Utveckling och distribution av Streamlit-applikation för klassificering av handskrivna siffror med XGBoost

Amanda Sumner

EC Utbildning

Machine learning kunskapskontroll 2

2025/03

Abstract

This project explores handwritten digit recognition using machine learning techniques, specifically focusing on the MNIST dataset. Three tree-based models, Random Forest, Extra Trees, and XGBoost, were evaluated, with XGBoost demonstrating the highest accuracy. Hyperparameter optimization was conducted to enhance model performance, achieving an accuracy of 0.981. A Streamlit web application was developed and deployed, enabling users to draw digits and receive real-time predictions. User feedback indicates variable prediction accuracy depending on handwriting clarity. This project demonstrates the practical application of machine learning in digit recognition and highlights the effectiveness of XGBoost for this task.

Innehållsförteckning

Abstract **0**
Inledning 3
Teori 4
Trädbaserade Modeller 4
Random Forest Classifier 4
Extra Trees Classifier 4
XGBoost Classifier 4
Hyperparametrar i XGBoost Classifier 5
Learning task parametrar 5
Tree Booster parametrar 5
Data 6
Metod 7
Datainsamling och förbearbetning 7
Modellval och träning 7
Hyperparameter optimering med RandomizedSearchCV 8
Slutlig modellträning och utvärdering 10
Utveckling av Streamlit-applikation 10
Resultat 12
Slutsatser 14
Teoretiska frågor 15
Självutvärdering 17
Appendix A: MNIST dataset beskrivning 19
Källförteckning 21

Inledning

Denna rapport beskriver ett maskininlärningsprojekt med fokus på igenkänning av handskrivna siffror, ett klassiskt problem inom mönsterigenkänning. Projektets huvudmål var att utveckla och implementera en modell som, med hjälp av MNIST-datasetet, kan göra noggranna förutsägelser av handskrivna siffror. Projektet, som ursprungligen var en kursuppgift, gav värdefull praktisk erfarenhet av tillämpning av maskininlärningsprinciper, förbättring av Python-färdigheter samt användning av Streamlit för distribution av webbapplikationer.

MNIST-datasetet, specifikt datasetet mnist_784 från openml.org, användes för att träna och utvärdera modellerna. Jag utforskade tre trädbaserade modeller: Random Forest, Extra Trees och XGBoost. Efter initial träning och utvärdering visade XGBoost den högsta accuracy score och valdes som den slutliga modellen. Hyperparameteroptimering utfördes för att maximera modellens prediktiva prestanda.

Den optimerade XGBoost-modellen uppnådde en accuracy score på 0.981. För att demonstrera modellens praktiska tillämpning utvecklades en Streamlit-webbapplikation och distribuerades på Streamlit Cloud. Denna applikation tillåter användare att rita siffror på en canvas och få förutsägelser i realtid från den tränade modellen. Användarfeedback indikerar rimligt god noggrannhet, även om prestandan varierar beroende på handskriftens tydlighet, där vissa siffror visar sig vara svårare att klassificera än andra.

Denna rapport kommer att fortsätta med en detaljerad beskrivning av den metodik som användes, inklusive dataförbehandling, modellval, hyperparameterjustering och applikationsutveckling. Jag kommer sedan att presentera och diskutera resultaten, följt av slutsatser och potentiella områden för framtida utforskning.

Rapporten skrevs ursprungligen på engelska och översattes till svenska med hjälp av Google Translate och Gemini.

