Machine Learning FINAL CA

Student Names:
Pouya Haji Mohammadi Gohari
SID:810102113

Faezeh Mozaffari
SID:810102254

Date of deadline
Sunday 14th July, 2024

Dept. of Computer Engineering

University of Tehran

Contents

1 Pre-processing

To enhance code robustness and readability, we will introduce a class named preprocess for handling pre-processing tasks. The needed libraries are added in the first cell in notebook:

```
import mne
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, sosfilt
from scipy import signal
import copy
from scipy.spatial.distance import cdist
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, FastICA
from sklearn.manifold import TSNE
from mne.decoding import CSP
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, roc_curve, auc,
    ↪ confusion_matrix
from sklearn.cluster import AgglomerativeClustering, KMeans
from clusteval import clusteval
```

We have consider following two files from calibration.All methods will be independently will be applied on both.Eventually we will compare the results between them.

```
first_file = 'G:\Master SE\Term 2\ML\Final Project\Data\BCICIV_calib_ds1b.mat'
second_file = 'G:\Master SE\Term 2\ML\Final Project\Data\BCICIV_calib_ds1c.mat'
```

1.1 Loading EEG Data

We utilize following method in order to load Matlab files and save them in organized way in dictionary data structure.

```
    def load_matlab_file(self, path:str) -> dict:
        data = {}
        mat = loadmat(path)
        mat = loadmat(path)

        eeg_data = mat['cnt']
        eeg_data = 0.1 * eeg_data.astype(np.float64)
        data['eeg_data'] = eeg_data.T

        info = mat['nfo']
        x_pos = info['xpos'][0][0]
```

```
        y_pos = info['ypos'][0][0]
        data['xy_pos'] = np.concatenate([x_pos, y_pos], axis=1)
        data['channel_names'] = [str(chan[0]) for chan in info['clab'][0][0][0]]
        data['classes'] = [cls[0] for cls in info['classes'][0][0][0]]
        data['sample_freq'] = info['fs'][0][0][0][0]


        markers = mat['mrk']
        data['pos'] = markers['pos'][0][0][0]
        data['y'] = markers['y'][0][0][0]
        print("Dataset has been loaded")
        return data
```

First with the help of 'loadmat' function from 'scipy' library the EEG data will be loaded with given path.According to description, the continuous data will be stored as follow:

$$\text{eeg\_data} = 0.1 * double(\text{cnt}) \tag{1}$$

Transpose the continuous data to have $(n_channels, n_samples)$ for further using in MNE library. Other information will be extracted from Matlab file and save them under a dictionary data structure(like positions, sample frequency, labels or y and xy positions correspond to head in 2D array)
An example has been provided how to load Matlab files:

```
preprocess = preprocess()
first_calib = preprocess.load_matlab_file(first_file)
```

1.2 Band Pass Filter

The method band pass filter, will filter a given signal with given low and high cutoff and sample frequency(Note that this parameter is unique for all subjects).

```
    def band_pass_filter(self, data:dict, low_freq:int, high_freq:int, order:int,
       ↪ axis=0) -> list[dict]:
        sos = butter(order, [low_freq, high_freq], btype='bandpass', fs=data['
           ↪ sample_freq'], output='sos')
        filtered_signal = copy.deepcopy(data)
        filtered = signal.sosfilt(sos, filtered_signal['eeg_data'])
        filtered_signal['eeg_data'] = filtered
        return filtered_signal
```

The butter function from the 'scipy.signal' module is used to design a 'Butterworth' band-pass filter. The function returns the filter coefficients in SOS format, which enhances numerical stability.
The 'sos' output refers to "Second-Order Sections". When designing digital filters, representing the filter as a series of second-order sections improves numerical stability, especially for higher-order filters. This representation divides the filter into multiple second-order filters that are applied in sequence. This is particularly useful in the implementation of the Butter worth filter in practice.
Eventually the 'eeg data' will be updated. An illustrative example has been provided for you. We applied band pass filter on first calib file. Also will discuss the differences before and after applying this method.

Before jumping to the example, we have add following two methods to plot the PSD[1] and at most 20 channels signal for better intution:
Plot PSD method:

```
    def plot_psd(self, data:dict, sample_freq:float) -> None:
        info = mne.create_info(ch_names=data['channel_names'], sfreq=sample_freq,
            ↪ ch_types='eeg')
        raw = mne.io.RawArray(data['eeg_data'], info)
        psd = raw.compute_psd(fmin=1, fmax=50)
        psd.plot(average=True, show=True)
```

Given a dictionary like the one will be created with 'load_matlab_file' method, first information will be created with 'MNE' library and raw data will be created afterward.Eventually PSD will be computed and plotted eventually.
Plot EEG with Events method:

```
    def plot_eeg_with_events(self, epochs:mne.Epochs, n_epochs:int, events:np.
       ↪ array) -> None:
        epochs.plot(n_epochs=n_epochs, title='Epochs', show=True, events=events,
            ↪ event_color= {-1: "r", 1: "b"}, scalings='auto')
```

This will generate a figure displaying up to 20 signals for the specified number of epochs and provided events. But you must create this epochs at first but do not worry, we have anotger method for that:

```
    def make_epochs(self, data:dict, tmin:int, tmax:int) -> mne.Epochs:
        eeg_data = data['eeg_data']
        if data['eeg_data'].shape[0] == 59:
            channel_names =  data['channel_names']
        else:
            channel_names = [str(i) for i in range(data['eeg_data'].shape[0])]
        sample_freq = data['sample_freq']
        event_pos = data['pos']
        event_codes = data['y']
        info = mne.create_info(ch_names=channel_names, sfreq=sample_freq, ch_types
            ↪ ='eeg')
        raw = mne.io.RawArray(eeg_data, info)
        event_id = dict(left = -1, right = 1)
        events = np.column_stack((event_pos, np.zeros_like(event_pos), event_codes
            ↪ ))
        return mne.Epochs(raw, events, event_id=event_id, tmin=tmin, tmax=tmax,
            ↪ baseline=None, preload=True)
```

Like method 'plot PSD', using channel names and sample frequency information will be created by MNE.Then raw data will created from given information.Event id are correspond to a dictionary that left is correspond to number minus one and right is correspond to plus 1.According to labels and positions of singals, events will be created. Note you should provid a window size by 'tmin', 'tmax'.
Let's give you an illustrative example for band pass filter and see if it is correclty implemented or not:

[1] Power Spectral Density

```
t_min = -0.3
t_max = 0.7
epochs_numbers = 10
first_calib = preprocess.load_matlab_file(first_file)
first_calib_epochs = preprocess.make_epochs(
    first_calib,
    tmin=t_min,
    tmax=t_max
)
first_events = np.column_stack((first_calib['pos'], np.zeros_like(first_calib['pos
    ↪ ']), first_calib['y']))
preprocess.plot_eeg_with_events(
    epochs=first_calib_epochs,
    n_epochs=epochs_numbers,
    events=first_events
)
```

20 electrode's signals are illustrated in Figure 1.

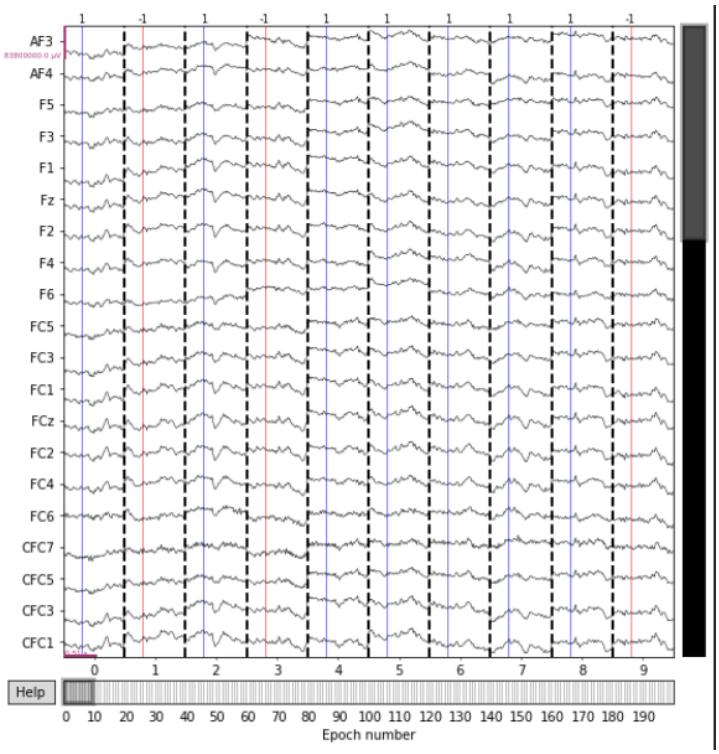

Figure 1: 20 Electrode Signals for First File

Lets see the signals for second file as well:

```
second_calib = preprocess.load_matlab_file(second_file)
second_calib_epochs = preprocess.make_epochs(
    second_calib,
```

```
    tmin=t_min,
    tmax=t_max
)
second_events = np.column_stack((second_calib['pos'], np.zeros_like(second_calib['
    ↪ pos']), second_calib['y']))
preprocess.plot_eeg_with_events(
    epochs=second_calib_epochs,
    n_epochs=epochs_numbers,
    events=second_events
)
```

20 channels signals are illustrated in Figure 2: Note that from now on till techniques subsection, we avoid to do on

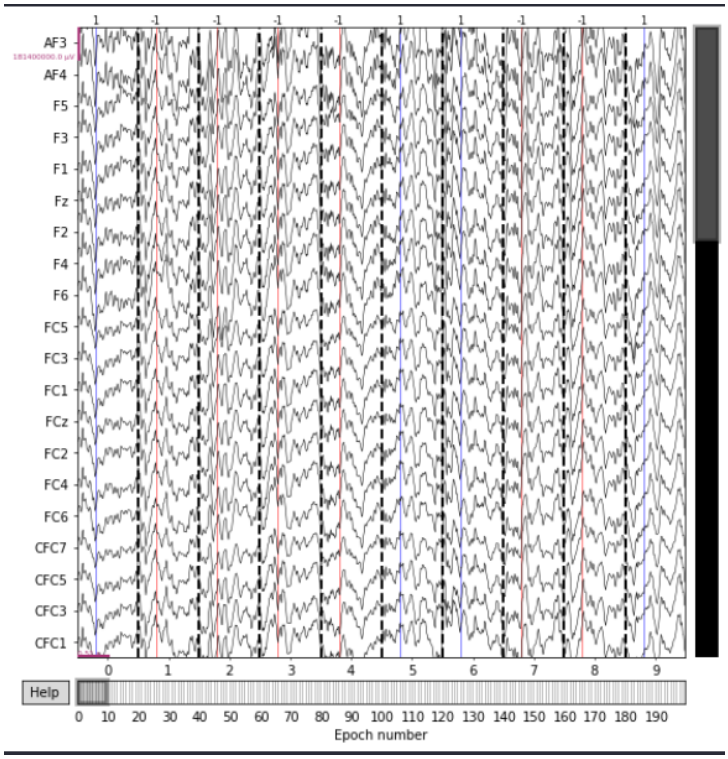

Figure 2: 20 Electrode Signals for Second File

second file since we just want to illustrate how these preprocessing methods will affect the data.Then in techniques subsection we do same things for both files.
Now let us proceed into band pass filter:

```
first_calib_preprocess = preprocess.band_pass_filter(
    first_calib,
    low_freq=8,
    high_freq=30,
    order=3,
    axis=0
)
epochs = preprocess.make_epochs(first_calib_preprocess, tmin=t_min, tmax=t_max)
```

```
preprocess.plot_eeg_with_events(epochs=epochs, n_epochs=epochs_numbers, events=
    ↪ first_events)
```

After band pass filter on first file, Figure3 has been illustrated:

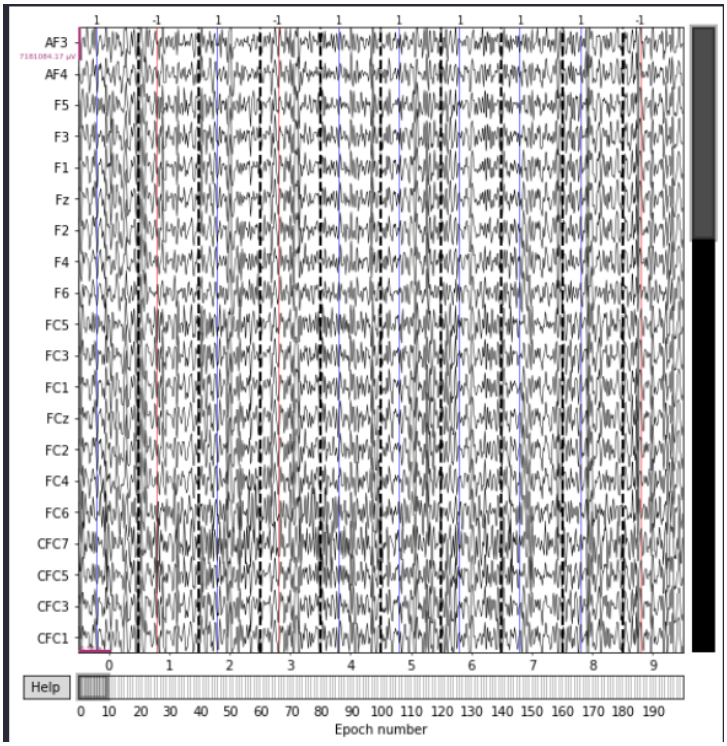

Figure 3: After band pass filter

After plotting PSD for before and after applying PSD, we will interpret the results.The Figure 4 are illustrated before and after applying band pass filter:

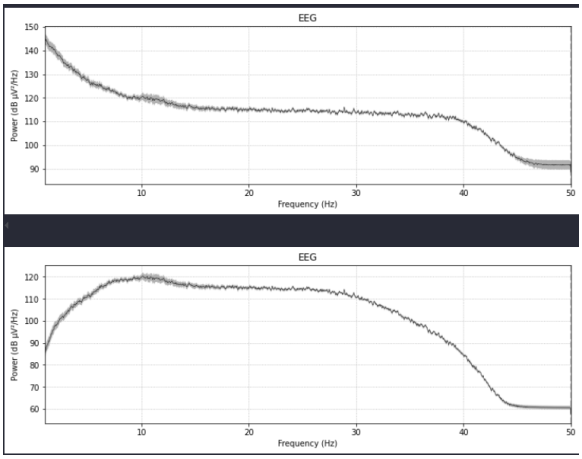

Figure 4: PSD for Raw and Filtered

Based on the Figure 1,Figure 3 and Figure 4 the interpretations are:

1. PSD Plots Analysis:

   (a) Before Band-Pass Filter (Top Plot):

   - The PSD plot indicates the distribution of power across various frequencies of the EEG signal before filtering.
   - Higher power is observed at lower frequencies, which is typical for raw EEG signals due to the presence of slow-wave activities.

   (b) After Band-Pass Filter (Bottom Plot):

   - The PSD plot shows the distribution of power after applying the band-pass filter.
   - The filter appears to have effectively removed the low-frequency components (below the low cut-off frequency) and high-frequency noise (above the high cut-off frequency), focusing on a specific frequency band.
   - This is evidenced by a noticeable drop in power at frequencies outside the band-pass range, indicating successful filtering.

2. Epochs Plot (Raw Signal):

   - The presence of low-frequency waves and possibly high-frequency noise can be observed.
   - The signal appears to have more variations and noise compared to the filtered signal.

3. Epochs Plot (Band-Pass Filtered Signal):

   - The epochs plot of the band-pass filtered signal shows a cleaner and more focused EEG signal.
   - The filtered signal appears smoother, indicating that unwanted frequency components have been successfully removed.

1.3 Common Average Reference

The CAR[2] is a straightforward method where the signal from each electrode is adjusted by subtracting the average signal of all electrodes.
Algorithm of this method is simple:

- Step 1: Calculating the Average Signal Calculate the average signal across all electrodes at each time point. This average represents the common noise present in the recording.
- Step 2: Subtracting the Average Signal: Subtract the calculated average signal from the individual electrode signals. This helps remove the common noise component.

The following method will do the rest:

```
    def apply_car(self, data:dict) -> dict:
        data_car = copy.deepcopy(data)
        average_potential = np.mean(data_car['eeg_data'], axis=0, keepdims=True)
        data_car['eeg_data'] -= average_potential
        return data_car
```

[2]Common Average Reference

Now let's see what would happen if car applied on the filtered data.

```
first_calib_preprocess = preprocess.apply_car(first_calib_preprocess)
epochs = preprocess.make_epochs(first_calib_preprocess, tmin=t_min, tmax=t_max)
preprocess.plot_eeg_with_events(epochs=epochs, n_epochs=epochs_numbers, events=
    ↪ first_events)
```

The Figure 5 has been shown the signals after applying the CAR method:

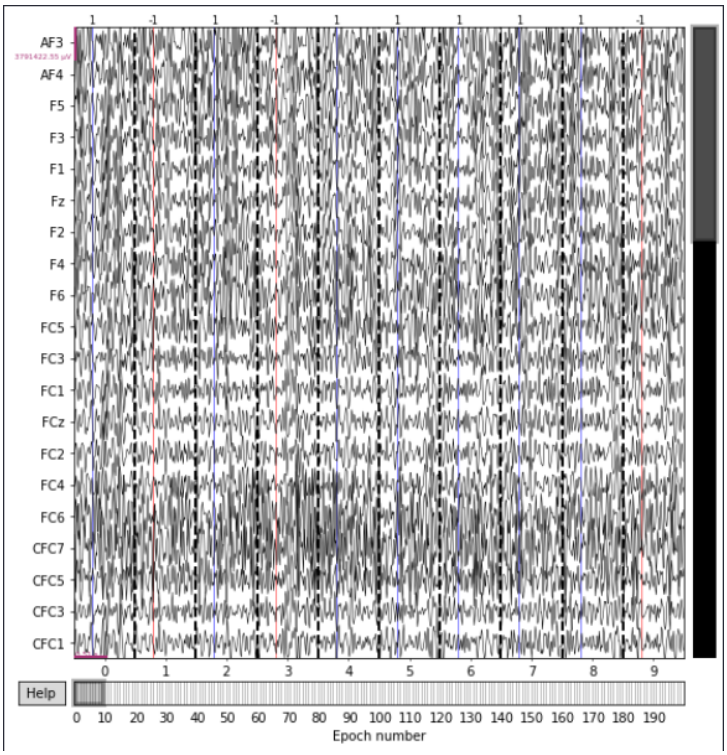

Figure 5: After Applying CAR

Based on the Figure 3 and Figure 5 interpretations are:

- Effect of CAR on EEG Signals:
    1. CAR method subtracts the average signal of all electrodes from each individual electrode's signal.
    2. This method will reduce the common noise present across all electrodes like environmental noise or physiological artifacts.
- Visual Differences:
    1. After Applying CAR Figure5:
        - The EEG signals appear more uniform and less noisy.
        - Common noise and artifacts have been significantly reduced, making the underlying brain activity signals more important.
        - The signals show clearer patterns which can help in better identifying and analyzing brain activity.

2. Without Applying CAR Figure 3:
   - The EEG signals exhibit more noise and variability across channels.
   - Common noise and artifacts are more obvious thus making it harder to identify clear patterns in the brain activity.

- Benefits of Using CAR

1. Noise Reduction: By averaging out the common noise across all electrodes, CAR enhances the signal-to-noise ratio.
2. Improved Signal Clarity: The EEG signals become clearer, making it easier to detect and analyze brain activity.
3. Enhanced Analysis: Cleaner signals facilitate more accurate analysis and interpretation, which is crucial for applications like brain-computer interfaces.

