Final Report: Assorted Algorithms on MNIST

Dylan Rodriquez
drodriqu@purdue.edu

Javad Raheel
jraheel@purdue.edu

April 23, 2020

1 Introduction

Recognizing handwritten digits is an important problem in several domains of computer science which include optical character recognition, computer vision, and machine learning, both in itself and as a prototype. MNIST, a dataset consisting of handwritten digits, is a padded version of the NIST handwritten images dataset [Deng (2012)] is widely used for these tasks. Images form this dataset are centered by a bounding box and are 28×28 in dimension (figure 1). This project is about implementing three algorithms, namely K-nearest neighbors (KNN), support vector machine (SVM), and neural network (NN) and studying their performance on the MNIST dataset. The focus is on tuning the models by employing k-fold cross-validation followed by statistical analysis to select the best hyperparameters for each model.

2 Methodology

2.1 Data

The data are from the MNIST dataset in idx format. The data are portioned into four files that contain the training features, testing features, training labels and testing labels [LeCun, Cortes, and Burges (2010)].

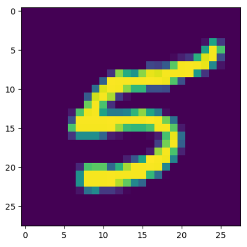

figure 1: Sample from MNIST Dataset

Each dataset was processed according to the provided offsets, types, and values and cast to a numpy array.
The samples are represented as a $n \times 784$ matrix, where n is the number of samples in the training or testing features. Labels consist of a single column vector with the same number of as the number of samples in its respective feature matrix.
Data points are shuffled each time upon loading and the data are split into roughly 85% training and 15% testing sets respectively.

2.2 K-nearest neighbors (KNN)

The KNN classifier was implemented using scikit-learn. The hyperparameter we tune for KNN is the number of neighbors (k).

2.3 Support vector machine (SVM)

SVM was implemented using scikit-learn. Hyperparameters C and the degree of the kernel polynomial are tuned with respect to a fixed regularization parameter ($\gamma = 0.5$).

2.4 Neural network (NN)

We experimented with using PyTorch to implement our NN. While that worked well, the scikit-learn implementation worked just as well. The code for our PyTorch implementation is available in the file *nnet.py*, but for the final submission we only use multilayer perceptron from the scikit-learn library. We tune number of hidden layers (2 or 3), and the number of neurons in each layer.

2.5 Platform Specifications

Experiments were conducted on a 2.60-GHz Intel hexa-core i7-9750H with 16 GB memory and an NVIDIA 1660 Ti 6 GB GPU in a Python 3.7.6 environment with scikit-learn [Pedregosa et al. (2011)], cuML (RapidsAI) [Raschka, Patterson, and Nolet (2020)] and PyTorch library [Paszke et al. (2019)].

2.6 Cross validation

(5-folds) Cross-validation was performed by partitioning the training data into five subsets. For each run, four of the subsets were used to train the model while the fifth subset was used as validation set to perform the cross-validation in a leave-one-out fashion (see Algorithm 1).

Algorithm 1 cross validation

```
$k \leftarrow$ number of folds
$S \leftarrow \{S_1, S_2, ...S_k\}$, k partitions of the training set
for i to k do
    train with sets$S_1, ..., S_{i-1}, S_{i+1}, ..., S_k$
    test on set $S_i$
    let $M_i$be the Mathew's correlation coefficient when the model is tested on $S_i$
end for
return the mean and variance of the set $\{M_i\}_{i=1}^{k}$
```

2.7 Parameter Tuning

Each run of the 5-folds cross-validation for a model's settings (i.e. the values of hyper-paramters) provides us with five Mathew's correlation coefficients (MCC) of the model, as a measure of its performance on the five different data settings (training and validation). We would henceforth get n means and variances of the model's MCC for n values of the hyperparameters. To determine the *best* model settings, we use statistical analysis of these means and variances.

