Final Project

MNIST Handwritten Digit Recognition

Git hub Link: https://github.com/nbhatt15397/ECE-CS532-Final-Project

Nikita Bhatt

CS/ECE 532

Fall 2020

Table of Contents

Introduction

Machine Learning has slowly made its way into every area of our modern lives. Handwritten digit recognition seems to be a machine learning problem popular among beginners and ML experts alike. Given the significance of such predictions, this semester I chose to analyze the MNIST Handwritten dataset using 3 ML models. The three models were applied, their parameters fine-tuned, and their results compared to determine the 'best' model to be used for making predictions based on different considerations. The project will introduce you to the MNIST dataset, and then analyze three distinct multi-class based classifications on the dataset. This is a large dataset, and so in the interest of time, only a small subset of the dataset was used to test and analyze these models.

Dataset Description

The MNIST Handwriting dataset [1] is a large dataset of gray-scale handwritten digits ranging from 0-9. The dataset has a training as well as a testing set. The train and test datasets contain 60,000 and 10,000 samples respectively. Each datapoint is a 28 x 28-pixel image where each pixel is treated as a feature, implying the presence of 784 features. Each feature has a number associated with it. Given that the images are in gray-scale, this number represents the intensity of the pixel. 255 represents a white pixel, 0 represents a dark pixel and numbers between 0 and 255 represent in-between shades.

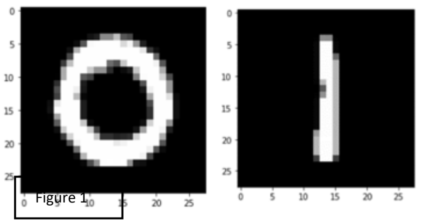
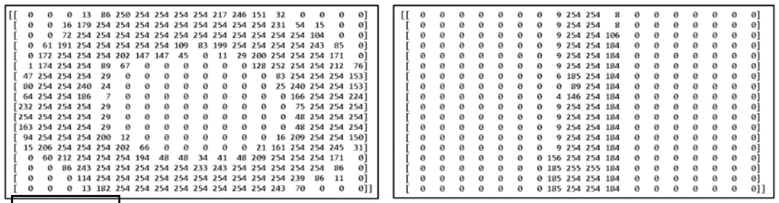


Figure 1



```
[[  0   0   0  13  86 250 254 254 254 254 217 246 151  32   0   0   0   0]
 [  0   0  16 179 254 254 254 254 254 254 254 254 254 231  54  15   0   0]
 [  0   0  72 254 254 254 254 254 254 254 254 254 254 254 254 104   0   0]
 [  0  61 191 254 254 254 254 254 109  83 199 254 254 254 254 243  85   0]
 [  0 172 254 254 254 202 147 147  45   0  11  29 200 254 254 254 171   0]
 [  1 174 254 254  89  67   0   0   0   0   0   0 128 252 254 254 212  76]
 [ 47 254 254 254  29   0   0   0   0   0   0   0   0  83 254 254 254 153]
 [ 80 254 254 240  24   0   0   0   0   0   0   0   0  25 240 254 254 153]
 [ 64 254 254 186   7   0   0   0   0   0   0   0   0   0 166 254 254 224]
 [232 254 254 254  29   0   0   0   0   0   0   0   0   0  75 254 254 254]
 [254 254 254 254  29   0   0   0   0   0   0   0   0   0  48 254 254 254]
 [163 254 254 254  29   0   0   0   0   0   0   0   0   0  48 254 254 254]
 [ 94 254 254 254 200  12   0   0   0   0   0   0   0  16 209 254 254 150]
 [ 15 206 254 254 254 202  66   0   0   0   0   0  21 161 254 254 245  31]
 [  0  60 212 254 254 254 194  48  48  34  41  48 209 254 254 254 171   0]
 [  0   0  86 243 254 254 254 254 254 233 243 254 254 254 254 254  86   0]
 [  0   0   0 114 254 254 254 254 254 254 254 254 254 254 239  86  11   0]
 [  0   0   0  13 182 254 254 254 254 254 254 254 254 243  70   0   0   0]]
```

```
[[  0   0   0   0   0   0   0   9 254 254   8   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254   8   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 106   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   6 185 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  89 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   4 146 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 156 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 185 255 255 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 185 254 254 184   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 185 254 254 184   0   0   0   0   0   0   0]]
```

Figure 2

Understanding the Dataset

In order to better visualize how the data set was structured, the Figures 1,2 and 3 were created for 2 digits, '0' and '1'. Figure 1 shows a random image of digits 0 and 1 printed from the dataset. Figure 2 shows the digits '0' and '1' as their corresponding arrays with the numbers representing pixel intensity. Figure 3 shows a heatmap representation of the digits 0 and 1 in terms of the pixel intensity of the handwritten digits

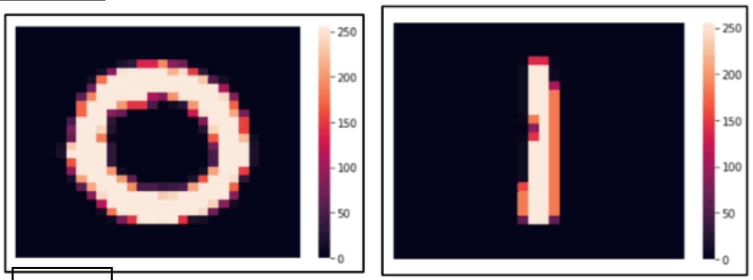


Figure 3

Algorithms

The three supervised learning algorithms used to carry out multi-class classifications were-

1) K-Nearest Neighbor;   2) Artificial Neural Network;   3) SVM Classifications

Table 1: This table shows the KNN model classification accuracy for different values of k, alongside their corresponding k values

Tested k	Classification Accuracy
K = 1	93.04%
K = 2	91.82%
K = 3	93.07%
K = 4	92.74%
K = 5	92.92%
K = 6	92.77%
K = 7	92.80%
K = 8	92.71%
K = 9	92.68%
K = 10	92.32%
K = 11	92.26%
K = 12	92.32%
K = 13	92.26%
K = 14	92.14%
K = 15	92.05%
K = 16	92.05%
K = 17	91.88%
K = 18	91.90%
K = 19	91.70%
K = 20	91.64%

Pre-Processing of Dataset

Firstly, the dataset I downloaded from Kaggle contained the training and the testing data in .csv format. On the original MNIST webpage[1], the data is zipped into .gz file format and the feature data is arranged as a 28*28 matrix instead of a flat 784-dimensional feature vector for each data point in the Kaggle files. Using the Kaggle provided .csv considerably reduced any sort of pre-processing I would have to do on the original dataset to parse through it. Secondly, the pictures in the data set have already been processed to a high degree. All images were normalized to have 28*28 pixels, anti-aliased which introduced the gray-scale aspect of the pixels, and then the digits were centered. Lastly, a pre-processing step I performed was to use the sklearn. preprocessing.scale() function to standardize the pixels(features) in the image. All the data points are divided by the standard deviation and then the mean is subtracted from each data point. This needs to be done because our dataset is sparse/widely spread out which can cause issues with our model being statistically skewed. Therefore, scaling the data in this manner makes the matrix entries have 0 mean and the standard variation of 1 in each field.

KNN Classifier

The KNN algorithm primarily assumes that feature vectors that are closer in an n-dimensional space will display similar visual features. It relies on selecting and grouping classes among the k-nearest neighbors of the data point, defined by a standard distance metric [2]. Given that the MNIST is an already well -curated data set (the data set is heavily pre-processed with perfect cropping and thresholding), KNN despite being a simple algorithm can achieve high classification accuracy.

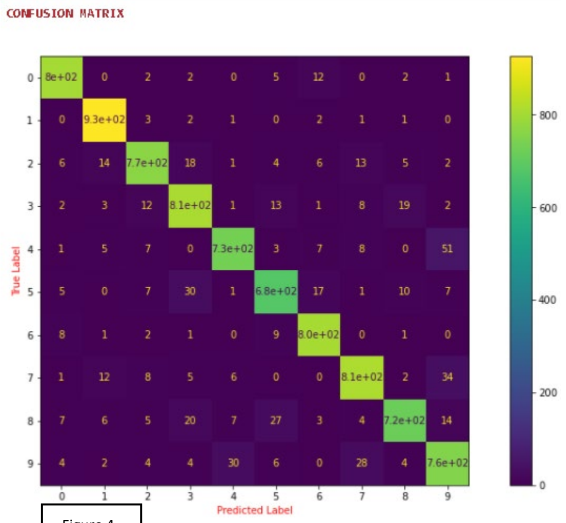
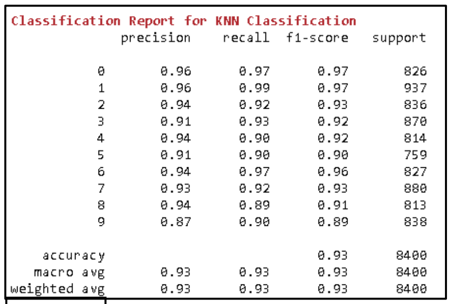


Figure 4

Hyperparameters for KNN- *Value of k*: if k is too small, the model can be more efficient, at the cost of being more susceptible to noise and outliers. If k is too large, then model bias can increase.

