**CS6301 MACHINE LEARNING LAB WEEK – 12 GMM and ICA**

**SRIHARI. S – 2018103601**

**Date**: 03-05-2021 Monday

**Aim**: To implement Independent Component Analysis and Gaussian Mixture Model.

**Gaussian mixture model (GMM) :**

* GMM is also a type of clustering algorithm.
* Each cluster is modelled according to a different Gaussian distribution.
* This flexible and probabilistic approach to modelling the data means that rather than having hard assignments into clusters like k-means, we have soft assignments.

**Algorithm:**

• Write N datapoints xi = (x1i, x2i, . . . , xMi) as row vectors

• Put these vectors into a matrix X (which will have size N ×M)

• Centre the data by subtracting off the mean of each column, putting it into matrix B

• Compute the covariance matrix C = 1/N BT B

• Compute the eigenvalues and eigenvectors of C, so V-1 CV = D, where V holds the eigenvectors of C and D is the M ×M diagonal eigenvalue matrix.

• Sort the columns of D into order of decreasing eigenvalues, and apply the same order to the columns of V

• Reject those with eigenvalue less than some n (eta), leaving L dimensions in the data

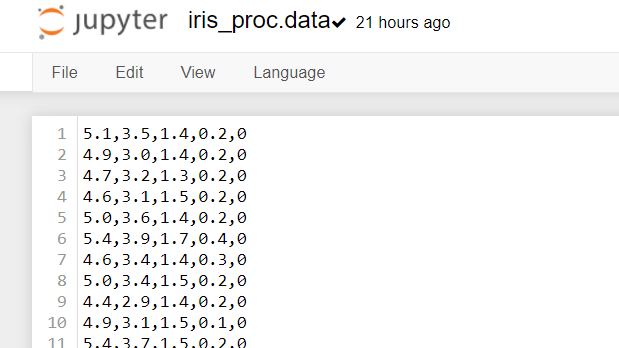
**Dataset : Iris**

**Url**: <https://archive.ics.uci.edu/ml/datasets/iris>

**Description**: The **Iris Dataset** contains four features (length and width of sepals and petals) of 50 samples of three species of **Iris** (**Iris** setosa, **Iris** virginica and **Iris** versicolor).

**Input**: The following 4 attributes

* sepal length in cm,
* sepal width in cm,
* petal length in cm,
* petal width in cm,



**Code:**

import pandas as pd

from GMM import \*

import util as plot

from matplotlib import pyplot as plt

def For\_Iris(features,No\_Component=2):

data = pd.read\_csv("Data/Iris.csv", header = 0)

data = data.reset\_index()

replace\_map = {'Species': {'Iris-virginica': 1, 'Iris-versicolor': 2,'Iris-setosa':3}}

data.replace(replace\_map, inplace=True)

label=data[['Species']]

col=['SepalLengthCm','SepalWidthCm']

x=data[col]

x=np.array(x)

gmm = GaussianMixModel(x,No\_Component)

gmm.fit()

plot.plot\_2D(gmm,x,col,label)

def main():

For\_Iris(2,3)

if \_\_name\_\_== "\_\_main\_\_":

main()

import numpy as np

import scipy.stats as sp

class GaussianMixModel(object):

def \_\_init\_\_(self, X, k=2):

X = np.asarray(X)

self.m, self.n = X.shape

self.data = X.copy()

print (np.mean(X))

self.k = k

def \_init(self):

self.mean\_arr = np.asmatrix(np.random.random((self.k, self.n))+np.mean(self.data))

self.sigma\_arr = np.array([np.asmatrix(np.identity(self.n)) for i in range(self.k)])

self.phi = np.ones(self.k)/self.k

self.Z = np.asmatrix(np.empty((self.m, self.k), dtype=float))

def fit(self, tol=1e-4):

self.\_init()

num\_iters = 0

logl = 1

previous\_logl = 0

while(logl-previous\_logl > tol):

previous\_logl = self.loglikelihood()

self.e\_step()

self.m\_step()

num\_iters += 1

logl = self.loglikelihood()

print('Iteration %d: log-likelihood is %.6f'%(num\_iters, logl))

print('Terminate at %d-th iteration:log-likelihood is %.6f'%(num\_iters, logl))

def loglikelihood(self):

logl = 0

for i in range(self.m):

tmp = 0

for j in range(self.k):

tmp += sp.multivariate\_normal.pdf(self.data[i, :],self.mean\_arr[j, :].A1,self.sigma\_arr[j, :]) \* self.phi[j]

