

# Jämförelse av Maskininlärningsmodeller för MNIST



ECUTBILDNING

Hani Abraksiea

EC Utbildning

ML\_kunskapskontroll\_2

2025–03

# 1 Abstract

In this study, machine learning was applied to model the MNIST dataset. Three models were evaluated: Logistic Regression, Random Forest, and Support Vector Machine (SVM). The data was split into 70% training, 15% validation, and 15% test sets. Random Forest achieved the highest accuracy of 96.69%, outperforming SVM (96.20%) and Logistic Regression (91.54%). Confusion matrices were plotted for all models, and the best model was deployed using Streamlit. The final Random Forest model was saved with Joblib for further use.

# Innehållsförteckning

Abstract .....	2
1 Inledning och syfte .....	1
1.1 Inledning .....	1
1.2 Syfte och Frågeställning .....	1
1.3 Text format .....	1
2 Teori.....	2
2.1 Klassificeringsmodeller .....	2
2.1.1 Logistic Regression.....	2
2.1.2 Random Forest – En Ensemble av Beslutsträd .....	3
2.1.3 Support Vector Machine (SVM).....	4
2.2 Streamlit.....	5
3 Metod.....	6
4 Resultat och Diskussion.....	8
4.1 Resultat .....	8
4.2 Diskussion .....	8
5 Slutsatser .....	10
6 Teoretiska frågor .....	11
7 Självutvärdering.....	13
Appendix A .....	14
Källförteckning.....	15

## 2 Inledning och syfte

### 2.1 Inledning

Maskininlärning har blivit en viktig teknik inom bildigenkänning, och MNIST-datasetet är ett välkänt benchmark för att klassificera handskrivna siffror. Genom att jämföra olika maskininlärningsmodeller kan vi identifiera deras styrkor och svagheter samt förbättra deras prestanda genom optimering.

I denna rapport undersöks tre modeller: Logistisk Regression, Random Forest och Support Vector Machine (SVM). Datasetet delas upp i träning (70%), validering (15%) och test (15%). För att förbättra prestandan normaliseras endast Random Forests data, medan Logistic Regression och SVM använder skalning. Slutligen implementeras den bästa modellen i en interaktiv applikation med Streamlit samt applikationen deployats i Streamlit Cloud

### 2.2 Syfte och Frågeställning

Syftet med denna rapport är att undersöka vilken maskininlärningsmodell som presterar bäst på MNIST-datasetet.

Följande frågeställningar besvaras:

1. *Vilken av Logistisk Regression, Random Forest och SVM ger högst noggrannhet?*
2. *Varför är noggrannhet det mest avgörande kriteriet för att välja den bästa modellen för detta projekt?*

### 2.3 Text format

Text är skriven på formatet Times New Roman med textstorlek 12.

# 3 Teori

## 3.1 Klassificeringsmodeller

Enligt Géron (2017) används klassificeringsmodeller för att förutsäga kategoriska etiketter (t.ex. ja/nej, hund/katt). Några vanliga modeller inkluderar:

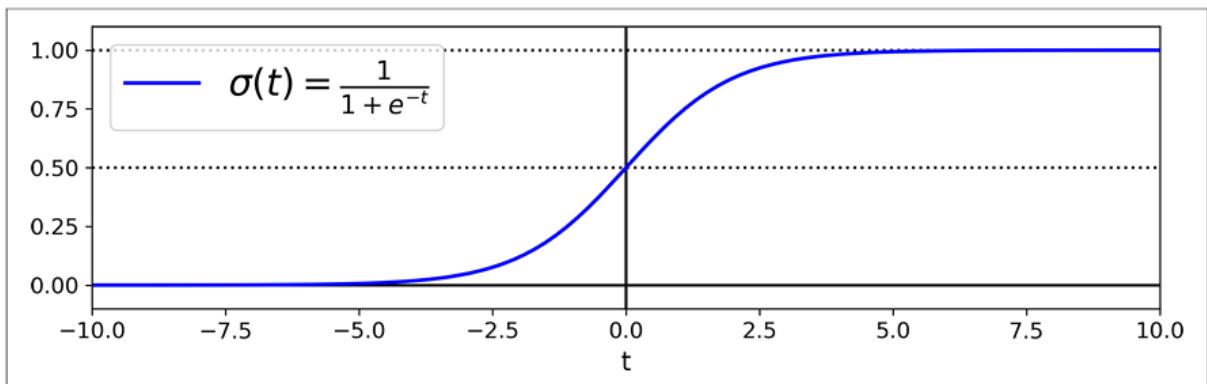
1. *Logistisk regression*: En enkel men kraftfull modell som använder sigmoid-funktionen för att uppskatta sannolikheter och klassificera data i två eller flera klasser.
2. *Support Vector Machine (SVM)*: Skapar en optimal separationslinje (hyperplan) för att maximera marginalen mellan klasserna. Bra för icke-linjära data med hjälp av kärnmetoder (kernels).
3. *Beslutsträd & Random Forest*:
  - Beslutsträd delar upp data genom att skapa ja/nej-frågor.
  - Random Forests bygger flera beslutsträd och kombinerar resultaten för mer robusta prediktioner.
4. *k-Nearest Neighbors (k-NN)*: Klassificerar en ny datapunkt baserat på dess närmaste grannar i datan. Enkel men kan vara långsam på stora dataset.
5. *Neurala nätverk*: Består av flera lager av noder som lär sig komplexa mönster i data. Används ofta för djupinlärning.

Dessa modeller har olika styrkor beroende på datan, storlek, och komplexitet av problemet. I denna rapport fokuseras på tre modeller om beskrivs mer nedan:

### 3.1.1 Logistic Regression

**Logistisk regression** (eller Logit Regression) är en vanlig metod för att uppskatta sannolikheten att en instans tillhör en viss klass, exempelvis om ett e-postmeddelande är skräppost eller inte. Modellen predikterar att instansen tillhör den positiva klassen (märkt som "1") om sannolikheten är över 50%, annars predikteras den tillhöra den negativa klassen (märkt som "0"). Detta gör logistisk regression till en binär klassificerare.

Logistisk regression beräknar en viktad summa av ingångsfunktionerna (plus en bias-term). Det beräknas ett resultat genom en logistisk funktion, även kallad sigmoidfunktion, som producerar ett värde mellan 0 och 1. Om den estimerade sannolikheten är över 0,5, klassificeras instansen som tillhörande den positiva klassen (Géron, 2017). Se *figur 1* som visar logistisk funktion!

