MASKININLÄRNING

Instruktör: Antonio Prgomet
Studerande: Rianna Aalto

I. Introduktion

Bakgrund

Maskininlärning är ett område inom artificiell intelligens som handlar om att skapa datoralgoritmer och statistiska modeller som kan lära och förbättra sig från data utan att behöva programmeras explicit.

I samband med maskininlärning kommer jag utföra ett end-to-end maskininlärningsprojekt på MNIST-datasetet och redovisa hur man kan träna och utvärdera modeller. MNIST-datasetet innehåller 70 000 handskrivna sifferbilder (28x28 pixlar) från 0 till 9, och målet är att träna en modell som kan känna igen vilken siffra som representeras på varje bild.

Frågeställning

Frågeställningen som jag kommer behandla i denna rapport är ”Hur bra prediktionsförmåga kan vi uppnå på MNIST-datan?”. Denna kommer besvaras genom att använda de modeller vi lärt oss i kursen.

II. Databeskrivning / EDA

MNIST-datasetet innehåller 70 000 handskrivna sifferbilder (28x28 pixlar) från 0 till 9, och målet är att träna en modell som kan känna igen vilken siffra som representeras på varje bild.
En 28x28 pixel betyder att en digital bild eller en yta är uppdelad i en kvadratisk matris som är 28 pixlar bred och 28 pixlar hög. Detta betyder att det totalt finns 784 pixlar i bilden eller ytan (28 x 28 = 784). Ju fler pixlar som finns i en bild eller yta, desto högre är dess upplösning och desto mer detaljerad kan bilden eller ytan visas.

Nedan är en bild som jag fick under utforskning av datasetet.

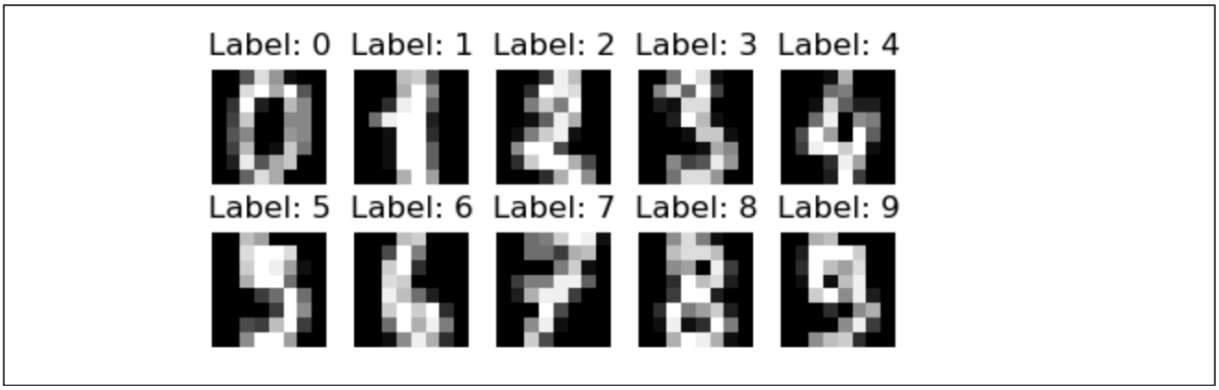


Bild 1:

Här är stegen för ett komplett maskininlärningsprojekt med MNIST-datasetet och modellerna med KNN, SVM och Random Forest som jag tycker att det lämpligt att använda:

1. Beskrivning av problemet:
Problemet är att bygga en modell som kan korrekt klassificera handskrivna siffror från bilder. Målet är att utveckla ett system för automatisk sifferigenkänning för att förbättra effektiviteten i uppgifter som checkhantering eller postsortering. Detta är ett problem inom supervised learning.

