We examine the UCI Machine Learning Repository Epileptic Seizure Data Set (attached) and Python to develop PCA in order to show its usefulness. 11,500 instances and 178 features (X1 through X178) make up the data set. Every instance in the feature reflects a patient's brain activity measurements across a one-second interval, and every feature represents a distinct measurement point for an EEG equipment attached to the patient. If the measurements were taken during a seizure, the response variable (y) has a value of 1, and if not, it has a value of 2, 3, 4, or 5.

* In order to forecast if each instance was recorded during a seizure, we will first break down the data set into its major components for this exercise. I will then utilize those components as input to a k-Nearest Neighbors (k-NN) method. In this particular scenario, I will also attempt to ascertain the ideal number of principal components (PCs) to use using k-NN.

|  |
| --- |
| Import numpy as np |
|  | import pandas as pd |
|  |  |
|  | # Visualization |
|  | import matplotlib.pyplot as plt |
|  |  |
|  | # Data processing, modeling, and model evaluation |
|  | from sklearn.preprocessing import StandardScaler |
|  | from sklearn.decomposition import PCA |
|  | from sklearn.neighbors import KNeighborsClassifier |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn.metrics import f1\_score, classification\_report, ConfusionMatrixDisplay |
|  |  |
|  | # Randomization |
|  | import random |

* Load the data

The data collection consists of 178 features labeled X1 through X178 (indicating EEG measurement points), one feature called "Unnamed" that appears to be an ID column, and a response variable called "y" that accepts integer values between 1 and 5.

Drop the ‘Unnamed’ feature

Re-cast target variable ‘y’ as binary, where 1 represents a seizure event, and all other numbers are 0 (no seizure).

Check for and fill any null or missing values.

* It's crucial to normalize or standardize features before executing the PCA transformation since PCA aims to optimize the variance reflected in each subsequent main component. If not, a portion of the variation found by PCA might just be the result of the size of the individual variables.
* To standardize the data in this instance, we employ StandardScaler from sklearn.
* PCA can be used now that the data have been scaled suitably. Those who often use sklearn machine learning methods would recognize that we achieve this using sklearn's PCA. Since there can never be more primary components than the features in the original data set, we set n\_components = 178 in this case.
* Now, let’s see how much of the variance in the data set can be explained by the principal components. To do this, we can make use of the .explained\_variance\_ratio attribute of the pca object. From the obtained graph, we can see that virtually all of the variance in the data set can be explained by the first 50 principal components. That’s less than 1/3 of the original number of features!
* The principal components (PCs) are then fed into a k-NN classifier as the X input. But as the number of features rises, k-NN rapidly succumbs to the curse of dimensionality, thus we are unsure exactly how many of these PCs to employ. We will therefore attempt to use a range of one to twenty PCs in order to observe the impact on categorization performance.
* To assess each k-NN classifier's efficacy we use the f1\_score as the classification metric. When classes are roughly balanced, accuracy is more appropriate. However, the f\_1 score is more resilient to class imbalance.
* The best f1\_score is achieved using the first four PCs — this seems to be the sweet spot for providing enough information for the k-NN algorithm while not overloading k-NN with extra features.
* Explore how the k-NN classifier with four PCs performs in a little more detail. Calculate accuracy, f1 score, recall etc. Plot a confusion matrix.