# Collaborative Intelligence: Enhancing Digit Recognition Through Human-Machine Partnerships

From Data Analysis to Real-World Application

Magdalena Wallner

EC Utbildning

Kunskapskontroll ML 2024

2024-03

## Abstract

This study examines the MNIST dataset for handwritten digit recognition, with an emphasis on exploratory data analysis (EDA) to discover essential data insights.

It compares different machine learning models to identify the most effective one for accurately predicting digits. The evaluation process focuses on metrics such as precision, recall, ultimately selecting the top-performing model for further application.

This model is integrated into a Streamlit application, enabling real-time digit prediction from user-uploaded images.

The project illustrates the practical application of machine learning in digit recognition tools, and to also the importance to create partnership between human analytical skills and machine precision.

Keywords: MNIST, Data Analysis, Machine Learning, Image Classification, Human-Machine Collaboration

## Content

## 1 Introduction

In the current era fully focused on artificial intelligence and machine learning, our daily interactions with technology that are developed by sophisticated algorithms are increasing and evolving Technology that once seemed accessible only to those with technical expertise is now engaging non-technical users, enabling conversations with AI across various platforms. This shift marks a significant democratization of AI technologies, making them more integral to our daily lives and accessible to a broader audience.

From unlocking smartphones with facial recognition to detect and diagnose diseases from images such as X-rays, MRIs and CT scans, image recognition technologies have become a part of our daily private and work-life.

This transition marks a significant shift in how we interact with technology: it's no longer just about machines learning to understand us, but also about us learning to interact with machines in more intuitive and meaningful ways.

The purpose of this report is to understand the MNIST dataset in an exploratory data analysis, EDA, and compares various machine learning models to identify the most effective algorithm for digit prediction and results in the development of a Streamlit application that predicts digits from user-uploaded images.

The following questions will be answered:

1) How can the collaboration between human analysis and machine learning algorithms enhance our understanding and utilization of the MNIST dataset?

2) Which model emerges as the most effective for accurate digit prediction?

3) How much precise are these models at predicting personal photos?

As a conclusion it underscores the pivotal role of preprocessing and model selection in enhancing the interface between humans and machines, facilitating a more integrated and interactive technological experience.

## 2 Theory

The models used are Support Vector Classifier (SVC), RandomForestClassifier, and KNeighborsClassifier. When working with the MNIST dataset, selecting the right classifier is important for achieving high accuracy, which is one of the goals for this projet. The MNIST dataset is a high-dimensional space, given its pixel-by-pixel representation of images, making it an excellent candidate for Support Vector Classifier (SVC), Random Forest Classifier, and k-Nearest Neighbours (kNN) classifiers. Each of these classifiers has characteristics that can influence their performance on the MNIST dataset.

## 2.1 SVC

SVC is one of the choices, since it's good in handling high-dimensional data like MNIST. It works by finding the hyperplane that best separates the digits into classes with the maximum margin. The effectiveness of SVC in this context is due to its ability to handle complex decision boundaries.

SVC finds the optimal boundary between data classes, a task requiring precision beyond human computation. They are supervised learning methods used for, among other things, for classification problems.

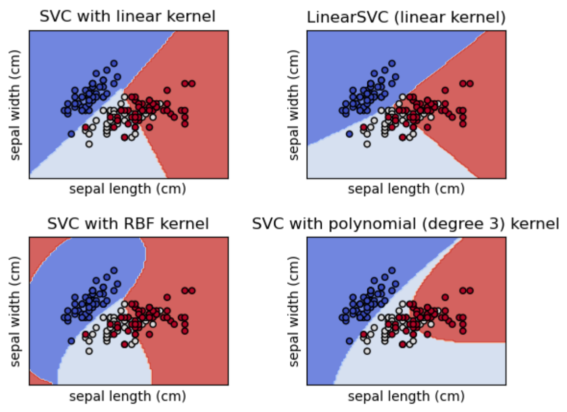
Support Vector Machines (SVMs) are recognized for their effectiveness in high-dimensional spaces, efficiency in memory usage, and versatility across different problem types ("Support Vector Machines," n.d.)

### 2.1.1 Parameters

Support Vector Classifier aims to find the best boundary that separates classes in the data. The 'C' parameter controls the trade-off between a smooth decision boundary and classifying training points correctly; a higher 'C' increases gives less margin violations, but to high C value can lead to overfitting of the data ("07_svm.ppt," n.d.). It's effective to decide the C parameter with Gridsearch in scitkit-Learn

$\gamma$, 'Gamma', determines the influence of points on the decision boundary. When set to 'scale', it is auto-calculated based on the data's features (1/(number of features * variance of X) as its value). 'Kernel' specifies the algorithm type; a 'rbf' kernel enables the model to create non-linear boundaries between classes. This flexibility allows SVC to handle complex datasets where classes cannot be separated by a straight line.

("Support Vector Machines," n.d.)



("Support Vector Machines," n.d.)

## 2.2 Random Forest Classifier

Random Forest is a versatile machine learning algorithm capable of performing both regression and classification tasks. It is also used for dimensionality reduction, treats missing values, outlier values, and other essential steps of data exploration.

As Leo Breiman, who introduced the Random Forest algorithm, explains, "Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest" (Breiman, 2001). This fundamental principle behind Random Forests emphasizes the algorithm's reliance on randomness and ensemble learning to achieve high accuracy and robustness against overfitting.

### 2.2.1 Parameters

The Random Forest algorithm is influenced by several parameters, but the most significant ones include the number of trees in the forest (n_estimators), the number of features considered for splitting at each leaf node (max_features), and the depth of each tree (max_depth). The parameter n_estimators adds more models to the ensemble.

The max_features determines the randomness of the forest. A lower value of max_features ensures that the trees in the forest are more diverse, making the ensemble model more robust to overfitting. max_depth controls the depth of the trees. Deeper trees are more complex and have a higher risk of capturing noise in the training data.

## 2.3 KNeighborsClassifier

The k-Nearest Neighbors (kNN) algorithm is a machine learning technique used for classification and regression. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

### 2.3.1 Parameters

Parameters like n_neighbors define the number of neighbors to use for queries. The weights parameter can be 'uniform' (all points in each neighborhood are weighted equally) or 'distance' (points are weighted by the inverse of their distance). The algorithm parameter specifies the algorithm used to compute the nearest neighbors, including 'auto', 'ball_tree', 'kd_tree', and 'brute'. Adjusting these parameters helps in fine-tuning the model's accuracy and performance on datasets like MNIST. For more detailed information, you can visit the scikit-learn documentation.

n_neighbors: This parameter specifies the number of neighbors to use for the k-nearest neighbors. Having too few neighbors might make the model sensitive to noise in the dataset, while too many can cause the model to be overly generalized.

weights: This parameter determines the weight function used in prediction. The options are 'uniform', where all points in each neighborhood are weighted equally, and 'distance', where points are weighted by the inverse of their distance to the query point. This means closer neighbors of a query point will have a greater influence than neighbors that are further away.

algorithm: This parameter indicates the algorithm used to compute the nearest neighbors. The options include 'auto', where the algorithm attempts to decide the most appropriate algorithm based on the values passed to fit method; 'ball_tree', 'kd_tree', and 'brute', each representing different approaches to finding the nearest neighbors. 'ball_tree' and 'kd_tree' are more efficient for datasets with a large number of samples, while 'brute' is a brute-force search. By testing each, you're looking for the fastest and most efficient method for your specific dataset.

# 3 Metod

## 3.1 Approach overview

Since one part of the project was to understand how the collaboration between a student and machine learning algorithms can enhance the understand of the MNIST data set, is using the scikit-learn library. The Library was started in 2007 by David Cournapeau at Google, but has many more contributors, Gitbub-scikot-learn.

Scikit-learn is highly regarded for its accessibility to beginners. The library's design for ease of use—much like engaging with ChatGPT—enables people to experiment with and learn from practical examples, reducing the barrier to entry in the world of AI and machine learning. Scikit-learn's extensive documentation and tutorials further support. Meaning, you don't have to be an expert to create a model that can predict a number.

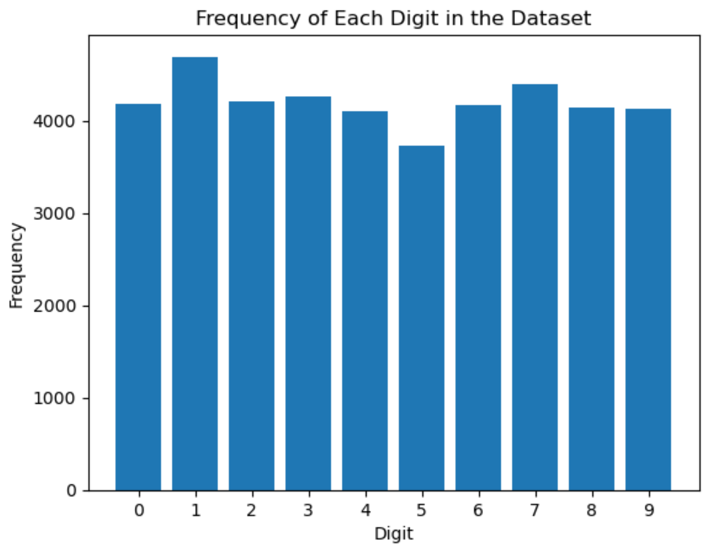
## 3.2 Exploratory data analysis

The initial step involved understanding the MNIST dataset, which was the most time-consuming part of the project. It was necessary to learn about its collection process, the characteristics of the handwritten numbers, and the image processing techniques applied. This task extended beyond performing a simple exploratory data analysis (EDA). Only after acquiring a thorough knowledge of the dataset could I proceed to preprocess the images for prediction.

By using print(mnist.DESCR), one get an understanding of the data set. It consists of 70,000 grayscale images of handwritten digits, each measuring 28x28 pixels, aimed at training and testing digit recognition models.

"For certain classification methods, notably those based on templates such as SVM and K-nearest neighbours, aligning the digits to the center using a bounding box rather than their center of mass is known to enhance the error rate" (Source: MNIST dataset description, accessible via print(mnist.DESCR) in your Python environment).
This emphasizes the importance of preprocessing steps in improving model accuracy for digit recognition tasks.

### 3.2.1 Frequency of each digit in the dataset

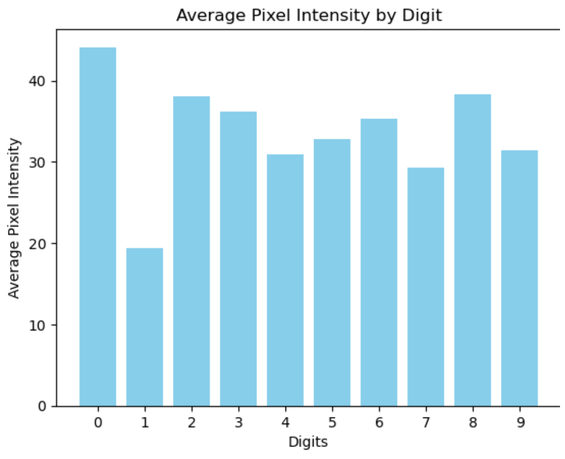


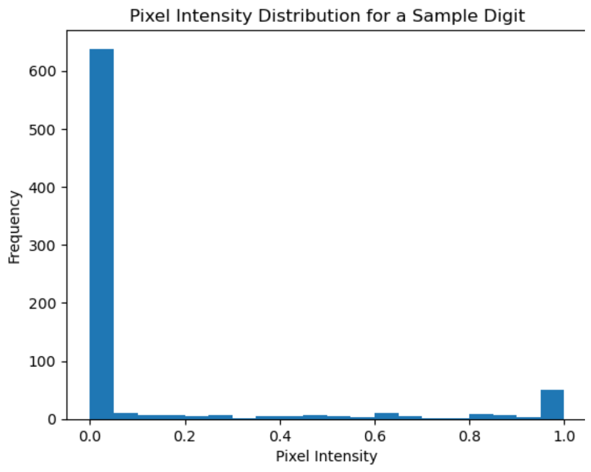
The dataset shows a balanced distribution among digits, with the frequency of digit 1 slightly exceedi ng that of digit 5. To prepare for image prediction, understanding the proportion of white to black pix els in the digits was crucial. For example, digit 0's pixel intensities range from 0 (white) to 255 (black), with an average intensity of 44.12, indicating a predominance of lighter pixels. The appendix provides detailed analyses.

```
Digit: 0, Max intensity: 255, Min intensity: 0 Digit: 0, Average intensity: 44.11847709517132
```

See appendix for full coverage.

