PCA-Sem-PBS-Eletrodos-HCV-ReviewMarli

August 13, 2022

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

df = pd.read_csv('../../mestrado/ReviewMarli/Eletrodo/allHCVData.csv')
print(df)

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

```
     Eletrodo    Freq(Hz)     Z'(a)     Z''(b)  antiHIVmicrog/ml  \
0         1.0  1000000.00     406.0      18.50            0.0000
1         1.0   794000.00     408.0       8.96            0.0000
2         1.0   631000.00     410.0       0.38            0.0000
3         1.0   501000.00     412.0      -7.90            0.0000
4         1.0   398000.00     414.0     -16.30            0.0000
..        …           …         …          …                …
788       3.0        2.51  264000.0 -466000.00            0.0001
789       3.0        2.00  311000.0 -538000.00            0.0001
790       3.0        1.58  362000.0 -619000.00            0.0001
791       3.0        1.26  424000.0 -716000.00            0.0001
792       3.0        1.00  486000.0 -823000.00            0.0001

     antiHCVmicrog/ml
0               0.002
1               0.002
2               0.002
3               0.002
4               0.002
..                …
788             0.002
789             0.002
790             0.002
791             0.002
792             0.002

[793 rows x 6 columns]
```

```
[1]:    Eletrodo   Freq(Hz)  Z'(a)  Z''(b)  antiHIVmicrog/ml  antiHCVmicrog/ml
     0       1.0  1000000.0  406.0   18.50               0.0             0.002
     1       1.0   794000.0  408.0    8.96               0.0             0.002
     2       1.0   631000.0  410.0    0.38               0.0             0.002
     3       1.0   501000.0  412.0   -7.90               0.0             0.002
     4       1.0   398000.0  414.0  -16.30               0.0             0.002
```

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

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

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

```
[3]: array([[ 1.00e+06,  4.06e+02,  1.85e+01,  0.00e+00,  2.00e-03],
            [ 7.94e+05,  4.08e+02,  8.96e+00,  0.00e+00,  2.00e-03],
            [ 6.31e+05,  4.10e+02,  3.80e-01,  0.00e+00,  2.00e-03],
            …,
            [ 1.58e+00,  3.62e+05, -6.19e+05,  1.00e-04,  2.00e-03],
            [ 1.26e+00,  4.24e+05, -7.16e+05,  1.00e-04,  2.00e-03],
            [ 1.00e+00,  4.86e+05, -8.23e+05,  1.00e-04,  2.00e-03]])
```

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

```
[4]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
            1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.])
```

[5]:
```
X.shape
```

[5]:
```
(793, 5)
```

[6]:
```
Y.shape
```

[6]:
```
(793,)
```

[7]:
```
# 2. PCA analysis
# 2.1 Load library

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

```
import pandas as pd # pandas
```

```
[8]: # 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
```

```
[8]: array([[ 4.71762672, -0.40381199,  0.43445191, -0.28867513, -0.9258201 ],
       [ 3.66164406, -0.40379185,  0.43442424, -0.28867513, -0.9258201 ],
       [ 2.82608496, -0.40377172,  0.43439936, -0.28867513, -0.9258201 ],
       …,
       [-0.4084946 ,  3.23658101, -1.36074157,  3.46410162, -0.9258201 ],
       [-0.40849624,  3.86077379, -1.64204781,  3.46410162, -0.9258201 ],
       [-0.40849758,  4.48496658, -1.95235469,  3.46410162, -0.9258201 ]])
```

```
[9]: # 2.3 Perform PCA analysis
pca = decomposition.PCA(n_components=3)
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
```

```
[9]: PCA(n_components=3)
```

```
[10]: # 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)
```

```
[11]: scores
```

```
[11]: array([[-1.78531263,  0.29003729,  4.40850899],
       [-1.51990227,  0.32094161,  3.38705653],
       [-1.30988818,  0.34539524,  2.57882116],
       …,
       [ 3.13720086,  3.27817358,  0.52989059],
       [ 3.7558864 ,  3.30941781,  0.69409369],
       [ 4.39444356,  3.34053706,  0.86322592]])
```

[12]:
```
scores_df = pd.DataFrame(scores, columns=['PC1', 'PC2', 'PC3']) # dataframe is␣
↪to make more readable
scores_df
```

[12]:
```
          PC1       PC2       PC3
0   -1.785313  0.290037  4.408509
1   -1.519902  0.320942  3.387057
2   -1.309888  0.345395  2.578821
3   -1.142387  0.364898  1.934218
4   -1.009667  0.380351  1.423495
..        ...       ...       ...
788  2.159919  3.228783  0.270505
789  2.625878  3.252488  0.394225
790  3.137201  3.278174  0.529891
791  3.755886  3.309418  0.694094
792  4.394444  3.340537  0.863226

[793 rows x 3 columns]
```

[13]:
```
y_label = []

for i in Y:
    if i == 1:
        y_label.append('SF_NS5A_1bic')
    elif i == 2:
        y_label.append('SF_NS5A_5bic')
    else:
        y_label.append('SF_NS5A_5bicp24')

