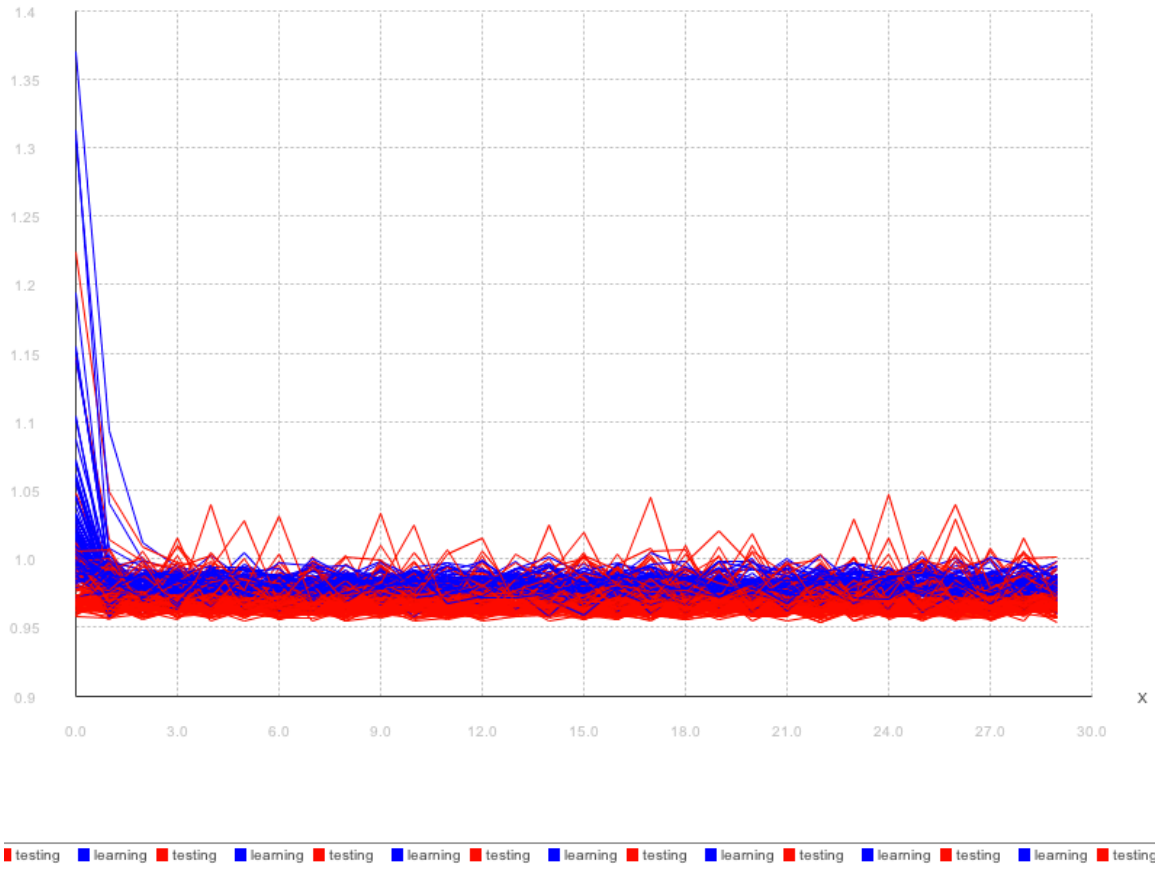

Figure 10: RMS error vs. number of epochs. The error for testing data (red) and the error for the learning data (blue) were taken after every epoch. The learning rate was incremented for each new LMS by 0.25 over 100 different neuron runs, all other variables stated with the same initial constants. The lowest learning rate converges the slowest and the highest learning rate (closest to y-axis (RMS axis)) converges much faster. The lower the learning rate, the less negative change when approaching the asymptote.

LMS Results and Conclusions

There are many things to be gleaned from the above data. From the LMS results presented above it is observed that learning rate increase positively correlates with the convergence rate increase but will cause the error to vary wildly once it has approached the asymptote. Starting the weights close to 0 makes the error rate converge faster and randomly shuffling the input learning data has very little effect.

Based on the results, the correct way to make the LMS rate reach converge the fastest is to choose starting parameters with a learning rate that is high enough to converge fast yet low enough to remain close to the asymptote and initial weight set close to 0.

Citations

1. "Perceptron." *Wikipedia*. Wikimedia Foundation, 30 Sept. 2014. Web. 03 Oct. 2014. <http://en.wikipedia.org/wiki/Perceptron>.

2. "Delta Rule." *Wikipedia*. Wikimedia Foundation, 24 Aug. 2014. Web. 05 Oct. 2014. <http://en.wikipedia.org/wiki/Delta_rule>.

3. Kanasz, Robert. "Single Layer Perceptron as Linear Classifier." - *CodeProject*. Robert Kanasz, n.d. Web. 06 Oct. 2014. <http://www.codeproject.com/Articles/125346/Single-Layer-Perceptron-as-Linear-Classifier>.

Source code Perceptron

```
import java.awt.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
import org.math.plot.*;

import javax.swing.*;

/**
 * Created by matthewletter on 9/30/14.
 * this class is designed to represent the single layer perceptron
 */
public class Perceptron {
    public Random rnd;
    public double w0;
    public double w1;
    public double w2;
    public double learningRate;
    private double x0;
    private int maxIterations;


    /**
     * this constructor build our random weights and sets learning rate and learning
iterations
     */
    Perceptron(){
        this.rnd = new Random(System.currentTimeMillis());
//        this.w0 = rnd.nextDouble();
//        this.w1 = rnd.nextDouble();
//        this.w2 = rnd.nextDouble();
        this.w0 = 0;
        this.w1 = 0;
        this.w2 = 0;
        this.learningRate = .25;
```

```
        this.x0 = -1;
        this.maxIterations = 300;

    }

    /**
     * this class is used to teach the perceptron its weights, when there are no
longer errors
     * or there are no longer iterations its stops learning
     * @param samples ArrayList of samples
     */
    public double learn(ArrayList<Sample> samples) {
        int iterations = 0;
        boolean error = true;
        double errorVal=0;
        double errorSumSqr;
        double n = samples.size();

        double alpha =  (learningRate / 1000);

        //go through the epoche's
        while (error && iterations < maxIterations) {
            errorVal = 0;
            errorSumSqr = 0;
            error = false;
            //Collections.shuffle(samples);
            //iterate through the epoche
            for(Sample sample : samples) {
                double x1 = sample.X1;
                double x2 = sample.X2;
                int y;

                if (((w1 * x1) + (w2 * x2) - w0) < 0) {
                    y = -1;
                } else {
                    y = 1;
                }

                if (y != sample.expectedClass) {
                    error = true;
                    errorSumSqr += (sample.expectedClass - y)*(sample.expectedClass -
y);
                    w0 = w0 + alpha * (sample.expectedClass - y) * x0 / 2;
                    w1 = w1 + alpha * (sample.expectedClass - y) * x1 / 2;
                    w2 = w2 + alpha * (sample.expectedClass - y) * x2 / 2;
                }
```

```
            //System.out.println("w0: "+w0+" w1: "+w1+" w2: "+w2 +"\n");
        }
        iterations++;
        errorVal = Math.sqrt(errorSumSqr/n);

        System.out.println("epoche: " + iterations + " |\n w0: "+w0+" w1: "+w1+"
w2: "+w2 +" RMS error: " +
                errorVal);
      }
      return errorVal;
   }
   public double test(ArrayList<Sample> samples) {
      int iterations = 0;
      boolean error = true;
      double errorVal;
      double errorSumSqr;
      double n = samples.size();

      //go through the epoche's
        errorVal = 0;
        errorSumSqr = 0;

        //iterate through the epoche
        for(Sample sample : samples) {
           double x1 = sample.X1;
           double x2 = sample.X2;
           int y;//output

           if (((w1 * x1) + (w2 * x2) - w0) < 0) {
              y = -1;
           } else {
              y = 1;
           }
           if (y != sample.expectedClass) {
              errorSumSqr += (sample.expectedClass - y)*(sample.expectedClass -
y);
           }
           //System.out.println("w0: "+w0+" w1: "+w1+" w2: "+w2 +"\n");
        }
        errorVal = Math.sqrt(errorSumSqr/n);

        System.out.println("Test: " + iterations + " |\n w0: "+w0+" w1: "+w1+"
w2: "+w2 +" RMS error: " +
                errorVal);
        return errorVal;
   }
```

```
    /**
     * used to plot all of our data points
     * @param cls1 class 1
     * @param cls2 class 2
     */
    public static void plotClasses(ArrayList<Sample> cls1, ArrayList<Sample>
cls2, Perceptron p){
        // define your data
        double[] x;
        double[] y;

        x = new double[cls1.size()];
        y = new double[cls1.size()];

        for (int i = 0; i < cls1.size(); i++) {
            x[i]=cls1.get(i).X1;
            y[i]=cls1.get(i).X2;
        }


        // create your PlotPanel (you can use it as a JPanel)
        Plot2DPanel plot = new Plot2DPanel();

        // define the legend position
        plot.addLegend("SOUTH");

        // add a line plot to the PlotPanel
        plot.addScatterPlot("class 1", Color.RED, x, y);


        double y0=0;
        double y1=1;
        double[] linex = {-4,-3,-2,-1,0,1,2,3,4,5};
        double[] liney = {-4,-3,-2,-1,0,1,2,3,4,5};
        for (int i = 0; i < linex.length; i++) {
            liney[i]= ((-p.w1/p.w2)*linex[i])+(p.w0/p.w2);
        }
        //add line plot weights
        plot.addLinePlot("calculated Decision bound",Color.GREEN,linex,liney);



        x = new double[cls2.size()];
        y = new double[cls2.size()];
```

```
        for (int i = 0; i < cls2.size(); i++) {
            x[i]=cls2.get(i).X1;
            y[i]=cls2.get(i).X2;
        }
        plot.addScatterPlot("class 2", Color.BLUE, x, y);

        // put the PlotPanel in a JFrame like a JPanel
        JFrame frame = new JFrame("class1 vs class2");
        frame.setSize(1000, 1000);
        frame.setContentPane(plot);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
    }

    /**
     * used to parse the provided text files
     * @param f file
     * @param classNumber +-1
     * @return ArrayList of Sample
     */
    public static ArrayList<Sample> parseFile(File f,int classNumber){
        Scanner scanner;
        String[] sA;
        String s;
        ArrayList<Sample> samples = new ArrayList<Sample>();
        try {
            scanner = new Scanner(f);
            s = scanner.nextLine();

            while(scanner.hasNext()){
                sA = s.split(" ");
                if(sA.length==3)
                samples.add(new Sample(Integer.parseInt(sA[0]),
Double.parseDouble(sA[1]), Double.parseDouble(sA[2]),
                        classNumber));
                s = scanner.nextLine();
            }
            scanner.close();

        }catch(FileNotFoundException e){
            e.printStackTrace();
        }
        return samples;
    }
```

```
  /**
   * sets up the ptron and plots data points
   * @param args not used
   */
  public static void main(String[] args){
    Perceptron p = new Perceptron();

    /*****************************/
    /*     Training section      */
    /*****************************/

    //class 1
    File f1 = new File("/Users/matthewletter/Documents/single-
perceptron/PercepClass1Training.txt");
    ArrayList<Sample> cls1 = parseFile(f1,1);

    //class 2
    File f2 = new File("/Users/matthewletter/Documents/single-
perceptron/PercepClass2Training.txt");
    ArrayList<Sample> cls2 = parseFile(f2,-1);

    //plotClasses(cls1, cls2, p);

    ArrayList<Sample> allLearningClasses = new ArrayList<Sample>();
    allLearningClasses.addAll(cls1);
    allLearningClasses.addAll(cls2);
    p.learn(allLearningClasses);

    //plotClasses(cls1, cls2, p);

    /*****************************/
    /*      Testing section      */
    /*****************************/

    //class 1
    File f3 = new File("/Users/matthewletter/Documents/single-
perceptron/PDRClass1Testing.txt");
    ArrayList<Sample> cls3 = parseFile(f3,1);

    //class 2
    File f4 = new File("/Users/matthewletter/Documents/single-
perceptron/PDRClass2Testing.txt");
    ArrayList<Sample> cls4 = parseFile(f4,-1);

    ArrayList<Sample> allTestingClasses = new ArrayList<Sample>();
    allTestingClasses.addAll(cls3);
```

```
        allTestingClasses.addAll(cls4);
        p.test(allTestingClasses);

        generalize(cls1, cls2, allLearningClasses, allTestingClasses);
        //plotClasses(cls3, cls4, p);
    }

    private static void generalize(ArrayList<Sample> cls1,ArrayList<Sample>
cls2,ArrayList<Sample> allLearningClasses,
                                   ArrayList<Sample> allTestingClasses) {
        // create your PlotPanel (you can use it as a JPanel)
        Plot2DPanel plot = new Plot2DPanel();
        // define the legend position
        plot.addLegend("SOUTH");
        double learningRate = .25;
        for (int j = 0; j < 50; j++) {
            Perceptron p = new Perceptron();
            p.w0 = p.rnd.nextDouble();
            p.w1 = p.rnd.nextDouble();
            p.w2 = p.rnd.nextDouble();
            p.maxIterations = 1;
            p.learningRate = learningRate;
            System.out.println("starting| w0:" + p.w0 + " w1:" + p.w1 + " w2:" +
p.w2);
            int length = 100;




            double[] x1 = new double[length];
            double[] x2 = new double[length];
            double[] y = new double[length];


            for (int i = 0; i < length; i++) {
                x1[i] = p.learn(allLearningClasses);
                x2[i] = p.test(allTestingClasses);
                y[i] = i;
            }
            plot.addLinePlot("learning", Color.BLUE, y, x1);
            plot.addLinePlot("testing", Color.RED, y, x2);
        }

        // put the PlotPanel in a JFrame like a JPanel
        JFrame frame = new JFrame("class1 vs class2");
        frame.setSize(1000, 1000);
```

```
        frame.setContentPane(plot);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
    }


}
```

Source code Sample

```
import java.util.HashSet;
import java.util.List;

/**
 * Created by matthewletter on 9/30/14.
 */
public class Sample {
    public double X1 = 0;
    public double X2 = 0;
    public double expectedClass = 0;
    public int index = 0;
    Sample(int index, double X1, double X2, double expectedClass){
        this.X1=X1;
        this.X2=X2;
        this.expectedClass=expectedClass;
        this.index=index;
    }
}
```

Source code LMS

```
import org.math.plot.Plot2DPanel;

import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import java.util.Scanner;

/**
 * Created by matthewletter on 10/5/14.
 */
```

```
public class LMS {
    public Random rnd;
    public double w0;
    public double w1;
    public double w2;
    public double learningRate;
    private double x0;
    private int maxIterations;


    /**
     * this constructor build our random weights and sets learning rate and learning
iterations
     */
    LMS(){
        this.rnd = new Random(System.currentTimeMillis());
//        this.w0 = rnd.nextDouble();
//        this.w1 = rnd.nextDouble();
//        this.w2 = rnd.nextDouble();
        this.w0 = 1;
        this.w1 = 1;
        this.w2 = 1;
        this.learningRate = .25;
        this.x0 = -1;
        this.maxIterations = 600;

    }

    /**
     * this class is used to teach the perceptron its weights, when there are no
longer errors
     * or there are no longer iterations its stops learning
     * @param samples ArrayList of samples
     */
    public double learn(ArrayList<Sample> samples) {
        int iterations = 0;
        boolean error = true;
        double errorVal=10;
        double errorValOld = 0;
        double errorSumSqr;
        double n = samples.size();
        int y = 0;
        double y0 = 0;

        double alpha =  (learningRate / 1000);
```

```
        //go through the epoche's
        while (error && iterations < maxIterations) {

            errorValOld = errorVal;
            errorVal = 0;
            errorSumSqr = 0;

            Collections.shuffle(samples);
            //iterate through the epoche
            for(Sample sample : samples) {
                double x1 = sample.X1;
                double x2 = sample.X2;
                error = false;

                if (((w1 * x1) + (w2 * x2) - w0) < 0) {
                    y = -1;
                } else {
                    y = 1;
                }
                y0=(w1 * x1) + (w2 * x2) - w0;
                if(y != sample.expectedClass){
                    errorSumSqr += (sample.expectedClass - y)*(sample.expectedClass -
y);
                }

                if ((sample.expectedClass - y0)!=0) {
                    w0 = w0 + alpha * (sample.expectedClass - y0) * x0 / 2;
                    w1 = w1 + alpha * (sample.expectedClass - y0) * x1 / 2;
                    w2 = w2 + alpha * (sample.expectedClass - y0) * x2 / 2;
                }
                //System.out.println("w0: "+w0+" w1: "+w1+" w2: "+w2 +"\n");
            }

            iterations++;
            errorVal = Math.sqrt(errorSumSqr/n);

            if( (errorValOld - errorVal) > 0.00001) {
                error = true;
            }

            System.out.println(errorValOld +" : "+ (errorValOld - errorVal));
            System.out.println("epoche: " + iterations + " |\n w0: "+w0+" w1: "+w1+"
w2: "+w2 +" RMS error: " +
                    errorVal);
        }
        return errorVal;
```

```
    }
    public double test(ArrayList<Sample> samples) {
        int iterations = 0;
        boolean error = true;
        double errorVal;
        double errorSumSqr;
        double n = samples.size();

        //go through the epoche's
        errorVal = 0;
        errorSumSqr = 0;

        //iterate through the epoche
        for(Sample sample : samples) {
            double x1 = sample.X1;
            double x2 = sample.X2;
            int y;//output

            if (((w1 * x1) + (w2 * x2) - w0) < 0) {
                y = -1;
            } else {
                y = 1;
            }
            if (y != sample.expectedClass) {
                errorSumSqr += (sample.expectedClass - y)*(sample.expectedClass -
y);
            }
            //System.out.println("w0: "+w0+" w1: "+w1+" w2: "+w2 +"\n");
        }
        errorVal = Math.sqrt(errorSumSqr/n);

        System.out.println("Test: " + iterations + " |\n w0: "+w0+" w1: "+w1+" w2:
"+w2 +" RMS error: " +
                errorVal);
        return errorVal;
    }


    /**
     * used to plot all of our data points
     * @param cls1 class 1
     * @param cls2 class 2
     */
    public static void plotClasses(ArrayList<Sample> cls1, ArrayList<Sample>
cls2, LMS p){
        // define your data
```

```
double[] x;
double[] y;

x = new double[cls1.size()];
y = new double[cls1.size()];

for (int i = 0; i < cls1.size(); i++) {
   x[i]=cls1.get(i).X1;
   y[i]=cls1.get(i).X2;
}


// create your PlotPanel (you can use it as a JPanel)
