Springboard DSC Capstone Project II

Handwritten Character Recognition Using Machine Learning

Prepared by Alexander Olden

November 2017

Introduction to the Problem

Image analysis is a good entry point for learning about the use of neural networks, which I aim to do in this project. Because image data is typically harder to work with—compared to working with textual data—more sophisticated forms of analysis often perform better. That said, massive computational power is not necessarily needed, as neural networks can perform the desired analysis. The problem to be solved in this project is that of recognizing single handwritten digits from images that have been captured following a uniform method. This work is therefore an instance that belongs to the general class of image recognition problems.

Data Attributes and Preparation

The dataset used in this project is the MNIST data, which I accessed through fetch_mldata in the sklearn.datasets library of Python. [1] The dataset's observations have two components. The first is the initial image of a handwritten digit, which is 28x28 pixels. Flattening this array produces a vector of 28x28 = 784 numbers. Each value is a floating-point number between 0 and 1, which represents the level of intensity of the pixel. (See Figure 1 on the next page.) [2]

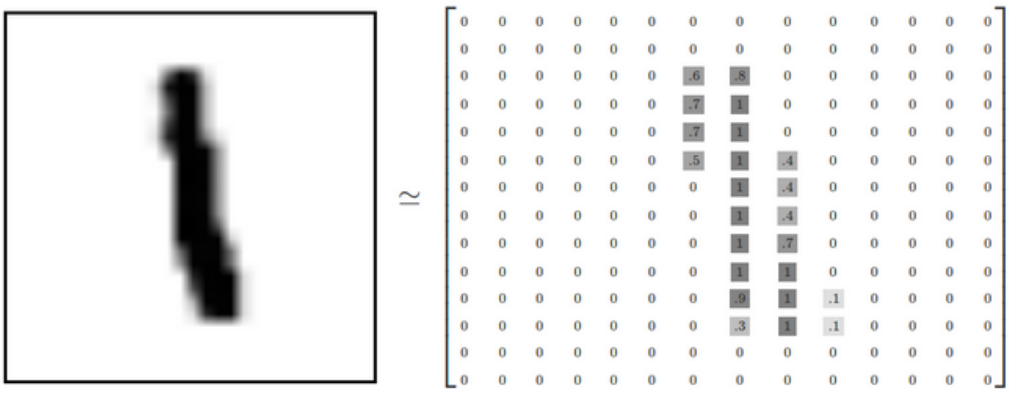


Figure 1: Translation of pixelated image into flattened vector

The collection of these 70,000 flattened matrices forms the design matrix in my analyses. More precisely, each row of the design matrix is a 1x784 vector, which corresponds to the flattening of a digit image. The design matrix, therefore, has 70,000 rows and 784 columns. The collection of all labels identifying the digit associated with each image forms a vector of 70,000 integer values between 0 and 9, which serves as the dependent variable. Because the dataset here is already arranged in this manner, no preprocessing was needed. In most real-world analyses, preprocessing would be needed, however.

Before proceeding to the analysis, I produced a bar chart to check that the numbers of labels in each of the 10 classes were reasonably balanced. As Figure 2 shows, the set of labels meets that criterion.