1.4 Laplacian

According to link[1], this is a spatial filtering method is utilized in EEG signal processing to enhance spatial resolution by emphasizing local EEG signal variations and reducing more extensive activity. It aids in detecting localized brain events and diminishes the influence of distant sources and common noise. The Laplacian x_i for electrode i is computed as follows:

$$x_i(t) = x_i(t) - \sum_{j \in N_i} w_{ij} x_j \tag{2}$$

$$w_{ij} = \frac{\frac{1}{d_{ij}}}{\sum_{j \in N_i} \frac{1}{d_{ij}}} \tag{3}$$

Here, $x_i(t)$ represents the potential of electrode i relative to the reference electrode, w_{ij} is the constant weight, d_{ij} is the Euclidean distance from electrode i to electrode j. N_i denotes the set of neighboring electrodes around electrode i.
Thus we will find all distances at first, neighbours will be found with specified radius and then apply laplacian mehtod:

```
    def _find_all_distances(self, given_positions:np.array) -> np.array:
        return cdist(given_positions, given_positions, metric='euclidean')
    def _finding_neighbors(self, distances:np.array, radius:int) -> dict:
        filtered_neighbors = {}
        for i in range(distances.shape[0]):
            neighbor_indices = np.where((distances[i] <= radius) & (distances[i] >
                ↪  0))[0]
            filtered_neighbors[i] = neighbor_indices.tolist()
        return filtered_neighbors
    def apply_small_laplacian(self, data:dict, radius:int) -> dict:
        neighbours_distances = self._find_all_distances(given_positions=data['
            ↪ xy_pos'])
        local_neighbors = self._finding_neighbors(distances=neighbours_distances,
            ↪ radius=radius)
```

```
        laplacian_applies = copy.deepcopy(data)
        eeg_data = laplacian_applies['eeg_data']
        local_averages = np.zeros_like(eeg_data)
        for electrode in range(eeg_data.shape[0]):
            neighbours = local_neighbors[electrode]
            weights = 1 / neighbours_distances[electrode, neighbours]
            local_averages[electrode, :] = np.dot(weights, eeg_data[neighbours,
                ↪ :])
        for electrode in range(eeg_data.shape[0]):
            laplacian_applies['eeg_data'][electrode] -= local_averages[electrode,
                ↪ :]
        return laplacian_applies
```

This function first find all distances based on 'cdist' function from 'scipy' library.Neighbours for each electrode will be obtained with specified radius. According to represented equations, each interested electrode will be subtracted the weighted average of all it's neighbors.
Now let's apply the Laplacian to preprocessed signals so far:

```
first_calib_preprocess = preprocess.apply_small_laplacian(first_calib_preprocess,
    ↪ radius=0.4)
epochs = preprocess.make_epochs(first_calib_preprocess, tmin=t_min, tmax=t_max)
preprocess.plot_eeg_with_events(epochs=epochs, n_epochs=epochs_numbers, events=
    ↪ first_events)
```

Figure6, showing the signals after applying Laplacian method: According to Figure 5 and Figure6, interpretations

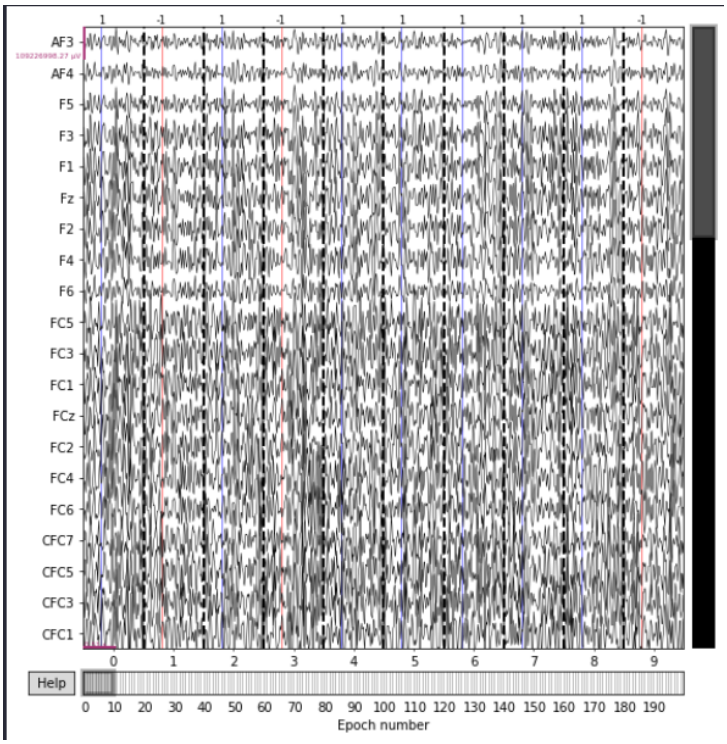

Figure 6: Apply Laplacian

has been provided:

- After Applying Laplacian Technique Figure6:
  - The EEG signals appear cleaner and more focused on local activities.
  - Common noise and widespread activity are significantly reduced, making the underlying localized brain signals more important.
  - The signals show clearer patterns, which can aid in better identifying and analyzing specific brain activities.

We can say that Laplacian and CAR approximately are same.Since both will improve signal clarity, enhance analysis, reduce noise of signals.However, CAR emphasis on the all electrodes(consider as a common noise across all electrodes tools), Laplacian are emphasis on the local and neighbors electrodes for interested electrode(noises where they effect each other).

1.5 PCA and ICA

PCA[3] and ICA[4] are methods commonly used for dimensionality reduction and signal decomposition in EEG data analysis. PCA focuses on finding orthogonal components that maximize the variance in the data, effectively reducing the dataset to its most informative features. This helps in simplifying the data, reducing noise, and improving computational efficiency for further analysis. On the other hand, ICA aims to separate the data into statistically independent components, which is particularly useful in identifying and isolating artifacts or specific brain sources within EEG signals. Both methods complement each other: PCA provides a way to manage large datasets by reducing dimensionality while retaining the most significant variance, and ICA offers a mixed signals into meaningful and independent sources.
We utilize following methods in order to apply PCA and fast ICA from 'sklearn' built-in functions.

```
    def apply_PCA(self, n_components:int, data:dict) -> np.array:
        pca_transformed = copy.deepcopy(data)
        X = pca_transformed['eeg_data']
        pca = PCA(n_components)
        pca_data = pca.fit_transform(X.T)
        pca_transformed['eeg_data'] = pca_data.T
        return pca_transformed

    def apply_ICA(self, n_components:int, data:dict) -> np.array:
        ica_transformed = copy.deepcopy(data)
        X = ica_transformed['eeg_data']
        ica = FastICA(n_components=n_components, whiten="unit-variance")
        ica_data = ica.fit_transform(X.T)
        ica_transformed['eeg_data'] = ica_data.T
        return ica_transformed
```

Both will reduce the number of channels to the number of components.In order to illustrate this techniques, we have add a method where use 'TSNE' of 'sklearn':

[3]Principal Component Analysis
[4]Independent Component Analysis

```
    def plot_tsne(self, data:dict, title:str):
        copy_data = copy.deepcopy(data)
        tsne = TSNE(n_components=2, random_state=42, init='random')
        reshaped = copy_data['eeg_data'].reshape(copy_data['eeg_data'].shape[0],
            ↪ -1)
        result = tsne.fit_transform(reshaped)
        plt.figure(figsize=(10, 7))
        plt.scatter(result[:, 0], result[:, 1], c=copy_data['y'], s=50, cmap='
            ↪ viridis')
        plt.title(title)
        plt.xlabel('t-SNE Component 1')
        plt.ylabel('t-SNE Component 2')
        plt.show()
```

Now let's see how PCA and ICA will work.

```
pca_transofrmed = preprocess.apply_PCA(n_components=30, data=
    ↪ first_calib_preprocess)
epochs = preprocess.make_epochs(pca_transofrmed, tmin=t_min, tmax=t_max)
preprocess.plot_tsne({'eeg_data':epochs.get_data(copy=True), 'y':epochs.events
    ↪ [:,-1]}, "TSNE visualization for PCA")
ica_transformed = preprocess.apply_ICA(n_components=30, data=
    ↪ first_calib_preprocess)
epochs = preprocess.make_epochs(ica_transformed, tmin=t_min, tmax=t_max)
preprocess.plot_tsne({'eeg_data':epochs.get_data(copy=True), 'y':epochs.events
    ↪ [:,-1]}, "TSNE visualization for ICA")
```

The Figure 7 and Figure8 are result of PCA and ICA respectively:

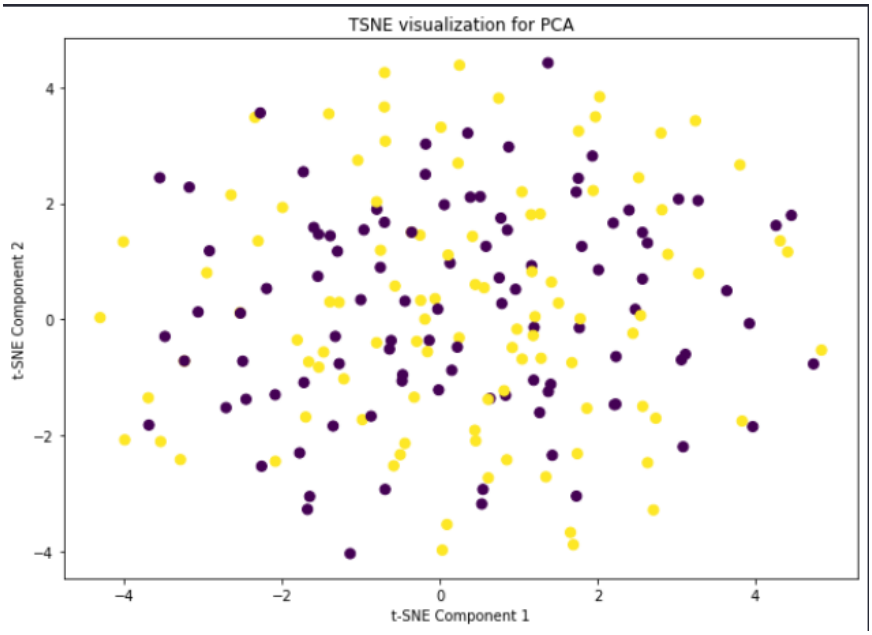

Figure 7: PCA Result Using Tnse

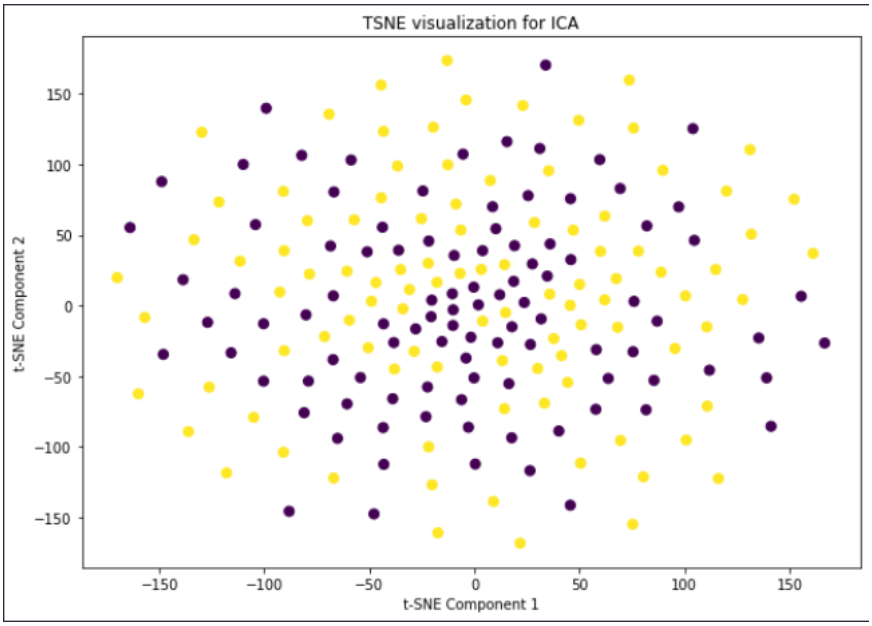

Figure 8: ICA Result Using Tnse

Some interpretations:

- In the PCA plot(Figure7), the classes are somewhat intermixed, indicating that the principal components are good at capturing the overall variance but not necessarily the class-specific structures.This plot also appears more dispersed and less structured, which can be typical when the main variances do not correspond to class separations.
- In the ICA plot(Figure8), the classes are more distinctly separated, suggesting that ICA has effectively identified independent sources that correspond well with the class labels.Moreover, this plot is more circular and structured, indicating that ICA has found components that provide better separation and clustering of the data points.

Note that same number of components has been set for PCA and ICA(30).

1.6 CSP

We had use CSP[5], to extract feature from signals.Utilizing the following method will help us to do so.

```
    def apply_CSP(self, epochs:mne.Epochs, n_components:int, data:dict) -> np.
       ↪ array:
        csp_data = copy.deepcopy(data)
        X = epochs.get_data(copy=True)
        y = epochs.events[:, -1]
        csp = CSP(n_components=n_components, reg=None, log=True, norm_trace=False)
        X_csp = csp.fit_transform(X, y)
        csp_data['eeg_data'] = X_csp
        return csp_data
```

[5]Common Spatial Pattern

Note that 'CSP' function in MNE library will get $(epoch, channels, nsamples)$. Therefore we should make epochs with 'make_epochs' method first.
Give an illustrative example:

```
csp_transformed = preprocess.apply_CSP(epochs=epochs, n_components=40, data=
    ↪ first_calib)
preprocess.plot_tsne(csp_transformed, title="TSNE visualizaiton for CSP")
```

Given 40 as number of components, Figure 9 will be depicted. Ineterpretation:

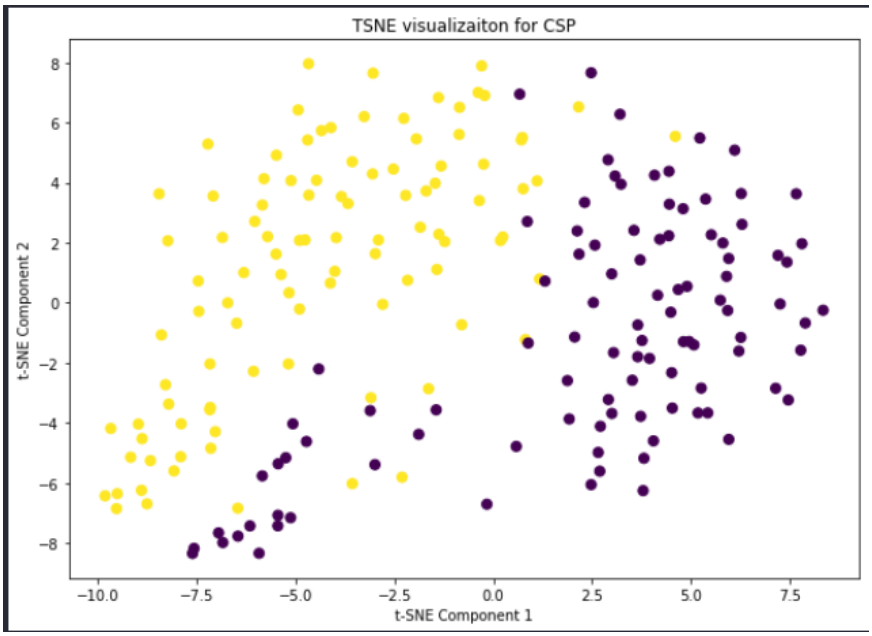

Figure 9: CSP Using Tnse

- The Figure 9 shows two distinct clusters of data points, one in yellow and one in purple.
- The clear separation suggests that CSP has effectively extracted features that distinguish between the two classes. This means the spatial filters identified by CSP have maximized the variance for each class in a way that makes them easily separable.
- The distinct separation indicates that the extracted features can be used to classify the EEG data into two categories with high accuracy.

The Figure 10, is illustrated for second file using CSP:

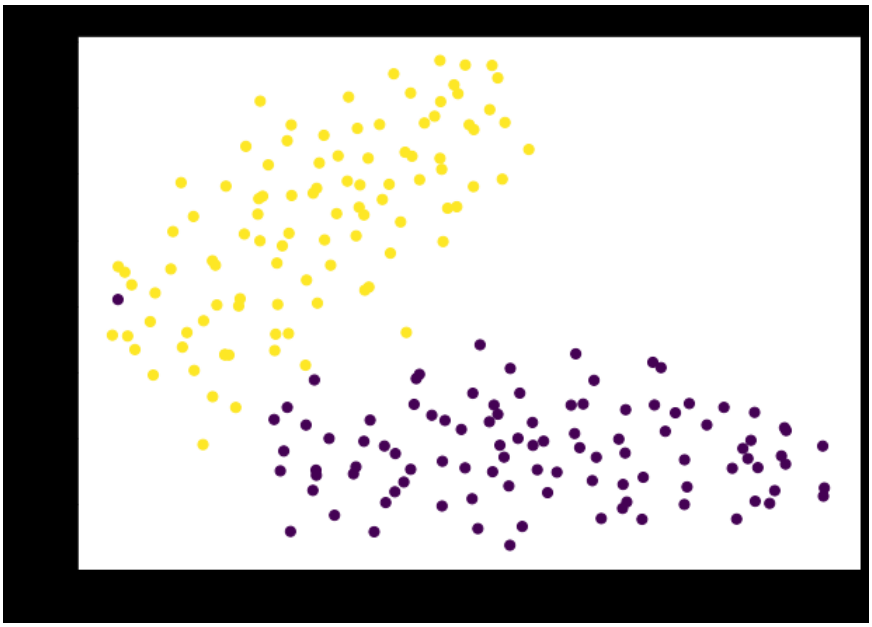

Figure 10: CSP Using Tnse for second file

As you can see data has been well separated.

