Modellering av MNIST med scikit-learn: Jämförelse av Flera Modeller

Machine Learning

Maryam Hult Nikpour

EC Utbildning

Machine Learning

2025

Abstract

In this project, I applied machine learning techniques to classify handwritten digits using the MNIST dataset. I compared five models including Logistic Regression, Linear SVC, k-Nearest Neighbors (k-NN), Random Forest, and Extra Trees, based on their accuracy scores.

After identifying k-NN and Extra Trees as the top performers, I tuned their hyperparameters and combined them into an ensemble model. The final ensemble achieved an accuracy of 97.42% on the test set, demonstrating improved performance over the individual models.

Further analysis using confusion matrices and classification reports showed that the model generalizes well, with only a few digits (e.g., 8 and 9) being more prone to misclassification.

Contents

1 Inledning

1.1 Bakgrund

Vi befinner oss idag i den fjärde industriella revolutionen, där data har blivit en central drivkraft för innovation och utveckling. Organisationer och industrier använder stora datamängder, artificiell intelligens (AI) och maskininlärning (ML) för att effektivisera processer, fatta smartare beslut och automatisera uppgifter. Maskininlärning används inom en rad olika områden, såsom bildigenkänning, medicinsk diagnostik, självkörande fordon och finansiell analys.

Ett klassiskt exempel på maskininlärning inom bildigenkänning är MNIST-datasetet. Det består av 70 000 små bilder av handskrivna siffror (0–9), insamlade från gymnasieelever och anställda vid USA:s folkräkningsbyrå (US Census Bureau). Varje bild har en tillhörande etikett som anger vilket nummer den representerar. MNIST har blivit ett standardbenchmark inom maskininlärning, eftersom forskare och utvecklare ofta testar sina nya klassificeringsalgoritmer på detta dataset.

MNIST är en multiklass klassificering problem, där varje bild ska klassificeras i en av 10 klasser (siffrorna 0 till 9). Detta gör det till ett utmärkt dataset för att utvärdera prestandan hos olika maskininlärningsmodeller.

I detta arbete används scikit-learn, ett av de mest populära Python-biblioteken för maskininlärning. Scikit-learn tillhandahåller kraftfulla verktyg för datahantering, modellträning, hyperparametertuning och utvärdering av maskininlärningsmodeller. Det erbjuder ett enkelt gränssnitt för att implementera och experimentera med olika algoritmer, vilket gör det idealiskt för både nybörjare och experter inom maskininlärning.

1.2 Syfte och Frågeställning

Syftet med denna rapport är att undersöka hur olika maskininlärningsmodeller presterar på MNIST-datasetet när de implementeras med scikit-learn. För att uppfylla detta syfte kommer följande frågeställningar att besvaras:

Vilka maskininlärningsmodeller ger bäst prestanda vid klassificering av MNIST?

Hur påverkar olika parametrar och inställningar modellernas prestanda?

Vilka för- och nackdelar har de olika modellerna i termer av noggrannhet och beräkningseffektivitet?

2 Teori

I detta projekt använde jag först två modeller som traditionellt används för binär klassificering, nämligen logistisk regression med One-vs-Rest (OvR) och linjär SVM (Support Vector Machine). Därefter testade jag även k-nearest neighbors (k-NN), Random Forest och Extra Trees.

Slutligen kombinerade jag de två bäst presterande modellerna i en ensemblemodell, som fungerade som den slutliga modellen för att klassificera MNIST-data. Här kommer jag kortfattat att förklara de modeller jag använde.

2.1 Modeller

2.1.1 Logistic Regression

Logistisk regression är en övervakad maskininlärningsalgoritm som främst används för binär klassificering. Den förutsäger sannolikheten att en instans tillhör en viss klass. För multiklassklassificering kan logistisk regression anpassas med strategier som One-Versus-Rest (OvR) och One-Versus-One (OvO).

