Digit Recognition

From a Live-Feed



Robert Westman

EC Utbildning

Kunskapskontroll 2 – Machine Learning

2024-03

# Abstract

This report tracks a beginners venture into machine learning, including choosing, training and evaluating models, image processing and achieving results. A Streamlit application is then set up for ease of use.

While the basic machine learning models can work alright for digit recognition, there are obvious flaws that forces plenty of requirements and the room for improvement is great. I have purposefully omitted moving on to more advanced models and techniques in favour of learning and optimizing the basic principles.

# Abbreviations and terms

OCR – Optical Character Recognition

EDA – Exploratory Data Analysis

SVM – Support Vector Machine

KNN – K-Nearest Neighbor

RBF – Radial Basis Function

OpenCV – Python package for image processing

MNIST – Modified National Institute of Standards and Technology

Array – Multi dimensional table of information

Matrix – 2-dimensional table of information

TLDR – Too long didn’t read

Content

[Abstract 2](#_Toc161828770)

[Abbreviations and terms 3](#_Toc161828771)

[1 Intro 1](#_Toc161828772)

[1.1 General problems 1](#_Toc161828773)

[1.1.1 Questions 1](#_Toc161828774)

[2 Theory 2](#_Toc161828775)

[2.1 Classification models 2](#_Toc161828776)

[2.1.1 Random Forest 2](#_Toc161828777)

[2.1.2 Support Vector Machine 2](#_Toc161828778)

[2.1.3 Voting Classifier 2](#_Toc161828779)

[2.1.4 K-Nearest Neighbor 2](#_Toc161828780)

[3 Method 3](#_Toc161828781)

[3.1.1 Dataset and EDA 3](#_Toc161828782)

[3.1.2 Model selection and evaluation 3](#_Toc161828783)

[3.1.3 Transition to custom images 3](#_Toc161828784)

[3.1.4 Streamlit-application 3](#_Toc161828785)

[4 Results and Discussion 4](#_Toc161828786)

[5 Conclusions 6](#_Toc161828787)

[5.1 TLDR 6](#_Toc161828788)

[6 Teoretiska frågor 7](#_Toc161828789)

[7 Självutvärdering 9](#_Toc161828790)

[Appendix A 10](#_Toc161828791)

[Källförteckning 11](#_Toc161828792)

# Intro

Digit recognition has been a relevant issue dating back to the 1800’s, however machines as we know them today have only been able to perform OCR since the 1970’s. The purpose of these early iterations was to sort mail or recognize handwritten numbers, very much like the purpose of our project today. (”What is OCR”, Amazon Web Services)

One can quickly see the benefit of having such a system in place at almost any place of work, the amount of automatization that can potentially be achieved is great, especially when looking even further, beyond digit recognition alone.

Thus we arrive at our project; creating a program that can take a snapshot from a live-feed and accurately predict which number is written on a piece of paper. To achieve this there are a few questions that need to be raised and answered.

## A quick word

This report assumes the reader is familiar with basic machine learning terms. Throughout this report I will frequently reference facts about machine learning and models that stem from the book ”Hands-On Machine Learning with Scikit-learn, Keras and Tensorflow” by Aurélien Géron, which provided a great overview of the subject.

If at any point the reader wants a deeper look into how something was done, some of the code used is available at Appendix A.

## General problems

The first step involved finding a general model that is accurate and adaptable to new incoming data. To be able to do this, I also needed to understand how the model ”views” the images it trains on.

Then I had to find a way to import a users own images, make sure they fit the format and color-scheme of our training data, evaluate the model performance on these images and make potential adjustments.

To complete the workflow our program should be easy for anyone to use, so we need to set up an application to achieve this.

A general problem throughout the project is the fact that handwritten digits can take on almost any style depending on who is writing. Even the type of pen used can matter alot.

### Questions

The following questions will be answered:

* Which model achieves the highest accuracy score?
* How is the model viewing the raw image data?
* Does a high accuracy score on training data translate well to custom images?
* How well do the model work on custom images?
* Can I make this program easy to use via a Streamlit application?