logl += np.log(tmp)

return logl

def e\_step(self):

for i in range(self.m):

den = 0

for j in range(self.k):

num = sp.multivariate\_normal.pdf(self.data[i, :],

self.mean\_arr[j].A1,

self.sigma\_arr[j]) \*\

self.phi[j]

den += num

self.Z[i, j] = num

self.Z[i, :] /= den

assert self.Z[i, :].sum() - 1 < 1e-4 # Program stop if this condition is false

def m\_step(self):

for j in range(self.k):

const = self.Z[:, j].sum()

self.phi[j] = 1/self.m \* const

\_mu\_j = np.zeros(self.n)

\_sigma\_j = np.zeros((self.n, self.n))

for i in range(self.m):

\_mu\_j += (self.data[i, :] \* self.Z[i, j])

\_sigma\_j += self.Z[i, j] \* ((self.data[i, :] - self.mean\_arr[j, :]).T \* (self.data[i, :] - self.mean\_arr[j, :]))

self.mean\_arr[j] = \_mu\_j / const

self.sigma\_arr[j] = \_sigma\_j / const

import matplotlib.pyplot as plt

import numpy as np

import scipy.stats as sp

import matplotlib as mpl

import pandas as pd

def make\_ellipses(gmm, ax):

colors = ['turquoise', 'orange']

for n, color in enumerate(colors):

covariances = gmm.sigma\_arr[n]

v, w = np.linalg.eigh(covariances)

u = w[0] / np.linalg.norm(w[0])

angle = np.arctan2(u[1], u[0])

angle = 180 \* angle / np.pi # convert to degrees

v = 3. \* np.sqrt(2.) \* np.sqrt(v)

mean=gmm.mean\_arr[n]

mean=mean.reshape(2,1)

print(mean)

ell = mpl.patches.Ellipse(mean, v[0], v[1],

180 + angle, color=color)

ell.set\_clip\_box(ax.bbox)

ell.set\_alpha(0.5)

ax.add\_artist(ell)

ax.set\_aspect('equal', 'datalim')

def plot\_2D(gmm,x,col,label):

h = plt.subplot(111, aspect='equal')

make\_ellipses(gmm, h)

plt.scatter(x[:,0],x[:,1],c=label['Species'],marker='x')

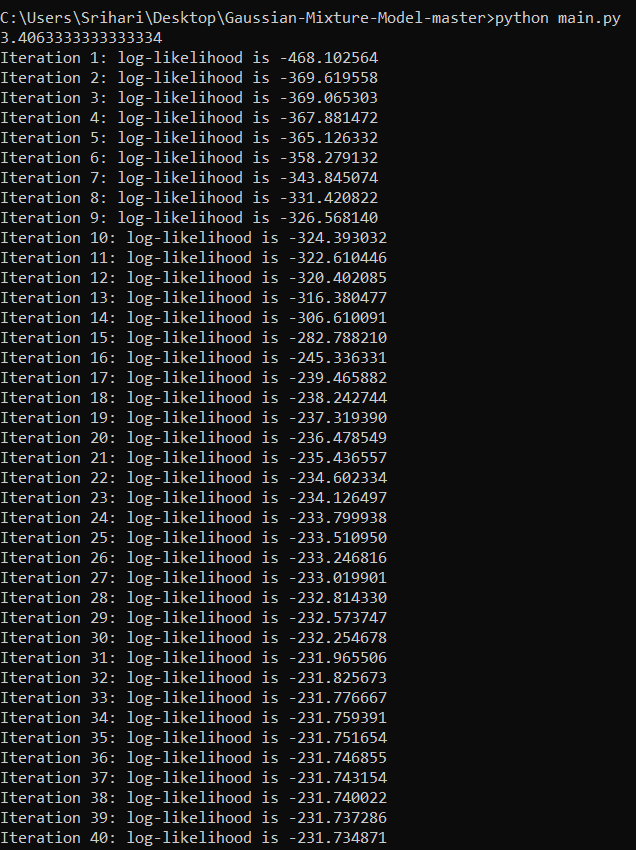
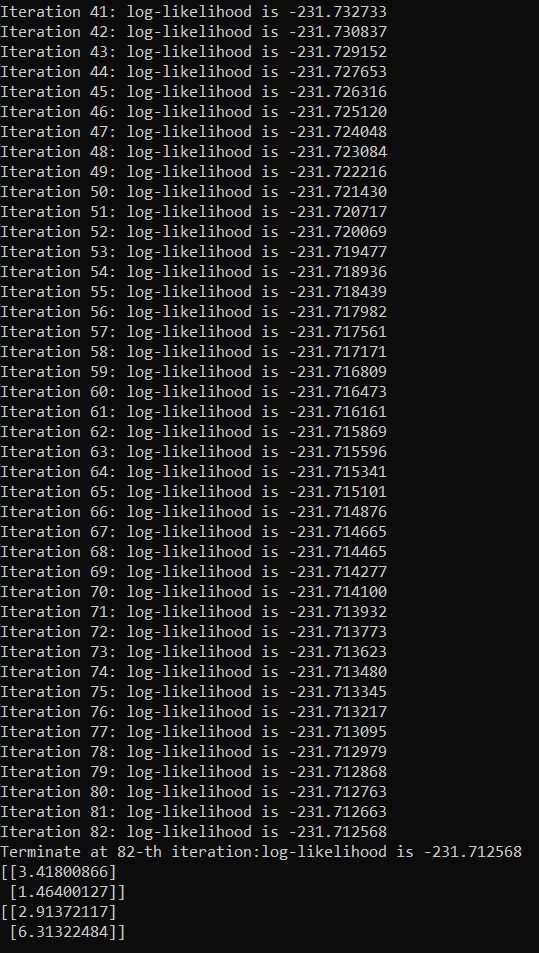
plt.xlim(-3, 9)

