

Automated Classification of Handwritten Digits

ISTA 331 Hw7, Due 11/21/2019 at 11:59 pm

Introduction. The MNIST dataset, contained in `mnist-original.mat` (matlab format), consists of 70,000 digitized handwritten digits and their labels. We are going to classify them using two different classifiers, a stochastic gradient descent classifier called `SGDClassifier` (we will talk about this in class) and a logistic regression classifier called `LogisticRegression`. We will evaluate our results visually with a graphical depiction of a confusion matrix, which we will also cover in class. **numpy's random state functionality has extensive issues! Do not expect the same results if you call a function vs. doing each step of the function one line at a time in the interpreter, even if you set the seed the same beforehand.** Just fyi for the debugging stage.

Instructions. Create a module named `hw7.py`.

Start your module with the following imports, then code up the 7 functions as specified below, and upload your module to the D2L Hw7 assignments folder.

```
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
```

Testing. Download `hw7_test.py`, `mnist-original.mat`, and auxiliary testing files and put them in the same folder as your `hw7.py` module. Each of the 6 functions not including `main` and `plot_probobality_matrices` is worth 16% of your correctness score. `main` and `plot_probobality_matrices` is worth 20% of your grade. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, ISTA 331 Hw7, and a brief summary of the module. Each function must contain a docstring. Each docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.

<https://www.openml.org/search?type=data>

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

`get_data`: this function takes no arguments. Open a shell and type in these commands:

```
import scipy.io as sio
mnist = sio.loadmat('mnist-original.mat')
```

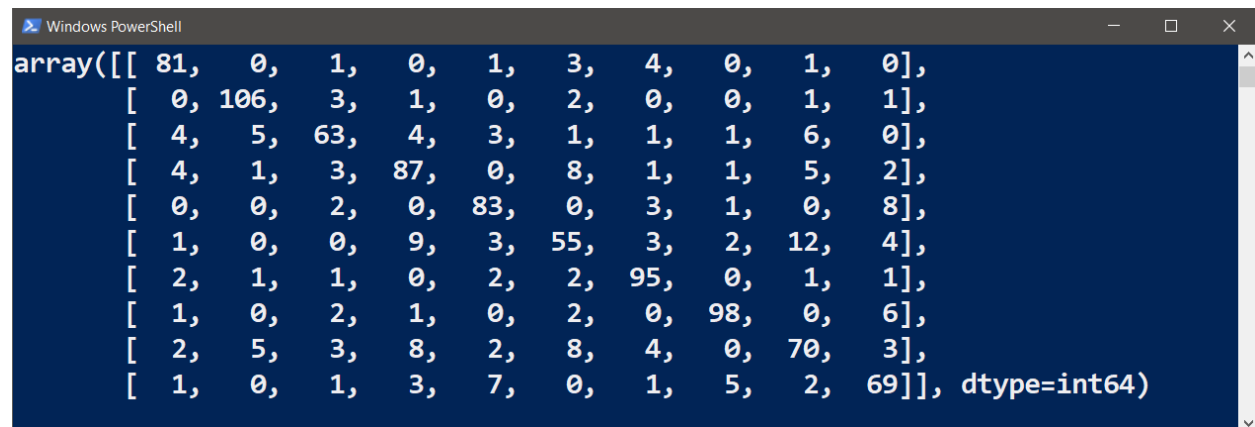
Of course your current working directory will have to contain the file. There are other ways that you might figure out to load the data, but bear in mind that if your code won't run because a server is down, you get 0 points. Explore the `mnist` variable to find what you need to return – your `X` and `y`. `X` should be a 70,000 x 784 2D array, `y` should be a 1D array with 70,000 elements in it.

`get_train_and_test_sets`: this function takes `X` and `y` as created by the previous function. The first 60,000 instances of each are for training, the rest for testing. The function breaks both into separate training and testing `X`'s and `y`'s. It uses `np.random.permutation` to get a shuffled sequence of indices and then uses these indices to shuffle the training `X`'s and `y`'s. It then returns the training `X`, the testing `X`, the training `y`, and the testing `y`, in that order.

`train_to_data`: this function takes a training `X`, a training `y`, and a string containing the name of the model. If it's an `SGDClassifier`, make one with a maximum of 50 iterations and a tolerance of 0.001. Otherwise, make a `LogisticRegression` with a 'multinomial' `multi_class` and an 'lbfgs' solver. Fit the model to the data and return it.

`get_confusion_matrix`: this function takes a model, an `X`, and a `y`. Pass the results of 5-fold cross-validation predicting on the training data to the `confusion_matrix` constructor and return the result. You'll want to take a look at your `import` statements and consult the documentation to figure out how to do some of this stuff.

`probability_matrix`: this function takes a confusion matrix and returns a probability matrix. Do not mess up the original confusion matrix. Row `i` of the confusion matrix contains the predictions for all of the digits that were actually `i`. Here is a sample confusion matrix:

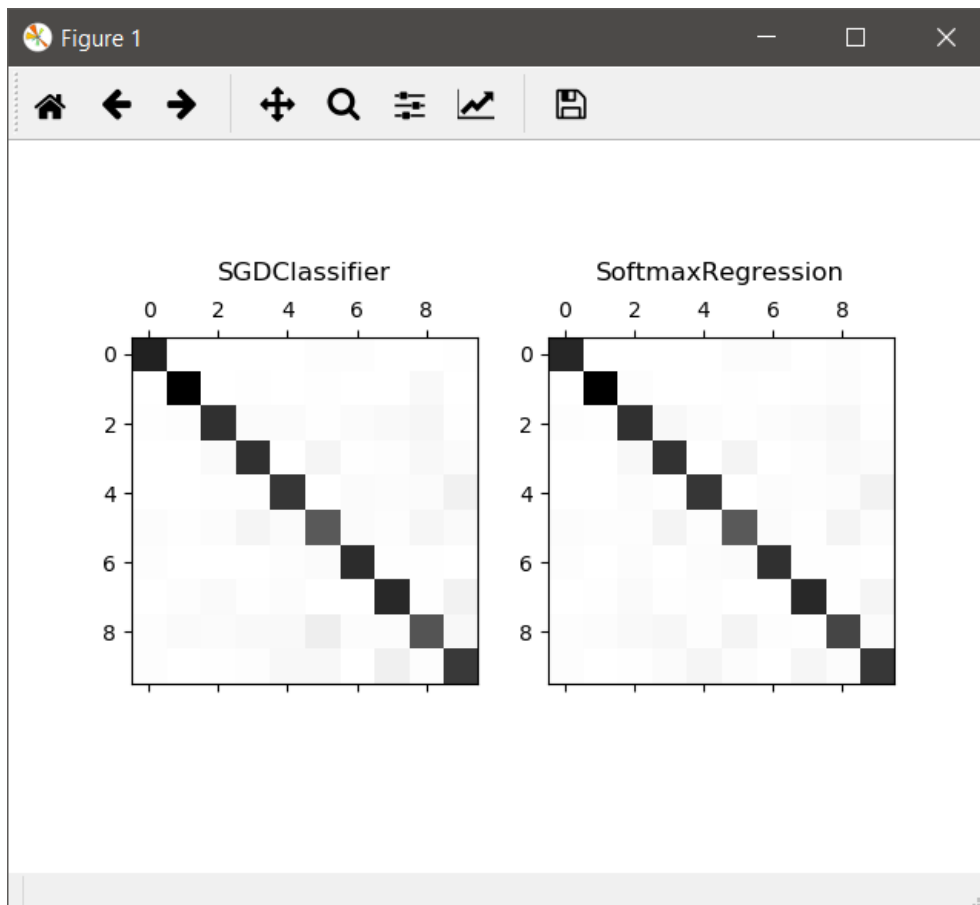


```
array([[ 81,  0,  1,  0,  1,  3,  4,  0,  1,  0],
       [  0, 106,  3,  1,  0,  2,  0,  0,  1,  1],
       [  4,  5, 63,  4,  3,  1,  1,  1,  6,  0],
       [  4,  1,  3, 87,  0,  8,  1,  1,  5,  2],
       [  0,  0,  2,  0, 83,  0,  3,  1,  0,  8],
       [  1,  0,  0,  9,  3, 55,  3,  2, 12,  4],
       [  2,  1,  1,  0,  2,  2, 95,  0,  1,  1],
       [  1,  0,  2,  1,  0,  2,  0, 98,  0,  6],
       [  2,  5,  3,  8,  2,  8,  4,  0, 70,  3],
       [  1,  0,  1,  3,  7,  0,  1,  5,  2, 69]], dtype=int64)
```

The first row represents the predictions for all of the digits that were actually 0. So 81 0's were correctly identified, one 0 was identified as a 2, one as a 4, three as 5's, etc. Make and return a new matrix of floats where each number in position $[i, j]$ is the estimated conditional probability that j was predicted given that i was the label (correct value). Round your probabilities to 3 decimal places. For the above example, this is part of the probability matrix:

```
Windows PowerShell
array([[0.89 , 0.   , 0.011, 0.   , 0.011, 0.033, 0.044, 0.   , 0.011,
        0.   ],
       [0.   , 0.93 , 0.026, 0.009, 0.   , 0.018, 0.   , 0.   , 0.009,
        0.009],
       [0.045, 0.057, 0.716, 0.045, 0.034, 0.011, 0.011, 0.011, 0.068,
        0.   ],
       [0.036, 0.009, 0.027, 0.777, 0.   , 0.071, 0.009, 0.009, 0.045,
        0.018],
       [0.   , 0.   , 0.021, 0.   , 0.856, 0.   , 0.031, 0.01 , 0.   ,
        0.082],
```

`plot_probobality_matrices`: this function takes two probability matrices and that looks like this when displayed (do not call `plt.show` in this function in your submitted version):



You will want to use `plt.subplots` and `axes.matshow`. This plot is worth 20 pts. It will show up when you run the test, but the test doesn't grade it. The test calls your `main`, which must work right for the plot to show up.

`main`: get your data, split it into training and testing sets, and train a `SGDClassifier` and a `LogisticRegression` model. Get confusion matrices for each of them, then probability matrices. Make your matrices plot. Put these lines in your code, which will print your matrices in a sweet format:

```
for mod in (('SGDClassifier:', probability_matrix(sgd_cmat)),
            ('Softmax:', probability_matrix(soft_cmat))):
    print(*mod, sep = '\n')
```

Call `plt.show`.