Most of the images consist of white pixels, value 0, but spanning from 0 to 255. When normalized (divided into max value, which is 255), between 0 and 1. The average per number/image would be around 30.

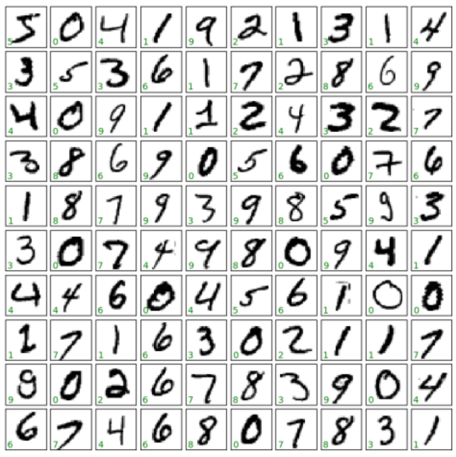


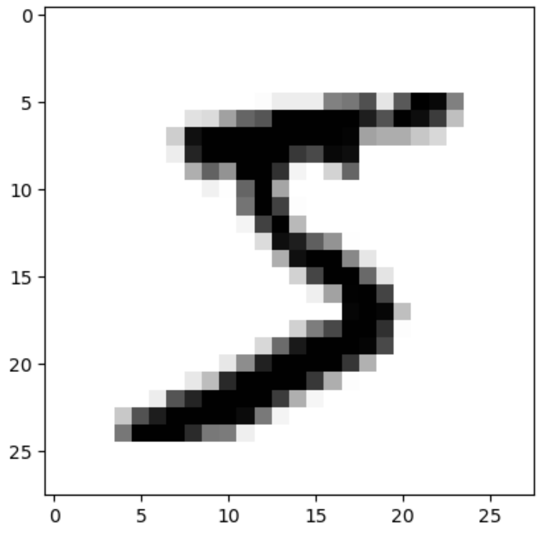


*3.2.1.1 Average Pixel Intensity by Digit 1*

*3.2.1.2 Pixel Intensity distribution for sample 1*

After analysing the digital composition of the dataset, a visual examination was crucial to assess the presence and proportion of black and grey pixels within the images. This step was essential for understanding the varying shades represented by pixels, from pure black (value 1) to various levels of grey (values between 0 and 1), contributing to the overall appearance and distinguishing features of each digit.





The insight was a crucial step knowing how to preprocess other images for accurately predicting new images based on pixel arrangement, thickness, and placement.

## 3.3 Data preparation

The dataset fetches the MNIST dataset using fetch_openml, storing the data in X and labels in y. These arrays are then divided into training and test sets, with a test size of 20% and using stratification to maintain label proportions across splits.

Additionally, the training set is further split into training and validation sets to facilitate model training and subsequent validation. Following the splits, the data undergoes normalization, scaling pixel values from a range of 0-255 to 0-1, enhancing model performance.

## 3.4 Model selection

The goal of this step was to identify the best machine learning model for digit classification. The selection process considered various factors, including accuracy. The models chosen for evaluation were SVC, RandomForestClassifier, and KNeighborsClassifier, detailed in the Theory section. Each model underwent parameter tuning via GridsearchCV, aiming to optimize for accuracy:

SVC parameters included 'C' values of 0.1, 1, and 10, 'gamma' settings of 'scale' and 'auto', and 'kernel' types 'rbf' and 'linear'.

RandomForestClassifier was adjusted for 'n_estimators' (50, 100, 200), 'max_depth' (None, 10, 20), 'min_samples_split' (2, 5), and 'min_samples_leaf' (1, 2).

KNeighborsClassifier settings covered 'n_neighbors' (3, 5, 7), 'weights' ('uniform', 'distance'), and 'algorithm' ('auto', 'ball_tree', 'kd_tree', 'brute').

Through GridsearchCV, using a parameter grid and setting cv=5 and scoring to 'accuracy', the models were fitted and evaluated. Additionally, a Voting Classifier was applied to ascertain if a combined model approach could yield better accuracy. The performance of each model was documented and saved for comparison.

### 3.4.1 Model Evaluation

The models were evaluated based on accuracy per class/digit, precision, recall and F1 Score

The top-performing model will be further trained with BaggingClassifier, utilizing additional data, and will be saved for comparison against its previous version. Its performance will be visualized using a confusion matrix to pinpoint classification challenges. Additionally, the ROC curve will be plotted. The ROC curve will be plotted.

## 3.5 Image preparation for image recognition

To ensure uploaded images in the Streamlit app are recognizable, they must resemble the training data

Step 1: involves uploading images, converting them to grayscale, resizing to 28x28 pixels, and loading them into an array to record the maximum and minimum values. Unlike the MNIST dataset normalized with a max of 255 and min of 0, these images are adjusted to the same scale for consistency. Inversion is the final step, aiming for a white background with a black digit.

Step 2 After extensive threshold testing for optimal white and gray nuances, images are processed with CV2.threshold at a value of 165 to achieve a black background with a primarily white center and some grey pixels.

Step 3: addresses the often too-thin digits by thickening the images using cv2.dilate with specified iteration and kernel sizes.

Step4: ensures image centering, following the "center image by bounding box" method described for MNIST, by aligning the image's center with its bounding box.

Step 5: Finally, images are normalized (divided by 255) to match the training data's scale.

The processed images are then predicted by the model, and performance, including accuracy and classification details, is evaluated, and displayed alongside a confusion matrix. Once this is done, the images are predicted by the best model and evaluated with its ability to recognize the correct image and print the information but also print the accuracy score and classification report with the confusion matrix.