ANOVA is used to determine if there is a statistically significant difference in mean MCC of the n groups, where each group represents the five MCC values resulting from the 5-folds cross-validation of the current model. If there is no statistically significant difference, a model is chosen at random. If on the other hand a statistically significant difference is found in the means of different model settings, then we do post-hoc analysis on these n groups.

During the post-hoc analysis we perform a Tukey's HSD test with all pairs of folds, under the null hypothesis that there is no difference in the n means. The alternative hypothesis is that there is a statistically significant difference in at least one pair of the means. We assume that the mean MCC is normally distributed with mean and standard deviation of the population being unknown.

If the null hypothesis is rejected, our post-hoc Tukey test returns to us a list of the pairs of indices $S = [(i, j), ..., (k, l)]$, indicating that the mean MCC for the model setting p is statistically significantly different from the mean MCC of model setting p for each $(p, q) \in S$.

3 Results

We ran the 5-folds cross-validation on a training data consisting of 60,000 samples, and computed the MCC for each settings of each model, along with precision and recalls for each class (i.e. digits 0 to 9) for each of the five runs when we test the model on a validation set during cross validation. In the following figures which show the box plots, the top plot in each is the MCC for each model. The rest of the plots depict the box plots for each class (digit) of either precision or recalls (see figures) for the five runs for each model.

The model which gave the best (statistically significant) mean MCC score was then trained on the entire training dataset and a confusion matrix was obtained for its performance on the test dataset. This confusion matrix appears as the last figure for each classifier.

3.1 Support Vector Machine

The range of values of the hyperparameters used are $C = 0.1 - 0.6$ in steps of 0.1, and for each value of C, the degree of the polynomial kernel to be $1-6$ in steps of 1. This makes for 36 total model settings (or simply models) for SVM. The results for each run of the 5-folds cross-validation are provided below, for each model. Each box plot shows the precision for one digit for one SVM model, for all the five folds.

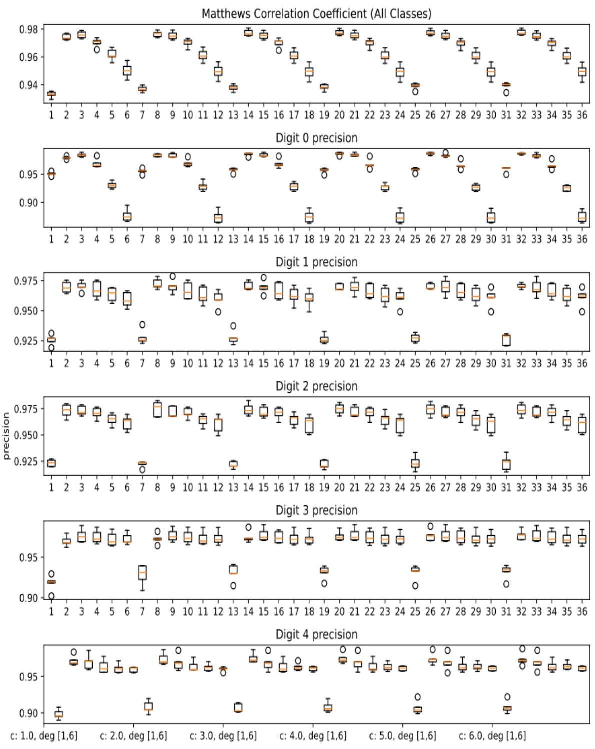

figure 2: The precision for all SVM models for digits 0 to 4 for all runs of 5-fold cross validation.

