

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**



**BÀI TẬP LỚN
EEG - EYE STATE DETECTION**

Sinh viên thực hiện

Lê Văn Huỳnh

Nguyễn Thùy Linh

Vũ Phương Nhi

Môn: Lập trình DSP

Giảng viên: TS. Nguyễn Hồng Thịnh

HÀ NỘI - 2024

TÓM TẮT

Tóm tắt: Báo cáo tập trung vào việc nhận diện trạng thái của mắt dựa trên dữ liệu tín hiệu EEG (điện não đồ) thu thập từ nhiều vị trí khác nhau trên não. Để đảm bảo độ chính xác trong dự đoán, dữ liệu thu được phải trải qua quá trình tiền xử lý nhằm loại bỏ nhiễu và các tín hiệu không mong muốn, chẳng hạn như do cử động cơ thể. Sau khi hoàn thiện quá trình tiền xử lý, nhóm đã phát triển và so sánh hiệu quả của ba mô hình dựa trên các thuật toán: K-Nearest Neighbors (KNN), Support Vector Machine (SVM) và Random Forest. Kết quả cho thấy mô hình sử dụng thuật toán Support Vector Machine (SVM) vượt trội hơn so với hai mô hình còn lại về độ chính xác và hiệu suất dự đoán. Do đó, mô hình SVM được lựa chọn để tích hợp vào ứng dụng EEG Detection nhằm phát hiện trạng thái mắt một cách hiệu quả.

Từ khóa: *Eye state detection, K-Nearest Neighbors(KNN), Support Vector Machine (SVM), Random Forest*

Mục lục

TÓM TẮT	i
1 Tiền xử lý tín hiệu EEG	1
1.1 Chuẩn bị dữ liệu	1
1.2 Chuẩn hóa dữ liệu	3
1.3 Nội suy giá trị Cubic Spline	4
1.4 Lọc nhiễu bằng Independent Component Analysis - ICA	6
1.5 Trích xuất sóng Alpha	8
2 Xây dựng và đánh giá các mô hình học máy	10
2.1 Phân chia dữ liệu	10
2.2 Trích xuất các đặc trưng bằng Common Spatial Patterns- CSP . . .	11
2.3 Mô hình K-Nearest Neighbors(KNN)	11
2.4 Mô hình Support Vector Machine (SVM)	14
2.5 Mô hình Random Forest	17
3 Xây dựng ứng dụng	20
3.1 Giao diện	20
3.2 Chức năng các hàm	23
3.3 Kết quả	24
3.3.1 Dự đoán một trạng thái bất kì	24
3.3.2 Dự đoán hàng loạt các dữ liệu từ file excel	28

Danh sách hình vẽ

1.1	Mô tả các vị trí đặt điện cực	1
1.2	Thực hiện đọc dữ liệu	2
1.3	Dữ liệu được đọc từ file	2
1.4	Lấy mẫu dữ liệu	2
1.5	Kết quả sau lấy	3
1.6	Chuẩn hóa dữ liệu	3
1.7	Tín hiệu sau chuẩn hóa	4
1.8	Tín hiệu sau chuẩn hóa	4
1.9	Nội suy Cubic Spline	5
1.10	Tín hiệu sau nội suy	5
1.11	Tín hiệu sau nội suy	5
1.12	Lọc nhiễu bằng ICA	6
1.13	Các thành phần tín hiệu độc lập	7
1.14	Tín hiệu sau khi lọc bằng ICA	7
1.15	Tín hiệu sau khi lọc bằng ICA	8
1.16	Lọc tín hiệu bằng bộ lọc Butterworth	8
1.17	Tín hiệu sau khi lọc	9
1.18	Tín hiệu sau khi lọc	9
2.1	Phân chia dữ liệu và chuẩn hóa	10
2.2	Trích xuất các đặc trưng bằng Common Spatial Patterns	11
2.3	Huấn luyện mô hình sử dụng KNN	12

