Linear Discriminant Functions

Chapter 4 Machine Learning

- This is a multiparameter classification approach (the precursor to neural nets), that is based off of the equation:

  $$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$$

  where w^T is the weight vector, and w_0 is the bias (and x is just one feature vector)
- We can classify (in the binary case) based on whether g(x) > or < 0. For two points on the decision boundary, g(x)= 0. The equation can be rearranged to form

  $$\boldsymbol{w}^T(\boldsymbol{x_1} - \boldsymbol{x_2}) = 0$$

  This insinuates that the dot product of the two vectors is 0 and they are perpendicular.
  - As x_1-x_2 is a line on the decision boundary, it naturally follows that the weight vector for classification, is merely a perpendicular line to the decision boundary.
  - Further manipulation can show that the signed distance r, of a point from the decision boundary is given by

    $$r = \frac{g(\boldsymbol{x})}{|\boldsymbol{w}|}$$

    the key takeaway here is that you can tell how far into a class a point is by the magnitude of g(x) as its proportional to this distance r (higher g(x) = more confidently in a particular class)

Multi-Class Classification

- In the case where the classification is non-binary, simply make a linear discriminant function for each class, then classify by the g(x) that gives the largest value - this is a **linear machine**

  $$g_i(\boldsymbol{x}) = \boldsymbol{w_i}^T\boldsymbol{x} + w_{i,0}$$

Side note on high order classification using this method

- And extending onwards, if we want a non-linear decision boundary we simply switch to the following **generalised linear discriminant function**:

  $$g(\boldsymbol{x}) = \boldsymbol{a^T y}$$

  where y is the set of high order functions of x. e.g. y = (1, x, x^2)^T. Notice the bias term w_0 is now in this y vector

- Funnily enough, these generalised functions are very rarely used cause of their high dimensionality - Support Vector Machines are preferred in such a case.

Training Linear Discriminant Functions:

- Very similarly to before, given a sample matrix, X, of m rows of feature vectors:

$$g(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{w}$$

  where the bias term w_0 is incorporated into X (as an extra feature of magnitude 1) and w (as a weight w_0)
- From this we do typical l2 error metric shenanigans:

$$E_2 = \sum_i (\boldsymbol{w}^T \boldsymbol{x_i} - y_i)^2$$

  then minimise via gradient descent

Side note: sometimes it's useful to scale parameters via normalisation ~~(larger parameters may dominate purely because of their size leading small parameters to not affect the model)~~