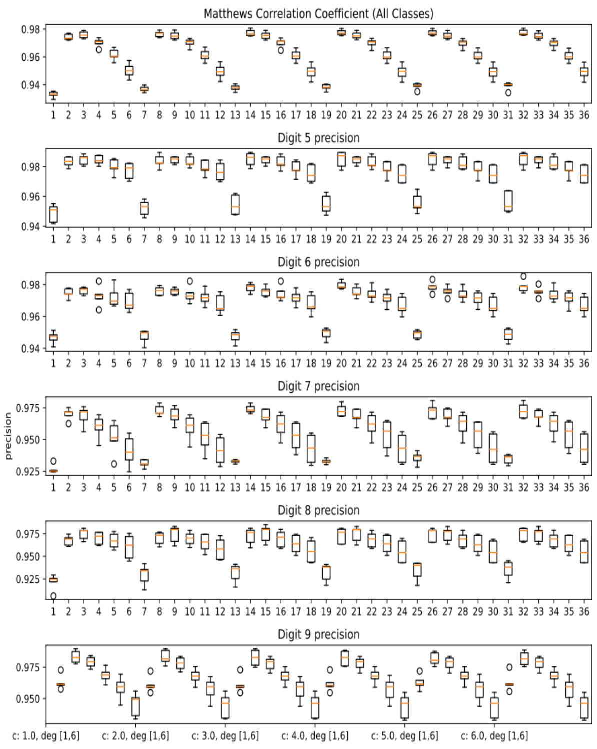

figure 3: The precision for all SVM models for digits 5 to 9 for all runs of 5-fold cross validation.

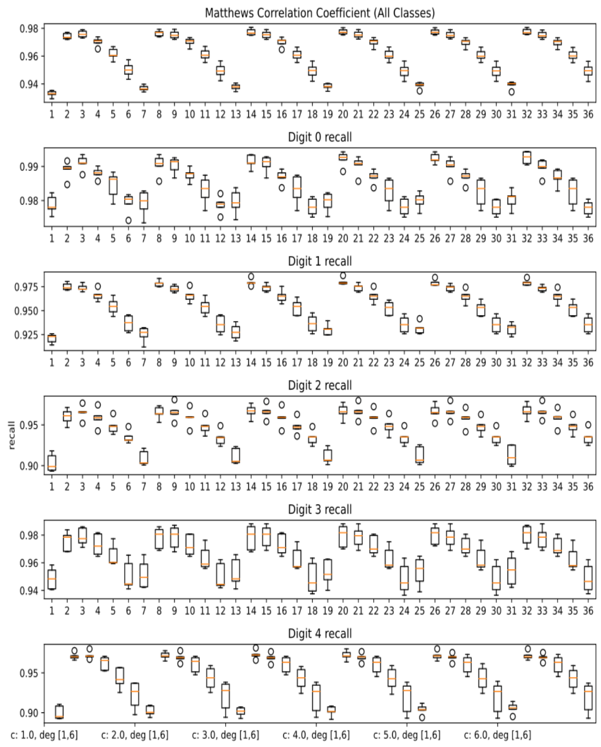

figure 4: The recall for all SVM models for digits 0 to 4 for all runs of 5-fold cross validation.

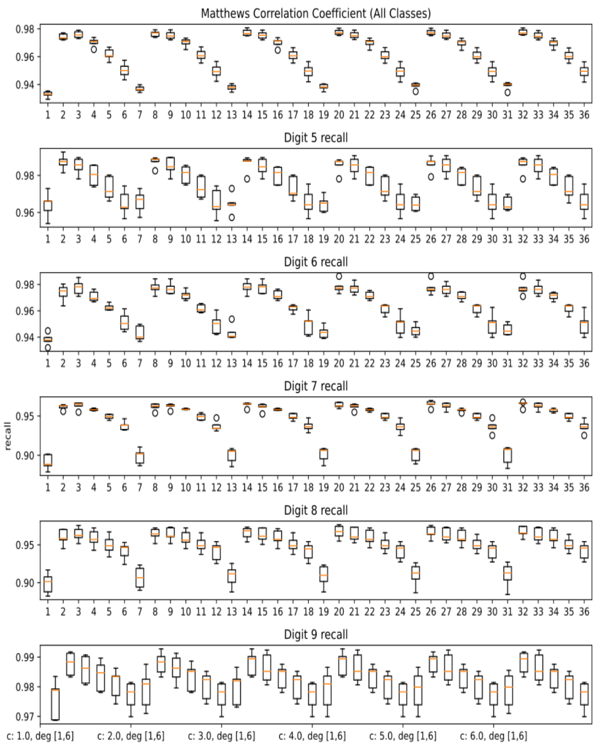

figure 5: The recall for all SVM models for digits 5 to 9 for all runs of 5-fold cross validation.