2.4	Kết quả của mô hình sử dụng KNN	13
2.5	Thực thi ma trận nhầm lẫn của thuật toán KNN	13
2.6	Ma trận nhầm lẫn của thuật toán KNN	14
2.7	Huấn luyện mô hình sử dụng Support Vector Machine	15
2.8	Kết quả mô hình sử dụng Support Vector Machine	15
2.9	Thực thi ma trận nhầm lẫn của thuật toán SVM	16
2.10	Ma trận nhầm lẫn của thuật toán SVM	16
2.11	Huấn luyện mô hình sử dụng Random Forest	17
2.12	Kết quả mô hình sử dụng Random Forest	18
2.13	Ma trận nhầm lẫn của thuật toán Random Forest	18
2.14	Ma trận nhầm lẫn của thuật toán Random Forest	19
3.1	Tích hợp mô hình SVM	20
3.2	Tạo các ô nhập dữ liệu	21
3.3	Tạo ô hiển thị kết quả	21
3.4	Tạo nút thực hiện dự đoán	21
3.5	Tạo nút xóa các ô đã nhập	21
3.6	Tạo nút tải file dữ liệu cần dự đoán	22
3.7	Tổng quan ứng dụng	22
3.8	Hàm <code>get_input_data()</code>	23
3.9	Hàm <code>predict()</code>	23
3.10	Hàm <code>load_and_predict_from_excel()</code>	24
3.11	Thực hiện nhập dữ liệu	25
3.12	Kết quả dự đoán	26
3.13	Thực hiện nhập dữ liệu	27
3.14	Kết quả dự đoán	28
3.15	File dữ liệu excel cần dự đoán	29
3.16	Dữ liệu trong file excel cần dự đoán	29

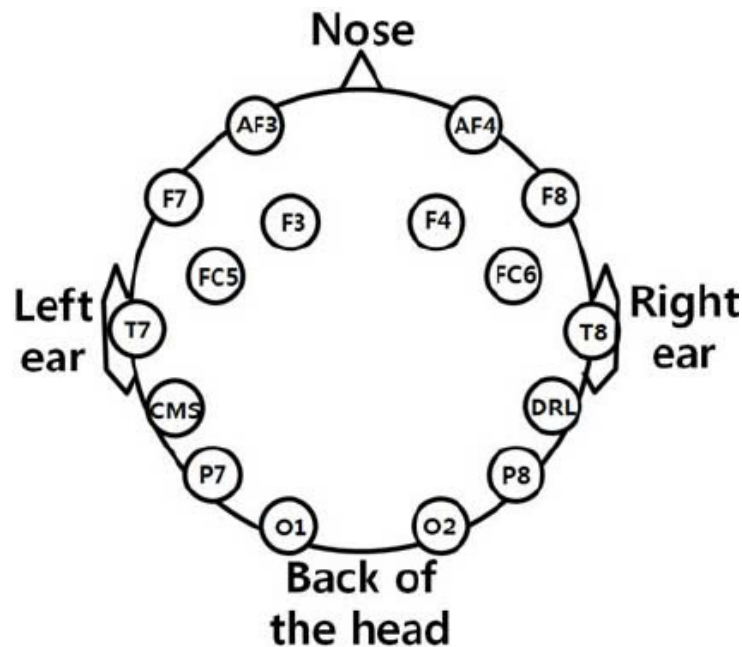
3.17 Chọn file lưu kết quả	30
3.18 Kết quả dự đoán	30

Tiền xử lý tín hiệu EEG

Chương 1 trình bày tổng quan về các bước thực hiện tiền xử lý tín hiệu. Các kỹ thuật chuẩn hóa, nội suy tín hiệu, lọc nhiễu bằng Independent Component Analysis được sử dụng nhằm loại bỏ các sai lệch, nhiễu, các thành phần không mong muốn khác.

1.1 Chuẩn bị dữ liệu

Thực hiện đọc dữ liệu từ file .arff và hiển thị dữ liệu dưới dạng các hàng và cột



Hình 1.1: Mô tả các vị trí đặt điện cực

```

from google.colab import files
import pandas as pd
import arff

uploaded = files.upload()

file_path = list(uploaded.keys())[0]

with open(file_path, 'r') as f:
    dataset = arff.load(f)

data = pd.DataFrame(dataset['data'], columns=[attr[0] for attr in dataset['attributes']])

# Bước 5: Hiển thị thông tin dataset
print(data.head())