2. Hämta data: Laddade MNIST-datasetet med hjälp av Scikit-learns fetch_openml-funktion.

Using MNIST DATASET

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

Download the dataset and set target data to integer

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)

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

3. Utforska data: Visualisera och analysera data med verktyg som Matplotlib och NumPy för att förstå dess egenskaper. Plotta några exempelbilder, kontrollera efter saknade värden eller anomalier och beräkna grundläggande statistik.

check and explore the dataset

```
print(mnist.DESCR)
```

```
**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
**Please cite**:

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
```

```
# Calculate basic statistics
print("Shape:", X.shape)
print("Data type:", X.dtype)
print("Label counts:", np.unique(y, return_counts=True))
```

```
Missing values: False
Shape: (70000, 784)
Data type: float64
Label counts: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([6903, 7877, 6990, 7141, 6824, 6313, 6876,
7293, 6825, 6958]))
```

4. Förbered data: Dela upp datan i tränings-, validerings- och test-set med hjälp av Scikit-Learns train_test_split() funktion. Förbehandla datan genom att scala pixelvärden***.***

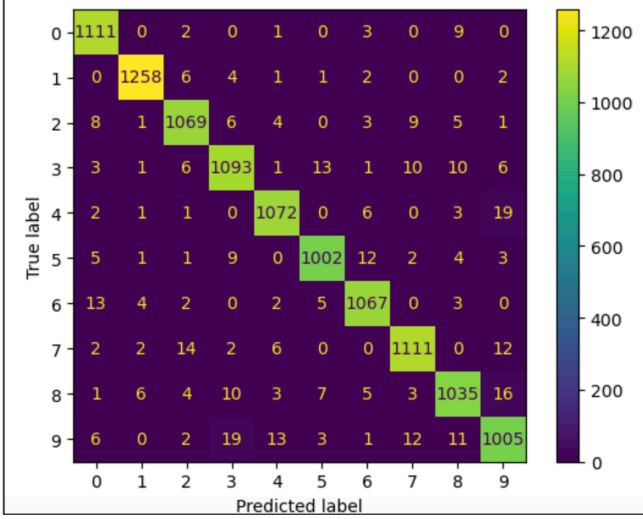
```
# Split the data into training, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)

# Scaling the pixel values
#X_train_scaled = X_train / 255.0
#X_val_scaled = X_val / 255.0
#X_test_scaled = X_test / 255.0

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

5. Välj modeller och träna: Jag har valt Random Forest, K-Nearest Neighbor(KNN) och Support Vector Machine(SVM) och utvärderar dem på training set och validation set.

```
# Train and evaluate a Random Forest model

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)

y_pred_rf = rf.predict(X_val_scaled)
print("Random Forest accuracy on validation set:", accuracy_score(y_val, y_pred_rf))
```

```
Random Forest accuracy on validation set: 0.9663392857142857
```

```
# display confusion Matrix

def display_confusion_matrix(y_val, y_pred_rf):
    cm = confusion_matrix(y_val, y_pred_rf)
    ConfusionMatrixDisplay(cm).plot()

display_confusion_matrix(y_val, y_pred_rf)
```

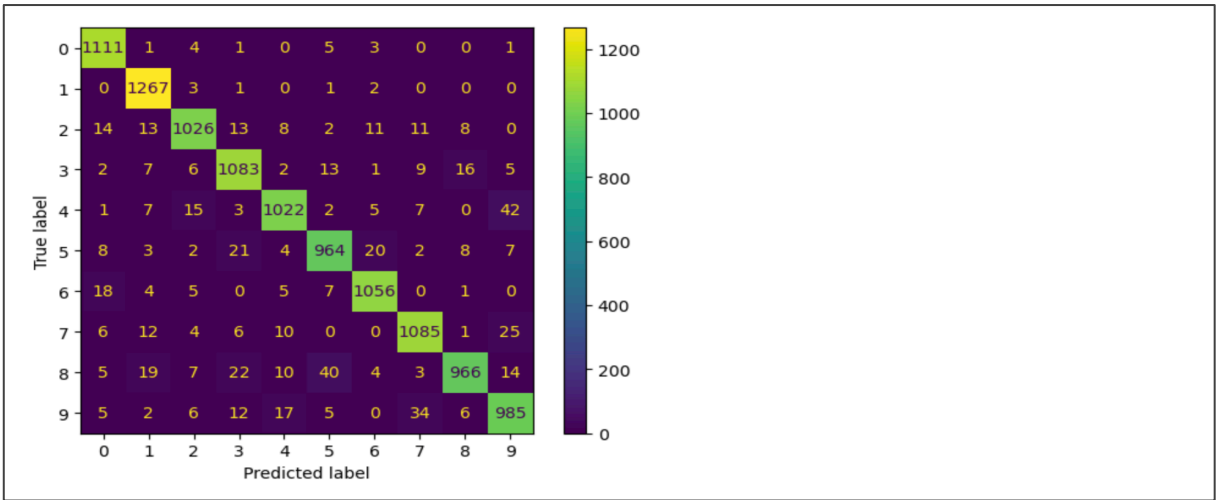


KNN Modell:

