Nearest Neighbour

The Nearest Neighbour for regression algorithm, is the simplest out of all the methods to be discussed throughout this report. As the name suggests, its ultimate goal is, given a new sample input, to iteratively compare it with the training data set, using a designated distance metric, and return the value that corresponds to the minimum distance. Assuming we have N exemplars in our training set, the sample will be compared N times before obtaining a prediction, which will be provided by the target value of the closest "neighbour". An extension to this method, is finding the k closest "neighbours" and returning the average of their target values as the prediction when compared with the sample input. This extension is called k-Nearest Neighbours (kNN), and it requires an extra step which sorts the distances in order to be able to collect the first k predictions. Following are the steps the algorithm takes to derive a prediction:

```
kNN (train data, test samples, k):
  1. Split the train data to train input and train output (target values)
  2. Find the number of samples in the test samples, assign in to N, and find the number of train
     input, assign it to M
  3. For I in test samples (1 to N) do:
     a. For j in train inputs (1 to M) do:
        i. Calculate Euclidean distance between i$^{th}$ test sample and j$^{th}$ train input and
           store it in a distance vector
     b. Sort the distance vector in order of increasing distances
     c. Derive the mean of the first k distances and store in a prediction vector
  4. Output prediction vector
```

One could figure out that since the algorithm contains a nested for loop and a sorting mechanism, it does not perform in a fast manner when the data set is of a large size. The next step after completing this procedure is to compare the predictions acquired with the actual target values of the test data to find the overall error, which will be discussed later on.

Hyperparameters: k
The number of nearest neighbours, k, to account when deriving the prediction is considered as the sole hyperparameter of the kNN approach as it is an immediate factor for the value to be returned.

Multivariate Linear Regression

When looking at a one-dimensional regression problem (a single feature), then using a simple linear regression is logical and quite fast. Considering that the SARCOS data set provides a multi-dimensional data to be used for training and predictions of dependent variable, the algorithm changes to fit feature vectors instead of single values. Looking back at Lab 3, we were assigned to produce a gradient descent algorithm that derives an optimal w, gradient, and c, y-intercept, values to produce the best prediction as seen in (1) below. The same notion stands for multivariate linear regression, but in this case all the coefficients, w_i, are stored in vectors and derived in a similar manner, (2) using dot products.

$$y = mx + c \tag{1}$$

$$Y = W^T X \tag{2}$$

Since this is a multi-dimensional regression problem, the values of each feature have different ranges. This would produce a problem when producing the linear regression algorithm as the coefficients stored in W could eventually influence the way we try to reach to an optimal solution. To avoid this problem we perform normalisation on the data, meaning we subtract the mean of each feature from the feature values, to make the data zero centred, and finally divide by the standard deviation, which would bring the range of the values of each feature to be symmetric in a form that resembles the following inequality $-1<x<1$. The algorithm to be used to find the optimal W vector, (a vector of gradients) is called the gradient descent algorithm. Scaling the feature values would allow gradient descent to converge faster and eventually be more efficient. Additionally, a column of ones is appended on our data to account for the value of c, when working with a multi-dimensional data set. In the case of the SARCOS dataset, this means that our W vector will consist of 22 values, instead of 21 which is the number of available features. In order for gradient descent to decide whether the W vector has converged, we need a cost function. Previously we used the Least Squares error as our cost function, with our goal being to minimize this error as we changed the gradient slightly based on a learning rate a, alpha, for a one-dimensional regression problem. To convert it to a vectorised cost function, equation (3) is used.

$$C(W) = \frac{1}{2m}(XW - Y)^T(XW - Y) \quad (3)$$

Where m is the size of the Y, target, vector and C(W) is our cost vector based on the current W, coefficient vector. Given these prerequisites, the gradient descent algorithm can be constructed as follows:

```
GradientDescent (train data, W, a, iterations):
    1. Split the train data to train input, X, and train output, Y, (target values)
    2. Find initial cost $C_0$ when W is a vector of zeros
    3. For 1 to number of iterations do:
        a. For j in range 1 to size of W:
            i. Update W by: $w_j = w_j - \frac{a}{m}X^T(XW^T - Y)$        (4)
        b. Find new cost value based on X, Y and new W vectors
        c. Check whether the algorithm converged based on the cost function
            i. If it has converged break the loop and return cost vector and W
            ii. Otherwise continue
    4. Output cost vector and W
```

Having produced the optimal coefficient vector W, based on the cost function, we can proceed with testing our linear regressor using equation (2), to obtain a prediction of target values, Y.

Hyperparameters: a, iterations
Both the learning rate, a, and the number of iterations affect the final prediction of the linear regression approach. Experimenting on these values, one can conclude that the number of iterations the gradient descent has to iteratively compute, is not very important if the algorithm converges before reaching that number. If the cost function has not converged before reaching the maximum number of iterations the error metric used is usually worst and it makes sense, since the gradient descent has not reached the optimal solution given that learning rate. The most important hyperparameter is the learning rate.