Teori

Trädbaserade Modeller

Trädbaserade modeller använder en trädliknande struktur för att fatta beslut. Modellen startar med en rotnod och grenar ut i noder som representerar feature-baserade beslut, och slutar i löv som förutsäger class labels eller värden. Trädets visuella struktur underlättar förståelsen av förutsägelser. Dessa modeller hanterar kategoriska och numeriska funktioner utan omfattande förbehandling och använda komplexa olinjära relationer mellan funktioner och målvariabeln. Exempel inkluderar decision trees, random forests och gradient boosting.[1]

Random Forest Classifier

Random Forest kombinerar flera beslutsträd för förbättrad förutsägningsnoggrannhet. För klassificering förutsäger den kategoriska resultat. Varje träd tränas på en bootstrapped data subset och en slumpmässig feature subset. Träden byggs genom att rekursivt dela data och välja den bästa feature från subset vid varje delning. Förutsägelser aggregeras genom majoritetsröstning. Random Forest hanterar stora, högdimensionella och kategoriska data, och minskar overfitting.[2]

Extra Trees Classifier

Extra Trees aggregerar de-korrelerade beslutsträd för klassificering. Träden byggs från träningsdata och delar noder med slumpmässigt valda features. Feature val använder Gini Importance, rangordnar features efter normaliserad Gini Index reduktion. De top k features väljs.[3]

XGBoost Classifier

XGBoost (Extreme Gradient Boosting) bygger beslutsträd sekventiellt och korrigerar tidigare trädfel för att minimera en förlustfunktion. Den använder gradientnedstigning för parameteroptimering och regularisering för att förhindra overfitting. Den hanterar varierande

---

[1] https://www.geeksforgeeks.org/comprehensive-guide-to-tree-based-models-for-classification-in-python/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

data och uppgifter (regression, klassificering, rangordning). XGBoost är beräkningseffektiv och använder parallell bearbetning för stora datamängder.[4]

Hyperparametrar i XGBoost Classifier[5]

Learning task parametrar

`objective: multi:softmax`, ställ in XGBoost att göra multiclass klassificering med hjälp av softmax-målet, behöver också ställa in num_class (antal klasser).

`eval_metric: mlogloss`, utvärderingsmått för valideringsdata, i detta fall, multiclass logloss. Detta är förlustfunktionen som används i multinomial logistisk regression, definierad som den negativa logistiska sannolikheten för en logistisk modell som returnerar (y_pred) sannolikheter för sina träningsdata (y_true).

`early_stopping_rounds`: Om det finns en validering dataset kan early stopping användas för att hitta det optimala antalet boosting ronder. Early stopping kräver eval_set från valideringsdata. Modellen kommer att tränas tills validerings score slutar förbättras. Valideringsfel måste minska minst varje early_stopping_rounds för att fortsätta träningen.

Tree Booster parametrar

`learning_rate` (default=0.3, range: [0,1]): ändrar hur mycket varje träd bidrar till den slutliga förutsägelsen. Medan det behövs fler träd leder mindre värden ofta till mer exakta modeller.

`max_depth` (default=6, range: [0,∞]): parametern styr djupet på varje träd, undviker overfitting och kontrollerar modellens komplexitet.

`subsample` (default=1, range: (0,1]): hanterar procentandelen data som tas slumpmässigt för att växa varje träd, vilket minskar variansen och förbättrar generaliseringen. Att ställa in den på 0,5 innebär att XGBoost slumpmässigt tar hälften av träningsdatan innan träd växer. Om värden är för lågt kan det leda till underfitting.

`colsample_bytree` (default=1, range (0, 1]): ställer procentandelen features som kommer att tas slumpmässigt för att odla varje träd.

[4] https://xgboost.readthedocs.io/en/stable/index.html
[5] https://xgboost.readthedocs.io/en/stable/parameter.html

Data

För detta projekt krävdes användningen av mnist_784 dataset från openml.org[6]. Detta är en del av en större datamängd från NIST, som innehåller handskrivna siffror. Denna dataset består av 70 000 gråskalebilder av handskrivna siffror (0-9), var och en representerad som en 784-dimensionell vektor, motsvarande de tillplattade bilderna på 28x28 pixlar. Varje pixelvärde sträcker sig från 0 till 255. Se Appendix A för den fullständiga originala beskrivningen av MNIST datasetet. Följande är ett exempel på en slumpmässigt vald image för varje label från 0 till 9 (figure 1) och distribution av labels (figure 2).

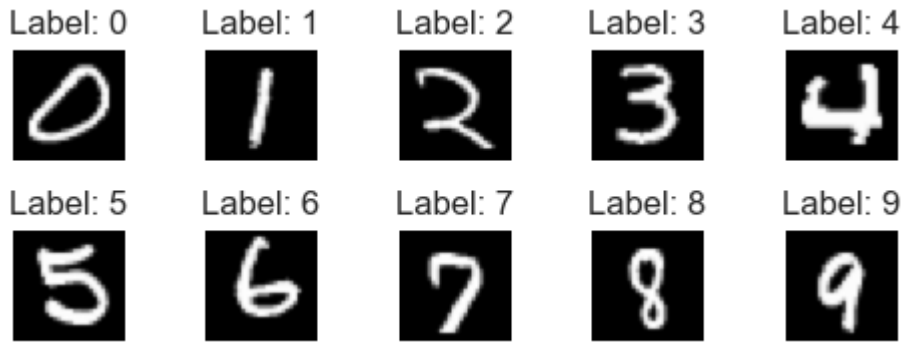

Figure 1. Exempel av siffror för varje label i MNIST datasetet

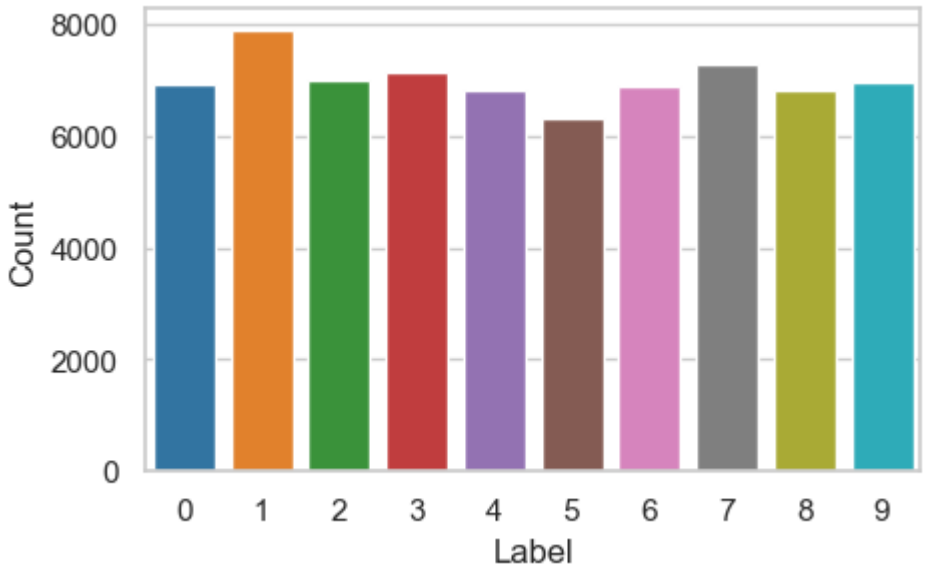

Figure 2. Distribution av labels i MNIST-datasetet

[6] https://openml.org/search?type=data&status=active&id=554

Metod

Datainsamling och förbearbetning

Datasetet för projektet hämtades med funktionen `fetch_openml` från `sklearn.datasets` library. Detta dataset består av 70 000 gråskalebilder av handskrivna siffror (0-9), där varje bild representeras som en 784-dimensionell vektor, motsvarande de utplattade 28x28 pixelbilderna. Varje pixelvärde varierar från 0 till 255.

För att förbereda datan för modellträning och utvärdering laddades datasetet initialt som en numpy array och delades upp i `features` (X) och `labels` (y). Features normaliserades sedan genom att dividera varje pixelvärde med 255, vilket skalar dem till intervallet [0, 1]. Detta normaliseringssteg är viktigt för att förbättra prestandan hos maskininlärnings algoritmer.

```
mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)
X = mnist.data
y = mnist.target.astype(np.uint8)
# Flatten the images
X = X.reshape(X.shape[0], 784)
# Normalize pixel values
X = X.astype('float32') / 255.0
```

Efter normaliseringen delades datasetet upp i tränings- och testset med hjälp av funktionen `train_test_split` från `sklearn.model_selection`. En testsetstorlek på 20% valdes, vilket resulterade i en 80/20-delning. Dessutom skapades ett valideringsset från träningsdatan, återigen med hjälp av `train_test_split`, för att underlätta hyperparameterjustering. En 20% delning av träningsdatan användes för validering, vilket lämnade 60% av den ursprungliga datan för träning.

```
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.1, random_state=26)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.1, random_state=26)
```

Modellval och träning

Detta projekt utforskade tre trädbaserade maskininlärningsmodeller: Random Forest, Extra Trees och XGBoost. Dessa modeller valdes på grund av deras bevisade effektivitet i klassificeringsuppgifter. Varje modell instansierades med standardhyperparametrar och tränades på träningssetet (X_train, y_train). Prestandan för varje modell utvärderades på valideringssetet (X_val, y_val) med hjälp av accuracy score.

```
rfc = RandomForestClassifier(n_estimators=100, random_state=26)
etc = ExtraTreesClassifier(n_estimators=100, random_state=26)
xgb = xgboost.XGBClassifier(random_state=26)

rfc.fit(X_train, y_train)
y_pred_rfc = rfc.predict(X_val)
accuracy_rfc = accuracy_score(y_val, y_pred_rfc)

etc.fit(X_train, y_train)
y_pred_etc = etc.predict(X_val)
accuracy_etc = accuracy_score(y_val, y_pred_etc)

xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_val)
accuracy_xgb = accuracy_score(y_val, y_pred_xgb)
```

De initiala accuracy scores för de tre modellerna var följande:

```
Random Forest: 0.9657
Extra Trees: 0.9671
XGBoost 0.9743
```

Baserat på de initiala utvärderingsresultaten visade XGBoost den högsta noggrannheten på valideringssetet. Följaktligen valdes XGBoost som den slutliga modellen för vidare optimering och distribution.

Hyperparameter optimering med RandomizedSearchCV

För att ytterligare förbättra prestandan för den valda XGBoost-modellen utfördes hyperparameteroptimering med `RandomizedSearchCV`[7] från `scikit-learn` library. Denna teknik möjliggör effektiv utforskning av ett fördefinierat hyperparametersökutrymme genom slumpmässigt urval av kombinationer av parametrar.

För att minska modellstorleken och bearbetningstiden valde jag att endast använda hyperparametrarna som är kända för att ha en betydande inverkan på XGBoost-modellens prestanda: `learning_rate`, `max_depth`, `subsample` och `colsample_bytree`. `RandomizedSearchCV` kördes med `eval_metric=mlogloss`, `early_stopping_rounds=10`, `cv=3`, `n_iter=30` och 3000 estimators, på en cuda device. Evalueringsset bestående av valideringsdata användes med `early_stopping_rounds`. `best_params_` attributet för `RandomizedSearchCV` objektet användes för att hämta de optimala hyperparametervärdena som hittades under sökningen.

