Digit Prediction Model

Identifying the Optimal Model for Handwritten Digits Recognition Trained on MNIST Dataset

Shriya Walia

EC Utbildning

Kunskapskontroll 2-ML

2024-03

Abstract

The primary objective of this thesis is to accurately identify handwritten digits and spot any challenges or issues with the classification process. This entails evaluating various machine learning models trained on the MNIST dataset and deploying the most effective model to predict handwritten digits from real-life images. To achieve this goal, a dedicated Streamlit application has been developed to facilitate the deployment and assessment of these models.

Content

Introduction

Imagine navigating life like dining at a restaurant. You peruse the menu, select your desired dishes, and are promptly served delicious meals ready to enjoy. In a world increasingly shaped by AI, our experiences often mirror this dining scenario: we encounter life's challenges, order assistance from AI tools, and are promptly presented with solutions tailored to our needs. Much like the food served at a restaurant, AI delivers services that are readily available and effortlessly usable.

Classification problems are encountered in our daily lives, often solved seamlessly by AI without our awareness. Consider instances like email spam detection, where algorithms sift through our inboxes, flagging unwanted messages. In healthcare, AI aids in medical diagnosis, predicting diseases based on symptoms and patient data. Banks rely on fraud detection systems to safeguard transactions, while image recognition technology enhances security and convenience in various applications. These examples show how AI quietly tackles classification tasks across different areas, profoundly affecting our daily lives.

The aim of this thesis is to evaluate the effectiveness of classification machine learning models using the MNIST dataset and apply them to real-life images. This study will address the following questions:

1. Which machine learning algorithms perform best for digit classification on the MNIST dataset?
2. What is the impact of training dataset size on model accuracy?
3. Can the predictor accurately classify digits (0-9) from real-life images?
4. What proportion of the 10 digits are correctly classified by the models?
5. What challenges may arise when predicting images of one's own?

Outline of the Thesis

The thesis commences with an introduction outlining the project's scope, followed by explanation of theoretical concepts essential for understanding the undertaken project. Subsequently, the methods employed, and the results obtained are discussed. A conclusive summary is provided, highlighting the insights received from the project, suggestions for enhancing performance, and a discussion of the challenges encountered.

1 Theory

Here we will go through the concepts used in the training of a classification machine learning model trained on the MNIST dataset.

Supervised Learning

In this thesis, we focus on supervised learning, one of the two primary branches of machine learning. Supervised learning is defined by its use of labeled data sets to train algorithms that classify data or predict outcomes accurately (IBM, n.d.). Much like a student receives guidance from a teacher in school, supervised machine learning models are guided by labeled datasets throughout their learning process. This approach ensures that the models learn from known examples, allowing them to make informed predictions or classifications when presented with new, unlabeled data. The following supervised learning models are used in this project:

1.1 Random Forest Classifier

Random forest classifier is a commonly used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result (IBM n.d.). As shown in figure 1, each tree is trained on a subset of data and features. During prediction, outputs are aggregated, typically by voting or averaging. This approach enhances robustness and generalization compared to individual trees.

1.1.1 Hyperparameter Tuning

The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as we might turn the knobs of an AM radio to get a clear signal (Koehrsen, 2018). The hyperparameters used for random forest classifier in this project include: ‘n_estimators’ = number of trees in the forest (higher value leading to better performance) & ‘max_depth’ = max number of levels in each decision tree (helps capture pattern intricacies in the data).

