

Delprov 2

Introduktion till maskininlärning
DA568B

Alexander Lagerqvist
1994-11-03-3377
Inlämnad: 2023-11-03

Task 1 (a)

Put **all data into one common DataFrame**, both features and targets. Show that you obtained the correct dataframe. **Concatenate two dataframes** (row-wise) with 'concat' and axis=0.

Task 1 (a)

Ladda in fashion MNIST set i en tränings/test split. Verifiera inläsning och split.

Observera att X-datan är en i en 3D array vilket kommer behövas "tillplattas"

```
1 #Imports
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from sklearn.model_selection import train_test_split
6 from keras.datasets import fashion_mnist as fashion_mnist
```

```
1 #Laddar in träningsdata och testdata där X är features och Y är targets
2 #När man laddar in på det här viset får man automatiskt en train_test_split.
3 (X_train,y_train),(X_test,y_test)=fashion_mnist.load_data()
```

```
1 #Vi kan visa att vi har en korrekt tränings/test split genom att kolla på formen av datan.
2 print("X_train shape:", X_train.shape)
3 print("y_train shape:", y_train.shape)
4 print("X_test shape:", X_test.shape)
5 print("y_test shape:", y_test.shape)
6
```

X_train shape: (60000, 28, 28)

y_train shape: (60000,)

X_test shape: (10000, 28, 28)

y_test shape: (10000,)

Task 1 (a)

För att mata in vår data i en PCA och Pandas är det att föredra datan i en 2D array istället för 3d

Därav måste vi platta till datan

Nu kan vi verifiera tillplattningen när vi kollar på "formen" av X-datan

```
1 #Utför tillplattningen av X-datan.  
2 X_train = X_train.reshape(X_train.shape[0], -1)  
3 X_test = X_test.reshape(X_test.shape[0], -1)
```

```
1 #Här kan vi observera tillplattningen där vi ser att vi har gått från att ha bildens dimensioner 28x28  
2 #till att representeras som en enda rad, vilket blir produkten av de ursprungliga dimensionerna  
3  
4 print("X_train shape:", X_train.shape)  
5 print("y_train shape:", y_train.shape)  
6 print("X_test shape:", X_test.shape)  
7 print("y_test shape:", y_test.shape)
```

```
X_train shape: (60000, 784)  
y_train shape: (60000,)  
X_test shape: (10000, 784)  
y_test shape: (10000,)
```

*Skapar två **Dataframes**:
En för träning, en för
test.*

*Slår ihop till en
gemensam dataframe.*

[illegible]

Task 1 (b)

Select out targets 0 to 3 if your last name starts with A to G.

Select out targets 4 to 7 if your last name starts with H to M.

Select out targets 6 to 9 if your last name starts with N to Ö.

Continue the rest of the assignment with only your targets !

Task 1 (b)

Efternamn = Lagerqvist
Filtrerar bort data som inte har targets mellan 4-7.

Verifierar att filtreringen är korrekt genomförd genom att kolla på formen där vi ser att av 10 targets har vi nu 4 targets och det innebär att vi arbetar vidare med 40% av ursprungsdatan

```
1 #Arbetar vidare med de separata tränings och test dataframsen.
2
3
4 #Filtrerar bort datan i både träning och test dfs som inte har targets mellan 4-7
5 train_df = train_df[(train_df['target'] >= 4) & (train_df['target'] <= 7)]
6
7 test_df = test_df[(test_df['target'] >= 4) & (test_df['target'] <= 7)]
8
9
10 print(train_df.shape)
11 print(test_df.shape)
12 train_df.head()
```

(24000, 785)
(4000, 785) *Filtrerad data*

Ursprungsdatan

X_train shape: (60000, 784)
y_train shape: (60000,)
X_test shape: (10000, 784)
y_test shape: (10000,)

~~0,1,2,3~~, **4,5,6,7**, ~~8,9~~

4/10 datan är kvar = 40%

24000 rader efter filtrering

$24000/60000 = 40\%$

Lika så testdatan kan vi verifiera -

$4000/10000 = 40\%$

Datan är jämnt fördelad och filtrerad!

Task 1 (b)

*Ett snabbt **stickprov** där vi kollar targets i de första 10 raderna i både träning- och test dataramen kan vi se att det ser riktigt ut. Dvs att vi endast jobbar med targets mellan 4-7*

```
In [12]: 1 #Stickprov för att se att vi fortsättningsvis arbetar med rätt targets.
          2
          3 print("Verifierar träningsdatan")
          4 print(train_df['target'].head(10))
          5
          6 print("Verifierar testdatan")
          7 print(test_df['target'].head(10))
```

Verifierar träningsdatan

6	7
8	5
9	5
12	5
13	5
14	7
18	6
19	4
22	4
24	4

Name: target, dtype: uint8

Verifierar testdatan

4	6
6	4
7	6
8	5
9	7
10	4
11	5
12	7
14	4
17	4

Name: target, dtype: uint8

Task 2 (a)

a) Make a **PCA** plot of your training data, with datapoints colored by the target class.

Task 2 (a)

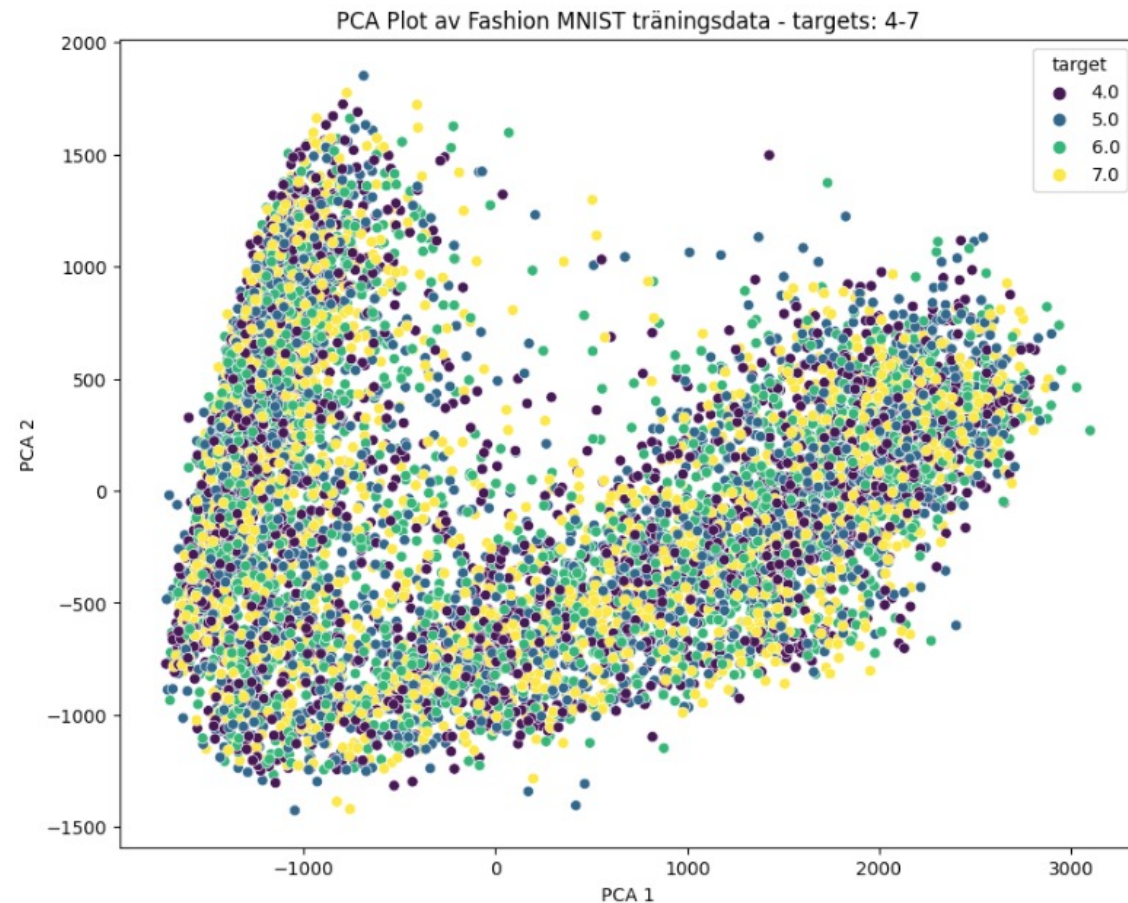
*För att göra en **PCA-plot** behöver vi först göra en PCA. För enkelhetens skull för plotting, valde jag en 2D PCA*

```
1 from sklearn.decomposition import PCA
2
3 #Instansierar ett 2D PCA objekt
4 pca = PCA(2)
5
6 #Vi använder här PCA objekten för att reducera dimensionerna i features.
7 #Notera att vi tar bort targets temporärt då vi enbart fokuserar på egenskaperna i datan (pixlarna)
8 pca_features = pca.fit_transform(train_df.drop('target', axis=1))
9
10 #Ordnar datan från PCA genom en egen dataframe där kolumnerna består av PCA 1 & 2
11 df_pca_features = pd.DataFrame(data=pca_features, columns=['PCA 1', 'PCA 2'])
12
13 #Lägger tillbaka 'target'-kolumnen från den ursprungliga dataframen
14 df_pca_features['target'] = train_df['target']
15
```