With respect to the combined MCC across all algorithms, MCC tends to decrease with respect to an increase in polynomial degree greater than 2. Additionally, this holds for all digits across all variants of the SVM. Digits with the greatest variance in cross validation precision means are 9 and 7 with lower variance occurring 1 and 5.

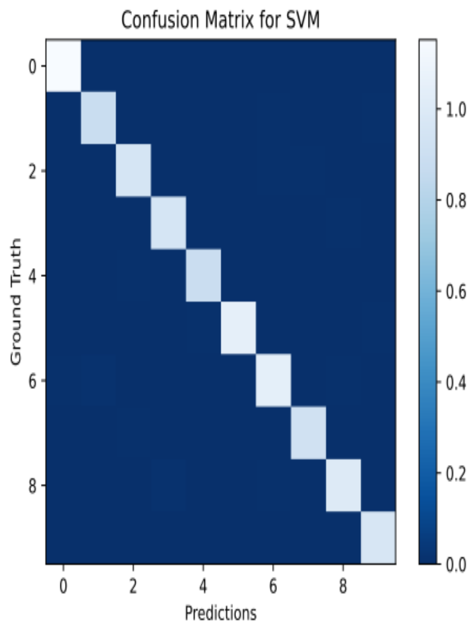

figure 6: The confusion matrix for SVM.

3.2 K-Nearest Neighbors

The range of values of the number of neighbors in KNN were from 1 to 15, giving us fifteen classifiers. The results, which follow the same pattern as for the figures for SVM are shown below.

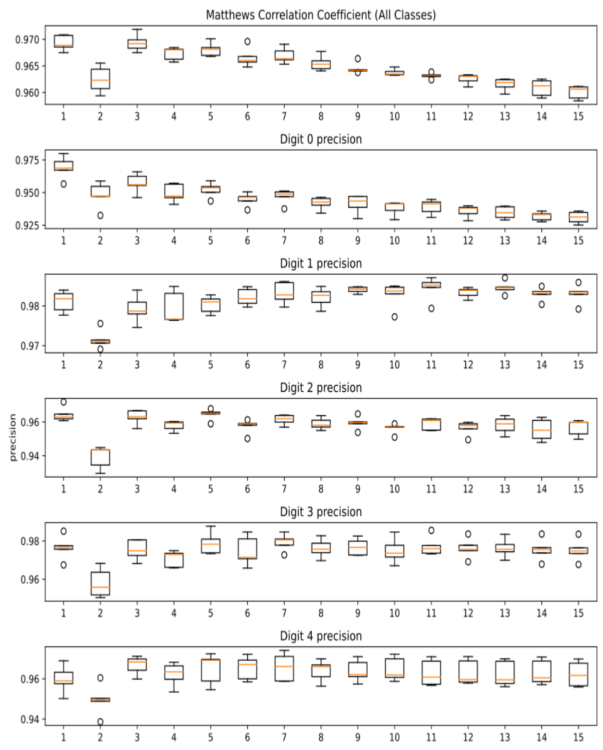

figure 7: The precision for all KNN models for digits 0 to 4 for all runs of 5-fold cross validation.

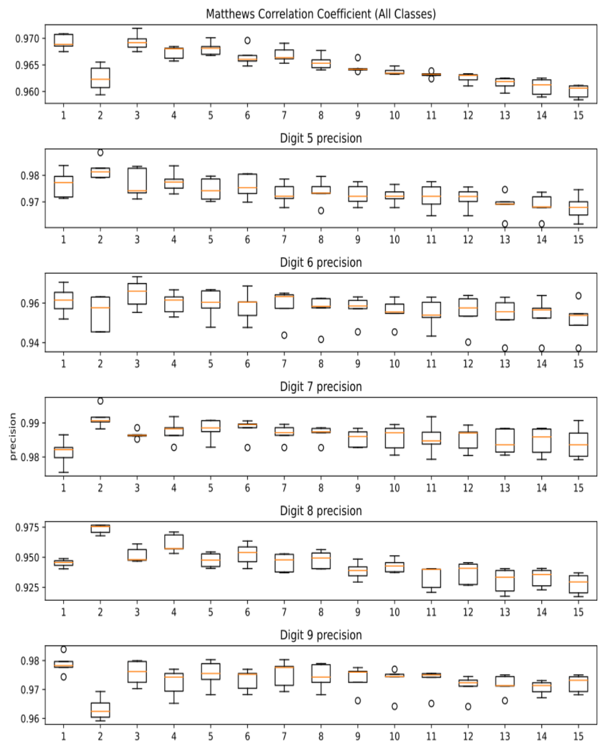

figure 8: The precision for all KNN models for digits 5 to 9 for all runs of 5-fold cross validation.

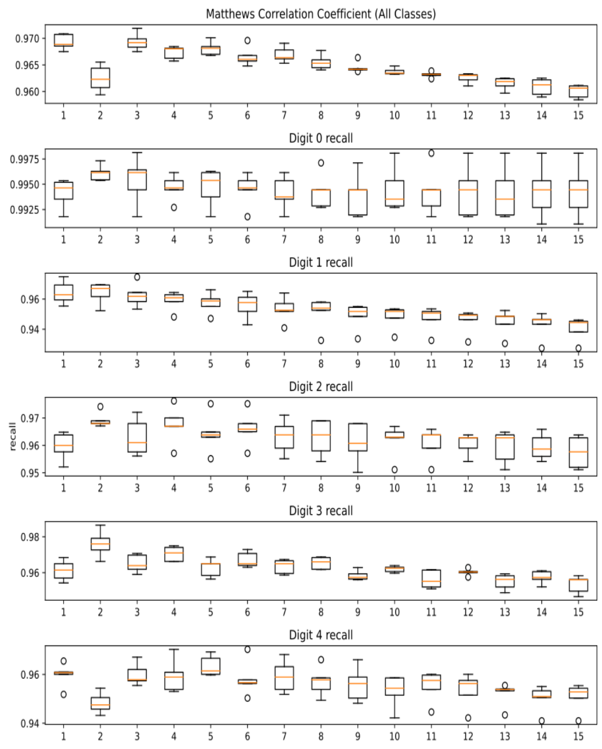

figure 9: The recall for all KNN models for digits 0 to 4 for all runs of 5-fold cross validation.

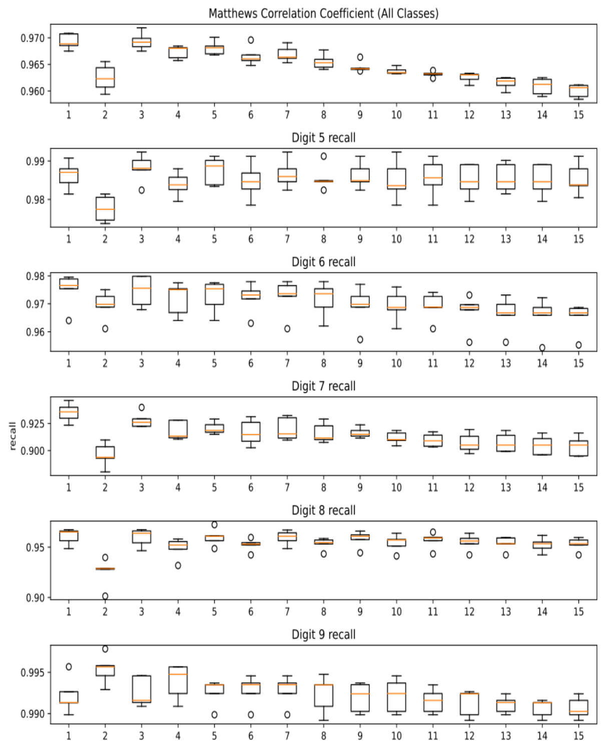

figure 10: The recall for all KNN models for digits 5 to 9 for all runs of 5-fold cross validation.