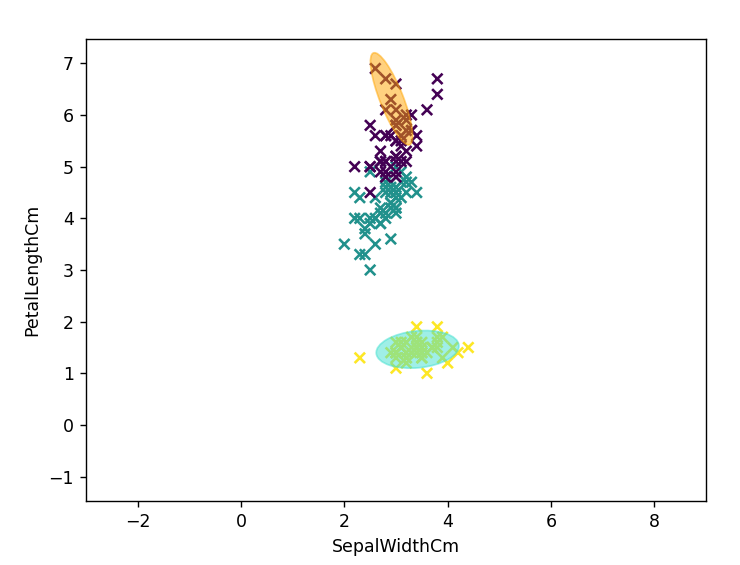
plt.ylim(-3, 9)

plt.xlabel(col[0])

plt.ylabel(col[1])

plt.show()



**Independent Component Analysis:**

* Independent Component Analysis is a signal processing method to separate independent sources linearly mixed in several sensors.
* ICA is a technique to separate linearly mixed sources.

**Algorithm:**

1. Center x by subtracting the mean
2. Whiten x

To *whiten*a given signal means that we transform it in such a way that potential correlations between its components are removed (covariance equal to 0) and the variance of each component is equal to 1.

The whitening process is simply a linear change of coordinate of the mixed data. Once the ICA solution is found in this “whitened” coordinate frame, we can easily reproject the ICA solution back into the original coordinate frame.

1. Choose a random initial value for the de-mixing matrix w
2. Calculate the new value for w
3. Normalize w
4. Check whether algorithm has converged and if it hasn’t, return to step 4
5. Take the dot product of w and x to get the independent source signals.

**Code**

import pandas as pd

import numpy as np

from scipy import signal

import matplotlib.pyplot as plt

%matplotlib inline

np.random.seed(23)

ns = np.linspace(0, 200, 1000)

S = np.array([np.sin(ns \* 1), signal.sawtooth(ns \* 1.9),

np.random.random(len(ns))]).T

A = np.array([[0.5, 1, 0.2],

[1, 0.5, 0.4],

[0.5, 0.8, 1]])