Task 2 (a)

*Plotten består av en kombination av grundfunktioner i **matplotlib** men kommer till liv med hjälp av **sns** som har möjliggör smidig färgkodning.*

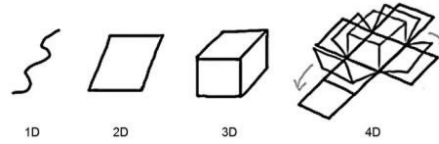
```
1 import seaborn as sns
2 #För att möjliggöra färgkodningen baserad på targets, använt genomgående under kursens gång.
3
4 #Storleken av plotten
5 plt.figure(figsize=(10, 8))
6
7 #Punkt/Spridningsdiagram: PCA 1 på X-axeln som innehåller mest information om variationen i datan.
8 #Det är hue som möjliggör färgkodningen på targets och paletten väljer en uppsättning färger.
9 #Fördelen mot matplotlib är att sns har fördefinierade färgpaletter.
10
11 sns.scatterplot(x='PCA 1', y='PCA 2', hue='target', data=df_pca_features, palette='viridis')
12 plt.title('PCA Plot av Fashion MNIST träningsdata - targets: 4-7')
13 plt.show()
```



Task 2 (b)

b) In your own words, explain **how PCA works** and what **mathematical Variance mean!**

Principal Component Analysis



Generellt:

PCA är användbart när vi vill visualisera data som är "multidimensionell" eftersom människan inte kan visualisera dimensioner över 3D.

PCA kan ta ett dataset såsom *fashion mnist* som har en stor mängd "features" och reducera ner till endast 3, 2, eller 1 features samtidigt som man försöker behålla så mycket man kan av ursprungsinformationen. Vi kan sedan använda dessa "*principal components*" för att visualisera den komplexa datan och identifiera mönster, kluster och utvärdera om datan är användbar för ML. Finns det tydliga kluster men viss överlappning mellan olika targets kan datasetet vara en kandidat för vidare ML-bearbetning.

Förklarar PCA vidare på nästa slide →

Task 2 (b)

Principal Component Analysis

	Sepal length	Sepal width	Petal length	Petal width	Class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
⋮	⋮	⋮	⋮	⋮	⋮
150	5.9	3.0	5.1	1.8	virginica

Hur:

Om vi tar exemplet med Iris-blommorna vi arbetat med så börjar vi med att bortse från target-kolumnen och enbart fokuserar på de 4 features.

Lite förenklat kan man förklara hur PCA går till:

1. För varje feature (*tex. sepal width, sepal length, petal... osv*) beräknar vi **medelvärdet** av alla observationer.
2. Vi **centrerar** sedan datan genom att ta varje observations värde subtraherat med medelvärdet för featuren.
 1. Nu är Iris-setet ett enkelt dataset med lite varians men om vi inte centrerar kan resultatet påverkas av skalan eller enheterna på de olika features.
3. Som du har beskrivit i din föreläsning försöker vi **"fit a line"** genom vår centrerade data för maximera variansen på de projekterade datapunkterna. Den linje som ger kortast avstånd (*sum of least squares*) mellan linjen och alla datapunkter blir den bästa linjen. Det är i kvadrat (squares) så vi inte påverkas av negativa värden. **Detta ger oss PCA1.**
 1. *Det är därför PCA1 innehåller mest information om variationen i datan.*
4. För att få fram PCA2, tar vi helt enkelt en linje som går vinkelrätt till PCA1.

Task 2 (b)

b) In your own words, explain how PCA works and what **mathematical Variance mean!**

I kontexten för PCA spelar mattematisk variation en stor betydelse.

Matematisk varians inriktar sig på att få en uppfattning om variationen hos varje datapunkt i förhållande till medelvärdet av datasetet.

Om vi jämför den definitionen till hur jag i förra slide förklarade hur en PCA fit fungerar, så finner vi likheter i det att PCAn iterativt försöker hitta den linje som är "least sum of squares" vilket syftar till att behålla den maximala variationen i datan.

Där finner vi likheter i definitionen av matematisk variation som beräknas genom att *ta kvadraten av avståndet mellan varje datapunkt och dess medelvärde*.

Personlig erfarenhet av arbeta med variation:

I egenskap av att arbetat som kundservicechef inom Apples tekniska support samarbetade vi med business intelligence analytiker varje månad för att objektivet ta fram kandidater (tekniska rådgivare) som vi chefer skulle sätta upp handlingsplaner på i syfte att förbättra resultatet på diverse nyckeltal inom supportverksamheten. Där arbetade vi just med variation, standardavvikelse i detta fall. På en avdelning, kollade vi då på ett nyckeltal, tex. samtalstid och mätte standardavvikelsen istället för en hård gräns på vad samtalstiden skulle vara vilket skulle diktera vilka rådgivare som skulle få en handlingsplan. Det innebar att vi fokuserade på de rådgivare som hade högst variation och som avvek från kollektivet (medelvärdet av ett nyckeltal).

Task 3

Make a **Random Forest** classification prediction on your Data and make a **Confusion matrix** of your results.

What is your accuracy ? Discuss your results!

Task 3

*Förbereder datan genom att
att extrahera från
dataframe.*

*Kör träningsdatan med två
hyperparametrar*

```
1 from sklearn.ensemble import RandomForestClassifier
2
3
4 # För att förbereda datan till en Random Forest tar vi ut datan från dataframen igen.
5 # Nu får vi formen som vi hade i början av uppgiften men vi arbetar endast med targets 4-7
6 X_train = train_df.drop('target', axis=1)
7 y_train = train_df['target']
8
9 # Repeterar för testdatan
10 X_test = test_df.drop('target', axis=1)
11 y_test = test_df['target']
12
13 #Hyperparametrarna jag använder är 1000 träd samt ett randomseed för att säkerställa samma resultat.
14 ran_for=RandomForestClassifier(n_estimators = 1000, random_state = 42)
15
16 ran_for.fit(X_train, y_train)
```

Task 3

Kör test-datan igenom modellen vi tränat på vår träningsdata.

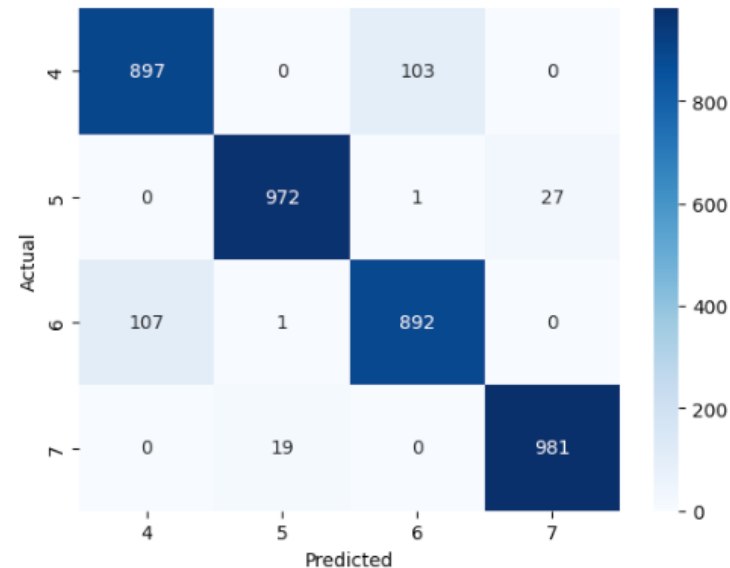
Visualiserar en confusion matrix med hjälp av sns och matplotlib.

```
RandomForestClassifier
RandomForestClassifier(n_estimators=1000, random_state=42)
```

```
1 #Efter vi tränat träningsdatan, testar vi nu vår modell på testdatan och sparar ner resultatet
2 test_prediction_array = ran_for.predict(X_test)
3 test_prediction_array
```

```
array([6, 4, 6, ..., 5, 6, 5], dtype=uint8)
```

```
In [14]: 1 from sklearn.metrics import confusion_matrix
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 # Vi gör en confusion matrix genom att ta de faktiska rätta target värdena och jämför med modellens output
6 confusion = confusion_matrix(y_test, test_prediction_array)
7
8 # Med hjälp av sns i kombination med matplotlib som vi även använde vid PCA plotten kan vi illustrera
9 # confusion matrisen på ett bra sätt.
10 sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', xticklabels=[4, 5, 6, 7], yticklabels=[4, 5, 6, 7])
11 plt.xlabel('Predicted')
12 plt.ylabel('Actual')
13 plt.show()
```



Task 3

Analys: Vi har en accuracy på 94% vilket är bra.