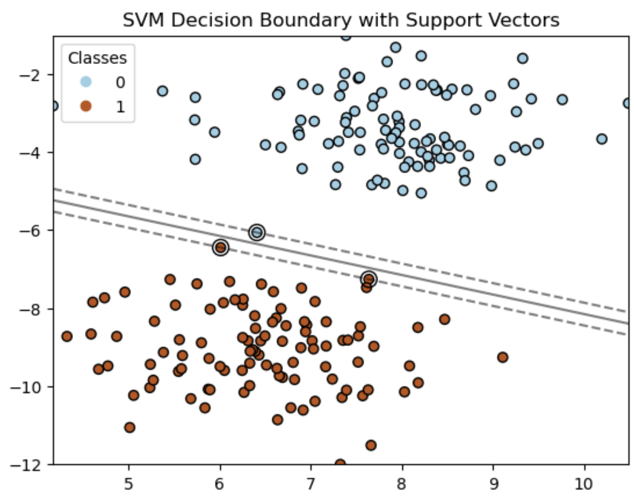
# 4 Results and discussion

| **Parameters found by Gridsearch** | | **Accuracy Score** | **F1** |
|---|---|---|---|
| Best parameters for SVC: | 'C': 10, 'gamma': 'scale', 'kernel': 'rbf | 0.979 | 0.98 |

| | | | |
|---|---|---|---|
| Best parameters for RandomForestClassifier | max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200 | 0.965 | 0.97 |
| Best parameters for KNeighborsClassifier | 'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'distance' | 0.968 | 0.97 |

*Tabell 1 Model evaluation*

SVC was selected due to that it had the best score in evaluating numbers as 7 and 9, which is a problem with new images. And this was confirmed by the Confusing matrix.

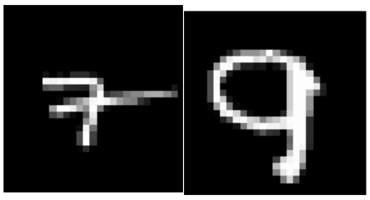


## 5 Conclusion

Working with the scikit-learn library was an accessible introduction to selecting and applying machine learning models, thanks to its user-friendly design tailored for beginners. However, it is not as easy as just “plug and play”, since it begins with understanding the dataset or "problem", which required human insight, underscoring the importance of human input and collaboration. This teamwork and interaction with the data were crucial for success, highlighting the balance between automated tools and human analytical skills in solving problems effectively.

The score of the different models were very high so more testing wasn’t necessary. The misclassified and it confirmed the score for each number, that there are problems with 9 and

The performance of the models was impressively high, eliminating the need for further testing. Analyzing misclassified images (images was printed ) revealed particular difficulties with distinguishing between the numbers 9 and 7, likely due to variations in how 7 is written, especially when it includes a horizontal line. When testing with my own versions of the number 7, both with and without the line, the model successfully recognized them, though with a slightly reduced accuracy. This observation underscores the nuanced challenges in pattern recognition that models face, especially with handwritten characters.



This model was further trained with baggingclassifer, but the score didn't improve, especially not for the critical numbers, so that models was not saved to be used.

This was confirmed by the ROC curve for all classes.

3) How precise are these models at predicting personal photos, and what implications does this precision have for future applications?
The selected model's performance on user-uploaded photos was incredible high, assessing its practicality and potential applications, such as digitalizing handwritten notes or enhancing accessibility through digit recognition.

It is with a very high accuracy that it predicts the photos, that was written on a white paper with crayon pen.

The model performed less well with the number 8, but still predicted with high score(81%) so no need to go further with that in this project. But it could be due to that I had focused on what the prediction of  7 and 9 in the training and test data, where we had not that man problems with the number 8.

However, I would expect that this solution would be better then a human is to recognize the number, since it would be a fairly easy task. I believe that increased knowledge and experience would provide a more accurate solution. A adjustment that could be done is to 1) Work with the space inside the numbers, such as 9 and 8. Since the numbers are made thicker, this space is decreasing and could be a factor in it's ability to recognize the number.

When researching about the solution, there were a lot of suggestions using Keras and Neural networks, so that could also be another solution to improve the accuracy.

# 6 Self evaluation

1. Utmaningar du haft under arbetet samt hur du hanterat dem.
   1) Har haft stora utmaningar med python eftersom jag inte är så bra på det (än) och har fått söka mkt efter lösningar, och har inte haft den tid jag önskat att gå igenom alla Datacamp kurser för just python.
   2) Skriva rapport. Valde engelska för det är det språket jag är van att skriva på i arbetet, men jag har brist på ordförråd och har använt mig av ChatGPT för att skriva om vissa texter, med fokus på att få hjälp grammatiskt. Därefter gått igenom texterna för att ta bort överdrivna ord som den tenderar att lägga in.
   3) I alla utmaningar jag haft har jag diskuterat med klasskompisar som varit mycket behjälpligt och peppande.

2. Vilket betyg du anser att du skall ha och varför.

Jag är så nöjd att jag har löst det, har lärt mig hur en dator ser på bilderna och hur jag måste lära mig att anpassa mitt sätt att ”kommunicera” med datan för att få fram de insikterna jag behövde. Jag kommer aldrig, och siktar inte i detta arbete att komma på de bästa python-lösningar. Mitt fokus har varit hur jag och de verktyg som finns bästa kan samarbeta för att lösa det, och jag anser att tröskeln är förvånansvärt låg (om jag gjort rätt). Självklart pga Scikit-learn.
Så, betyg är som vanligt inte viktigt för mig.

3. Något du vill lyfta fram till Antonio?
   Ja, jag önskar en recap på sista lektionen. Det kommer naturligt när vi har prov, men det är mkt värt att få några timmar av vad vi lärt oss under kursen, och diskutera det ihop, igen.