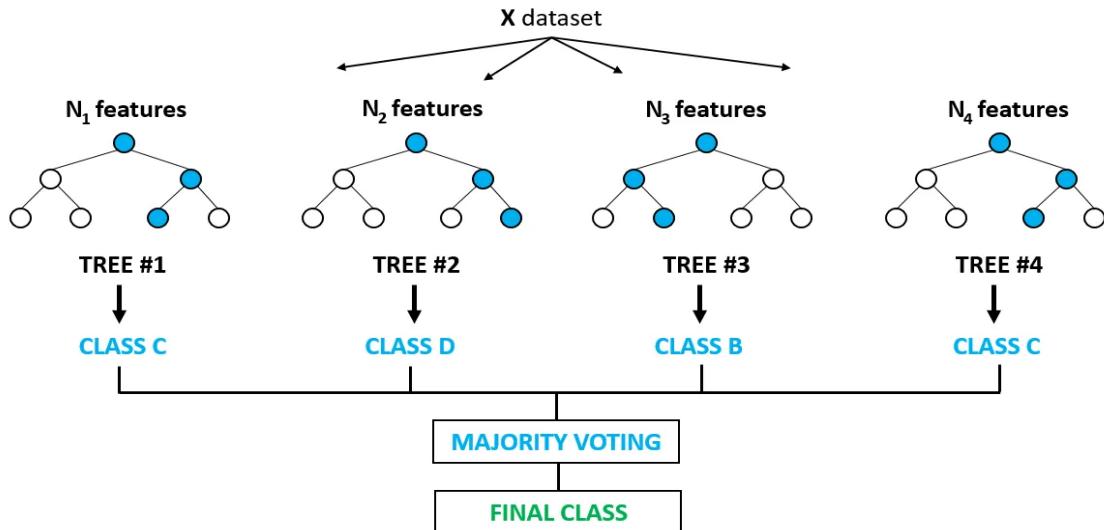


Figur 1: Logistisk Funktion. "Bildkälla: Géron, A. (2017, s. 143)".

Modellen beräknar en logit, som är det omvänta av den logistiska funktionen och representerar logaritmen av oddsen mellan sannolikheten för den positiva klassen och den negativa klassen (Géron, 2017).

### 3.1.2 Random Forest – En Ensemble av Beslutsträd

**Random Forest** är en ensemblemodell som bygger på flera beslutsträd och använder bagging-metoden för att förbättra modellens prestanda. Istället för att skapa ett enda beslutsträd, som riskerar att överanpassa sig till träningsdatan, kombinerar Random Forest flera träd för att öka modellens generaliseringsförstående (Géron, 2017). Se *figur 2* om visar Random Forest algoritm (Abilash, n.d.).



Figur 2: Random Forest Algorithm. "Bildkälla: Abilash (n.d.), Medium."

Random Forest tränas genom bagging (bootstrap aggregating), vilket innebär att varje träd i skogen tränas på en slumpmässig delmängd av träningsdatan. Bagging minskar variansen och gör modellen mer stabil. Istället för att varje träd alltid väljer den bästa möjliga egenskapen

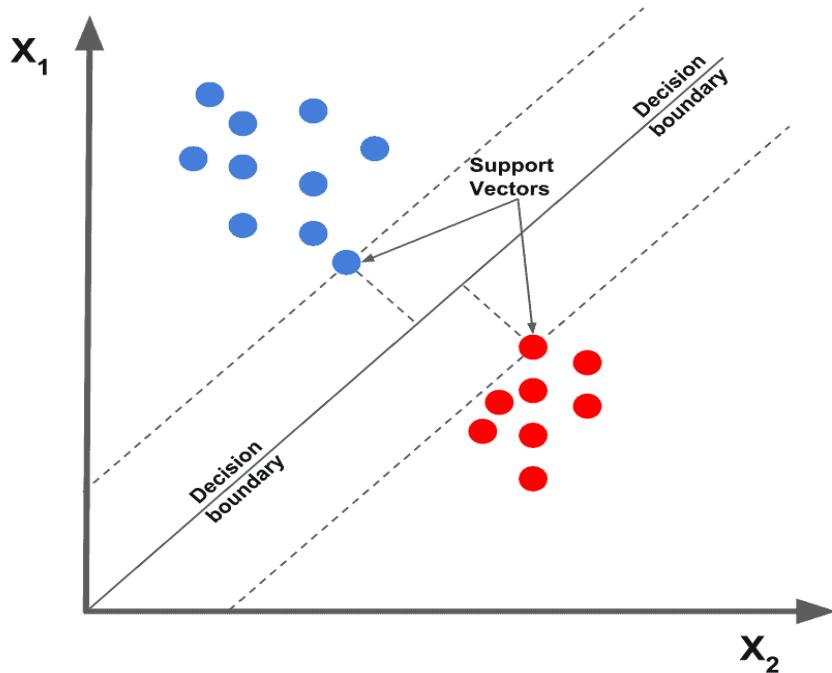
vid varje splittring, väljer Random Forest en slumpmässig delmängd av egenskaper, vilket gör träden mer varierade och minskar risken för att hela modellen blir överanpassad (Géron, 2017).

Enligt Géron (2017) är RandomForestClassifier i Scikit-Learn en mer optimerad och bekväm implementation av bagging-metoden för beslutsträd. Den har flera viktiga hyperparametrar:

- **n\_estimators**: Antal beslutsträd i skogen. Fler träd förbättrar ofta modellens prestanda men ökar beräkningstiden.
- **max\_leaf\_nodes**: Begränsar antalet blad i varje träd, vilket gör modellen enklare och minskar risken för överanpassning.
- **n\_jobs**: Anger hur många CPU-kärnor som ska användas för träningen (-1 betyder att alla tillgängliga kärnor används).

### 3.1.3 Support Vector Machine (SVM)

Support Vector Machine (SVM) är en kraftfull modell för både klassificering och regression, men används främst för klassifieringsproblem. Huvudidén är att SVM försöker hitta en optimal separerande hyperplan som maximerar marginalen mellan klasserna i ett dataset. SVM letar efter det hyperplan som har störst avstånd (margin) till de närmaste datapunkterna i varje klass, som kallas support vectors (Géron, 2017). Se *figur 3* som visar SVM metod (Mallick, 2018).



*Figur 3: The support vector machines SVM method. "Bildkälla: Mallick (2018), LearnOpenCV."*

Det finns Linjär och Icke-linjär Klassificering. Vid linjära problem räcker det med ett linjärt hyperplan för att separera klasserna. Vid icke-linjära problem kan SVM använda kärntrick (kernel trick) för att projicera data till en högre dimension där den blir linjärt separerbar (Géron, 2017).

Enligt Géron (2017) kan SVM hantera komplexa, icke-linjära relationer genom att använda olika kärnfunktioner (kernels), exempelvis:

- Polynomiell kärna
- Radial Basis Function (RBF)

## 3.2 Streamlit

Streamlit är ett open-source Python-ramverk som gör det enkelt att skapa interaktiva webbapplikationer för dataanalys och maskininlärning utan att behöva kunskaper i webbutveckling. Med en enkel och intuitiv API kan användare snabbt bygga visuella gränssnitt för sina modeller och data. Streamlit uppdaterar applikationerna automatiskt vid varje ändring i kodén, vilket gör det smidigt att iterera och testa nya funktioner. Tack vare inbyggda widgets kan användare skapa dynamiska och interaktiva appar utan att behöva hantera frontend-programmering.

För att installera Streamlit används:

```
bash  
pip install streamlit
```

Därefter kan en demoapplikation köras med:

```
bash  
streamlit hello
```

Streamlit är särskilt användbart för datavetare och maskininlärningsstudenter som vill snabbt visualisera sina projekt och dela sina resultat (Streamlit, n.d.).

## 4 Metod

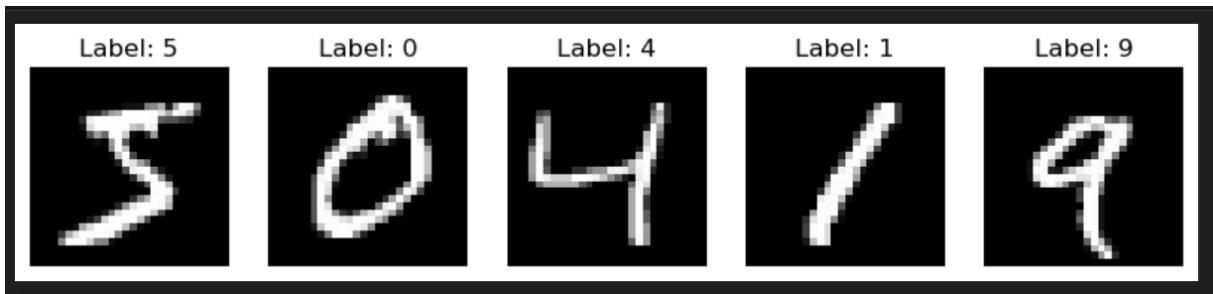
Arbetet genomfördes i flera steg:

### 1. Datainsamling och förbehandling

- MNIST-datasetet hämtades från OpenML med hjälp av funktionen `fetch_openml` från scikit-learn, vilket gjorde det möjligt att ladda ner datasetet direkt i Python.