X = S.dot(A).T

fig, ax = plt.subplots(1, 1, figsize=[18, 5])

ax.plot(ns, S, lw=5)

ax.set\_xticks([])

ax.set\_yticks([-1, 1])

ax.set\_xlim(ns[0], ns[200])

ax.tick\_params(labelsize=12)

ax.set\_title('Independent sources', fontsize=25)

fig, ax = plt.subplots(3, 1, figsize=[18, 5], sharex=True)

ax[0].plot(ns, X[0], lw=5)

ax[0].set\_title('Mixed signals', fontsize=25)

ax[0].tick\_params(labelsize=12)

ax[1].plot(ns, X[1], lw=5)

ax[1].tick\_params(labelsize=12)

ax[1].set\_xlim(ns[0], ns[-1])

ax[2].plot(ns, X[2], lw=5)

ax[2].tick\_params(labelsize=12)

ax[2].set\_xlim(ns[0], ns[-1])

ax[2].set\_xlabel('Sample number', fontsize=20)

ax[2].set\_xlim(ns[0], ns[200])

plt.show()

def center(x):

mean = np.mean(x, axis=1, keepdims=True)

centered = x - mean

return centered, mean

def covariance(x):

mean = np.mean(x, axis=1, keepdims=True)

n = np.shape(x)[1] - 1

m = x - mean

return (m.dot(m.T))/n

def whiten(x):

coVarM = covariance(X)

U, S, V = np.linalg.svd(coVarM)

d = np.diag(1.0 / np.sqrt(S))

whiteM = np.dot(U, np.dot(d, U.T))

Xw = np.dot(whiteM, X)

return Xw, whiteM

def fastIca(signals, alpha = 1, thresh=1e-8, iterations=5000):

m, n = signals.shape

W = np.random.rand(m, m)

for c in range(m):

w = W[c, :].copy().reshape(m, 1)

w = w / np.sqrt((w \*\* 2).sum())

i = 0

lim = 100

while ((lim > thresh) & (i < iterations)):

ws = np.dot(w.T, signals)

wg = np.tanh(ws \* alpha).T

wg\_ = (1 - np.square(np.tanh(ws))) \* alpha

wNew = (signals \* wg.T).mean(axis=1) - wg\_.mean() \* w.squeeze()

wNew = wNew - np.dot(np.dot(wNew, W[:c].T), W[:c])

wNew = wNew / np.sqrt((wNew \*\* 2).sum())

lim = np.abs(np.abs((wNew \* w).sum()) - 1)

w = wNew

i += 1

W[c, :] = w.T

return W

Xc, meanX = center(X)

Xw, whiteM = whiten(Xc)

print(np.round(covariance(Xw)))

W = fastIca(Xw, alpha=1)

unMixed = Xw.T.dot(W.T)

unMixed = (unMixed.T - meanX).T

fig, ax = plt.subplots(1, 1, figsize=[18, 5])

ax.plot(S, lw=5)

ax.tick\_params(labelsize=12)

ax.set\_xticks([])

ax.set\_yticks([-1, 1])

ax.set\_title('Source signals', fontsize=25)

ax.set\_xlim(0, 100)

fig, ax = plt.subplots(1, 1, figsize=[18, 5])

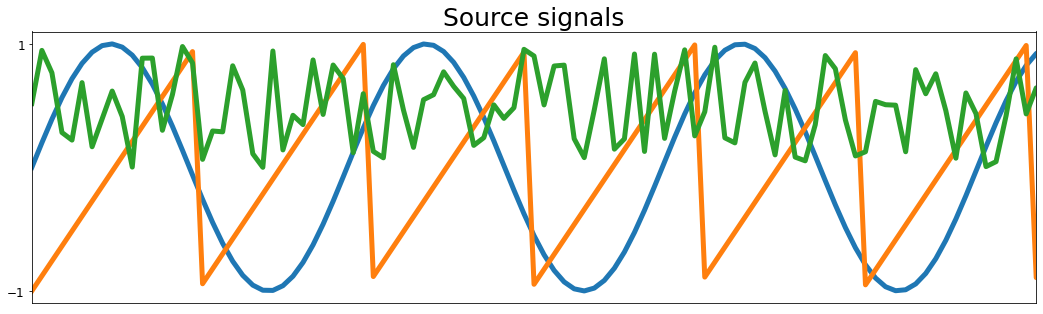
ax.plot(unMixed, '--', label='Recovered signals', lw=5)