```

Hình 1.2: Thực hiện đọc dữ liệu

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8	FC6	F4	F8	AF4	eyeDetection
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46	4211.28	4280.51	4635.90	4393.85	0
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67	4207.69	4279.49	4632.82	4384.10	0
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05	4206.67	4282.05	4628.72	4389.23	0
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38	4210.77	4287.69	4632.31	4396.41	0
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10	4212.82	4288.21	4632.82	4398.46	0
...
14975	4281.03	3990.26	4245.64	4116.92	4333.85	4614.36	4074.87	4625.64	4203.08	4221.54	4171.28	4269.23	4593.33	4340.51	1
14976	4276.92	3991.79	4245.13	4110.77	4332.82	4615.38	4073.33	4621.54	4194.36	4217.44	4162.56	4259.49	4590.26	4333.33	1
14977	4277.44	3990.77	4246.67	4113.85	4333.33	4615.38	4072.82	4623.59	4193.33	4212.82	4160.51	4257.95	4591.79	4339.49	1
14978	4284.62	3991.79	4251.28	4122.05	4334.36	4616.41	4080.51	4628.72	4200.00	4220.00	4165.64	4267.18	4596.41	4350.77	1
14979	4287.69	3997.44	4260.00	4121.03	4333.33	4616.41	4088.72	4638.46	4212.31	4226.67	4167.69	4274.36	4597.95	4350.77	1

14980 rows x 15 columns

Hình 1.3: Dữ liệu được đọc từ file

Dữ liệu gồm 14 điểm được lấy mẫu theo thời gian biểu thị bằng các cột và các hàng (14980 hàng \times 15 cột). Cột eyeDetection biểu thị trạng thái mở mắt(1) và nhắm mắt(0) tại các thời điểm được lấy mẫu. Tiếp theo, thực hiện lấy mẫu dữ liệu tại tần số 128 và kiểm tra dữ liệu Null

```

# define sampling rate, time vector, and electrode list (columns list)
Fs = 128 # (number of samples / 117s length of data mentioned on the data description) rounded to the closest integer.
t = np.arange(0, len(df) * 1 / Fs, 1/Fs)
cols = df.columns.tolist()[:-1]

print( 'Number of null samples:\n' + str(df.isnull().sum()) )
df.head()

```

Hình 1.4: Lấy mẫu dữ liệu

Number of null samples:

AF3	0
F7	0
F3	0
FC5	0
T7	0
P7	0
O1	0
O2	0
P8	0
T8	0
FC6	0
F4	0
F8	0
AF4	0
eyeDetection	0

dtype: int64

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8	FC6	F4	F8	AF4	eyeDetection
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46	4211.28	4280.51	4635.90	4393.85	0
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67	4207.69	4279.49	4632.82	4384.10	0
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05	4206.67	4282.05	4628.72	4389.23	0
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38	4210.77	4287.69	4632.31	4396.41	0
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10	4212.82	4288.21	4632.82	4398.46	0

Hình 1.5: Kết quả sau lấy

1.2 Chuẩn hóa dữ liệu

Thực hiện chuẩn hóa dữ liệu nhằm loại bỏ các sai lệch trong biên độ tín hiệu. Bên cạnh đó, xác định và loại bỏ các giá trị ngoại lai vượt quá ngưỡng chấp nhận (Nếu giá trị $|z| > 4$, giá trị đó được coi là ngoại lai và được thay bằng NaN). Mục đích nhằm giảm thiểu tác động của các giá trị bất thường trong tín hiệu EEG và đảm bảo tín hiệu đồng nhất hơn.

```
# Find outliers and put Nan instead
X = X.apply(stats.zscore, axis=0)
X = X.applymap(lambda x: np.nan if (abs(x) > 4) else x )

# recalculate outliers with ignoring nans since the first calculation was biased with the huge outliers!
X = X.apply(stats.zscore, nan_policy='omit', axis=0)
X = X.applymap(lambda x: np.nan if (abs(x) > 4) else x )

plot_data([X])
```