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

[14]:
```
df_scores = pd.concat([scores_df, sensors], axis=1) # combine dataframes with␣
↪concat
df_scores
```

[14]:
```
          PC1       PC2       PC3           Sensor
0   -1.785313  0.290037  4.408509     SF_NS5A_1bic
1   -1.519902  0.320942  3.387057     SF_NS5A_1bic
2   -1.309888  0.345395  2.578821     SF_NS5A_1bic
3   -1.142387  0.364898  1.934218     SF_NS5A_1bic
4   -1.009667  0.380351  1.423495     SF_NS5A_1bic
..        ...       ...       ...              ...
788  2.159919  3.228783  0.270505  SF_NS5A_5bicp24
789  2.625878  3.252488  0.394225  SF_NS5A_5bicp24
790  3.137201  3.278174  0.529891  SF_NS5A_5bicp24
791  3.755886  3.309418  0.694094  SF_NS5A_5bicp24
792  4.394444  3.340537  0.863226  SF_NS5A_5bicp24
```

```
[793 rows x 4 columns]
```

[15]:
```
feature_names = df.columns[1:]
feature_names
```

[15]:
```
Index(['Freq(Hz)', 'Z'(a)', 'Z''(b)', 'antiHIVmicrog/ml', 'antiHCVmicrog/ml'],
dtype='object')
```

[16]:
```
# 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', 'PC3'],␣
↪index=feature_names)
df_loadings
```

[16]:
```
                       PC1       PC2       PC3
Freq(Hz)         -0.251309 -0.029265  0.967308
Z'(a)             0.682369  0.051997  0.186473
Z''(b)           -0.685216  0.004309 -0.169956
antiHIVmicrog/ml -0.021294  0.707893  0.025577
antiHCVmicrog/ml  0.035252 -0.703782 -0.001761
```

[17]:
```
# 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
```

[17]:
```
array([0.3866742 , 0.25362993, 0.18738105])
```

[18]:
```
# 3. Scree plot
import numpy as np
import plotly.express as px
```

[19]:
```
# 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
```

[19]:
```
array([0.        , 0.3866742 , 0.25362993, 0.18738105])
```

[20]:
```python
# 3.2 Preparing cumulative variance data
cumulative_variance = np.cumsum(np.round(explained_variance, decimals=3))
```

[21]:
```python
# 3.3 Combining dataframe
pc_df = pd.DataFrame(['', 'PC1', 'PC2', 'PC3'], columns=['PC'])
explained_variance_df = pd.DataFrame(explained_variance, columns=['Explained␣
 ↪Variance'])
cumulative_variance_df = pd.DataFrame(cumulative_variance, columns=['Cumulative␣
 ↪Variance'])
```

[22]:
```python
df_explained_variance = pd.concat([pc_df, explained_variance_df,␣
 ↪cumulative_variance_df], axis=1)
df_explained_variance
```

[22]:
```
    PC  Explained Variance  Cumulative Variance
0                 0.000000                0.000
1  PC1            0.386674                0.387
2  PC2            0.253630                0.641
3  PC3            0.187381                0.828
```

[23]:
```python
# 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)
```

[24]:
```python
fig.update_traces(texttemplate='%{text:.3f}', textposition='outside') # limit␣
 ↪decimal cases and text outside the bar
fig.show() #explained variance
```

[1]:
```python
# 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()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_49296/3253997947.py in <module>
      7 fig.add_trace(
      8     go.Scatter(
----> 9         x=df_explained_variance['PC'],
     10         y=df_explained_variance['Cumulative Variance'],
     11         marker=dict(size=15, color='LightSeaGreen')

NameError: name 'df_explained_variance' is not defined
```

[26]:
```
# 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()
```

[27]:
```
# 4. Scores plot
# check API documentation for plotly.express https://plotly.com/python/
↪3d-scatter-plots

import plotly.express as px
```

```
fig = px.scatter_3d(df_scores, x='PC1', y='PC2', z='PC3', color='Sensor')
fig.show()
```

[28]:
```
# 4.1 Customize 3D Scatter Plot

fig = px.scatter_3d(df_scores, x='PC1', y='PC2', z='PC3', color='Sensor',␣
 ↪symbol='Sensor', opacity=0.4)

# 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
```

[29]:
```
# 5. Loadings Plot

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

fig = px.scatter_3d(df_loadings, x='PC1', y='PC2', z='PC3', text=loadings_label)
fig.show()
```

[30]:
```
loadings_label
```

[30]:
```
Index(['Freq(Hz)', 'Z'(a)', 'Z''(b)', 'antiHIVmicrog/ml', 'antiHCVmicrog/ml'],
dtype='object')
```

[31]:
```
df_loadings
```

[31]:
```
                       PC1       PC2       PC3
Freq(Hz)         -0.251309 -0.029265  0.967308
Z'(a)             0.682369  0.051997  0.186473
Z''(b)           -0.685216  0.004309 -0.169956
antiHIVmicrog/ml -0.021294  0.707893  0.025577
antiHCVmicrog/ml  0.035252 -0.703782 -0.001761
```

[ ]:
```

```