

Noah Buchanan  
Problem Set 5  
Machine Learning at 6:40 PM

April 30, 2021

New Sample Weight = sample weight \*  $e^{\text{amount of say}}$

Number of Input Records: 600

Number of Features: 10

Bias weight: -0.401

Iterations Required for Convergence in Training Data: 38

Loss Function output for Testing Data: 649.249

predicted: +1, true value: +1

predicted: +1, true value: +1

predicted: +1, true value: -1

predicted: -1, true value: -1

predicted: -1, true value: -1

predicted: -1, true value: +1

predicted: -1, true value: +1

predicted: -1, true value: -1

predicted: -1, true value: -1

predicted: +1, true value: +1



## 1 Source Code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

/*****
Name: Noah Buchanan
Username: ua100
Problem Set: PS5
Due Date: April 23, 2021
*****/

public class SVM {

    public static double [][] X;
    public static double [] Y;
    public static int epoch;
    public static double [] weights;
    public static double learnRate;
    public static double epsilon;
    public static double lambda = 8;
```

```

public static void main(String[] args) throws IOException{
    //read in data in "svmtrain"
    X = readin(args[0]);
    //rearrange y at end depending on y index given
    X = rearrange(X,Integer.parseInt(args[1]));
    //grab y from rearranged matrix
    Y = getY(X);
    //trim y from X matrix and normalize all data except bias
    X = normalize(X);

    learnRate = Double.parseDouble(args[3]);
    epsilon = Double.parseDouble(args[4]);
    GradientDescent();

    System.out.printf("Number of Input Records: %d\n", X.length);
    System.out.printf("Number of Features: %d\n", X[0].length);
    System.out.printf("Bias weight: %.3f\n", weights[0]);
    System.out.printf("Iterations Required for Convergence in Training Data: %d\n", iterations);
    X = readin("svmtest");
    X = rearrange(X,Integer.parseInt(args[1]));
    Y = getY(X);
    X = normalize(X);
    System.out.printf("Loss Function output for Testing Data: %.3f\n", loss(X,weights));

    for(int i = 0; i < 10; i++) {
        double yhat = 0;
        for(int j = 0; j < X[0].length; j++) {
            yhat += X[i][j] * weights[j];
        }
        if(yhat > 0 && Y[i] > 0) {
            System.out.println("predicted: +1, true value: +1");
        } else if(yhat > 0 && Y[i] < 0){
            System.out.println("predicted: +1, true value: -1");
        } else if(yhat < 0 && Y[i] > 0){
            System.out.println("predicted: -1, true value: +1");
        } else if(yhat < 0 && Y[i] < 0){
            System.out.println("predicted: -1, true value: -1");
        }
    }
}

public static void GradientDescent() throws IOException{

    BufferedWriter bw = new BufferedWriter(new FileWriter("loss"));

```

```

double[] weightsAdjust = new double[X[0].length];
weights = weightsAdjust;
double previousLoss = loss(X, weights);

double currentLoss = 0;
epoch = 0;

while(cost(currentLoss, previousLoss) >= epsilon) {

    for(int k = 1; k < weights.length; k++) {
        weightsAdjust[k] = weightsAdjust[k] - learnRate * lossDerivWeights(X, wei
    }
    weightsAdjust[0] = weightsAdjust[0] - learnRate * lossDerivBias(X, weights)

    weights = weightsAdjust;

    previousLoss = currentLoss;
    currentLoss = loss(X, weights);

    epoch++;

    bw.write(String.format("%d, %.3f", epoch, currentLoss));
    bw.newLine();

}

bw.close();

}

public static double lossDerivWeights(double[][] X, double[] weights, int k) {

    double summation = 0;
    double[] yhat = yhat(X, weights);

    for(int i = 0; i < X.length; i++) {

        if( (Y[i] * yhat[i]) < 1) {
            summation += -Y[i]*X[i][k];
        }
    }
    return weights[k] + (lambda*summation);
}

```

```

public static double lossDerivBias(double [][] X, double [] weights) {

    double summation = 0;
    double [] yhat = yhat(X, weights);
    for(int i = 0; i < X.length; i++) {

        if( (Y[i] * yhat[i]) < 1) {
            summation += -Y[i];
        }
    }
    return lambda*summation;
}

public static double [] yhat(double [][] X, double [] W) {

    double [] yhat = new double[X.length];

    for (int i = 0; i < X.length; i++) {
        for (int j = 0; j < X[0].length; j++) {
            yhat[i] += (X[i][j] * W[j]);
        }
    }
    return yhat;
}

public static double loss(double [][] X, double [] weights) {

    double weightsSquared = 0;
    for(int i = 0; i < weights.length; i++) {
        weightsSquared += weights[i] * weights[i];
    }
    weightsSquared /= 2;

    double [] yhat = yhat(X, weights);

    double summation = 0;
    for(int i = 0; i < X.length; i++) {
        if( (1 - (Y[i] * yhat[i])) > 0) {
            summation += (1 - (Y[i] * yhat[i]));
        }
    }
    return weightsSquared + (lambda * summation);
}

public static float cost(double current, double previous) {

```

```

        return (float) ((Math.abs(previous - current) * 100) / previous);
    }

    public static double [][] rearrange(double [][] X, int yindex){

        double [][] temp = new double[X.length][X[0].length];
        for(int i = 0; i < X.length; i++) {
            for( int j = 0; j < X[0].length; j++) {
                temp[i][j] = X[i][j];
            }
        }
        for(int i = 0; i < X.length; i++) {
            temp[i][yindex] = X[i][X[0].length-1];
            temp[i][X[0].length-1] = X[i][yindex];
        }
        return temp;
    }

    public static double [] getY(double [][] X) {
        double [] temp = new double[X.length];
        for(int i = 0; i < X.length; i++) {
            temp[i] = X[i][X[0].length-1];
        }
        return temp;
    }

    public static double [][] normalize(double [][] X){

        double [][] temp = new double[X.length][X[0].length-1];
        //add bias to temp var
        for(int i = 0; i < X.length; i++) {
            temp[i][0] = 1;
        }
        //normalize everything but bias column and y column
        for(int j = 1; j < X[0].length-1; j++) {

            double sum = 0;
            for(int i = 0; i < X.length; i++) {
                sum += X[i][j];
            }
            double mean = sum/X.length;
            sum = 0;
            for(int i = 0; i < X.length; i++) {
                sum += Math.pow((X[i][j]-mean),2);
            }
            sum /= X.length-1;
        }
    }

```

```

        double stdv = Math.sqrt(sum);
        for(int i = 0; i < X.length; i++) {
            temp[i][j] = (X[i][j] - mean)/stdv;
        }
    }

    return temp;
}

public static double [][] readin(String filename){

    ArrayList<ArrayList<Double>> data = new ArrayList<>();
    try {

        BufferedReader br = new BufferedReader(new FileReader(filename));
        int row = 0;
        String line = "";

        while(( line = br.readLine() ) != null) {
            data.add(new ArrayList<>());
            String [] split = line.split(",");
            for(int i = 0; i < split.length; i++) {
                data.get(row).add(Double.parseDouble(split[i]));
            }
            row++;
        }
        double [][] matrix = new double[data.size()][data.get(0).size()];
        for(int i = 0; i < data.size(); i++){
            for(int j = 0; j < data.get(0).size(); j++){
                matrix[i][j] = data.get(i).get(j);
            }
        }

        return matrix;
    } catch(Exception ex) {
        ex.printStackTrace();
        return null;
    }
}
}

```