# Noah Buchanan
# Problem Set 4
# AI at 5:25 PM

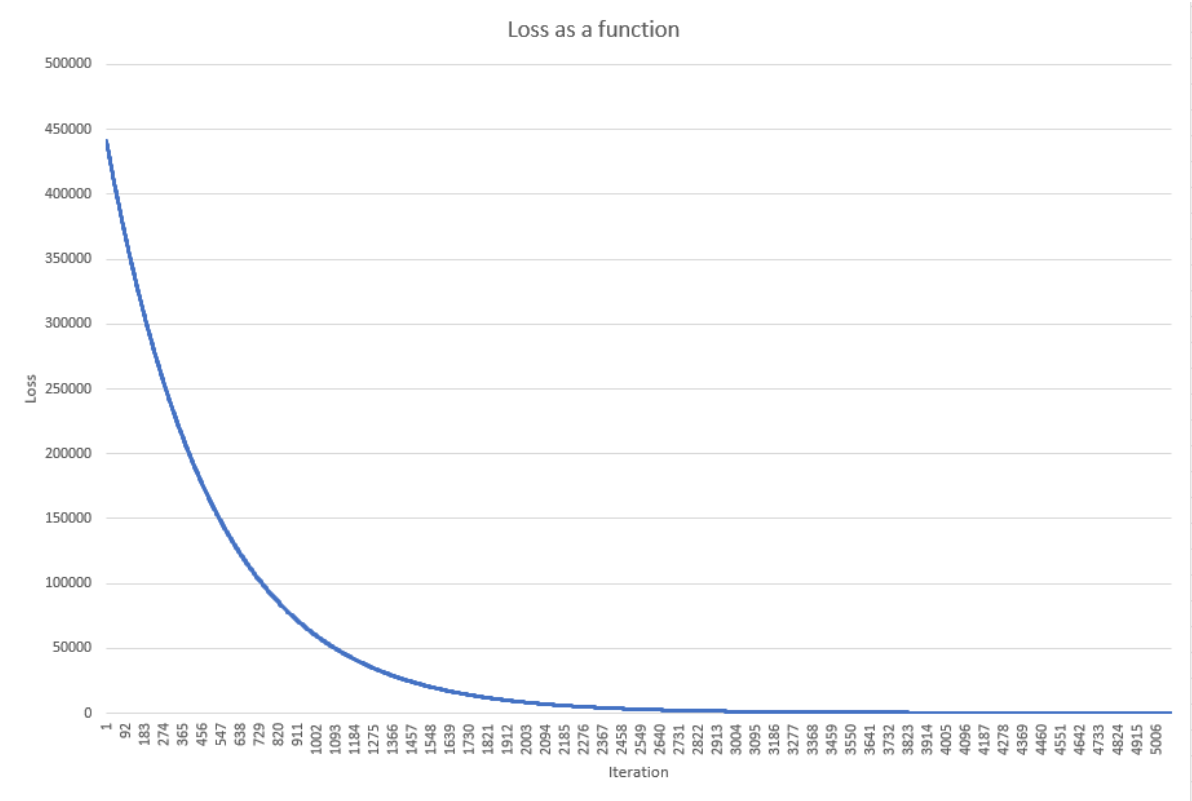
March 3, 2021

The number of Input Records was 48, the number of features was 16, the Y-intercept was 932.095703, it required 5077 iterations for convergence, the final loss function output was 688.062256 and here is a sample against 10 of the test values

True Y: 931.88275, Predicted Y: 911.817
True Y: 899.3581, Predicted Y: 934.7
True Y: 882.5407, Predicted Y: 871.338
True Y: 1032.0085, Predicted Y: 1006.49
True Y: 899.5871, Predicted Y: 997.875
True Y: 903.31665, Predicted Y: 904.155
True Y: 890.0173, Predicted Y: 899.264
True Y: 1029.3625, Predicted Y: 1071.289
True Y: 938.4584, Predicted Y: 946.185
True Y: 908.0982, Predicted Y: 950.672

*Disclaimer* These numbers are subject to change due to randomization of the input file specified in the command line. I randomize the data into a 80 20 split everytime it runs and then use the files generated from the input file as testing and training. To stop this from happening simply run it once to generate the 80 and 20 split text files and then at the top of the main method comment out the splitDataOf() method and you can run it without a different randomization everytime.

## Loss as a function




## 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: PS4
 * Due Date: March 1st,2021
 *******************************/

public class PS4 {
```

```
public static float[][] X;
public static float[] Y;
public static float[] weights;

public static void main(String[] args) throws IOException {

   // randomly splits data in 80.txt and 20.txt for training and testing
   splitDataOf(args[0]);

   // this is not hardcoded technically I split the data randomly from the in
   // file the user
   // wants and created two files called 80.txt and 20.txt for training and t
   // and
   // it does this everytime
   X = getNormalizedData("80.txt", Integer.parseInt(args[1]));
   int iterations = GradientDescent(X, Y,
         weights, Float.parseFloat(args[3]), Float.parseFloat(args[4]));

   // storing final weights
   BufferedWriter bw = new BufferedWriter(new FileWriter(args[2]));
   for (int i = 0; i < weights.length; i++) {
      bw.write(weights[i] + "\n");
   }
   bw.close();