1.7 Combinations of Preprocessed Method

We have 4 techniques(combination) of our methods, where will applied for both calibrated files.The Table 1 will illustrate these combinations,

	band pass	CAR	Laplacian	PCA (component)	ICA (component)	CSP(component)
Combination 1	✓	✓	✓	✗	✗	✓(40)
Combination 2	✓	✓	✓	✓(40)	✗	✓(20)
Combination 3	✓	✓	✓	✗	✓(40)	✓(20)
Combination 4	✓	✓	✓	✓(40)	✓(20)	✓(10)

Table 1: Combinations

Some of parameters are same:

```
preprocess_paramters = {
    'low_freq': 8,
    'high_freq': 30,
    'order':3,
    'radius':0.4,
    'tmin':-0.3,
    'tmax':0.7
}
```

Combination 1 for both file:

```
preprocess_first_calib_t1 = preprocess.band_pass_filter(first_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_first_calib_t1 = preprocess.apply_car(preprocess_first_calib_t1)
preprocess_first_calib_t1 = preprocess.apply_small_laplacian(
    ↪ preprocess_first_calib_t1, radius=preprocess_paramters['radius'])
first_epochs_t1 = preprocess.make_epochs(preprocess_first_calib_t1,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
first_technique_1 = preprocess.apply_CSP(first_epochs_t1, n_components=40, data=
    ↪ preprocess_first_calib_t1)


### Second File also
preprocess_second_calib_t1 = preprocess.band_pass_filter(second_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_second_calib_t1 = preprocess.apply_car(preprocess_second_calib_t1)
preprocess_second_calib_t1 = preprocess.apply_small_laplacian(
    ↪ preprocess_second_calib_t1, radius=preprocess_paramters['radius'])
second_epochs_t1 = preprocess.make_epochs(preprocess_second_calib_t1,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
second_technique_1 = preprocess.apply_CSP(second_epochs_t1, n_components=40, data=
    ↪ preprocess_second_calib_t1)
```

Second Combination:

```
preprocess_first_calib_t2 = preprocess.band_pass_filter(first_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_first_calib_t2 = preprocess.apply_car(preprocess_first_calib_t2)
preprocess_first_calib_t2 = preprocess.apply_small_laplacian(
    ↪ preprocess_first_calib_t2, radius=preprocess_paramters['radius'])
preprocess_first_calib_t2 = preprocess.apply_PCA(n_components=40, data=
    ↪ preprocess_first_calib_t2)
first_epochs_t2 = preprocess.make_epochs(preprocess_first_calib_t2,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
first_technique_2 = preprocess.apply_CSP(first_epochs_t2, n_components=20, data=
    ↪ preprocess_first_calib_t2)


### Second File also
preprocess_second_calib_t2 = preprocess.band_pass_filter(second_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_second_calib_t2 = preprocess.apply_car(preprocess_second_calib_t2)
preprocess_second_calib_t2 = preprocess.apply_small_laplacian(
    ↪ preprocess_second_calib_t2, radius=preprocess_paramters['radius'])
preprocess_second_calib_t2 = preprocess.apply_PCA(n_components=40, data=
```

```
    ↪ preprocess_second_calib_t2)
second_epochs_t2 = preprocess.make_epochs(preprocess_second_calib_t2,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
second_technique_2 = preprocess.apply_CSP(second_epochs_t2, n_components=20, data=
    ↪ preprocess_second_calib_t2)
```

Third Combination:

```
preprocess_first_calib_t3 = preprocess.band_pass_filter(first_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_first_calib_t3 = preprocess.apply_car(preprocess_first_calib_t3)
preprocess_first_calib_t3 = preprocess.apply_small_laplacian(
    ↪ preprocess_first_calib_t3, radius=preprocess_paramters['radius'])
preprocess_first_calib_t3 = preprocess.apply_ICA(n_components=40, data=
    ↪ preprocess_first_calib_t3)
first_epochs_t3 = preprocess.make_epochs(preprocess_first_calib_t3,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
first_technique_3 = preprocess.apply_CSP(first_epochs_t3, n_components=20, data=
    ↪ preprocess_first_calib_t3)


### Second File also
preprocess_second_calib_t3 = preprocess.band_pass_filter(second_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_second_calib_t3 = preprocess.apply_car(preprocess_second_calib_t3)
preprocess_second_calib_t3 = preprocess.apply_small_laplacian(
    ↪ preprocess_second_calib_t3, radius=preprocess_paramters['radius'])
preprocess_second_calib_t3 = preprocess.apply_ICA(n_components=40, data=
    ↪ preprocess_second_calib_t3)
second_epochs_t3 = preprocess.make_epochs(preprocess_second_calib_t3,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
second_technique_3 = preprocess.apply_CSP(second_epochs_t3, n_components=20, data=
    ↪ preprocess_second_calib_t3)
```

Fourth Combination:

```
preprocess_first_calib_t4 = preprocess.band_pass_filter(first_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_first_calib_t4 = preprocess.apply_car(preprocess_first_calib_t4)
preprocess_first_calib_t4 = preprocess.apply_small_laplacian(
    ↪ preprocess_first_calib_t4, radius=preprocess_paramters['radius'])
preprocess_first_calib_t4 = preprocess.apply_PCA(n_components=40, data=
    ↪ preprocess_first_calib_t4)
preprocess_first_calib_t4 = preprocess.apply_ICA(n_components=20, data=
    ↪ preprocess_first_calib_t4)
first_epochs_t4 = preprocess.make_epochs(preprocess_first_calib_t4,
```

```
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
first_technique_4 = preprocess.apply_CSP(first_epochs_t4, n_components=10, data=
    ↪ preprocess_first_calib_t4)


### Second File also
preprocess_second_calib_t4 = preprocess.band_pass_filter(second_calib, low_freq=
    ↪ preprocess_paramters['low_freq'], high_freq=preprocess_paramters['high_freq'
    ↪ ], order=preprocess_paramters['order'])
preprocess_second_calib_t4 = preprocess.apply_car(preprocess_second_calib_t4)
preprocess_second_calib_t4 = preprocess.apply_small_laplacian(
    ↪ preprocess_second_calib_t4, radius=preprocess_paramters['radius'])
preprocess_second_calib_t4 = preprocess.apply_PCA(n_components=40, data=
    ↪ preprocess_second_calib_t4)
preprocess_second_calib_t4 = preprocess.apply_ICA(n_components=20, data=
    ↪ preprocess_second_calib_t4)
second_epochs_t4 = preprocess.make_epochs(preprocess_second_calib_t4,
    ↪ preprocess_paramters['tmin'], preprocess_paramters['tmax'])
second_technique_4 = preprocess.apply_CSP(second_epochs_t4, n_components=10, data=
    ↪ preprocess_second_calib_t4)
```

This codes are simply implemented and since we explain each of them in previous subsections, we avoid to explain this combinations.

2 Classifiers

2.1 Implementation

We consider three classifier SVM[6], KNN[7] and Logistic Regression.For simplicity we have implemented a class for this purpose. Note that we have use Grid Search for tuning hyper parameters for each classifier.
Before jumping to these classifiers, we have some method for confusion matrix and classification and also roc curve.

```
    def _classification_report(self, y_test:np.array, y_pred:np.array) -> dict:
        return classification_report(y_test, y_pred)
    def _confusion_matrix(self, y_test:np.array, y_pred:np.array) -> dict:
        return confusion_matrix(y_test, y_pred)
    def _plot_roc_curve(self, y_test:np.array, y_pred_proba:np.array) -> None:
        fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
        roc_auc = auc(fpr, tpr)
        plt.figure()
        plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
        plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver Operating Characteristic (ROC) Curve')
        plt.legend()
        plt.show()
```

The classification report and confusion matrix are simply the built-in function of 'sklearn'. Now using decision function and predict probability of best estimator of grid search(SVM using decision function and rest will use predict prob) we will calculate the FPR[8] and TPR[9]. These FPR and TPR are obtained by 'roc_curve' function from sklearn library.
Implementing splitting dataset into train, test:

```
    def split_data(self, X:np.arange, y:np.array, test_size:float=0.25,
       ↪ random_state:int=42) -> tuple:
        X = copy.deepcopy(X)
        y = copy.deepcopy(y)
        X = X.reshape(X.shape[0], -1)
        return train_test_split(X, y, test_size=test_size, stratify=y,
           ↪ random_state=random_state)
```

Implementing Grid search based for SVM as a method of our class:

```
    def SVM(self, X_train:np.array, y_train:np.array, X_test:np.array, y_test:np.
       ↪ array, param_grid:dict, cv:int=5) -> None:
```

[6]Support Vector Machine
[7]K Nearest Neighbors
[8]False Positive Rate
[9]True Positive Rate

```
        grid_search = GridSearchCV(SVC(), param_grid, cv=cv, scoring='accuracy',
            ↪ n_jobs=-1, verbose=1)
        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_
        print("--------------------------------")
        print(f'The best parameters are:\n {grid_search.best_params_}')
        y_pred = best_model.predict(X_test)
        y_score = best_model.decision_function(X_test)
        print("--------------------------------")
        print('classification report:')
        print(self._classification_report(y_test, y_pred))
        print("--------------------------------")
        print("Confusion matrix:")
        print(self._confusion_matrix(y_test, y_pred))
        print("--------------------------------")
        self._plot_roc_curve(y_test=y_test, y_pred_proba=y_score)
```

Given train and test set and interested parameters as dictionary the best SVM will be found.
Attention: Utilizing confusion matrix, classification report and roc curve methods, we will print and plot them after each classifier we have designed.
Implementing Grid search based for KNN as a method of our class:

```
    def KNN(self, X_train:np.array, y_train:np.array, X_test:np.array, y_test:np.
        ↪ array, param_grid:dict, cv:int=5) -> None:
        grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5,
            ↪ scoring='accuracy')
        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_
        print(f'The best parameters are:\n {grid_search.best_params_}')
        y_pred = best_model.predict(X_test)
        y_score = best_model.predict_proba(X_test)
        print("--------------------------------")
        print('classification report:')
        print(self._classification_report(y_test, y_pred))
        print("--------------------------------")
        print("Confusion matrix:")
        print(self._confusion_matrix(y_test, y_pred))
        print("--------------------------------")
        self._plot_roc_curve(y_test=y_test, y_pred_proba=y_score[:, 1])
```

Same as SVM!
Implementing Grid search based for Logistic Regression as a method of our class:

```
    def LogReg(self, X_train:np.array, y_train:np.array, X_test:np.array, y_test:
        ↪ np.array, param_grid:dict, cv:int=5) -> None:
        y_train_1 = np.copy(y_train)
        y_test_1 = np.copy(y_test)
        y_train_1 = (y_train_1 == 1).astype(int)
        y_test_1 = (y_test_1 == 1).astype(int)
```

```
        grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring
            ↪ ='accuracy')
        grid_search.fit(X_train, y_train_1)
        best_model = grid_search.best_estimator_
        print(f'The best parameters are:\n {grid_search.best_params_}')
        y_pred = best_model.predict(X_test)
        y_score = best_model.predict_proba(X_test)
        print("--------------------------------")
        print('classification report:')
        print(self._classification_report(y_test_1, y_pred))
        print("--------------------------------")
        print("Confusion matrix:")
        print(self._confusion_matrix(y_test_1, y_pred))
        print("--------------------------------")
        self._plot_roc_curve(y_test=y_test_1, y_pred_proba=y_score[:, 1])
```

Grid parameters for each classification has been provided as follow:

```
C_range = [0.1, 1, 10, 100, 1000]

svm_grid = [
    {'kernel': ['linear'], 'C': C_range},
    {'kernel': ['poly'], 'C': C_range, 'degree': [2, 3, 4], 'gamma': ['scale', '
        ↪ auto']},
    {'kernel': ['rbf'], 'C': C_range, 'gamma': ['scale', 'auto']},
]

knn_grid = {
    'n_neighbors': [i+1 for i in range(20)],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

log_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}
```

We will interpret each result for each file calibration:

2.2 Results of Combination 1

The Figure 11 is SVM result on first File:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
------------------------------
The best parameters are:
 {'C': 1, 'kernel': 'linear'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.83      1.00      0.91        25
           1       1.00      0.80      0.89        25

    accuracy                           0.90        50
   macro avg       0.92      0.90      0.90        50
weighted avg       0.92      0.90      0.90        50

------------------------------
Confusion matrix:
[[25  0]
 [ 5 20]]
------------------------------
```

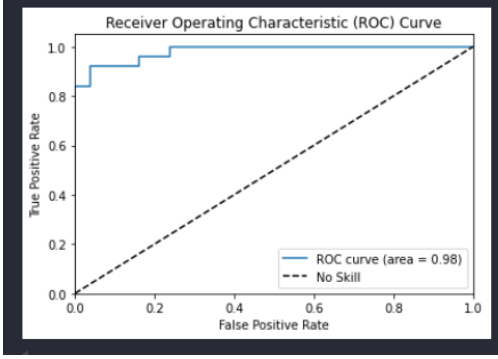

Figure 11: SVM Result on First File Using First Technique

Interpretation:

- **Best Parameters:**
  - `C`: 1
  - `kernel`: 'linear'
- **Classification Report:**
  - **Class -1:**
    - Precision: 0.83
    - Recall: 1.00
    - F1-Score: 0.91
    - Support: 25
  - **Class 1:**
    - Precision: 1.00
    - Recall: 0.80
    - F1-Score: 0.89
    - Support: 25

- **Overall:**
  - Accuracy: 0.90
  - Macro Average Precision: 0.92
  - Macro Average Recall: 0.90
  - Macro Average F1-Score: 0.90
  - Weighted Average Precision: 0.92
  - Weighted Average Recall: 0.90
  - Weighted Average F1-Score: 0.90

- **Confusion Matrix:**
  - True Negative (Class -1 correctly classified): 25
  - False Negative (Class -1 misclassified): 0
  - False Positive (Class 1 misclassified): 5
  - True Positive (Class 1 correctly classified): 20

- **ROC Curve:**
  - **AUC (Area Under Curve):** 0.98

Wrap all this up:
The SVM classifier with a linear kernel and C=1 demonstrates strong performance, achieving an overall accuracy of 90%. Class -1 shows high precision (0.83) and perfect recall (1.00), leading to a high F1-score (0.91). Class 1 achieves perfect precision (1.00) but has slightly lower recall (0.80), resulting in an F1-score of 0.89. These metrics indicate the classifier is effective in distinguishing between the two classes.

The ROC curve further validates the model's effectiveness, with an AUC of 0.98, suggesting excellent discrimination capability between the classes. The confusion matrix reveals a minor imbalance, with all instances of Class -1 correctly classified, but 5 instances of Class 1 misclassified as Class -1. This slight imbalance in recall for Class 1 suggests an area for potential improvement.

Overall, the SVM model with the chosen parameters is highly effective for this dataset, evidenced by high precision, recall, and AUC values. While the model performs well, future enhancements could focus on addressing the recall imbalance for Class 1 to achieve even more robust performance. The high AUC value underscores the model's reliability and ability to effectively distinguish between the classes.
The Figure 12 is KNN result on first File:

```
The best parameters are:
 {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'uniform'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.86      1.00      0.93        25
           1       1.00      0.84      0.91        25

    accuracy                           0.92        50
   macro avg       0.93      0.92      0.92        50
weighted avg       0.93      0.92      0.92        50

------------------------------
Confusion matrix:
[[25  0]
 [ 4 21]]
------------------------------
```

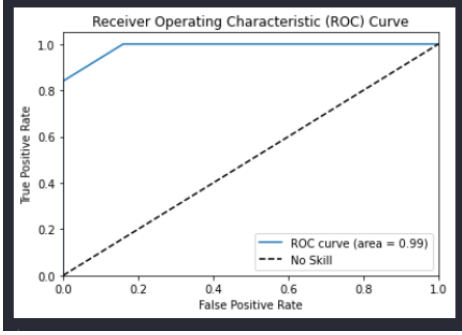

Figure 12: KNN Result on First File Using First Technique

From Now on we will just summarize the resuls obtained by the Grid search and corresponding classifier.
Interpretations:
The KNN classifier, using Euclidean distance, 8 neighbors, and uniform weights, demonstrates exceptional performance on the provided dataset. With an overall accuracy of 92%, the model shows it can correctly classify the majority of instances. Both classes exhibit high precision and recall values, particularly Class -1, which achieves a perfect recall score of 1.00, indicating all instances of this class were correctly identified.

The ROC curve further underscores the model's efficacy, with an AUC of 0.99, suggesting near-perfect classification ability. This high AUC value signifies that the model has an excellent capacity to distinguish between the two classes. The confusion matrix reveals a small number of misclassifications, specifically 4 false positives for Class 1, while all instances of Class -1 were accurately classified, reinforcing the model's reliability.

In conclusion, the KNN classifier with the selected parameters is highly effective for this dataset, as evidenced by its high precision, recall, and AUC values. The slight imbalance in recall between the two classes indicates potential areas for further refinement, yet the overall performance remains outstanding. Future efforts might focus on addressing the few misclassifications to further enhance the model's robustness and accuracy.
The Figure **??**, results of Logistic Regression:

```
The best parameters are:
 {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.86      0.96      0.91        25
           1       0.95      0.84      0.89        25

    accuracy                           0.90        50
   macro avg       0.91      0.90      0.90        50
weighted avg       0.91      0.90      0.90        50

------------------------------
Confusion matrix:
[[24  1]
 [ 4 21]]
------------------------------
```

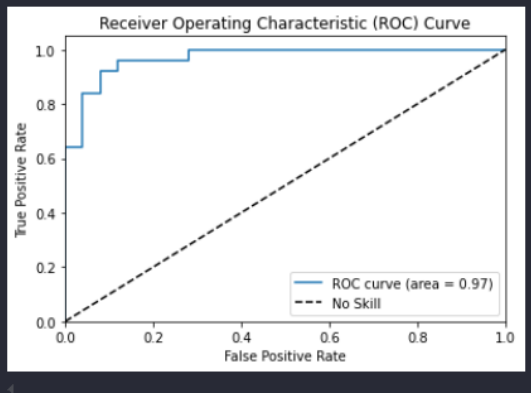

Figure 13: Logistic Regression Result on First File Using First Technique

Interpretations:

The Logistic Regression model with an L1 penalty, C=1, and the liblinear solver shows robust performance, achieving an overall accuracy of 90%. Class 0 demonstrates high precision (0.86) and recall (0.96), resulting in an F1-score of 0.91. Class 1 achieves an even higher precision (0.95) but has slightly lower recall (0.84), leading to an F1-score of 0.89. These results indicate the model is effective in distinguishing between the two classes, although there is a slight imbalance in recall.

The ROC curve with an AUC of 0.97 indicates excellent classification ability, suggesting that the model has a high capability to distinguish between the classes. The confusion matrix reveals that most instances are correctly classified, with only a few misclassifications: 1 instance of Class 0 is misclassified, and 4 instances of Class 1 are misclassified. This indicates a slight tendency for the model to misclassify Class 1 instances as Class 0.

In conclusion, the Logistic Regression model with the specified parameters performs very well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, there is a small imbalance in recall between the classes. Future efforts could focus on addressing these misclassifications to further enhance model performance. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes.

The Figure 14, The result of SVM for second file using first technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
------------------------------
The best parameters are:
 {'C': 0.1, 'kernel': 'linear'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        25

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50

------------------------------
Confusion matrix:
[[25  0]
 [ 0 25]]
------------------------------
```

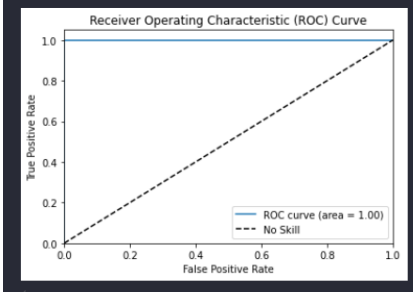

Figure 14: SVM Result on Second File Using First Technique

Interpretations:

The SVM classifier with a linear kernel and C=0.1 demonstrates perfect performance on the given dataset. It achieves an overall accuracy of 100%, with both classes showing perfect precision, recall, and F1-scores of 1.00. This indicates that the model correctly classified all instances in the dataset, with no misclassifications.

The ROC curve further supports the model's exceptional performance, with an AUC of 1.00, indicating perfect discrimination capability between the classes. The confusion matrix confirms that all instances were correctly classified, with 25 true negatives and 25 true positives, and no false negatives or false positives.

In conclusion, the SVM model with the chosen parameters is highly effective for this dataset, achieving perfect classification performance. The model's ability to accurately distinguish between the classes is evidenced by the perfect precision, recall, F1-scores, and AUC. This level of performance suggests that the model is highly reliable and robust for the given classification task.

