PCA-Sensores-ReviewMarli-2D

August 13, 2022

[2]:
```
# 1. PCA dataset
import pandas as pd

df = pd.read_csv('../../mestrado/ReviewMarli/Sensor/allSensorsData.csv')
print(df)

# df['Sensor'] = df['Sensor'].astype('string')
df.head()
```

```
      Sensor  Eletrodo    Freq(Hz)     Z'(a)      Z''(b)  antiHIVmicrog/ml  \
0        1.0       6.0  1000000.00      72.6       11.30             0.001
1        1.0       6.0   794000.00      72.8        7.04             0.001
2        1.0       6.0   631000.00      72.9        3.16             0.001
3        1.0       6.0   501000.00      73.1       -0.51             0.001
4        1.0       6.0   398000.00      73.4       -4.17             0.001
…        …         …          …         …          …                 …
1825     2.0       2.0        2.51  281000.0  -660000.00             0.000
1826     2.0       2.0        2.00  347000.0  -778000.00             0.000
1827     2.0       2.0        1.58  425000.0  -912000.00             0.000
1828     2.0       2.0        1.26  519000.0 -1070000.00             0.000
1829     2.0       2.0        1.00  630000.0 -1260000.00             0.000

      antiHCVmicrog/ml
0                 0.01
1                 0.01
2                 0.01
3                 0.01
4                 0.01
…                 …
1825              0.10
1826              0.10
1827              0.10
1828              0.10
1829              0.10

[1830 rows x 7 columns]
```

[2]:
```
   Sensor  Eletrodo   Freq(Hz)  Z'(a)  Z''(b)  antiHIVmicrog/ml  \
0     1.0       6.0  1000000.0   72.6   11.30             0.001
1     1.0       6.0   794000.0   72.8    7.04             0.001
2     1.0       6.0   631000.0   72.9    3.16             0.001
3     1.0       6.0   501000.0   73.1   -0.51             0.001
4     1.0       6.0   398000.0   73.4   -4.17             0.001

   antiHCVmicrog/ml
0              0.01
1              0.01
2              0.01
3              0.01
4              0.01
```

[3]:
```
print(df.dtypes)
```

```
Sensor              float64
Eletrodo            float64
Freq(Hz)            float64
Z'(a)               float64
Z''(b)              float64
antiHIVmicrog/ml    float64
antiHCVmicrog/ml    float64
dtype: object
```

[4]:
```
df_features = df.iloc[:,1:].copy()
X = df_features.to_numpy()
# X = df.values
X
```

[4]:
```
array([[ 6.00e+00,  1.00e+06,  7.26e+01,  1.13e+01,  1.00e-03,  1.00e-02],
       [ 6.00e+00,  7.94e+05,  7.28e+01,  7.04e+00,  1.00e-03,  1.00e-02],
       [ 6.00e+00,  6.31e+05,  7.29e+01,  3.16e+00,  1.00e-03,  1.00e-02],
       …,
       [ 2.00e+00,  1.58e+00,  4.25e+05, -9.12e+05,  0.00e+00,  1.00e-01],
       [ 2.00e+00,  1.26e+00,  5.19e+05, -1.07e+06,  0.00e+00,  1.00e-01],
       [ 2.00e+00,  1.00e+00,  6.30e+05, -1.26e+06,  0.00e+00,  1.00e-01]])
```

[5]:
```
Y = df.iloc[:, 0].to_numpy()
Y
```

[5]:
```
array([1., 1., 1., …, 2., 2., 2.])
```

[6]:
```
X.shape
```

[6]:
```
(1830, 6)
```

[7]:
```
Y.shape
```

[7]:
```
(1830,)
```

[8]:
```
# 2. PCA analysis
# 2.1 Load library

from sklearn.preprocessing import scale # Data scaling
from sklearn import decomposition # PCA
import pandas as pd # pandas
```

[9]:
```
# 2.2 Data scaling
x = scale(X)

# Standardize the Data
# PCA is effected by scale so you need to scale the features in your data␣
 ↪before applying PCA. Use StandardScaler to help you standardize the␣
 ↪dataset's features onto unit scale (mean = 0 and variance = 1) which is a␣
 ↪requirement for the optimal performance of many machine learning algorithms.
x
```

[9]:
```
array([[ 2.62202212,  4.71762672, -0.38241265,  0.35464826, -0.44958185,
        -0.39013907],
       [ 2.62202212,  3.66164406, -0.38241034,  0.35463491, -0.44958185,
        -0.39013907],
       [ 2.62202212,  2.82608496, -0.38240919,  0.35462276, -0.44958185,
        -0.39013907],
       …,
       [-0.23836565, -0.4084946 ,  4.51211975, -2.50287873, -0.4525691 ,
         1.17273259],
       [-0.23836565, -0.40849624,  5.59486011, -2.99792661, -0.4525691 ,
         1.17273259],
       [-0.23836565, -0.40849758,  6.87341521, -3.59323736, -0.4525691 ,
         1.17273259]])
```

[11]:
```
# 2.3 Perform PCA analysis
pca = decomposition.PCA(n_components=2)
pca.fit(x) # when build the model we use pca.fit function. Where by the␣
 ↪argument will be the input data, which essencially is the x variable
# we are using the x variable because pca is an unsupervised learning approach,␣
 ↪meaning that it does not need the y variable or the class label in order to␣
 ↪learn
# it`ll goind to cluster the data based on similarity and differences, based on␣
 ↪the eigenvalue that are inherently present in the data set
```

[11]:
```
PCA(n_components=2)
```

[12]:
```
# 2.4 compure the scores value
# scores value will essentially be represented by the data samples so we`re␣
 ↪gonna use the pca.transform function
scores = pca.transform(x)
```

[13]:
```
scores
```

[13]:
```
array([[-2.11922391,  0.2748272 ],
       [-1.86599962,  0.40940752],
       [-1.66563223,  0.51589593],
       …,
       [ 4.99762954, -0.21307917],
       [ 6.05106421, -0.01729866],
       [ 7.30213618,  0.21516028]])
```

[14]:
```
scores_df = pd.DataFrame(scores, columns=['PC1', 'PC2']) # dataframe is to make␣
 ↪more readable
scores_df
```

[14]:

	PC1	PC2
0	-2.119224	0.274827
1	-1.866000	0.409408
2	-1.665632	0.515896
3	-1.505828	0.600826
4	-1.379211	0.668117
…	…	…
1825	3.363152	-0.516694
1826	4.117486	-0.376609
1827	4.997630	-0.213079
1828	6.051064	-0.017299
1829	7.302136	0.215160

[1830 rows x 2 columns]

[15]:
```
y_label = []

for i in Y:
    if i == 1:
        y_label.append('HCV')
    else:
        y_label.append('HIV')

sensors = pd.DataFrame(y_label, columns=['Sensor'])
```

[16]:
```
df_scores = pd.concat([scores_df, sensors], axis=1) # combine dataframes with␣
 ↪concat
df_scores
```

[16]:
```
            PC1       PC2 Sensor
0     -2.119224  0.274827    HCV
1     -1.866000  0.409408    HCV
2     -1.665632  0.515896    HCV
3     -1.505828  0.600826    HCV
4     -1.379211  0.668117    HCV
…           …         …      …
1825   3.363152 -0.516694    HIV
1826   4.117486 -0.376609    HIV
1827   4.997630 -0.213079    HIV
1828   6.051064 -0.017299    HIV
1829   7.302136  0.215160    HIV

[1830 rows x 3 columns]
```

[18]:
```
feature_names = df.columns[1:]
feature_names
```

[18]:
```
Index(['Eletrodo', 'Freq(Hz)', 'Z'(a)', 'Z''(b)', 'antiHIVmicrog/ml',
       'antiHCVmicrog/ml'],
      dtype='object')
```

[19]:
```
# 2.5 retrieve the loading values (remember PCA scores and loadings)
# loadings value tell about the descriptor and scores value tell about the data␣
 ↪samples
# 150 flowers 150 score values 4 descriptors 4 loading values (1 for each␣
 ↪descriptor)
loadings = pca.components_.T
df_loadings = pd.DataFrame(loadings, columns=['PC1', 'PC2'],␣
 ↪index=feature_names)
df_loadings
```

[19]:
```
                        PC1       PC2
Eletrodo          -0.184445  0.379189
Freq(Hz)          -0.239790 -0.127444
Z'(a)              0.669512  0.126659
Z''(b)            -0.663627 -0.118457
antiHIVmicrog/ml  -0.074718  0.624234
antiHCVmicrog/ml   0.119381 -0.648253
```

[20]:
```
# 2.6 explained variance
# what`s the contribuition to the percent variance of the entire model␣
 ↪contributed by each of the principal components
explained_variance = pca.explained_variance_ratio_
explained_variance
```

[20]:
```
array([0.30596482, 0.21490754])
```

[21]:
```
# 3. Scree plot
import numpy as np
import plotly.express as px
```

[22]:
```
# 3.1 preparing the explained variance data
# add origin value, x and y are going to have origin zero, the subsequent lines␣
 ↪of code are going to create the scree plot
# the scree plot does not start from zero, then we need to manually create the␣
 ↪zero origin
explained_variance = np.insert(explained_variance, 0, 0)
explained_variance
```

[22]:
```
array([0.        , 0.30596482, 0.21490754])
```

[23]:
```
# 3.2 Preparing cumulative variance data
cumulative_variance = np.cumsum(np.round(explained_variance, decimals=3))
```

[26]:
```
# 3.3 Combining dataframe
pc_df = pd.DataFrame(['', 'PC1', 'PC2'], columns=['PC'])
explained_variance_df = pd.DataFrame(explained_variance, columns=['Explained␣
 ↪Variance'])
cumulative_variance_df = pd.DataFrame(cumulative_variance, columns=['Cumulative␣
 ↪Variance'])
```

[27]:
```
df_explained_variance = pd.concat([pc_df, explained_variance_df,␣
 ↪cumulative_variance_df], axis=1)
df_explained_variance
```

[27]:

	PC	Explained Variance	Cumulative Variance
0		0.000000	0.000
1	PC1	0.305965	0.306
2	PC2	0.214908	0.521

[28]:
```
# 3.4 Creating Scree Plot
# https://plotly.com/python/bar-charts/

fig = px.bar(df_explained_variance, x='PC', y='Explained Variance',␣
 ↪text='Explained Variance', width=800)
```

[29]:
```
fig.update_traces(texttemplate='%{text:.3f}', textposition='outside') # limit␣
 ↪decimal cases and text outside the bar
fig.show() #explained variance
```

[30]:
```
# explained variance + cumulative variance
# https://plotly.com/python/creating-and-updating-figures

import plotly.graph_objects as go
```

```
fig = go.Figure()

fig.add_trace(
    go.Scatter(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Cumulative Variance'],
        marker=dict(size=15, color='LightSeaGreen')
    ))

fig.add_trace(
    go.Bar(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Explained Variance'],
        marker=dict(color='RoyalBlue')
    ))

fig.show()
```

[31]:
```
# Explained variance + cumulative variance (Separate Plot)

from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig = make_subplots(rows=1, cols=2)

fig.add_trace(
    go.Scatter(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Cumulative Variance'],
        marker=dict(size=15, color='LightSeaGreen')
    ), row=1, col=1
)

fig.add_trace(
    go.Bar(
        x=df_explained_variance['PC'],
        y=df_explained_variance['Explained Variance'],
        marker=dict(color='RoyalBlue')
    ), row=1, col=2
)

fig.show()
```

[40]:
```
# 4. Scores plot
# check API documentation for plotly.express https://plotly.com/python/
 ↪3d-scatter-plots
```

```
import plotly.express as px

fig = px.scatter(df_scores, x='PC1', y='PC2', color='Sensor')
fig.show()
```

[41]:
```
# 4.1 Customize 3D Scatter Plot

fig = px.scatter(df_scores, x='PC1', y='PC2', color='Sensor', symbol='Sensor',␣
↪opacity=0.5)

# tight layout
fig.update_layout(margin=dict(l=0, r=0, b=0, t=0)) # left, right, bottom, top␣
↪with zero margin

fig.show()

# https://plotly.com/python/templates/
# fig.update_layout(template='plotly_white')
# plotly, plotly_white, plotly_dark, ggplot2, seaborn, simple_white, none
```

[42]:
```
# 5. Loadings Plot

loadings_label = df_loadings.index
# loadings_label = df_loadings.index.str.strip(' (cm)')

fig = px.scatter(df_loadings, x='PC1', y='PC2', text=loadings_label)
fig.show()
```

[43]:
```
loadings_label
```

[43]:
```
Index(['Eletrodo', 'Freq(Hz)', 'Z'(a)', 'Z''(b)', 'antiHIVmicrog/ml',
       'antiHCVmicrog/ml'],
      dtype='object')
```

[44]:
```
df_loadings
```

[44]:

	PC1	PC2
Eletrodo	-0.184445	0.379189
Freq(Hz)	-0.239790	-0.127444
Z'(a)	0.669512	0.126659
Z''(b)	-0.663627	-0.118457
antiHIVmicrog/ml	-0.074718	0.624234
antiHCVmicrog/ml	0.119381	-0.648253

[ ]:

[ ]: