

Problem Set 2 - Part 1

Dan McQuillan

Handed In: February 11, 2014

1. Solution to problem 1

(a) Training on data set 1

a) The final weights

 $StartingThreshold = 0.0$ $CalculatedThreshold = -54.0$ **Weights:** $w_1 \rightarrow -752.0$ $w_2 \rightarrow 4771.0$ $w_3 \rightarrow 17714.0$ $w_4 \rightarrow 762.0$ $w_5 \rightarrow 6.0$ $w_6 \rightarrow 676.0$ $w_7 \rightarrow 3060.0$ $w_8 \rightarrow -2004.0$ $w_9 \rightarrow 5459.0$ $w_{10} \rightarrow 9591.2999999989$ $w_{11} \rightarrow 3832.0$ $w_{12} \rightarrow 14963.0$ $w_{13} \rightarrow 20912.0$

b) The number of training epochs required

The training took 1980 epochs.

c) The margin

 $\gamma \rightarrow 1.5226515850810045 \times 10^{-5}$

(b) Test on data set 1

a) A confusion matrix

54	0
0	63

b) Two lists of example indices

No errors were found with this dataset since the weight vectors were calculated from this dataset.

- c) The total loss summed over the misclassified examples
The loss for this dataset is 0.0 since it is the training data set.

(c) **Test on data set 2**

- a) A confusion matrix

12	1
0	20

- b) Two lists of example indices

False Negatives:

Index: 22

Inputs: (60.0, 1.0, 4.0, 140.0, 293.0, 0.0, 2.0, 170.0, 0.0, 1.2, 2.0, 2.0, 7.0)

False Positives:

There were no false positives

- c) **The total loss summed over the misclassified examples**

Total loss: 77.440000001312

(d) **Application to data set 3**

Classifications:

1 \rightarrow 0.0

2 \rightarrow 1.0

3 \rightarrow 0.0

4 \rightarrow 1.0

5 \rightarrow 1.0

6 \rightarrow 0.0

7 \rightarrow 0.0

8 \rightarrow 0.0

9 \rightarrow 0.0

10 \rightarrow 1.0

11 \rightarrow 1.0

12 \rightarrow 0.0

13 \rightarrow 1.0

14 \rightarrow 0.0

15 \rightarrow 0.0

16 \rightarrow 0.0

17 \rightarrow 1.0

18 \rightarrow 0.0

19 \rightarrow 0.0

20 \rightarrow 0.0

21 \rightarrow 0.0

Which feature is the most important?

In order to determine the most influential property I have used the correlation

coefficients to find the most correlation between each property and the classification. The source code to find this property is also within the project.

Label	Correlation Coefficient
age	0.576399638726158
sex	0.15811388300841894
chest	0.4934637712198269
resting blood pressure	0.08008953852726183
serum cholestoral	−0.32382553481251514
fasting blood sugar	0.31622776601683766
resting electrocardiographic results	0.1386750490563073
maximum heart rate achieved	−0.5353426981014223
exercise induced angina	0.685994340570035
oldpeak	0.8086701966434094
slope	0.6123724356957945
number of major vessels	0.8152133857595864
thal	0.43905703995876144

Property With Maximum Correlation

Label: number of major vessels

Correlation Coefficient: 0.8152133857595864

1 Source Code

The following is the source code for the assignment. It consists of 5 classes to get all the required output.

- **CorrelationCoefficient**
This class is used to calculate the correlation coefficient on the resultant classifications when running the learner in application mode to tell which is the most important input.
- **Pair**
This is mainly a utility class used to group the weight and input together for clarity in traversal of the arrays.
- **VectorUtils**
This is a collection of vector utilities that are used throughout the application such as: dot product, addition, scaling, finding the norm, etc.
- **WidrowHoffLearner**
This class has the Widrow Hoff Learning specific code as defined in the assignment and lectures.

- Main

This class simply runs all the classes and displays the output.

I have excluded the source for DataSet for brevity.