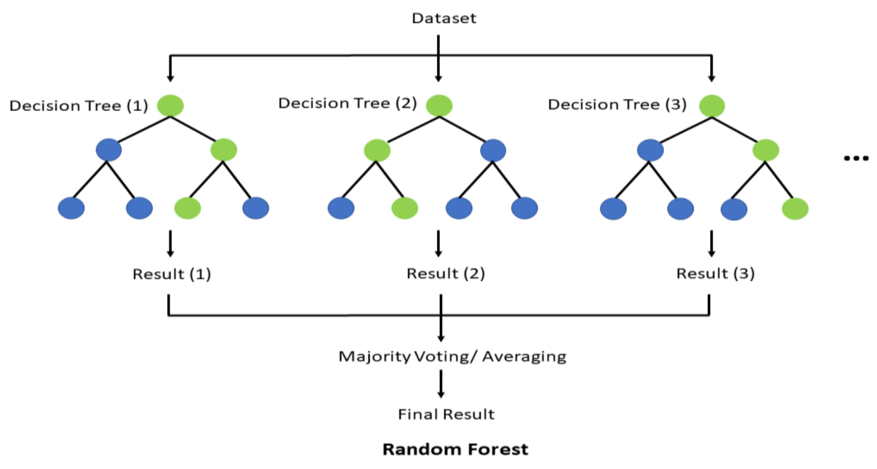


Figure 1: Working of a Random Forest Classifier

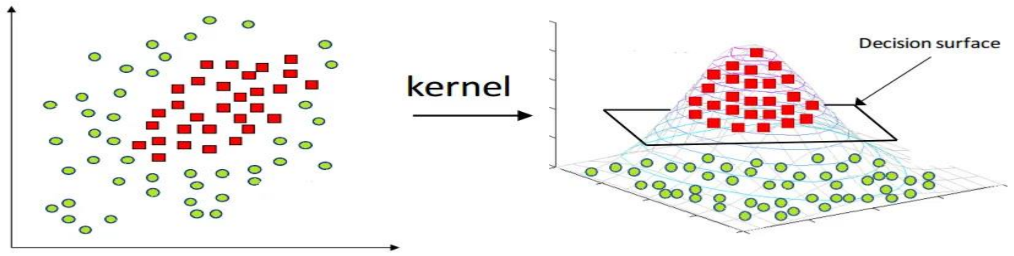


Figure 2: Functioning of non-linear support vector machine with kernel trick

1.2 Support Vector Classifier (SVC)

Support vector machines are used in both regression and classification models and can be divided into linear and non-linear support vector machines. This project uses non-linear support vector classifier (SVC) since the MNIST data cannot be separated by a linear decision boundary, so kernel functions (kernel tricks) are used to transform the non-linear spaces into linear spaces, basically transforming the data into a different dimension (Figure2).

1.2.1 Hyperparameter Tuning

To optimize support vector classifier's (SVC) performance, key hyperparameters like kernel, gamma, and C are utilized. Kernels such as 'Polynomial' and 'RBF' transform low-dimensional input spaces into higher dimensions. 'C' is used as an error control, where a higher C leads to a higher error and vice versa. It is a penalty which tells the model about how much error is bearable. 'Gamma' helps regulate the influences of the points near the line of separation. Higher gamma leads to influence on nearby points and lower gamma influences even the faraway points. Tuning these parameters is vital for achieving robust SVM performance.

1.3 K-Nearest Neighbour (KNN)

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. "Birds of a feather flock together." Harisson, O. (2018). KNN uses the distance to make classification predictions about the group the datapoint belongs to.

1.3.1 Hyperparameter Tuning

The hyperparameter 'n_neighbors' in KNN denotes the number of neighbors or data points considered in closest proximity for classification. Additionally, other parameters such as distance metrics, weights, and algorithm types are available but not utilized in this particular project.

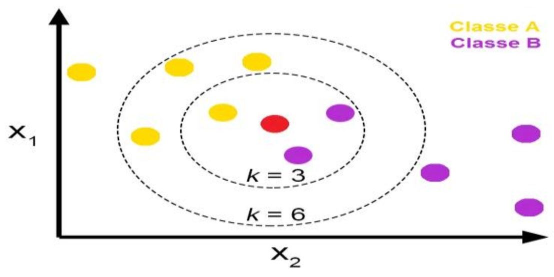


Figure 3: Functioning of KNN Classifier

1.4 Evaluation Metrics

After training the model, it's crucial to evaluate its performance. Common metrics for classification models include accuracy, F1 score, precision, recall, and confusion matrix. In this project, we focus on assessing performance using accuracy and confusion matrix.

1.4.1 Accuracy

Accuracy calculates the number of correctly classified instances over the total number of instances in the dataset. In this project 'accuracy_score' metric on Python calculates the number of correctly predicted labels in the MNIST dataset over the total number of images. The 'accuracy_score' is calculated with the given formula and uses the following terms:

$$\boldsymbol{Accuracy = \frac{TP + TN}{TP + TN + FP + FN}}$$

- True Positives (TP): The number of correctly predicted digits.
- True Negatives (TN): The number of correctly predicted non-digits.
- False Positives (FP): The number of incorrectly predicted digits.
- False Negatives (FN): The number of incorrectly predicted non-digits.

This metric provides insight into the overall correctness of the model's predictions, answering the question: "How often does the model make right predictions?".

