Hide code

In [1]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore', category=Warning)
```

In [2]:

```
s00 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s00.csv',header=Non
e)
s01 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s01.csv',header=Non
e)
s02 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s02.csv',header=Non
e)
s03 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s03.csv',header=Non
e)
s04 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s04.csv',header=Non
e)
s05 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s05.csv',header=Non
e)
s06 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s06.csv',header=Non
e)
s07 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s07.csv',header=Non
e)
s08 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s08.csv',header=Non
e)
s09 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s09.csv',header=Non
e)
s10 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s10.csv',header=Non
e)
s11 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s11.csv',header=Non
e)
s12 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s12.csv',header=Non
e)
s13 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s13.csv',header=Non
e)
s14 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s14.csv',header=Non
e)
s15 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s15.csv',header=Non
e)
s16 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s16.csv',header=Non
e)
s17 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s17.csv',header=Non
e)
s18 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s18.csv',header=Non
e)
s19 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s19.csv',header=Non
e)
```

```
s20 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s20.csv',header=Non
e)
s21 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s21.csv',header=Non
e)
s22 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s22.csv',header=Non
e)
s23 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s23.csv',header=Non
e)
s24 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s24.csv',header=Non
e)
s25 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s25.csv',header=Non
e)
s26 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s26.csv',header=Non
e)
s27 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s27.csv',header=Non
e)
s28 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s28.csv',header=Non
e)
s29 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s29.csv',header=Non
e)
s30 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s30.csv',header=Non
e)
s31 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s31.csv',header=Non
e)
s32 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s32.csv',header=Non
e)
s33 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s33.csv',header=Non
e)
s34 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s34.csv',header=Non
e)
s35 = pd.read_csv('/kaggle/input/complete-eeg-dataset/s35.csv',header=Non
e)
```

In [3]:

```
s00.shape
```

Out[3]:

```
(31000, 19)
```

In [4]:

```
s00=s00.transpose().to_numpy()
s01=s01.transpose().to_numpy()
s02=s02.transpose().to_numpy()
s03=s03.transpose().to_numpy()
s04=s04.transpose().to_numpy()
s05=s05.transpose().to_numpy()
s06=s06.transpose().to_numpy()
s07=s07.transpose().to_numpy()
s08=s08.transpose().to_numpy()
s09=s09.transpose().to_numpy()
s10=s10.transpose().to_numpy()
s11=s11.transpose().to_numpy()
s12=s12.transpose().to_numpy()
s13=s13.transpose().to_numpy()
s14=s14.transpose().to_numpy()
s15=s15.transpose().to_numpy()
s16=s16.transpose().to_numpy()
s17=s17.transpose().to_numpy()
s18=s18.transpose().to_numpy()
s19=s19.transpose().to_numpy()
s20=s20.transpose().to_numpy()
s21=s21.transpose().to_numpy()
s22=s22.transpose().to_numpy()
s23=s23.transpose().to_numpy()
s24=s24.transpose().to_numpy()
s25=s25.transpose().to_numpy()
s26=s26.transpose().to_numpy()
s27=s27.transpose().to_numpy()
s28=s28.transpose().to_numpy()
s29=s29.transpose().to_numpy()
s30=s30.transpose().to_numpy()
s31=s31.transpose().to_numpy()
s32=s32.transpose().to_numpy()
s33=s33.transpose().to_numpy()
s34=s34.transpose().to_numpy()
s35=s35.transpose().to_numpy()
```

Original dataset with 36 samples

In [5]:
```
dataset= np.array([[s00],[s01],[s02],[s03],[s04],[s05],[s06],[s07],
                   [s08],[s09],[s10],[s11],[s12],[s13],[s14],[s15],[s16],
[s17],[s18],[s19],
                   [s20],[s21],[s22],[s23],[s24],[s25],[s26],[s27],
                   [s28],[s29],[s30],[s31],[s32],[s33],[s34],[s35]])
```

In [6]:
```
dataset.shape
```

Out[6]:
```
(36, 1, 19, 31000)
```

Setting global random seed for model stability

In [7]:
```
seed = 42
tf.random.set_seed(seed)
```

In [10]:
```
y=np.array([0,1,1,1,0,1,0,1,1,0,0,1,1,1,0,1,1,1,1,0,1,0,0,1,1,1,1,1,1,1,
0,1,1,1,1,1])
```

In [12]:
```
dataset = dataset.reshape(36,1,760,775)
```

In [15]:
```
#Implementation of algorthims
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