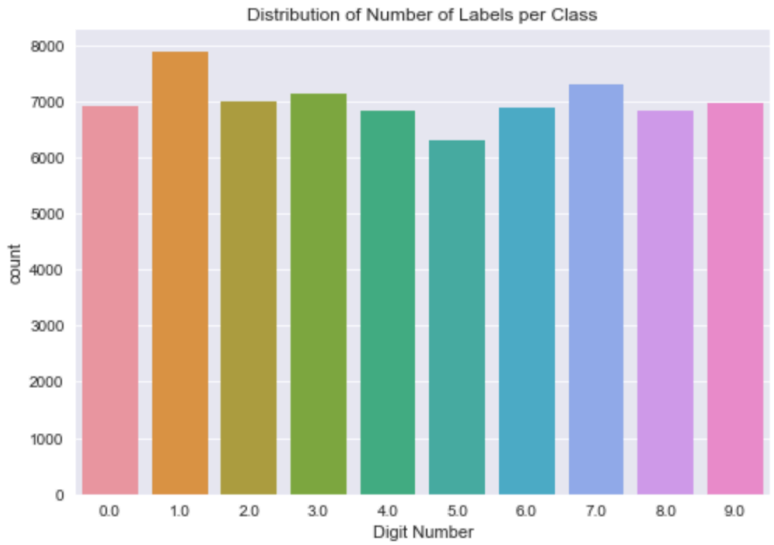


Figure 2

Logistic Regression and Results

As a baseline for comparison to results from neural networks, I first ran a logistic regression, which is commonly used in binary classification problems but can also be used for classification problems with more than two classes (a.k.a. multi-class classification problems). It works by first approximating the probability that a given image belongs to each class, and then selecting the class with the maximum estimated probability.

To start, I split the data into training and test sets. The training set was used for model building, with the test data set aside to evaluate how well the model generalizes. I also scaled the data using standard scaling, which sets a mean of zero and a variance of one unit. This step generally makes computations more stable.

From there, I produced a model with tuned regularization parameters. Because the L1 penalty often performs better than the L2 does with sparse matrices, I used L1 with the "SAGA" solver for multinomial classification. The regularization process introduces a small amount of bias into the model to reduce variance significantly. The tradeoff further helps the model generalize.

I also tweaked the tolerance in the model to expedite the convergence of the algorithm. This step can save a notable amount of time with a large amount of data like that which comes from images. The default tolerance is .0001, and I did not want (or need) to deviate too far from that, as doing so could have unduly increased the bias. Table 1 shows digit-level performance for the training data. As expected, the model performs a bit better here than with the test data, but not so much as to indicate overfitting.

	precision	recall	f1-score	support
0.0	0.95	0.94	0.94	5611
1.0	0.96	0.93	0.94	6528
2.0	0.87	0.90	0.88	5405
3.0	0.86	0.88	0.87	5645
4.0	0.91	0.90	0.91	5493
5.0	0.81	0.86	0.84	4750
6.0	0.93	0.92	0.93	5549
7.0	0.90	0.91	0.90	5739
8.0	0.84	0.82	0.83	5588
9.0	0.87	0.85	0.86	5692
avg / total	0.89	0.89	0.89	56000

Table 1: Classification Report for Logistic Regression with Training Data

This model provided an accuracy score of 89.04% on the test data. For a more granular look at the model's performance, I generated a classification report to reflect accuracy on each digit, too. The report for test data, which appears in Table 2, also provides metrics for precision (percent of correct predictions), recall (percent of true digits[1] classified correctly), and support (the number of observations in the relevant class, or digit).

[1] E.g., a digit that is truly a 5 being classified as a 5 by the model.

	precision	recall	f1-score	support
0.0	0.96	0.94	0.95	1415
1.0	0.97	0.92	0.94	1686
2.0	0.86	0.89	0.87	1342
3.0	0.87	0.87	0.87	1399
4.0	0.89	0.91	0.90	1332
5.0	0.81	0.86	0.83	1190
6.0	0.93	0.92	0.93	1408
7.0	0.90	0.91	0.91	1444
8.0	0.83	0.82	0.82	1371
9.0	0.87	0.85	0.86	1413
avg / total	0.89	0.89	0.89	14000

Table 2: Classification Report for Logistic Regression with Test Data

The global accuracy scores and the metrics in Tables 1 and 2 indicate a strong model overall, as precision and recall are both well above .5 (the equivalent of random guessing) for each digit.

Finally, I produced at a heat map (Figure 3) to visualize the performance of the model. The heat map is based on a confusion matrix, which captures the predicted-actual pairings for each digit. Each row/column represents a digit, in order from zero to nine. The heat map depicts a model that performs well and offers solid grounds for comparison to the neural network.

