Matric number: s1632798

Task 1

Task 1.3

The elapsed time for my_knn_classify () is around 24.338572 seconds.

The requested information for every k from Kb:

Number of nearest neighbours	Number of test samples	Number of wrongly classified test samples	Accuracy
1	7800	1072	0.86256
3	7800	1037	0.86705
5	7800	1052	0.86513
10	7800	1130	0.85513
20	7800	1224	0.84308

Task 1.4

The k-nn algorithm is easy to implement using nested for-loops, because it is really straightforward – go through each row of the test samples and evaluate the distance to the training data matrix and classify it by the majority vote of its k neighbours. However, this implementation is very slow when it comes to big matrices. So we are using vectorization. It is not efficient at all in terms of memory, but it is very efficient in terms of speed.

Let Xtrn denotes the training data M-by-D matrix and let Xtst denotes the test data N-by-D matrix. Here we use a row vector to denote the feature vector of a data. We construct DI matrix which is the N-by-M distance matrix. Let DI_{ij} denotes the i-th row and the j-th column of the matrix DI. We use Euclidean distance to calculate the distance. The Eauclidean distance is given by:

$DI_{ij} = (Xtst_i - Xtrn_j)^2 = (Xtst_i - Xtrn_j) * (Xtst_i - Xtrn_j)^T$ where $(Xtst_i - Xtrn_j)^T$ denotes the transpose matrix of $(Xtst_i - Xtrn_j)$

$DI = (Xtst - Xtrn)^2 = (Xtst - Xtrn) * (Xtst - Xtrn)^T = Xtst * Xtst^T – 2 * Xtst * Xtrn^T + Xtrn * Xtrn^T$

$Xtst_{1..N} * Xtst_{1..N}^T$ is just the diagonal of $Xtst * Xtst^T$. The first element in the diagonal is the scalar product of the first row of Xtst with the first column of Xtst. The second element in the diagonal is the scalar product of the second row of Xtst with the second column of Xtst and so on.

So $Xtst_{1..N} * Xtst_{1..N}^T$ = test = sum(Xtst.*Xtst,2) – we need a column vector containing the sum of each row. And so we get an N-by-1 matrix. It is the same for the training data matrix where we get M-by-1 matrix - training = sum(Xtrn.*Xtrn,2). But the desired distance matrix is N-by-M so we replicate the test matrix M times and the training matrix N times and substitute them in the formula.

When we get the distance matrix, we sort it, take the nearest k entries and classify the test example. So even though that vectorization can be potentially buggy and requires a lot of memory, it is really fast compared to the nested for-loops.