In [21]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
logistic_preds = logistic_model.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, logistic_pr
eds))
print("Logistic Regression Classification Report:")
print(classification_report(y_test, logistic_preds))
```

```
Logistic Regression Accuracy: 0.375
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.50      0.29         2
           1       0.67      0.33      0.44         6

    accuracy                           0.38         8
   macro avg       0.43      0.42      0.37         8
weighted avg       0.55      0.38      0.40         8
```

In [22]:

```
# Decision Tree
decision_tree_model = DecisionTreeClassifier(random_state=50)
decision_tree_model.fit(X_train, y_train)
decision_tree_preds = decision_tree_model.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, decision_tree_pre
ds))
print("Decision Tree Classification Report:")
print(classification_report(y_test, decision_tree_preds))
```

```
Decision Tree Accuracy: 0.625
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.71      0.83      0.77         6

    accuracy                           0.62         8
   macro avg       0.36      0.42      0.38         8
weighted avg       0.54      0.62      0.58         8
```

In [20]:

```
from sklearn.ensemble import GradientBoostingClassifier

# Gradient Boosting
gradient_boost_model = GradientBoostingClassifier(random_state=42)
gradient_boost_model.fit(X_train, y_train)
gradient_boost_preds = gradient_boost_model.predict(X_test)
print("Gradient Boosting Accuracy:", accuracy_score(y_test, gradient_boos
t_preds))
print("Gradient Boosting Classification Report:")
print(classification_report(y_test, gradient_boost_preds))
```

```
Gradient Boosting Accuracy: 0.5
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.67      0.67      0.67         6

    accuracy                           0.50         8
   macro avg       0.33      0.33      0.33         8
weighted avg       0.50      0.50      0.50         8
```

In [25]:
```
# Random Forest
random_forest_model = RandomForestClassifier(random_state=40)
random_forest_model.fit(X_train, y_train)
random_forest_preds = random_forest_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, random_forest_pre
ds))
print("Random Forest Classification Report:")
print(classification_report(y_test, random_forest_preds))
```

```
Random Forest Accuracy: 0.75
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.75      1.00      0.86         6

    accuracy                           0.75         8
   macro avg       0.38      0.50      0.43         8
weighted avg       0.56      0.75      0.64         8
```

In [27]:
```
pip install joblib
```

