Live TestExam 2025-11-10

November 10, 2025

1 Test Exam live 2025-11-10

1.1 Q3

[9]:
```
import pandas as pd
```

[10]:
```
X = pd.read_csv("../PMCM_WS2425/Data01.csv")
```

[11]:
```
X.head()
```

[11]:

	Unnamed: 0	f1	f2	f3	f4	target
0	0	2.470518	2.094533	7.464342	-0.898171	1
1	1	-1.290985	3.834506	3.713694	0.829274	0
2	2	1.131218	4.681950	6.135816	1.703968	-1
3	3	2.711527	1.489895	7.714867	-1.508245	1
4	4	2.442849	5.156339	7.453527	2.162256	-1

[6]:
```
X.target.unique()
```

[6]:
```
array([ 1,  0, -1])
```

[8]:
```
for t in X.target.unique():
    c = X.target[X.target == t].count()
    print(f"value {t} count {c}")
```

```
value 1 count 3000
value 0 count 6000
value -1 count 1000
```

The total count is 10000, 3000 instances with 1, 6000 with 0, 1000 with -1

1.2 3a

[12]:
```
X.min()
```

[12]:
```
Unnamed: 0    0.000000
f1           -5.973536
f2           -2.494636
```

```
f3          -0.961878
f4          -5.498820
target      -1.000000
dtype: float64
```

[13]:
```
X.max()
```

[13]:
```
Unnamed: 0    9999.000000
f1               5.856928
f2               7.063142
f3              10.852296
f4               4.071513
target           1.000000
dtype: float64
```

[14]:
```
X.max() - X.min()
```

[14]:
```
Unnamed: 0    9999.000000
f1              11.830464
f2               9.557779
f3              11.814174
f4               9.570333
target           2.000000
dtype: float64
```

The numerical range of the features is quite different (if not terribly so :-). Therefore we need scaling.

1.3 3b

[23]:
```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

[24]:
```
#scaler = StandardScaler()
scaler = MinMaxScaler()
```

[25]:
```
Xscaled = scaler.fit(X)
```

[26]:
```
Xscaled = pd.DataFrame(scaler.transform(X),columns=X.columns)
```

MinMaxScaler scales all features to same range. Fit trains Scaler, transform does the scaling.

[30]:
```
Xscaled.min()
```

[30]:
```
Unnamed: 0    0.0
f1            0.0
f2            0.0
f3            0.0
```

```
f4            0.0
target        0.0
dtype: float64
```

[31]:
```
Xscaled.max()
```

[31]:
```
Unnamed: 0    1.0
f1            1.0
f2            1.0
f3            1.0
f4            1.0
target        1.0
dtype: float64
```

[32]:
```
Xscaled.max()-Xscaled.min()
```

[32]:
```
Unnamed: 0    1.0
f1            1.0
f2            1.0
f3            1.0
f4            1.0
target        1.0
dtype: float64
```

As expected all features have the same range :-)

1.4 3d

[33]:
```
from sklearn.decomposition import PCA
```

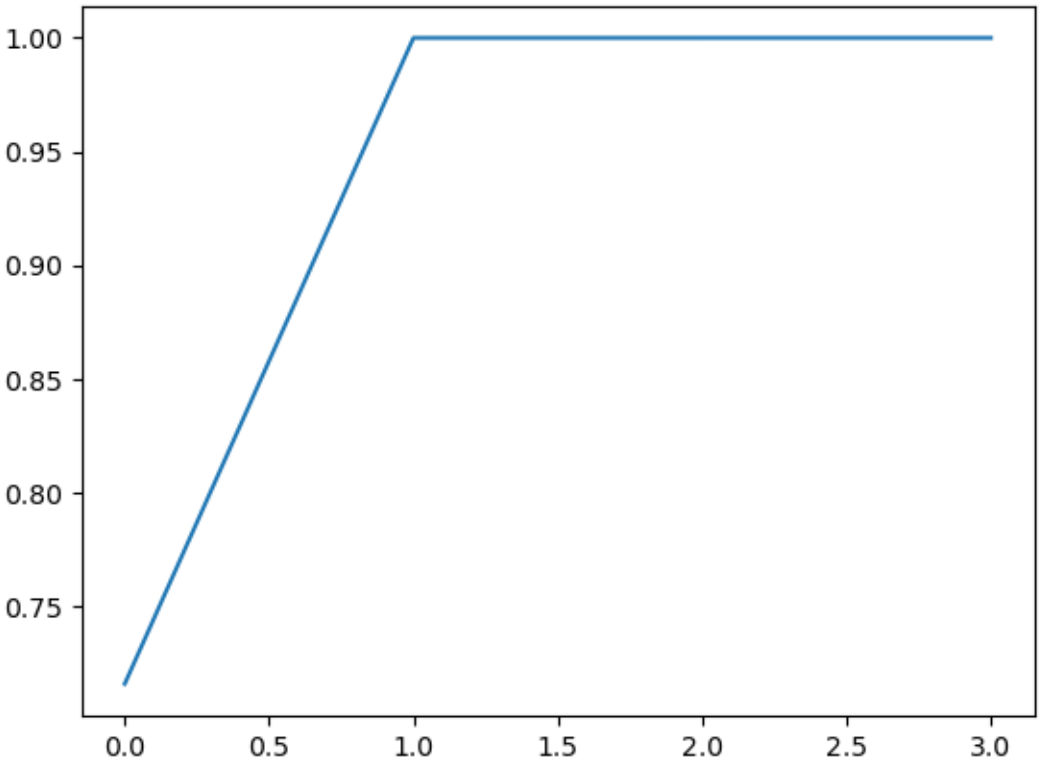
[34]:
```
myPca = PCA(n_components=4)
```

[ ]:
```
# Plot Cumsum
```

[40]:
```
import matplotlib.pyplot as plt