---

[7] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

```
RandomizedSearchCV(cv=3,
                   estimator=XGBClassifier(base_score=None,
                         booster=None,
                         callbacks=None,
                         colsample_bylevel=None,
                         colsample_bynode=None,
                         colsample_bytree=None,
                         device='cuda',
                         early_stopping_rounds=10,
                         enable_categorical=False,
                         eval_metric='mlogloss',
                         feature_types=None, gamma=None,
                         grow_policy=None,
                         importance_type=None,
                         interaction_constraints=None,
                         learning_rate=np.float64(0.060000000000000005),
                         max_bin=None,
                         max_cat_threshold=None,
                         max_cat_to_onehot=None,
                         max_delta_step=None,
                         max_depth=np.int64(5),
                         max_leaves=None,
                         min_child_weight=None,
                         missing=nan,
                         monotone_constraints=None
                         multi_strategy=None,
                         n_estimators=3000,
                         n_jobs=None,
                         num_class=10,
                         num_parallel_tree=None, ...),
                   n_iter=30,
                   param_distributions={'colsample_bytree': array([0.7 ,
                       0.75, 0.8 , 0.85,
                       0.9 , 0.95, 1.  ]),
                      'learning_rate': array([0.01, 0.02, 0.03, 0.04, 0.05,
                       0.06, 0.07, 0.08, 0.09]),
                      'max_depth': array([3, 4, 5, 6]),
                      'subsample': array([0.7 , 0.75, 0.8 , 0.85, 0.9 ,
                       0.95, 1.  ])},
                   random_state=26, verbose=2)

search.fit(X_train, y_train, eval_set=[(X_val, y_val)])

search.best_params_
```

Slutlig modellträning och utvärdering

Med hjälp av de optimala hyperparametrarna som erhållits från `RandomizedSearchCV` tränades den slutliga XGBoost-modellen på de kombinerade tränings- och

valideringsdatasetet. Denna metod maximerar mängden data som används för träning, vilket leder till förbättrad modellprestanda. `early_stopping_rounds` var inte nödvändigt för det och CPU användes som enhet för att undvika missmatchning när modellen användes i den distribuerade webbappen.

```
best_model = search.best_estimator_
best_model.set_params(early_stopping_rounds=None, device='cpu')
final_model = best_model.fit(X_train_full, y_train_full)
```

Den slutliga modellens prestanda utvärderades på testdatasetet (X_test, y_test) med hjälp av accuracy score. Den slutliga accuracy score representerar modellens generaliserings prestanda på osynliga data.

```
y_pred_final = final_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_final)
print(f"Final model accuracy: {accuracy:.4f}")
Final model accuracy: 0.9810
```

Sedan exporterades modellen med `joblib` för att användas i webappen.

Utveckling av Streamlit-applikation

För att demonstrera den praktiska tillämpningen av den tränade XGBoost-modellen utvecklades en Streamlit-webbapplikation. Denna applikation tillåter användare att rita handskrivna siffror på en canvas och få förutsägelser i realtid från modellen.

Streamlit-applikationen designades för att:

- Visa en ritcanvas för användarinmatning. Canvaskomponenten som används i appen är `drawable_canvas` av [andfanilo (Fanilo Andrianasolo) · GitHub](https://github.com/andfanilo).
- Förbehandla den användarritade bilden för att matcha modellens indataformat: konvertera till gråskala, ändra storlek till 28x28 pixlar, invertera, normalisera genom att dividera med 255, flatten genom att omforma till en endimensionell array.
- Använda den tränade XGBoost-modellen för att förutsäga siffran.
- Visa förutsägelsen för användaren.

Streamlit-applikationen distribuerades på Streamlit Cloud, vilket gör den tillgänglig via en webbläsare: https://ml-kk2.streamlit.app/ (figure 3).

```
import streamlit as st
from streamlit_drawable_canvas import st_canvas
from PIL import Image, ImageOps
import numpy as np
```

```
import joblib

model = joblib.load("model.joblib")
canvas_result = st_canvas(
    stroke_width=20,
    stroke_color="#000000",
    background_color="#FFFFFF",
    display_toolbar=True,
    height=200,
    width=200
)
img = Image.fromarray(canvas_result.image_data.astype('uint8'), 'RGBA')
gray_image = img.convert('L') #Convert to grayscale
resized_image = gray_image.resize((28, 28), Image.Resampling.LANCZOS)
#Resize
inverted_image = ImageOps.invert(resized_image) # Invert
normalized_image = np.array(inverted_image).astype('float32') / 255.0
#Normalize
flattened_image = normalized_image.reshape(1, 784)
prediction = model.predict(flattened_image)
```

Figure 3. Streamlit webbappen i aktion.

