CS440 : Introduction to Artificial Intelligence **Spring 2014**

Problem Set 2 - Part 1

Dan McQuillan *Handed In: February 10, 2014*

1. **Solution to problem 1**

   (a) **Training on data set 1**

   a) The final weights

   $Threshhold = 0.0$

   Weights:
   $w_0 \rightarrow -748.0$
   $w_1 \rightarrow 4775.0$
   $w_2 \rightarrow 17744.0$
   $w_3 \rightarrow 780.0$
   $w_4 \rightarrow -6.0$
   $w_5 \rightarrow 657.0$
   $w_6 \rightarrow 3041.0$
   $w_7 \rightarrow -2008.0$
   $w_8 \rightarrow 5501.0$
   $w_9 \rightarrow 9662.399999998896$
   $w_{10} \rightarrow 3846.0$
   $w_{11} \rightarrow 14912.0$
   $w_{12} \rightarrow 21058.0$

   b) The number of training epochs required
   The training took 1975 epochs.

   c) The margin
   $\gamma \rightarrow 2.206549570044712 * 10^{-5}$

   (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: 448.1200000013341

(d) **Application to data set 3**
Classifications:
$0 \rightarrow 0.0$
$1 \rightarrow 1.0$
$2 \rightarrow 0.0$
$3 \rightarrow 1.0$
$4 \rightarrow 1.0$
$5 \rightarrow 0.0$
$6 \rightarrow 0.0$
$7 \rightarrow 0.0$
$8 \rightarrow 0.0$
$9 \rightarrow 1.0$
$10 \rightarrow 1.0$
$11 \rightarrow 0.0$
$12 \rightarrow 1.0$
$13 \rightarrow 0.0$
$14 \rightarrow 0.0$
$15 \rightarrow 0.0$
$16 \rightarrow 1.0$
$17 \rightarrow 0.0$
$18 \rightarrow 0.0$
$19 \rightarrow 0.0$
$20 \rightarrow 0.0$

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	CorrelationCoefficient
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, and zipping two vectors into one array of Pairs. This is used to traverse the arrays in parallel.

- 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    @SuppressWarnings("unused")
8    public static void main(String[] args) {
9      final String dataSet1Csv = "DataSet1.csv";
10     final String dataSet2Csv = "DataSet2.csv";
11     final String dataSet3Csv = "DataSet3.csv";
12
13     DataSet dataSet1 = new DataSet(dataSet1Csv);
14     DataSet dataSet2 = new DataSet(dataSet2Csv);
15     DataSet dataSet3 = new DataSet(dataSet3Csv);
16
17     WidrowHoffLearner learner = new WidrowHoffLearner( dataSet1, 0.0 );
18     learner.trainWeightVector();
19
20     for( int i = 0; i < learner.getWeights().length; i++ ) {
21       System.out.println("w" + i + ": -> " + learner.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 + " \\rightarrow " + classifications.get(
           i) + " \\) \\\\" );
40     }
41
42     final List<Pair<String,Double>> coefficients = CorrelationCoefficient.
         calculate(dataSet3, classifications);
43     System.out.println("-----------------------------");
44     System.out.println("Correlation Coefficients");
45     System.out.println("-----------------------------");
46     String maxProp = "";
47     Double maxCorrelation = Double.MIN_VALUE;
```