1.4.2 Confusion Matrix

A confusion matrix provides a comprehensive overview of a model's accuracy by displaying the counts of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) predictions. It offers a detailed analysis of how well the model performs, showcasing both correct classifications and errors. This visual tool enables easy comparison between different models, allowing for a deeper understanding of their strengths and weaknesses.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

2 Method

The method involves assessing various models and their hyperparameters to select the best-performing model. This chosen model is then trained on the entire dataset and saved for deployment, enabling predictions on real-life uploaded images.

2.1 Tools

The project was initially developed in Python using Jupyter Notebooks, which allowed the use of a range of essential libraries such as Sci-kit learn, Numpy, and Matplotlib. Subsequently, to create the Streamlit application, the code was moved to VS Code for further development and deployment.

2.2 Data

The MNIST dataset is fetched using the '**fetch_openml'** function from the '**sklearn.datasets'** module. This function retrieves the dataset from the OpenML platform, a collaborative platform for machine learning, where it is available as the 'mnist_784' dataset. It comprises 70,000 handwritten digit images, each of which is grayscale and has a fixed size of 28x28 pixels.

2.3 Processing of data

In the initial stages of the project, a portion of the data was set aside for model selection to ensure unbiased evaluation using the Python 'train_test_split' function. The dataset was divided into three parts: training, validation, and test sets. This division enables the assessment of various models' performances before exposing them to the entire dataset. Specifically, the data was split into 10,000 samples for training, 2,000 samples for validation, and another 2,000 samples for testing purposes. The 'Standard Scaler()' class from 'scikit-learn' library (sklearn) in Python was applied to normalize the dataset values of X (features) using which the predictions are supposed to be made, enhancing model performance by ensuring uniformity in the feature scales across the dataset.

2.4 Model Evaluation

In this project, three machine learning models were evaluated: Random Forest Classifier, Support Vector Classifier, and K-Nearest Neighbors. Each model underwent the same evaluation process, starting with a 'GridSearchCV' class from 'scikit-learn' library (sklearn) in Python, on the hyperparameters. This method helped identify the optimal combination of hyperparameters by conducting cross-validation on the training set. Once the best hyperparameters were determined, they were applied to the training set using the '.fit()' method to train the model. The trained model was then saved using the 'Joblib' library. Subsequently, predictions were made on the validation set (unseen data) using the '.predict()' method, and accuracy was assessed using the 'accuracy_score' metric. A confusion matrix was also generated to evaluate the model's performance. Following these steps, the model's performance was evaluated on the test data (unseen data). The model with the highest accuracy on the test data was selected as the best-performing model to proceed with further analysis.

2.5 Training on whole dataset

After selecting the best model, it was fitted to the entire dataset. Initially, the dataset was divided into features (X) and labels (y), after which the features (X) were normalized using 'Standard Scaler()' to enhance performance. Subsequently, the previously saved model 'svc_final.pkl' was retrained on the entire dataset using the '.fit()' method, and finally saved 'svc_final_wholedataset.pkl' for making predictions on real-life images.

2.6 Preprocess the image data

In order to feed real images into the trained model, the images had to be converted into the same format as the MNIST images that the model was trained on. To achieve that 2 functions were created, one to preprocess the uploaded image for predictions and the second to make predictions on the preprocessed image.

- **Preprocess the uploaded image:** The preprocessing function first converts the uploaded image to grayscale with the OpenCV library (cv2). Subsequently, the colors of the image are inverted to ensure consistency, transforming the image from a black digit on a white background (which would commonly be used in real life) to a white digit on a black background. The image is then resized to 28x28 pixels to match the dimensions of the MNIST dataset. Next, a Gaussian blur is applied to the image to reduce noise and smooth out any irregularities, flattened from a 2D array to a 1D array and normalized to ensure uniformity with the dataset.
- **Predicting the preprocessed image:** The prediction function reads the path to the uploaded image, uses the trained model ''svc_final_wholedataset.pkl' to make predictions on the preprocessed image.

Combining these two functions allows for seamless prediction on an uploaded image. By providing the image path as input, the functions process the image, predict its label, and display both the image and the predicted label as the end product.

2.7 App Development

The app is developed using Python code within the VS Code platform. Utilizing the same preprocessing techniques used for image data, an interactive interface is created for users. Upon deployment, users can upload images of handwritten digits (0-9) on paper with a white background, in formats such as jpg, jpeg, or png. Upon uploading, users can click the predict button to obtain the predicted value.

3 Result

The result of the project includes the identification of the best performing model and the successful deployment of the Streamlit app.