1.1 Main.java

```

1 package hw2.widrowhoff;
2
3 import java.util.List;
4
5 public class Main {
6
7     public static void main(String[] args) {
8         final String dataSet1Csv = "DataSet1.csv";
9         final String dataSet2Csv = "DataSet2.csv";
10        final String dataSet3Csv = "DataSet3.csv";
11
12        DataSet dataSet1 = new DataSet(dataSet1Csv);
13        DataSet dataSet2 = new DataSet(dataSet2Csv);
14        DataSet dataSet3 = new DataSet(dataSet3Csv);
15
16        WidrowHoffLearner learner = new WidrowHoffLearner( dataSet1, 0.0 );
17        learner.trainWeightVector();
18
19        for( int i = 0; i < learner.getWeights().length; i++ ) {
20            System.out.println("\\"( w_{ " + (i+1) + " } \\\rightarrow " + learner.
21                getWeights()[i] + " \\\)");
22        }
23
24        System.out.println("_____");
25        System.out.println("Data set 1 test");
26        System.out.println("_____");
27        learner.testWeightVector(dataSet1);
28        System.out.println("_____");
29        System.out.println("Data set 2 test");
30        System.out.println("_____");
31        learner.testWeightVector(dataSet2);
32
33        System.out.println("_____");
34        System.out.println("Data set 3 application");
35        System.out.println("_____");
36
37        final List<Double> classifications = learner.applyWeightVector(dataSet3);
38        for( int i = 0; i < classifications.size(); i++ ) {
39            System.out.println( " \\\( " + (i+1) + " \\\rightarrow " + classifications.
40                get(i) + " \\\) \\\\" );
41        }
42
43        final List<Pair<String,Double>> coefficients = CorrelationCoefficient.
44            calculate(dataSet3, classifications);
45        System.out.println("_____");

```

```

43 System.out.println(" Correlation Coefficients");
44 System.out.println("_____");
45 String maxProp = "";
46 Double maxCorrelation = Double.MIN.VALUE;
47 System.out.println("\\( \\begin{array}{l|l}");
48 System.out.println("\\bf{Label} & \\bf{Correlation Coefficient} \\\\");
49 System.out.println("\\hline");
50 for (final Pair<String, Double> coefficient : coefficients ) {
51     System.out.println("\\text{ " + coefficient.getLeft().trim().replaceAll(
52         "[^A-Za-z0-9]", " ") + " } & " + coefficient.getRight() + " \\\\");
53     if ( coefficient.getRight().compareTo(maxCorrelation) > 0 ) {
54         maxProp = coefficient.getLeft();
55         maxCorrelation = coefficient.getRight();
56     }
57 System.out.println("\\end{array} \\)");
58 System.out.println();
59
60
61 System.out.println("_____");
62 System.out.println("Maximum Correlation Prop");
63 System.out.println("_____");
64 System.out.println("\\bf{ Property With Maximum Correlation} \\\\ ");
65 System.out.println("Label: " + maxProp.replaceAll("[^A-Za-z0-9]", " ") + "
66     \\\\");
67 System.out.println("Correlation Coefficient: \\(" + maxCorrelation + "\\)");
68 }
69 }

```

1.2 Pair.java

```

1 package hw2.widrowhoff;
2
3 public class Pair<T1, T2> {
4     private final T1 left;
5     private final T2 right;
6
7     public Pair(T1 left, T2 right) {
8         this.left = left;
9         this.right = right;
10    }
11
12    public T1 getLeft() {
13        return left;
14    }
15    public T2 getRight() {
16        return right;
17    }
18 }

```

1.3 VectorUtils.java

```
1 package hw2.widrowhoff;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class VectorUtils {
7
8     public static double dotProduct( final double[] x, final double[] y ) {
9         double sum = 0;
10        for( int i = 0; i < x.length && i < y.length; i++ ) {
11            sum += x[i] * y[i];
12        }
13
14        return sum;
15    }
16
17    public static double[] scale( final double[] x, final double scalar ) {
18        double[] xCopy = x.clone();
19        for( int i = 0; i < x.length; i++ ) {
20            xCopy[i] *= scalar;
21        }
22        return xCopy;
23    }
24
25    public static double[] add( double[] x, final double[] y ) {
26        double[] xCopy = x.clone();
27        double[] yCopy = y.clone();
28        for( int i = 0; i < xCopy.length && i < yCopy.length; i++ ) {
29            xCopy[i] += yCopy[i];
30        }
31
32        return xCopy;
33    }
34
35    public static List<Pair<Double, Double>> zip( double[] x, double[] y ) {
36        List<Pair<Double, Double>> zippedVector = new ArrayList<Pair<Double,
37            Double>>();
38        for( int i = 0; i < x.length && i < y.length; i++ ) {
39            zippedVector.add( new Pair<Double, Double>(x[i], y[i]) );
40        }
41        return zippedVector;
42    }
43
44    public static double[] concat( double[] left, double[] right ) {
45        if( left == null || right == null ) {
46            throw new IllegalArgumentException("The left or right arguments must not
47                be null");
48        }
49
50        final int totalLength = left.length + right.length;
51        double[] concatted = new double[totalLength];
52        for(int i = 0; i < left.length; i++ ) {
```

```

51     concatted[i] = left[i];
52 }
53 for(int i = 0; i < right.length; i++) {
54     concatted[i + left.length] = right[i];
55 }
56 return concatted;
57 }
58
59 public static double[] unitVector( double[] x ) {
60     double[] copy = x.clone();
61     double norm = norm(copy);
62     return scale(x, 1.0 / norm );
63 }
64
65 public static double norm( double[] x ) {
66     double sum = 0;
67     for( int i = 0; i < x.length; i++) {
68         sum += Math.pow(x[i], 2.0);
69     }
70     return Math.sqrt(sum);
71 }
72 }