Distance between nearest neighbors: can be chosen from any of the well-known metrics such as Euclidean, Manhattan, cosine etc. Most papers that worked on this data set used a Euclidean distance metric [2]. In my code, I have used a pre-developed KNN classifier function from sklearn which utilizes the Euclidean distance, so we will not be altering this parameter.



```
Classification Report for KNN Classification
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       826
           1       0.96      0.99      0.97       937
           2       0.94      0.92      0.93       836
           3       0.91      0.93      0.92       870
           4       0.94      0.90      0.92       814
           5       0.91      0.90      0.90       759
           6       0.94      0.97      0.96       827
           7       0.93      0.92      0.93       880
           8       0.94      0.89      0.91       813
           9       0.87      0.90      0.89       838

    accuracy                           0.93      8400
   macro avg       0.93      0.93      0.93      8400
weighted avg       0.93      0.93      0.93      8400
```

Figure 5

Procedure- I first trained the data using a specific set with different values of k. Then, I tested the data on the validation set to observe the metrics that give the highest accuracy. Finally, I used these parameters to create a an optimal model

The data split was as follows: Training data points used: 30240, Testing data points: 8400, Validation data points: 3360. The model accuracy for each k value can be observed in Table 1. The accuracy for K=1 and K=3 was almost equal, and so I chose the ideal model to have K = 1. Subsequently, the classification data was analyzed qualitatively using the confusion matrix. The confusion matrix in Figure 4 allows us to visualize the areas where the classifier might be "confused". It gives us a heatmap type visual that will help us determine which samples were well/poorly predicted. As can be seen along the diagonal of the heatmap, most samples were correctly classified. The most types of misclassifications occurred when 4 was interpreted as a 9 and vice versa. Our final KNN-model accuracy was ***93.04***% as can be seen in Figure 5.

Neural Networks

Neural Networks are powerful tools to create models which are in the form of densely connected nodes in order to learn trends and recognize patterns.

Some basic pre-processing performed

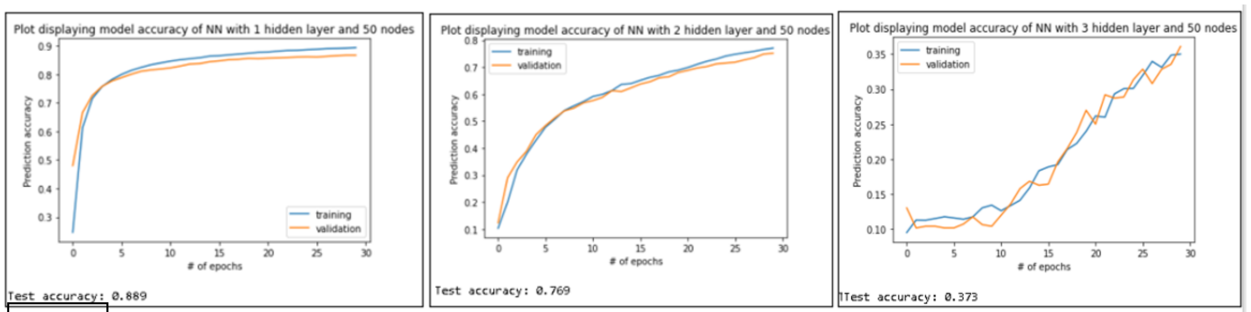
In order to prepare our data set to be used to generate a neural network model, the label vector had to be converted into a 1-hot vector. Initially, the labels are in digit form from 0-9, however this representation is inadequate. This is because, within the neural network, it is important that our algorithm treats each individual label as an independent entity rather than ordinal values. For example, within the MNIST data set, misclassifying a '9' for a '0', we should treat them as categorical sets and not as continuous sets. A 1-hot vector is a vector of length 10 with exactly one of the positions in the vector =1, and the rest = 0. After this one-hot conversion is done, we can proceed with the Neural Network Formation.

