MNIST Dataset

Ranaa Ansari, Daniel Arenson, Yining Feng, Han Nguyen

Northwestern University

MSDS-422 Assignment 5: Principal Components Analysis

Summary and problem definition for management

Assignment 5 focused on applying Principal Component Analysis (PCA) and random forest learning methods to a multi-class classification of handwritten digits in the MNIST dataset. A variety of random forest models were built to assign a digit that is equal to the handwritten one; as there were slight differences among handwritten digits, the challenge was for the models to accurately distinguish between each digit with a reasonable run time. Ultimately, the end goal was to provide a recommendation on whether to use PCA as a preliminary to machine learning classification.

Research Design

The steps for the analysis are shown as below:

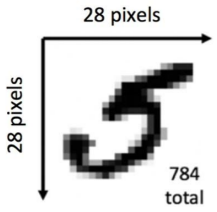
(1) Fit a random forest multi-class model using all 784 variables

(2) Use Principal Components Analysis (PCA) to reduce the data

(3) Fit another random forest model using a reduced number of estimators

(4) Compare each model's runtime, F1, precision, and recall scores

The data set itself is fairly straight forward. There are 70,000 handwritten digits. Each row represents one of these digits. There are 785 columns of data, and 784 of them are the integer grayscale values of each pixel in a 28×28 pixel square. The first column is the 'response' variable, which is the actual value to test the predicted estimate against. For example, the plot on the left is a binary plot showing a row of data that has a y value of '5'. The first column of data is the actual value – for training & testing. The subsequent 784 columns of data are the grayscale values for each of the 28×28 pixels representing the digit. The next step in the process is to split the data by training & testing data sets. According to the instructions, we are told to use 60,000 rows for training, and 10,000 for testing. We then shuffle the training data set by the index to randomize the data. Subsequently, we begin the experiment by creating a random forest classifier using all 784 variables, a principal components analysis (PCA), and a random forest model based on a reduced dataset that leverages the PCA output. For model evaluation, we compare performance metrics including each model's runtime, F1, precision and recall scores of training & testing data sets.



An example of a plotted row of data (784 pixels)

Programming Work

The programming dataset MINST was retrieved using *sklearn.datasets*. The MNIST database contains 60,000 training images and 10,000 testing images. The entire dataset was used to create a multi classifier. Four models were created: Random Forest Classifier, PCA, Random Forest reduced dataset, and re-run dimension reduction on training data model. *Scikit-learn* library was used to import Precision, Recall, F1 score and confusion matrix. Run times were measured for each model to evaluate classification performance. `Max_features='sqrt'` was used in model 1 and 3 so that the explanatory variables were included in the individual trees.

The first model was a Random Forest Classifier using the full set of 784 variables that developed a set of 60,000 observations. *Scikit-learn* library `RandomForestClassifier` and *sklearn.metrics* import `confusion_matrix`, which was used for this model.

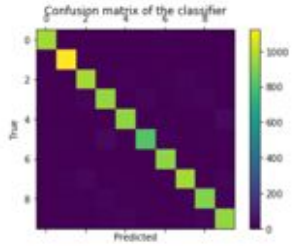
The second model was a principal components analysis (PCA). *Sklearn.decomposition PCA* was used to run a PCA analysis. The model used 70,000 observations generating principal components that represented 95% of the variability in the explanatory variables.

The third model was a Random Forest Reduced Dataset that leveraged the output of model 2. This model used the *Scikit-learn* library `RandomForestClassifier` with a smaller dataset `n_estimators=154`. The reduced dataset number was the output from model 2.

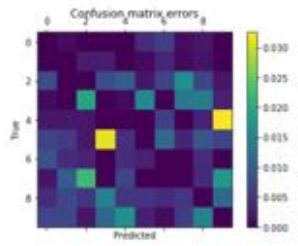
Finally the fourth model was programmed to run a Random Forest with reduced variables and PCA model.

Results and Recommendation

A combination of Random Forest, PCA, RF + PCA on reduced dataset and reduced estimators was applied. We ran a confusion matrix on each model to see if it could potentially reveal any findings that would indicate which one would be the most optimal model to use. We also included an error analysis matrix to identify which numbers were being classified incorrectly as a way to help improve the model.