Resultat

Accuracy score för XGBoost-modellen efter hyperparameteroptimering och slutlig träning var 0,9810 vilket är högt. Accuracy score ökade i jämförelse med den ursprungliga modellens prestanda med 0,0067.

De bästa hyperparametrarna som hittades med `RandomizedSearchCV` var:

```
           Parameter  Value
0          subsample   0.75
1          max_depth   5.00
2      learning_rate   0.06
3  colsample_bytree   0.90
```

Confusion matrix (figure 4) visar antalet klassificeringsfel och korrekt förutsagda labels. Följande klassificeringsrapport beskriver precision, recall, f1 score för varje label, och support som är antalet förekomster av varje label i y_true.

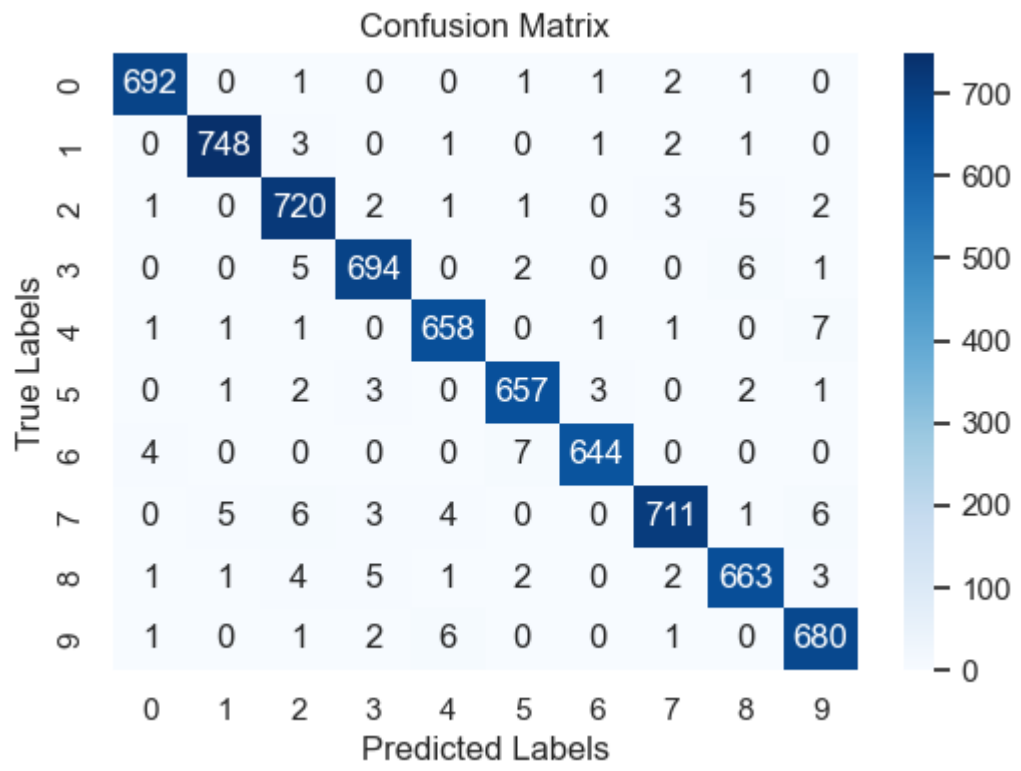

Figure 4. Confusion matrix

```
Classification report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       698
           1       0.99      0.99      0.99       756
           2       0.97      0.98      0.97       735
           3       0.98      0.98      0.98       708
           4       0.98      0.98      0.98       670
           5       0.98      0.98      0.98       669
           6       0.99      0.98      0.99       655
           7       0.98      0.97      0.98       736
           8       0.98      0.97      0.97       682
           9       0.97      0.98      0.98       691

    accuracy                           0.98      7000
   macro avg       0.98      0.98      0.98      7000
weighted avg       0.98      0.98      0.98      7000
```

Implementeringen av Streamlit-applikationen hindrades initialt av dependencies som måste ändras från `cv2` till `pillow` och från `pickle` till `joblib`, för att kunna distribuera den på Streamlit Cloud. När webbapplikationen väl fungerade rapporterade användarna varierande framgångar med handskriftsigenkänning. Framgången för igenkänningen verkade bero på hur nära användarens handstil var bilderna i träningsdatasetet. Särskilt siffrorna 4, 6, 7 och 9 fungerade minst, men siffrorna 2, 3 och 5 förutsades nästan alltid korrekt (figure 5). Andra kommentarer från användare var att det var svårt att rita på canvas med mus eller finger och att det inte var som deras vanliga handstil. Flera användare rapporterade 100% noggrannhet, men andra fick bara ungefär hälften korrekt.

Figure 5. Exempel på korrekt och felaktigt förutsagda siffror

Slutsatser

Genom att använda XGBoost-modellen och MNIST-datasetet, uppnådde projektet en noggrannhet på 98,1% i igenkänning av handskrivna siffror. Detta visar att modellen effektivt kan lära sig och känna igen olika handskriftsmönster.

En Streamlit-webbapplikation utvecklades för att användare skulle kunna testa modellen i realtid. Trots modellens höga noggrannhet, indikerade feedback från användare att det fanns utrymme för förbättringar, särskilt när det gäller användargränssnitt och prestanda.

Modellens noggrannhet begränsades av den ursprungliga datamängden som baserades på två grupper av amerikanska försökspersoner. Resultaten skulle kunna förbättras genom en träningsdataset med fler handstilar från hela världen. Jag skulle föreslå att man skapar en sådan uppdaterad dataset. Andra maskininlärningsmodeller bör utforskas för att hitta om det finns en som ger ännu mer exakta resultat.

Teoretiska frågor

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

Träningsdataset används för att träna modeller. Valideringsdataset används för att köra score eller RMSE och bestämma vilket modell som är bäst och sedan kombinera med träningsdataset och träna den bästa modellen på båda. Test dataset används bara för att utvärdera modellen på ny data när allt är klar.

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”?

Julia kan använda cross validation av träningsdata. Det betyder att träningsdataset delas upp i flera subsets (folds), vanligt med K-Fold där K står för antalet folds. Modellen utvärderas på en av dessa folds, efter att den tränats på resten av folds. Processen upprepas med varje fold som används som valideringsset en gång. Då tar Julia en medelvärde av resultaten.

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

Regressionsproblem är ett problem inom maskininlärning där målet är att förutsäga en numerisk värde, i motsats till klassificeringsproblem som förutsäger en kategori av data. Det är som en funktion som tar flera indata variabler (features) och ger motsvarande kontinuerlig data som svaret (target).

Exempel är förutsägelse av huspriser baserat på läge, byggår, storlek osv., eller prognosera sannolikheten att en kund inte kommer att kunna betala tillbaka ett lån.

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

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

RMSE betyder root mean square error. Den mäter den genomsnittliga skillnaden mellan värden som förutsägs av en modell och den faktiska värden i en regressionsmodell. Ett lägre RMSE indikerar att modellen gör mer exakta förutsägelser. RMSE används till regressions problemer.

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

Ett klassificeringsproblem är ett problem med målet att förutsäga vilken kategori data tillhör. Istället för ett kontinuerligt värde som i ett regressionsproblem, förutsäger ett klassificeringsproblem en kategori eller klass. Exemplar är t.ex., om man söker med mobil kamera, resultaten visar liknande objekter baserad på klassificering av objekt i bilden. Klassificering av handskrivna siffror i denna projektet är ett annat exempel.

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

K-means är en klustringsalgoritm som lokaliserar kluster i data och delar data i distinkta grupper baserad på likheter. Skillnaden från klassificeringsproblem är att klassificering använder fördefinierade klasser, men K-means använder unsupervised learning utan några förubestämda klasser. Exempel av k-means användning är för att optimisera leveransvägar genom att klustera kunder.

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 när det finns inbyggd ordning eller rangordning av kategorierna. Till exempel, om kategorierna är “låg”, “medel” och “stor”, ordinal encoding tilldelar dom numeriska värden 1, 2 och 3, om “stor” är bättre än “medel” och “medel” är bätter än “låg”.

One-hot-encoding används när det inte finns någon inbyggd rangordning av kategorierna. Varje kategori får en dummyvariabel som en ny kolumn, när 1 betyder att data tillhör den kategorin och 0 betyder att den inte gör det. Till exempel, tre färger av RGB, röd, grön, blå har ingen rangordning, så one-hot-encoding skapar tre kolumner, 1, 0, 0 för röd, 0, 1, 0 för grön och 0, 0, 1 för blå.

Dummy variable encoding är lika one-hot-encoding med skillnaden att det skapar antal kolumner som är 1 mindre än antal kategorier. Som i exempel övan med färger, dummy variable encoding skapar bara två kolumner, röd och grön, och om både kolumner har värde 0 färgen är blå.

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 at ibland data kan tolkas som ordinal, det beror på kontexten. Det är viktigt att vara medveten om innebörden och symboliken av t.ex., en röd skjorta. Tolkningen kan också vara väldigt subjektiv, vad som är vackrast för en person är det inte för en annan. Andra kategorier kan med största säkerhet vara ordinal.

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

Streamlit är ett Python framework med öppen källkod för att leverera dynamiska dataappar, särskilt inom data science och AI. En app kan byggas och distribueras med några rader python kod. En exempel på en Streamlit app är den som används i mitt projekt.

Självutvärdering

Jag hade väldigt roligt med Streamlit-appen, det var första gången jag skapade en app. Det tog mycket tid att ta reda på hur det skulle distribueras utan fel. Det kördes inte på Streamlit Cloud först, jag försökte distribuera det i Azure Cloud, jag hade problem med det, sedan distribuerade jag det på Heroku där det äntligen fungerade. Efter att jag hade tillgång till Heroku SSH kunde jag hitta problemen med det, fixa det och distribuera det i Streamlit Cloud. Det fick mig att önska att jag fortfarande hade min egen Linux-server.

Den andra stora utmaningen var att skriva denna rapport på svenska. Det slutade med att jag använde Google Translate mycket och kontrollerade att mina tekniska termer var korrekta med Gemini eftersom jag inte litar på att Google Translate får det rätt.

Appendix A: MNIST dataset beskrivning

The original MNIST dataset description:

Author: Yann LeCun, Corinna Cortes, Christopher J.C. Burges

Source: MNIST Website http://yann.lecun.com/exdb/mnist/

> The MNIST database of handwritten digits with 784 features, raw data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the first 60,000 examples, and a test set of 10,000 examples
>
> It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.
>
> With some classification methods (particularly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications. The MNIST database was constructed from NIST's NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the

complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint. SD-1 contains 58,527 digit images written by 500 different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample training set is available.

Källförteckning

DataCamp, (Accessed: March 2025). Supervised Learning with scikit-learn. Available at: https://app.datacamp.com/learn/courses/supervised-learning-with-scikit-learn

EducationTopicsExplained (2024). Maskininlärning [YouTube playlist]. Available at: https://youtube.com/playlist?list=PLgzaMbMPEHEx9Als3F3sKKXexWnyEKH45&si=dTAF6qe9lUJwupZA (Accessed: March 2025).

GeeksforGeeks, (Accessed: March 2025). Comprehensive Guide to Tree-Based Models for Classification in Python. Available at: https://www.geeksforgeeks.org/comprehensive-guide-to-tree-based-models-for-classification-in-python/

Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd edn. Sebastopol, CA: O'Reilly Media.

scikit-learn.org, (Accessed: March 2025). scikit-learn: Machine Learning in Python. Available at: https://scikit-learn.org

xgboost.readthedocs.io, (Accessed: March 2025). XGBoost Documentation. Available at: https://xgboost.readthedocs.io