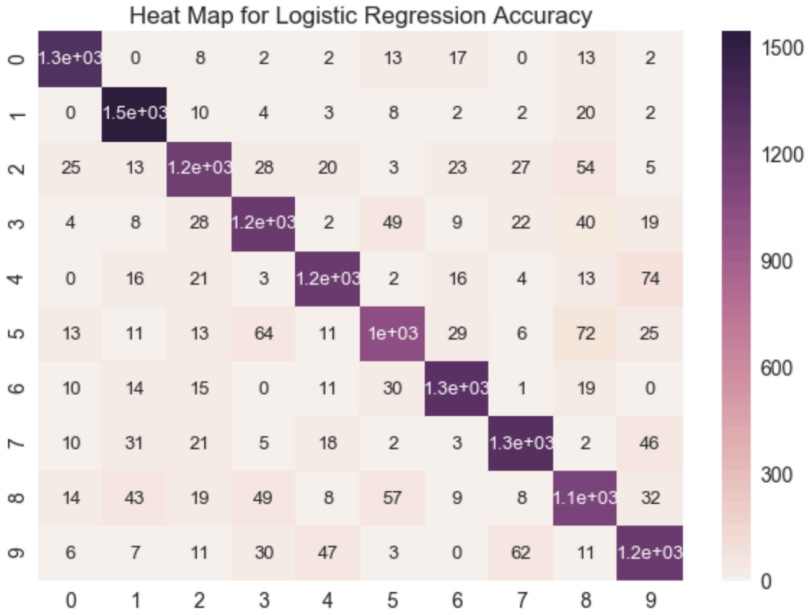


Figure 3

Random Forest and Results

For an additional comparison to neural networks (and to logistic regression), I also tried random forests. The structure of this model is notably different from a logistic regression, so its output will offer further comparative value. Specifically, a random forest makes classification decisions using decision trees that rely on a series of split points (e.g., where to draw the line between a 2 and a 3) of different predictors. The algorithm randomly generates many decision trees to harness the benefits of an ensemble model.

After splitting the original dataset in the same fashion used for logistic regression, I produced an initial random forest without tuning any parameters. It yielded 94.36% overall accuracy on the test data. While this offered a promising start, I needed to next tune parameters in order to make sure the model was not overfitting the training data. This step also helped improve the model's stability. These adjustments sometimes herald certain drops in accuracy, but they are a worthwhile trade-off to help the model generalize effectively.

Through a bit of trial and error[2], I determined that using both the square root and $\log_2$ options for the maximum number of features produced nearly the same results, which makes sense mathematically for 10 features. As such, I left the default of square root in place for this parameter. For the number of estimators (or trees), I tested 10, 20, and 30, which all produced similar results. With a relatively small number of features, 20 was a good middle ground to limit error while still having enough randomization in the ensemble. Model performance also held with a depth of 5 levels in the trees, so I kept this relatively low value to reduce the potential for overfitting. The updated model gave an overall accuracy of 83.44% and the following classification report that appears in Table 3.

[2] I favored this method over plotting errors—which I used in my previous project—to identify stabilizing error rates because it helped avoid the computational delays that came with plotting such a large dataset.

	precision	recall	f1-score	support
0.0	0.93	0.90	0.92	1403
1.0	0.97	0.84	0.90	1794
2.0	0.83	0.86	0.84	1357
3.0	0.85	0.74	0.79	1602
4.0	0.82	0.83	0.82	1342
5.0	0.64	0.90	0.75	882
6.0	0.88	0.86	0.87	1408
7.0	0.89	0.82	0.86	1603
8.0	0.72	0.84	0.78	1251
9.0	0.77	0.79	0.78	1358
avg / total	0.85	0.83	0.84	14000

Table 3: Classification report for random forest

After parameters were tuned, the random forest's performance was worse than that of the logistic regression. The juxtaposed heat maps appear in Figure 4.