För att bättre förstå datan, plottades några exempel från MNIST-datasetet.

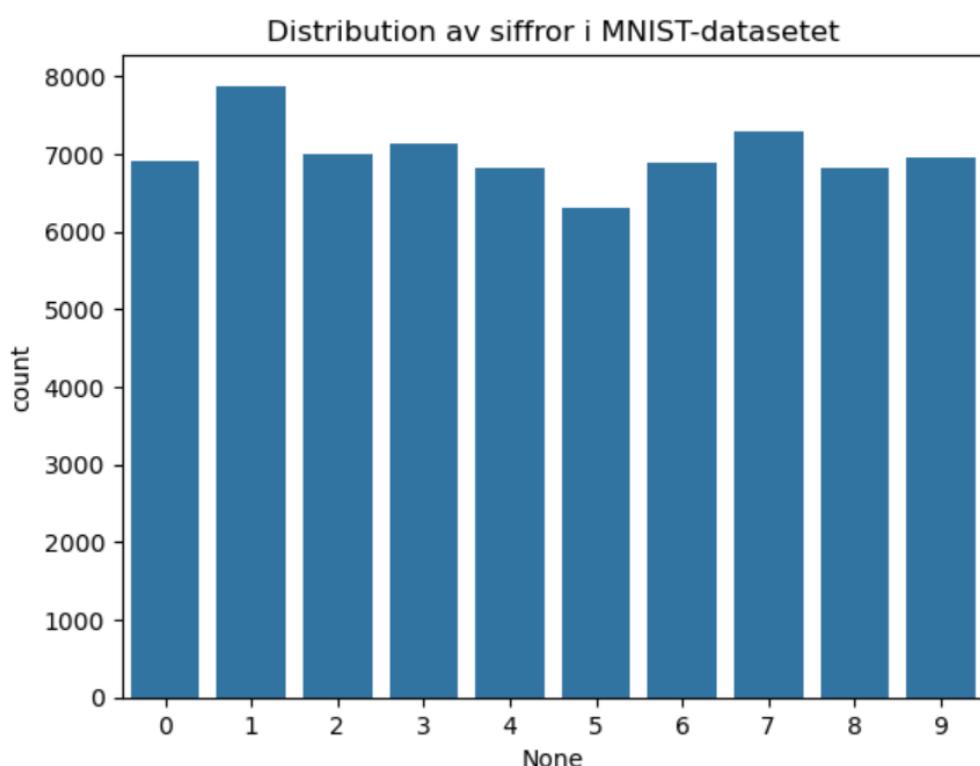
Figur 4 visar ett urval av exempel från MNIST-datasetet.



Figur 4: Ett urval av exempel från MNIST-datasetet

Innan träning av modellerna analyserades fördelningen av siffror i datasetet.

Figur 5 visar antal förekomster av varje siffra i MNIST-datasetet.



Figur 5: Antal förekomster av varje siffra i MNIST-datasetet

- Datasetet delades in i träning (70 %), validering (15 %) och test (15 %).
- Data normaliseras endast för Random Forest-modellen.

- Data skalades med StandardScaler för Logistisk Regression och SVM.

