Implementing a Perceptron in C#

Kieran Mulchrone

November 6, 2020

1 Introduction

In this note I examine the concept and formulation of an elementary machine learning/neural networks item called a perceptron. I look at the concepts and mathematics of the perceptron before discussing one way to implement the perceptron in C#.

2 A Case Study

Consider the following subset of the dataset contained in the file "data.csv":

ID	RPM	VIBRATION	STATUS
1	568	585	1
2	586	565	1
3	609	536	1
4	616	492	1
5	632	465	1
6	652	528	1
7	655	496	1
8	660	471	1
9	688	408	1
54	891	156	0
55	911	79	0
56	939	99	0

This is output from the analysis of a bunch of engines which are classified as good (STATUS = 1) or faulty (STATUS = 0). The full dataset is plotted in Fig. 1. It is quite clear that a line could be drawn to separate the distinct datasets. The job of a perceptron is to find the equation of such a line, or for higher dimensional problems the equation of the hyperplane, that divides the data. In the present example, the data are fully linearly seperable whereas in real applications this may not be the case. There are more advanced techniques which can deal with non-linear and complex boundaries between data classes. We can think of the dataset as having two inputs (RPM and VIBRATION) giving an output (STATUS).

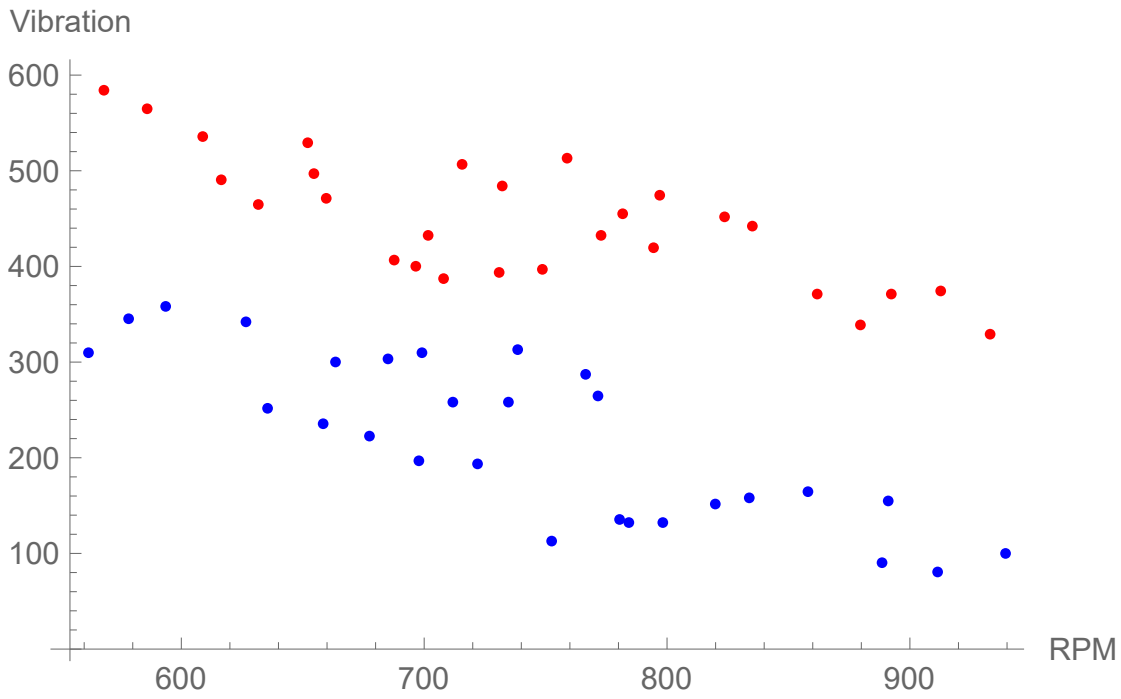

Figure 1: Plot of engine data. Red points are good engines and blue points are faulty engines.

3 So what is a Perceptron?

A perceptron is biologically motivated by a single neuron cell from the brain. It takes a series of inputs and generates an output. In our simple example the output is simply 0 or 1. A graphical representation is given in Fig.2. This is not the only way to mathematically represent a perceptron, but it leads too a nice compact notation. We have a vector of inputs $\mathbf{x} = \{x_0 = 1, x_1, \ldots, x_{n-1}, x_n\}$ and a vector of weights $\mathbf{w} = \{w_0, w_1, \ldots, w_{n-1}, w_n\}$, both of length $n + 1$. The sum of the inputs by the weights is calculated first as $\mathbf{x} \cdot \mathbf{w}$ (the scalar product). Then an activation function is applied. In this case a step function $\mathcal{H}(x)$ is used, defined as follows $\mathcal{H}(x) = 0$ if $x \leq 0$ and $\mathcal{H}(x) = 1$ if $x > 0$. Therefore the output of the perceptron is $\mathcal{H}(\mathbf{x} \cdot \mathbf{w})$.

Note that $x_0 = 1$ in $\mathbf{x}$ and the corresponding weight w_0 is a special weight termed the bias (and is often given the symbol b). Furthermore $\mathbf{x} \cdot \mathbf{w} = w_0 + x_1 w_1 + \ldots + x_n w_n$, in other words the bias is a constant term (much like the intercept in the equation of straight line). This gives flexibility to the perceptron to deal with lines and hyperplanes that do not contain the origin.