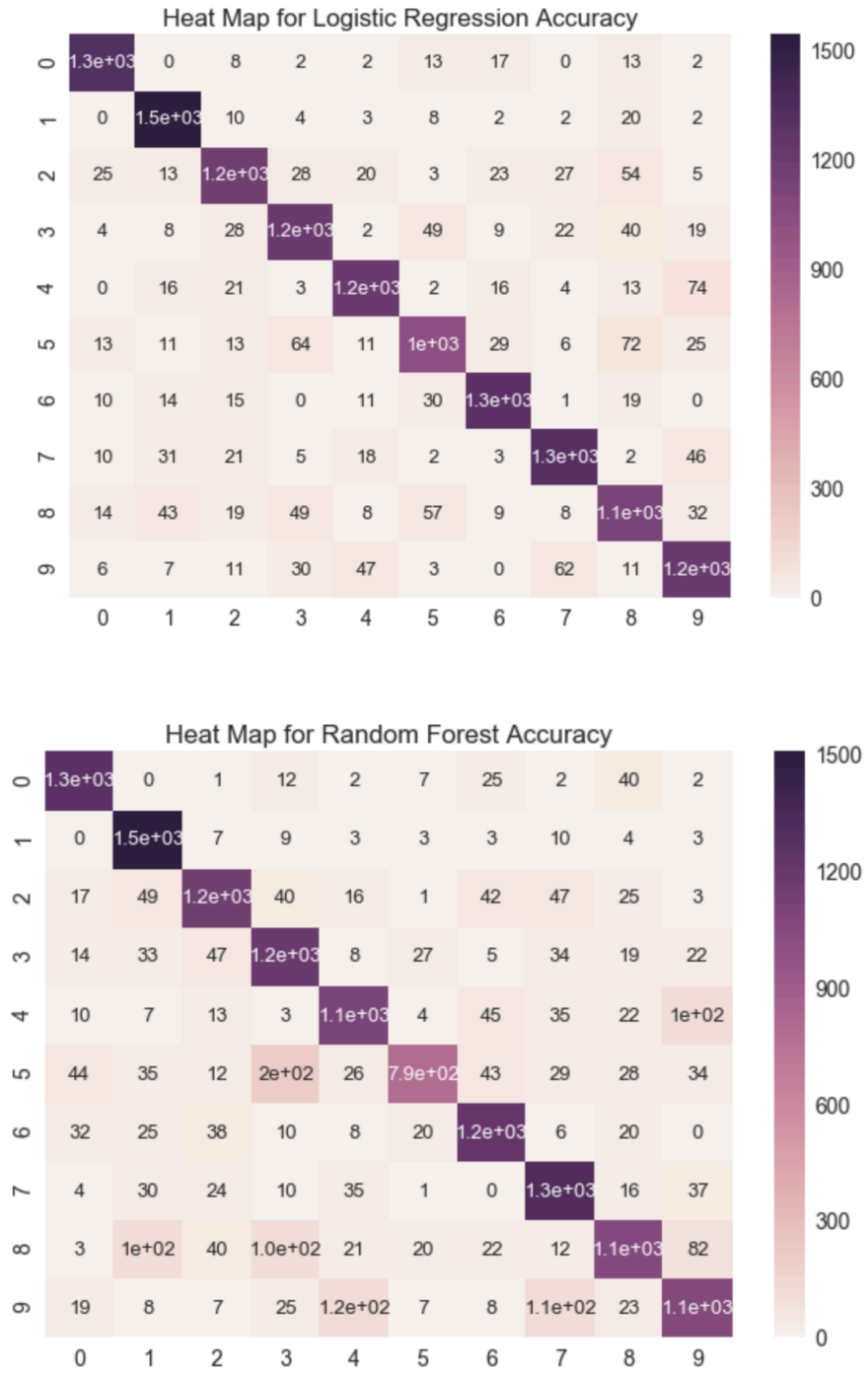


Figure 4

The slightly lighter shading in the diagonal of the random forest's heat map reflects its lower performance. Aligned with that trend are several instances off darker shading in

off-diagonal cells, which reflect higher rates of missclassification. The logistic regression will offer more “competitive” comparison to the neural networks.