plt.plot(myPca.explained_variance_ratio_.cumsum())
```

[40]:
```
[<matplotlib.lines.Line2D at 0x7f6f92bb72b0>]
```



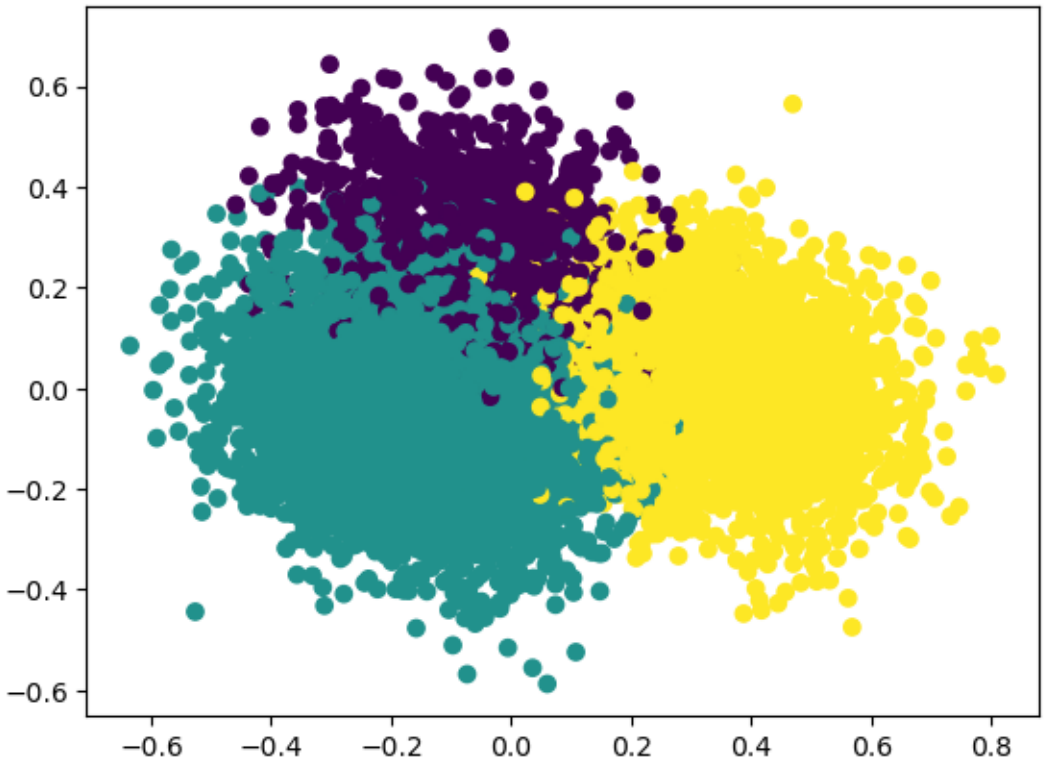
[38]:
```
myPca.fit(Xscaled[['f1', 'f2', 'f3', 'f4']])
```

[38]:
```
PCA(n_components=4)
```

[39]:
```
Xpca = myPca.transform(Xscaled[['f1', 'f2', 'f3', 'f4']])
```

[46]:
```
plt.scatter(Xpca[:,0],Xpca[:,1],c=X.target)
```

[46]:
```
<matplotlib.collections.PathCollection at 0x7f6f9108b460>
```



1.5 Q3e

From Q3: The total count is 10000, 3000 instances with 1, 6000 with 0, 1000 with -1

[47]:
```
#Create prediction vector with most frequent target value (0) for all instances

y_stupidPredictor = (X.target - X.target) + 0
```

[50]:
```
from sklearn.metrics import accuracy_score

accu = accuracy_score(X.target,y_stupidPredictor)

print(f"Most stupid predictor gives accuracy of {accu}")
```

```
Most stupid predictor gives accuracy of 0.6
```

[ ]:
```

```