```
# Train and evaluate a KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_val_scaled)
print("KNN accuracy on validation set:", accuracy_score(y_val, y_pred_knn))

# display confusion Matrix
display_confusion_matrix(y_val, y_pred_knn)
```

```
/Users/riannaaalto/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:228: FutureWarnin
g: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves t
he axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become Fals
e, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted.
Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

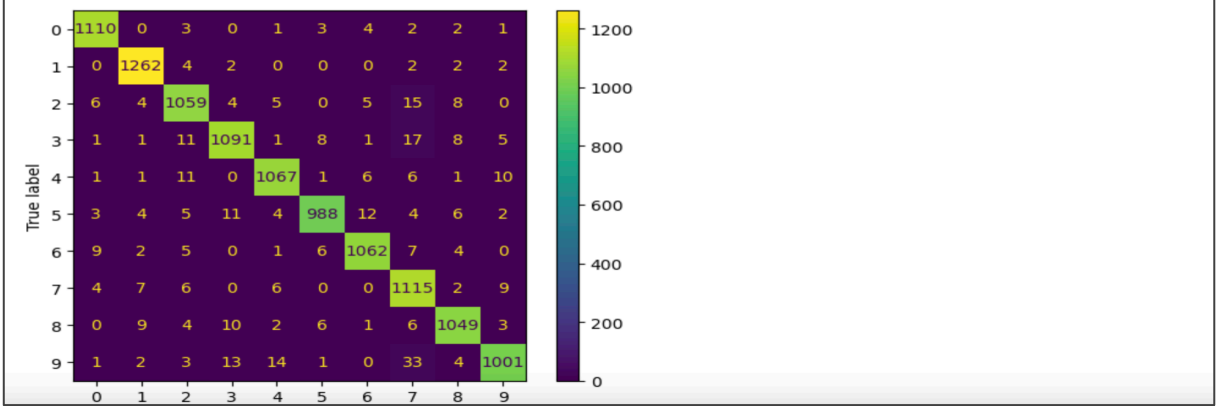
```
KNN accuracy on validation set: 0.9433035714285715
```



Support Vector Machine (SVM) modell:

```
# Train and evaluate an SVM model
svm = SVC(random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_val_scaled)
print("SVM accuracy on validation set:", accuracy_score(y_val, y_pred_svm))

# display confusion Matrix
display_confusion_matrix(y_val, y_pred_svm)
```

```
SVM accuracy on validation set: 0.9646428571428571
```



6. Finjustera utvalda modeller

```
# Define the Random Forest classifier and the hyperparameters to tune
rf_clf = RandomForestClassifier(random_state=42)
param_grid = {'n_estimators': [50, 100, 200, 500], 'max_depth': [10, 20, 30]}

# Define the GridSearchCV object and fit it to the training data
grid_search = GridSearchCV(rf, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [10, 20, 30],
                         'n_estimators': [50, 100, 200, 500]})
```

