Digit Classification Using Machine Learning

Parabjot Chander
parabjot.chander49@qmail.cuny.edu
CUNY ID: 23898849

Hassan Bashir
hassan.bashir98@qmail.cuny.edu
CUNY ID: 24138998

Abstract

In this report, we present an extensive analysis of a machine learning-based classification problem using the given dataset. The training and test dataset consists of pixel intensities corresponding to images of handwritten digits, where the task is to classify the digits accurately. The dataset is divided into training and testing sets, and the primary goal is to develop a classifier to recognize the handwritten digits with good accuracy. Random Forest classifier is one of the classifiers used to make a model to classify handwritten digits and it is a powerful Sci-kit learning method and we used it in our model. This model uses Random Forest along with Cross-Validation to validate along with grid search to produce the best hyperparameters, ensuring that the model generalizes well to unseen data. Furthermore, we split the test data into validation data to evaluate our model's performance on a separate dataset not used during training. To assess the model's performance, we compute various metrics such as accuracy, confusion matrix, precision, recall, F-score, and precision-recall curves. These metrics provide a comprehensive understanding of the model's capabilities, strengths, and weaknesses. Moreover, we calculate per-class and overall accuracy metrics to analyze the model's effectiveness in classifying each individual digit, as well as the classifier's overall performance.

1 Introduction:

1.1 Problem Definition/Motivation

We have been provided with the data set that contains pixel values for 16x16 grayscale handwritten images. We are instructed to either create our own algorithm or to use an already existing algorithm and modify it to increase the model's accuracy which will recognize the digit written on the image. We plan to employ machine learning algorithms such as decision trees, stochastic gradient descent, or random forest. Our goal is to create a model with the best accuracy, and it will take less time to train the model. In simple words, we plan to develop a model with more accuracy and fewer resources with exposure to research in digital image processing and machine learning.

1.2 Literature Review

Recognition accuracy for handwritten digits using random forest can reach up to 98%. Moment-based features are useful for tasks, particularly for the recognition of handwritten digits. Ram Sarkar during their research experimented with 8 different classifiers along with feature extraction based on moment-based features. Their model gave 96-99% accuracy for random forest classifiers and they used 12 different data sets. While moment-based features generally exhibit excellent performance incorporating complementary features such as concavity analysis could potentially aid in distinguishing between digits (Singh and Sarkar 11). KNN classifiers can also be used to get decent accuracy. The algorithm looks for the top nearest K instances and outputs the class with the highest frequency as the prediction (Banik & Doran, 2017, 2). It is a good approach but comparing the results of these two authors' random forests is more accurate than KNN.

2 Methodology:

The algorithm we used to train our data is known as the Random Forest, a supervised machine-learning algorithm that is widely used in regression and classification problems. This algorithm uses an ensemble learning technique meaning it uses a collection of models (decision trees) to make the prediction, the majority vote or average from the decision trees outputs becomes the prediction. Some difference between a single decision tree and a random forest is that a single decision tree is faster in computation but more prone to overfitting when compared to a random forest. Overfitting is a situation when your training model performs flawlessly only on training data but is horrible on the testing data since it memorizes the patterns/features of the training data.

2.1 Random Forest Algorithm

1. A bootstrapped dataset (training data) is selected for the construction of the decision trees.
2. A subset of features is selected for constructing each decision tree, not all features are considered when making an individual decision tree, each decision tree is unique.
3. Individual decision trees are constructed from step 2.
4. Each decision tree generates an output when fed data.
5. The final output or prediction is computed based on majority voting or average from the decision trees, this step is known as aggregation.

3 Implementation and Experimentation:

3.1 Description of Dataset

The dataset contains handwritten digits (0-9) scanned from envelopes by the U.S. Postal Service, the original images of the digits are of different sizes and are slanted. The dataset contains the scanned images which are normalized and deslanted, being 16x16 grayscale images. There are 7921 digits used for training and 2007 used for testing, below is a chart showing a breakdown based on the digits.

Table 1: Dataset breakdown by digits

Digit	0	1	2	3	4	5	6	7	8	9	Total
Train	1194	1005	731	658	652	556	664	645	542	644	7291
Test	359	264	198	166	200	160	170	147	166	177	2007

3.2 Description of Performance Measurements

Let's say we are given a data set for defining whether images represent 5 or not, and let's say 10 % of the dataset represents images of 5. If we predict that all images in the dataset don't represent 5, then we have a 90% percent accuracy, but this is misleading due to the skewed dataset. The point is that accuracy isn't a perfect measurement as it relates to classification models, we need to look at more performance measurements. Some metrics used to evaluate classification models are true positive, true negative, false positive, and false negative. True Positive is the number of outcomes the training model correctly predicts the image corresponding to a class. True Negative is the number of outcomes the training model correctly predicts the image not corresponding to a class. False Negative is the number of outcomes the training model predicts the image doesn't represent a class when it does. False Positive is the number of outcomes the training model predicts the image to represent a class when it isn't.

$$\textbf{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad \textbf{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\textbf{F1 Score} = \frac{2*Recall*Precision}{Recall + Precision} \quad \textbf{Specificity} = \frac{True\ Negative}{True\ Negative + False\ Positive}$$

$$\textbf{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

For tables 2 and 3: the rows represent the reality and columns represent the prediction.

Table 2: Binary classification when class = 0 confusion matrix

	Not 0 (-)	0 (+)
Not 0 (-)	True Negative	False Positive
0 (+)	False Negative	True Positive

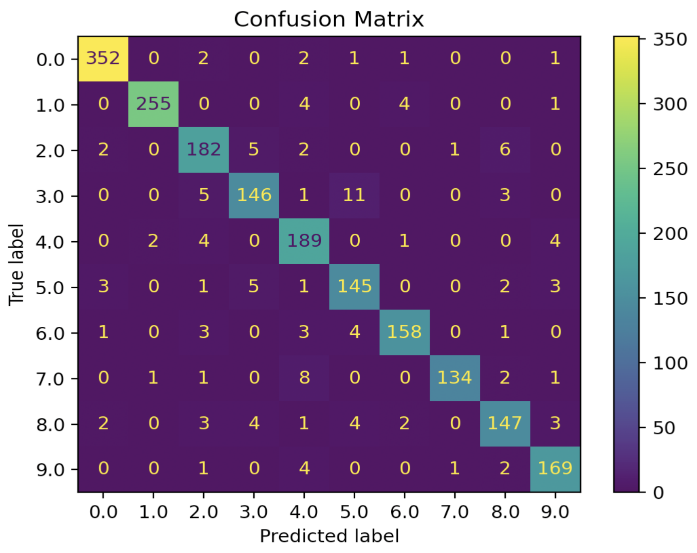
Table 3: Multi classification when class = 0 confusion matrix

0	**TP**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**
1	**FP**	**TN**								
2	**FP**		**TN**							
3	**FP**			**TN**						
4	**FP**				**TN**					
5	**FP**					**TN**				
6	**FP**						**TN**			
7	**FP**							**TN**		
8	**FP**								**TN**	
9	**FP**									**TN**

0	**TP**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**	**FN**
1	**FP**	**TN**								
2	**FP**		**TN**							
3	**FP**			**TN**						
4	**FP**				**TN**					
5	**FP**					**TN**				
6	**FP**						**TN**			
7	**FP**							**TN**		
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