2.1.2 One-Versus-Rest (OvO)

Ett sätt att klassificera handskrivna siffror (0–9) är att träna 10 binära klassificerare, där varje modell är specialiserad på att känna igen en specifik siffra (t.ex. en 0-detektor, 1-detektor osv.). Vid klassificering beräknas beslutsvärdet för varje modell, och siffran med det högsta värdet väljs som klass. Denna metod kallas One-Versus-Rest (OvR)-strategin

2.1.3 Linear Support Vector Machine

Linear SVC är en klassificeringsmodell baserad på Support Vector Machines (SVMs), som används för att hitta en optimal hyperplan som separerar olika klasser. Modellen är särskilt effektiv för binär klassificering, men kan även hantera multiklassproblem med strategier som One-Versus-Rest (OvR). Linear SVC är snabbare och mer minneseffektiv än den vanliga SVC-modellen, särskilt för stora dataset.

2.1.4 k-nearest neighbors (K-NN)

k-NN är en instansbaserad klassificeringsalgoritm som klassificerar en ny datapunkt baserat på de k närmaste grannarna. Klassens etikett bestäms genom majoritetsröstning. Antalet grannar (k) är en användardefinierad parameter som påverkar modellens noggrannhet och känslighet för brus. Algoritmen kan användas för både binär och multiklassklassificering, där den klass med flest grannar väljs

2.1.5 Random forest

Random Forest är en ensemblemetod som kombinerar flera beslutsträd för att förbättra noggrannhet och minska överanpassning. Modellen tränar beslutsträd på olika delmängder av datasetet och använder medelvärdesbildning (för regression) eller majoritetsröstning (för

klassificering) för att göra en slutlig förutsägelse. Träden använder en bästa delningsstrategi för att optimera uppdelningen av data.

2.1.6 Extra Trees

Extra Trees är en ensemblemetod liknande Random Forest, men med mer slumpmässighet i hur träden byggs. Istället för att välja den bästa delningen, väljer modellen delningspunkter slumpmässigt, vilket gör träningen snabbare och minskar risken för överanpassning. Den slutliga förutsägelsen görs genom majoritetsröstning vid klassificering.

2.1.7 Ensamble modell

Ensemblemetoder kombinerar förutsägelser från flera basmodeller för att förbättra generaliseringsförmåga och robusthet jämfört med en enskild modell. Genom att använda flera estimatorer reduceras risken för överanpassning och ökar modellens noggrannhet.

2.2 Justering av hyperparametrar

Hyperparametrar styr hur en modell tränas och påverkar dess prestanda, exempelvis inlärningshastighet, regularisering och maxdjup för beslutsträd. De väljs innan träning och måste ofta justeras för att undvika över- eller underanpassning. Eftersom manuell justering är tidskrävande används automatiserade metoder som GridSearchCV och RandomizedSearchCV.

2.2.1 GridSearchCV

GridSearchCV testar systematiskt alla kombinationer av angivna hyperparametrar med korsvalidering. Det är noggrant men kan bli tidskrävande vid stora sökrymder.

2.2.2 RandomizedSearchCV

RandomizedSearchCV testar ett slumpmässigt urval av parametrar. Det är snabbare än GridSearchCV och passar bra vid stora eller komplexa sökutrymmen.

2.3 Utvärdering av klassificeringsmodeller

För att bedöma hur väl en klassificeringsmodell presterar används flera olika mått. Nedan beskrivs några vanliga metoder.

2.3.1 Accuracy Score

Accuracy (noggrannhet) är andelen korrekta förutsägelser jämfört med det totala antalet. Det är ett enkelt och ofta användbart mått, men kan vara missvisande vid obalanserade datamängder.

$$Accuracy\ score = \frac{Antal\ korrekta\ förutsägelser}{Total\ antal\ förutsägelser}$$

2.3.2 Confusion Matrix

En confusion matrix visar antalet rätt och fel förutsägelser för varje klass. Den ger en tydlig bild av hur ofta modellen förväxlar olika klasser.

2.3.3 Classification Report

Classification report innehåller precision, recall och F1-score för varje klass. Dessa mått ger en mer detaljerad bild av modellens styrkor och svagheter, särskilt vid obalanserade da

3 Metod

3.1 Importera nödvändiga moduler och ladda in data

Jag började detta projekt med att importera de nödvändiga biblioteken. Om ytterligare bibliotek behövdes under arbetets gång, installerade och uppdaterade jag dem vid behov. Därefter laddade jag in MNIST-datasetet, ett standarddataset för bildklassificering.