The Figure 15, The result of KNN for second file using first technique:

```
The best parameters are:
 {'metric': 'euclidean', 'n_neighbors': 16, 'weights': 'uniform'}
--------------------------------
classification report:
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        25

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50

--------------------------------
Confusion matrix:
[[25  0]
 [ 0 25]]
--------------------------------
```

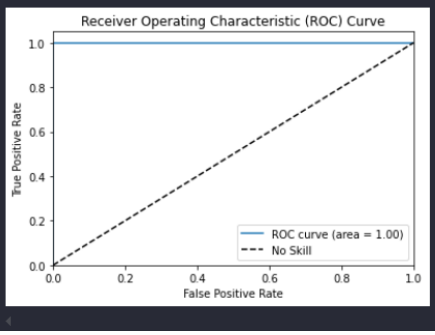

Figure 15: KNN Result on Second File Using First Technique

Interpretations:

The KNN classifier with Euclidean distance, 16 neighbors, and uniform weights shows perfect performance on the given dataset. It achieves an overall accuracy of 100%, with both classes exhibiting perfect precision, recall, and F1-scores of 1.00. This indicates that the model correctly classified all instances in the dataset, with no misclassifications.

The ROC curve further validates the model's exceptional performance, with an AUC of 1.00, indicating perfect discrimination capability between the classes. The confusion matrix confirms that all instances were correctly classified, with 25 true negatives and 25 true positives, and no false negatives or false positives.

In conclusion, the KNN model with the chosen parameters is highly effective for this dataset, achieving perfect classification performance. The model's ability to accurately distinguish between the classes is evidenced by the perfect precision, recall, F1-scores, and AUC. This level of performance suggests that the model is highly reliable and robust for the given classification task. The Figure 16, The result of Logistic Regression for second file using first technique:

```
The best parameters are:
 {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        25
           1       0.96      1.00      0.98        25

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50

------------------------------
Confusion matrix:
[[24  1]
 [ 0 25]]
------------------------------
```

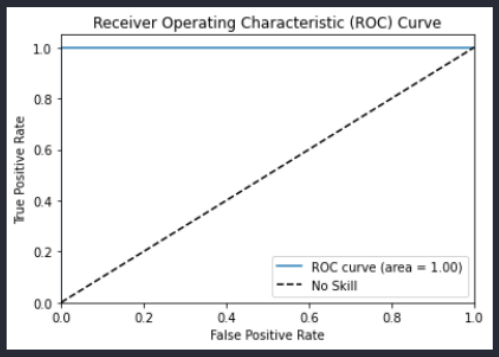

Figure 16: Logistic Regression Result on Second File Using First Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=0.01, and the liblinear solver shows excellent performance, achieving an overall accuracy of 98%. Class 0 demonstrates perfect precision (1.00) and high recall (0.96), leading to an F1-score of 0.98. Class 1 achieves high precision (0.96) and perfect recall (1.00), resulting in an F1-score of 0.98. These results indicate the model is highly effective in distinguishing between the two classes.

The ROC curve with an AUC of 1.00 indicates perfect classification ability, suggesting that the model has an excellent capability to distinguish between the classes. The confusion matrix reveals that most instances are correctly classified, with only 1 instance of Class 0 misclassified, and no instances of Class 1 misclassified. This indicates a strong performance with minimal misclassifications.

In conclusion, the Logistic Regression model with the specified parameters performs exceptionally well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, the slight misclassification of one instance in Class 0 suggests a potential area for further refinement. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes.

2.3 Results of Combination 2

The Figure 17, The result of SVM for first file using second technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
--------------------------------
The best parameters are:
 {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
--------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.88      0.92      0.90        25
           1       0.92      0.88      0.90        25

    accuracy                           0.90        50
   macro avg       0.90      0.90      0.90        50
weighted avg       0.90      0.90      0.90        50

--------------------------------
Confusion matrix:
[[23  2]
 [ 3 22]]
--------------------------------
```

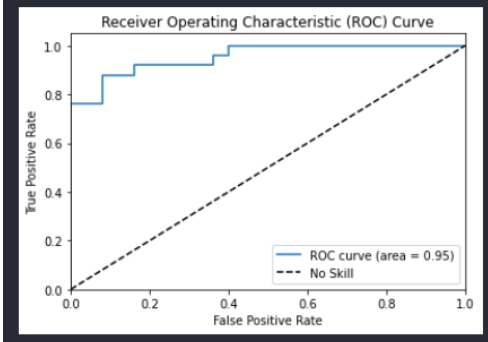

Figure 17: SVM Result on First File Using Second Technique

Interpretations:

The SVM classifier with an RBF kernel, C=10, and gamma set to 'scale' shows strong performance, achieving an overall accuracy of 90%. Class -1 demonstrates high precision (0.88) and recall (0.92), leading to an F1-score of 0.90. Class 1 achieves even higher precision (0.92) but has slightly lower recall (0.88), resulting in an F1-score of 0.90. These metrics indicate the classifier is effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.95 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 23 true negatives and 22 true positives. There are minor misclassifications, with 2 instances of Class -1 and 3 instances of Class 1 incorrectly classified, indicating a small imbalance.

In conclusion, the SVM model with the chosen parameters performs very well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes.

The Figure 18, The result of KNN for first file using second technique:

```
The best parameters are:
 {'metric': 'euclidean', 'n_neighbors': 10, 'weights': 'distance'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.86      0.76      0.81        25
           1       0.79      0.88      0.83        25

    accuracy                           0.82        50
   macro avg       0.82      0.82      0.82        50
weighted avg       0.82      0.82      0.82        50

------------------------------
Confusion matrix:
[[19  6]
 [ 3 22]]
------------------------------
```

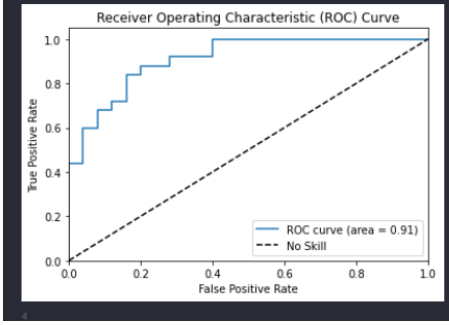

Figure 18: KNN Result on First File Using Second Technique

Interpretations:

The KNN classifier with Euclidean distance, 10 neighbors, and distance-based weights shows reasonable performance, achieving an overall accuracy of 82%. Class -1 demonstrates higher precision (0.86) but lower recall (0.76), resulting in an F1-score of 0.81. Class 1 has lower precision (0.79) but higher recall (0.88), leading to an F1-score of 0.83. These results indicate the model has a balanced but moderate ability to distinguish between the two classes.

The ROC curve with an AUC of 0.91 suggests good discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 19 true negatives and 22 true positives. However, there are notable misclassifications, with 6 instances of Class -1 and 3 instances of Class 1 incorrectly classified, indicating some areas for improvement.

In conclusion, the KNN model with the chosen parameters performs well on this dataset, as evidenced by reasonably high precision, recall, and AUC values. While the model shows overall good performance, addressing the misclassifications and improving the recall for Class -1 could further enhance its robustness. The high AUC value confirms the model's ability to effectively differentiate between the classes, but there is room for improvement in its classification accuracy.

The Figure 19, The result of Logistic Regression for first file using second technique:

```
The best parameters are:
 {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.81      0.88      0.85        25
           1       0.87      0.80      0.83        25

    accuracy                           0.84        50
   macro avg       0.84      0.84      0.84        50
weighted avg       0.84      0.84      0.84        50

------------------------------
Confusion matrix:
[[22  3]
 [ 5 20]]
------------------------------
```

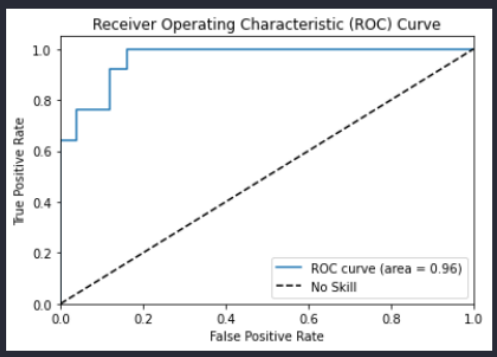

Figure 19: Logistic Regression Result on First File Using Second Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=10, and the liblinear solver shows good performance, achieving an overall accuracy of 84%. Class 0 demonstrates a precision of 0.81 and a recall of 0.88, leading to an F1-score of 0.85. Class 1 achieves a higher precision of 0.87 but a lower recall of 0.80, resulting in an F1-score of 0.83. These metrics indicate the model has a balanced but moderate ability to distinguish between the two classes. The ROC curve with an AUC of 0.96 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 22 true negatives and 20 true positives. However, there are notable misclassifications, with 3 instances of Class 0 and 5 instances of Class 1 incorrectly classified, indicating areas for improvement.

The Figure 20, The result of SVM for second file using second technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
------------------------------
The best parameters are:
 {'C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.89      1.00      0.94        25
           1       1.00      0.88      0.94        25

    accuracy                           0.94        50
   macro avg       0.95      0.94      0.94        50
weighted avg       0.95      0.94      0.94        50

------------------------------
Confusion matrix:
[[25  0]
 [ 3 22]]
------------------------------
```

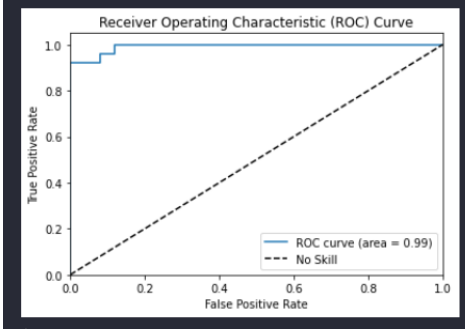

Figure 20: SVM Result on Second File Using Second Technique

Interpretations:

The SVM classifier with an RBF kernel, C=0.1, and gamma set to 'scale' shows strong performance, achieving an overall accuracy of 94%. Class -1 demonstrates high precision (0.89) and perfect recall (1.00), leading to an F1-score of 0.94. Class 1 achieves perfect precision (1.00) but has slightly lower recall (0.88), resulting in an F1-score of 0.94. These metrics indicate the classifier is highly effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.99 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 25 true negatives and 22 true positives. However, there are minor misclassifications, with 3 instances of Class 1 incorrectly classified as Class -1, indicating a slight imbalance.

In conclusion, the SVM model with the chosen parameters performs exceptionally well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a reliable choice for this classification task.

The Figure 21, The result of KNN for second file using second technique:

```
The best parameters are:
 {'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'uniform'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.88      0.88      0.88        25
           1       0.88      0.88      0.88        25

    accuracy                           0.88        50
   macro avg       0.88      0.88      0.88        50
weighted avg       0.88      0.88      0.88        50

------------------------------
Confusion matrix:
[[22  3]
 [ 3 22]]
------------------------------
```

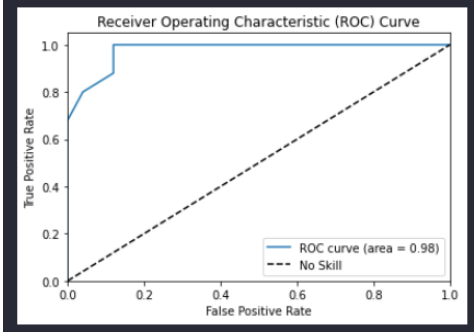

Figure 21: KNN Result on Second File Using Second Technique

Interpretations:

The KNN classifier with Euclidean distance, 9 neighbors, and uniform weights shows strong performance, achieving an overall accuracy of 88%. Both classes demonstrate equal precision (0.88), recall (0.88), and F1-scores (0.88). This indicates the model has a balanced ability to distinguish between the two classes, with consistent performance across both classes.

The ROC curve with an AUC of 0.98 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 22 true negatives and 22 true positives. However, there are a few misclassifications, with 3 instances of Class -1 and 3 instances of Class 1 incorrectly classified, indicating areas for improvement.

In conclusion, the KNN model with the chosen parameters performs well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a strong choice for this classification task.

The Figure 22, The result of Logistic Regression for second file using second technique:

```
The best parameters are:
 {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        25
           1       1.00      0.88      0.94        25

    accuracy                           0.94        50
   macro avg       0.95      0.94      0.94        50
weighted avg       0.95      0.94      0.94        50

------------------------------
Confusion matrix:
[[25  0]
 [ 3 22]]
------------------------------
```

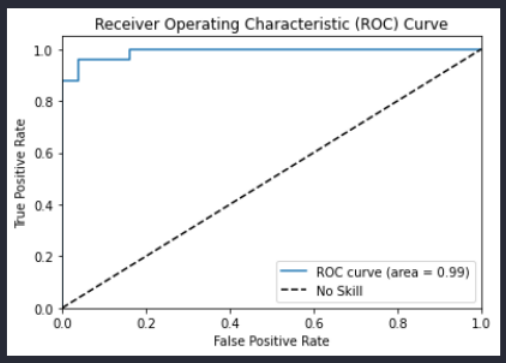

Figure 22: Logistic Regression Result on Second File Using Second Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=1, and the liblinear solver shows strong performance, achieving an overall accuracy of 94%. Class 0 demonstrates high precision (0.89) and perfect recall (1.00), leading to an F1-score of 0.94. Class 1 achieves perfect precision (1.00) but has slightly lower recall (0.88), resulting in an F1-score of 0.94. These metrics indicate the classifier is highly effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.99 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 25 true negatives and 22 true positives. However, there are minor misclassifications, with 3 instances of Class 1 incorrectly classified as Class 0, indicating a slight imbalance.

In conclusion, the Logistic Regression model with the chosen parameters performs exceptionally well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a reliable choice for this classification task.

2.4 Results of Combination 3

The Figure 23, The result SVM for first file using third technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
--------------------------------
The best parameters are:
 {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
--------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.88      0.92      0.90        25
           1       0.92      0.88      0.90        25

    accuracy                           0.90        50
   macro avg       0.90      0.90      0.90        50
weighted avg       0.90      0.90      0.90        50

--------------------------------
Confusion matrix:
[[23  2]
 [ 3 22]]
--------------------------------
```

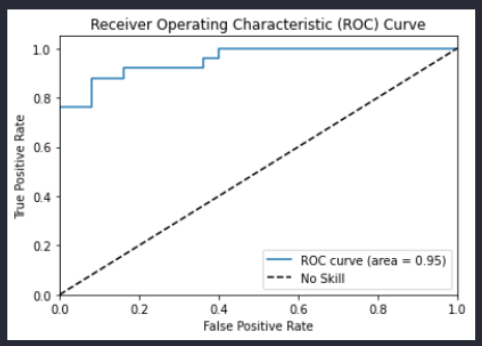

Figure 23: SVM Result on First File Using Third Technique

Interpretations:

The SVM classifier with an RBF kernel, C=10, and gamma set to 'scale' shows strong performance, achieving an overall accuracy of 90%. Class -1 demonstrates high precision (0.88) and recall (0.92), leading to an F1-score of 0.90. Class 1 achieves higher precision (0.92) but has slightly lower recall (0.88), resulting in an F1-score of 0.90. These metrics indicate the classifier is effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.95 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 23 true negatives and 22 true positives. However, there are minor misclassifications, with 2 instances of Class -1 and 3 instances of Class 1 incorrectly classified, indicating a slight imbalance.

In conclusion, the SVM model with the chosen parameters performs very well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a reliable choice for this classification task.

The Figure 24, The result KNN for first file using third technique:

```
The best parameters are:
 {'metric': 'euclidean', 'n_neighbors': 6, 'weights': 'distance'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.79      0.76      0.78        25
           1       0.77      0.80      0.78        25

    accuracy                           0.78        50
   macro avg       0.78      0.78      0.78        50
weighted avg       0.78      0.78      0.78        50

------------------------------
Confusion matrix:
[[19  6]
 [ 5 20]]
------------------------------
```

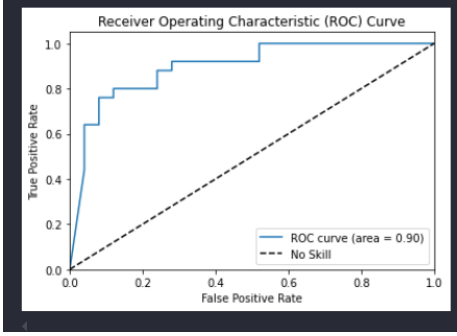

Figure 24: KNN Result on First File Using Third Technique

Interpretations:

The KNN classifier with Euclidean distance, 6 neighbors, and distance-based weights shows moderate performance, achieving an overall accuracy of 78%. Class -1 demonstrates higher precision (0.79) but slightly lower recall (0.76), leading to an F1-score of 0.78. Class 1 has lower precision (0.77) but higher recall (0.80), resulting in an F1-score of 0.78. These results indicate the model has a balanced but moderate ability to distinguish between the two classes.

The ROC curve with an AUC of 0.90 suggests good discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 19 true negatives and 20 true positives. However, there are notable misclassifications, with 6 instances of Class -1 and 5 instances of Class 1 incorrectly classified, indicating areas for improvement.

In conclusion, the KNN model with the chosen parameters performs reasonably well on this dataset, as evidenced by moderate precision, recall, and AUC values. While the model shows overall good performance, addressing the misclassifications and improving the recall for Class -1 could further enhance its robustness. The AUC value confirms the model's ability to differentiate between the classes, but there is room for improvement in its classification accuracy.

The Figure 25, The result Logistic Regression for first file using third technique:

```
The best parameters are:
 {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86        25
           1       0.88      0.84      0.86        25

    accuracy                           0.86        50
   macro avg       0.86      0.86      0.86        50
weighted avg       0.86      0.86      0.86        50

------------------------------
Confusion matrix:
[[22  3]
 [ 4 21]]
------------------------------
```

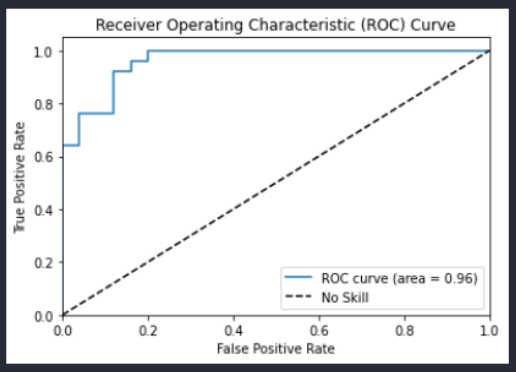

Figure 25: Logistic Regression Result on First File Using Third Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=10, and the liblinear solver shows good performance, achieving an overall accuracy of 86%. Class 0 demonstrates a precision of 0.85 and a recall of 0.88, leading to an F1-score of 0.86. Class 1 achieves a higher precision of 0.88 but a lower recall of 0.84, resulting in an F1-score of 0.86. These metrics indicate the classifier has a balanced ability to distinguish between the two classes.

The ROC curve with an AUC of 0.96 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 22 true negatives and 21 true positives. However, there are some misclassifications, with 3 instances of Class 0 and 4 instances of Class 1 incorrectly classified, indicating areas for improvement.

In conclusion, the Logistic Regression model with the chosen parameters performs well on this dataset, as evidenced by reasonably high precision, recall, and AUC values. While the model shows overall good performance, addressing the misclassifications and improving the recall for Class 1 could further enhance its robustness. The high AUC value confirms the model's ability to effectively differentiate between the classes, but there is room for improvement in its classification accuracy.

The Figure 26, The result SVM for second file using third technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
------------------------------
The best parameters are:
 {'C': 10, 'degree': 2, 'gamma': 'auto', 'kernel': 'poly'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.89      1.00      0.94        25
           1       1.00      0.88      0.94        25

    accuracy                           0.94        50
   macro avg       0.95      0.94      0.94        50
weighted avg       0.95      0.94      0.94        50

------------------------------
Confusion matrix:
[[25  0]
 [ 3 22]]
------------------------------
```

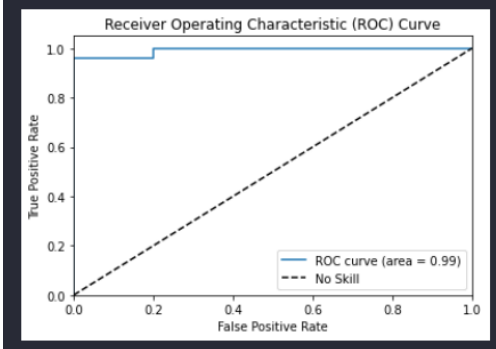

Figure 26: SVM Result on Second File Using Third Technique

Interpretations:

The SVM classifier with a polynomial kernel (degree=2), C=10, and gamma set to 'auto' shows strong performance, achieving an overall accuracy of 94%. Class -1 demonstrates high precision (0.89) and perfect recall (1.00), leading to an F1-score of 0.94. Class 1 achieves perfect precision (1.00) but has slightly lower recall (0.88), resulting in an F1-score of 0.94. These metrics indicate the classifier is highly effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.99 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 25 true negatives and 22 true positives. However, there are minor misclassifications, with 3 instances of Class 1 incorrectly classified as Class -1, indicating a slight imbalance.

In conclusion, the SVM model with the chosen parameters performs exceptionally well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a reliable choice for this classification task.

The Figure 27, The result KNN for second file using third technique:

```
The best parameters are:
 {'metric': 'euclidean', 'n_neighbors': 13, 'weights': 'uniform'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.89      1.00      0.94        25
           1       1.00      0.88      0.94        25

    accuracy                           0.94        50
   macro avg       0.95      0.94      0.94        50
weighted avg       0.95      0.94      0.94        50

------------------------------
Confusion matrix:
[[25  0]
 [ 3 22]]
------------------------------
```

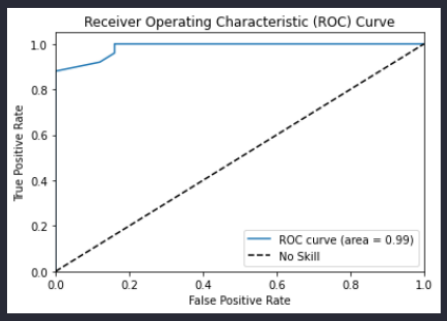

Figure 27: KNN Result on Second File Using Third Technique

Interpretations:

The KNN classifier with Euclidean distance, 13 neighbors, and uniform weights shows strong performance, achieving an overall accuracy of 94%. Class -1 demonstrates high precision (0.89) and perfect recall (1.00), leading to an F1-score of 0.94. Class 1 achieves perfect precision (1.00) but has slightly lower recall (0.88), resulting in an F1-score of 0.94. These metrics indicate the classifier is highly effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.99 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 25 true negatives and 22 true positives. However, there are minor misclassifications, with 3 instances of Class 1 incorrectly classified as Class -1, indicating a slight imbalance.

In conclusion, the KNN model with the chosen parameters performs exceptionally well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a reliable choice for this classification task.

The Figure 28, The result Logistic Regression for second file using third technique:

```
The best parameters are:
 {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        25
           1       1.00      0.88      0.94        25

    accuracy                           0.94        50
   macro avg       0.95      0.94      0.94        50
weighted avg       0.95      0.94      0.94        50

------------------------------
Confusion matrix:
[[25  0]
 [ 3 22]]
------------------------------
```

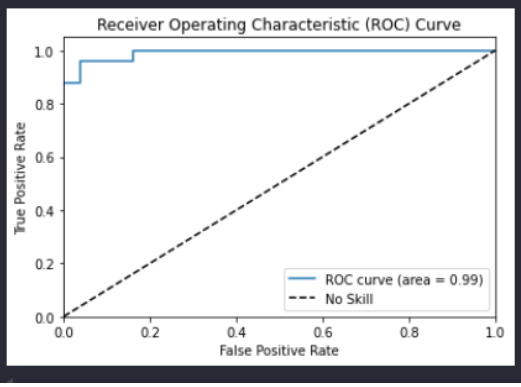

Figure 28: Logistic Regression Result on Second File Using Third Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=1, and the liblinear solver shows strong performance, achieving an overall accuracy of 94%. Class 0 demonstrates high precision (0.89) and perfect recall (1.00), leading to an F1-score of 0.94. Class 1 achieves perfect precision (1.00) but has slightly lower recall (0.88), resulting in an F1-score of 0.94. These metrics indicate the classifier is highly effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.99 suggests excellent discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 25 true negatives and 22 true positives. However, there are minor misclassifications, with 3 instances of Class 1 incorrectly classified as Class 0, indicating a slight imbalance.

In conclusion, the Logistic Regression model with the chosen parameters performs exceptionally well on this dataset, as evidenced by high precision, recall, and AUC values. While the model shows strong overall performance, addressing the minor misclassifications could further enhance its robustness. The high AUC value confirms the model's reliability and its capability to effectively differentiate between the classes, making it a reliable choice for this classification task.

2.5 Results of Combination 4

The Figure 29, The result SVM for first file using fourth technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
--------------------------------
The best parameters are:
 {'C': 1, 'degree': 4, 'gamma': 'auto', 'kernel': 'poly'}
--------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.64      0.56      0.60        25
           1       0.61      0.68      0.64        25

    accuracy                           0.62        50
   macro avg       0.62      0.62      0.62        50
weighted avg       0.62      0.62      0.62        50

--------------------------------
Confusion matrix:
[[14 11]
 [ 8 17]]
--------------------------------
```

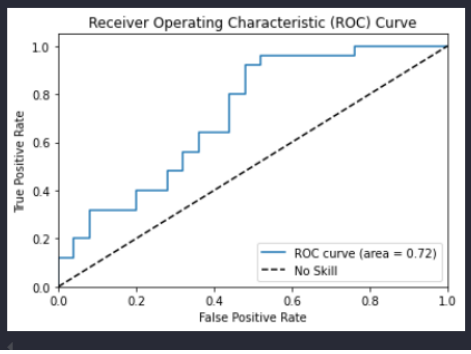

Figure 29: SVM Result on First File Using Fourth Technique

Interpretations:

The SVM classifier with a polynomial kernel (degree=4), C=1, and gamma set to 'auto' shows moderate performance, achieving an overall accuracy of 62%. Class -1 demonstrates lower precision (0.64) and recall (0.56), leading to an F1-score of 0.60. Class 1 achieves similar precision (0.61) and higher recall (0.68), resulting in an F1-score of 0.64. These metrics indicate the classifier has limited effectiveness in distinguishing between the two classes.

The ROC curve with an AUC of 0.72 suggests moderate discrimination capability between the classes. The confusion matrix reveals that a significant number of instances are incorrectly classified, with 11 false negatives for Class -1 and 8 false positives for Class 1. This indicates a substantial area for improvement in the model's classification accuracy.

In conclusion, the SVM model with the chosen parameters performs moderately well on this dataset, as evidenced by the moderate precision, recall, and AUC values. The model shows some ability to differentiate between the classes, but the high number of misclassifications suggests it needs significant improvements. Future efforts could focus on tuning the model further or exploring alternative algorithms to achieve better performance and higher classification accuracy.

The Figure 30, The result KNN for first file using fourth technique:

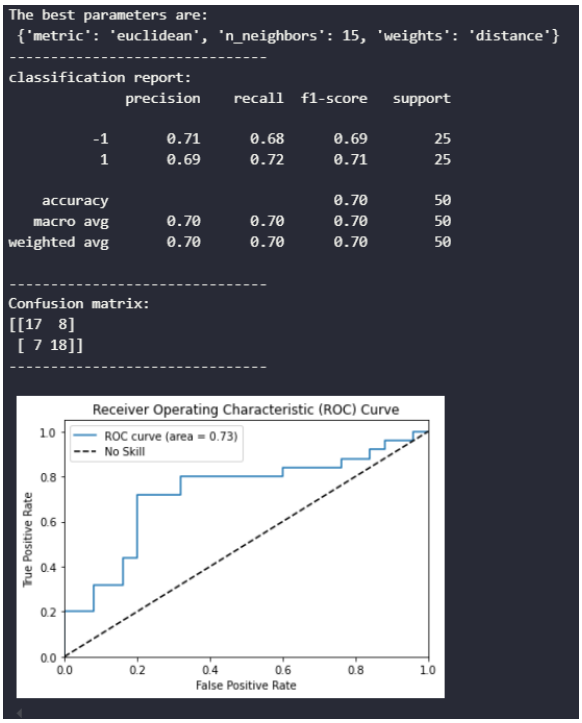

Figure 30: KNN Result on First File Using Fourth Technique

Interpretations:

The KNN classifier with Euclidean distance, 15 neighbors, and distance-based weights shows moderate performance, achieving an overall accuracy of 70%. Class -1 demonstrates slightly higher precision (0.71) but lower recall (0.68), leading to an F1-score of 0.69. Class 1 has lower precision (0.69) but higher recall (0.72), resulting in an F1-score of 0.71. These results indicate the classifier has a balanced but moderate ability to distinguish between the two classes.

The ROC curve with an AUC of 0.73 suggests moderate discrimination capability between the classes. The confusion matrix reveals that a significant number of instances are incorrectly classified, with 8 false negatives for Class -1 and 7 false positives for Class 1. This indicates a substantial area for improvement in the model's classification accuracy.

In conclusion, the KNN model with the chosen parameters performs moderately well on this dataset, as evidenced by moderate precision, recall, and AUC values. The model shows some ability to differentiate between the classes, but the high number of misclassifications suggests it needs significant improvements. Future efforts could focus on tuning the model further or exploring alternative algorithms to achieve better performance and higher classification accuracy.

The Figure 31, The result Logistic Regression for first file using fourth technique:

```
The best parameters are:
 {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.57      0.64      0.60        25
           1       0.59      0.52      0.55        25

    accuracy                           0.58        50
   macro avg       0.58      0.58      0.58        50
weighted avg       0.58      0.58      0.58        50

------------------------------
Confusion matrix:
[[16  9]
 [12 13]]
------------------------------
```

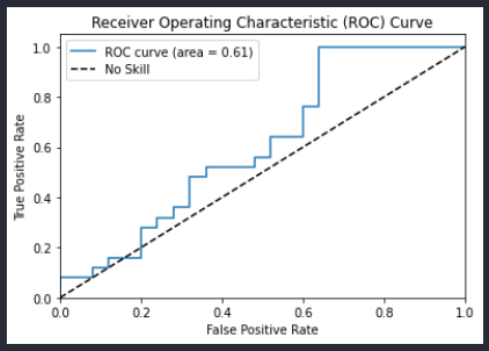

Figure 31: Logistic Regression on First File Using Fourth Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=0.1, and the liblinear solver shows limited performance, achieving an overall accuracy of 58%. Class 0 demonstrates slightly lower precision (0.57) but higher recall (0.64), leading to an F1-score of 0.60. Class 1 has higher precision (0.59) but lower recall (0.52), resulting in an F1-score of 0.55. These metrics indicate the classifier has limited effectiveness in distinguishing between the two classes. The ROC curve with an AUC of 0.61 suggests poor discrimination capability between the classes. The confusion matrix reveals a significant number of instances are incorrectly classified, with 9 false negatives for Class 0 and 12 false positives for Class 1. This indicates a substantial area for improvement in the model's classification accuracy. In conclusion, the Logistic Regression model with the chosen parameters performs poorly on this dataset, as evidenced by low precision, recall, and AUC values. The model shows limited ability to differentiate between the classes, with a high number of misclassifications. Future efforts could focus on tuning the model further or exploring alternative algorithms to achieve better performance and higher classification accuracy.

The Figure 32, The result SVM for second file using fourth technique:

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
------------------------------
The best parameters are:
 {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.73      0.88      0.80        25
           1       0.85      0.68      0.76        25

    accuracy                           0.78        50
   macro avg       0.79      0.78      0.78        50
weighted avg       0.79      0.78      0.78        50

------------------------------
Confusion matrix:
[[22  3]
 [ 8 17]]
------------------------------
```

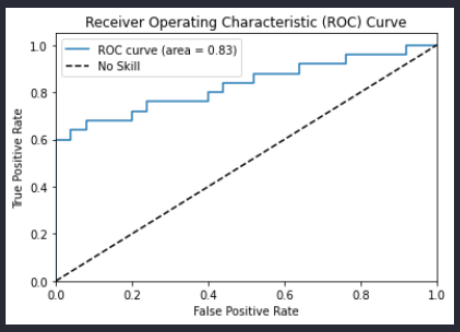

Figure 32: SVM Result on Second File Using Fourth Technique

Interpretations:

The SVM classifier with an RBF kernel, C=1, and gamma set to 'auto' shows moderate performance, achieving an overall accuracy of 78%. Class -1 demonstrates lower precision (0.73) but higher recall (0.88), leading to an F1-score of 0.80. Class 1 has higher precision (0.85) but lower recall (0.68), resulting in an F1-score of 0.76. These results indicate the classifier has a balanced but moderate ability to distinguish between the two classes.

The ROC curve with an AUC of 0.83 suggests good discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 22 true negatives and 17 true positives. However, there are some misclassifications, with 3 instances of Class -1 and 8 instances of Class 1 incorrectly classified, indicating areas for improvement.

In conclusion, the SVM model with the chosen parameters performs reasonably well on this dataset, as evidenced by moderate precision, recall, and AUC values. The model shows a balanced ability to differentiate between the classes, but addressing the misclassifications could further enhance its robustness. The AUC value confirms the model's capability to effectively distinguish between the classes, but there is room for improvement in its classification accuracy.

The Figure 33, The result KNN for second file using fourth technique:

```
The best parameters are:
 {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}
------------------------------
classification report:
              precision    recall  f1-score   support

          -1       0.65      0.80      0.71        25
           1       0.74      0.56      0.64        25

    accuracy                           0.68        50
   macro avg       0.69      0.68      0.68        50
weighted avg       0.69      0.68      0.68        50

------------------------------
Confusion matrix:
[[20  5]
 [11 14]]
------------------------------
```

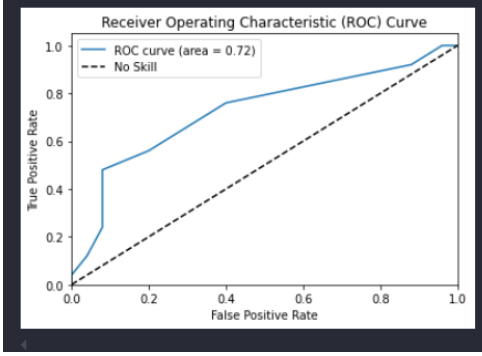

Figure 33: KNN Result on Second File Using Fourth Technique

Interpretations:

The KNN classifier with Manhattan distance, 9 neighbors, and uniform weights shows moderate performance, achieving an overall accuracy of 68%. Class -1 demonstrates higher precision (0.65) and recall (0.80), leading to an F1-score of 0.71. Class 1 has higher precision (0.74) but lower recall (0.56), resulting in an F1-score of 0.64. These results indicate the classifier has a balanced but moderate ability to distinguish between the two classes.

The ROC curve with an AUC of 0.72 suggests moderate discrimination capability between the classes. The confusion matrix reveals that a significant number of instances are incorrectly classified, with 5 false negatives for Class -1 and 11 false positives for Class 1. This indicates a substantial area for improvement in the model's classification accuracy.

In conclusion, the KNN model with the chosen parameters performs moderately well on this dataset, as evidenced by moderate precision, recall, and AUC values. The model shows some ability to differentiate between the classes, but addressing the high number of misclassifications could further enhance its robustness. Future efforts could focus on tuning the model further or exploring alternative algorithms to achieve better performance and higher classification accuracy.

The Figure 34, The result Logistic Regression for second file using fourth technique:

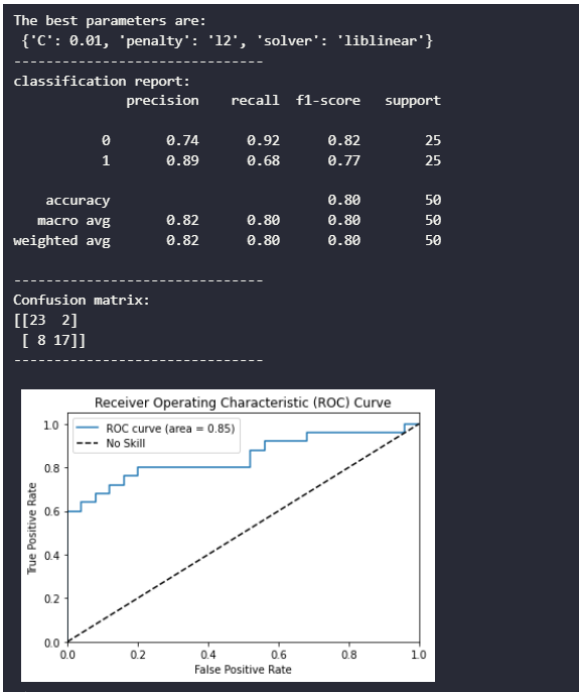

Figure 34: Logistic Regression Result on Second File Using Fourth Technique

Interpretations:

The Logistic Regression model with an L2 penalty, C=0.01, and the liblinear solver shows good performance, achieving an overall accuracy of 80%. Class 0 demonstrates high precision (0.74) and high recall (0.92), leading to an F1-score of 0.82. Class 1 achieves higher precision (0.89) but has lower recall (0.68), resulting in an F1-score of 0.77. These metrics indicate the classifier is fairly effective in distinguishing between the two classes.

The ROC curve with an AUC of 0.85 suggests good discrimination capability between the classes. The confusion matrix reveals that most instances are correctly classified, with 23 true negatives and 17 true positives. However, there are some misclassifications, with 2 instances of Class 0 and 8 instances of Class 1 incorrectly classified, indicating areas for improvement.

In conclusion, the Logistic Regression model with the chosen parameters performs well on this dataset, as evidenced by reasonably high precision, recall, and AUC values. While the model shows overall good performance, addressing the misclassifications and improving the recall for Class 1 could further enhance its robustness. The AUC value confirms the model's ability to effectively differentiate between the classes, but there is room for improvement in its classification accuracy.

2.6 Summary

Based on the accuracy, the summary presented in Table 2. According to Table 2, single CSP(First Combination) are so effective without PCA and ICA. Either first or second file has the highest accuracy.However, last combination where PCA and ICA and CSP was applied to data, has worse accuracy amon all.

In summary, on average single CSP or ICA+CSP or PCA + ICA combinations has very good scores.Note that number of components of each are so crucial to see how it would effect these models.(We have tried multiple components but not the all of them!)