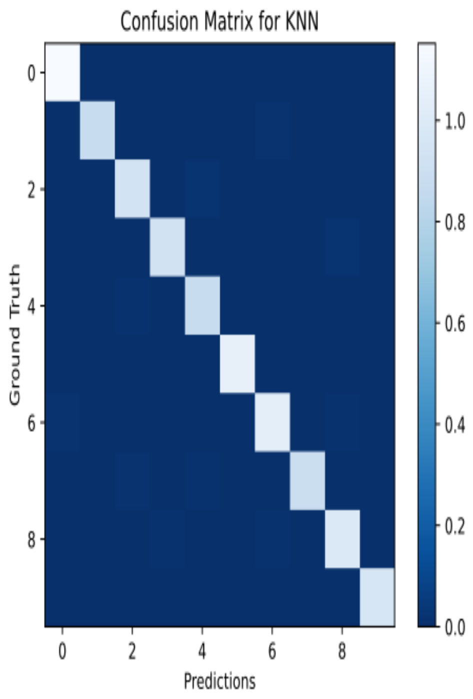

figure 11: Confusion matrix for KNN

Mean cross fold MCC decreased as the number of neighbors increased. The majority of digits exhibited this pattern with the exception of 0, 5, and 8 for which the number of neighbors resulted in similar recall. Digits with high precision exhibited lower recall with an increasing number of neighbors.

3.3 Neural Network

We used the following NNs: [784,100,10], [784,200,10], [784,300,10], [784,400,10], [784,500,10], [784,300,200,10], [784,300,100,10], [784,500,200,10], [784,500,100,10], [784,128,64,10]. Each of these is a setting of the format $[N_{in}, h_1, h_2, N_{out}]$ where , N_{in} := number of neurons in the input layer, h_1 := number of neurons in the first hidden layer, h_2 := number of neurons in the second hidden layer (if present), and N_{out} := number of neurons in the output layer of the neural network. The N_{in} and N_{out} were fixed at 784 and 10 respectively. These are a total of ten different models. The results similar to the style of the models of KNN and SVM are displayed below

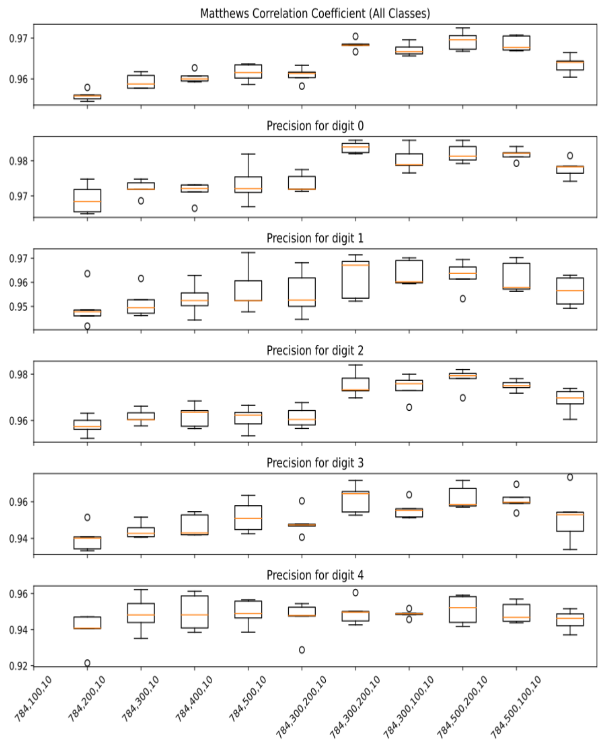

figure 12: The precision for all NN models for digits 0 to 4 for all runs of 5-fold cross validation.