# Theory

## Classification models

Since my objective was to classify an image based on the intensity of pixels present in said image, it made sense to evaluate classification models. They can find the patterns of pixels in an image based on certain criteria such as proximity to other colored pixels or based on where the pixel is located in our training set. (Géron, 2019).

### Random Forest

Random Forests work much like several Decision Trees put together, where each Tree looks at our features (datapoints) and gives its own input, after every Tree has given its input, a majority vote takes place. I used bagging, meaning each tree takes random samples that were taken with replacement.

### Support Vector Machine

SVM’s use datapoints as coordinates and tries to find the hyperplane that best separates the datapoints from eachother, trying to find the hyperplane with maximum margin. The points furthest from a classification (and closest to the hyperplane) is called ”support vectors”

#### SVM Hyperparameter tuning

Using GridSearchCV, the optimal hyperparameters was (kernel=RBF, C=3 and gamma=scale). RBF measures distance in the hyperplane, with points close to eachother receiving higher similarity scores. C controls the penalization for misclassification. ”Scale” notes the reach of influence for each datapoint using this equation: ’1 / (n\_features / X.variance)’.

### Voting Classifier

This model takes several separate models and weigh their individual responses before coming to a decision. In my case I used a classifier consisting of random forest, extra trees and a LinearSVC model.

### K-Nearest Neighbor

This model looks at a given datapoints neighbors to decide which class it belongs to.

#### KNN Hyperparameter tuning

I again used GridSearchCV to find the optimal hyperparameters (n\_neighbors=3, leaf\_size=15). N\_neighbors indicates that the 3 closest datapoints will be asked and leaf\_size indicates that 15 datapoints will be considered a group.

# Method

### Dataset and EDA

The very first aspect in a project such as this is to choose a dataset on which to train our models, in this case I used the MNIST dataset containing 70,000 images of handwritten digits, a crucial point here was to observe the format and colorscheme of the images. The MNIST dataset is available via the SKLearn package in Python.

After performing EDA, I could see that the images are grayscale and represented in the form of arrays containing the numbers 0-255. Higher numbers are indicating a darker tone (from white to black), this is what our model will be training to recognize, the pattern of these values.

### Model selection and evaluation

Next up was model selection, and since my goal was to classify the label (or target-value) of an image, I opted to explore classification models such as Random Forest, SVM, a Voting Classifier and K-Nearest Neighbor.

To evaluate the models accurately without sacrificing too much time, I split the data into a training set consisting of only 5000 images, a validation set of 1000 and a test set of 1000. The models were fitted and trained on the training data, a score was then generated on the validation data and the model with highest accuracy was then ran on the test data to ensure decent generalization.

### Transition to custom images

This ”best” model was then trained on the entire dataset and evaluated on my own images. To import and process images, I opted to use OpenCV for Python, as it had ample documentation online (OpenCV Tutorials, 2024)

Ofcourse, to be able to feed custom images in to the model, said images would have to fit the format of the training images, so any image inserted would have to be resized, turned into grayscale and color-inverted.

Lastly, the program should be able to function for anyone, even those not versed in programming or machine learning, so a third-party application called Streamlit was used to deploy my program. *Figure 1* and *Figure 2* showcases the simple user interface of my application.

### Streamlit-application

Using the documentation (Streamlit documentation), setting up a basic application was quite straightforward, I used the same image processing as before baked into a function.

Setting up a live camera-feed (OpenCV Python-Tutorials) was easy thanks to the well written documentation in OpenCV. I then added a button that captures the image currently visible in the live-fed frame. This image is then fed to the processing function to ensure correct color and shape for model prediction.

En bild som visar skärmbild, design

Automatiskt genererad beskrivningEn bild som visar text, skärmbild, Teckensnitt, nummer

Automatiskt genererad beskrivningThe goal was to create a very simple program that just works. Theoretically, one could spend a long time polishing and making the app shine, but this was not the purpose of the project.

Figure 2: The processed image along with model prediction.

Figure 1: The live-feed as seen in the application.