Vi kan notera att modellen har relativt svårt att urskilja mellan target 4 och 6.

Vad kan det bero på?



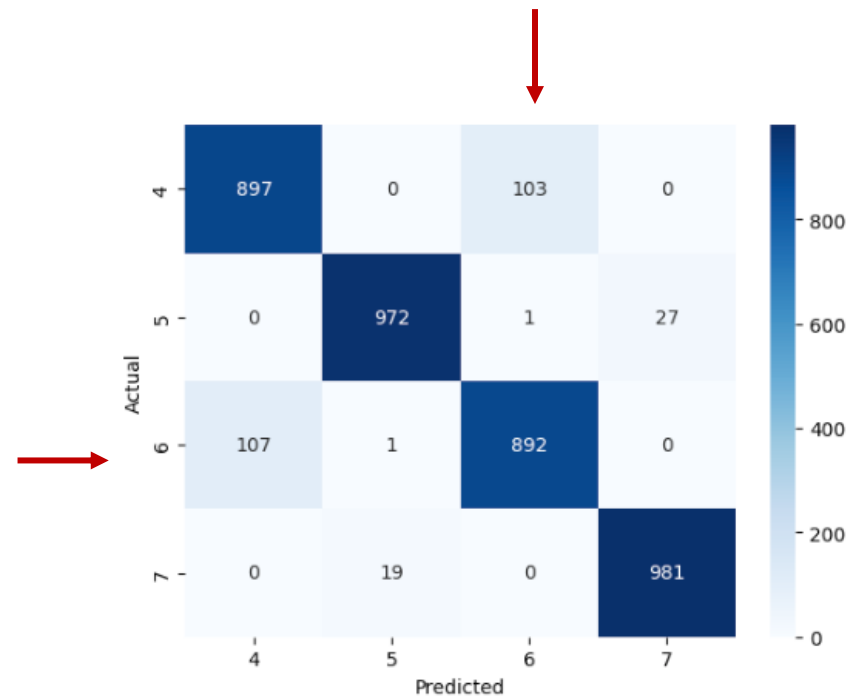
In [213]:

```
1 from sklearn.metrics import classification_report
2
3 # Vi gör en classification report genom att ta de faktiska rätta target värdena och jämför med modellens output
4 report = classification_report(y_test, test_prediction_array)
5
6 |
7 print(report)
```

```
Classification report: Random Forest
              precision    recall  f1-score   support

     4         0.89      0.90      0.90       1000
     5         0.98      0.97      0.98       1000
     6         0.90      0.89      0.89       1000
     7         0.97      0.98      0.98       1000

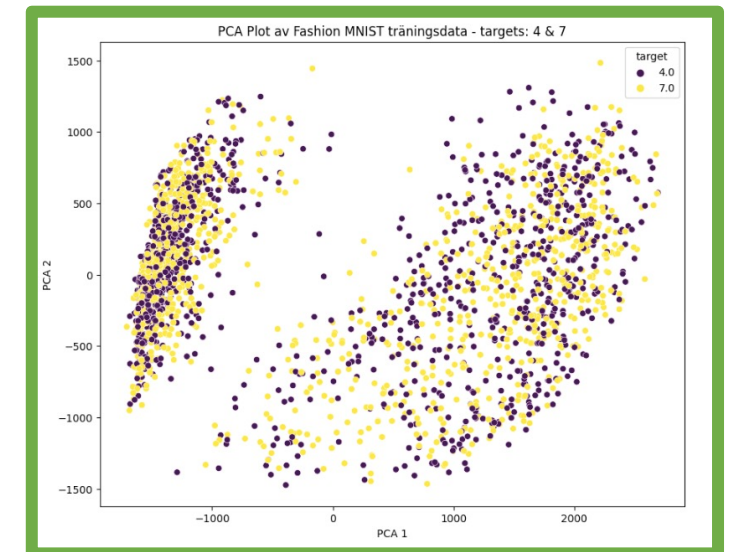
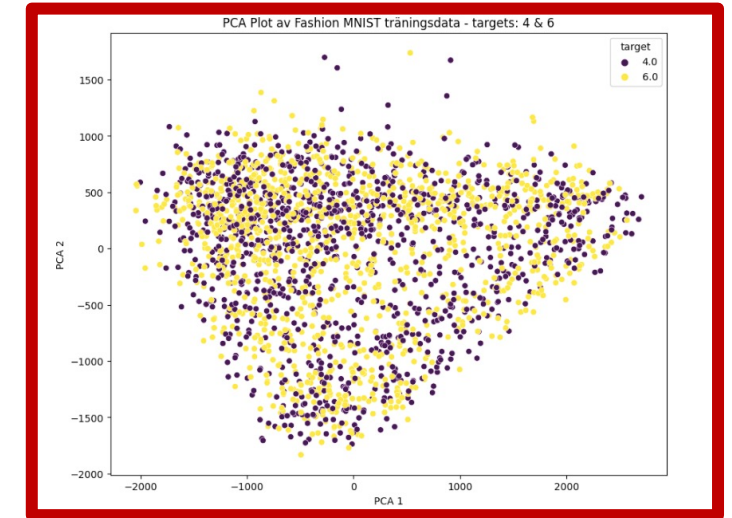
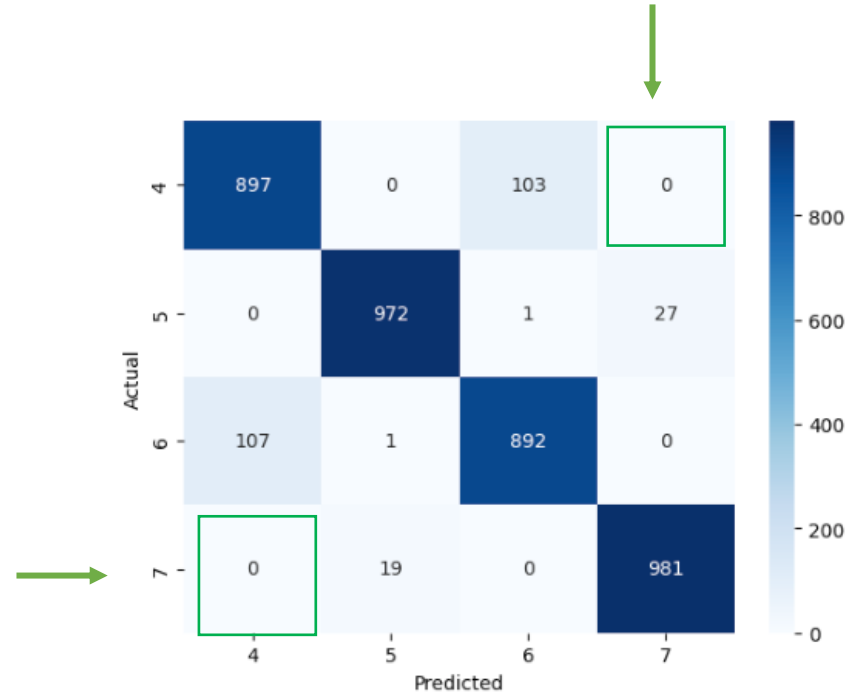
 accuracy          0.94       4000
 macro avg         0.94       4000
 weighted avg      0.94       4000
```



Task 3

***Analys:** Om vi gör en PCA plot på endast två features där vi jämför target 4 vs 6 som har svårigheter att urskilja varandra kontra 4 vs 7 där motsatsen råder kan vi i PCA plotten se ett större överlapp mellan 4 & 6 jämfört med 4 & 7 som tycks "klustra" lite tydligare. Detta kan vara en anledning till varför.*

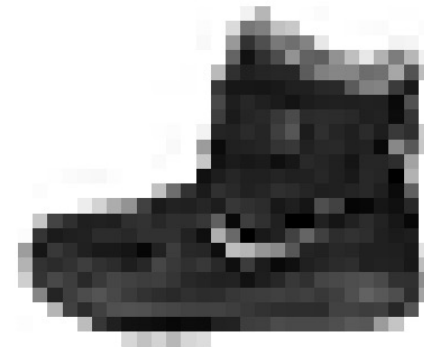
Ytterligare förklaring →



Task 3

För att analysera resultatet vidare blev jag inspirerad av att återbesöka en övning från vecka 39 där jag just gjorde en random forest på fashion MNIST. Då blev jag först nyfiken på att kolla på hur samtliga bilder i setet såg ut.