Procedure- Forming Neural Networks comes with three essential architectural decisions; the number of hidden layers in the neural network, the number of nodes in each hidden layer and the activation function to be used at each of those layers. In class we learnt that creating an accurate neural network is more of an art than a science and the best model can only be determined empirically.

In my code, I maintained the same activation function (sigmoid/Logistic) for all hidden layer neurons but changed the number of layers and number of nodes to observe how accuracy levels varied and depended on these factors for the MNIST data set. The Sigmoid Activation function was selected because it is the most popular activation function for multi-class neural networks. [4]. Other parameters that were kept constant were that of the Optimization function which was chosen to be the simplest kind, SGD, and the Loss function which was chosen to be categorical cross entropy because of the multi-class nature of the dataset.

Increasing the number of hidden layers

The number of hidden layers were changed from 1 – 3 and the accuracy of the model was plotted as a function of the # of epochs. The plots in Figure 6 show the trends in model accuracy observed as we increased the number of hidden layers from 1 to 3 but maintained the exact same structure for each hidden layer (50 nodes and sigmoid activation function). The plots show a negative trend in model accuracy as the number of hidden layers is increased. Consequently, the next stage of the experimentation was performed with just one hidden layer. Additionally, as expected, as the number of epochs increases, the model accuracy also increases.



```
Test accuracy: 0.889
```

```
Test accuracy: 0.769
```

```
Test accuracy: 0.373
```

Figure 6

Increasing the nodes per hidden layer

The number of nodes in the single hidden layer was changed to 50, 100, 500 and 1000 and the accuracy of the neural network model was plotted as a function of the # of epochs. The plots in Figure 7 depict a positive trend in model accuracy as the number of nodes is increased to 500, but the accuracy remains almost the same with a further increase in nodes to 1000. This means that the optimal number of nodes of this single hidden-layered neural network is somewhere between the 100-500 nodes ballpark.

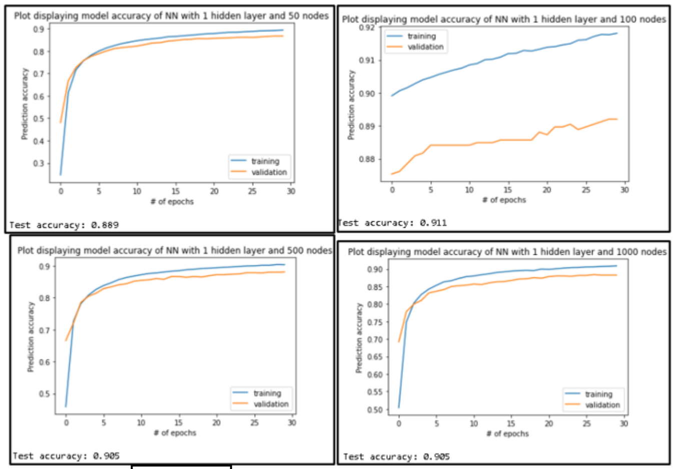


Figure 7

After experimenting around a bit with different node numbers, layers, and activation functions, the maximum Neural Network Model Accuracy I was able to obtain was ***92%*** using 3 hidden layers; layer 1 used the sigmoid activation function and had 800 nodes, layer 2 used the relu activation function and had 300 nodes, whereas layer 3 used the sigmoid function and had 300 nodes. Figure 8 shows the trends in training and validation data as a function of epochs

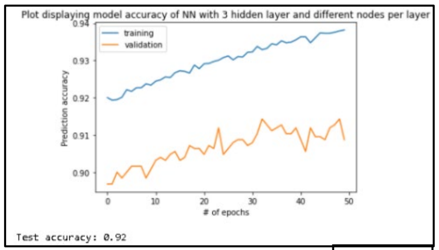


Figure 8

Classification using SVM

SVM is a supervised learning algorithm that selects the best hyperplane (decision boundary), that would maximize the margin between different classes. In this manner the SVM attempts to define an equation for a hyperplane in n-dimensions, in order to separate data in a specific class. They are predominantly used for classification and regression analysis both for linear and non-linear decision boundaries.[3]

While a Linear SVM model separates data using a straight line, non-linear data cannot be well classified in this way. That is why SVMs can be augmented with different types of kernels (such as polynomial kernels, linear kernels, gaussian kernels etc.) to allow for non-linear classifications. Because the MNIST Handwriting dataset is a non-linear dataset, a non-linear SVM algo on data would show better accuracy. Figure 9 shows a 3D mapping of the neighbor embeddings in the MNIST Handwriting data set for visualization. It is clear that the data set cannot be classified well with a simple linear SVM.