Hitta bästa estimatorn:

```
#Get the best estimator from the grid search
best_rf = grid_search.best_estimator_
print(f"Best estimator:{best_rf}")
```

```
Best estimator:RandomForestClassifier(max_depth=30, n_estimators=500, random_state=42)
```

Utvärdera prestandan för den bästa estimatorn på validerings- set.

```
y_val_pred = best_rf.predict(X_val_scaled)
accuracy = accuracy_score(y_val, y_val_pred)
print(f"Accuracy on the validation set: {accuracy}")
```

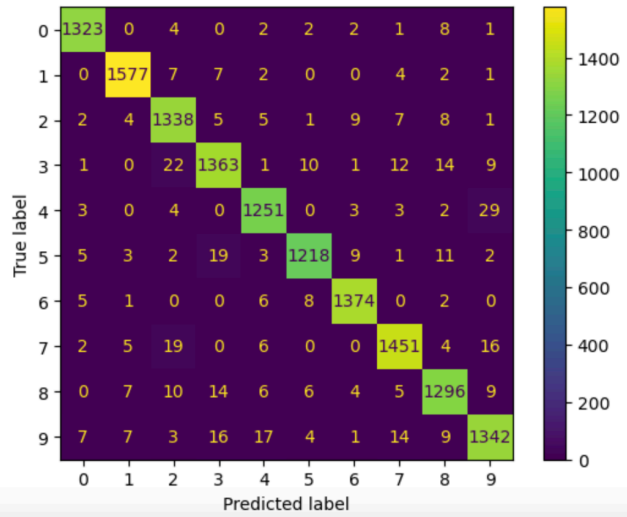
```
Accuracy on the validation set: 0.9685714285714285
```

Utvärdera prestandan för den bästa estimatorn på test- set.

```
y_test_pred = best_rf.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_test_pred)
print(f"Accuracy on the test set: {accuracy}")
```

```
Accuracy on the test set: 0.9666428571428571
```

```
display_confusion_matrix(y_test, y_test_pred)
```



III. Metod och Modeller (Teori)

Jag laddade MNIST-datasetet med hjälp av Scikit-learns fetch_openml-funktion och kontrollerade om det fanns saknade värden och fann att det inte fanns några. Därefter visualiserade jag en provbild från datasetet med hjälp av Matplotlib. För att förbereda för träning delade jag upp datasetet i tränings-, validerings- och testuppsättningar med hjälp av Scikit-learns train_test_split-funktion. För att se till att funktionerna är jämförbara, skalade jag pixelvärdena på bilderna med hjälp av Scikit-learns StandardScaler-klass.

Därefter tränade och utvärderade jag tre olika modeller på validerings-set: Random Forest, KNN och SVM. Jag fann att Random Forest presterade bäst på validerings-set. För att optimera Random Forest-

modellen använde jag GridSearch för att finjustera hyperparametrarna och hittade den bästa estimatorn.

Modeller

1. Random Forest

Random Forest är en maskininlärningsalgoritm som tillhör familjen av beslutsträdsbaserade algoritmer. Random Forest är en kraftfull algoritm som kan användas för både klassificering och regressionsuppgifter. Den kan hantera stora datamängder med högdimensionella funktioner, brusiga data och saknade värden. Huvudfördelen med Random Forest är att den minskar risken för överanpassning genom att använda en ensemble av beslutsträd.

2. Support Vector Machine (SVM):

SVM är en maskininlärningsalgoritm som används för klassificerings- och regressionsuppgifter. Den fungerar genom att hitta hyperplanet som bäst separerar datat i olika klasser. Den är särskilt användbar när antalet funktioner är större än antalet prover. SVM är också användbar när datat inte är linjärt separerbart, eftersom den kan använda kärnfunktioner för att transformera datat till ett högre dimensionellt utrymme, där det är lättare att separera klasserna.

3. K-Nearest Neighbors (KNN):

KNN är en maskininlärningsalgoritm som används för klassificerings- och regressionsuppgifter. Den fungerar genom att hitta de k närmaste grannarna till en ny datapunkt från träningsuppsättningen, baserat på ett likhetsmått.