I nästa slide kan vi tydligare förstå överlappningen då vissa av de bilder (4,5,6,7) jag har arbetat med i denna uppgift är lika varandra i strukturen



Jupyter

V39 *FASHION MNIST Random Forest Last Checkpoint: 2023-09-25 (unsaved changes)

Logout

FileEditViewInsertCellKernelHelp

TrustedPython 3 (ipykernel)

+

⌕

📄

📁

⬆

⬇

▶ Run

⏏

↺

▶▶

Code

⌵

🗨

In [2]:


```
1 #Imports
2
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import matplotlib as mpl
7 import keras
8
```

In [3]:

```
1 #Ladda in fashion MNIST och fördela tränings/test-split
2 from keras.datasets import mnist
3 fashion_mnist = keras.datasets.fashion_mnist
4
5 (train_X, train_y), (test_X, test_y)= fashion_mnist.load_data()
6
7
```

In [4]:

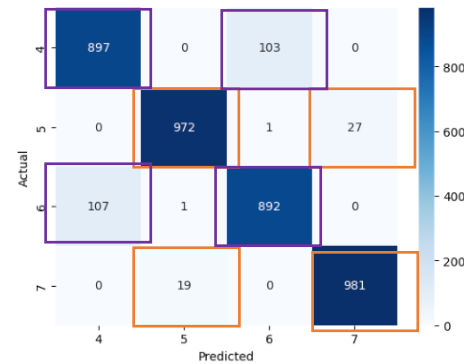
```
1 #Loopa igenom första 10 bilder i träningsset
2 #Plotta ut bild tillsammans med utskrift för tillhörande target label.
3 for i in range(0,10):
4     some_digit = train_X[i]
5     some_digit_image = some_digit.reshape(28, 28)
6     plt.imshow(some_digit_image, cmap = mpl.cm.binary, interpolation="nearest")
7     plt.axis("off")
8     plt.show()
9
10 print(train_y[i])
```



9

Task 3

Analys: När vi kollar på vad de faktiska targets representerar blir det ganska logiskt till överlappningen och svårigheten att urskilja. Vi noterar tydligt att 4 & 6 är tröjor som har liknande pixelstruktur. 5 & 7 har mindre svårt att urskilja sig och kan möjligtvis ha att göra med target 5 som verkar vara en sandal med ett större "tomt vitt" område i sin bild.



4



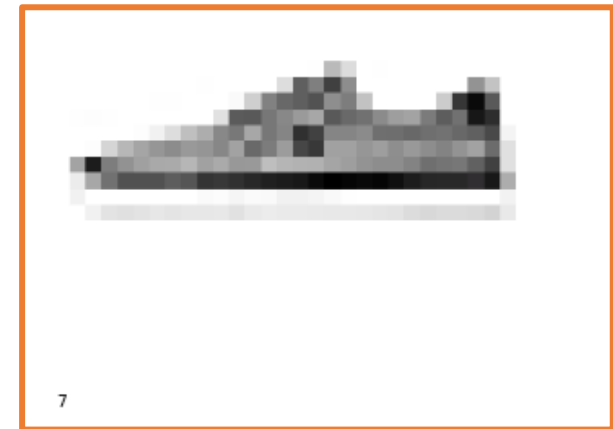
5



6



7



Task 4

*Make a **Random Forest** classification prediction on your Data and make a **Confusion matrix** of your results.*

What is your accuracy ? Discuss your results!

Repeat the above test 3, but use Logistic regression instead. Which one worked better on your data ?

Task 4

Plan för logistic regression.

För logistic regression på MNIST Fashion vill jag se om jag kan utmana random forest-modellen genom att först köra träna datan som den är och utvärdera resultatet.

Sen vill jag även se om jag kan förbättra resultatet genom att normalisera datan i förmån för logistic regression som bör ha det lite tuffare i detta scenario jämfört med en random forest.

Visar i kommande slides när jag normaliserar datan men passar på att visa en jämförelse i onormaliserad vs normaliserad feature data (vänster onormaliserad, höger normaliserad).

[illegible]

```

1 #Loopa igenom pixeldata
2 for item in X_train[:10]:
3     print(item)

```

```

[-6.00827340e-03 -2.09570862e-02 -3.10785572e-02 -3.52429887e-02
-5.27492128e-02 -6.34772032e-02 -8.85501981e-02 -1.41675178e-01
-1.96001814e-01 -2.49624528e-01 -3.16514977e-01 -4.82169596e-01
-6.66456955e-01 -6.72214668e-01 -6.66695702e-01 -6.82830268e-01
-5.84655388e-01 -6.37203583e-01 -2.79565881e-02 -2.21495946e-01
-1.68460647e-01 -1.20667196e-01 -9.61036939e-02 -8.04413535e-02
-5.66969544e-02 -3.42915045e-02 -2.01028710e-02 -9.18171177e-03
-1.79738017e-02 -3.01836294e-02 -2.8752004e-02 -5.17883264e-02
-7.47958266e-02 -1.16184226e-01 -2.00294078e-01 -2.71662336e-01
-3.34277939e-01 -4.21656534e-01 -5.60055815e-01 -7.66235677e-01
-8.7103708e-01 -8.65377257e-01 -8.60982424e-01 -8.78507193e-01
-8.36993677e-01 -6.42873039e-01 -4.78508053e-01 -3.76421632e-01
-3.12382238e-01 -2.48455910e-01 -1.81767727e-01 -1.35169938e-01
-9.76916485e-02 -6.00447055e-02 -3.26586050e-02 -1.65655092e-02
-1.69649236e-02 -1.94200792e-02 -3.86665488e-02 -6.35734115e-02
-1.09663699e-01 -1.94985460e-01 -2.80174442e-01 -3.76113603e-01
-5.15558994e-01 -6.81208929e-01 -8.01044977e-01 -8.58535841e-01
-8.79788208e-01 -9.96873979e-01 -9.84326916e-01 -9.02146746e-01
-7.74936351e-01 -8.34456222e-01 -7.58356336e-01 -6.9926463e-01

```


Task 4

Behöver inte allt för mycket förberedelse förutom att importera in Logistic Regression och konfigurera specifik hyperparameter. Annars repeterar vi metodiken för uppgift 3.

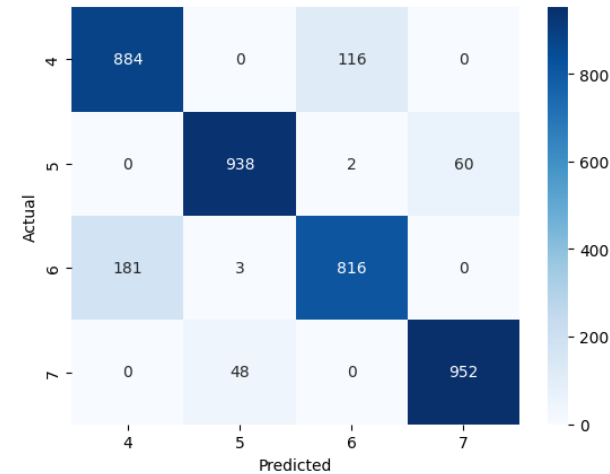
```
1 from sklearn.linear_model import LogisticRegression
2
3 # Likt uppgift 3 importerar vi vår ML modell.
4 # Istället för antalet träd anger vi max-iterations vilket anger hur många iterationer vi tillåter modellen -
5 # att optimera/justera för att anpassa sig till träningsdatan.
6 logreg = LogisticRegression(max_iter=5000, random_state = 42)
7
8 # Vi kan direkt använda datan som vi förberedde för uppgift 3
9 logreg.fit(X_train, y_train)
10
11 # Vi kan sedan använda vår tränade modell mot testdatan.
12 pred_logArray = logreg.predict(X_test)
13 pred_logArray
14
```

Task 4

Vi analyserar resultatet på exakt samma sätt som för random forest (därav avsaknaden av kommentarer i koden).