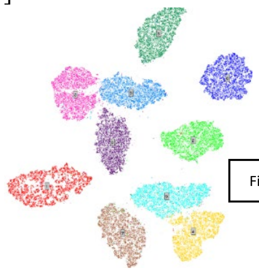


Figure 9

Procedure- During my experimentation, I first observed the prediction accuracy with a linear kernel SVM and a non-linear RBF based SVM with default parameters of the functions. After this, the hyperparameters were tuned using cross-validation on the RBG based SVM model to determine the best hyperparameters. Given that SVM is a very computation dense procedure, the size of the testing and training data sets was selected as 5% (2100 samples) and 15 % (6300 samples) of the 42k datapoints, respectively. Furthermore, the data was split such that the number of samples of each number would be balanced so as to not skew the decision boundary.

Hyperparameters- *Gamma*: This parameter defines the reach of the influence of a specific training sample. Low values imply a father influence, whereas high values imply and closer influence.

C/ Cost: The cost or C parameter behaves like a regularization parameter in SVM. Smaller values of C are associated with a larger margin and simpler smoother decision boundary at the cost of higher accuracy whereas a higher C is associated with a tighter decision boundary.

Linear Kernel SVM without parameter tuning

Because the dataset is not a linear one, the accuracy of this model was in line with our expectations.The model obtained a **90%** accuracy with these default parameters. A confusion matrix (Figure 10) was also plotted to observe the classification graphically.

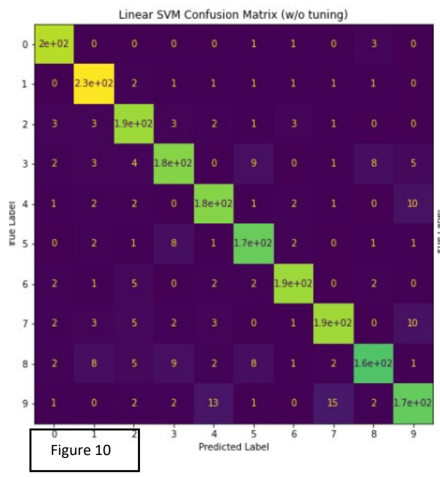


Figure 10

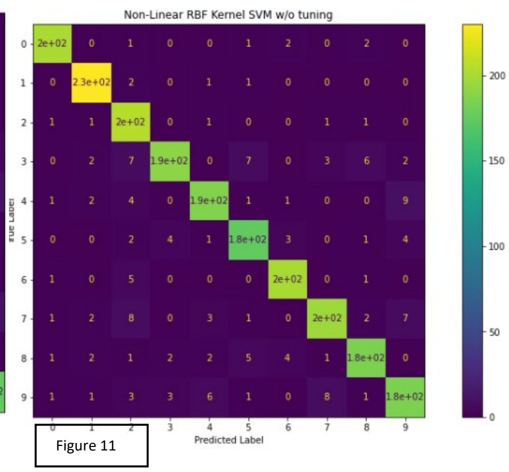


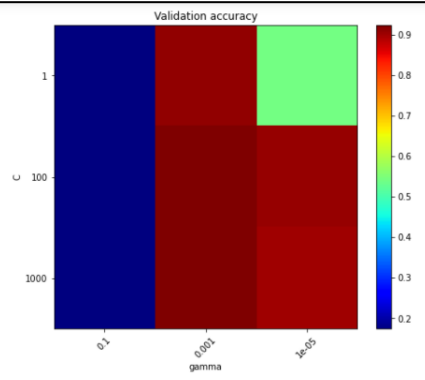
Figure 11

Non-Linear RBF Kernel SVM without parameter Tuning

Because the dataset is non-linear one, the accuracy of this model was in line with our expectations. The model obtained a **92.97%** accuracy with these default parameters. A confusion matrix (Figure 11) was also plotted to observe the classification graphically.