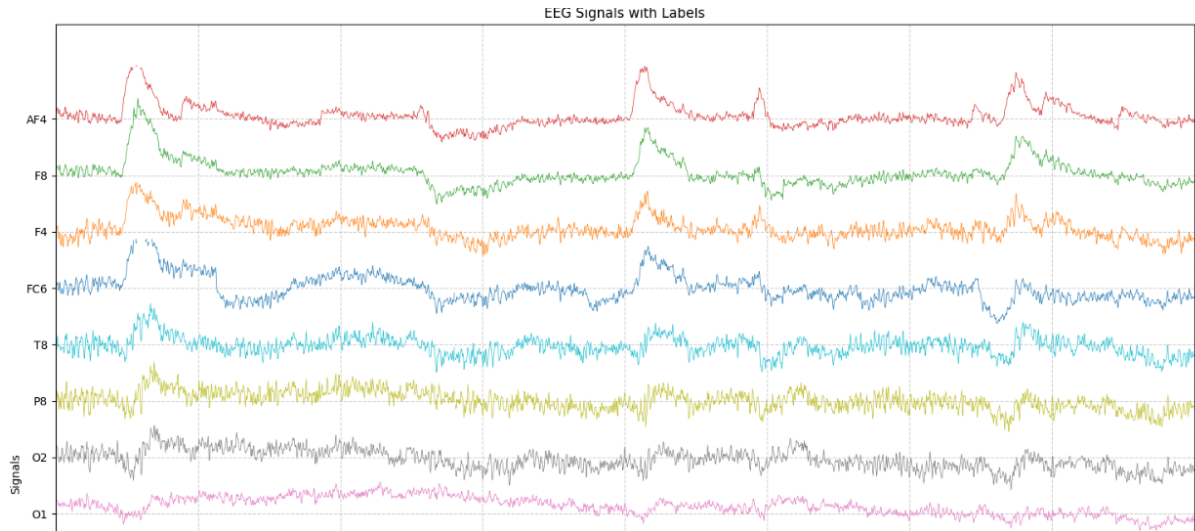
Hình 1.6: Chuẩn hóa dữ liệu

Sử dụng stats.zscore để tính z-score cho từng giá trị của tín hiệu.

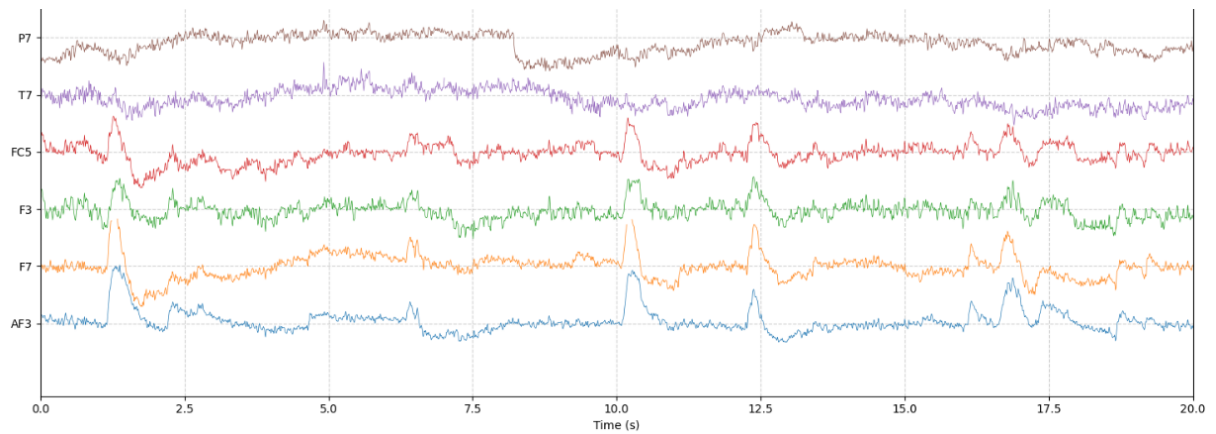
$$z = \frac{x - \mu}{\sigma}$$

Trong đó:

- μ là giá trị trung bình ,
- σ là độ lệch chuẩn .



Hình 1.7: Tín hiệu sau chuẩn hóa



Hình 1.8: Tín hiệu sau chuẩn hóa

1.3 Nội suy giá trị Cubic Spline

Ở bước trên với những giá trị ngoại lai ta thực hiện loại bỏ và thay thế bằng NaN. Bây giờ ta thực hiện điền các giá trị bị thiếu bằng phương pháp nội suy cubic spline để giữ được tính liên tục của tín hiệu.