# Appendix A

```
MNIST data set Intensity
Digit: 0, Max intensity: 255, Min intensity: 0 Digit: 0, Average intens
ity: 44.11847709517132

Digit: 1, Max intensity: 255, Min intensity: 0 Digit: 1, Average intens
ity: 19.411879402155233

Digit: 2, Max intensity: 255, Min intensity: 0 Digit: 2, Average intens
ity: 38.08761426192741

Digit: 3, Max intensity: 255, Min intensity: 0 Digit: 3, Average intens
ity: 36.188118601499724

Digit: 4, Max intensity: 255, Min intensity: 0 Digit: 4, Average intens
ity: 30.965696586026457

Digit: 5, Max intensity: 255, Min intensity: 0 Digit: 5, Average intens
ity: 32.80824271547268

Digit: 6, Max intensity: 255, Min intensity: 0 Digit: 6, Average intens
ity: 35.350084300434766

Digit: 7, Max intensity: 255, Min intensity: 0 Digit: 7, Average intens
ity: 29.302138986148115

Digit: 8, Max intensity: 255, Min intensity: 0 Digit: 8, Average intens
ity: 38.331265232269025

Digit: 9, Max intensity: 255, Min intensity: 0 Digit: 9, Average intens
ity: 31.42270451411371
```

### 6.1.1 Model Evalutation

```
SVC Model Evaluation
Accuracy: 0.9816
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1381
           1       0.99      0.99      0.99      1575
           2       0.98      0.98      0.98      1398
           3       0.98      0.98      0.98      1428
           4       0.98      0.98      0.98      1365
           5       0.98      0.97      0.98      1263
           6       0.98      0.99      0.99      1375
           7       0.98      0.98      0.98      1459
           8       0.98      0.97      0.98      1365
           9       0.97      0.97      0.97      1391

    accuracy                           0.98     14000
   macro avg       0.98      0.98      0.98     14000
weighted avg       0.98      0.98      0.98     14000

RandomForest Model Evaluation
Accuracy: 0.9659
              precision    recall  f1-score   support
           0       0.97      0.99      0.98      1381
           1       0.98      0.98      0.98      1575
           2       0.96      0.96      0.96      1398
           3       0.96      0.96      0.96      1428
           4       0.97      0.96      0.97      1365
           5       0.97      0.95      0.96      1263
           6       0.97      0.98      0.97      1375
           7       0.97      0.97      0.97      1459
           8       0.96      0.96      0.96      1365
           9       0.94      0.95      0.94      1391

    accuracy                           0.97     14000
   macro avg       0.97      0.97      0.97     14000
weighted avg       0.97      0.97      0.97     14000

KNeighborsClassifier Model Evaluation
Accuracy: 0.9696
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1381
           1       0.96      0.99      0.98      1575
           2       0.99      0.96      0.97      1398
           3       0.97      0.97      0.97      1428
           4       0.98      0.95      0.97      1365
           5       0.97      0.96      0.96      1263
           6       0.97      0.99      0.98      1375
           7       0.96      0.98      0.97      1459
           8       0.98      0.94      0.96      1365
           9       0.95      0.96      0.95      1391

    accuracy                           0.97     14000
   macro avg       0.97      0.97      0.97     14000
weighted avg       0.97      0.97      0.97     14000
```

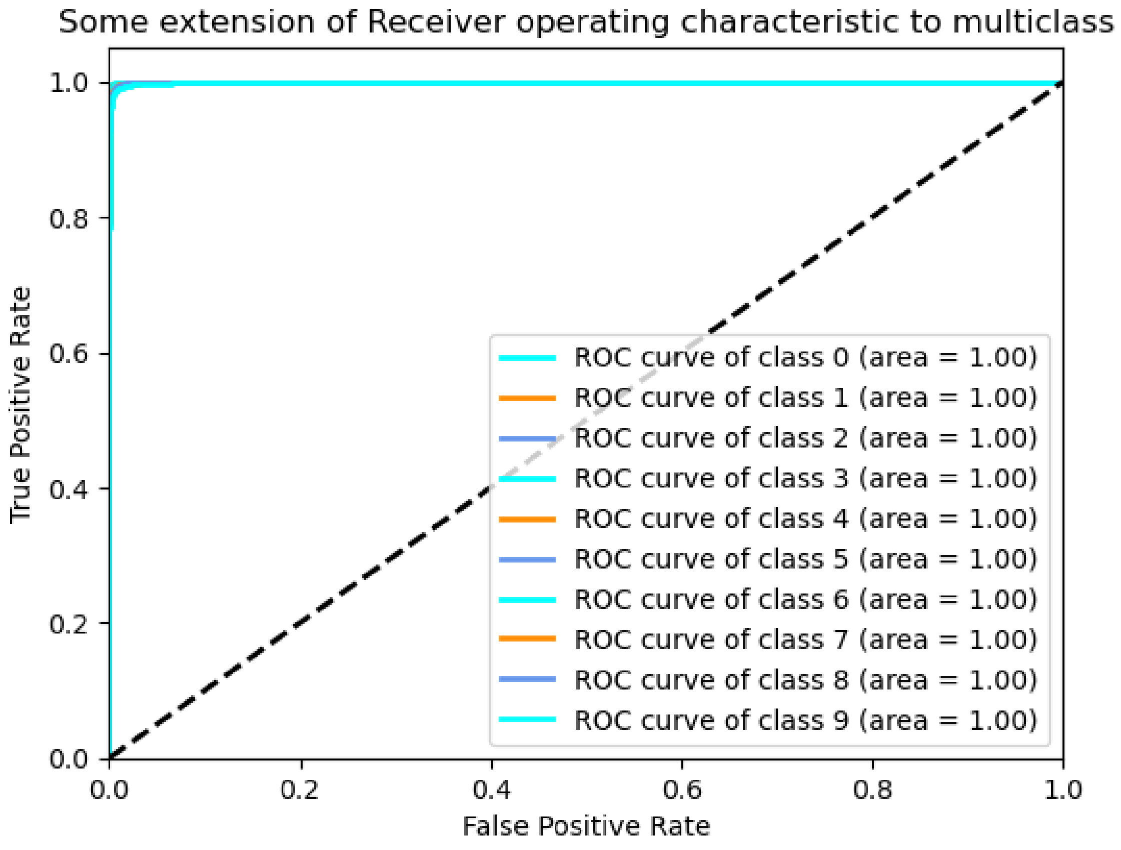
### 6.1.2 SVC Model Evaluation Bagging

```
Accuracy: 0.9803
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1381
           1       0.99      0.99      0.99      1575
           2       0.98      0.98      0.98      1398
           3       0.98      0.98      0.98      1428
           4       0.98      0.98      0.98      1365
           5       0.98      0.97      0.98      1263
           6       0.98      0.98      0.98      1375
           7       0.98      0.98      0.98      1459
           8       0.97      0.98      0.97      1365
           9       0.97      0.97      0.97      1391

    accuracy                           0.98     14000
   macro avg       0.98      0.98      0.98     14000
weighted avg       0.98      0.98      0.98     14000
```

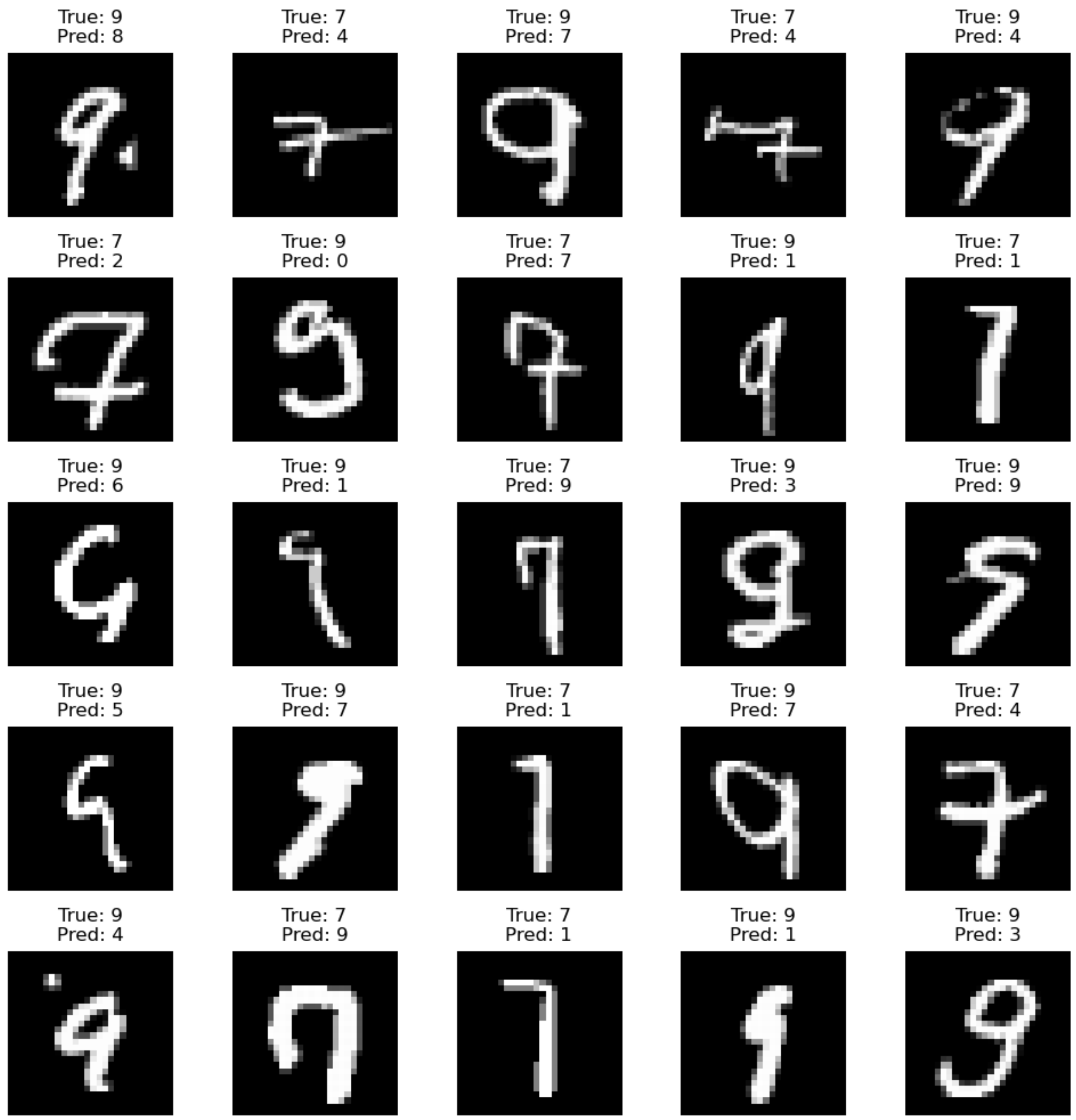
### 6.1.3 Confusion matrix:

```
[[1374    1    0    0    0    2    1    0    2    1]
 [   0 1559    6    2    1    1    0    4    2    0]
 [   4    0 1374    2    3    0    3    6    5    1]
 [   1    1    3 1396    0    9    0    9    7    2]
 [   2    4    2    0 1334    0    5    1    0   17]
 [   3    0    0    9    1 1227    9    0    8    6]
 [   9    1    2    0    2    3 1353    0    5    0]
 [   2    4    8    0    8    0    0 1427    1    9]
 [   3    2    5    5    0   10    3    1 1332    4]
 [   3    5    0    7   10    1    1    9    7 1348]]
```