	SVM		KNN		Logistic Regression	
	First file	Second	First	Second	First	Second
Preprocess						
Combination 1	0.90	1.00	0.92	1.00	0.90	0.98
Combination 2	0.90	0.94	0.82	0.88	0.84	0.94
Combination 3	0.90	0.94	0.78	0.94	0.86	0.94
Combination 3	0.62	0.78	0.70	0.68	0.58	0.80

Table 2: Performance metrics for different classifiers and combinations

3 Clustering

3.1 Implementation

We also implement a class in order to use it to have two different kind of clustering.Clustering methods we consider to apply on our dataset are:

1. Kmeans: Fully discussed in the our ML course but in summary, the algorithm is like Expected Maximization.
2. Agglomerative clustering: is a type of hierarchical clustering method used to group objects based on their similarity. It works by iteratively merging the closest pairs of clusters until all objects are in a single cluster or until a predefined stopping condition is met

Before applying any clustering, we define two useful methods.'_calculate_purity' will calculate the preplexity based on the contingency table:

```
    def _calculate_purity(self, y_true:np.array, y_pred:np.array) -> float:
        contingency_matrix = confusion_matrix(y_true, y_pred)
        return np.sum(np.max(contingency_matrix, axis=0)) / np.sum(
            ↪ contingency_matrix)
```

The other method will transform data to see the clusters and how they are separated and correctly implemented.

```
    def _plot_clusters(self, top_3_models:list, X:np.array, top_scores:list[float
        ↪ ], top_clusters:list[int], kind:str) -> None:
        tsne = TSNE(n_components=2, random_state=42, init='random')
        X_tsne = tsne.fit_transform(X)
        fig, axes = plt.subplots(1, 3, figsize=(20, 7))

        for i, model in enumerate(top_3_models):
            labels = model.labels_

            axes[i].scatter(X_tsne[:, 0], X_tsne[:, 1], c=labels, s=50, cmap='
                ↪ viridis')
            axes[i].set_title(f'{kind} with {top_clusters[i]} clusters\nSilhouette
                ↪  Score: {top_scores["score"].iloc[i]:.4f}')
            axes[i].set_xlabel('t-SNE Component 1')
            axes[i].set_ylabel('t-SNE Component 2')

```

```
        plt.tight_layout()
        plt.show()
```

This will plot top 3 clusters(3 best clusters) to interpret the result.
For clustering, we use 'clusteval' library. This is a popular library which based on a defined model(let's say kmean) and score(like silhouette score), can find the best cluster!Now lets see how it would be implemented for kmeans:

```
    def apply_kmeans(self, X:np.array, y:np.array, random_state:int=42) -> None:
            ce = clusteval(cluster='kmeans', evaluate='silhouette')
            results = ce.fit(X)
            top_3_scores = results['score'].nlargest(3, 'score')
            top_3_clusters = top_3_scores['clusters'].values
            print("--------------------------------")
            print("Top 3 scores:")
            print(top_3_scores)
            print("--------------------------------")
            print("Number of clusters for the top 3 scores:", top_3_clusters)
            top_3_models = []
            for n_clusters in top_3_clusters:
                kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
                kmeans.fit(X)
                top_3_models.append(kmeans)
            for i, model in enumerate(top_3_models):
                labels = model.labels_
                purity = self._calculate_purity(y, labels)
                print(f"Purity for model {i+1} with {top_3_clusters[i]} clusters:
                   ↪ {purity:.4f}")
            self._plot_clusters(top_3_models=top_3_models, X=X, top_scores=
               ↪ top_3_scores, top_clusters=top_3_clusters, kind="K-means")
```

The implementation is so simple. You must define which model you want and a evaluate score. Then based on the results, we will train top 3 clusters with sklearn. For each top 3 clusters, we will print the purity and plot them using method '_plot_clusters'.
Agglomerative is just like kmeans in our implementation.

```
    def apply_agglomerative(self, X:np.array, y:np.array) -> None:
        ce = clusteval(cluster='agglomerative', evaluate='silhouette')
        results = ce.fit(X)
        top_3_scores = results['score'].nlargest(3, 'score')
        top_3_clusters = top_3_scores['clusters'].values
        print("--------------------------------")
        print("Top 3 scores:")
        print(top_3_scores)
        print("--------------------------------")
        print("Number of clusters for the top 3 scores:", top_3_clusters)
        top_3_models = []
        for n_clusters in top_3_clusters:
            aggle = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
            aggle.fit(X)
```

```
            top_3_models.append(aggle)
        for i, model in enumerate(top_3_models):
            labels = model.labels_
            purity = self._calculate_purity(y, labels)
            print(f"Purity for model {i+1} with {top_3_clusters[i]} clusters: {
                ↪ purity:.4f}")
        self._plot_clusters(top_3_models=top_3_models, X=X, top_scores=
            ↪ top_3_scores, top_clusters=top_3_clusters, kind="Agglomerative")
```

Now lets again see how it would work on 4 combination of preprocessing.

3.2 Apply Cluster on Combination 1

Figure 35, is the result of Kmeans for first file using first technique:

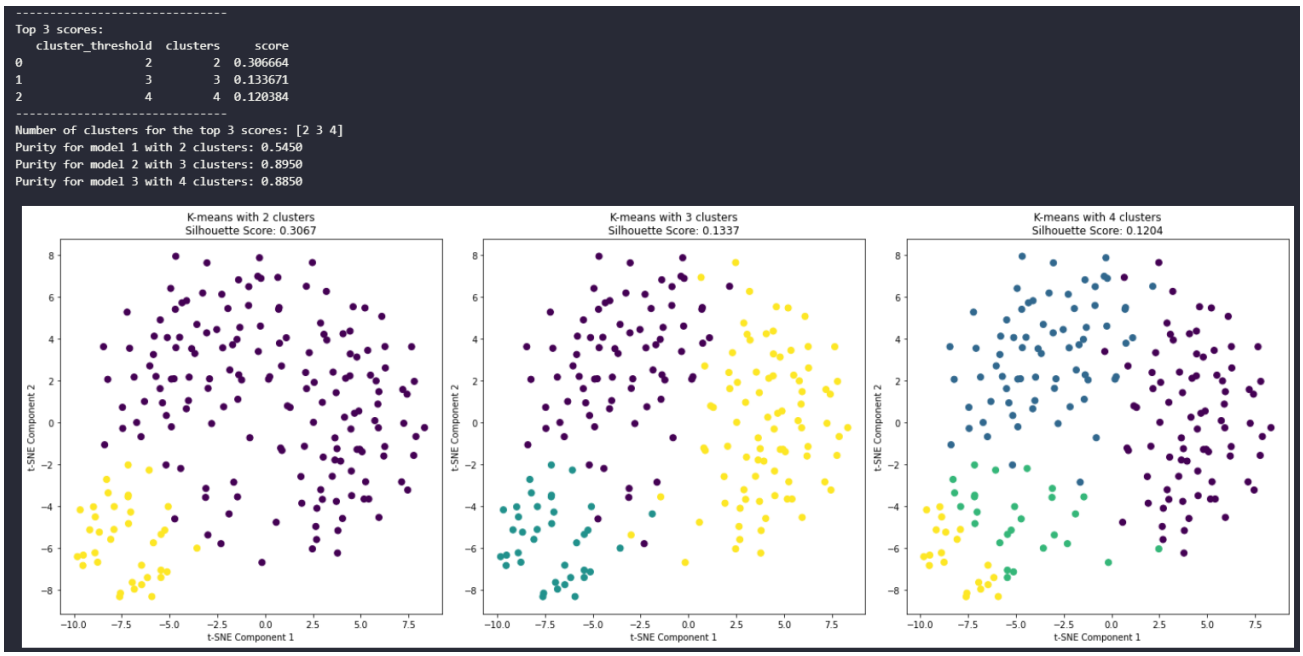

Figure 35: Kmeans Result For First File Using First Technique

- Cluster Analysis:
  - **K-means with 2 Clusters:**
    - The K-means model with 2 clusters has the highest silhouette score (0.3067), indicating relatively better-defined clusters compared to the other models.
    - However, it has the lowest purity (0.5450), suggesting that the clusters are not well-aligned with the true labels.
    - Visualization shows two broad clusters, but with considerable overlap, indicating mixed groupings.

  - K-means with 3 Clusters:
    * The K-means model with 3 clusters has a lower silhouette score (0.1337), indicating less well-defined clusters.
    * It has the highest purity (0.8950), suggesting a good alignment with the true labels.
    * Visualization shows three distinct clusters with some overlap, indicating a more nuanced grouping.
  - K-means with 4 Clusters:
    * The K-means model with 4 clusters has the lowest silhouette score (0.1204), indicating the least well-defined clusters.
    * It has a high purity (0.8850), slightly lower than the 3-cluster model but still indicating good alignment with true labels.
    * Visualization shows four clusters with considerable overlap, indicating mixed and possibly over-split groupings.

- The three plots show different levels of clustering performance. The K-means model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The K-means model with 3 clusters has the highest purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The K-means model with 4 clusters, despite having a high purity, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.
- In conclusion, the K-means model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the K-means model with 4 clusters, despite good alignment, suffers from poorly defined clusters.

Since we say all information about the clusters, for having consice report, we will avoid to completely explain all.We will summarize the interpretation of each.
Figure 36, is the result of Agglomerative for first file using first technique:

```
--------------------------------
Top 3 scores:
   cluster_threshold  clusters     score
0                  2         2  0.317188
1                  3         3  0.110794
3                  5         5  0.105235
--------------------------------
Number of clusters for the top 3 scores: [2 3 5]
Purity for model 1 with 2 clusters: 0.5600
Purity for model 2 with 3 clusters: 0.8700
Purity for model 3 with 5 clusters: 0.9150
```

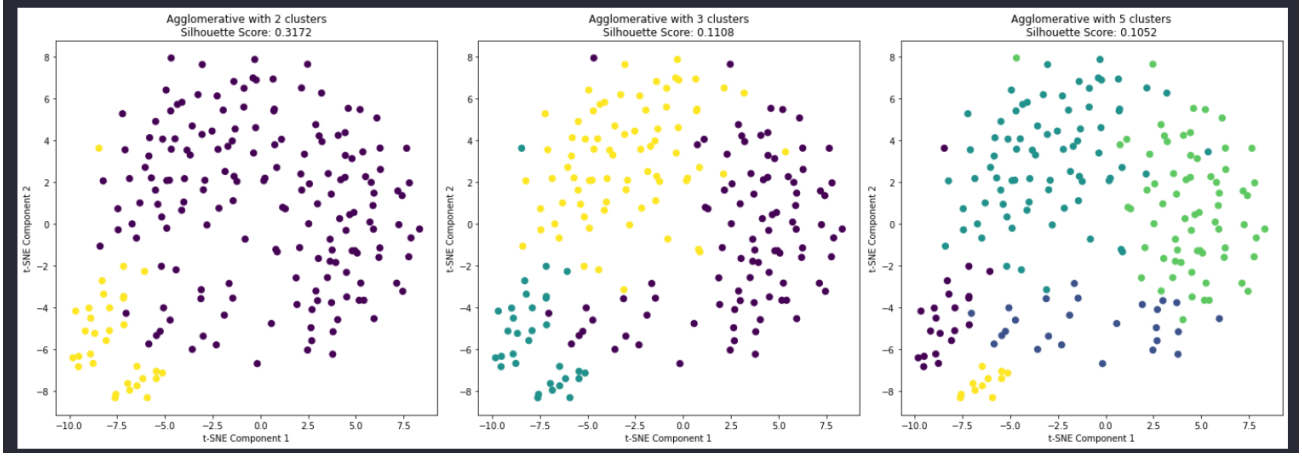

Figure 36: Agglomerative Result For First File Using First Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The agglomerative clustering model with 3 clusters has a high purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The agglomerative clustering model with 5 clusters, despite having the highest purity, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the agglomerative clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the agglomerative clustering model with 5 clusters, despite good alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 5-cluster model showing high alignment but least defined clusters.

Figure 37, is the result of Kmeans for second file using first technique:

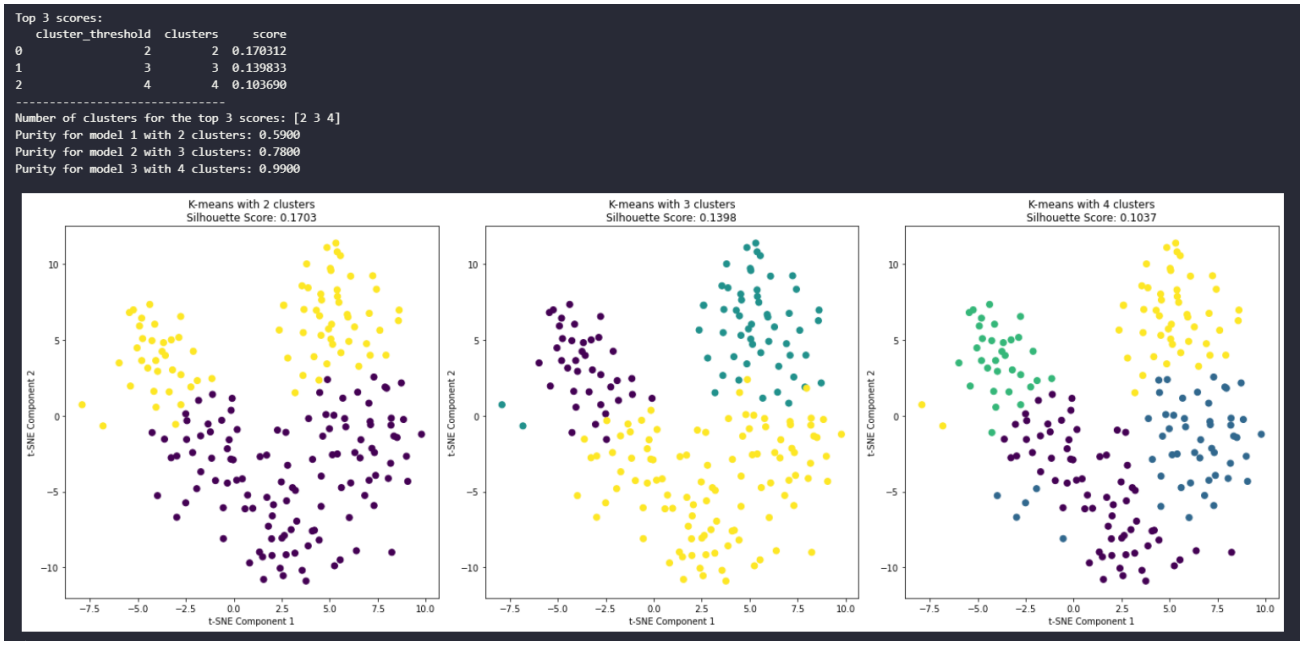

Figure 37: Kmeans Result For Second File Using First Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The K-means model with 3 clusters has a higher purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The K-means model with 4 clusters, despite having the highest purity, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the K-means model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the K-means model with 4 clusters, despite excellent alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing excellent alignment but least defined clusters.

Figure 38, is the result of Agglomerative for Second file using first technique:

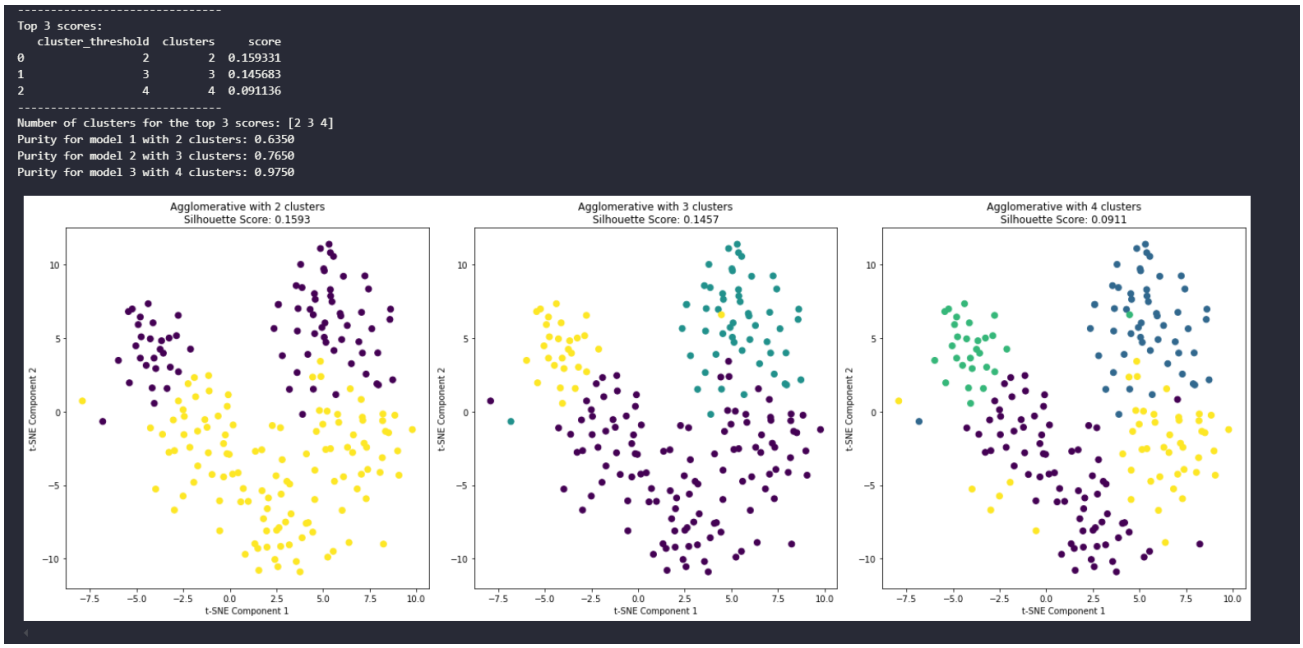

Figure 38: Agglomerative Result For Second File Using First Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The agglomerative clustering model with 3 clusters has a higher purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The agglomerative clustering model with 4 clusters, despite having the highest purity, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the agglomerative clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the agglomerative clustering model with 4 clusters, despite excellent alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing excellent alignment but least defined clusters.

3.3 Apply Cluster on Combination 2

Figure 39, is the result of Kmeans for first file using second technique:

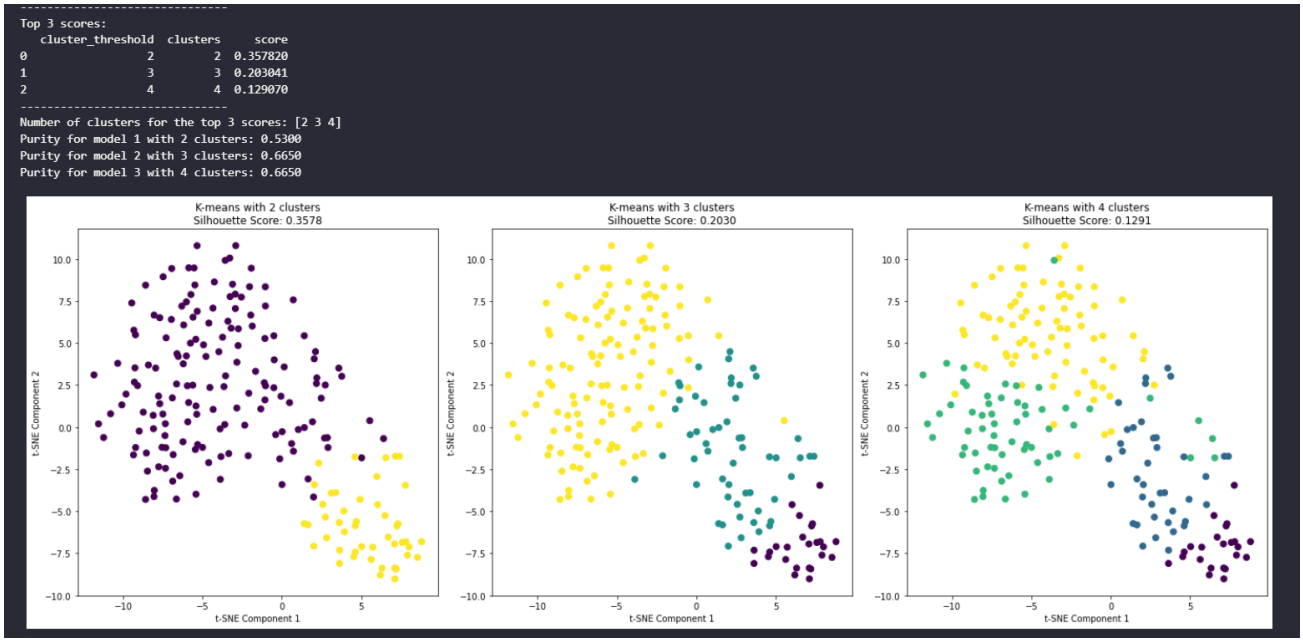

Figure 39: Kmeans Result For First File Using Second Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The K-means model with 3 clusters has a higher purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The K-means model with 4 clusters, despite having the same purity as the 3-cluster model, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the K-means model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the K-means model with 4 clusters, despite having good alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing good alignment but least defined clusters.