Neural Networks and Results

In constructing a neural-network model, I started with decisions about parameters. I chose “adam” as the solver because it provides a good compromise between computational demand and advanced weighting. I used the logistic activation function as well. It often performs better in classification problems because small changes in variable weights and bias cause only a small change in its output. Finally, to determine the best alpha value (L2 regularization penalty), I used cross-validation, which yielded an alpha value of 0.01. The L2 penalty is important for striking a balance between variance and bias so that the model will generalize appropriately.

The multi-layer perceptron network had three layers of perceptrons to process the data. The first layer, the input layer, had 784 cells/perceptrons (one for each feature/image in the matrix X) and was activated by the inputs of X. These inputs are also called activation units. The second, middle layer was the hidden layer. It was activated by a linear combination of the activation units in the input layer. For a relatively basic network like this one, I left the default setting of 100 cells in place. The third and final layer was the output layer, which was activated by a linear combination of activation units in the hidden layer. The output layer had 10 cells, one for each class in the multi-class classification problem. Ultimately, the tuned networks showed the strongest results: the overall accuracy scores were 94.02% and 93.62% on the training and test data, respectively. Its classification reports (Table 4 for training data and Table 5 for test data) confirm its superior performance to the other two models.

	precision	recall	f1-score	support
0.0	0.97	0.98	0.97	5482
1.0	0.97	0.98	0.98	6271
2.0	0.95	0.90	0.93	5910
3.0	0.95	0.91	0.93	6000
4.0	0.88	0.98	0.93	4906
5.0	0.94	0.89	0.91	5361
6.0	0.96	0.95	0.95	5547
7.0	0.97	0.94	0.95	5967
8.0	0.87	0.95	0.91	4929
9.0	0.93	0.92	0.93	5627
avg / total	0.94	0.94	0.94	56000

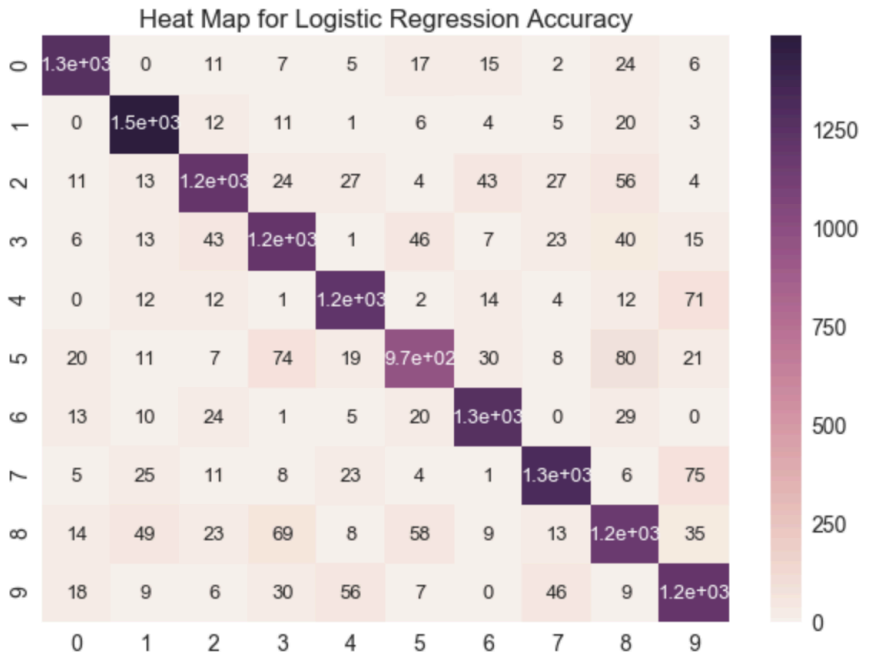
Table 4: Classification report for neural networks with training data