```
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site
-packages (1.0.1)
WARNING: Running pip as the 'root' user can result in broken permissio
ns and conflicting behaviour with the system package manager. It is re
commended to use a virtual environment instead: https://pip.pypa.io/wa
rnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [28]:

```
import time
from joblib import Parallel, delayed

# Function to train and evaluate a model
def train_and_evaluate(model, X_train, X_test, y_train, y_test):
    start_time = time.time()

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    preds = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, preds)
    report = classification_report(y_test, preds)

    elapsed_time = time.time() - start_time
    return accuracy, report, elapsed_time

# Serial Execution
print("Serial Execution:")
for model, model_name in zip([logistic_model, decision_tree_model, gradie
nt_boost_model, random_forest_model],
                             ["Logistic Regression", "Decision Tree", "Gr
adient Boosting", "Random Forest"]):
    acc, rep, elapsed_time = train_and_evaluate(model, X_train, X_test, y
_train, y_test)
    print(f"{model_name} - Accuracy: {acc:.4f}, Elapsed Time: {elapsed_ti
me:.2f} seconds")
    print(f"Classification Report:\n{rep}\n")

# Parallel Execution
print("Parallel Execution:")
models = [logistic_model, decision_tree_model, gradient_boost_model, rand
om_forest_model]
names = ["Logistic Regression", "Decision Tree", "Gradient Boosting", "Ra
ndom Forest"]

results_parallel = Parallel(n_jobs=-1)(
```

```
    delayed(train_and_evaluate)(model, X_train, X_test, y_train, y_test)
for model in models
)

for (acc, rep, elapsed_time), model_name in zip(results_parallel, names):
    print(f"{model_name} - Accuracy: {acc:.4f}, Elapsed Time: {elapsed_ti
me:.2f} seconds")
    print(f"Classification Report:\n{rep}\n")
```

```
Serial Execution:
Logistic Regression - Accuracy: 0.3750, Elapsed Time: 3.93 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.50      0.29         2
           1       0.67      0.33      0.44         6

    accuracy                           0.38         8
   macro avg       0.43      0.42      0.37         8
weighted avg       0.55      0.38      0.40         8


Decision Tree - Accuracy: 0.6250, Elapsed Time: 1.66 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.71      0.83      0.77         6

    accuracy                           0.62         8
   macro avg       0.36      0.42      0.38         8
weighted avg       0.54      0.62      0.58         8


Gradient Boosting - Accuracy: 0.5000, Elapsed Time: 197.91 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.67      0.67      0.67         6

    accuracy                           0.50         8
   macro avg       0.33      0.33      0.33         8
weighted avg       0.50      0.50      0.50         8


Random Forest - Accuracy: 0.7500, Elapsed Time: 0.41 seconds
Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.00      0.00      0.00         2
              1       0.75      1.00      0.86         6

       accuracy                           0.75         8
      macro avg       0.38      0.50      0.43         8
   weighted avg       0.56      0.75      0.64         8


Parallel Execution:
Logistic Regression - Accuracy: 0.3750, Elapsed Time: 8.15 seconds
Classification Report:
              precision    recall  f1-score   support

              0       0.20      0.50      0.29         2
              1       0.67      0.33      0.44         6

       accuracy                           0.38         8
      macro avg       0.43      0.42      0.37         8
   weighted avg       0.55      0.38      0.40         8


Decision Tree - Accuracy: 0.6250, Elapsed Time: 2.63 seconds
Classification Report:
              precision    recall  f1-score   support

              0       0.00      0.00      0.00         2
              1       0.71      0.83      0.77         6

       accuracy                           0.62         8
      macro avg       0.36      0.42      0.38         8
   weighted avg       0.54      0.62      0.58         8


Gradient Boosting - Accuracy: 0.5000, Elapsed Time: 204.35 seconds
Classification Report:
              precision    recall  f1-score   support

              0       0.00      0.00      0.00         2
              1       0.67      0.67      0.67         6
```

```
    accuracy                           0.50         8
   macro avg       0.33      0.33      0.33         8
weighted avg       0.50      0.50      0.50         8


Random Forest - Accuracy: 0.7500, Elapsed Time: 0.90 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.75      1.00      0.86         6

    accuracy                           0.75         8
   macro avg       0.38      0.50      0.43         8
weighted avg       0.56      0.75      0.64         8
```

