Week 11 - Assessed Exercises

Data Programming with Python

In these assessed exercises, we're going to perform some model comparison on a handwriting recognition multi-class data set. We're going to divide it up into training, validation and test sets. We're going to run different parameter values on the training and validation sets to determine the optimal parameters for the classifiers. Then we're going to use the classifiers with the optimal parameters on the test set to compare the different classifiers types.

The classifiers we're going to use are:

- Random forests, `RandomForestClassifier`
- k nearest neighbours, `KNeighborsClassifier`
- Multi-layer perceptron (neural network), texttttMLPClassifier

The commands for loading them into Python can be found in the *.py* template.

Each question asks you to write a function with a specific set of input arguments. The *.py* template defines the function name and inputs for each question, **do not** change these. Be sure you test your functions before you submit your code to make sure that they are outputting the correct answer. Unless otherwise stated, all functions must return a value.

The digits handwriting data set is part of the sklearn datasets module. The data concern handwriting recognition so the response is a digit (0 to 9) and the explanatory variables are levels of grey on an 8 by 8 grid. The *.py* template contains an example of how to visualise the data. Note that your functions should generalise to other datasets. Other datasets in the sklearn package that you could use include the iris dataset, the wine dataset and the breast cancer dataset.

The *.py* template includes a function for creating the training, validation and test set for a given matrix of observations `X` and vector of responses `y`. You will need to use this when testing your functions.

1. Write a function that runs each of the three classifiers with their default parameter values. The function inputs are the training and test sets `X_train`, `X_test`, `y_train`, `y_test` and a seed value `s`. The seed value should be used as the `random_state` argument in `RandomForestClassifier` and `MLPClassifier`. The function should return a `dict` with keys *knn*, *rf* and *svm*. The values should be the misclassification rate for each classifier (rounded to 3dp). Remember that there are more than two categories, so your misclassification table will have more rows and columns to interpret.

Each of the above models has key parameters which we might like to estimate. For example, we might want to estimate the *best* number of neighbours to use in kNN. To do this, we fit kNN with different values of k to the training set and evaluate the performance of each model using the validation set. The k value that gives the best performance on the validation data is chosen as the best model. We then evaluate the performance of this model on data the classifier hasn't seen before, the test set.

2. Write a function that determines the *best* number of neighbours k to use in the kNN classifier and evaluates the performance of the best model on the test set. The function inputs are the training, validation and test sets and a list of values of k to try. The function should return a `dict` with the best k value (key: k) and the misclassification rate for the test set (key: MR) (rounded to 3dp). Ensure that you use these exact keys.

3. Write a function that determines the *best* number of trees (`xn_estimators`) to use in the random forest classifier and evaluates the performance of the best model on the test set. The function inputs are the training, validation and test sets, a list of values of `n_estimators` to try and a seed value `s` to use as the `random_state` for the classifier. The function should return a `dict` with the best number of trees (key: $Trees$) and the misclassification rate for the test set (key: MR) (rounded to 3dp). Ensure that you use these exact keys.

4. The parameter we wish to estimate for the multi-layer perceptron classifier is the number of neurons in the hidden layers of the neural network. To change this parameter include `hidden_layer_sizes=num_neurons` as an input to the `MLPClassifier` function. Write a function that determines the *best* number of neurons in the multi-layer perceptron classifier and evaluates the performance of the best model on the test set. The function inputs are the training, validation and test sets, a list of values of `hidden_layer_sizes` to try and a seed value `s` to use as the `random_state` for the classifier. The function should return a `dict` with the best number of neurons (key: $Neurons$) and the misclassification rate for the test set (key: MR) (rounded to 3dp). Ensure that you use these exact keys.

All of your code should be written into the *.py* template. Save your filled *.py* file with the following name structure *SurnameFirstname_Week11.py* (where *Surname* and *Firstname* should be replaced with your name) and upload it to Brightspace. You must also upload a PDF of your code.