3.1 Selecting the best performing model

The selection of the best performing model was done by checking the performance of all the 3 models Random Forest Classifier, Support Vector Classifier and K-Nearest Neighbour, on both validation and test set using accuracy metric and confusion matrices. The results are as follows:

- Best performing model on validation set: Support Vector Classifier with 96.35% accuracy
- Best performing model on test set: Support Vector Classifier with 95.7% accuracy.
- The confusion matrices of the model performances on test set clearly show Support Vector Classifier as better performing than Random Forest Classifier, with a better performance on the digits 3,4,5,7,8 & 9.

Validation Set (2,000):

Model	Accuracy
Random Forest Classifier	95.05%
Support Vector Classifier	96.35%
K-Nearest Neighbour	91.95%

Test Set (2,000):

Model	Accuracy
Random Forest Classifier	94.8%
Support Vector Classifier	95.7%
K-Nearest Neighbour	91.75%

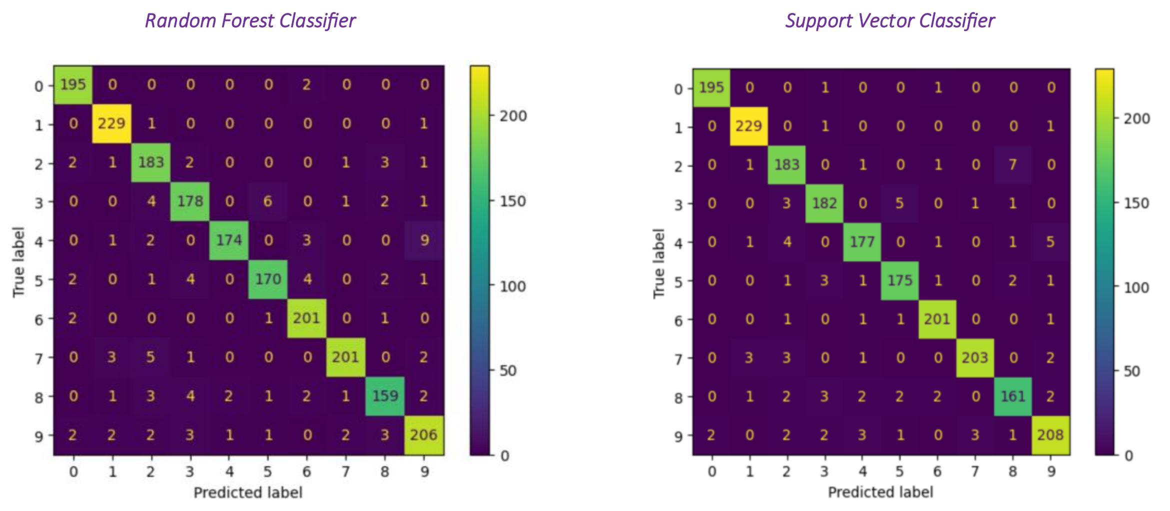
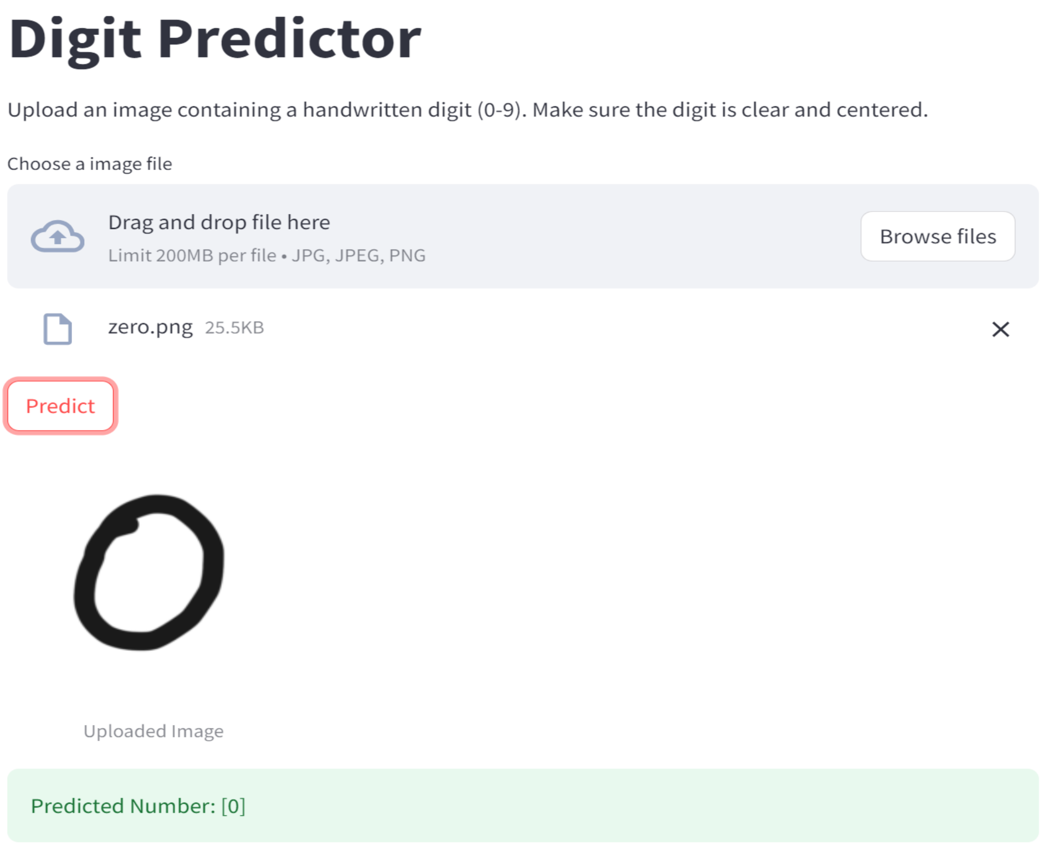