Helt okej resultat, 90% accuracy men sämre än random forest.

```
1 #Vi repeterar samma steg för att plotta confusion-matrisen som tidigare.
2
3 confusion = confusion_matrix(y_test, pred_logArray)
4
5
6 sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', xticklabels=[4, 5, 6, 7], yticklabels=[4, 5, 6, 7])
7 plt.xlabel('Predicted')
8 plt.ylabel('Actual')
9 plt.show()
```



```
1 #Vi använder samma princip med importen vi tog in för klassificeringsrapporten tidigare och --
2 #tillämpar på logistic regression modellen.
3
4 report = classification_report(y_test, pred_logArray)
5
6 # Skriv ut rapporten
7
8
9 print(report)
```

	precision	recall	f1-score	support
4	0.84	0.87	0.85	1000
5	0.95	0.94	0.95	1000
6	0.86	0.83	0.85	1000
7	0.95	0.96	0.95	1000
accuracy			0.90	4000
macro avg	0.90	0.90	0.90	4000
weighted avg	0.90	0.90	0.90	4000

Task 4

*Ett försök att förbättra
resultatet på modellen:
Normalisera datan i förmån för
logistic regression.*

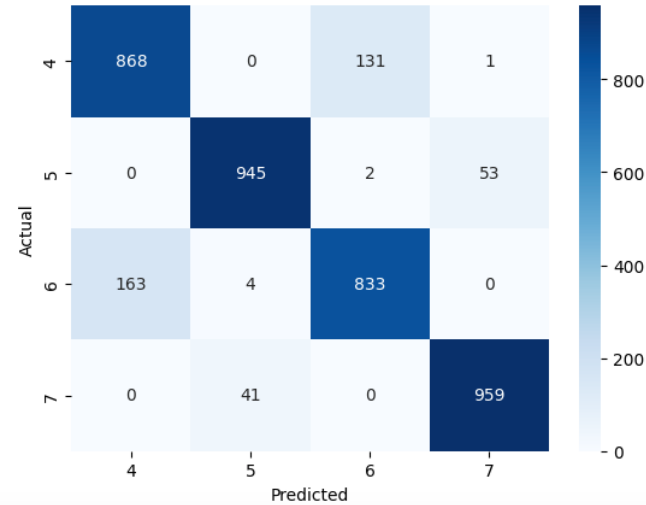
```
1 from sklearn.preprocessing import StandardScaler
2
3 # Normalisera datan i förmån för logistic regression
4 # Drar bort medelvärdet för varje feature. Centrerar datan runt 0.
5 #OBS Vi normaliserar inte targets, endast x-värden.
6 scaler = StandardScaler()
7 X_train = scaler.fit_transform(X_train)
8 X_test = scaler.transform(X_test)
9
10 # Efter normalisering tränar vi modellen som vanligt.
11 logreg = LogisticRegression(max_iter=5000, random_state = 42)
12 logreg.fit(X_train, y_train)
13
14 pred_logArray = logreg.predict(X_test)
15
```

Task 4

Vi kan konkludera att vi inte lyckades förbättra accuracy när vi normaliserade datan.

Det kan bero på att fashion mnist data setet är ett relativt enkelt dataset där normaliseringen inte hade någon påverkan.

```
4 confusion = confusion_matrix(y_test, pred_logArray)
5
6
7 sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', xticklabels=[4, 5, 6, 7], yticklabels=[4, 5, 6, 7])
8 plt.xlabel('Predicted')
9 plt.ylabel('Actual')
10 plt.show()
```



```
1 #Vi använder samma princip med importen vi tog in för klassificeringsrapporten tidigare och --
2 #tillämpar på logistic regression modellen.
3
4 report = classification_report(y_test, pred_logArray)
5
6 # Skriv ut rapporten
7
8
9 print(report)
```

	precision	recall	f1-score	support
4	0.83	0.88	0.86	1000
5	0.95	0.94	0.94	1000
6	0.87	0.82	0.84	1000
7	0.94	0.95	0.95	1000
accuracy			0.90	4000
macro avg	0.90	0.90	0.90	4000
weighted avg	0.90	0.90	0.90	4000

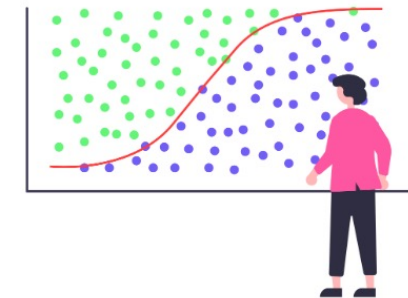
Slutsatsen är att random forest presterar bättre på just det här datasetet.

Det kan bero på att random forest tycks vara mer robust mot överanpassning än Logistic regression i den mån att datasetet har relativt många features (pixlar, 28x28) vilket ger en rätt högdimensionell datamängd vilket verkar vara till förmån för en random forest..

(<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>)



Classification report: Random Forest				
	precision	recall	f1-score	support
4	0.89	0.90	0.90	1000
5	0.98	0.97	0.98	1000
6	0.90	0.89	0.89	1000
7	0.97	0.98	0.98	1000
accuracy			0.94	4000
macro avg	0.94	0.94	0.94	4000
weighted avg	0.94	0.94	0.94	4000



	precision	recall	f1-score	support
4	0.84	0.87	0.85	1000
5	0.95	0.94	0.95	1000
6	0.86	0.83	0.85	1000
7	0.95	0.96	0.95	1000
accuracy			0.90	4000
macro avg	0.90	0.90	0.90	4000
weighted avg	0.90	0.90	0.90	4000

Task 5

Perform an investigation of your choice of different neuron architectures in a basic classification NN on your dataset and discuss your results.

Task 5

Förbereda datan för NN:

Skapa ett valideringsset som vi kan använda för att övervaka och utvärdera träningen.

Verifierar att splitten är rätt genom att se till att 20% av träningsdatan är fördelad till valideringen.

24 000 träning
20% -> validering
20% = 4800

Det ser rätt ut!

```
In [46]: 1 #Förbereda NN - Skapa valideringsset
2
3 print("Form innan valideringssplit:")
4 print(f"X train: {X_train.shape}")
5 print(f"Y train: {y_train.shape}")
6
7 # 20% av träningsdatan är lika med 0.2 * 24000 = 4800.
8 # Validerings setet kommer ta 4800 från träningsdatan.
9 val_size = int(0.2 * len(X_train))
10
11 print(f"Valideringsstorlek: {val_size} ")
12
13
14 X_val = X_train[:val_size]
15 y_val = y_train[:val_size]
16
17 X_train = X_train[val_size:]
18 y_train = y_train[val_size:]
19
20 print("Form efter valideringssplit:")
21 print(f"X train: {X_train.shape}")
22 print(f"Y train: {y_train.shape}")
23
24 print("Form på valideringsset:")
25 print(f"X val: {X_val.shape}")
26 print(f"Y val: {y_val.shape}")

Form innan valideringssplit:
X train: (24000, 784)
Y train: (24000,)
Valideringsstorlek: 4800
Form efter valideringssplit:
X train: (19200, 784)
Y train: (19200,)
Form på valideringsset:
X val: (4800, 784)
Y val: (4800,)
```

Task 5

Grund-Konfigruering av NN:

-Lager 1: Input layer, tillplattad input till förmån för den dataprep vi gjorde tidigare.

-Lager 2&3: Två hidden layers med 100 noder vardera. Vi använder ReLu för att införa icke-linjäritet i modellen.

-Outputlayer: Notera att det står 10.

Egentligen ska det vara 4. Trots försök att omkoda etiketterna till 0-4 från 4-7. Slutresultatet påverkas dock inte men har försökt hantera detta, notera senare i klassificeringsrapporten att den endast visar resultat för etiketterna 4,5,6,7.

Vi använder softmax som aktivering i output som är vanligt i dessa typer av multi-klassificeringsuppgifter

```
1 #Importerar keras och lagermoduler för att kunna skapa ett basic NN
2 from keras.models import Sequential
3 from keras.layers import Dense, Activation, Dropout
4
```

```
1 # Eftersom att vi redan normaliserat datan när vi körde logistic regression kan vi ...
2 # mata in input shapen som den är utan att behöva flatta 28x28
3
4 model = Sequential()
5 model.add(Dense(100, input_shape=(784,), activation="relu"))
6 model.add(Dense(100, activation="relu"))
7 model.add(Dense(10, activation="softmax"))
8
9 model.summary()
```

Model: "sequential_16"

Layer (type)	Output Shape	Param #
dense_33 (Dense)	(None, 100)	78500
dense_34 (Dense)	(None, 100)	10100
dense_35 (Dense)	(None, 10)	1010

```
=====  
Total params: 89610 (350.04 KB)  
Trainable params: 89610 (350.04 KB)  
Non-trainable params: 0 (0.00 Byte)
```


Task 5

Sista konfiguration & påbörja träning: Innan vi kör igång träningen så väljer vi ut ett par sista hyperparametrar som passar vårt ändamål:

”Loss-funktion” & ”Learning-optimizerare”

När stoppar in träning och valideringsdatan i vår förberedda modell väljer vi också träningsrundor (epochs) samt batchsize (styckad data).