3.2 Exploray Data Analysis (EDA)

Jag började den utforskande dataanalysen genom att undersöka datasetets struktur, vilket visade att MNIST är ett dictionary-liknande objekt (Bunch), inte en DataFrame. Eftersom det fungerar som en ordbok, listade jag alla tillgängliga nycklar med metoden .keys().

Bland dessa använde jag endast två nycklar:

- data – Innehåller en array där varje rad representerar ett prov (en bild) och varje kolumn representerar en funktion (pixelvärde).
- target – Innehåller en array med etiketter (siffror 0–9).

Dessutom ger ”DESCR” en beskrivning av datasetet, vilken kan nås via mnist.DESCR.

Därefter undersökte jag datasetets form, vilket bekräftade att det innehåller:

- 70 000 prover (rader) och 784 funktioner (kolumner).Varje prov representerar en utplattad 28×28-bild som konverterats till en 1D-array med 784 pixelvärden.
- 70 000 etiketter, där varje etikett motsvarar en siffra (0–9).

3.2.1 Visualisering av ett slumpmässigt prov

För att verifiera att bilderna och etiketterna var korrekt lagrade i mnist, omformade jag ett slumpmässigt valt prov tillbaka till 2D (28×28 pixlar) och visade det tillsammans med dess motsvarande etikett.

Efter det delade jag upp datasetet i X (funktioner) och y (etiketter) för modellträning. Jag kontrollerade även om det fanns saknade värden, men datasetet visade sig vara rent, utan några saknade värden.

3.2.2 Klassfördelning

Jag undersökte hur många prover som fanns för varje siffra (0–9). Datasetet är något obalanserat, där vissa siffror förekommer oftare än andra. Till exempel förekommer siffran 5 totalt 6 313 gånger, medan siffran 1 förekommer 7 877 gånger. Idealt bör alla tio klasser ha en liknande fördelning för att undvika att modellen blir snedvriden mot mer frekventa siffror.

3.3 Förbereda Data

Eftersom MNIST-datasetet inte innehåller några saknade värden, krävdes ingen datarengöring. Däremot varierar pixelvärdena i X mellan 0 och 255, så normalisering tillämpades innan datasetet delades upp. Detta är särskilt viktigt för modeller som är känsliga för skalning av funktioner, eftersom det säkerställer att stora pixelvärden (255) inte dominerar inlärningsprocessen.

Datasetet delades upp i tre delmängder:

- 50 000 prover för träning,
- 10 000 för validering,
- 10 000 för testning.

Att dela upp datasetet i tre delmängder förhindrar överanpassning och säkerställer en rättvis utvärdering av modellen.

3.4 Skapa och Jämföra Klassificeringsmodeller

I detta avsnitt tränar jag flera modeller för att klassificera MNIST-datasetet, vilket är ett multiklassklassificeringsproblem där bilder ska kategoriseras i tio klasser (siffrorna 0–9).

Enligt Géron (2019) kan multiklassklassificering hanteras antingen med modeller som naturligt stödjer flera klasser, eller genom att utöka binära klassificerare med strategier som One-Versus-One (OvO) eller One-Versus-Rest (OvR) (s. 100).

Härnäst kommer jag att träna och jämföra olika modeller för att identifiera den bäst presterande modellen för denna uppgift.

3.4.1 Logistisk Regression

För min första modell valde jag logistisk regression, som ursprungligen är utformad för binär klassificering. Eftersom MNIST är ett multiklassklassificeringsproblem (siffrorna 0–9), tillämpade jag One-Versus-Rest (OvR)-strategin (med hjälp av OneVsRestClassifier). Denna metod tränar en binär klassificerare per klass, där varje siffra behandlas som "denna klass mot alla andra", vilket gör att logistisk regression effektivt kan hantera multiklassklassificering (1).