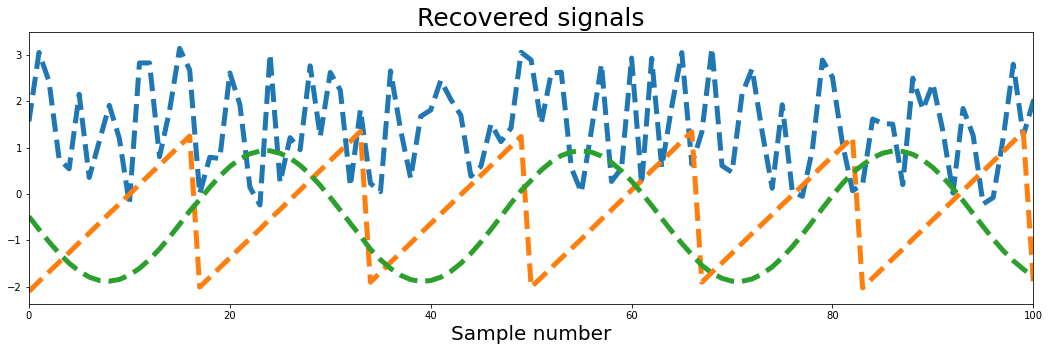
ax.set\_xlabel('Sample number', fontsize=20)

ax.set\_title('Recovered signals', fontsize=25)

ax.set\_xlim(0, 100)

plt.show()

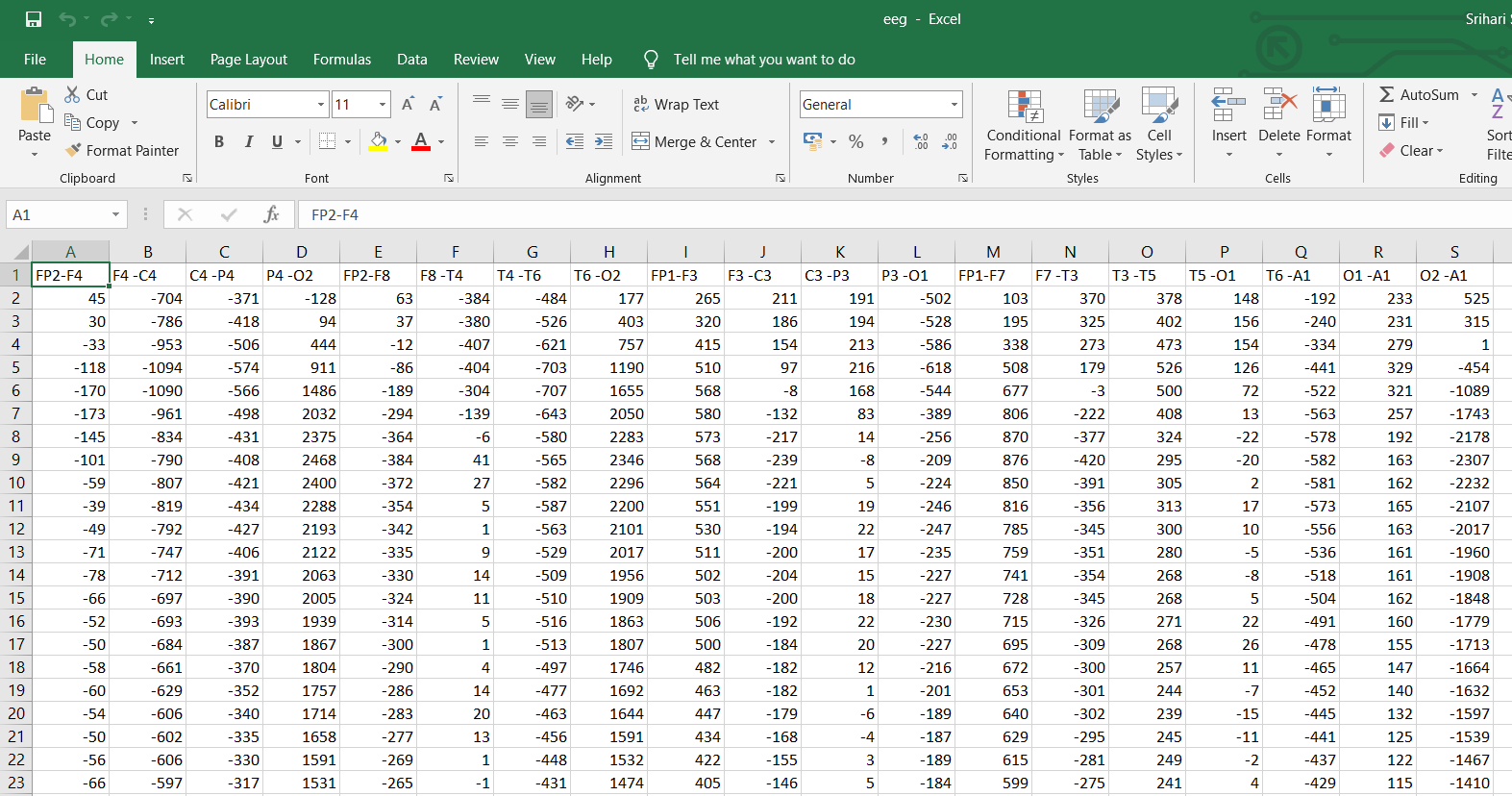
****

****

**INDEPENDENT COMPONENT ANALYSIS ON EEG DATASET:**

This data arises from a large study to examine EEG correlates of genetic predisposition to alcoholism. It contains measurements from 64 electrodes placed on subject's scalps which were sampled at 256 Hz (3.9-msec epoch) for 1 second. There were two groups of subjects: alcoholic and control. Each subject was exposed to either a single stimulus (S1) or to two stimuli (S1 and S2) which were pictures of objects chosen from the 1980 Snodgrass and Vanderwart picture set. When two stimuli were shown, they were presented in either a matched condition where S1 was identical to S2 or in a non-matched condition where S1 differed from S2.

**Input:**



**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from collections import deque

from sklearn.decomposition import FastICA

eeg = pd.read\_csv('eeg.csv')

eeg \*= 10\*\*6 # from V to uV

eeg.iloc[500:2500].plot(figsize=(15,5), legend=False)

plt.xlabel('Time [samples]', fontsize=14, labelpad=10)

plt.ylabel('Voltage [\u03BCV]', fontsize=14)

plt.title('Resting state EEG (19 channels)', fontsize=14)

plt.show()

fig, axs = plt.subplots(2,1, figsize=(15, 7), sharex=True, sharey=True)

axs = axs.ravel()

plt.margins(x=0.001)

fig.add\_subplot(111, frameon=False)

plt.tick\_params(labelcolor='none', top=False, bottom=False, left=False, right=False)

axs[0].plot(eeg.iloc[500:800,0], label='FP2-F4', color='rosybrown')

axs[0].legend(loc="upper right", fontsize=12)

axs[1].plot(eeg.iloc[500:800,1], label='F4 -C4', color='silver')

axs[1].legend(loc="upper right", fontsize=12)

plt.xlabel('Time [samples]', fontsize=14, labelpad=15)

plt.ylabel('Voltage [\u03BCV]', fontsize=14, labelpad=15)

plt.show()

ica = FastICA(n\_components=19, random\_state=0, tol=0.05)

comps = ica.fit\_transform(eeg)

fig, axs = plt.subplots(5,4, figsize=(18, 13), sharex=True, sharey=True)

fig.subplots\_adjust(hspace = .4, wspace=0)

axs = axs.ravel()

fig.add\_subplot(111, frameon=False)

plt.tick\_params(labelcolor='none', top=False, bottom=False, left=False, right=False)

plt.xlabel('Time [samples]', fontsize=14, labelpad=15)

for i in range(19):

axs[i].plot(comps[1200:1600, i], color='slategrey')

axs[i].set\_title(str(i))

comps\_restored = comps.copy()

comps\_restored[:,[4,10]] = 0 # set artefact components to zero

restored = ica.inverse\_transform(comps\_restored)

fig, axs = plt.subplots(2,1, figsize=(15, 7), sharex=True, sharey=True)

axs = axs.ravel()

plt.margins(x=0.001)

fig.add\_subplot(111, frameon=False)

plt.tick\_params(labelcolor='none', top=False, bottom=False, left=False, right=False)

axs[0].plot(eeg.iloc[500:1300,0], label='Fp1\_pre', color='rosybrown')

axs[0].plot(np.arange(500,1280), restored[500:2500, 11], label='Fp1\_post', color='maroon')

axs[0].legend(loc="upper right", fontsize=12)

axs[1].plot(eeg.iloc[500:1300,1], label='Fp2\_pre', color='silver')