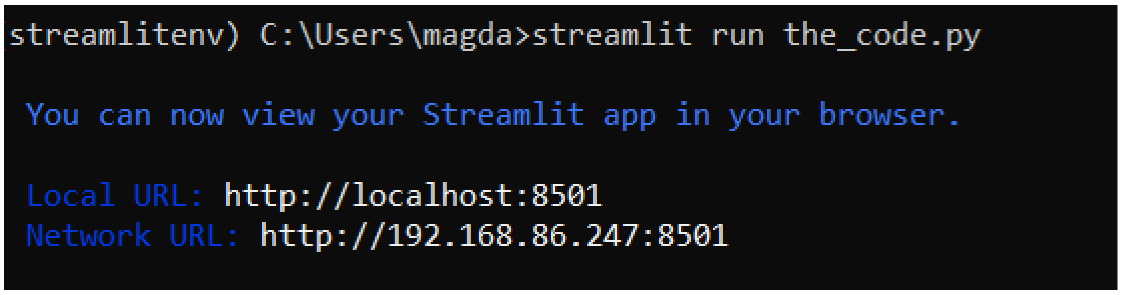
### 6.1.4 ROC Curve



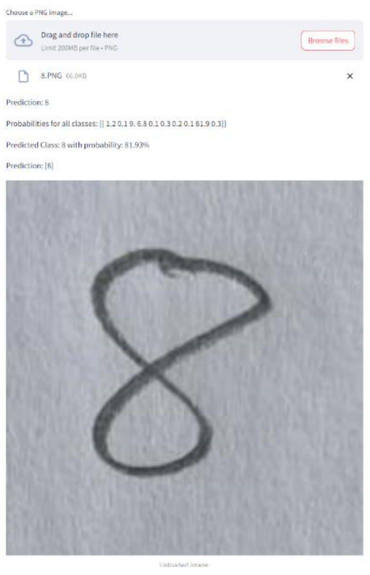
### 6.1.5 Incorrect predictions



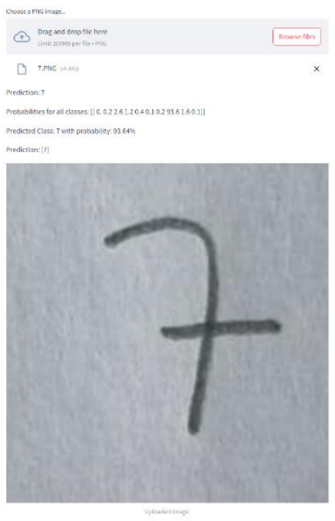
### 6.1.6 Predicted photos in streamlit app



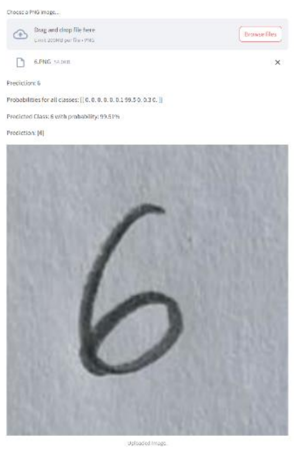
```
streamlitenv) C:\Users\magda>streamlit run the_code.py

 You can now view your Streamlit app in your browser.

 Local URL: http://localhost:8501
 Network URL: http://192.168.86.247:8501
```



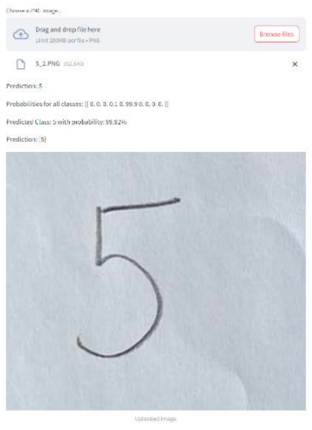
```
Choose a PNG image...
Drag and drop file here
Limit 200MB per file • PNG
8.PNG 66.0KB
Prediction: 8
Probabilities for all classes: [[ 1.2 0.1 9. 6.8 0.1 0.3 0.2 0.1 81.9 0.3]]
Predicted Class: 8 with probability: 81.93%
Prediction: [8]
Uploaded image.
```



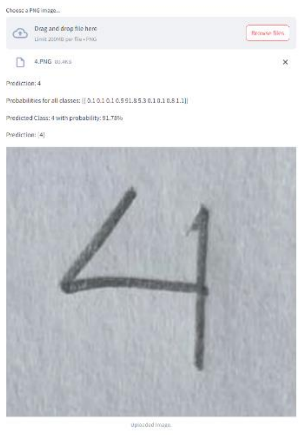
```
Choose a PNG image...
Drag and drop file here
Limit 200MB per file • PNG
7.PNG 54.4KB
Prediction: 7
Probabilities for all classes: [[ 0. 0.2 2.6 1.2 0.4 0.1 0.2 93.6 1.6 0.1]]
Predicted Class: 7 with probability: 93.64%
Prediction: [7]
Uploaded image.
```



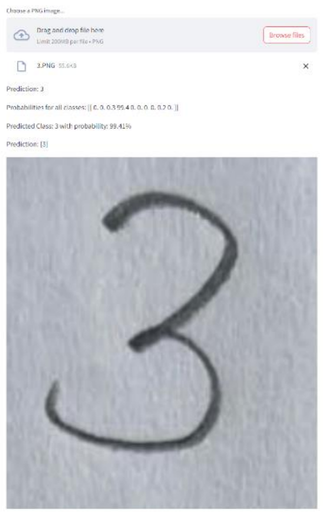
```
Choose a PNG image...
Drag and drop file here
Limit 200MB per file • PNG
6.PNG 54.5KB
Prediction: 6
Probabilities for all classes: [[ 0. 0. 0. 0. 0. 0.1 99.5 0. 0.3 0. ]]
Predicted Class: 6 with probability: 99.51%
Prediction: [6]
Uploaded image.
```



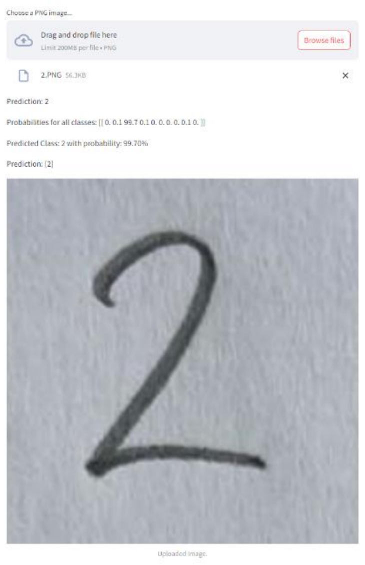
```
Choose a PNG image...
Drag and drop file here
Limit 200MB per file • PNG
5_2.PNG 151.8KB
Prediction: 5
Probabilities for all classes: [[ 0. 0. 0. 0.1 0. 99.9 0. 0. 0. 0. ]]
Predicted Class: 5 with probability: 99.92%
Prediction: [5]
Uploaded image.
```