```
48      System.out.println("\\( \\begin{array}{l|l}");
49      System.out.println("\\bf{Label} & \\bf{Correlation Coefficient} \\\\");
50      System.out.println("\\hline");
51      for(final Pair<String, Double> coefficient : coefficients ) {
52        System.out.println("\\text{ " + coefficient.getLeft().trim().replaceAll(
              "[^A-Za-z0-9]", " ") + " } & " + coefficient.getRight() + " \\\\");
53        if( coefficient.getRight().compareTo(maxCorrelation) > 0 ) {
54          maxProp = coefficient.getLeft();
55          maxCorrelation = coefficient.getRight();
56        }
57      }
58      System.out.println("\\end{array} \\)");
59      System.out.println();
60
61
62      System.out.println("-------------------------");
63      System.out.println("Maximum Correlation Prop");
64      System.out.println("-------------------------");
65      System.out.println("\\bf{ Property With Maximum Correlation} \\\\ ");
66      System.out.println("Label: " + maxProp.replaceAll("[^A-Za-z0-9]", " ") + "
            \\\\");
67      System.out.println("Correlation Coefficient: \\(" + maxCorrelation + "\\)"
            );
68    }
69
70  }
```

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,
         Double>>();
37     for( int i = 0; i < x.length && i < y.length; i++ ) {
38       zippedVector.add( new Pair<Double,Double>(x[i], y[i]) );
39     }
40     return zippedVector;
41   }
42
43   public static double[] unitVector( double[] x ) {
44     double[] copy = x.clone();
45     double norm = norm(copy);
46     return scale(x, 1.0 / norm );
47   }
48
49   public static double norm( double[] x ) {
50     double sum = 0;
51     for( int i = 0; i < x.length; i++) {
52       sum += Math.pow(x[i], 2.0);
```

```
53      }
54      return Math.sqrt(sum);
55    }
56  }
```

1.4 CorrelationCoefficient.java

```
1  package hw2.widrowhoff;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.List;
6
7  public class CorrelationCoefficient {
8
9    public static List<Pair<String,Double>> calculate(final DataSet dataSet,
         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 ) {
27         final double xDiff = x - mean;
28         final double yDiff = classifications.get(j) - classificationMean;
29
30         numeratorSum += xDiff * yDiff;
31         xDenominatorSum += xDiff * xDiff;
32         yDenominatorSum += yDiff * yDiff;
33
34         j++;
35       }
36
37       final Double correlationCoefficient = numeratorSum / Math.sqrt(
             xDenominatorSum * yDenominatorSum);
38       final Pair<String,Double> coefficientWithLabel = new Pair<String,Double
             >(label, correlationCoefficient);
39       coefficients.add(coefficientWithLabel);
40       i++;
41     }
```

```
42
43      return coefficients;
44    }
45
46    private static List< List<Double> > transform(double[][] data ) {
47      List<List<Double>> transformed = new ArrayList<List<Double>>();
48
49      for( int i = 0; i < data[0].length; i++ ) {
50        List<Double> column = new ArrayList<Double>();
51        for( int j = 0; j < data.length; j++ ) {
52          column.add(data[j][i]);
53        }
54        transformed.add(column);
55      }
56
57      return transformed;
58    }
59
60    private static double mean(final double[] array) {
61      double sum = 0.0;
62      for( int i = 0; i < array.length; i++ ) {
63        sum+= array[i];
64      }
65      return sum / (double) array.length;
66    }
67
68    private static double mean(final List<Double> array) {
69      double sum = 0.0;
70      for( int i = 0; i < array.size(); i++ ) {
71        sum+= array.get(i);
72      }
73      return sum / (double) array.size();
74    }
75  }
```

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[][] x;
10   private double alpha = 1;
11   private double threshold;
12   private double[] labels;
13   private double margin;
14
```

```
15    public WidrowHoffLearner( final DataSet dataSet, final double[] weights,
          final double threshold ) {
16      this.x = dataSet.exData;
17      this.labels = dataSet.exLabels;
18      this.threshold = threshold;
19    }
20
21    public WidrowHoffLearner( final DataSet dataSet, final double threshold ) {
22      this.x = dataSet.exData;
23      this.labels = dataSet.exLabels;
24      final int inputLength = x[0].length; // get input vector length less the
            value of the class
25      double[] weightsVector = new double[inputLength];
26      for( int i = 0; i < inputLength; i++ ) {
27        weightsVector[i] = 0;
28      }
29      this.weights = weightsVector;
30      this.threshold = threshold;
31    }
32
33    public void trainWeightVector() {
34      int iterations = 0;
35      while( true ) {
36        int errors = 0;
37        for( int i = 0; i < x.length; i++ ) {
38          final double[] inputs = x[i];
39
40          double err = labels[i] - percepW( inputs ); // use last value in array
                as label
41
42          // 0 - correct
43          // -1 - false positive
44          // 1 - false negative
45          if( err != 0.0 ) {
46            errors++;
47  //          double loss = -1.0 * err * VectorUtils.dotProduct(inputs, weights)
        ;
48
49            final double[] scaledInputs = VectorUtils.scale(inputs, alpha * err
                );
50            weights = VectorUtils.add(scaledInputs, weights);
51          }
52        }
53
54        if(errors == 0) {
55          break;
56        }
57
58        iterations++;
59      }
60
61      calculateMargin();
62
63      System.out.println("The training took " + iterations + " epochs.");
```

```
64      System.out.println("Threshhold: " + threshold);
65      System.out.println("Margin: " + margin );
66    }
67
68    public void testWeightVector(DataSet dataSet) {
69      int falseNegatives = 0;
70      int falsePositives = 0;
71      int truePositives = 0;
72      int trueNegatives = 0;
73      double totalLoss = 0.0;
74      for( int i = 0; i < dataSet.exData.length; i++ ) {
75        final double[] inputs = dataSet.exData[i];
76
77        final double percepResult = percepW( inputs );
78        final double err = dataSet.exLabels[i] - percepResult; // use last value
              in array as label
79
80        // 0 - correct
81        // -1 - false positive
82        // 1 - false negative
83        if( err > 0.0 ) {
84          falseNegatives++;
85          System.out.println("False Negative Found: ");
86          System.out.println("Index: " + i);
87          System.out.println("Inputs: " + vectorToString(inputs));
88          totalLoss += -1.0 * err * VectorUtils.dotProduct(inputs, weights);
89        } else if ( err < 0.0 ) {
90          falsePositives++;
91          System.out.println("False Positive Found: ");
92          System.out.println("Index: " + i);
93          System.out.println("Inputs: " + vectorToString(inputs));
94          totalLoss += -1.0 * err * VectorUtils.dotProduct(inputs, weights);
95        } else {
96          if( percepResult == 1 ) {
97            truePositives++;
98          } else {
99            trueNegatives++;
100         }
101       }
102     }
103
104     System.out.println("True Positives: " + truePositives );
105     System.out.println("True Negatives: " + trueNegatives );
106     System.out.println("False positives: " + falsePositives);
107     System.out.println("False negatives: " + falseNegatives);
108     System.out.println("Total loss: " + totalLoss);
109     System.out.println("Total items: " + dataSet.exData.length );
110   }
111
112   private String vectorToString( final double[] x ) {
113     StringBuilder stringBuilder = new StringBuilder();
114     stringBuilder.append("(");
115     for( int i = 0; i < x.length; i++ ) {
116       stringBuilder.append(x[i]);
```

```
117       if( i != x.length - 1) {
118         stringBuilder.append(", ");
119       }
120     }
121     stringBuilder.append(")");
122     return stringBuilder.toString();
123   }
124
125   public List<Double> applyWeightVector(DataSet dataSet) {
126     List<Double> classifications = new ArrayList<Double>();
127     for( int i = 0; i < dataSet.exData.length; i++ ) {
128       final double[] inputs = dataSet.exData[i];
129       double classification = percepW(inputs);
130       classifications.add(classification);
131     }
132     return classifications;
133   }
134
135   private void calculateMargin() {
136     margin = Double.MAX_VALUE;
137     final double[] unitWeightVector = VectorUtils.unitVector(weights);
138
139     for( int i = 0; i < x.length; i++ ) {
140       double[] inputs = x[i];
141
142       double dotProduct = Math.abs( VectorUtils.dotProduct(unitWeightVector,
              inputs) );
143       double norm = VectorUtils.norm(inputs);
144
145       double marginPart = dotProduct / norm;
146
147       margin = Math.min(margin, marginPart);
148     }
149   }
150
151   private double percepW( final double[] curInputVector ) {
152     double sum = threshold * -1;
153
154     sum += VectorUtils.dotProduct( curInputVector, weights);
155
156     if( sum > 0 ) return 1;
157     else if ( sum < 0 ) return 0;
158     else return 0;
159   }
160
161   public double[] getWeights() {
162     return weights;
163   }
164   public double[][] getX() {
165     return x;
166   }
167
168   public double getAlpha() {
169     return alpha;
```

```
170     }
171
172     public double getThreshold() {
173       return threshold;
174     }
175
176     public double[] getLabels() {
177       return labels;
178     }
179
180     public double getMargin() {
181       return margin;
182     }
183
184 }
```