In [29]:

```
import time
from joblib import Parallel, delayed

# Function to train and evaluate a model
def train_and_evaluate(model, X_train, X_test, y_train, y_test):
    start_time = time.time()

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    preds = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, preds)
    report = classification_report(y_test, preds)

    elapsed_time = time.time() - start_time
    print(f"{model.__class__.__name__} - Accuracy: {accuracy:.4f}, Elapse
d Time: {elapsed_time:.2f} seconds")
    print(f"Classification Report:\n{report}\n")

    return accuracy, elapsed_time

# Serial Execution
serial_start_time = time.time()
print("Serial Execution:")
serial_elapsed_times = []
for model in [logistic_model, decision_tree_model, gradient_boost_model,
random_forest_model]:
    acc, elapsed_time = train_and_evaluate(model, X_train, X_test, y_trai
n, y_test)
    serial_elapsed_times.append(elapsed_time)

serial_total_time = time.time() - serial_start_time
print(f"Total Time (Serial): {serial_total_time:.2f} seconds\n")

# Parallel Execution
parallel_start_time = time.time()
print("Parallel Execution:")
```

```
models = [logistic_model, decision_tree_model, gradient_boost_model, rand
om_forest_model]
names = ["Logistic Regression", "Decision Tree", "Gradient Boosting", "Ra
ndom Forest"]

results_parallel = Parallel(n_jobs=-1)(
    delayed(train_and_evaluate)(model, X_train, X_test, y_train, y_test)
for model in models
)

parallel_total_time = time.time() - parallel_start_time
print(f"Total Time (Parallel): {parallel_total_time:.2f} seconds\n")

# Print elapsed times for each model in parallel
for (acc, elapsed_time), model_name in zip(results_parallel, names):
    print(f"{model_name} - Accuracy: {acc:.4f}, Elapsed Time: {elapsed_ti
me:.2f} seconds\n")
```

```
Serial Execution:
LogisticRegression - Accuracy: 0.3750, Elapsed Time: 4.11 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.50      0.29         2
           1       0.67      0.33      0.44         6

    accuracy                           0.38         8
   macro avg       0.43      0.42      0.37         8
weighted avg       0.55      0.38      0.40         8


DecisionTreeClassifier - Accuracy: 0.6250, Elapsed Time: 1.73 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.71      0.83      0.77         6

    accuracy                           0.62         8
   macro avg       0.36      0.42      0.38         8
weighted avg       0.54      0.62      0.58         8



/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classificatio
n.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero
_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
GradientBoostingClassifier - Accuracy: 0.5000, Elapsed Time: 198.54 se
conds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.67      0.67      0.67         6

    accuracy                           0.50         8
   macro avg       0.33      0.33      0.33         8
weighted avg       0.50      0.50      0.50         8


RandomForestClassifier - Accuracy: 0.7500, Elapsed Time: 0.40 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.75      1.00      0.86         6

    accuracy                           0.75         8
   macro avg       0.38      0.50      0.43         8
weighted avg       0.56      0.75      0.64         8


Total Time (Serial): 204.79 seconds

Parallel Execution:
Total Time (Parallel): 201.39 seconds

Logistic Regression - Accuracy: 0.3750, Elapsed Time: 8.86 seconds

Decision Tree - Accuracy: 0.6250, Elapsed Time: 2.73 seconds

Gradient Boosting - Accuracy: 0.5000, Elapsed Time: 198.72 seconds

Random Forest - Accuracy: 0.7500, Elapsed Time: 0.91 seconds

RandomForestClassifier - Accuracy: 0.7500, Elapsed Time: 0.91 seconds
Classification Report:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	0.75	1.00	0.86	6
accuracy			0.75	8
macro avg	0.38	0.50	0.43	8
weighted avg	0.56	0.75	0.64	8

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classificatio
n.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero
_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
DecisionTreeClassifier - Accuracy: 0.6250, Elapsed Time: 2.73 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.71      0.83      0.77         6

    accuracy                           0.62         8
   macro avg       0.36      0.42      0.38         8
weighted avg       0.54      0.62      0.58         8


LogisticRegression - Accuracy: 0.3750, Elapsed Time: 8.86 seconds
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.50      0.29         2
           1       0.67      0.33      0.44         6

    accuracy                           0.38         8
   macro avg       0.43      0.42      0.37         8
weighted avg       0.55      0.38      0.40         8


GradientBoostingClassifier - Accuracy: 0.5000, Elapsed Time: 198.72 se
conds
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.67      0.67      0.67         6

    accuracy                           0.50         8
   macro avg       0.33      0.33      0.33         8
weighted avg       0.50      0.50      0.50         8
```

In [ ]: