Digit Classification

You are given data (data_digits_8_vs_9_noisy) corresponding to images of handwritten digits (8 and 9 in particular). Data has been split into various training, and testing sets; each set is given in CSV form, and is divided into inputs (x) and outputs (y).

Each row of the input data consists of pixel data from a (28 × 28) image with gray-scale values between 0.0 (black) and 1.0 (white); this pixel data is represented as a single feature-vector of length $28^2 = 784$ such values. The output data is a binary label, with 0 representing an 8, and 1 representing a 9.

1. Fit logistic regression models to the training data using sklearn's implementation of the model, with the liblinear solver. Leaving all other parameters with default values, you will explore what happens when we limit the iterations allowed for the solver to converge on its solution.

   For the values i = 1,2,...,40, build a logistic regression model with the max_iter set to i. Fit each such model to the training data, and keep track of the accuracy of resulting model (via the model's own score() function) along with the log loss of that model on the training data[†].

   Produce two plots, each with the values of i as x-axis and with the accuracy/loss, respectively, as y, with **captions** labeling each appropriately. Write to discuss the results you see; what do they show, and why?

2. After fitting a logistic model, you can access weights it assigns to each feature in the data using its "coef_" attribute. For each of the i models you generated, record the first such weight, which is the one the model applies to feature pixel000 in the input data. Produce a plot with the values of i as x-axis and with the feature weight as y, with a caption labeling it appropriately. Write to discuss the results you see; what do they show, and why?

[†]When doing this, you will probably see warnings about non-convergence for lower values of parameter i. You can ignore these warnings, as they are expected.

3. You will explore different values of the inverse penalty strength C. Your code should explore a range of values for this last parameter, using a regularly-spaced grid of values:

```
C_grid = np.logspace(-9, 6, 31)

for C in C_grid:
 # Build and evaluate model for each value C
```

For each such value of C create a model and fit it to the training data, and then compute the log loss of that model on the test data. Determine which value gives you the least loss on the test data. Record that value, along with the accuracy score of the model. Also include a table for the confusion matrix of that model on the test data.

4. Analyze some of the mistakes that your best model makes. Produce two plots, one consisting of 9 sample images that are false positives in the test set, and one consisting of 9 false negatives. You can display the images by converting the pixel data using the matplotlib function imshow(), using the Grey colormap, with vmin=0.0 and vmax=1.0, captioned figure. Write to discuss the results you are seeing. What mistakes is the classifier making?

5. Analyze all of the final weights produced by your classifier. Reshape the weight coefficients into a (28 × 28) matrix, corresponding to the pixels of the original images, and plot the result using imshow(), with colormap RdYlBu, vmin=-0.5, and vmax=0.5m with a properly captioned figure. Write to discuss what it shows. Which pixels correspond to an 8 (have negative weights), and which correspond to a 9 (have positive weights)? Why do you think this is the case?

Sneakers versus Sandals

You are provided some image data, in the same format as before, of sneakers (output label 0) and sandals (output label 1). You are given input and output data for a training set, along with input data only for a test set. Your task is to build a logistic regression classifier for this data. You will describe the approach you took and results you saw.

When doing regression, you should explore different feature transformations, transforming the input features (in any way you see fit) that are given to the regression classifier. You are free to use ANY feature transform you can think of to turn the image into a feature. Be creative in thinking about how to transform data. You should try a few and here are what I could think.

- Consider things like histograms of parts of the data.
- Consider adding features that count overall numbers of white or black pixels.
- Consider adding features that capture spatial patterns in the original data.
- Explore data augmentation, where you add to your data set via transformations of the data. For example, if you flip each image horizontally, you can double the training set size without needing fundamentally new data.

You should explain what feature transformations you tried, and what processes you used to build your classifier (parameters you tried, etc.). Your work should contain at least two figures comparing the results you get by regression using the original features of the data and some modified features. Your discussion should include the error rate on the testing data for various models. Generate the predicted probabilities like what you did before in the testing input set in the form of a text-file, yproba1_test.txt, containing one probability value (the probability of a positive binary label, 1) per example in the test input. Code like the following can be used to produce that file: x_test = np.loadtxt('data_sneaker_vs_sandal/x_test.csv')

```
yproba1_test = model.predict_proba(x_test)[:, 1]

np.savetxt('yproba1_test.txt', yproba1_test)
```