3.4.2 Linjär SVC

För min andra modell valde jag Linjär SVC, en klassificeringsmodell baserad på Support Vector Machines (SVMs). Även om den ursprungligen är utformad för binär klassificering, kan den hantera multiklassproblem med strategier som One-Versus-Rest (OvR). Jag valde Linjär SVC eftersom den är snabbare än en standard SVM-klassificerare, vilket gör den mer

effektiv för stora dataset som MNIST. OvR är standardstrategin för LinearSVC när den används för multiklassklassificeringsproblem.

3.4.3 k-Nearest Neighbors (k-NN)

För den tredje modellen valde jag k-Närmsta Grannar (k-NN), som rekommenderas i *Hands-On Machine Learning* (Övning 1, Kapitel 3) (2). k-NN klassificerar prover baserat på deras närmsta grannar och kräver ingen explicit träningsfas.

3.4.4 Random Forest

För den fjärde modellen valde jag Random Forest, en ensemblemetod som kombinerar flera beslutsträd för att förbättra noggrannhet och minska överanpassning. Den är väl lämpad för multiklassklassificering och hanterar stora dataset effektivt.

3.4.5 Extra Trees

Den sista modellen jag valde var Extra Trees, en ensemblemetod liknande Random Forest, men med mer slumpmässighet i uppdelningen. Detta gör träningen snabbare och kan förbättra generaliseringen för stora dataset.

3.5 Hyperparameter Tuning

I detta steg jämförde jag Accuracy scores för de fem modellerna. Två modeller, k-NN och Extra Trees, uppnådde de högsta score, så jag beslutade att finjustera deras hyperparametrar.

Först använde jag GridSearchCV för båda modellerna, men Extra Trees var mycket långsam. För att förbättra effektiviteten bytte jag därför till RandomizedSearchCV för Extra Trees.

De bästa hyperparametrarna som hittades var:

Extra Trees: {'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 20}
k-NN: {'n_neighbors': 3, 'weights': 'distance'}

Därefter tränade jag om modellerna med de bästa hyperparametrarna, vilket ledde till en liten ökning i noggrannhet. Jag jämförde sedan confusion matrix för de två optimerade modellerna och analyserade även deras klassificeringsrapporter för en mer detaljerad utvärdering av deras prestanda.

3.6 Ensemble Model

Efter att ha finjusterat hyperparametrarna skapade jag en ensemblemodell genom att kombinera de två bäst presterande modellerna: k-NN och Extra Trees med hjälp av "Voting Classifier". Ensemblemodellen uppnådde en högre score än de individuella modellerna, vilket gjorde den till den bästa modellen för denna uppgift. Jag valde denna ensemble som min slutliga modell och använde den för testning på osedd data.

3.7 Utvärdering av den bästa modellen på testdatan

För att göra detta tränade jag om modellen med både tränings- och valideringsdata. Därefter gjorde jag förutsägelser på testdatan (X_test). Slutligen utvärderade jag modellens prestanda med accuracy score, confusion matrix och klassificeringsrapport.

Jag analyserade även misclassification report för att identifiera vanliga fel. För att bättre förstå felen visualiserade jag några felklassificerade bilder tillsammans med deras korrekta och förutsagda etiketter.

3.8 Att spara den bästa modellen

Slutligen sparade jag den bästa modellen för framtida användning med joblib.dump. Detta gör det möjligt att ladda in modellen senare utan att behöva träna om den

3.9 Interaktiv Modellapplikation med Streamlit

För att göra modellen interaktiv utvecklade jag en Streamlit-applikation där användaren kan rita en siffra (0–9) direkt på en canvas. Den ritade bilden förbehandlas genom gråskaleomvandling, storleksändring till 28×28 pixlar, normalisering och konvertering till rätt format innan modellen gör sin förutsägelse. Applikationen fungerar lokalt med god noggrannhet, men vid försök att publicera den på Streamlit Cloud uppstod problem med beroenden och modellens storlek, vilket förhindrade att den fungerade online.