# Results and Discussion

|  |  |
| --- | --- |
| **Accuracy scores** | |
| Random Forest | 0.956 |
| SVC | 0.9817 |
| Voting Classifier | 0.965 |
| KNN | 0.9687 |

Table 1: Accuracy scores for the models evaluated.

As seen in Table 1, the SVC model achieved the highest score on the training/test data, naturally this was the first model I tried on my own images. However, it did not work at all on my own images, even though extensive image preprocessing was made to ensure the images would be uniform to the training set. Parameters were tweaked, the image input data was changed which helped a bit, but ultimately I couldn’t get this model to generalize well. I came to the conclusion that the image inputs were too noisy for this model and I went on to evaluate the KNN model before investigating further.

This yielded promising results, with the model being able to correctly classify 5/10 digits from the get-go. This led to a rabbit-hole of image preprocessing techniques and a lot of tests began.

En bild som visar nummer, text, skärmbild, Teckensnitt

Automatiskt genererad beskrivningAs seen in *Figure 3,* the raw image isn’t great (for our purpose), it has a lot of gray noise as opposed to the clear-white background that the model was trained on, the dimensions are also wrong.

Figure 3: Unprocessed image.

En bild som visar text, skärmbild, nummer, Teckensnitt

Automatiskt genererad beskrivningTo combat the noise of the background, Gaussian Blur was applied which helped a lot as long as the written digit is thick and clearly outlined. This didn’t generalize well to different digits however, where Adaptive Thresholding often performed better (”Adaptive Thresholding with OpenCV”, 2021). (Figure 4)

Figure 4: Gaussian Blur applied.

En bild som visar skärmbild, svart och vit, kvadrat, svart

Automatiskt genererad beskrivningThe final steps involved resizing the image and color-inverting it. This produces a decent replica of the training images and helped the model a lot. The image is then flattened, reshaped and scaled to fit our model. (Figure 5)

Figure 5: Resized and inverted

After the image processing was applied, the model could predict all 10 digits correctly, however the angle, shadows and size (distance) of the digits still played a big role.

At this point it was becoming clear that these models can indeed work well for identifying a single digit on a clear background, but for more complex identification such as multiple digits at once, mixing digits/letters or discerning where to ”look” in a picture, this technique is simply not sufficient. This greatly limits the usage potential of such a program and is in itself a great motivator for continued learning and research into more advanced techniques.

As for the Streamlit application, I quickly ran into trouble when trying to fetch files from my local system. After extensive troubleshooting, I found no reason why it would throw a permission error, and since time was limited, I changed my approach. Rather than importing a ready image, I set up a live-feed from a webcam that can snap a picture. This involved more steps than the ready image, but could also be used in a wider sense.

En bild som visar handskrift, Teckensnitt, kalligrafi

Automatiskt genererad beskrivningFigure 6: The custom written digits I mainly used for identification.

Looking at the digits in Figure 6 I now clearly see the problem areas, the bottom part of the 6 is not distinguished enough, this applies to the 8 aswell. The 0 has more of a circle shape rather than an oval.

These tiny things matter alot for our model and made it struggle with these specific digits. Look at Figure 7 for an idea of how the optimal digit looks. A solid black digit on a white background, with distinct features.

En bild som visar dimma

Automatiskt genererad beskrivning

Figure 7: Optimal digit.

# Conclusions

As for finding the best model for this task, it was less straight forward than I’d have thought. Even though the SVC model was clearly superior on the testing data, it did not adapt well to real-life images. I attribute this to the data being too noisy for the linear separation that such a model requires to function well. The KNN model proved superior for this task, most likely because it classifies data based on the datapoints surrounding it, making it easier to adapt.

The preprocessing of the image itself is absolutely *vital* for your model to be able to identify the digits, with the correct processing techniques you could turn almost any handwritten digit into something that the model is able to read and comprehend. This is by far where I spent the majority of my time, trying to find a solution that is not *customized* to my images, but actually works on new images aswell. Theoretically, a ***very*** long time could be spent here if one wanted to account for all possible anomalies that could be present in images.

While setting up a Streamlit application didn’t present much difficulty in itself, there is always the chance for unknown errors to pop up, testing ones ability to problem solve. When the application for some unknown reason did not have permission to fetch files from my local system, I changed my approach rather than getting stuck on a problem that I realised might not be easily fixable.

## TLDR

* Q: Which model achieves the highest accuracy score?
* A: The SVC model achieved the highest accuracy score.
* Q: How is the model viewing the raw image data?
* A: The model views the training data as a matrix with 28x28 values in the range 0-255 (since this is the format of the training data). It views our own images as arrays, this is why we need to grayscale our images, the color in our pictures add an axis.
* Q: Does a high accuracy score on training data translate to custom images?
* A: No, most likely some models overfit the general format of the training data.
* Q: How well do the model work on custom images?
* A: It *can* work well, but extensive image processing is required and the risk is ”overfitting” for the images you test with.
* Q: Can I make this program easy to use via a Streamlit application?
* A: Yes, Streamlit made it quite easy with the end-product being a program with a user-friendly graphical interface.

# Teoretiska frågor

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

* Tränings-settet används för att träna modellen, här används den största mängden av datan. Validerings-settet är ett mindre dataset där man utvärderar de olika modeller man testat. Slutligen, när alla hyperparametrar och annat är utvärderat så kör man modellen på test-settet för att få fram ett slutgiltigt resultat.

1. 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 ”valideringsdataset”?

* Man kan använda sig av Cross Validation för att utvärdera modeller på enbart träningsdatan. CV utvärderar valda modeller på slumpmässigt utvalda delar av träningsdatan och spottar ut MSE för varje, på så sätt kan man gå vidare med den modell som ger bäst score.

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

* Regressionsproblem beräknar kontinuerliga värden, för detta kan man använda tillexempel en linjär regressionsmodell, random forest eller Support Vector Machines. Ett konkret exempel kan vara att värdet på ett hus (Y) stiger i en viss takt beroende på husets ”features” (X) där är förändringskoefficienten. Säg att Y stiger med 10000kr för varje X, där X = 1 kvm.

1. Hur kan du tolka RMSE och vad används det till:

* Plugga in och summera Y – värdena, kvadrera för att undvika negativa värden, dra roten ur för att motverka kvadreringen (så vi kan tolka resultatet). Detta beräknas för att få fram ett mått på hur långt bort modellen är från de riktiga värdena, ju lägre resultat desto bättre.

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

* Ett klassificeringsproblem innebär att man har datapunkter och vill ge varje punkt en viss ”klass” eller ”kategori”, några modeller som kan göra detta är Logistisk Regression, Beslutsträd och Support Vector Machines som använder clustering.

Potentiella tillämpningsområdet kan innefatta; kommer den här kunden lämna företaget? (Binär klassificering, JA eller NEJ).

Är personen på bilden; Barn, tonåring, vuxen, gammal? (Multiklassificering).

Ett confusion matrix kan användas för att utvärdera sin modell, där man har **Riktiga** värden längs ena axeln och **Predikterade** värden längs den andra axeln, på så sätt kan man lätt se vilka fel modellen gör. Exempelvis är det vanligt att datan delas in i ”True positive”, ”False Positive”, True negative”, ”False negative” för att enkelt se modellens effektivitet.

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

* K-means delar in datan i kluster, där centret i varje kluster kallas för cluster centroid. Varje observation tilldelas det cluster som ligger närmast medelvärdet för observationen, på så sätt grupperas observationer som är ”liknande”.

K-means är unsupervised, den används alltså när man inte vet vad target values är.

Detta kan användas till mycket, tillexempel kundsegmentering eller för att kartlägga politiska åsikter (gruppera kunder eller väljare som liknar varandra).

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

* Ordinal encoding: Används när man har en rankning i de värden man ska encoda (Exempel: på en betygsskala IG-MVG skulle följande encoding ske; IG=0, G=1, VG=2, MVG=3) där IG klassas som ”sämst” och MVG ”bäst”
* One-hot encoding: En encoding där enbart en av varje observation är ”true” samtidigt, alla andra är ”false” (Exempel: först kollar modellen om IG är ”1”, isåfall är alla andra kategorier ”0”. Nästa iteration är IG ”0”, G är ”1” och resterande är ”0”, osv.)
* Dummy Variable: Fungerar nästan som one-hot, men alltid med en färre kolumn, då den använder uteslutningsmetoden för att hitta en ”baseline”. (Exempel: IG är baseline, resterande resultat är antingen ”1” eller ”0”)

1. 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?

* Det här är en tolkningsfråga enligt mig, då färger inte har en *naturlig* inbördes rankning, dvs Julia kan tycka att röd är vackrast, medan Göran tycker blå är vackrast. Mitt svar blir ändå att Julia har rätt; om man väljer att ranka enligt ”vackrast” så är det ordinalt.

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

* Streamlit är en tjänst (ett open source python library) som används för att snabbt och smidigt lägga upp pythonkod i applikationer på en ”hemsida”, så du enkelt kan dela det du skapat med andra utan att behöva kunna Javascript eller annat kodspråk.

# Självutvärdering

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

En stor utmaning var att förstå exakt hur modellen ser på vår data, vilken form den har och hur olika processeringstekniker påverkar formen på datan för att de ska likna träningsdatan, men om man tar ett steg i taget och förstår varje steg så klarnar bilden ganska fort.

En annan utmaning var att förstå varför vissa modeller fungerade bättre på mina egna bilder än andra, och ärligt talat är jag inte helt säker att jag förstår det även nu i efterhand. Jag har läst mycket i boken Hands-On machine learning~ för att få bättre förståelse.

Men den största utmaningen var att min Streamlit app inte hade tillåtelse att hämta filer från min dator, jag satt i timmar och felsökte, hittade inte felet och ändrade helt enkelt målet med mitt projekt för att ”gå runt” problemet istället. Jag kan säkert lösa det med tillräcklig tid, men kände att det var onödigt att lägga energi där.

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

Förhoppningsvis har jag påvisat tillräckligt mycket kunskap och insikt för att uppnå VG-kriterierna samt att min problemlösningsförmåga är tillräcklig för att lösa de problem som kan uppstå. VG, förutsatt att rapporten är tillräcklig.

1. Något du vill lyfta fram till Antonio?

Kul projekt! Det visade på ett bra sätt hur modellerna fungerar, interagerar med datan man har och speciellt hur mycket förbättringspotential som finns. Ett väldigt bra intro till machine learning! Är väldigt osäker på rapportskrivandet, det är inget jag gjort tidigare.

# Appendix A

mnist = fetch\_openml('mnist\_784', version=1, cache=True, as\_frame=False)

X = mnist["data"]

y = mnist["target"].astype(np.uint8)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=55, stratify=mnist["target"])

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

scaler\_fit = scaler.fit(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

joblib.dump(scaler\_fit, "minmaxscaler2.pkl")

#Params already optimized and reduced by GridSearchCV

params = {

'n\_neighbors' : 3,

'weights' : 'distance',

'p' : 1,

'leaf\_size' : 15,

'n\_jobs' : 1

}

knn = KNeighborsClassifier(\*\*params)

knn.fit(X\_train\_scaled, y\_train)

y\_pred = knn.predict(X\_test\_scaled)

joblib.dump(knn, "knn\_model.pkl")

# Källförteckning

Géron, A. (2019). *Hands-On Machine Learning with Scikit-learn, Keras and Tensorflow* (2nd edition). O’Reilly

”What is OCR (Optical Character Recognition)?”

[*https://aws.amazon.com/what-is/ocr/*](https://aws.amazon.com/what-is/ocr/)

”OpenCV-Python Tutorials”

[*https://docs.opencv.org/4.x/d6/d00/tutorial\_py\_root.html*](https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html)

”Adaptive Thresholding with OpenCV (cv2.adaptiveThreshold)”

*h*[*ttps://pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-a*](ttps://pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-a)*daptivethreshold/*

”Streamlit documentation”

https://docs.streamlit.io/