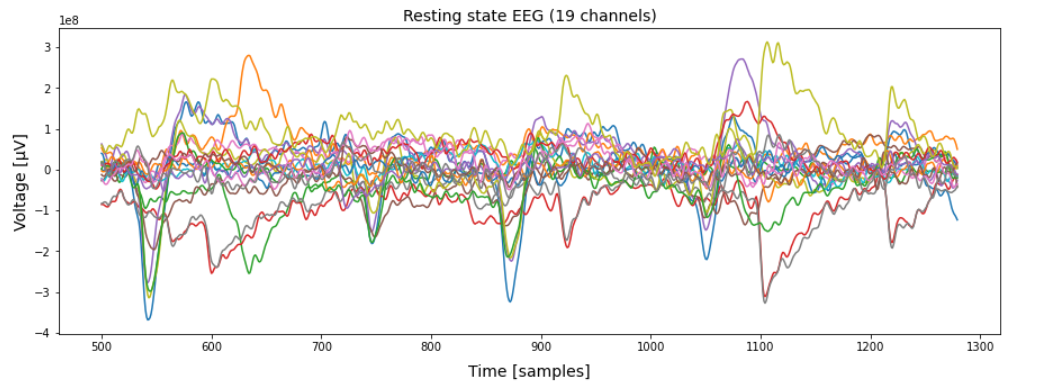
axs[1].plot(np.arange(500,1280), restored[500:2500, 11], label='Fp2\_post', color='dimgray')

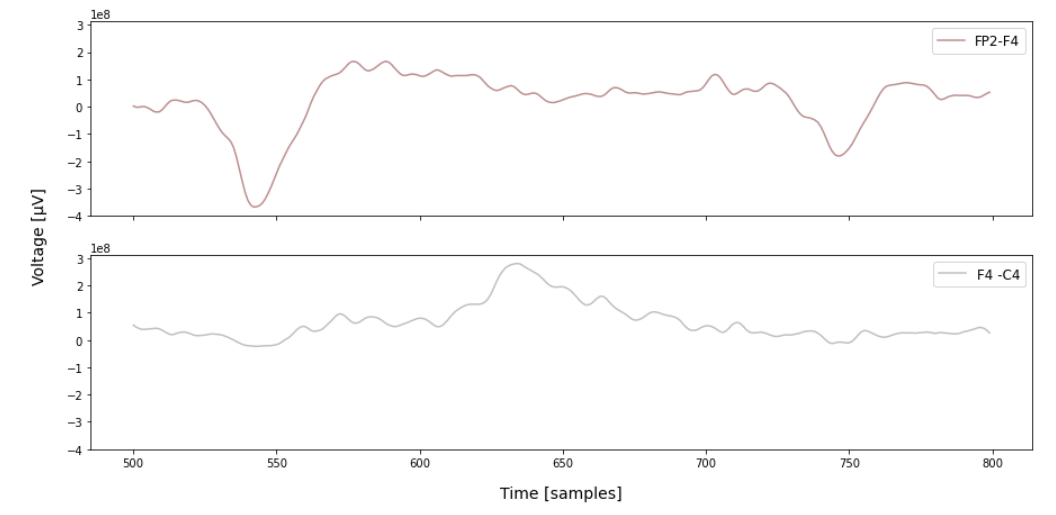
axs[1].legend(loc="upper right", fontsize=12)