3.10 Utvärdering på Egna Skannade Siffror

Ett annat tillvägagångssätt jag använde var att skriva siffrorna 0 till 9 på ett papper och skanna det. Med hjälp av biblioteket cv2 i Python bearbetade jag den skannade bilden och skapade 80 mindre bilder, där varje bild representerade en enskild siffra. Dessa bilder användes som ny, tidigare osedd data för att utvärdera min tidigare tränade modell. Jag analyserade modellens förutsägelser genom att räkna ut accuracy och titta på confusion matrix.

4 Resultat och Diskussion

4.1 Accuracy score för olika modeller

Tabell 1 visar accuracy score före och efter hyperparameteroptimering (för två modeller), samt prestandan för den slutliga ensemblemodellen. Logistisk regression (0,9164) och Linjär SVC (0,8753) hade den lägsta noggrannheten. Random Forest (0,9692) presterade bra men optimerades inte vidare. k-NN (0,9716) och Extra Trees (0,972) var de bästa enskilda modellerna efter optimering. Ensemblemodellen (k-NN + Extra Trees) nådde högst accuracy (0,9742), vilket visar att en kombination av modeller förbättrade prestandan.

Logistisk regression och Linjär SVC hade den lägsta prestandan i denna studie. Dessa är binära klassificerare som hanterar multiklassproblem med One-Versus-Rest (OvR), men detta kan vara mindre effektivt än modeller som naturligt stödjer multiklassklassificering, såsom beslutsträd och k-NN.

En viktig begränsning är att de använder linjära beslutsgränser, vilket gör dem mindre lämpade för icke-linjärt separerbara data som MNIST. Modeller som k-NN, Random Forest och Extra Trees är bättre anpassade till denna typ av problem.

En annan utmaning är deras känslighet för feature scaling. Även efter normalisering av pixelvärden kan dessa modeller vara mindre effektiva i högdimensionella miljöer. Dessutom kan OvR-strategin leda till överlappande beslutsområden, vilket ökar risken för felklassificering. Dessa faktorer förklarar varför Logistisk regression och Linjär SVC presterade sämre jämfört med mer flexibla och icke-linjära modeller.

Model	Accuracy Score	Hyperparameter Tuning
Logistisk Regression	0.9164	-
Linear SVC	0.8753	-
k-Nearest Neighbors (k-NN)	0.9702	0.9716
Random Forest	0.9692	-
Extra Trees	0.9715	0.972
Ensemble model	0.9742	

Tabell 1. Accuracy score för olika modeller

4.2 Utvärdering av den bästa modellen på testdatan

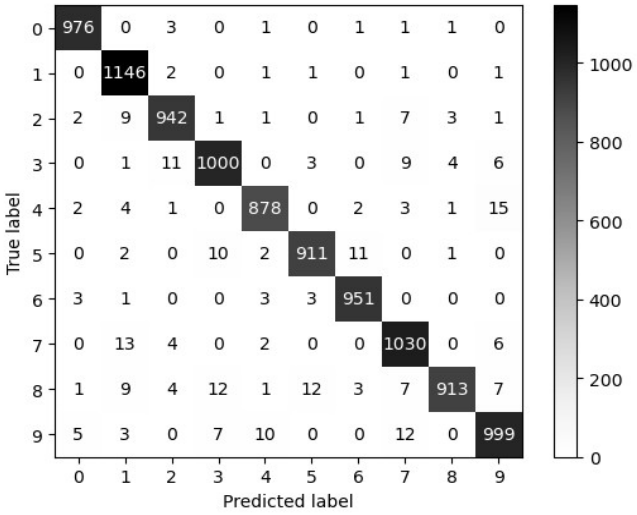
4.2.1 Accuuracy score

Utvärderingen av ensemblemodellen på testdatan visade en accuracy score på 0.9746. Eftersom accuracy score på validerings- och testdatan är mycket nära varandra, tyder det på att modellen generaliserar väl till osedd data. Den lilla skillnaden indikerar att det inte finns någon betydande överanpassning, vilket innebär att modellen har lärt sig mönster som gäller även utanför träningsdatan.