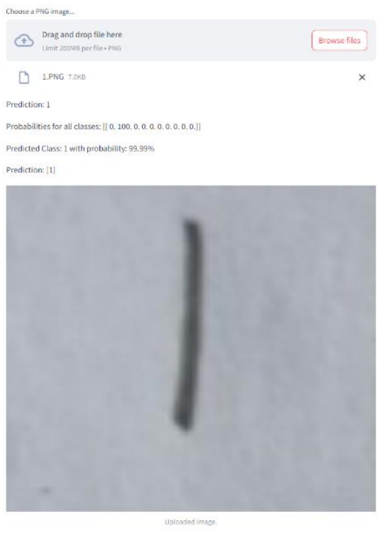
```
Choose a PNG image...
Drag and drop file here
Limit 200MB per file • PNG
4.PNG 83.4KB
Prediction: 4
Probabilities for all classes: [[ 0.1 0.1 0.1 0.5 91.8 5.3 0.1 0.1 0.8 1.1]]
Predicted Class: 4 with probability: 91.78%
Prediction: [4]
Uploaded image.
```



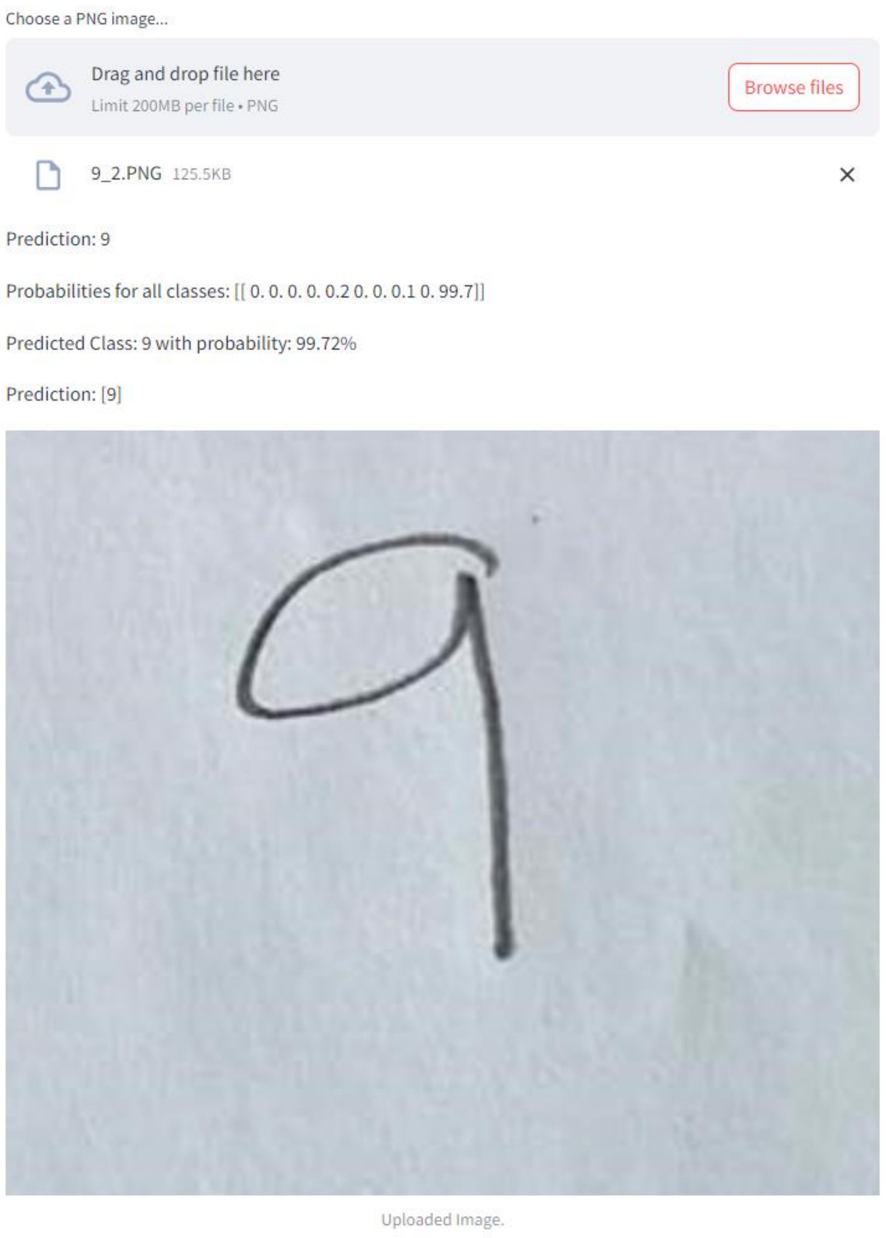
```
Choose a PNG image...
Drag and drop file here
Limit 200MB per file • PNG
3.PNG 55.6KB
Prediction: 3
Probabilities for all classes: [[ 0. 0. 0.3 99.4 0. 0. 0. 0. 0.2 0. ]]
Predicted Class: 3 with probability: 99.41%
Prediction: [3]
```



```
Prediction: 2

Probabilities for all classes: [[ 0. 0.1 99.7 0.1 0. 0. 0. 0. 0.1 0. ]]

Predicted Class: 2 with probability: 99.70%

Prediction: [2]
```



```
Prediction: 1

Probabilities for all classes: [[ 0. 100. 0. 0. 0. 0. 0. 0. 0. 0.]]

Predicted Class: 1 with probability: 99.99%

Prediction: [1]
```



Prediction: 9

Probabilities for all classes: [[ 0. 0. 0. 0. 0.2 0. 0. 0.1 0. 99.7]]

Predicted Class: 9 with probability: 99.72%

Prediction: [9]

Uploaded Image.

## Referenses

Support Vector Machines. (n.d.). In scikit-learn documentation. Retrieved from https://scikit-learn.org/stable/modules/svm.html