Parameter Tuning for RBF Kernel SVM

In order to obtain the 'best case' hyper parameter values for the rbf kernel based SVM, I first assigned 3 different values of C and gamma to carry out cross-validation with. C was assigned values of 1, 100, 100 whereas gamma was assigned values of 1e-1, 1e-3 and 1e-5. Subsequently, these values were used to perform cross-validations and evaluate the best parameter combinations on the basis of model validation accuracy. The top part of Figure 12 has a heat map to pictorially depict the validation accuracy for different combinations of C and gamma. It was found that Gamma value of 0.001 and C value of 100 created the model with the highest average prediction accuracy while using this Support Vector Classifier. The classification report allowed us to view the prediction accuracy for different numbers, and it can be noted that some numbers had considerably higher chance of being predicted accurately than others. On average, the rbf kernel SVM gave us a prediction accuracy of ***94%.***



```
Classification report for classifier SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       207
           1       0.95      0.97      0.96       234
           2       0.87      0.98      0.92       209
           3       0.95      0.90      0.92       217
           4       0.94      0.93      0.94       204
           5       0.93      0.94      0.93       190
           6       0.97      0.97      0.97       207
           7       0.93      0.91      0.92       220
           8       0.96      0.92      0.94       203
           9       0.90      0.88      0.89       209

    accuracy                           0.94      2100
   macro avg       0.94      0.94      0.94      2100
weighted avg       0.94      0.94      0.94      2100
```

Figure 12

Comparison of the three models

The three algorithms namely SVM, KNN and Neural Networks are all good at making predictions. It is difficult to determine through my experimentation which would be the best to classify the MNIST dataset. This is because there are many more criteria that could be fine-tuned (that I was not able to fine tune in my project due to time constraints). Through my analysis, I found that using KNN proved to be the easiest and least time-consuming algorithm to train, but was somewhat slower to compute the results. The KNN model had an average prediction accuracy of 93%. While SVM techniques have great caliber to make accurate predictions, they were exceedingly time consuming to train and test and led to my kernel dying several times possibly because of their high computational density. Despite the struggles experienced while running the algorithm, the SVM model made with optimal hyperparameter values showed the most promising prediction accuracy (94%) out of all the models. The neural network model was slow to train but relatively quick to test. Through my empirical experimentation, I was only able to get my model to a prediction accuracy of 92%. In my opinion, neural networks are the best models to build with regards to computational ease and accuracy. Even though arriving at a very high accuracy might take considerable amounts of empirical experimentation, the model eventually runs smoothly and is capable of very high prediction accuracies.

Strengths and Limitation of the Models

The following table discusses some limitation and strengths associated with each of the classifiers used in this project:

Table 2: This table contains the limitations and strengths of each of the three models covered in this project

Limitations		
KNN	**SVM**	**Neural Network**
The presence of noise can lead to higher misclassifications. It can prove to be useful to carry out dimensionality reduction to extract useful features. This would reduce noise to some extent and also give better predictions.	Due to the high computational intensity of the SVM algorithm, I could only test out 3 different values of the two hyperparameters (C and gamma). The 'optimal' hyperparameter values selected could have been more fine-tuned to increase model accuracy.	Given that the image data was flattened into one single 784 pixel vector instead of it's actual 28*28 size, some information about which pixels are next to each other was bifurcated contributing to mispredictions. In order to maintain such spatial relationships convolutional neural networks are often used.
Strengths		
KNN	**SVM**	**Neural Network**
It is the simplest and easiest to implement. No training and testing phases like in other algorithms are present. It is also great for when there are non-linear boundaries, and we have large amounts of data like in the case of the MNIST dataset.	Support Vector Machine based classifiers are relatively robust and have features in place that allow them to ignore outliers and look for a maximum margin hyperplane.	Neural networks are rapidly able to perform predictions after training in comparison to SVM. This occurs because the number of weight matrix multiplications and activation functions is directly dependent on the number of hidden layers. .

