Assignment 1: Classification of MNIST Handwritten Digits

At a high level, you will be building and evaluating different classifiers for recognizing handwritten digits of MNIST dataset. At a more granular level, you will be doing the following tasks:

1. **Binary Classification of MNIST Dataset:** For the first set of tasks, you will evaluate a few popular classifiers to recognize handwritten digits from the MNIST dataset. Specifically, we will focus on distinguishing between 7 and 9 which are known to be a hard pair. We will not use any sophisticated ideas from Computer Vision/Image Processing and use classifiers directly over the data. The idea is to show that lot of times, you can simply run a set of classifiers and still get great results. While I will be giving some basic classifier code, you will have some opportunity to improve them by tuning the parameters. Note that all the helper code can be found in the *mnist_assignment_starter.py* python file. The template code is provided purely to help you get started and you are free to move code around as you please.

   A. We will start by making sure your python environment is configured correctly and that you have all the required packages installed. For information about setting up python please consult the following link: https://www.anaconda.com/products/individual. To test that your environment is set up correctly, simply import and print the *mnist_assignment_starter.py* module. You may also want to run the file as a script.

   B. Load and prepare the *mnist* dataset, i.e., call the *prepare_data* and filter_out_7_9s functions, as a data matrix X consisting of only 7s and 9s. Make sure that every element in the data matrix is a floating point number and scaled between 0 and 1. Also check that the labels are

Where is the data downloaded? Path?

integers. Print out the length of the filtered X and y, and the maximum value of X for both training and test sets.

C. Train your first classifier using k-fold cross validation (see *train_simple_classifier_with_cv function*). Use k=5 and a Decision tree classifier. Print the mean and standard deviation for the accuracy scores in each validation set in cross validation. Also print the mean and std of the fit (or training) time.

D. Repeat Part C with a random permutation cross-validator.

E. Repeat part D for k=2,5,8,16, but do not print the training time. Note that this may take a long time (2-5 mins) to run. Do you notice anything about the mean and/or standard deviation of the scores for each k?

F. Repeat part D with another classifier of your choice (e.g., logistic regression, support vector machine). Make sure the train test splits are the same for both models when performing cross-validation. Which model has the highest accuracy on average? Which model has the lowest variance on average? Which model is faster to train?

G. For the classifier you trained in part F, manually (or systematically i.e., using grid search) modify hyperparameters, and see if you can get a higher mean accuracy. Finally train the classifier on all the training data (e.g., without cross-validation) and get an accuracy score on the test set. Print out the training and testing accuracy and comment on how it relates to the mean accuracy when performing cross validation. Is it higher, lower or about the same?

2. **Multi-Class Classification of MNIST Dataset:** In the second set of tasks, we will do multi-class classification where the idea is to classify an image as one of ten digits (0-9).

A. Repeat part 1.B but make sure that your data matrix (and labels) consists of all classes by also printing out the number of unique elements in y.

B. Repeat part 1.C, 1.D, 1.F, and 1.G for the multi-class problem, but use a different classifier than you did in part 1.F. Make sure that the classifier supports *multi class* (either inherently or with a certain configuration). Specify how you handled the multi class problem, i.e., inherent, one-vs-one, one-vs-the-rest, etc.

C. For the final classifier you trained in 2.B. Plot a confusion matrix for the test predictions. Earlier we stated that 7s and 9s were a challenging pair to distinguish. Do your results support this statement? Why or why not?

3. **Exploration of Different Evaluation Metrics:** In the first two set of tasks, we will narrowly focus on accuracy - what fraction of our predictions were correct. However, there are several popular evaluation metrics. You will learn how (and when) to use these evaluation metrics.

   A. Using the same classifier and hyperparameters as the one used at the end of part 2.B. Get the accuracies of the training/test set scores using the top_k_accuracy score for k=1,2,3,4,5. Make a plot of k vs. score and comment on the rate of accuracy change. Do you think this metric is useful for this dataset?

   B. Repeat part 1.B but return an imbalanced dataset consisting of 99% of all 9s removed. Also convert the 7s to 0s and 9s to 1s.

   C. Repeat part 1.C for this dataset but use a support vector machine (*SVC* in sklearn). Make sure to use a stratified cross-validation strategy. In addition to regular accuracy also print out the mean/std of the f1 score, precision, and recall. Is precision or recall higher? Explain. Finally, train the classifier on all the training data and plot the confusion matrix.

   D. Repeat the same steps as part 3.C but apply a weighted loss function (see the *class_weights* parameter). Print out the class weights, and comment on the performance difference. Tip: *compute_class_weight*