4.2.2 Confusion Matrix

Den följande confusion matrix visar modellens klassificeringsresultat på tidigare osedd data. Confusion matrixen visar hur modellen klassificerar siffrorna 0–9. Raderna representerar de faktiska etiketterna och kolumnerna modellens förutsägelser. De mörka rutorna längs diagonalen (nära 1000) visar att modellen oftast gissar rätt. De ljusare rutorna utanför diagonalen har låga värden (0–15), vilket betyder att få siffror blandas ihop. Matrisen tyder på att modellen har mycket hög noggrannhet.

För att analysera modellens felklassificeringsmönster undersökte jag hur många gånger varje siffra klassificerades felaktigt. Tabell 2 visar det totala antalet felklassificerade prover per siffra. De högsta felnivåerna uppstod för siffra 8 (56 fel) och siffra 9 (37 fel), medan siffra 0 (7 fel) och siffra 1 (6 fel) hade de lägsta felen. Detta tyder på att modellen har svårigheter med visuellt liknande siffror, såsom 8, 9 och 3.



Figur 1: Confusion Matrix

Siffra	Felklassificerade prover
0	7
1	6
2	25
3	34
4	28
5	26
6	10
7	25
8	56
9	37
Total	254

Tabell 2: total antal felklassificerade prover per siffra

4.2.3 Classification report

Tabellen 3 visar modellens klassificeringsresultat på testdatan, inklusive precision, recall, F1-score och support för varje siffra (0–9). Precision visar hur många av modellens förutsägelser som var korrekta, medan recall visar hur många av de faktiska instanserna som identifierades korrekt. F1-score balanserar dessa två mått. Support anger hur många sanna exempel som finns för varje klass.

Modellen uppnådde en total noggrannhet på 97 %, vilket innebär att 97 % av siffrorna i testdatan klassificerades korrekt. Både macro average (0,98 / 0,97 / 0,97) och weighted average (0,97 / 0,97 / 0,97) ligger nära varandra, vilket visar att klassfördelningen är relativt balanserad och att modellen generaliserar väl över alla klasser.
Modellen presterade bäst på siffrorna 0, 6 och 8, som hade högst precision och F1-score (kring 0,99), vilket tyder på mycket tillförlitlig klassificering. Siffrorna 8 och 9 hade något lägre recall (0,94 respektive 0,96), vilket innebär att dessa siffror ibland förväxlades med andra. Intressant nog hade siffran 1 en mycket hög recall (0,99), vilket innebär att nästan alla ettor identifierades korrekt, men precisionen (0,96) antyder att modellen ibland felaktigt klassificerade andra siffror som 1.

Digit	Precision	Recall	F1-score	Support
0	0.99	0.99	0.99	983
1	0.96	0.99	0.98	1152
2	0.97	0.97	0.97	967
3	0.97	0.97	0.97	1034
4	0.98	0.97	0.97	906
5	0.98	0.97	0.98	937
6	0.98	0.99	0.99	961
7	0.96	0.98	0.97	1055
8	0.99	0.94	0.97	969
9	0.97	0.96	0.96	1036
Accuracy			0.97	10000
Macro Avg	0.98	0.97	0.97	10000
Weighted Avg	0.97	0.97	0.97	10000

Tabell 3: : Klassificeringsrapport för modellen

4.2.4 Utvärdering på Egna Skannade Siffror

Resultatet visade på lägre noggrannhet än med MNIST-testdatan, vilket tyder på att modellen har svårare att hantera handskrivna siffror i verkligheten.

Även om både MNIST-datan och mina skannade handskrivna siffror normaliserades på samma sätt, presterade modellen dåligt på det skannade datasetet. Detta kan bero på skillnader i stil, placering och brus som introducerats vid skanningen. MNIST-siffror är centrerade, gråskaliga och rena, medan skannade siffror kan vara något snedställda, ojämnt belysta eller innehålla artefakter från papperets struktur eller skuggor. Dessa subtila variationer, tillsammans med det lilla testunderlaget (80 exempel), påverkade sannolikt modellens förmåga att generalisera.