Dựa trên các điểm không bị thiếu trong tín hiệu, ta sử dụng nội suy cubic spline để xây dựng một hàm nội suy nhằm ước tính giá trị ở các vị trí bị thiếu.

```

from scipy import signal, interpolate

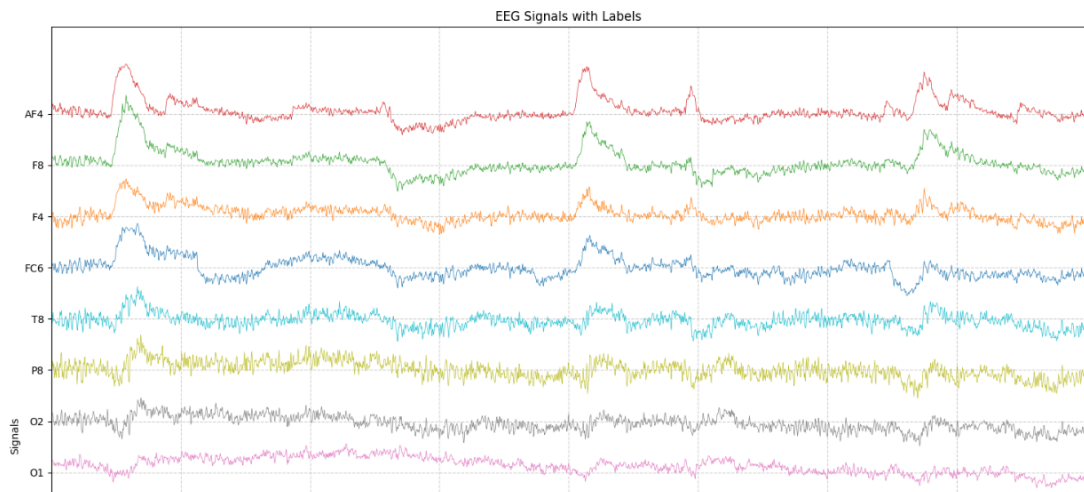
def interp(x):
    t_temp = t[ x.index[ ~x.isnull() ] ]
    x = x[ x.index[ ~x.isnull() ] ]
    clf = interpolate.interp1d(t_temp, x, kind='cubic')
    return clf(t)

# interpolate the nans using cubic spline method
X_interp = X.apply(interp, axis=0)

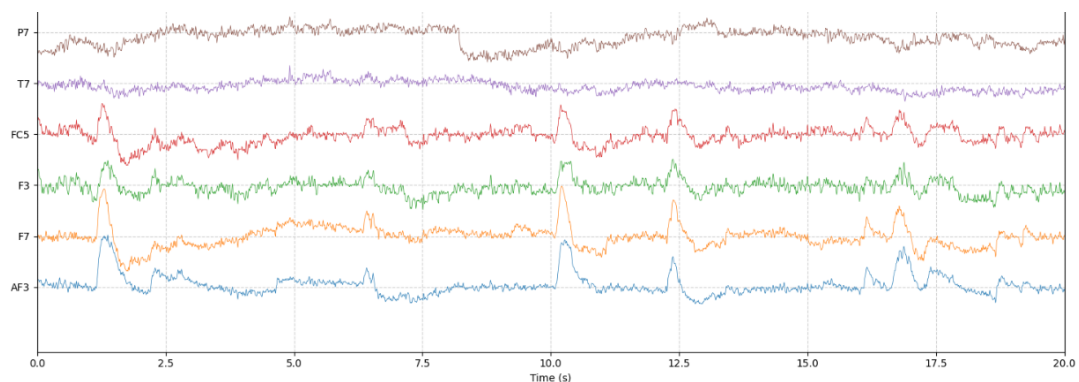
plot_data([X_interp])