Figure 40, is the result of Agglomerative for first file using second technique:

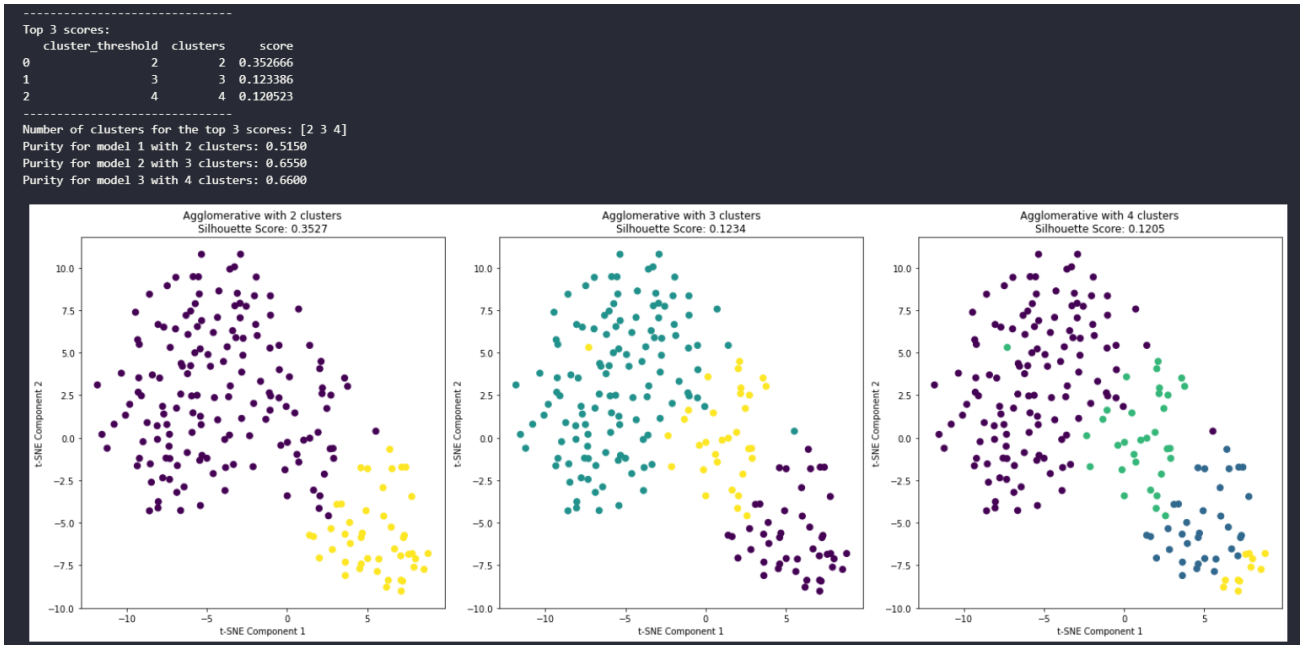

Figure 40: Agglomerative Result For First File Using Second Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The agglomerative clustering model with 3 clusters has a higher purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The agglomerative clustering model with 4 clusters, despite having the highest purity, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the agglomerative clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the agglomerative clustering model with 4 clusters, despite having good alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing good alignment but least defined clusters.

Figure 41, is the result of Kmeans for second file using second technique:

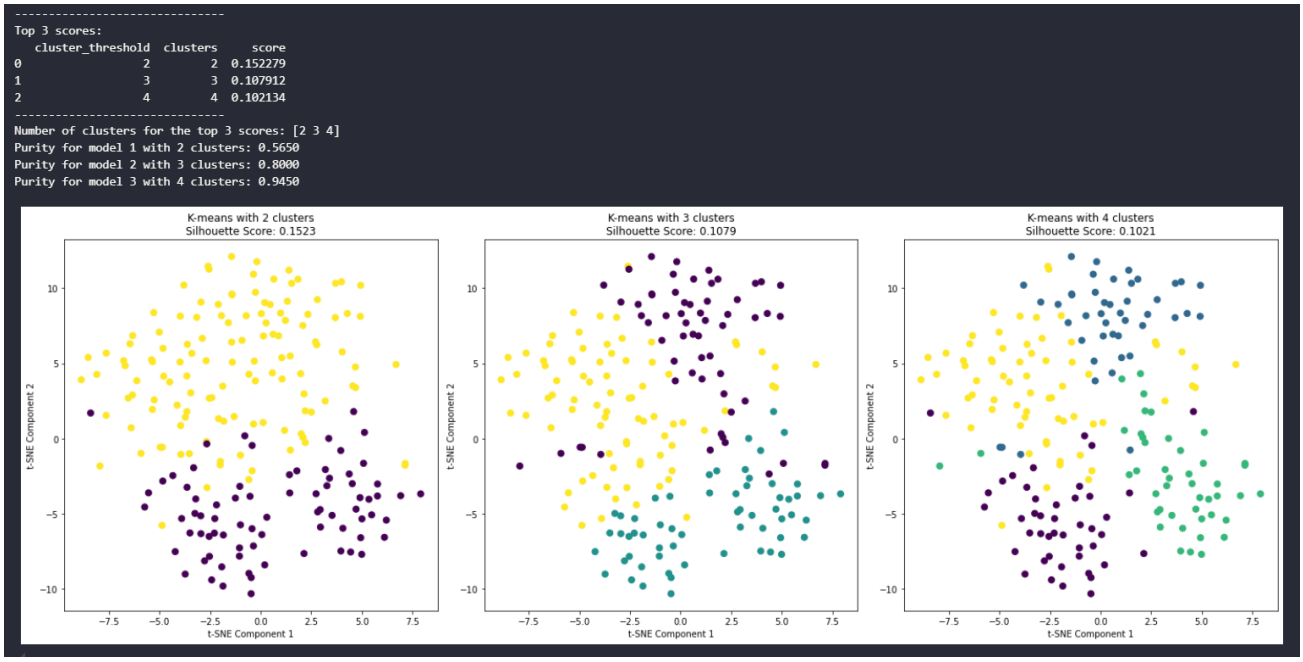

Figure 41: Kmeans Result For Second File Using Second Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The K-means model with 3 clusters has a higher purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The K-means model with 4 clusters, despite having the highest purity, shows the lowest silhouette score, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the K-means model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the K-means model with 4 clusters, despite excellent alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing excellent alignment but least defined clusters.

Figure 42, is the result of Agglomerative for second file using second technique:

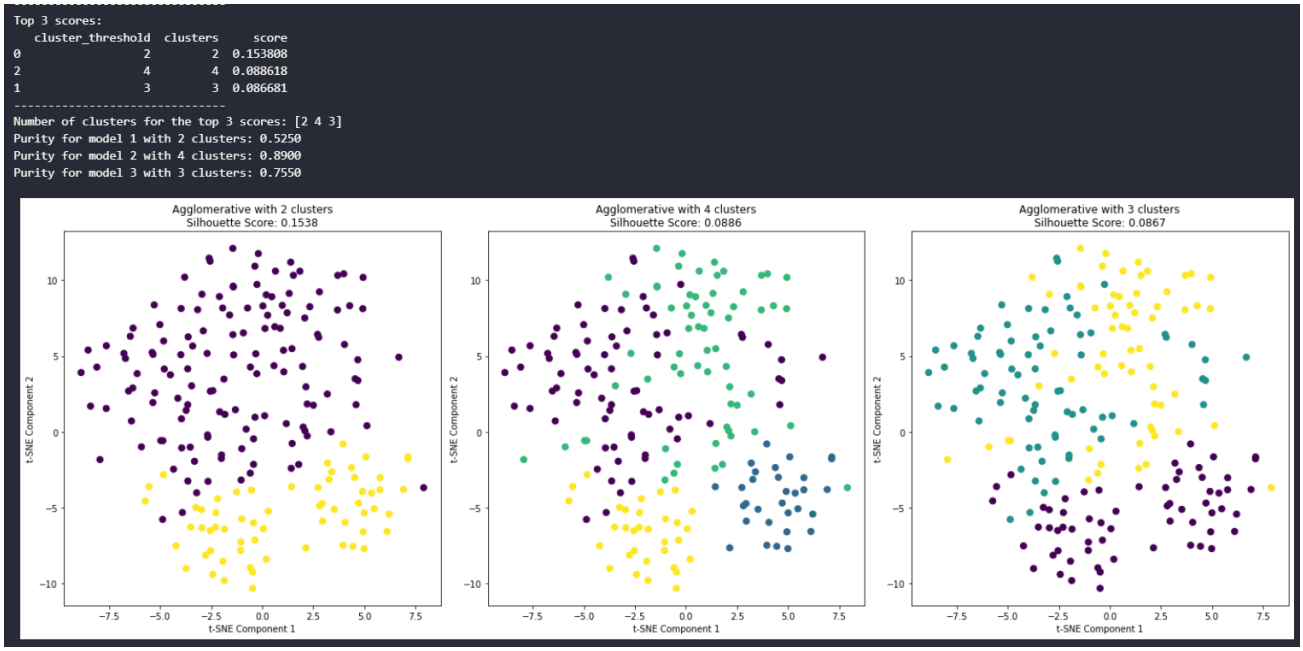

Figure 42: Agglomerative Result For Second File Using Second Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The agglomerative clustering model with 4 clusters has a higher purity and a moderate silhouette score, suggesting it strikes a good balance between well-defined clusters and alignment with true labels. The agglomerative clustering model with 3 clusters, despite having the lowest silhouette score, shows a moderate purity, indicating that the clusters are not well-defined and may be over-split.

In conclusion, the agglomerative clustering model with 4 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the agglomerative clustering model with 3 clusters, despite having a moderate alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 4-cluster model showing a balanced performance, and the 3-cluster model showing moderate alignment but least defined clusters.

3.4 Apply Cluster on Combination 3

Figure 43, is the result of Kmeans for first file using Third technique:

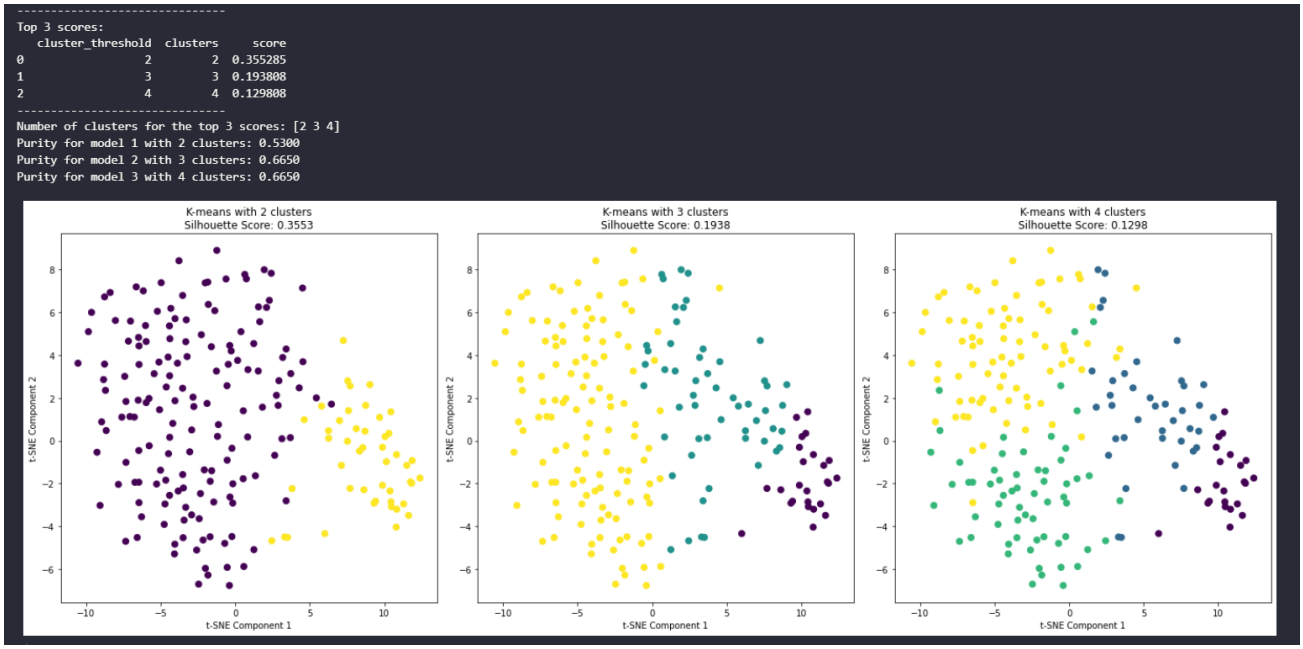

Figure 43: Kmeans Result For First File Using Third Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means clustering model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The K-means clustering model with 3 clusters shows a moderate silhouette score and higher purity, suggesting better-defined clusters and better alignment with true labels. The K-means clustering model with 4 clusters, despite having the lowest silhouette score, shows similar alignment with true labels as the 3-cluster model but suffers from poorly defined clusters.

In conclusion, the K-means clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means clustering model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the K-means clustering model with 4 clusters, despite having similar alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing similar alignment but least defined clusters.

Figure 44, is the result of Agglomerative for first file using third technique:

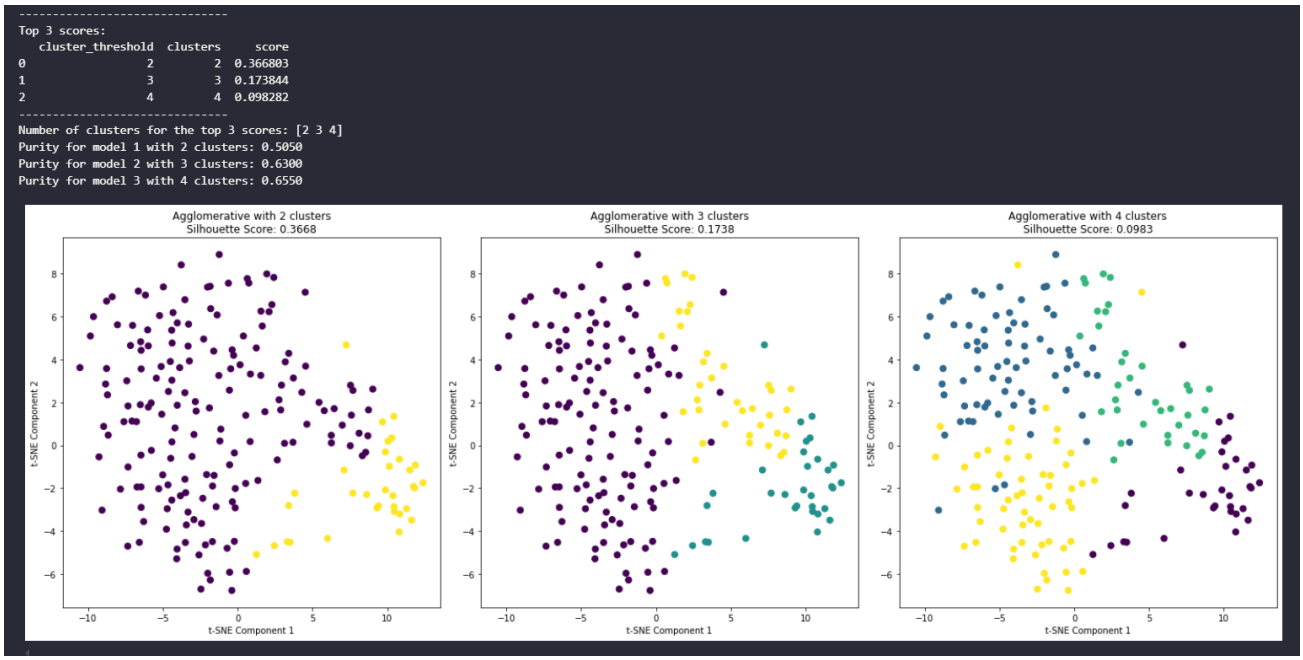

Figure 44: Agglomerative Result For First File Using Third Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but the lowest purity, indicating that while the clusters are well-defined, they do not match the true labels well. The agglomerative clustering model with 3 clusters shows a moderate silhouette score and higher purity, suggesting better-defined clusters and better alignment with true labels. The agglomerative clustering model with 4 clusters, despite having the lowest silhouette score, shows the highest alignment with true labels but suffers from poorly defined clusters.

In conclusion, the agglomerative clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align well with the true labels, and the agglomerative clustering model with 4 clusters, despite having the highest alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but lowest alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing the highest alignment but least defined clusters.

Figure 45, is the result of Kmeans for second file using Third technique:

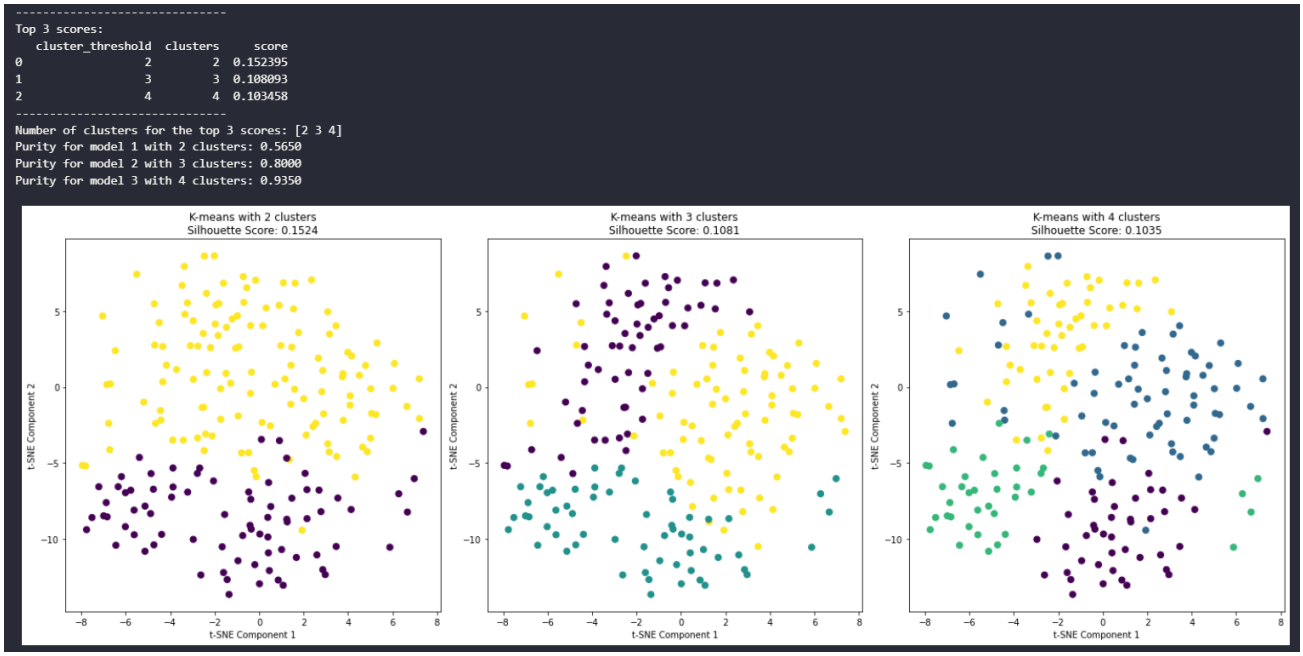

Figure 45: Kmeans Result For Second File Using Third Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means clustering model with 2 clusters has the highest silhouette score but moderate purity, indicating that while the clusters are well-defined, they do not match the true labels perfectly. The K-means clustering model with 3 clusters shows a lower silhouette score but higher purity, suggesting less well-defined clusters but better alignment with true labels. The K-means clustering model with 4 clusters, despite having the lowest silhouette score, shows the highest alignment with true labels but suffers from poorly defined clusters.