Figure 4: Confusion matrices of the two best performing models on test set

3.2 Model performance on Streamlit app

The model performs exceptionally well on real-life handwritten images. If one uploads images containing digits with some visible but not excessive empty edges around them, and if the digits are sufficiently bright, the model is able to predict accurately. In multiple tests, the app successfully predicted all digits correctly when uploaded appropriately. Below are the results from the app:



Digit Predictor

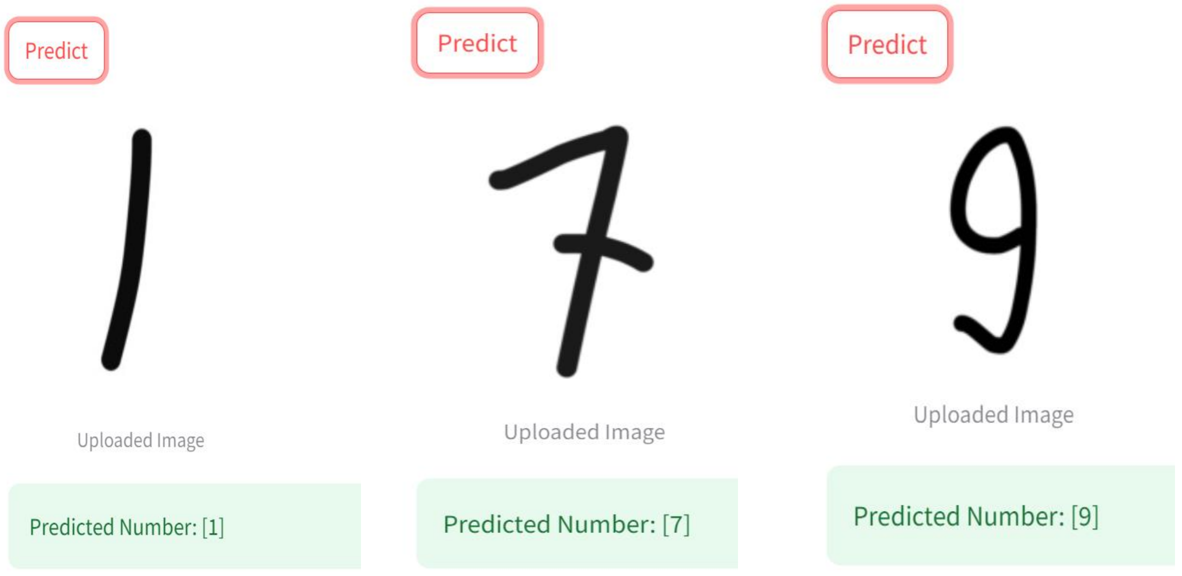
Upload an image containing a handwritten digit (0-9). Make sure the digit is clear and centered.