```

Hình 1.9: Nội suy Cubic Spline



Hình 1.10: Tín hiệu sau nội suy



Hình 1.11: Tín hiệu sau nội suy

1.4 Lọc nhiễu bằng Independent Component Analysis - ICA

ICA là một kỹ thuật phân tích tín hiệu được sử dụng để tách các nguồn tín hiệu độc lập từ một hỗn hợp tín hiệu quan sát được nhằm loại bỏ nhiễu hoặc các thành phần không mong muốn.

```
# ICA
from sklearn.decomposition import FastICA

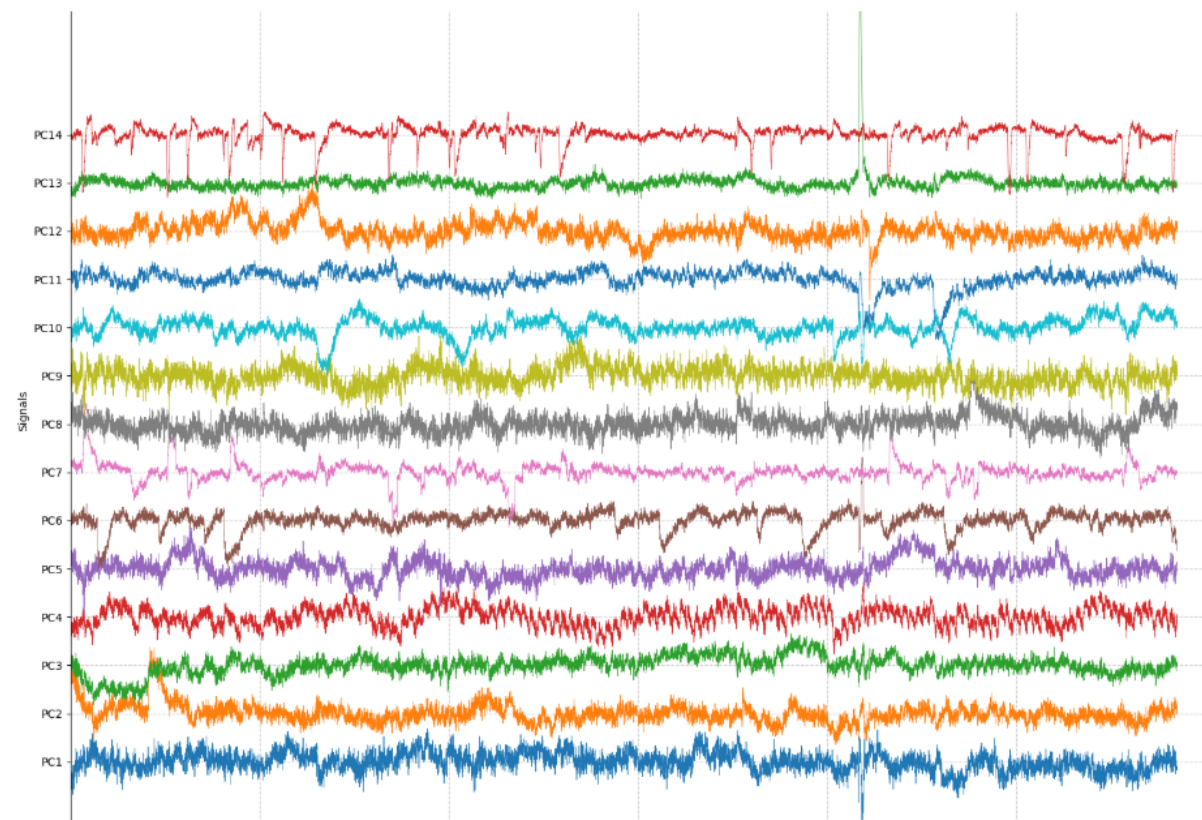
# apply ICA to drop non-electrophysiological components (requires familiarity with EEG data)
ica = FastICA(max_iter=2000, random_state=0)
X_pcs = pd.DataFrame( ica.fit_transform(X_interp) )
X_pcs.columns = ['PC' + str(ind+1) for ind in range(X_pcs.shape[-1])]

plot_data([X_pcs], xlim=[0, 120])
X_pcs = X_pcs.drop(columns=['PC1', 'PC7'])
# reconstruct clean EEG after dropping the bad components
ica.mixing_ = np.delete(ica.mixing_, [0, 6], axis = 1)
X_interp_clean = pd.DataFrame( ica.inverse_transform(X_pcs) )
X_interp_clean.columns = cols

plot_data([X_interp, X_interp_clean], xlim=[0, 20])
```

