Stats 598z: Homework 4

Due before class on Tue, Mar 19

Important:

`R` code, tables and figures should be part of a single .pdf or .html files from `R Markdown` and `knitr`. See the class reading lists for a short tutorial.
Include `R` commands for all output unless explicitly told not to.

If you collaborated with anyone else, mention their names and the nature of the collaboration

1 Problem 1: k-nearest neighbours [100pts]

(a) We are going to implement k-nearest neighbours. Recall how this works: you're given some training data with labels. Given a new test datapoint, you decide its label by performing a majority vote among its k-nearest neighbours. Write down a skeleton or outline of how you might implement this (in `R` or in English/pseudocode). This needn't bear any similarity to your final program, but you should have an idea of all the pieces you'll need and how they fit together. [10pts].

(b) Install package `RnavGraphImageData` and use the `data()` function to load the `digits` dataset: `data(digits)`. This dataframe contains the so-called USPS dataset, with each column having length 256, and representing an 16×16 image of a handwritten digit. See `?digits` details. (WARNING: for some reason, digits 6 and 7 are overwritten by 5, don't let that confuse you later) [2pts]

(c) It's helpful to visualize the digits, and we will write a function `plot_digit` to do this. It should take a vector (a column of `digits`) as input, and plot it using the `image()` function. For this you'll have to convert it to a 16×16 matrix (experiment with the `byrow` option of `matrix`). You can also play with the `color` argument of `image`, I used `col = gray(0:255/255)`. [10pts]

(d) What are the index ranges of each digit in `digits`? Plot the first instance of each digit (since `image` doesn't involve `ggplot` you don't have to organize it in a panel). However, instead of calling your function 10 times manually, write a for loop (or use `*pply`). Careful: the order of digits in `digits` is `1,2,…,9,0` rather than `0,1,2,…`. You should notice that digits 6 and 7 are just 5. [10pts]

(e) Write a function `get_digits` to convert `digits` to a smaller, more convenient dataset. `get_digits` should take two arguments: the first is a vector `select_digs` and the second is `size`. The function should return the first `size` elements of each digit in `select_digs`. Thus if `get_digits` is `c(0,5,4)` and `size` is `50`, then the function should return a data frame of length 150, consisting of the first 50 images of 0, 5 and 4. Create a dataset `my_train` consisting of 100 instances of 0 and 8. [10pts]

(f) Write a function `euc_dist` that accepts two vectors (of any length), and returns the Euclidean distance between them. You can normalize this by the vector length if you want. [8pts]

(g) Given any new digit, we want to calculate the distance to every element of `my_train`. Use `laply` from package `plyr`. `laply` accepts a list as input and returns an array obtained by appling some function `fun` to each element of that list. Recall that `my_train` is a dataframe which is just a list. If the function needs more than one argument you can pass those to `laply`, see the documentation and recall how '…'

works in functions. You overall syntax with look like `my_arr <- laply(my_list, fun, second_arg)` where you plug in appropriate variable names. [10pts]

(h) Now write a function `get_knn` that takes 3 inputs: a number k, the training data, and a single test-vector. It should return the indices of the k nearest neighbours of the test-vector in the training data. It does this in three steps: first calculate the distance to each element in the training set, and then sort them using the `sort` command. Calling `sort` with the option `index.return` set to `TRUE` also returns the indices of the sorted elements. Finally, return the first k indices. [8pts]

(i) Now write a function to get the majority label of the returned k indices. For this, it is useful to complement `my_train` from step (e) with a vector `my_labels` containing the corresponding labels. [8pts]

(j) Wrap the previous few functions into a function `my_knn` that takes four inputs: `k, my_train, my_labels` and `test_ip`, and returns the predicted label of `test_ip` from applying k-nearest neighbours. [8pts]

(k) Set $k = 5$. Apply your function setting `test_vec` to each element of `my_train`. How many 0's does it get wrong? How many 8's? [8pts]

(l) Apply your function to a hundred 0's and 8's from `digits` NOT present in `my_train`. How many of these does it get wrong? Apply it to one hundred 5's. How many of these does it classify as 0, and how many as 8? [8pts]