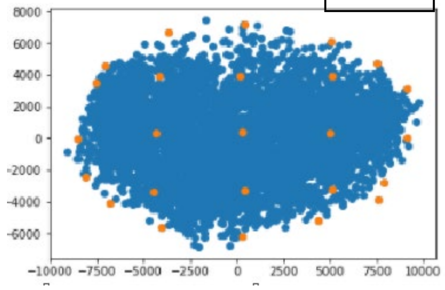
Conclusions & Future Scope

In this project, I used 3 different approaches for Supervised Machine Learning classification of the MNIST Handwritten digits data. Working on this project allowed me to understand the highly convoluted world of machine learning models. Although, all attempts were made to evaluate and optimize the various algorithms discussed in this project, there still remain a plethora of different parameters that could have been experimented with for all three models. Moreover, accuracy of prediction can also vary greatly depending on the training and testing sample size. Each of the models were trained and tested on a small subset of the massive dataset. Naturally, increasing the training size would have led to higher prediction accuracies. With regards to future scope, I am of the opinions that the true usefulness of a model is revealed when it is applied in real life. Given that the MNIST dataset is very well curated and formatted, testing the models on samples from the dataset does not represent real-life examples, which are not as well-formulated. It would have been interesting to see how the models would have performed while predicting digits written by me. Overall, this project was a great learning experience and allowed me to explore different ML algorithms and their intricacies practically.

Appendix: Unsupervised Learning with Isomap

While exploring possible dimensionality reduction techniques, I stumbled upon a recently researched unsupervised learning technique called Isomap. Isomaps estimate geodesic distances between data points and then use Multi-Dimensional Scaling to induce low dimensional manifolds. In the case of the MNIST handwritten dataset, the data lives in a high-dimensional space, but effectively can be narrowed down into a low-dimensional manifold. Isomap helps us by finding a mapping that recovers this manifold. This manifold unsupervised learning process uses the raw training data (independent variables) to extract meaningful sets of coordinates that characterize important features about the data by considering the manifold structure of the data [5].Isomap is the best way to carry out dimensionality reduction for a data set as non-linear as the MNIST Handwritten data set as it uses non-linear geodesic distance approach among the multivariate data points. Other popular techniques like PCA, LDA etc., would not give us proper results as most of them are linear methods that reduce dimensions based on the Euclidean distances.

Figure 13



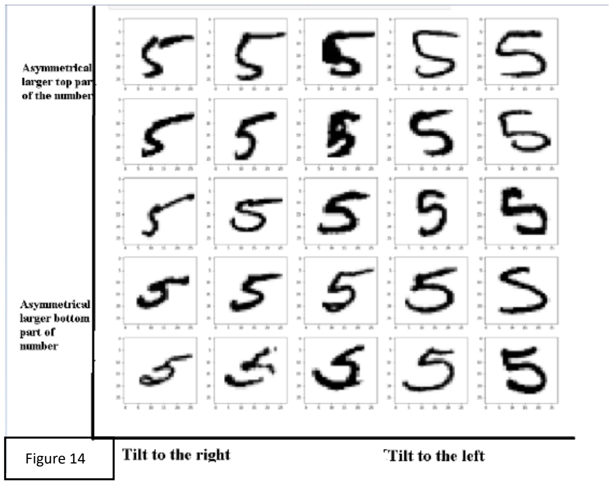

Asymmetrical larger top part of the number
Asymmetrical larger bottom part of number
Tilt to the right
Tilt to the left

Figure 14

Procedure: Only looked at the Isomap of the digit '5' to look at the dataset in a more digestible manner. The two main parameters in the Isomap are n_neighbours and n_components. *n_neighbours* : is the number of nearest neighbors we want to consider for dimensionality reduction; *n_components* : is the number of dimensions we want to extract In order to correlate this plot with some features of the digit 5.

I plotted a grid to select equidistant landmark points pertaining to the digit '5' as seen in Figure 13. Subsequently, Figure 14 plots the digits associated with the landmark points to observe visual trends that the Isomap technique learned.

References

[1] "THE MNIST DATABASE," MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[2] D. Grover and B. Toghi, "MNIST Dataset Classification Utilizing k-NN Classifier with Modified Sliding-window Metric," *arXiv:1809.06846 [cs]*, Mar. 2019, Accessed: Dec. 04, 2020. [Online]. Available: http://arxiv.org/abs/1809.06846.

[3] S. Ghosh, A. Dasgupta, and A. Swetapadma, "A Study on Support Vector Machine based Linear and Non-Linear Pattern Classification," in *2019 International Conference on Intelligent Sustainable Systems (ICISS)*, Feb. 2019, pp. 24–28, doi: 10.1109/ISS1.2019.8908018.

[4] S. Sharma, "Activation Functions in Neural Networks," Medium, 06-Sep-2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 04-Dec-2020].

[5] J. B. Tenenbaum, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000, doi: 10.1126/science.290.5500.2319.