5 Slutsatser

I denna rapport har jag använt maskininlärning för att modellera och klassificera MNIST-datasetet. Jag jämförde fem olika modeller Logistisk Regression, Linjär SVC, k-NN, Random Forest och Extra Trees och utvärderade deras prestanda baserat på accuracy.
Efter att ha identifierat k-NN och Extra Trees som de bäst presterande modellerna, finjusterade jag deras hyperparametrar och skapade en ensemblemodell. Denna ensemble uppnådde högsta noggrannhet (97,42 %), vilket visar att kombinationen av dessa modeller förbättrade klassificeringen.
Vid analys av felklassificeringar framkom att vissa siffror, särskilt 8 och 9, var mer benägna att bli felklassificerade. Detta kan bero på deras visuella likhet med andra siffror. Samtidigt presterade siffrorna 0, 6 och 8 bäst i precision och F1-score, vilket visar på stark modellprestanda för dessa klasser.
Även om jag initialt valde de bästa modellerna baserat på accuracy, genomförde jag en djupare analys efter hyperparameteroptimering med hjälp av confusion matrix och klassificeringsrapporter. Detta gav en mer detaljerad bild av hur modellerna presterade på enskilda siffror och bekräftade att ensemblemetoden gav en bättre balans mellan precision och recall.

Resultaten visar därmed vilka modeller som presterar bäst på MNIST, hur optimering förbättrar prestandan, samt vilka styrkor och svagheter varje modell uppvisar vid klassificering av handskrivna siffror.

6 Teoretiska frågor

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

Träningsdata används för att skapa modeller, träna dem och lära dem de mönster som finns i datan. Efter detta steg måste vi välja den bästa modellen. Vi utvärderar därför alla modeller med hjälp av valideringsdata och väljer den bästa modellen baserat på ett lämpligt mått. När vi har valt den bästa modellen, tränar vi om den på både träningsdata och valideringsdata, och därefter testar vi den på testdatan för att få en uppskattning av modellens generaliseringsförmåga på helt ny data eller för att beräkna out-of-sample error (3).

6.2 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”?

Eftersom Julia har delat upp datan i träningsdata och testdata behöver hon använda en annan metod. K-fold cross-validering är en lämplig metod här.

I denna metod, efter att träningsdatan slumpmässigt har blandats, delas den upp i K delar (vanligtvis 5 eller 10). I varje iteration tränas modellen på K-1 delar av datan och valideras på den återstående delen. Detta upprepas K gånger, där varje del används som valideringsdata en gång.

Efter varje iteration får vi ett mått, och när alla iterationer är klara beräknas medelvärdet av dessa mått. Vi kan sedan jämföra detta medelvärde med andra modellers medelvärden och välja den bästa modellen för att testa i nästa steg (3).

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

Ett regressionsproblem innebär att den beroende variabeln (y) är kontinuerliga värde. Exempelvis kan man prediktera inkomst baserat på ålder, utbildning och erfarenhet. Tillämpningar finns inom fastighetsvärdering, ekonomi och hälsa. Vanliga modeller inkluderar linjär regression, lasso regression, Support Vector Machines (SVM och Beslutsträd (4).

6.4 kan du tolka RMSE och vad används det till: $RMSE$?

I ett regressionsproblem är root mean squared error (RMSE) ett av de vanligaste måtten för att utvärdera en modell på valideringsdata eller testdata. RMSE kan vi tolka som våra prediktioners medelavstånd till de sanna värdena. En lägre RMSE indikerar en bättre modell**,** medan en hög RMSE tyder på större fel i prediktionerna (4).

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

Ett klassificeringsproblem innebär att den beroende variabeln (y) har två eller flera klasser. Exempel på modeller är logistisk regression, beslutsträd, Random Forest och SVM. Vanliga tillämpningar inkluderar spamdetektering, kundbortfall (churn), och sjukdomsprognoser. Confusion Matrix är en metod för att utvärdera klassificeringsmodeller. Den visar hur många förutsägelser som var korrekta och felaktiga (5).