3.3 Confusion Matrix

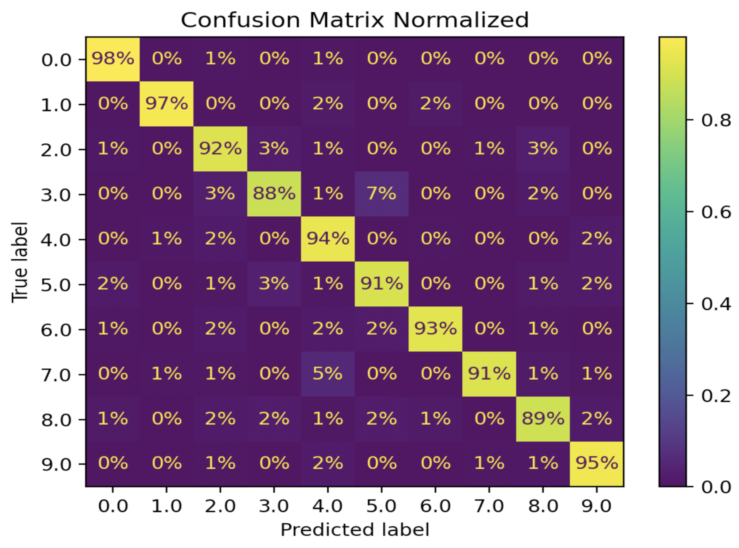
If we analyze the confusion matrix, we can see that the 3 was confused as 5 for 11 times in Figure 1. In Figure 2 label 0.0 gave the highest accuracy of 98% which means that it was confused as another label for only 2% of the predictions.



True label \ Predicted label	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	352	0	2	0	2	1	1	0	0	1
1.0	0	255	0	0	4	0	4	0	0	1
2.0	2	0	182	5	2	0	0	1	6	0
3.0	0	0	5	146	1	11	0	0	3	0
4.0	0	2	4	0	189	0	1	0	0	4
5.0	3	0	1	5	1	145	0	0	2	3
6.0	1	0	3	0	3	4	158	0	1	0
7.0	0	1	1	0	8	0	0	134	2	1
8.0	2	0	3	4	1	4	2	0	147	3
9.0	0	0	1	0	4	0	0	1	2	169

Figure 1: Confusion Matrix

The most error rate is found in label 3 which is predicted correct 88% times, which means it gave a 12% error rate and label 8 gave an 11% error rate which means it was predicted correct 89% times.



	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	98%	0%	1%	0%	1%	0%	0%	0%	0%	0%
1.0	0%	97%	0%	0%	2%	0%	2%	0%	0%	0%
2.0	1%	0%	92%	3%	1%	0%	0%	1%	3%	0%
3.0	0%	0%	3%	88%	1%	7%	0%	0%	2%	0%
4.0	0%	1%	2%	0%	94%	0%	0%	0%	0%	2%
5.0	2%	0%	1%	3%	1%	91%	0%	0%	1%	2%
6.0	1%	0%	2%	0%	2%	2%	93%	0%	1%	0%
7.0	0%	1%	1%	0%	5%	0%	0%	91%	1%	1%
8.0	1%	0%	2%	2%	1%	2%	1%	0%	89%	2%
9.0	0%	0%	1%	0%	2%	0%	0%	1%	1%	95%

Figure 2: Confusion Matrix Normalized

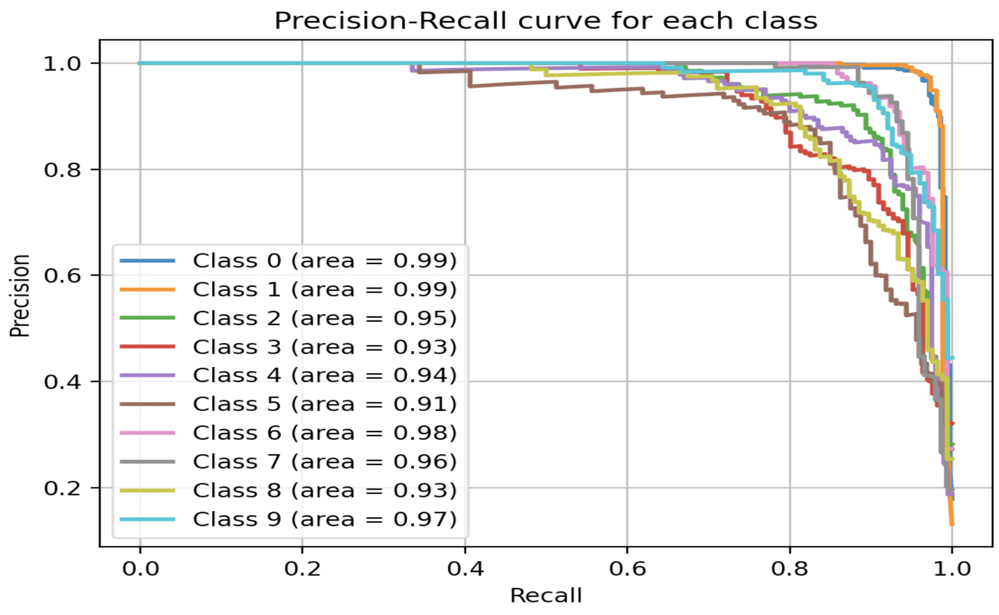


Figure 3: Precision and Recall Curve

A big area under the precision-recall curve for a class indicates that precision and recall scores are relatively close and high. For example, in Table 4, the precision and recall scores for class 0.000 (0) are 0.977778 and 0.980501 respectively, resulting in an area of 0.99 shown in Figure 3.

3.4 Per-Class Accuracy

Table 4: Per class accuracy

Class	Label	Specificity	Recall	Precision	F_Score
0.000	0	0.995146	0.980501	0.977778	0.979138
1.000	1	0.998279	0.965909	0.988372	0.977011
2.000	2	0.988944	0.919192	0.900990	0.910000
3.000	3	0.992395	0.879518	0.912500	0.910843
4.000	4	0.985612	0.945000	0.879070	0.940476
5.000	5	0.989172	0.906250	0.878788	0.892308
6.000	6	0.995645	0.929412	0.951807	0.940476
7.000	7	0.998925	0.911565	0.985294	0.946996
8.000	8	0.991309	0.885542	0.901840	0.893617
9.000	9	0.992896	0.954802	0.928571	0.941504

Table 5: Averages from each class

Average Precision	0.9361991215834929
Average Recall	0.935226706527155
Average F_Score	0.9353874368221916

3.5 Overall Accuracy

Accuracy

$$\frac{Number\ of\ Accurate\ Predictions}{Total\ Number\ of\ Test\ Cases} = \frac{1877}{2007} = \mathbf{0.935226706527155}$$

3.6 Validation process:

We used Cross-Validation to validate along with grid search to produce the best hyperparameters. Hyperparameters are used on random forest classifiers to enhance the performance and predictive power of the models. Grid search is an exhaustive method to find the best hyperparameters. We used a parameter grid of n_estimaters = [10,50,100,200] and depth = [none , 5, 10 ,20] . Then using cross-validation we trained the model on the valDatapixel and valDataID(image and label). We used 20% of the train data for the validation. The train_test_split function from the sklearn.model_selection was used to split the test data into validation and train data. For the validation process, the hyperparameters are modified, and the number of estimators or the number of decision trees built before the averaging or majority voting is changed from 100 to 200. The max depth or number of splits (splitting a tree node into sub-nodes) each decision tree is allowed to make is modified to 20. Validation accuracy was 100% which means that it correctly predicted all the instances of the validation data set.

Accuracy After Hyper Parameter Tuning

$$\frac{Number\ of\ Accurate\ Predictions}{Total\ Number\ of\ Test\ Cases} = \frac{1459}{1459} = \mathbf{1.0}$$

Error Rate in validation

$$\frac{Number\ of\ Inaccurate\ Predictions}{Total\ Number\ of\ Test\ Cases} = \frac{0}{1459} = \mathbf{0.0}$$

3.7 Results

Our network has given 93.52% overall accuracy. If we look at the errors normalized by rows. Label 7.0 is mostly confused by label 4.0 for up to 62% of the total errors. Similarly, label 5.0 was predicted as 3.0 for up to 55% of the total errors for label 3.0. The top-1 Error rate in this model is 6.47% and the Top-5 error rate is 0.448%