Plot2DPanel plot = new Plot2DPanel();

// define the legend position
plot.addLegend("SOUTH");

// add a line plot to the PlotPanel
plot.addScatterPlot("class 1", Color.RED, x, y);


double y0=0;
double y1=1;
double[] linex = {-4,-3,-2,-1,0,1,2,3,4,5};
double[] liney = {-4,-3,-2,-1,0,1,2,3,4,5};
for (int i = 0; i < linex.length; i++) {
   liney[i]= ((-p.w1/p.w2)*linex[i])+(p.w0/p.w2);
}
//add line plot weights
plot.addLinePlot("calculated Decision bound",Color.GREEN,linex,liney);



x = new double[cls2.size()];
y = new double[cls2.size()];

for (int i = 0; i < cls2.size(); i++) {
   x[i]=cls2.get(i).X1;
   y[i]=cls2.get(i).X2;
}
plot.addScatterPlot("class 2", Color.BLUE, x, y);

// put the PlotPanel in a JFrame like a JPanel
JFrame frame = new JFrame("class1 vs class2");
frame.setSize(1000, 1000);
```

```
        frame.setContentPane(plot);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
    }

    /**
     * used to parse the provided text files
     * @param f file
     * @param classNumber +-1
     * @return ArrayList of Sample
     */
    public static ArrayList<Sample> parseFile(File f,int classNumber){
        Scanner scanner;
        String[] sA;
        String s;
        ArrayList<Sample> samples = new ArrayList<Sample>();
        try {
            scanner = new Scanner(f);
            s = scanner.nextLine();

            while(scanner.hasNext()){
                sA = s.split(" ");
                if(sA.length==3)
                    samples.add(new Sample(Integer.parseInt(sA[0]),
Double.parseDouble(sA[1]), Double.parseDouble(sA[2]),
                            classNumber));
                s = scanner.nextLine();
            }
            scanner.close();

        }catch(FileNotFoundException e){
            e.printStackTrace();
        }
        return samples;
    }

    /**
     * sets up the ptron and plots data points
     * @param args not used
     */
    public static void main(String[] args){
        LMS p = new LMS();

        /*****************************/
        /*     Training section      */
        /*****************************/
```

```
        //class 1
        File f1 = new File("/Users/matthewletter/Documents/single-
perceptron/DeltaRuleClass1Training.txt");
        ArrayList<Sample> cls1 = parseFile(f1,1);

        //class 2
        File f2 = new File("/Users/matthewletter/Documents/single-
perceptron/DeltaRuleClass2Training.txt");
        ArrayList<Sample> cls2 = parseFile(f2,-1);

        //plotClasses(cls1, cls2, p);

        ArrayList<Sample> allLearningClasses = new ArrayList<Sample>();
        allLearningClasses.addAll(cls1);
        allLearningClasses.addAll(cls2);
        p.learn(allLearningClasses);

        //plotClasses(cls1, cls2, p);

        /*****************************/
        /*      Testing section      */
        /*****************************/

        //class 1
        File f3 = new File("/Users/matthewletter/Documents/single-
perceptron/PDRClass1Testing.txt");
        ArrayList<Sample> cls3 = parseFile(f3,1);

        //class 2
        File f4 = new File("/Users/matthewletter/Documents/single-
perceptron/PDRClass2Testing.txt");
        ArrayList<Sample> cls4 = parseFile(f4,-1);

        ArrayList<Sample> allTestingClasses = new ArrayList<Sample>();
        allTestingClasses.addAll(cls3);
        allTestingClasses.addAll(cls4);
        p.test(allTestingClasses);

        generalize(cls1, cls2, allLearningClasses, allTestingClasses);
        //plotClasses(cls3, cls4, p);
    }

    private static void generalize(ArrayList<Sample> cls1,ArrayList<Sample>
cls2,ArrayList<Sample> allLearningClasses,
                                   ArrayList<Sample> allTestingClasses) {
```

```
        // create your PlotPanel (you can use it as a JPanel)
        Plot2DPanel plot = new Plot2DPanel();
        // define the legend position
        plot.addLegend("SOUTH");
        double learningRate = .25;
        for (int j = 0; j < 100; j++) {
           LMS p = new LMS();
           p.w0 = p.rnd.nextDouble();
           p.w1 = p.rnd.nextDouble();
           p.w2 = p.rnd.nextDouble();
           p.maxIterations = 1;
           p.learningRate = learningRate +=.25;
           System.out.println("starting| w0:" + p.w0 + " w1:" + p.w1 + " w2:" +
p.w2);
           int length = 6;




           double[] x1 = new double[length];
           double[] x2 = new double[length];
           double[] y = new double[length];


           for (int i = 0; i < length; i++) {
              x1[i] = p.learn(allLearningClasses);
              x2[i] = p.test(allTestingClasses);
              y[i] = i;
           }
           plot.addLinePlot("learning", Color.BLUE, y, x1);
           plot.addLinePlot("testing", Color.RED, y, x2);
        }

        // put the PlotPanel in a JFrame like a JPanel
        JFrame frame = new JFrame("class1 vs class2");
        frame.setSize(1000, 1000);
        frame.setContentPane(plot);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
     }
}
```