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

K-means är en algoritm som används för klustring inom unsupervised machine learning. Med hjälp av K-means bestämmer vi ett värde för K, vilket anger antalet kluster. Algoritmen hittar sedan klustercentra (centroid observationer) och tilldelar varje datapunkt till det närmaste klustercentrumet. Processen upprepas tills klustren stabiliseras (6).

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

För att använda kategoriska data i maskininlärning behöver de omvandlas till numeriska värden:

Med ordinal encoding får varje kategori ett tal baserat på en viss ordning, t.ex. Liten = 0, Medium = 1, Stor = 2.

Vid one-hot encoding skapas en ny kolumn för varje kategori, där endast en har värdet 1 och resten 0. Exempelvis blir "Medium" till [0, 1, 0].

Dummy variable encoding liknar one-hot men tar bort en kolumn för att undvika multikollinaritet, t.ex. representeras "Stor" då som [0, 0] om den är referenskategorin.

6.8 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?

Göran har rätt, eftersom ordinal data har en tydlig ordning (t.ex. skolbetyg, utbildningsnivå), medan nominal data saknar inbördes ordning (t.ex. färger, kön, djurarter). Julias argument bygger på en subjektiv tolkning och det beror på individuella åsikter. Färger har ingen naturlig rangordning, så de förblir nominal data oavsett hur någon tolkar dem (7).

6.9 Kolla följande video om Streamlit: https://www.youtube.com/watch?v=ggDa RzPP7A&list=PLgzaMbMPEHEx9Als3F3sKKXexWnyEKH45&index=12 Och besvara följande fråga: - Vad är Streamlit för något och vad kan det användas till?

Streamlit är ett open source framwork för att enkelt skapa interaktiva dataapplikationer i Python, främst för maskininlärning och dataanalys (8).

Självutvärdering

1. Utmaningar du haft under arbetet samt hur du hanterat dem.
   En utmaning var att få Streamlit-applikationen att fungera online. Lokalt fungerade allt bra, men i molnet uppstod problem med installation av vissa paket, särskilt opencv-python. Trots flera försök fick jag det inte att fungera online, så jag fokuserade på att köra applikationen lokalt istället

2. Vilket betyg du anser att du skall ha och varför. Jag anser att jag förtjänar betyget *VG*.
   Jag testade fem olika modeller och skapade sedan en ensemblemodell som uppnådde en hög noggrannhet på 97,4 %. Jag tillämpade så mycket som möjligt av de teoretiska kunskaperna och dokumenterade allt i detalj i min rapport. Efter att modellen var färdig testade jag den även på två olika sätt för att utvärdera hur den presterar på verklig och helt okänd data. Jag följde arbetsflödet för maskininlärningsprojekt för att säkerställa en korrekt och strukturerad process genom alla steg.

3. Något du vill lyfta fram till Antonio?
   Tack för en mycket givande kurs! Jag var ganska förvirrad under den första veckan, men nu i slutet av kursen känner jag att jag har lärt mig väldigt mycket på bara sex veckor. Att skriva en rapport var ett riktigt bra inslag , det hjälpte mig att arbeta mer strukturerat med projektet och gav mig en djupare förståelse för de teoretiska koncepten.

Källförteckning

1. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed., Chapter 3, p. 100). O'Reilly Media.

2. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed., Chapter 3, p. 108). O'Reilly Media.

3. *Introduktion till Maskininlärning* [Video]. YouTube. https://www.youtube.com/watch?v=zORv5vxrwok

4. *Linjär Regression & Logistisk Regression* [Video]. YouTube. https://www.youtube.com/watch?v=0f1tWgarTkQ

5. *Klassificering* [Video]. YouTube. https://www.youtube.com/watch?v=-QzFLifZgNw

6. Kapitel 9: Unsupervised Learning Techniques – Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow [Video]. YouTube. https://www.youtube.com/watch?v=Pj-eAU1hrGk

7. Shedden, K. (n.d.). *Types of data*. University of Michigan, Department of Statistics. https://dept.stat.lsa.umich.edu/~kshedden/introds/topics/types_of_data/

8. *Supervised learning - Decision Trees* [Video]. YouTube. https://www.youtube.com/watch?v=ggDa-RzPP7A