Choose a image file

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

zero.png 25.5KB

Predicted Number: [0]



Predicted Number: [1]

Predicted Number: [7]

Predicted Number: [9]

Figure 5: Deployment of Streamlit app

4 Conclusion

The conclusion of this thesis answers the questions asked in the beginning:

4.1 Best performing model for digit classification

In this project, Support Vector Classification model demonstrated effective performance. However, exploring additional models could potentially yield superior results. As this project is being conducted by a beginner in this field, a deliberate choice was made to commence with widely recognized models. This decision was driven by the desire to establish a solid foundation in this field and progressively venture into more complex models.

4.2 Training size

Due to time constraints, the model's training size in this project was limited. However, with a larger training size, improved results could be achieved.

4.3 GridSearchCV

Due to time limitations, only a limited number of hyperparameter values were explored in the hyperparameter tuning process. Despite this constraint, GridSearchCV proved valuable in identifying optimal hyperparameters for the model.

4.4 StandardScaler()

In this project, it was found that employing a StandardScaler() class is crucial for optimizing the performance of Support Vector Classifiers. StandardScaler() was consistently applied at various stages, including on the training data, the entire dataset, and during preprocessing of uploaded images. This consistent application ensures uniformity and enhances the effectiveness of the model.

4.5 Accuracy of the digit predictor

In order for the model to predict correctly on real life images, they had to be converted into the same format as the MNIST dataset by using a preprocessing function on the image. After trying out the predictions on real life images it has been found that:

- When the uploaded image features bold digits (0-9) and is well-cropped into a square with visible edges, along with a clear background where the digit's brightness stands out, the predictions are consistently accurate, achieving a 10/10 success rate.
- If the uploaded image contains non-bold digits, has a rough background, is improperly cropped or centered, the predictions tend to be inaccurate.

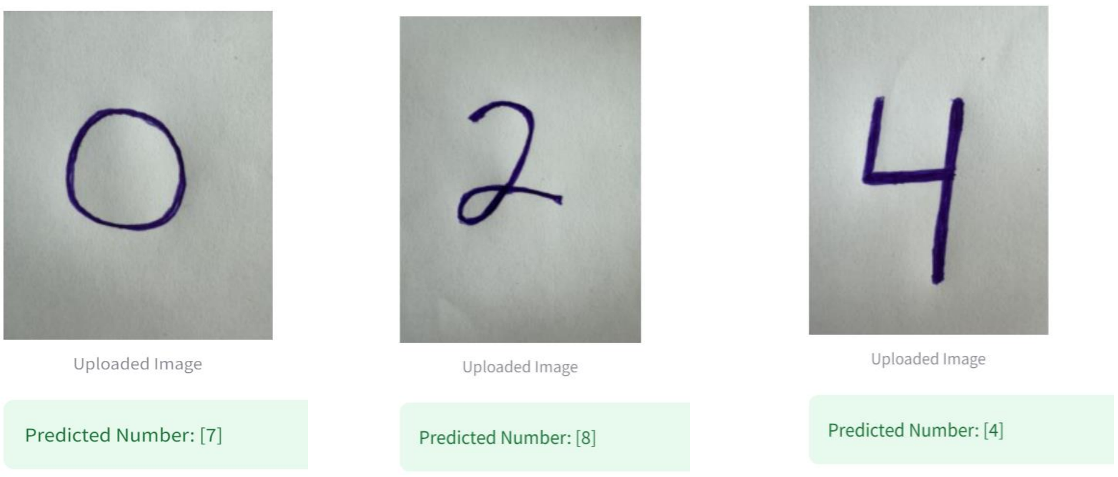


Figure 5: Predictions of non-edited uploaded images

In conclusion, this project demonstrates potential for further exploration. While the current results show accuracy with well-defined, properly cropped images, there remains room for improvement. Achieving accurate predictions for any type of uploaded image is feasible with additional refinements. This can be accomplished through techniques such as data augmentation to diversify the training set, implementing advanced preprocessing methods to enhance image quality, and experimenting with ensemble learning to leverage the strengths of multiple models. By continuing to refine the model and training procedures, the goal of accurately predicting any type of uploaded image can be achieved.

Teoretiska frågor

1. Kalle delar upp sin data i ”Träning”, ”Validering” och ”Test”, vad används respektive del för?

Svar: Alla tre uppdelningarna av data har olika syften. ‘Träningssetet’ används för att träna modellen, ‘valideringssetet’ används för att ställa in hyperparametrar och utvärdera prestanda under träning, och ‘testsetet’ används för att bedöma den tränade modellens slutliga prestanda.