In conclusion, the K-means clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means clustering model with 2 clusters, while having well-defined clusters, does not align as well with the true labels, and the K-means clustering model with 4 clusters, despite having the highest alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but moderate alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing the highest alignment but least defined clusters.

Figure 46, is the result of Agglomerative for second file using third technique:

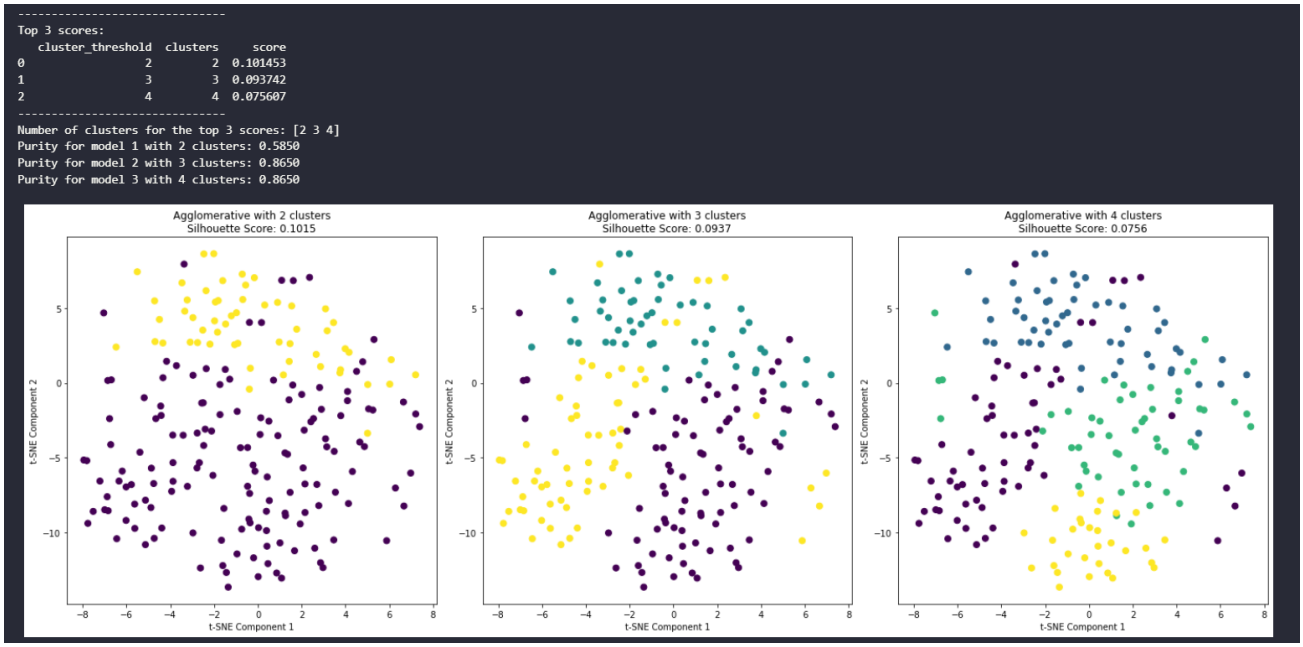

Figure 46: Agglomerative Result For Second File Using Third Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but moderate purity, indicating that while the clusters are well-defined, they do not match the true labels perfectly. The agglomerative clustering model with 3 clusters shows a lower silhouette score but higher purity, suggesting less well-defined clusters but better alignment with true labels. The agglomerative clustering model with 4 clusters, despite having the lowest silhouette score, shows good alignment with true labels but suffers from poorly defined clusters.

In conclusion, the agglomerative clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align as well with the true labels, and the agglomerative clustering model with 4 clusters, despite having good alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but moderate alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing good alignment but least defined clusters.

3.5 Apply Cluster on Combination 4

Figure 47, is the result of Kmeans for first file using fourth technique:

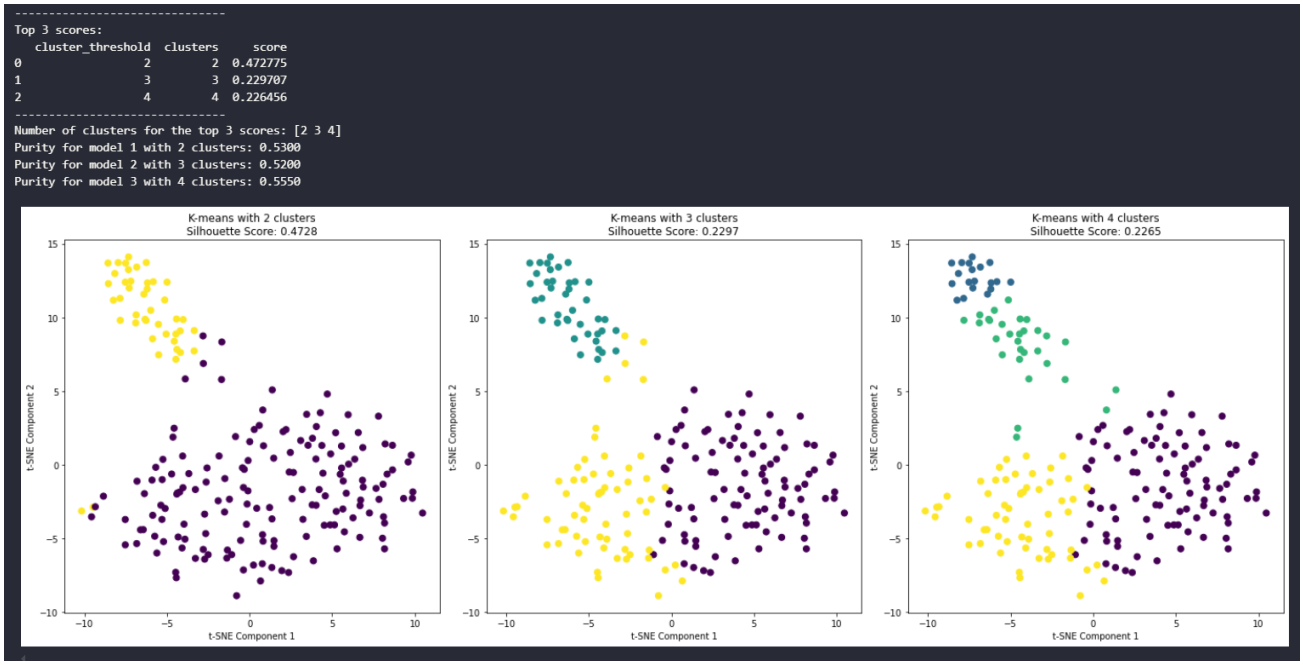

Figure 47: Kmeans Result For First File Using Forth Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means clustering model with 2 clusters has the highest silhouette score but moderate purity, indicating that while the clusters are well-defined, they do not match the true labels perfectly. The K-means clustering model with 3 clusters shows a lower silhouette score and slightly lower purity, suggesting less well-defined clusters and fair alignment with true labels. The K-means clustering model with 4 clusters, despite having the lowest silhouette score, shows good alignment with true labels but suffers from poorly defined clusters.

In conclusion, the K-means clustering model with 4 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means clustering model with 2 clusters, while having well-defined clusters, does not align as well with the true labels, and the K-means clustering model with 3 clusters, despite having fair alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but moderate alignment, the 3-cluster model showing a balanced performance, and the 4-cluster model showing good alignment but least defined clusters.

Figure 48, is the result of Agglomerative for first file using fourth technique:

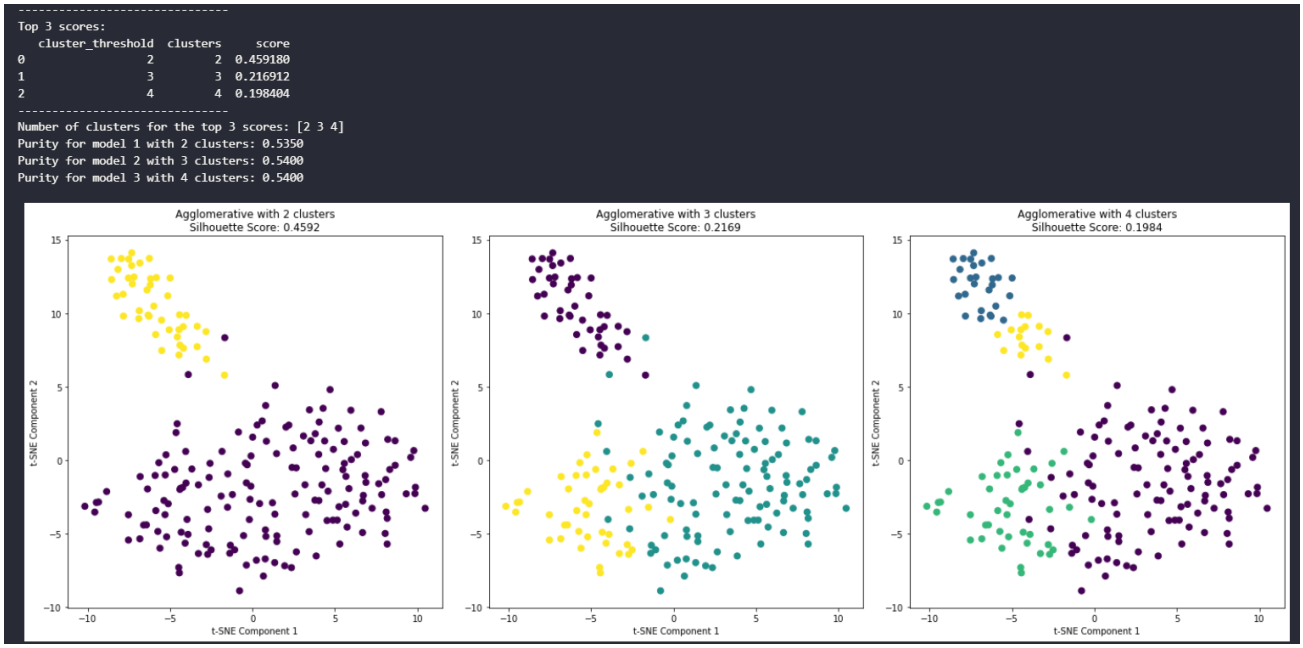

Figure 48: Agglomerative Result For First File Using Third Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative clustering model with 2 clusters has the highest silhouette score but moderate purity, indicating that while the clusters are well-defined, they do not match the true labels perfectly. The agglomerative clustering model with 3 clusters shows a lower silhouette score and slightly better purity, suggesting less well-defined clusters and better alignment with true labels. The agglomerative clustering model with 4 clusters, despite having the lowest silhouette score, shows fair alignment with true labels but suffers from poorly defined clusters.

In conclusion, the agglomerative clustering model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative clustering model with 2 clusters, while having well-defined clusters, does not align as well with the true labels, and the agglomerative clustering model with 4 clusters, despite having fair alignment, suffers from poorly defined clusters.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but moderate alignment, the 3-cluster model showing balanced performance, and the 4-cluster model showing fair alignment but least defined clusters.

Figure 49, is the result of Kmeans for second file using fourth technique:

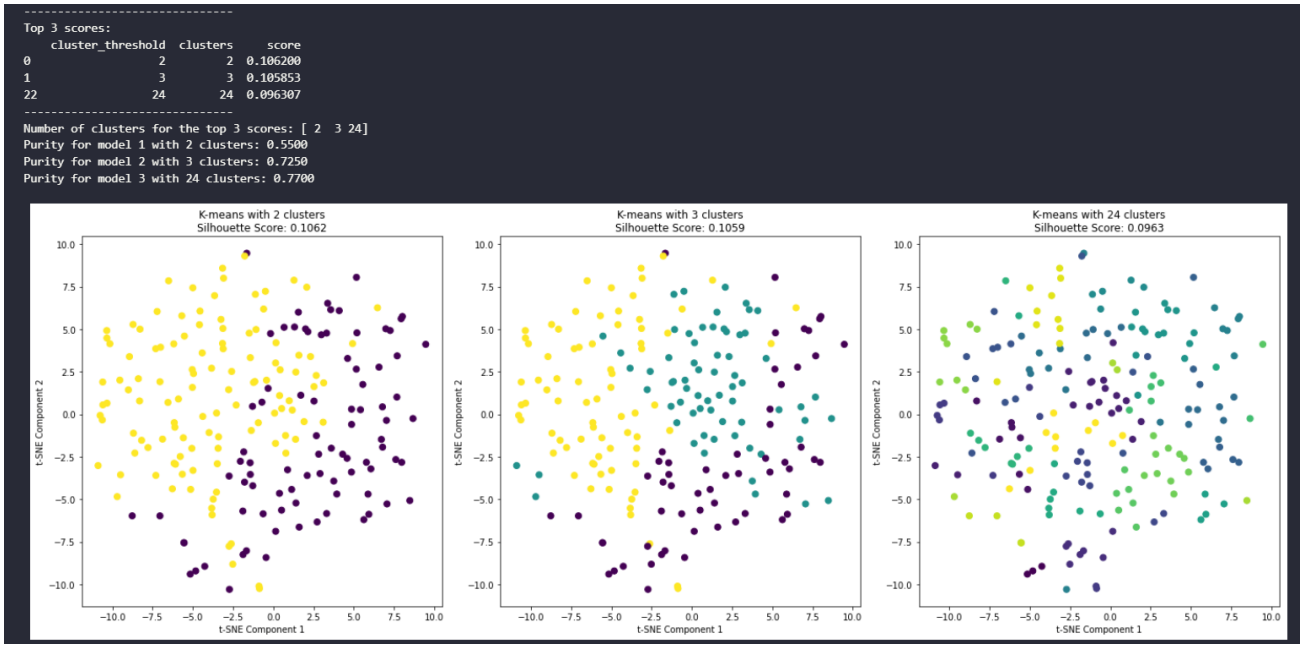

Figure 49: Kmeans Result For Second File Using Fourth Technique

Interpretation:

The three plots show different levels of clustering performance. The K-means model with 2 clusters has the highest silhouette score, indicating well-defined clusters, but its moderate purity suggests fair alignment with true labels. The K-means model with 3 clusters shows a balanced performance with a slightly lower silhouette score but higher purity, suggesting better alignment with true labels. The K-means model with 24 clusters, despite having the lowest silhouette score, has the highest purity, indicating the best alignment with true labels but suffers from over-segmentation.

In conclusion, the K-means model with 3 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The K-means model with 2 clusters shows well-defined clusters but fair alignment with true labels, while the K-means model with 24 clusters, despite having the best alignment, suffers from poorly defined clusters and over-segmentation.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 2-cluster model showing the best-defined clusters but fair alignment, the 3-cluster model showing balanced performance, and the 24-cluster model showing the best alignment but least defined clusters and over-segmentation.

Figure 50, is the result of Agglomerative for second file using fourth technique:

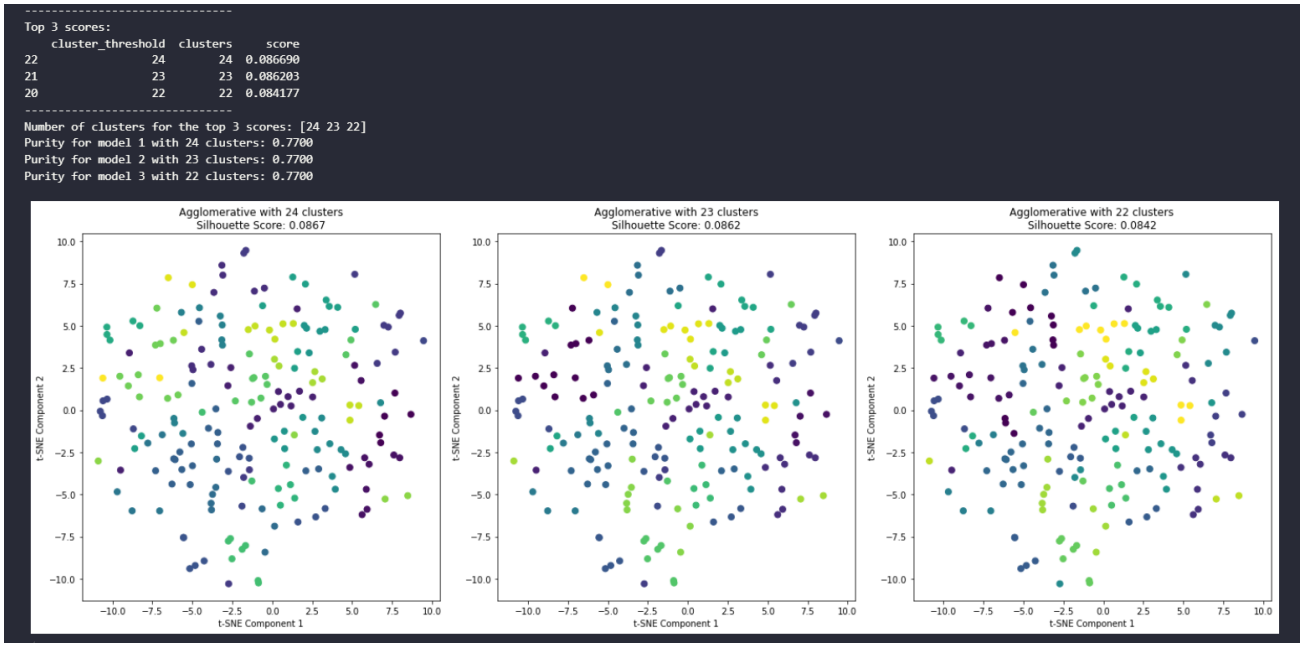

Figure 50: Agglomerative Result For Second File Using Fourth Technique

Interpretation:

The three plots show different levels of clustering performance. The agglomerative model with 24 clusters has the highest silhouette score, indicating well-defined clusters, and its high purity suggests good alignment with true labels. The agglomerative model with 23 clusters shows a balanced performance with a slightly lower silhouette score but the same high purity. The agglomerative model with 22 clusters, despite having the lowest silhouette score, maintains the same high purity, indicating good alignment with true labels but less defined clusters.

In conclusion, the agglomerative model with 24 clusters seems to provide the best overall clustering solution, balancing cluster definition and alignment with true labels. The agglomerative models with 23 and 22 clusters show slightly less well-defined clusters but maintain good alignment with true labels.

The given three plots differ in terms of cluster definition and alignment with true labels, with the 24-cluster model showing the best-defined clusters, the 23-cluster model showing slightly less defined clusters, and the 22-cluster model showing the least defined clusters but all maintaining good alignment with true labels.

3.6 Summary

Based on obtained purity and silhouette score of models, personally think first combination is best(Only with CSP alone) since I have considered a trade off between purity and how clusters are well sperated.

References

[1] Reza Saadatyar. *EEG-Lab*. GitHub repository. 2024. URL: https://github.com/RezaSaadatyar/EEG-Lab.