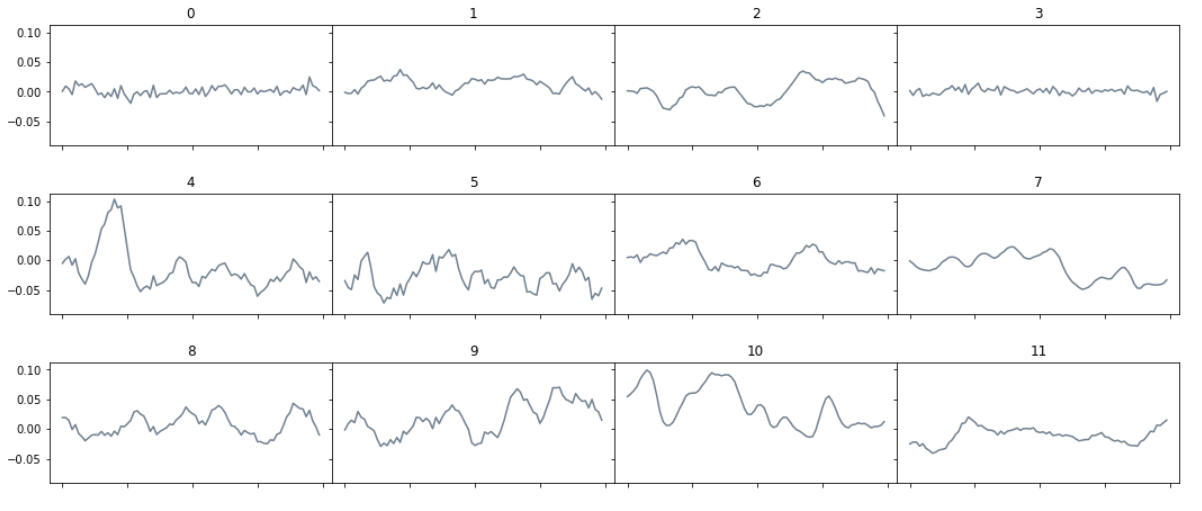
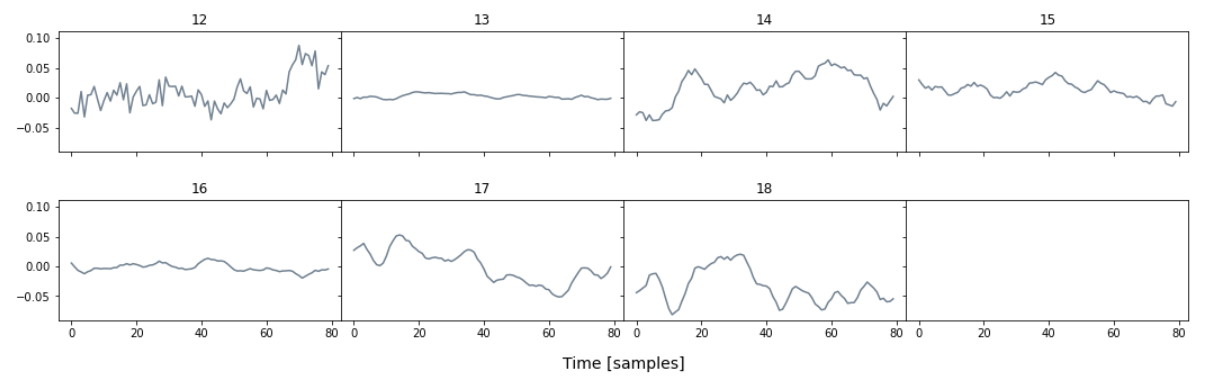
plt.xlabel('Time [samples]', fontsize=14, labelpad=15)

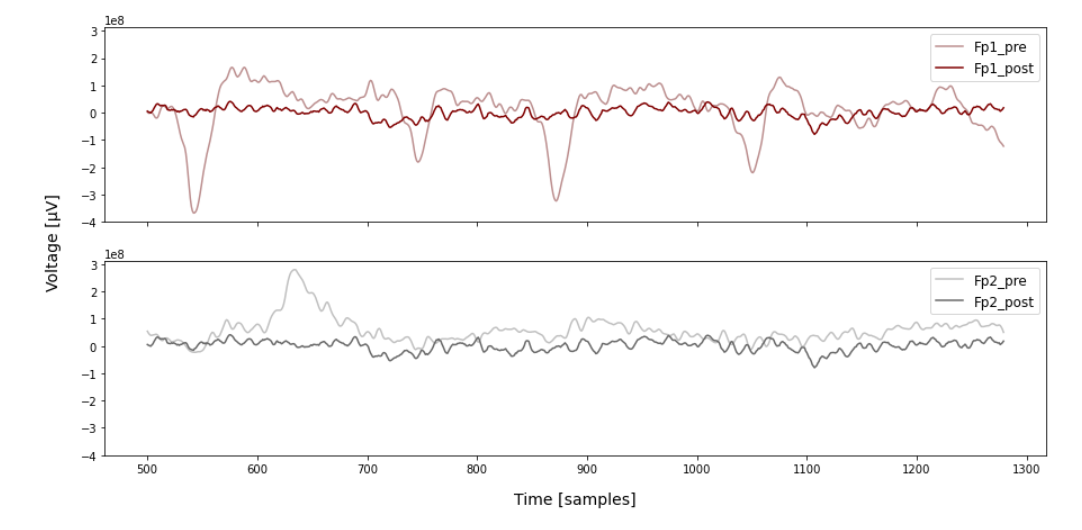
plt.ylabel('Voltage [\u03BCV]', fontsize=14, labelpad=15)

plt.show()







**Inference:**

Thus as we can see above the lighter lines indicate the signals captured before the distortions due to the blinking of the eye were removed. The darker version represents the final signals obtained after the eye blinking weren’t considered. Thus the spikes in the signals are removed as the independent components 4 and 10 represents the contribution of the blinking of the eye.