```

1.4 CorrelationCoefficient.java

```

1 package hw2.widrowhoff;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class CorrelationCoefficient {
7
8     public static List<Pair<String,Double>> calculate( final DataSet dataSet ,
9         final List<Double> classifications ) {
10         final String[] labels = dataSet.fields;
11         List<Pair<String,Double>> coefficients = new ArrayList<Pair<String,Double>>();
12
13         final List<List<Double>> transformedArray = transform(dataSet.exData);
14         final double classificationMean = mean(classifications);
15
16         int i = 0;
17         for( final List<Double> array : transformedArray ) {
18             final double mean = mean(array);
19             final String label = labels[i];
20
21             double numeratorSum = 0.0;
22             double xDenominatorSum = 0.0;
23             double yDenominatorSum = 0.0;
24
25             int j = 0;
26             for( final Double x : array ) {

```

```

26     final double xDiff = x - mean;
27     final double yDiff = classifications.get(j) - classificationMean;
28
29     numeratorSum += xDiff * yDiff;
30     xDenominatorSum += xDiff * xDiff;
31     yDenominatorSum += yDiff * yDiff;
32
33     j++;
34 }
35
36     final Double correlationCoefficient = numeratorSum / Math.sqrt(
37         xDenominatorSum * yDenominatorSum);
38     final Pair<String,Double> coefficientWithLabel = new Pair<String,Double>
39         >(label, correlationCoefficient);
40     coefficients.add(coefficientWithLabel);
41     i++;
42 }
43
44     return coefficients;
45 }
46
47 private static List< List<Double> > transform(double[][] data ) {
48     List<List<Double>> transformed = new ArrayList<List<Double>>();
49
50     for( int i = 0; i < data[0].length; i++ ) {
51         List<Double> column = new ArrayList<Double>();
52         for( int j = 0; j < data.length; j++ ) {
53             column.add(data[j][i]);
54         }
55         transformed.add(column);
56     }
57
58     return transformed;
59 }
60
61 @SuppressWarnings("unused")
62 private static double mean(final double[] array) {
63     double sum = 0.0;
64     for( int i = 0; i < array.length; i++ ) {
65         sum+= array[i];
66     }
67     return sum / (double) array.length;
68 }
69
70 private static double mean(final List<Double> array) {
71     double sum = 0.0;
72     for( int i = 0; i < array.size(); i++ ) {
73         sum+= array.get(i);
74     }
75     return sum / (double) array.size();
76 }

```


1.5 WidrowHoffLearner.java

```

1 package hw2.widrowhoff;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class WidrowHoffLearner {
7
8     private double[] weights;
9     private double[] homogeneousWeights;
10    private double[][] x;
11    private double alpha = 1;
12    private double threshold;
13    private double[] labels;
14    private double margin;
15
16    public WidrowHoffLearner( final DataSet dataSet, final double[] weights,
17                             final double threshold ) {
18        this.x = dataSet.exData;
19        this.labels = dataSet.exLabels;
20        this.threshold = threshold;
21    }
22
23    public WidrowHoffLearner( final DataSet dataSet, final double threshold ) {
24        this.x = dataSet.exData;
25        this.labels = dataSet.exLabels;
26        final int inputLength = x[0].length; // get input vector length less the
27                                                value of the class
28
29        double[] weightsVector = new double[inputLength];
30        for( int i = 0; i < inputLength; i++ ) {
31            weightsVector[i] = 0;
32        }
33        this.weights = weightsVector;
34
35        homogeneousWeights = VectorUtils.concat(new double[] { threshold },
36                                                weights );
37
38        this.threshold = threshold;
39    }
40
41    public void trainWeightVector() {
42        int iterations = 0;
43        int errors;
44        do {
45            errors = 0;
46            for( int i = 0; i < x.length; i++ ) {
47                double[] heterogeneousInputs = VectorUtils.concat(new double[] { -1.0

