GENERALMENTE

-Luego de hacer el fit , se hace un predict

-El accuracy se calcula comparando (y\_test,y\_predic).

import pandas as pd

import sklearn

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.decomposition import IncrementalPCA

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import jaccard\_score

#PREPARE DATA

#============

dt\_heart = pd.read\_csv('./data/heart.csv')

#print(dt\_heart.head(5))

# axis=1 columnas, copie sin columna target

dt\_features = dt\_heart.drop('target', axis =1)

# DataFrame solo con target, si tiene o no problemas cardiacos

dt\_target = dt\_heart['target']

# Normalizando los datos ???? PARA QUE????

dt\_features = StandardScaler().fit\_transform(dt\_features)

# Get train and test sets, ENTREME DATOS

X\_train,X\_test, y\_train, y\_test = train\_test\_split(dt\_features,dt\_target, test\_size=0.3, random\_state=42)

# Default si no asigna n\_components = min(n\_columnas, n\_filas)

# Entrenaremos con 3 componentes de los 13 que tiene

# Que PCA evalue cuales son los mas relevantes

N1 = 3

pca = PCA(n\_components=N1)

pca.fit(X\_train)

# Batch 10 es con bloques de 10 muestras a la vez

ipca = IncrementalPCA(n\_components=N1, batch\_size=10)

ipca.fit(X\_train)

# Grafica mostrara aporte de cada uno de los 3 features

plt.plot(range(len(pca.explained\_variance\_)),

pca.explained\_variance\_ratio\_)

plt.show()

|  |  |  |
| --- | --- | --- |
| n\_components = 3 (N1) | n\_componentes = 5 | n\_components=9 |
|  |  |  |

NOTAMOS QUE:

------------

CON 3 EL TERCERO APORTA POCO, 0.10

CON 9 EL 8,9 APORTAN 0.06

LOS MAS IMPORTANTES SON LOS 2 PRIMEROS

PERO TENIENDO MAS COMPONENTES AUMENTA EXACTITUD

logistic = LogisticRegression(solver='lbfgs')

dt\_train = pca.transform(X\_train)

dt\_test = pca.transform(X\_test)

logistic.fit(dt\_train, y\_train)

print("Score PCA: ", logistic.score(dt\_test, y\_test))

dt\_train = ipca.transform(X\_train)

dt\_test = ipca.transform(X\_test)

logistic.fit(dt\_train, y\_train)

print("Score IPCA ", logistic.score(dt\_test, y\_test))

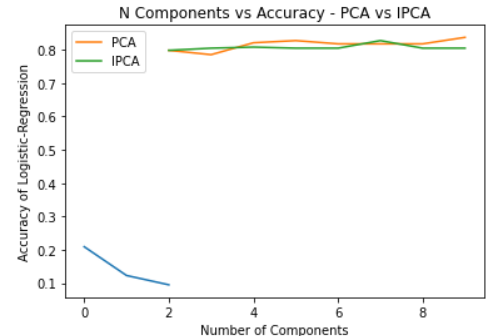
Score PCA: 0.7857142857142857

Score IPCA 0.8051948051948052

**Comparación entre PCA e IPCA**  
Variando numero de componentes (2-9) varia la exactitud

Previamente, si le decimos a PCA que solo queremos 3 componentes nos da exactitude de .80 y .78, segun usaramos PCA o IPCA

Cuando lo hacemos con mas componentes no cambia la exactitude



# GRAFICA COMPARATIVA

logistic = LogisticRegression(solver='lbfgs')

pca\_data = {'accuracy': [],'n\_components': []}

ipca\_data = {'accuracy': [],'n\_components': []}

for n in range(2, 10):

    #PCA

    pca = PCA(n\_components=n)

    pca.fit(X\_train)

    #IPCA

    # Batch 10 es con bloques de 10 muestras a la vez

    ipca = IncrementalPCA(n\_components=n, batch\_size=10)

    ipca.fit(X\_train)

    #Train PCA

    df\_train = pca.transform(X\_train)

    df\_test = pca.transform(X\_test)

    logistic.fit(df\_train, y\_train)

    acccuracy = logistic.score(df\_test, y\_test)

    pca\_data['accuracy'].append(acccuracy)

    pca\_data['n\_components'].append(n)

    # TRAIN IPCA

    df\_train = ipca.transform(X\_train)

    df\_test = ipca.transform(X\_test)

    logistic.fit(df\_train, y\_train)

    acccuracy = logistic.score(df\_test, y\_test)

    ipca\_data['accuracy'].append(acccuracy)

    ipca\_data['n\_components'].append(n)

plt.plot(pca\_data['n\_components'], pca\_data['accuracy'], label='PCA')

plt.plot(ipca\_data['n\_components'], ipca\_data['accuracy'], label='IPCA')

plt.title('N Components vs Accuracy - PCA vs IPCA')

plt.xlabel('Number of Components')

plt.ylabel('Accuracy of Logistic-Regression')

plt.legend()

plt.show()