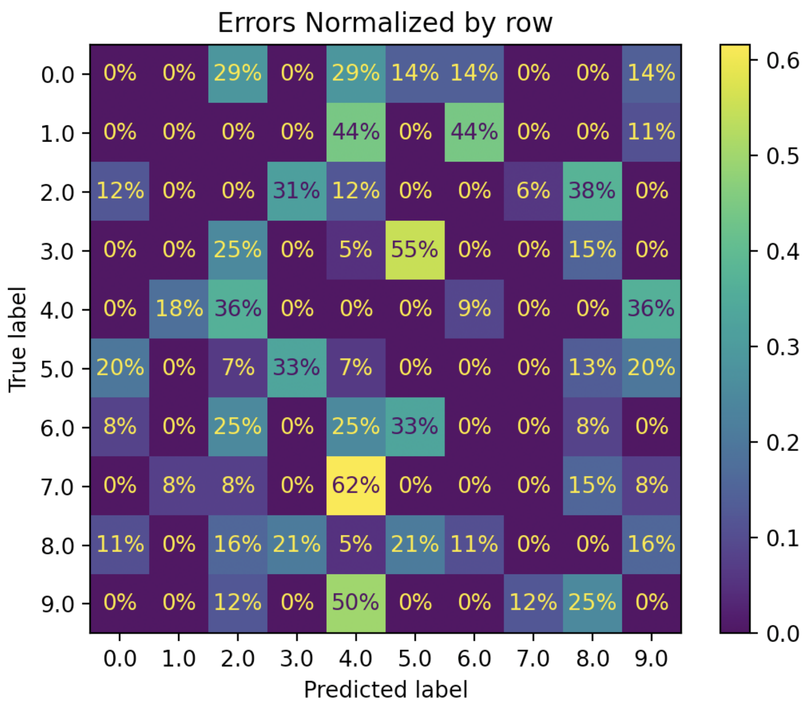


Figure 3: Error Normalized by Row

4 Discussion and Conclusion:

The error rate during validation being 0% is good news since our goal was to have it at a maximum of around 2.5%. We did something that is common in the field, the training set being approximately 80% and the testing set being approximately 20% of the overall dataset. The average precision and accuracy per class are relatively close and being greater than 90%, a high precision and recall indicate an ideal area under the ROC curve. We initially used Stochastic Gradient Descent (SGD) to train and test for this dataset, but the accuracy wasn't ideal. Through trial and error, we were led to train our model using Random Forest with hyperparameters being modified to produce the best accuracy, resulting in improvements.

Sources:

https://www.geeksforgeeks.org/cross-validation-machine-learning/

https://towardsdatascience.com/cross-validation-and-grid-search-efa64b127c1b

https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#:~:text=An%20ideal%20system%20with%20high,number%20of%20false%20positives%20(%20).

References

1. *Banik, D., & Doran, S. (2017). Digit Recognition: A Case Study. Tennessee Tec, 1(2), 10. file:///Users/hassanbashir/Downloads/Banik_Dipayan_Poster+Entry+No+143%20(2).pdf*
2. *Singh, P. K., & Sarkar, R. (2016). A Study of Moment Based Features on Handwritten Digit Recognition. Applied Computational Intelligence and So Computing, 2(4), 17. https://www.researchgate.net/publication/297743645_A_Study_of_Moment_Based_Features_on_Handwritten_Digit_Recognition/link/61845eada767a03c14f6b95b/download*

3. Geÿron, Aureìlien. *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 3rd ed O'Reilly, 2022.

4. *Harrison, Onel, "Machine Learning Basics with the K-Nearest Neighbor,"Towards Data Science,2018,https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761*
5. *Kumar P., Sharma V.,"Handwritten Digit Recognition using Machine Learning Techniques"*
6. *Egecioglu O., Sabir E.,"An Empirical Evaluation of k-NN on Handwritten Digit Recognition"*
7. *Oza N.C., Tumer K, "Random Forests for Handwritten Digit Recognition"* d_Features_on_Handwritten_Digit_Recognition/link/61845eada767a03c14f6b95b

   /download

Appendix:

We both collaborated through GitHub Desktop. We contributed equal efforts toward the code and the documentation of our work. We kept each other updated about all the changes that we made throughout the process and discussed what should be implemented before working on it so that both of us stay on the same page and understand the code properly to avoid the confusion in future.