2. Julia delar upp sin data i träning och test. På träningsdatan så tränar hon tre modeller; ”Linjär Regression”,”Lasso regression” och en ”Random Forest modell”. Hur skall hon välja vilken av de tre modellerna hon skall fortsätta använda när hon inte skapat ett explicit ”validerings dataset”?

Svar: Om Julia inte har skapat en separat ‘validerings dataset’ kan hon använda ‘cross validation’ för att utvärdera prestandan för varje modell. ‘Cross validation’ innebär att dela upp data i önskat antal ‘folds’ och ‘score’ dem med hjälp av ett mått som ‘negative mean squared error’.

3. Vad är ”regressionsproblem”? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden?

Svar: Ett ‘regressionsproblem’ innebär att prediktera kontinuerliga värden baserat på ‘features’ eller X värde. Regressionsproblem kan vara av olika slag: linjär regression, multipel linjär regression, polynomregression, osv. Exempel på regressionsmodeller inkluderar linjär regression, lasso regression, ridge regression, osv. Potentiella tillämpningar inkluderar att prediktera huspriser, aktiekurser, medicinsk diagnos, osv.

4. Hur kan du tolka RMSE och vad används förmeln till?

Svar: ‘RMSE’ står för ‘Root Mean Squared Error’ och är ett mått för att utvärdera prestandan hos regressionsmodeller. Den mäter den genomsnittliga storleken på skillnaderna mellan de faktiska värdena och de predikterade värdena. En lägre RMSE indikerar bättre prestanda, eftersom det betyder mindre avvikelser mellan de predikterade och faktiska värdena.

Förmeln används för att ge oss ett enda värde som sammanfattar den övergripande ‘accuracy’ av regressionsmodellens prediktioner. Vi överväger inte om skillnaden mellan det faktiska värdet och det predikterade värdet är negativt eller positivt, så vi beräknar kvadratfelet som $(yi - \hat{y}i)$^2. Sedan beräknar vi medelvärdet av dessa kvadratiska skillnader och tar kvadratroten av medelvärdet för att återgå till målvariabelns ursprungliga skala.

5. Vad är ”klassificieringsproblem”? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden? Vad är en ”Confusion Matrix”?

Svar: Ett ‘klassificeringsproblem’ innebär att prediktera en kategorisk ‘label’ eller klass baserat på ‘input features’. Klassificeringsproblem kan vara av olika slag: binär klassificering, multiklassklassificering, multi-label-klassificering, osv. Exempel på klassificeringsmodeller inkluderar logistic regression, beslutsträd, random forest, Support Vector Machines, K-Nearest Neighbour,osv. Potentiella tillämpningar inkluderar: e-postspamdetektering, bildigenkänning, sjukdomsdetektering, osv.

En ‘Confusion Matrix’ är en tabell som ofta används för att utvärdera prestandan hos en klassificeringsmodell. Den visar antalet korrekta prediktionerna (sanna positiva och sanna negativa) och de felaktiga prediktionerna (falska positiva och falska negativa). Från “Confusion Matrix” kan

olika prestandamått såsom accuracy, precision, recall och F1 score beräknas för att bedöma modellens prestanda.

6. Vad är K-means modellen för något? Ge ett exempel på vad det kan tillämpas på?

Svar: ‘K-means modellen’ är en ‘unsupervised’ maskininlärningsalgoritm som används för att gruppera data i grupper eller kluster. Den syftar till att dela upp data i K-kluster, där varje datapunkt tillhör klustret med närmaste medelvärde. K-means kan tillämpas till: kundsegmentering, bedrägeriupptäckt, identifiering av cyberbrott, optimering av leveransväg, osv.

7. Förklara (gärna med ett exempel): Ordinal encoding, one-hot encoding, dummy variable encoding.

Svar: ‘Ordinal encoding’ är en teknik för att omvandla kategoriska egenskaper till ett numeriskt format. Vid ordinal encoding översätts etiketter till siffror baserat på deras ordningsförhållande till varandra. Exempel: Tänk på kryddighetsnivån med kategorierna: ["Mild", "Medium", "Hot"]. Vi kan mappa dessa kategorier till heltal som {"Mild": 1, "Medium": 2, "Hot": 3}.

Med ‘One-hot encoding’ konverterar vi varje kategoriskt värde till en ny kategorisk kolumn och tilldelar ett binärt värde på 1 eller 0 till dessa kolumner. Varje heltalsvärde representeras som en binär vektor. Exempel: "Kryddighetsnivån" med kategorierna ["Mild", "Medium", "Hot"]. Efter one-hot encoding kan "Mild" representeras som [1, 0, 0], "Medium" som [0, 1, 0] och "Hot" som [0, 0, 1].