Vi kan exempelvis reglera mängden träningsrundor om vi ser att valideringssetet försämras genom overfitting om vi har för många träningsrundor.

```
1 #Denna del sätter samman det neurala nätverk innan träningen utförs. Dvs. en del av våra hyperparametrar
2
3 #lossfunktionen fungerar bra när vi har flera klasser och etiketterna (target värden) är heltal (4,5,6,7)
4 #optimizer dikterar hur modellen ska anpassa inlärningsförmågan (gradient decent). Finns andra tex. Adam
5 #accuracy är det mått vi använder för att utvärdera hur väl modellen klassificerar
6
7 model.compile(loss="sparse_categorical_crossentropy",
8               optimizer="sgd",
9               metrics=["accuracy"])

```

```
1 #Här tränar vi nätverket på träningsdatan samt valideringsdatan.
2 #Epochs och ev. batchsize är hyperparametrar vi kan konfigurera för att exempelvis:
3 # diktera hur många gånger/tid vi ger nätverket att träna på datan.
4 # batchsize kan stycka upp träningsdatan i mindre delar för varje träningsrunda (epoch)
5
6 history=model.fit(X_train, y_train, epochs=40, validation_data=(X_val,y_val))

```

Task 5

Utvärdera & övervaka träning:

För att få en bättre insikt i hur träningen går har jag skapat två plots där vi kan följa hur tränings- respektive valideringsdatan presterar med avseende på loss och accuracy.

I nästa slide vill jag demonstrera användningsområdet när vi exempelvis använder 100 träningsrundor och hur plotten vi förberett här kan ge oss insikt i överfitting.

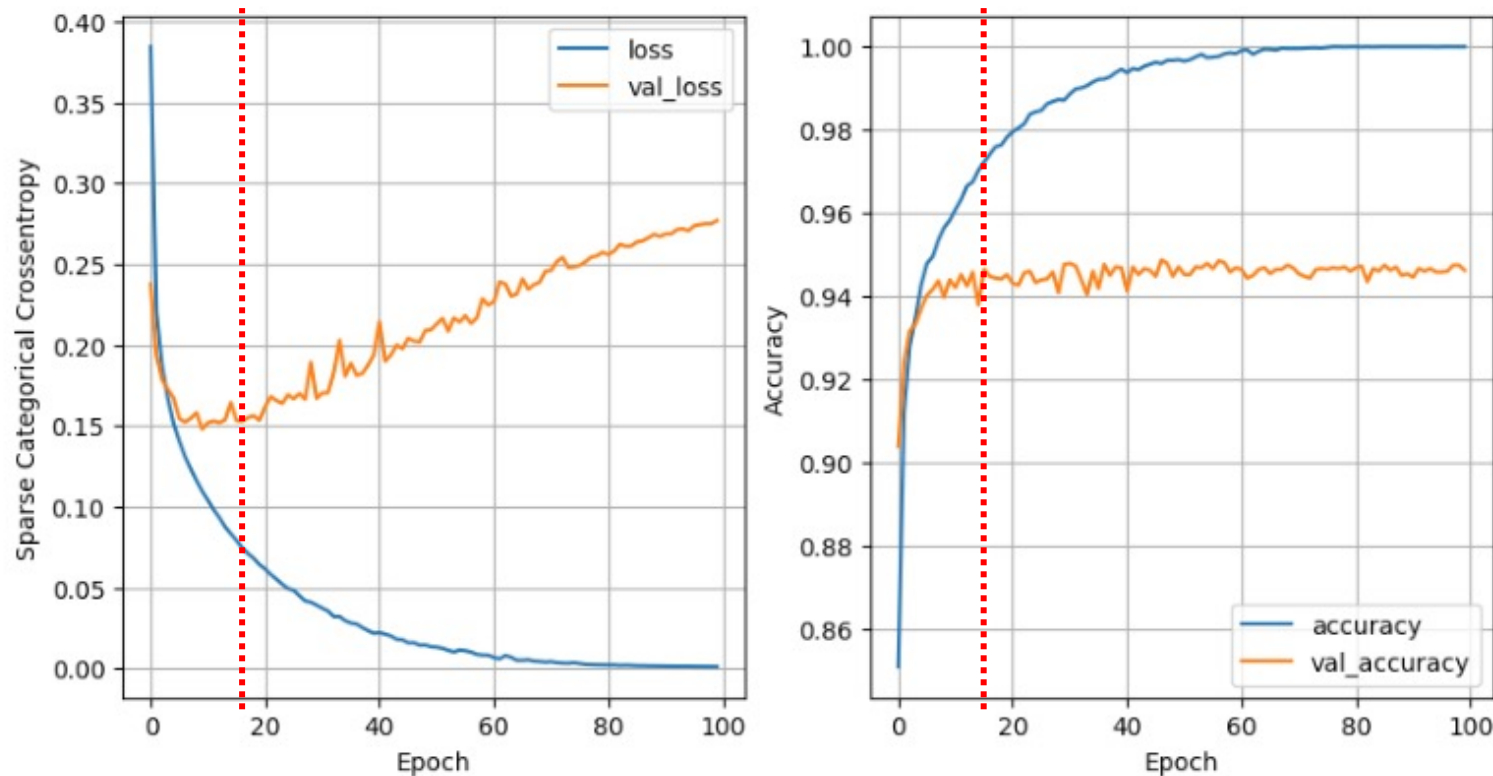
```
1 #Här tränar vi nätverket på träningsdatan samt valideringsdatan.
2 #Epochs och ev. batchsize är hyperparametrar vi kan konfigurera för att exempelvis:
3 # diktera hur många gånger/tid vi ger nätverket att träna på datan.
4 # batchsize kan stycka upp träningsdatan i mindre delar för varje träningsrunda (epoch)
5
6 history=model.fit(X_train, y_train, epochs=100, validation_data=(X_val,y_val))
```

```
1 #Utvärdera träningsdata och valideringsdata
2
3 #Unyttjar endast matplotlib här med att deklarera en subplot vilket ger två plots bredvid varandra.
4 #Datan för plotten tar vi från history som har sparad information från modellen.
5 #Vi kan därmed tilldela x & y axeln loss och validerings loss respektive accuracy.
6 fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,5))
7 ax1.plot(history.history['loss'], label = 'loss')
8 ax1.plot(history.history['val_loss'], label = 'val_loss')
9 ax1.set_xlabel('Epoch')
10 ax1.set_ylabel('Sparse Categorical Crossentropy')
11 ax1.legend()
12 ax1.grid(True)
13
14 ax2.plot(history.history['accuracy'], label = 'accuracy')
15 ax2.plot(history.history['val_accuracy'], label = 'val_accuracy')
16 ax2.set_xlabel('Epoch')
17 ax2.set_ylabel('Accuracy')
18 ax2.legend()
19 ax2.grid(True)
20 plt.show()
```

Task 5

Overfitting alert: För att utvärdera hur många träningsrundor (epochs) är rimligt ser vi värdet av att kolla på hur valideringssetet presterar. I exemplet nedan har vi 100 träningsrundor. Vi ser att tränings datan ser ut att få jättebra accuracy men det sker pågrund av overfitting. Notera att det krävs inte alls många epochs innan validerings setet når en platå eller rentav blir sämre.

Ser ut att räcka med omrking 10-15 epochs för att inte få en overfittad modell. Noterar även att vi är runt 94% här med en basic NN vilket är likställt med prestationen vi uppnådde med vår random forest. Låt oss utforska om vi kan justera vår NN-arkitektur för att utmana random forest.