Given the weights and a set of inputs $(\mathbf{x}_i)$ where $i = 1, 2, \ldots, m$ for m inputs we can calculate the corresponding set of perceptron outputs $\hat{\mathbf{y}} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m\}$. Additionally we know the true set of outputs from the dataset $\mathbf{y} = \{y_1, y_2, \ldots, y_m\}$.

4 Training the Perceptron

We are now in a position to train the perceptron to find the set of weights that classifies each known input correctly. We begin by initialising the weight vector either to be the zero vector or we could assign random values, say between 0 and 1. We also choose a final parameter $0 < \alpha \leq 1$, termed the learning rate, a value of 0.1 or less is typical. The the perceptron algoirthm proceeds as follows:

1. set variable error = 0.
2. for each element of the input set as i goes from 1 to m
   (a) calculate $\hat{y}_i = \mathcal{H}(\mathbf{x}_i \cdot \mathbf{w})$
   (b) if $y_i \neq \hat{y}_i$ then update the weight vector by $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - \hat{y}_i)\mathbf{x}_i$ and set error $\leftarrow$ error + 1
3. if error > 0 go back to step 1, otherwise end.

If the data is linearly seperable this algorithm is guaranteed to converge. If the data are not linearly seperable it will never converge. Therefore it is advisable to have a limit to the number of iterations performed.

5 Coding the Perceptron: assignment

In this assignment you are to implement code in C# which classifies data according to the Perceptron approach. Specific requirements are:

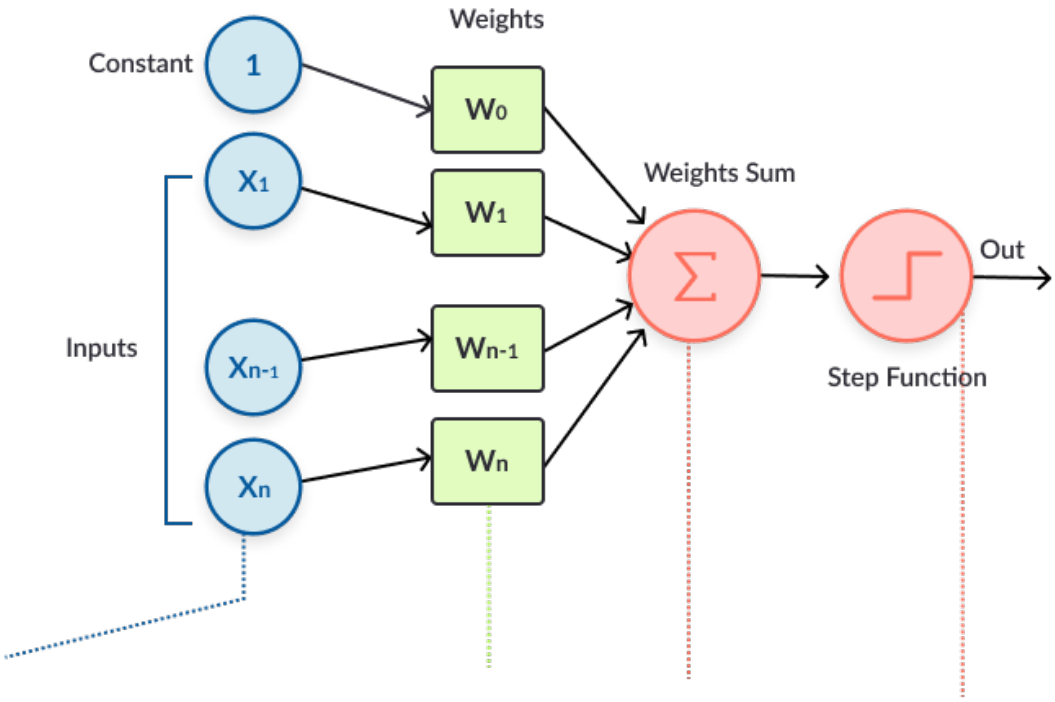

Figure 2: The form of a perceptron.

1. Create a Vector class which overloads the multiplication operator (*) to perform the scalar or dot product, and overloads the addition and subtraction operators to perform vector addition and subtraction.

2. Create a Perceptron which implements the following methods and makes use of the Vector class:

   (a) Contains a method
   public void ReadData(string filename);
   which reads in the contents of a csv file and stores the data in appropriate structures (your choice). Check for errors and the presence of a header line. Vectors etc should be sized based on the contents of the data file.

   (b) Contains a method
   public void TrainData();
   which finds the optimal weights (decision boundary).

   (c) Contains a method
   public int ClassifyPoint(Vector x);
   which classifies the point and returns either 1 or 0 as output.

   (d) Contains a method
   public void Output();
   which gives a nicely formatted summary of the weights etc.

3. Test your code with the following in the Main() method.
   Perceptron p = new Perceptron();
   p.ReadData("data.csv"); //file provided
   p.TrainData();
   p.Output();
   p.ClassifyPoint();//input depends on your implementation

4. Submit your answer by putting all code into a single file with an extension .cs. Check that your code runs and works before submitting. There will be marks allocated for tidy, readable and commented code.