‘Dummy variable encoding’ liknar one-hot encoding men innebär att skapa en dummy-variabel mindre än antalet kategorier. I denna kodning används en kategori som referenskategori, och närvaron eller frånvaron av andra kategorier representeras av binära variabler. Exempel: Vi skapar tre binära variabler för att representera anställningsstatus, en för varje kategori utom referenskategorin. Låt oss skapa variablerna "IsUnEmployed", "IsSelfEmployed" och "IsRetired". Varje variabel sätts till 1 om individen har motsvarande anställningsstatus och 0 i övrigt. För individer som är "IsEmployed" sätts alla dummyvariabler till 0, vilket indikerar att de inte är 'Unemployed’, ‘Self-Employed’ eller ‘Retired’.

8. Göran påstår att datan antingen är ”ordinal” eller ”nominal”. Julia säger att detta måste tolkas. Hon ger ett exempel med att färger såsom {röd, grön, blå} generellt sett inte har någon inbördes ordning (nominal) men om du har en röd skjorta så är du vackrast på festen (ordinal) – vem har rätt?

Svar: Både Göran och Julia har giltiga poäng. Görans påstående beskriver på ett korrekt sätt de allmänna definitionerna av ‘nominal’ och ‘ordinal’ data: ‘nominal’ data saknar inneboende ordning eller rangordning, medan ‘ordinal’ data har en meningsfull ordning eller rangordning. Julias perspektiv introducerar nyanser genom att antyda att även om färgerna i sig vanligtvis anses vara ‘nominal’ på grund av sin brist på inneboende ordning, kan de potentiellt ses som ordinala när man överväger ytterligare faktorer som attraktionskraft. I detta sammanhang föreslår Julia att färger skulle kunna rangordnas på ett ‘ordinal’ sätt baserat på deras upplevda attraktionskraft vid ett evenemang. Därför är Görans påstående om de generella definitionerna tekniskt korrekt medan Julias syn på sammanhanget ger ytterligare insikt, det ändrar inte den fundamentala definitionen av nominal och ordinal data.

9. Vad är Streamlit för något och vad kan det användas till?

Svar: ‘Streamlit’ är ett Python-bibliotek som är ‘open-source’ och gör det enkelt att skapa och dela anpassade webbappar för maskininlärning och datavetenskap.

Självutvärdering

1. Utmaningar du haft under arbetet samt hur du hanterat dem:

Utmaningar som jag mötte när jag gjorde det här projektet var när jag 'preprocessed' bilderna. Det tog ett tag att få korrekta prediktioner på de uppladdade bilderna. Jag var tvungen att redigera de uppladdade bilderna mycket för att kunna få korrekta prediktioner. Det tog också lite tid att skapa appen, eftersom det tog ett tag att förstå kopplingarna och vad som skulle vara var för att starta appen. Efter många felmeddelanden lärde jag mig vad som var fel och kunde fixa det.

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

Jag ser mig själv få VG eftersom jag har visat ordentligt arbetsflöde med ordentliga steg i att bygga projektet. Jag uppfyllde också VG-kraven för att bygga en app och ladda upp bilder för att få prediktioner.

3. Något du vill lyfta fram till Antonio?

Tacksam för att ha en så bra lärare som gjorde maskininlärning så kul att lära sig. Allt var supertydligt, och utmaningarna fick mig att lära mig ännu bättre. Frågorna hjälpte till att revidera alla ämnen en gång till för att göra dem fasta i hjärnan. Tack för allt.

References

- IBM. (n.d.). What is Supervised Learning. Retrieved from https://www.ibm.com/topics/supervised-learning
- IBM. (n.d.). What is Random Forest. Retrieved from https://www.ibm.com/topics/random-forest
- Figure 1. Random forest explain. Retrieved from https://en.m.wikipedia.org/wiki/File:Random_forest_explain.png
- Koehrsen, W. (2018). Hyperparameter Tuning the Random Forest in Python. Toward Data Science. Retrieved from https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
- Figure 2. Retrieved from https://medium.com/analytics-vidhya/how-to-classify-non-linear-data-to-linear-data-bb2df1a6b781
- Harisson, O. (2018). Machine Learning Basics with the K-Nearest Neighbors Algorithm. Toward Data Science. Retrieved from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- Figure 3. Retrieved from https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d