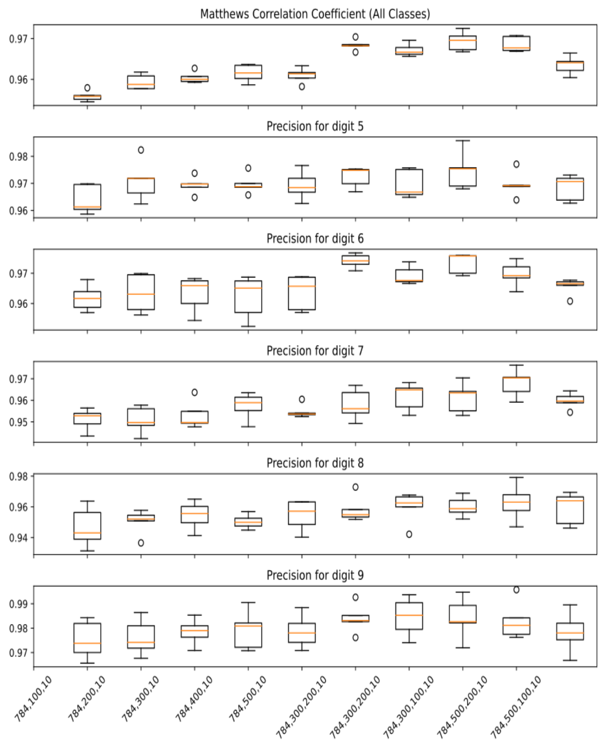

figure 13: The precision for all NN models for digits 5 to 9 for all runs of 5-fold cross validation.

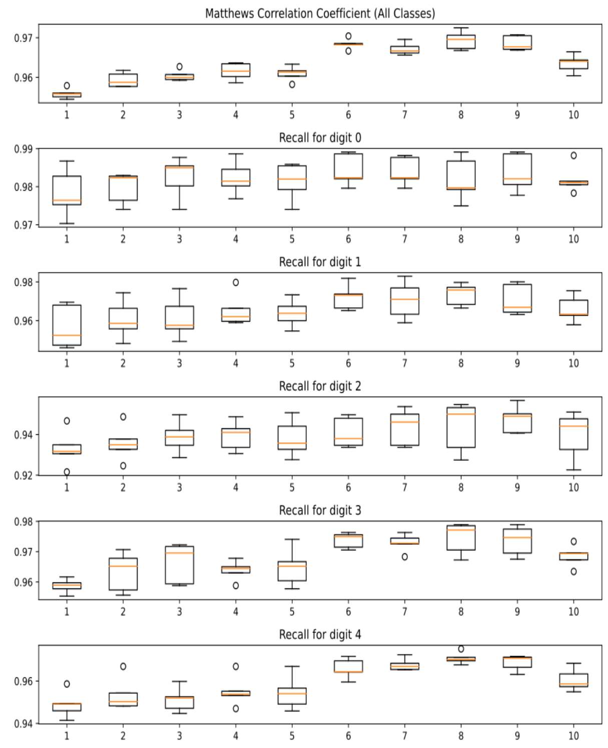

figure 14: The recall for all NN models for digits 0 to 4 for all runs of 5-fold cross validation.

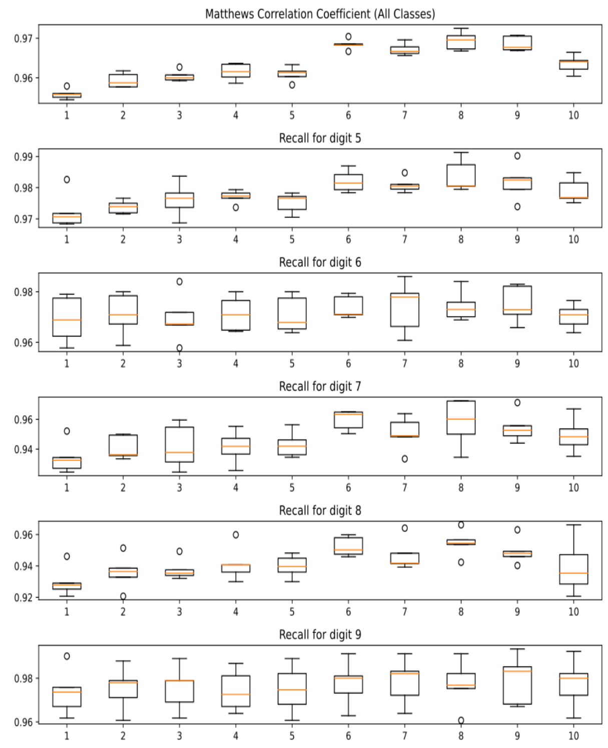

figure 15: The recall for all NN models for digits 5 to 9 for all runs of 5-fold cross validation.