Task 5

***Utvärdera vidare:** Om vi vill göra en klassisk confusion matrix och rapport som vi gjort tidigare i uppgiften behöver vi ta hänsyn till faktumet att neurala nätverket producerar resultatet i form av "sannolikheter".*

*Därav använder vi **numpy** för att konvertera från sannolikheter till en ren klassetikett modellen sannolikhet har predictat.*

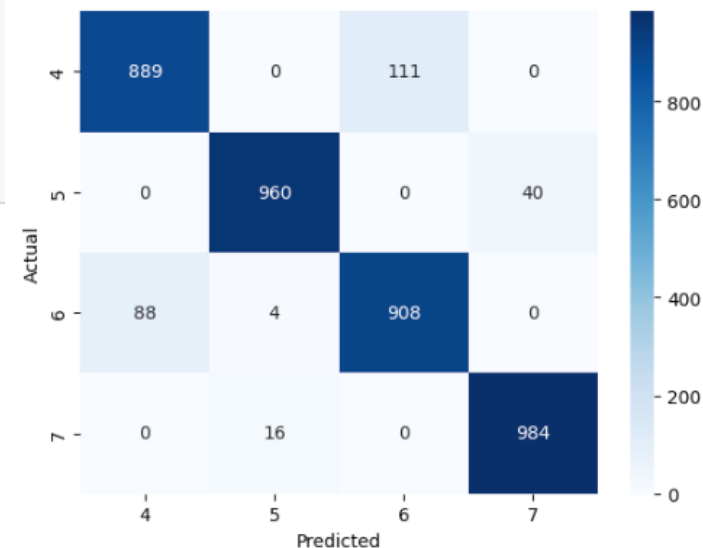
```
1 y_pred_sannolikheter = model.predict(X_test)
array([[1.1024708e-03, 5.8498309e-04, 3.9506363e-04, ..., 1.1864705e-04,
        5.0052220e-04, 3.1079259e-04],
       [6.9802176e-08, 4.1062847e-08, 6.1898540e-08, ..., 1.4658174e-08,
        9.2743285e-08, 5.5896248e-08],
       [1.1771098e-06, 6.2950257e-08, 3.3530353e-07, ..., 4.7853939e-09,
        1.9590712e-07, 8.2784751e-08],
       ...,
       [1.2781683e-10, 1.2431138e-09, 7.7404936e-09, ..., 5.4792190e-07,
        2.3990797e-11, 3.4157295e-11],
       [4.0416030e-06, 1.0144710e-05, 1.1316350e-06, ..., 1.8610477e-07,
        5.7743860e-06, 3.3235649e-06],
       [7.7026095e-07, 5.4548450e-06, 2.8182449e-06, ..., 2.2794653e-03,
        2.0944935e-06, 1.9779448e-07]], dtype=float32)
```

```
1 #Klassificerings rapport
2
3
4 y_pred_sannolikheter = model.predict(X_test)
5
6 # Konvertera från sannolikheter till klassetiketter
7 y_pred = np.argmax(y_pred_sannolikheter, axis=1)
8
9 # Skapa rapporten som vi tidigare gjort.
10 rapport = classification_report(y_test, y_pred)
11
12 # Skriv ut rapporten
13 print(rapport)
14
```

```
125/125 [=====] - 0s 947us/step
              precision    recall  f1-score   support

         4         0.91      0.89      0.90      1000
         5         0.98      0.96      0.97      1000
         6         0.89      0.91      0.90      1000
         7         0.96      0.98      0.97      1000

 accuracy          0.94
 macro avg          0.94
 weighted avg       0.94
```



Task 5

Konfiguration 1 och resultat

```

1 # Eftersom att vi redan normaliserat datan när vi körde logistic regression kan vi ...
2 # mata in input shapen som den är utan att behöva flatta 28x28
3
4 model = Sequential()
5 model.add(Dense(100, input_shape=(784,), activation="relu"))
6 model.add(Dense(100, activation="relu"))
7 model.add(Dense(10, activation="softmax"))
8
9 model.summary()

```

Model: "sequential_2"

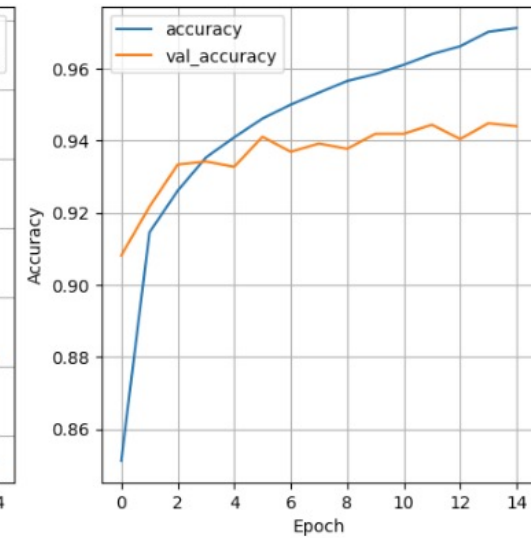
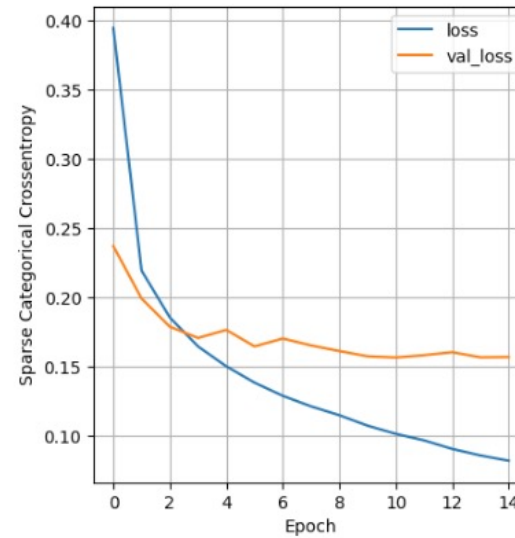
Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 100)	78500
dense_7 (Dense)	(None, 100)	10100
dense_8 (Dense)	(None, 10)	1010
Total params: 89610 (350.04 KB)		
Trainable params: 89610 (350.04 KB)		
Non-trainable params: 0 (0.00 Byte)		

```
history=model.fit(X_train, y_train, epochs=15, validation_data=(X_val,y_val))
```

```
1 model.evaluate(X_test,y_test)
```

125/125 [=====] - 0s 2ms/step - loss: 0.1709 - accuracy: 0.9352

[0.1709408313035965, 0.9352499842643738]



	precision	recall	f1-score	support
4	0.91	0.89	0.90	1000
5	0.98	0.96	0.97	1000
6	0.89	0.91	0.90	1000
7	0.96	0.98	0.97	1000
accuracy			0.94	4000
macro avg	0.94	0.94	0.94	4000
weighted avg	0.94	0.94	0.94	4000

Task 5

Konfiguration 2 och resultat

Pyttelite bättre!

```
1 #Importerar keras och lagermoduler för att kunna skapa ett basic NN
2 from keras.models import Sequential
3 from keras.layers import Dense, Activation, Dropout
4
```

```
1 # Eftersom att vi redan normaliserat datan när vi körde logistic regression kan vi ...
2 # mata in input shapen som den är utan att behöva flatta 28x28
3
4 model = Sequential()
5 model.add(Dense(100, input_shape=(784,), activation="relu"))
6 model.add(Dense(100, activation="relu"))
7 model.add(Dropout(0.25))
8 model.add(Dense(128, activation = "relu"))
9 model.add(Dropout(0.25))
10 model.add(Dense(10, activation="softmax"))
11
12 model.summary()
```

Dropout stänger ned vissa neuroner vilket bidrar till att nätverket behöver hitta nya mönster i datan och inte memorisera träningsdatan

Model: "sequential_4"

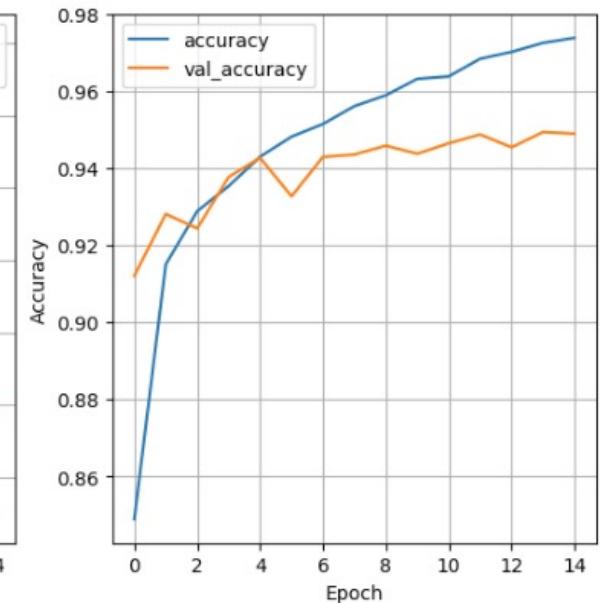
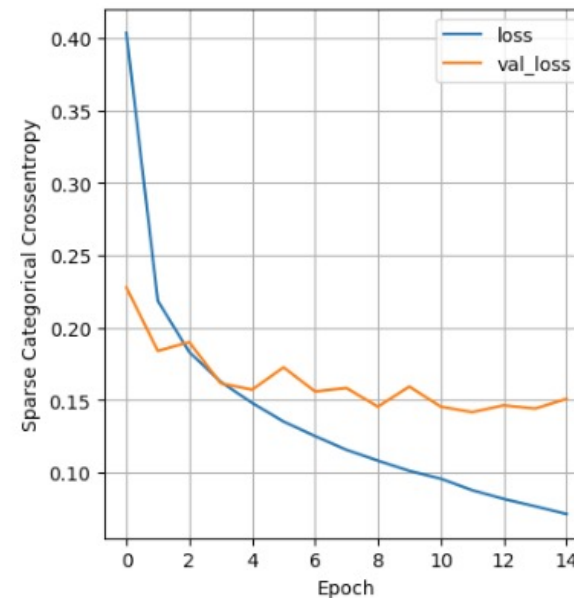
Layer (type)	Output Shape	Param #
dense_11 (Dense)	(None, 100)	78500
dense_12 (Dense)	(None, 100)	10100
dropout (Dropout)	(None, 100)	0
dense_13 (Dense)	(None, 128)	12928
dropout_1 (Dropout)	(None, 128)	0
dense_14 (Dense)	(None, 10)	1290

=====
Total params: 102818 (401.63 KB)
Trainable params: 102818 (401.63 KB)
Non-trainable params: 0 (0.00 Byte)

```
history=model.fit(X_train, y_train, epochs=15, validation_data=(X_val,y_val))
```

```
1 model.evaluate(X_test,y_test)
```

125/125 [=====] - 0s 1ms/step - loss: 0.1745 - accuracy: 0.9415
[0.1744939237833023, 0.9415000081062317]



125/125 [=====] - 0s 975us/step
precision recall f1-score support

4	0.90	0.92	0.91	1000
5	0.97	0.97	0.97	1000
6	0.92	0.89	0.91	1000
7	0.97	0.97	0.97	1000
accuracy			0.94	4000
macro avg	0.94	0.94	0.94	4000
weighted avg	0.94	0.94	0.94	4000

Konfiguration 3 och resultat

Metod och resonemang kring att förbättra NN-arkitekturen:

För sista konfigurationen var målet att skapa ett NN som kunde förbättra tidigare modeller men även random forrest som hade en accuracy på 94%.

I nästa slide presenterar jag konfigurationen och resultatet där jag till slut **lyckades förbättra till 95%**.

Metoden jag använde var att successivt addera hidden layers och utöka antalet neuroner. Experimenterade även med Dropout, batchsize samt bytte även optimizer från sgd till adam efter jag på nätet såg en beskrivning av adam som en "förbättring" av sgd.

Det är svårt att sia om exakt vilka faktorer det är i arkitekturen som bidrar till förbättringen men en målsättning var att inte endast fokusera på att öka neuroner utan **fokusera på valideringssetet och kontrollera overfitting**. Att observera vid vilket antal träningsrundor modellen får en bra balans innan valideringsdatan försämrats eller nå en platå. Att experimentera med dropout och batchsize bidrar till att modellen hittar nya sätt att lära sig om datan och därmed förbättra generaliseringen, till skillnad om den hela tiden lär sig samma mönster. Dvs. att dropout stänger av vissa noder och batchsize styckar träningsdatan för varje epoch vilket gör att nätverket får utforska och lära sig andra "mönster" i datan

Task 5

Konfiguration 3 och resultat

```
In [194]: 1 # Eftersom att vi redan normaliserat datan när vi körde logistic regression kan vi ...
2 # mata in input shapen som den är utan att behöva flatta 28x28
3
4 model = Sequential()
5 model.add(Dense(100, input_shape=(784,), activation="relu"))
6 model.add(Dense(512, activation="relu"))
7 model.add(Dropout(0.3))
8 model.add(Dense(1024, activation="relu"))
9 model.add(Dropout(0.2))
10 model.add(Dense(2000, activation="relu"))
11 model.add(Dropout(0.2))
12 model.add(Dense(3200, activation="relu"))
13 model.add(Dropout(0.2))
14 model.add(Dense(10, activation="softmax"))
15
16 model.summary()
```

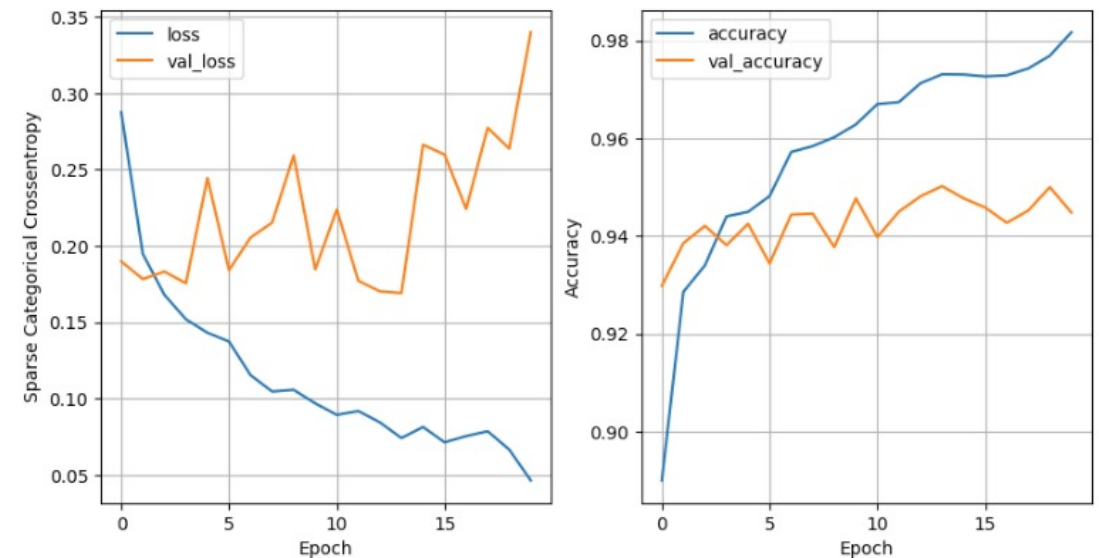
Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_32 (Dense)	(None, 100)	78500
dense_33 (Dense)	(None, 512)	51712
dropout_11 (Dropout)	(None, 512)	0
dense_34 (Dense)	(None, 1024)	525312
dropout_12 (Dropout)	(None, 1024)	0
dense_35 (Dense)	(None, 2000)	2050000
dropout_13 (Dropout)	(None, 2000)	0
dense_36 (Dense)	(None, 3200)	6403200
dropout_14 (Dropout)	(None, 3200)	0
dense_37 (Dense)	(None, 10)	32010

=====
Total params: 9140734 (34.87 MB)
Trainable params: 9140734 (34.87 MB)
Non-trainable params: 0 (0.00 Byte)

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

```
history=model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val,y_val))
```



	precision	recall	f1-score	support
4	0.89	0.95	0.92	1000
5	0.98	0.97	0.98	1000
6	0.94	0.88	0.91	1000
7	0.98	0.98	0.98	1000
accuracy			0.95	4000
macro avg	0.95	0.95	0.95	4000
weighted avg	0.95	0.95	0.95	4000

Task 5

För att vidare utvärdera vilken uppsättning av hyperparametrar som skulle generera en bättre modell skulle vi kunna ta en mer kvantitativ metod (*GRID-SEARCH*) istället för att godtyckligt välja hyperparametrarna. Fokuset blir ofta på att öka accuracy i modellen men vi bör lika mycket övervaka "valideringsloss" i modellen. För precis som med accuracy, om lossen på träningsdatan minskar men valideringsloss ökar så har vi också ett overfitting problem.

Under kursens gång (ett par veckor sedan tillämpade jag en GRID-search som fick jobba i över 24h där modellen fick testa en massa olika konfigurationer av hyperparametrar där jag lät python hålla koll på modellen med "least loss". Målet är ju att hitta en modellen som generaliserar bra till osedd data.

Analys: Trots att vi lyckades öka accuracy till 95% i den sista konfigurationen (föregående slide) kan vi se att Valideringsförlusten och förlusten på träningsdatan går åt diametrala håll ju längre tid vi tränar modellen. Valideringsaccuracy ser ut att ha en långsamare ökning. Men ju längre tiden går så börjar valideringsförlusten kraftigt öka. Det innebär att vid vidare konfigurering kan man försöka hitta en bättre balans. Möjligtvis reducera epochs till runt 11 där vi ser att valideringsförlusten är låg och valideringsaccuracy peakar. (*OBS. när vi försöker upprepa resultatet så kan vi givetvis inte garantera att grafen kommer upprepa sig på samma sätt*)