```

```

48      // 0 - correct
49      // -1 - false positive
50      // 1 - false negative
51      if( err != 0.0 ) {
52          errors++;
53      //      double loss = -1.0 * err * VectorUtils.dotProduct(inputs, weights)
54      ;
55
56      final double[] scaledInputs = VectorUtils.scale(heterogeneousInputs,
57          alpha * err );
58      homogeneousWeights = VectorUtils.add(scaledInputs,
59          homogeneousWeights);
60      }
61      }
62      iterations++;
63      } while( errors != 0);
64      calculateMargin();
65
66      for( int i = 1; i < homogeneousWeights.length; i++ ) {
67          weights[i-1] = homogeneousWeights[i];
68      }
69
70      System.out.println("The training took " + iterations + " epochs.");
71      System.out.println("Threshold: " + homogeneousWeights[0]);
72      System.out.println("\\( \\gamma \\rightarrow " + margin + " \\");
73  }
74
75  public void testWeightVector(DataSet dataSet) {
76      int falseNegatives = 0;
77      int falsePositives = 0;
78      int truePositives = 0;
79      int trueNegatives = 0;
80      double totalLoss = 0.0;
81      for( int i = 0; i < dataSet.exData.length; i++ ) {
82          double[] heterogeneousInputs = VectorUtils.concat(new double[] { -1.0 },
83              dataSet.exData[i]);
84
85          final double percepResult = percepW( heterogeneousInputs );
86          final double err = dataSet.exLabels[i] - percepResult; // use last value
87                          in array as label
88
89          // 0 - correct
90          // -1 - false positive
91          // 1 - false negative
92          if( err > 0.0 ) {
93              falseNegatives++;
94              System.out.println("False Negative Found: ");
95              System.out.println("Index: " + i);
96              System.out.println("Inputs: " + vectorToString(heterogeneousInputs));
97              totalLoss += Math.abs(-1.0 * err * VectorUtils.dotProduct(
98                  heterogeneousInputs, homogeneousWeights ));

```

```

96         } else if ( err < 0.0 ) {
97             falsePositives++;
98             System.out.println("False Positive Found: ");
99             System.out.println("Index: " + i);
100             System.out.println("Inputs: " + vectorToString(heterogeneousInputs));
101             totalLoss += Math.abs(-1.0 * err * VectorUtils.dotProduct(
                heterogeneousInputs, homogeneousWeights ));
102         } else {
103             if( percepResult == 1 ) {
104                 truePositives++;
105             } else {
106                 trueNegatives++;
107             }
108         }
109     }
110
111     System.out.println("True Positives: " + truePositives );
112     System.out.println("True Negatives: " + trueNegatives );
113     System.out.println("False positives: " + falsePositives);
114     System.out.println("False negatives: " + falseNegatives);
115     System.out.println("Total loss: " + totalLoss);
116     System.out.println("Total items: " + dataSet.exData.length );
117 }
118
119 private String vectorToString( final double[] x ) {
120     StringBuilder stringBuilder = new StringBuilder();
121     stringBuilder.append("(");
122     for( int i = 1; i < x.length; i++ ) {
123         stringBuilder.append(x[i]);
124         if( i != x.length - 1 ) {
125             stringBuilder.append(", ");
126         }
127     }
128     stringBuilder.append(")");
129     return stringBuilder.toString();
130 }
131
132 public List<Double> applyWeightVector(DataSet dataSet) {
133     List<Double> classifications = new ArrayList<Double>();
134     for( int i = 0; i < dataSet.exData.length; i++ ) {
135         double[] heterogeneousInputs = VectorUtils.concat(new double[] { -1.0 },
            dataSet.exData[i]);
136         double classification = percepW(heterogeneousInputs);
137         classifications.add(classification);
138     }
139     return classifications;
140 }
141
142 private void calculateMargin() {
143     margin = Double.MAX_VALUE;
144     final double[] unitWeightVector = VectorUtils.unitVector(
        homogeneousWeights);
145
146     for( int i = 0; i < x.length; i++ ) {

```

```
147     double[] heterogeneousInputs = VectorUtils.concat(new double[] { -1.0 },
148               x[i]);
149     double dotProduct = Math.abs( VectorUtils.dotProduct(unitWeightVector,
150               heterogeneousInputs) );
151     double norm = VectorUtils.norm(heterogeneousInputs);
152     double marginPart = dotProduct / norm;
153     margin = Math.min(margin, marginPart);
154 }
155 }
156 }
157
158 private double percepW( final double[] curInputVector ) {
159     double sum = 0.0;
160
161     sum += VectorUtils.dotProduct( curInputVector, homogeneousWeights );
162
163     if( sum > 0 ) return 1;
164     else if ( sum < 0 ) return 0;
165     else return 0;
166 }
167
168 public double[] getWeights() {
169     return weights;
170 }
171 public double[][] getX() {
172     return x;
173 }
174
175 public double getAlpha() {
176     return alpha;
177 }
178
179 public double getThreshold() {
180     return threshold;
181 }
182
183 public double[] getLabels() {
184     return labels;
185 }
186
187 public double getMargin() {
188     return margin;
189 }
190
191 }
```