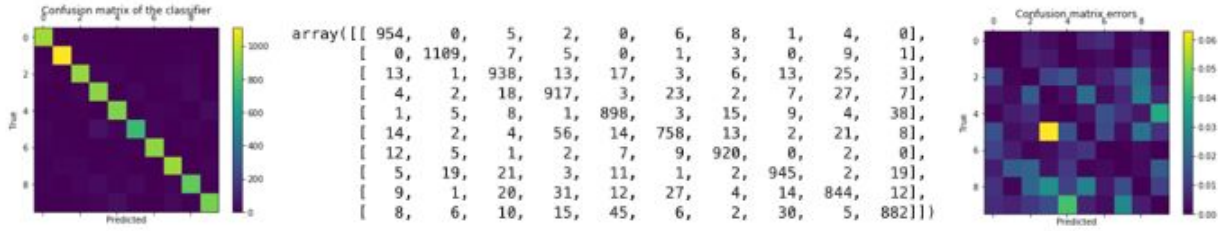



```
[[ 966    1    0    0    0    4    6    1    2    0]
 [   0 1121    3    3    0    1    4    1    2    0]
 [   8    0  971    8   10    2    7   17    7    2]
 [   1    0   20  942    1   16    0   13   14    3]
 [   2    2    2    0  935    0    6    0    3   32]
 [   6    5    2   28    4  820   10    3    8    6]
 [   8    4    3    0    4    3  932    0    4    0]
 [   2   10   23    0    7    0    0  964    4   18]
 [   4    7    5   14    5    5    4    5  914   11]
 [   7    7    2   11   17    3    1    6   15  940]]
```

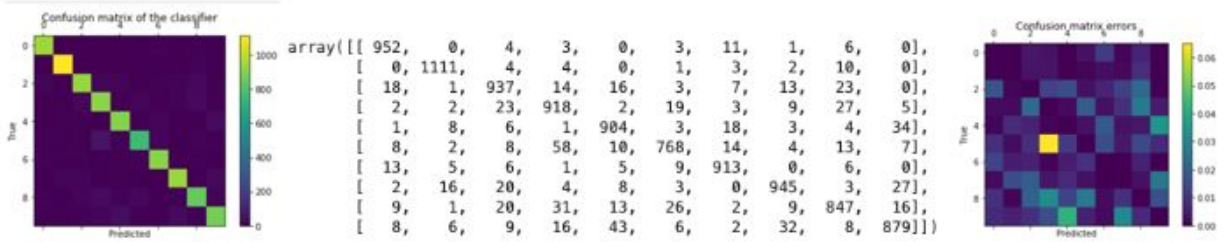


Model 1: With a test accuracy of 0.95, the model performed well. However, this would not be the recommended model to use, mainly because it takes ~ 24 minutes to run. This confusion matrix looks pretty good, with a slight deviation in performance classification on 5's. The

classifier here is making frequent errors for predicting 5's to be 3's and 4's to be 9's and less frequently making errors on predicting 7's to be 2's or 9's and 3's to be 2's or 5's.



Model 3: With a test accuracy of 0.915, the model performs slightly less as accurate to model 1. However, the time it took to run this model in comparison to model 1 was drastically cut down, resulting in ~11 minutes. The confusion matrix here is very similar to Model 1, but with less errors in misclassifying the numbers, now mainly predicting 3's to be 5's and 4's to be 9's.



Model 4: With a test accuracy of 0.916, the model performed almost identical to model 3. Since we addressed earlier that there was a design flaw in the earlier models applying PCA to the entire dataset, Model 4 was an improvement to this, because the dataset was now split into train and test sets prior to running PCA and making predictions on the test set. The classification errors were also cut down to being less frequent, similar to model 3. However, the time it took to run this model in comparison to the was the quickest, running at ~10 minutes.

Models	**n_Features**	**Duration**	**F1 Score**	
			Train	*Test*
Random Forest (with all Variables)	784	24.28	0.969	0.95
Principal Component Analysis (PCA)	154	47 s	n/a	n/a
Random Forest and PCA with Reduced Dataset (flawed)	154	10.43	0.946	0.915
Random Forest and PCA with Reduced Dataset (fixed)	149	10.31	0.947	0.916

With all of these factors coming into play, we would recommend using Model 4 (Random Forest + PCA with reduced dataset), using 149 estimators. It produced less errors than the remaining models, had a good test accuracy score, and also ran the quickest.

Because these models did take a substantial amount of time to run both collectively and individually, we would also recommend using a machine learning platform, like Tensorflow, because it has the capability to increase latency with running complex models, especially for classification use cases.

Reference

"2.5. Decomposing Signals in Components (Matrix Factorization Problems)." *Scikit*, scikit-learn.org/stable/modules/decomposition.html.