4. GridSearch:

GridSearch är en teknik inom maskininlärning som används för att finjustera hyperparametrar i en modell för att öka dess prestanda. Hyperparametrar är parametrar som inte kan tränas direkt från data utan måste ställas in manuellt, till exempel antalet träd i en Random Forest-modell eller värdet på C i en SVM-modell.

IV. Resultat och Analys

Resultat

Modell	**Accuracy resultat – Training/Validation Set**
Random Forest Classifier	0.968
K Nearest Neighbor Classifier	0.943
Support Vector Machine Classifier	0.964

Analys

MNIST-dataset har ett klassificeringsproblem - som är att bygga en modell som kan korrekt klassificera handskrivna siffror från bilder. Målet är att utveckla ett system för automatisk sifferigenkänning för att förbättra effektiviteten i uppgifter som checkhantering eller postsortering. Detta är ett problem inom supervised learning.

I projektet tränade och utvärderade jag tre olika modeller på validerings-set: Random Forest, KNN och SVM. Jag fann att Random Forest presterade bäst på validerings-set. För att optimera Random Forest-modellen använde jag GridSearch för att finjustera hyperparametrarna och hittade den bästa estimatorn.

Efter att jag hade den bästa estimatorn utvärderade jag dess prestanda på validerings-set och fann en noggrannhet på 0,968. Slutligen utvärderade jag modellens prestanda på test-set och fann en noggrannhet på 0,967. Denna mätning kallas modellens generaliseringsfel och indikerar att den presterar bra på nya, osedda data.

V. Slutsats

Baserat på resultaten av modellträning och utvärdering som utförts på MNIST-data genom olika algoritmer, kan det dras slutsatsen att vi kan uppnå hög prediktionsförmåga för att klassificera handskrivna siffror med hjälp av dessa modeller. Bland de tre algoritmerna som utvärderades - Random Forest, KNN och SVM - visade Random Forest sig vara den mest effektiva med en noggrannhet på 0,968 på valideringsuppsättningen och 0,967 på testuppsättningen. Med vidare finjustering av hyperparametrar och ökad datamängd kan det vara möjligt att öka noggrannheten ytterligare och uppnå ännu högre prediktionsförmåga.

VI. Potentiell vidareutveckling

En potentiell vidareutveckling på MNIST maskininlärnings projekt skulle kunna vara att använda en mer avancerad modell för att förbättra prestandan. En annan vidareutveckling kan vara att öka storleken på datasetet genom att lägga till fler handskrivna siffror, eller genom att inkludera andra typer av handskrivna tecken som bokstäver och symboler. En större och mer varierad datamängd kan hjälpa modellen att bli mer allmän och effektivare på att hantera olika typer av handskrivna tecken.

Kort Redogörelse

1. Utmaningar du haft under arbetet samt hur du hanterat dem.
Storlek på datamängden av MNIST-datasetet som innehåller 60 000 bilder för träning och 10 000 bilder för testning. Det var en utmaning när jag tränar modeller, särskilt när jag använder en långsam dator och det tog tid att få svar. Jag valde att använda mindre data för att köra modeller.
Det var utmanande också att välja algoritmer och modeller som passar just på MNIST-data eftersom det finns så många olika att välja på.

2. Vilket betyg du anser att du skall ha och varför.
Jag anser att få VG eftersom jag gjorde mitt bästa och la mycket tid för att förstå kursen. Jag studerar också utanför kursmaterialet för att öka min förståelse.

3. Tips du hade ”gett till dig själv” i början av kursen nu när du slutfört den.
På grund av att Data Science-program är intensivt, tidskrävande och fullt med utmaningar borde jag ha tagit 100% tjänstledigt för att kunna fokusera bara på studierna. Det är väldigt tufft att kombinera studier och jobb plus familjeliv med små barn. Jag skulle även rekommendera att man studerar och undersöker externt material för att få mer förståelse för kursens innehåll.