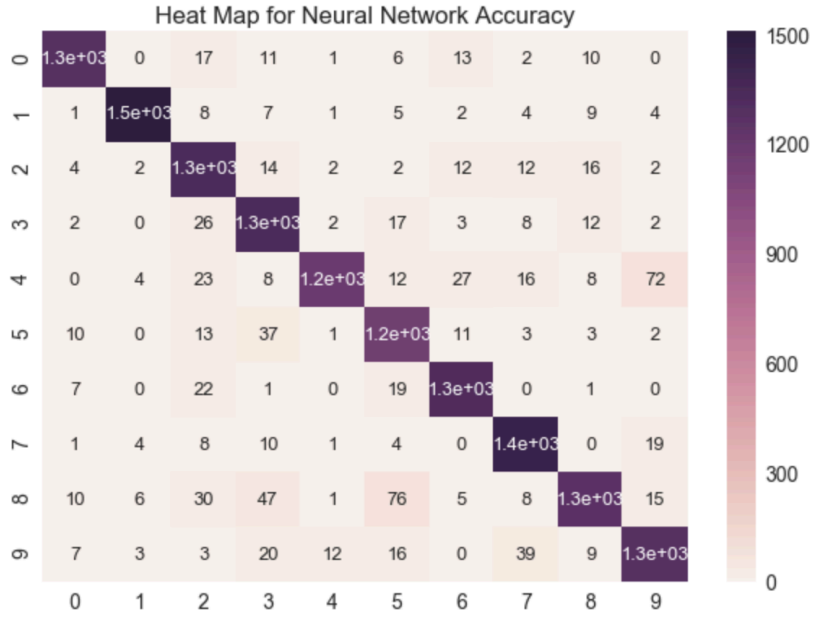
	precision	recall	f1-score	support
0.0	0.96	0.97	0.96	1333
1.0	0.97	0.99	0.98	1526
2.0	0.95	0.90	0.93	1492
3.0	0.95	0.90	0.92	1480
4.0	0.87	0.98	0.93	1201
5.0	0.94	0.88	0.91	1320
6.0	0.96	0.95	0.96	1397
7.0	0.97	0.94	0.95	1520
8.0	0.86	0.95	0.90	1330
9.0	0.92	0.92	0.92	1401
avg / total	0.94	0.94	0.94	14000

Table 5: Classification report for neural networks with test data

Summary and Future Work

The classification reports indicate that the neural network outperforms both the logistic regression and the random forest. The overall accuracy for the neural network is several percentage points higher than are the scores of the other two models, and its heat map shows much improvement on individual digits, including 5, which seemed to be the hardest digit for the models to detect.





For an easy numerical comparison among the three models, overall accuracy scores are reproduced in Table 6. The first value in each cell reflects training data, and the second value captures test-data performance.

	Logistic Regression	Random Forest	Neural Network
Accuracy	.91, .89	n/a, .83	.94, .93
Precision	.89, .89	n/a, .85	.94, .94
Recall	.89, .89	n/a, .83	.94, .94
F1	.89, .89	n/a, .84	.94, .94

Table 6: Summary metrics for all models

In future work, I would like to explore hidden layers more extensively. The scikit-learn package's multi-layer perceptron classifier easily accommodates adjustments to the numbers of hidden layers and the neurons that compose them. It would be interesting to see how much improvement deep learning would offer in the image-classification process.

Recommendations

Clients in this project cover a broad range of interests. Any organization that could benefit from automated recognition of handwritten digits could be considered a stakeholder. A few examples would be medical facilities (recognizing entries on forms completed by hand) and banks (for recognizing checks or other forms). As general guidance, I would recommend the use of neural networks to these groups. In my experience, they have offered the best accuracy in a stable model. The neural network here was also less computationally intensive than the logistic regression was, though it should be noted that the random forest was the least intensive.

References

1. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python.” Journal of Machine Learning Research, vol. 12, pages 2825-2830, 2011.

2. “MNIST for ML Beginners,” Tensorflow.org. <https://www.tensorflow.org/get_started/mnist/beginners#about_this_tutorial>. Last updated August 17, 2017.