   // output
   System.out.printf(
         "Number of Input Records: %d\n"
         + "Number of Features: %d\n"
         + "Y-intercept: %f\n"
         + "Iterations Required for Convergence in Training Data: %d\n",
         X.length, X[0].length, weights[0], iterations);

   String comparisons = finalLoss(Integer.parseInt(args[1]));
   System.out.printf("\n%s", comparisons);

}

public static String finalLoss(int y) throws IOException {

   float[][] x = getNormalizedData("20.txt", y);
   String comp = "";
   float loss = 0;
   for (int j = 0; j < x.length; j++) {
      float sum = 0;
```

```
            for (int i = 0; i < x[0].length; i++) {
                sum += x[j][i] * weights[i];
            }
            if (j < 10) {
                comp += "True Y: " + sum + ", Predicted Y: " + Y[j] + "\n";
            }
            loss += Math.pow((Y[j] - sum), 2);
        }
        System.out.printf("Loss Function for Testing Data: "
                + "%f", loss / (2 * Y.length));
        return comp;
    }

    public static float loss(float[][] X, float[] Y, float[] W) {

        float loss = 0;

        float[] yhat = yhat(X, W);

        for (int j = 0; j < Y.length; j++) {
            loss += Math.pow((Y[j] - yhat[j]), 2);
        }

        loss /= (2 * Y.length);
        return loss;
    }

    public static float rateOfLoss(float[][] X,
            float[] Y, float[] W, int weightIndex) {

        float rate = 0;

        float[] yhat = yhat(X, W);

        for (int j = 0; j < Y.length; j++) {
            rate += ((Y[j] - yhat[j]) * (-X[j][weightIndex]));
        }

        rate /= Y.length;

        return rate;
    }

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

        float[] yhat = new float[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 int GradientDescent(float[][] X,
        float[] Y, float[] W, float alpha, float epsilon)
        throws IOException {

    float previous = 0;
    float current = 0;
    W = new float[X[0].length];
    weights = new float[W.length];
    int i = 0;

    while (cost(current, previous) >= epsilon || i < 2) {

        for (int k = 0; k < W.length; k++) {
            W[k] = W[k] - alpha * rateOfLoss(X, Y, weights, k);
        }
        i++;
        previous = current;
        current = loss(X, Y, W);

        weights = W;
    }
    return i;
}

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

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

public static float[][] getNormalizedData(String filename,
        int y) throws IOException {

    ArrayList<ArrayList<Float>> list = new ArrayList<>();
    BufferedReader br = new BufferedReader(new FileReader(filename));
    String line = "";
    int i = 0;
```

```
while ((line = br.readLine()) != null) {
   String[] split = line.split(",");
   list.add(new ArrayList<Float>());
   for (int j = 0; j < split.length; j++) {
      list.get(i).add(Float.parseFloat(split[j]));
   }
   i++;
}
br.close();
float[][] data = new float[list.size()][list.get(0).size() - 1];

Y = new float[list.size()];
for (int k = 0; k < Y.length; k++) {
   Y[k] = list.get(k).get(y);
}

for (int k = 0; k < data.length; k++) {
   for (int p = 0; p < data[0].length; p++) {
      data[k][p] = list.get(k).get(p);
   }
}

float[] means = new float[data[0].length];

for (int k = 0; k < data[0].length; k++) {
   for (int p = 0; p < data.length; p++) {
      means[k] += data[p][k];
   }
   means[k] /= data.length;
}

float[] residual = new float[data[0].length];

for (int k = 0; k < data[0].length; k++) {
   for (int p = 0; p < data.length; p++) {
      residual[k] += Math.abs((data[p][k] - means[k]));
   }
   residual[k] /= data.length - 1;
}

for (int k = 0; k < data.length; k++) {
   for (int p = 0; p < data[0].length; p++) {
      data[k][p] = (data[k][p] - means[p]) / residual[p];
   }
}
float[][] hold = new float[data.length][data[0].length + 1];
```

```
        for (int k = 0; k < data.length; k++) {
            hold[k][0] = 1;
            for (int p = 0; p < data[0].length; p++) {
                hold[k][p + 1] = data[k][p];
            }
        }

        return hold;
    }

    public static void splitDataOf(String filename) throws IOException {

        BufferedReader br = new BufferedReader(new FileReader(filename));
        BufferedWriter bw1 = new BufferedWriter(new FileWriter("80.txt"));
        BufferedWriter bw2 = new BufferedWriter(new FileWriter("20.txt"));
        String line = "";
        ArrayList<String> holder = new ArrayList<>();
        while ((line = br.readLine()) != null) {
            holder.add(line);
        }
        int size = holder.size() - 1;
        bw1.write(holder.get(0));
        bw1.newLine();
        holder.remove(0);

        for (int i = 0; i < size; i++) {

            int rand = (int) (Math.random() * (holder.size() - 1));

            if (i < (int) (size * .8)) {
                bw1.write(holder.get(rand));
                bw1.newLine();
            } else {
                bw2.write(holder.get(rand));
                bw2.newLine();
            }
            holder.remove(rand);

        }
        br.close();
        bw1.close();
        bw2.close();
    }
}
```