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:

Preliminaries

- You are given the file `utils.py` with some functions you will need and others you might find useful. Do not change
  this file. If there is an error, you can report it to me and the TA, and we'll tell you what to do.
- There is a file `new_utils.py` for you to add your own functions that you can reuse across classes.
- There are files run_part_1.py through run_part_3.py, one per part of this assignment. You are NOT to change these files. Your program will execute if you run them. They also generate a pickle file with all your answers, which I might or might not use.
- You are provided three files in which to produce your code:
  `part_1_template_solution.py` through `part_3_template_solution.py`. There files can import the `new_utils` if you add your own functions.
- In each class file, there are instructions for each questions. Typically, you will return a python dictionary with specified keys and structure.
- Follow the instructions in the three class files.
- There is report to write on this first homework. Some of the dictionary items are explanatory strings. The strings can be arbitrarily long. In the future, these will be included in a report. Sometimes, you are asked to print out various results. Do so to stdout. We might run your code
- Please email us the link to your solution repository. We will use this to collect information and check your code manually if necessary.

Part 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; instead, we apply classifiers directly over the data. The idea is to demonstrates that one can often execute a set of classifiers and still get excellent results. While I will be giving some basic classifier code, you will have some opportunity to improve them by tuning the parameters. All the helper code is located in the Python file *mnist_assignment_starter.py*. 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 ensuring that 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 execute `starter_code` in the `utils` module. This is done for you.

```
- Alternatively, install poetry, which allows a different python environment for e
- To use poetry:
    - download and install poetry
    - Execute: *poetry new assignment_1*
    - This generates a new folder `assignment_1/` in your current folder with a ce
      structure.
    - Add the required modules:
        - poetry add numpy sklearn
```

B. Load and prepare the *mnist* dataset, i.e., call the prepare_data and filter_out_7_9s functions in utils.py, to obtain a data matrix X consisting of only the digits 7 and 9. Make sure that every element in the data matrix is a floating point number and scaled between 0 and 1 (write a function `def scale() in new_utils.py` that returns a bool to achieve this. Checking is not sufficient.) Also check that the labels are integers. Print out the length of the filtered X and y, and the maximum value of X for both training and test sets. Use the routines provided in utils. When testing your code, I will be using matrices different than the ones you are using to make sure the instructions are followed.

C. Train your first classifier using k-fold cross validation (see train_simple_classifier_with_cv function). Use 5 splits and a Decision tree classifier. Print the mean and standard deviation for the accuracy scores in each validation set in cross validation. (with k splits, cross_validate generates k accuracy scores.) Remember to set the random_state in the classifier and cross-validator.

D. Repeat Part C with a random permutation (Shuffle-Split) k-fold cross-validator. Explain the pros and cons of using Shuffle-Split versus k-fold cross-validation.

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 a Random-Forest classifier with default parameters.

```
- Make sure the train test splits are the same for both models when performing cro
- Which model has the highest accuracy on average?
- Which model has the lowest variance on average? Which model is faster to train?
- Make sure your answers are calculated and not copy/pasted. Otherwise, the automa
- Use a Random Forest classifier (an ensemble of DecisionTrees).
```

. G. For the Random Forest classifier 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 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?

```
Choose among the following hyperparameters:
  1) criterion,
  2) max_depth,
  3) min_samples_split,
  4) min_samples_leaf,
  5) max_features
```

Part 2: Multi-class classification

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 10 classes by also printing out the number of elements in each class y and print out the number of classes for both training and testing datasets.

B. Repeat part 1.C, 1.D, and 1.F, for the multiclass problem.

```
- Use the Logistic Regression for part F with 300 iterations.
- Explain how multi-class logistic regression works (inherent, one-vs-one, one
- Repeat the experiment for ntrain=1000, 5000, 10000, ntest = 200, 1000, 2000.
- Comment on the results. Is the accuracy higher for the training or testing s
- What is the scores as a function of ntrain.

Given X, y from mnist, use:
- Xtrain = X[0:ntrain, :]
- ytrain = y[0:ntrain]
- Xtest = X[ntrain:ntrain+test]
- ytest = y[ntrain:ntrain+test]
```

Part 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. Do the following:

```
- Using the same classifier and hyperparameters as the one used at the end of part
- Get the accuracies of the training/test set scores using the top_k_accuracy scor
- Make a plot of k vs. score for both training and testing data and comment on the
- Do you think this metric is useful for this dataset?
```

B. Repeat part 1.B but return an imbalanced dataset consisting of 90% 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, print out the mean/std of the F1 score, precision, and recall. As usual, use 5 splits.
- Is precision or recall higher? Explain. Finally, train the classifier on all the training data and plot the confusion matrix.
- Hint: use the make_scorer function with the average='macro' argument for a multiclass dataset.

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.
- Use the `compute_class_weight` argument of the estimator to compute the class weights.