## 2. Träning av modeller

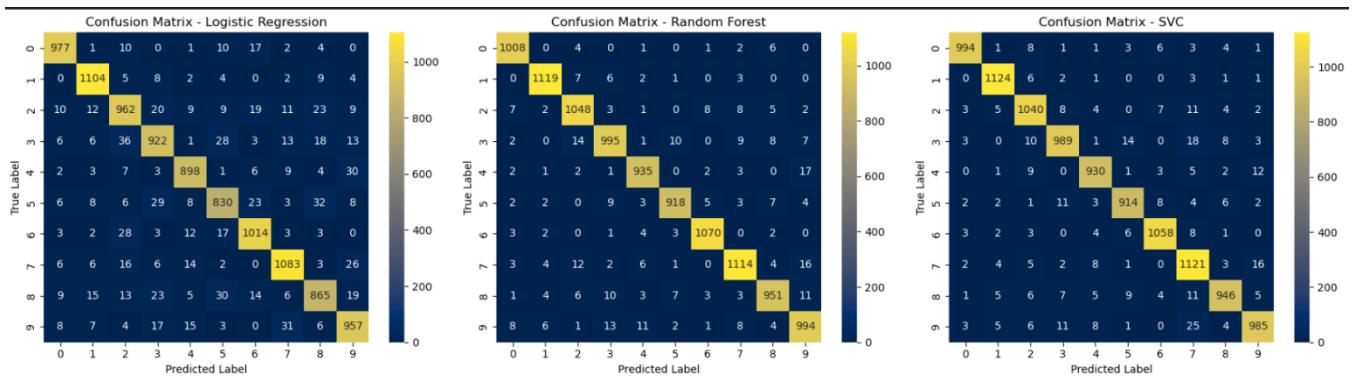
- Tre modeller testades: Logistisk Regression, Random Forest och SVM.

## 3. Utvärdering och analys

- Modellerna utvärderades baserat på noggrannhet.
- Confusion matrix användes för att analysera prestandan hos varje modell på valideringsdata.

Confusion matrices för alla tre modellerna på valideringsdata visas nedan.

Figur 6 visar dessa matriser, som är baserade på resultaten från de tränade modellerna.



Figur 6: Confusion matrices för alla tre modellerna på valideringsdata

- Confusion matrix plottades även för den bästa modellen (Random Forest) på testdata.

## 4. Implementering av bästa modell

- Den bästa modellen (Random Forest) sparades med joblib och integrerades i en interaktiv applikation med Streamlit, där Canvas användes för att möjliggöra användarinmatning. Efter det applikationen deployats som en webbaserad app i Streamlit Cloud.

## 5 Resultat och Diskussion

### 5.1 Resultat

I detta avsnitt presenteras resultaten från de tre maskininlärningsmodellerna: Logistisk Regression, Random Forest och Support Vector Machine (SVM). Modellerna utvärderades baserat på deras noggrannhet, och resultaten för varje modell är sammanfattade i tabellen nedan:

Noggrannhet (Accuracy) för olika modeller	
Logistisk Regression	91.54%
Random Forest	96.69%
Support Vector Machine (SVM)	96.20%

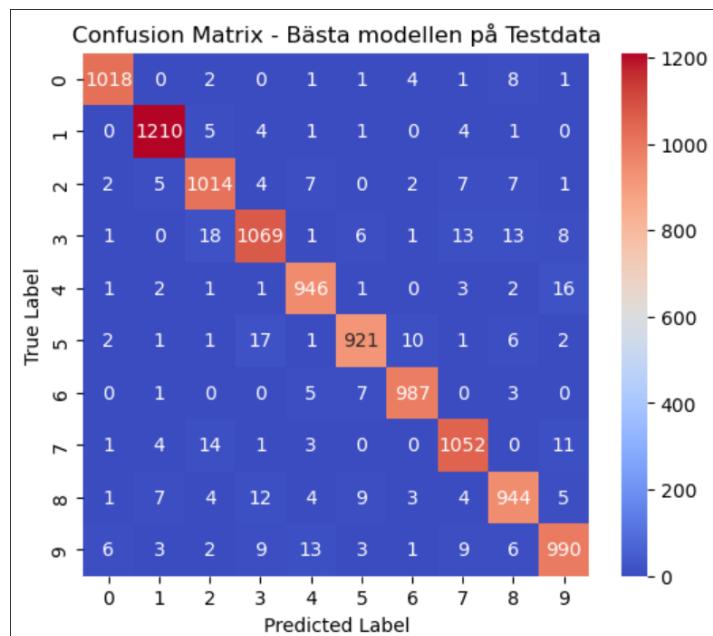
Tabell 1: Noggrannhet (Accuracy) för de tre valda modellerna.

### 5.2 Diskussion

Random Forest-modellen gav den högsta noggrannheten på 96.69%, vilket gör den till den bästa modellen för denna uppgift. Denna höga noggrannhet kan bero på att Random Forest använder en ensemblemetod, vilket gör att den kan generalisera bättre och undvika överanpassning. Även om både SVM och Logistisk Regression visade bra resultat, presterade Random Forest överlägset.

Confusion matrixen för varje modell visade att Random Forest hade den lägsta mängden felklassificeringar, särskilt för de svåraste siffrorna. Baserat på dessa resultat valdes Random Forest som den bästa modellen för vidare implementering.

Figur 7 visar confusion matrix för Random Forest-modellen på testdata, vilket ger en detaljerad bild av hur den bästa modellen presterade.



Figur 7: Confusion matrices för Random Forest bästa modellen på testdata

Noggrannhet var det främsta måttet för att välja den bästa modellen, eftersom det är den viktigaste faktorn för att korrekt klassificera handskrivna siffror. Detta val säkerställer att modellen uppfyller projektets primära mål.

## 6 Slutsatser

Syftet med denna rapport var att undersöka vilken maskininlärningsmodell som presterar bäst på MNIST-datasetet och att jämföra modellerna Logistisk Regression, Random Forest och Support Vector Machine (SVM) utifrån noggrannhet och prestanda.

***Frågeställning 1: Vilken av Logistisk Regression, Random Forest och SVM ger högst noggrannhet?***

Resultaten visar att Random Forest gav den högsta noggrannheten på 96.69%, vilket gör den till den bästa modellen för detta projekt. SVM följde strax efter med en noggrannhet på 96.20%, medan Logistisk Regression hade en noggrannhet på 91.54%.

***Frågeställning 2: Varför är noggrannhet det mest avgörande kriteriet för att välja den bästa modellen för detta projekt?***

Noggrannhet är det viktigaste kriteriet i detta projekt eftersom målet är att korrekt klassificera handskrivna siffror från MNIST-datasetet. Felklassificeringar kan leda till att siffror tolkas felaktigt, vilket kan få allvarliga konsekvenser i tillämpningar som digitala formulär eller automatiserade system. Därför prioriterades en modell med högst noggrannhet.

Sammanfattningsvis kan det dras slutsatsen att Random Forest är den mest lämpliga modellen för denna typ av bildklassificering på MNIST-datasetet. Modellen visade den bästa prestandan och ger en robust lösning för vidare implementation i applikationer. Modellen har också deployats som en webbaserad app på Streamlit Cloud, vilket gör det möjligt för användare att interagera med modellen och se prediktioner i realtid.

## 7 Teoretiska frågor

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

”Träning” används för att träna modellen så den lär sig mönster i datan. ”Validering” används för att justera modellens parametrar och undvika överträning. Medan ”Test” används för att utvärdera modellens slutliga prestanda på ny data.

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”?

Hon kan använda cross-validation (korsvalidering) på träningsdatan för att utvärdera modellerna och välja den som presterar bäst.

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

Ett regressionsproblem är en typ av maskininlärningsproblem där målet är att förutsäga ett kontinuerligt värde snarare än en kategori. Exempel på modeller är Linjär regression, Lasso regression och Random Forest. Exempel på tillämpningsområden är fastighetspriser, förutsäga bostadspriser baserat på storlek, läge och andra faktorer samt hälsa, förutsäga patienters blodtryck eller livslängd baserat på hälsodata.

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

$$RMSE = \sqrt{\frac{1}{n} \sum_1^n (y_i - \hat{y}_i)^2}$$

Root Mean Squared Error (RMSE) är ett mått på hur bra en modell presterar genom att beräkna medelavvikelsen mellan de faktiska värdena och de förutsagda värdena. Ett lägre RMSE betyder att modellen gör mer exakta förutsägelser medan ett högre RMSE betyder att modellen har större fel och är mindre pålitlig. RMSE användas i regressionsproblem för att se vilken modell som har bäst prediktionsförmåga.

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

Ett klassificeringsproblem handlar om att förutsäga vilken kategori eller klass ett objekt tillhör baserat på dess egenskaper. Exempel på modeller är Logistisk

regression, Random Forest och Support Vector Machine (SVM). Exempel på tillämpningsområden är spamfiltering och medicinsk diagnos.

En "Confusion Matrix" är en tabell som används för att utvärdera prestanda hos en klassificeringsmodell genom att visa antalet rätta och felaktiga förutsägelser för varje klass.

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

K-means är en metod för att gruppera liknande data i kluster. Modellen delar upp data i k antal kluster baserat på avståndet till klustrets medelpunkt. Exempel på det är kundsegmentering för att skapa olika grupper baserat på köpbeteende.

7. Förklara (gärna med ett exempel): Ordinal encoding, one-hot encoding, dummy variable encoding. Se mappen "l8" på GitHub om du behöver repetition.

- *Ordinal Encoding*: Används för att konvertera kategoriska data med en naturlig ordning till siffror. Exempel: "Låg", "Medium", "Hög" blir 0, 1, 2.

- *One-hot Encoding*: Skapar nya kolumner för varje kategori och markerar med 1 eller 0 beroende på om kategorin är närvarande. Exempel: "Röd", "Blå", "Grön" blir 3 nya kolumner där bara en får 1 för varje observation.

- *Dummy Variable Encoding*: Liknar One-hot, men en kategori tas bort för att undvika multikollinearitet. Exempel: Om vi har "Röd", "Blå", "Grön", skapas två kolumner, t.ex. "Blå" och "Grön", där "Röd" är basen och representeras av 0 i båda kolumnerna.

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?

Julia har rätt. Färger är normalt nominal (ingen ordning) men om de används för att visa rang t.ex. röd skjorta = vackrast blir de ordinal. Det beror på hur datan tolkas.

9. Kolla följande video om Streamlit: <https://www.youtube.com/watch?v=ggDaRzPP7A&list=PLgzaMbMPEHEx9Als3F3sKKXexWnyEKH45&index=12> Och besvara följande fråga: - Vad är Streamlit för något och vad kan det användas till?

Streamlit är ett Python-bibliotek som gör det enkelt att skapa interaktiva webbaserade appar för dataanalys och maskininlärning. Det kan användas för att snabbt visualisera data, bygga och dela appar utan att behöva mycket webbutvecklingskunskap.

## 8 Självutvärdering

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

I streamlit app har jag haft ett problem att appen gissa samma siffra heltiden och har löst problem med hjälp av mina klasskamrater.

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

Gärna VG då mitt projekt uppfyller kriterierna för VG betyg.

3. Något du vill lyfta fram till Antonio?

Det var en spännande och utmanade kurs och tack att du uppmuntrat oss att samarbeta med varandra.

## Appendix A

Källkoden för projektet finns tillgänglig på GitHub:

🔗 GitHub-repository [https://github.com/HaniAbraksiea/ds24\\_ml](https://github.com/HaniAbraksiea/ds24_ml)

Den färdiga webbaserade applikationen kan testas här:

🔗 Streamlit-app <https://randomforesthani.streamlit.app/>

## Källförteckning

- Géron, A. (2017). *Hands-on machine learning with Scikit-Learn, Keras & TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.
- Streamlit. (n.d.). *The fastest way to build and share data apps*. Hämtad från <https://streamlit.io>
- Abilash, R. (n.d.). *Applying Random Forest (Classification) — Machine Learning Algorithm from Scratch with Real Datasets*. Medium. Hämtad från <https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real-24ff198a1c57>
- Mallick, S. (2018, July 11). *Support Vector Machines (SVM)*. LearnOpenCV. Hämtad från <https://learnopencv.com/support-vector-machines-svm/>
- LeCun, Y., Cortes, C., & Burges, C. J. (2010). *The MNIST database of handwritten digits*. Hämtad från <http://yann.lecun.com/exdb/mnist/>