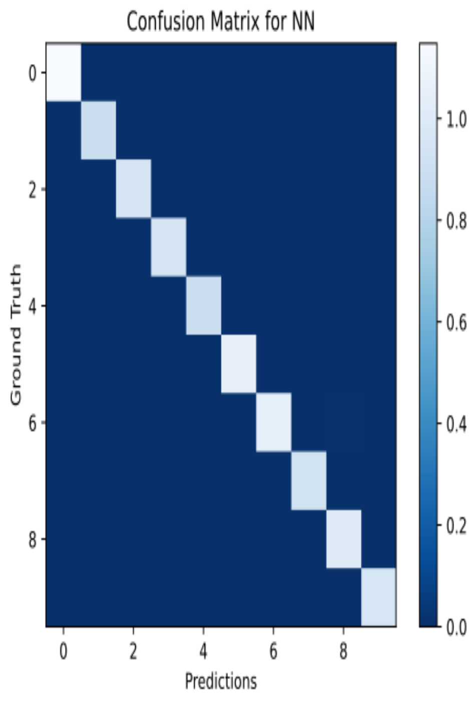

figure 16: Confusion matrix for NN

Precision and recall remained largely the same amongst digits.

4 Running the Code

Please see the file manual.txt in the submission. Please see compinst.txt file to get a list of the libraries needed for the program.

5 Conclusions and Discussion

The results show high inter- and intradigit precision and recall with a high MCC. The use of Matthew's Correlation Coefficient illustrates that among all models, predictions were aligned with their labels with few false positives and false negatives. In the multi-class case, few false positives and false negatives illustrate that classes were accurately predicted and discriminated from others

Precision and recall varied amongst digits with SVM and KNN. With respect to KNN, recall for a portion of the digits showed increased recall with decreased precision as models decreased. With SVM, recall was more affected across all digits with an increase in model complexity. Finally, NN models exhibited the effect that precision was largely affected with an increase in model complexity while recall, for most digits, remained largely the same.

ANOVA was used to find a statistically significant difference between models and Tukey's for post hoc analysis. However, in using Tukey's test we were unable to reject the null that, pairwise, means were not different. Several reasons for this may occur, one hypothesis is such that Tukey's corrects for type 1 error amongst pairs that would otherwise compound with pairwise T tests. Although the t-test is not used in ANOVA, the procedure for calculating an F statistic in this omnibus test is largely similar. Another possibility for this discrepancy is that sample sizes were small in each group.

This was replicated with the built-in ANOVA and Tukey's HSD test in the statsmodel library as well. A sample data for such a case is produced in the source folder, titled *population.txt*. This is a numpy matrix, and can be loaded using np.loadtxt('population.txt', delimiter=','). The commented out code in the file anova.py can be uncommented and run standalone to verify this. Although a very rare occurrence, it does happen. In this case, we side with Tukey's HSD conclusion, and pick the first model.

Parameters that affected mean MCC across folds usually contributed to the variance of the model. Higher degree polynomials in SVM resulted in lower performance as did a greater number of neighbors in KNN. However there is a paradoxical result with respect to NN, in which parameters that increased variance of the model tended to perform well. Additional layers provided a limited improvement in MCC as did increasing the number of nodes per layer. MCC decreased upon a sufficiently complex model. A more thorough observation of this trend should be investigated as only limited data were generated.

References

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

LeCun, Y., Cortes, C., & Burges, C. (2010). Mnist handwritten digit database. 2010. *URL http://yann. lecun. com/exdb/mnist*, *7*, 23.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2019). Pytorch: An imperative style, high-performance deep learning library.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Raschka, S., Patterson, J., & Nolet, C. (2020). Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *arXiv preprint arXiv:2002.04803*.