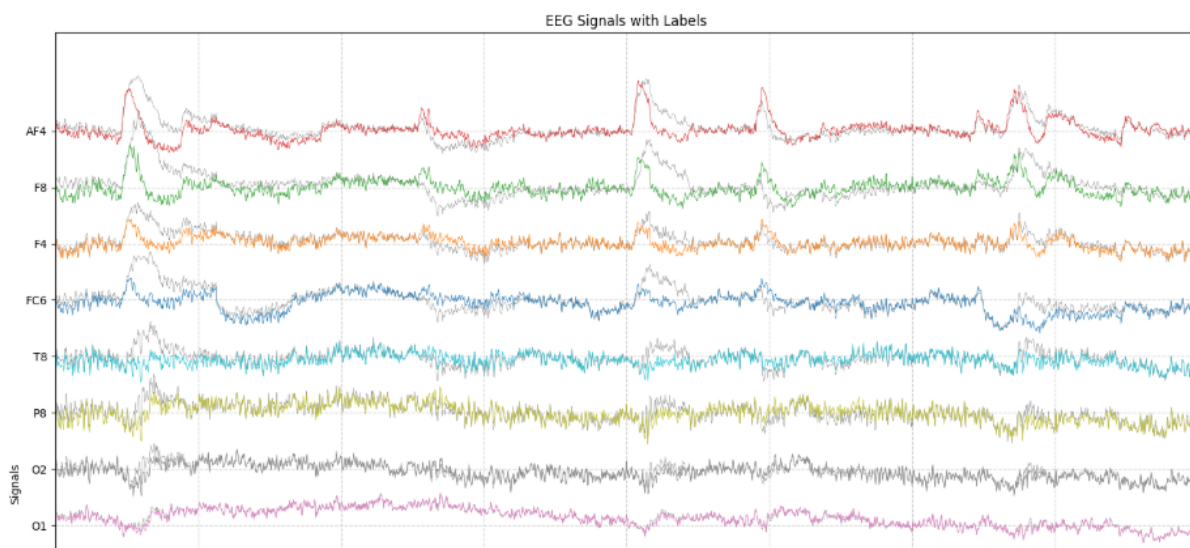
Hình 1.12: Lọc nhiễu bằng ICA

Áp dụng ICA trên dữ liệu `X_interp` (dữ liệu tín hiệu EEG đầu vào đã qua xử lý trước). Ta nhận được các thành phần độc lập (Independent Components - ICs) dưới dạng ma trận. Sau đó chuyển các thành phần độc lập sang dạng bảng dữ liệu để dễ xử lý. Dựa vào các thành phần độc lập này, ta có thể xác định cột 'PC1' và 'PC7', là các thành phần được coi là nhiễu và thực hiện loại bỏ tín hiệu ở 2 cột này.

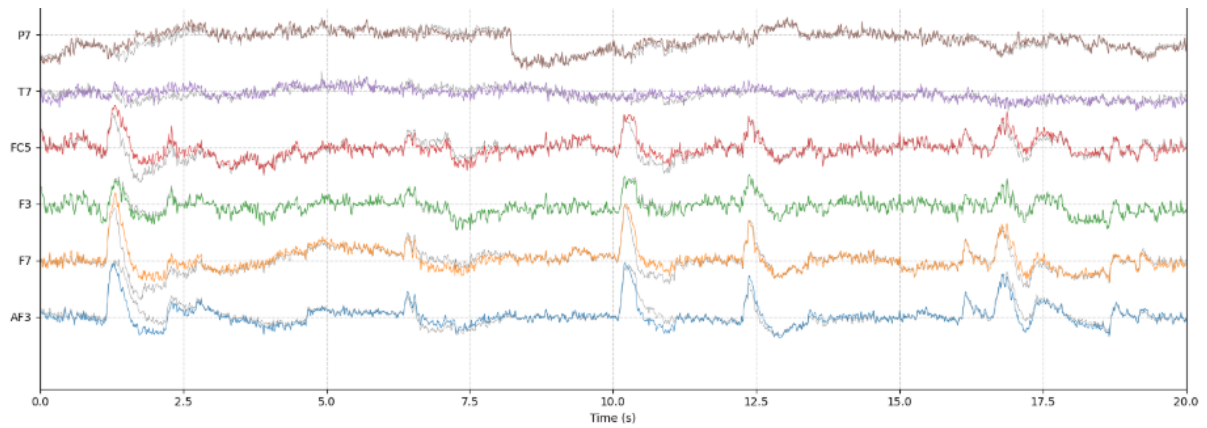


Hình 1.13: Các thành phần tín hiệu độc lập

Thực hiện tái tổ hợp tín hiệu sạch từ các thành phần độc lập còn lại.



Hình 1.14: Tín hiệu sau khi lọc bằng ICA



Hình 1.15: Tín hiệu sau khi lọc bằng ICA

1.5 Trích xuất sóng Alpha

Thực hiện lọc tín hiệu bằng bộ lọc Butterworth bậc 6 (`scipy.signal.butter`) để thực hiện lọc băng thông (bandpass filter) trong khoảng 8-12 Hz. Tín hiệu sau khi lọc được chuẩn hóa về biên độ ban đầu để dễ so sánh. Tín hiệu sau khi lọc chỉ còn chứa thông tin liên quan đến sóng Alpha, giúp tăng độ chính xác của phân tích.

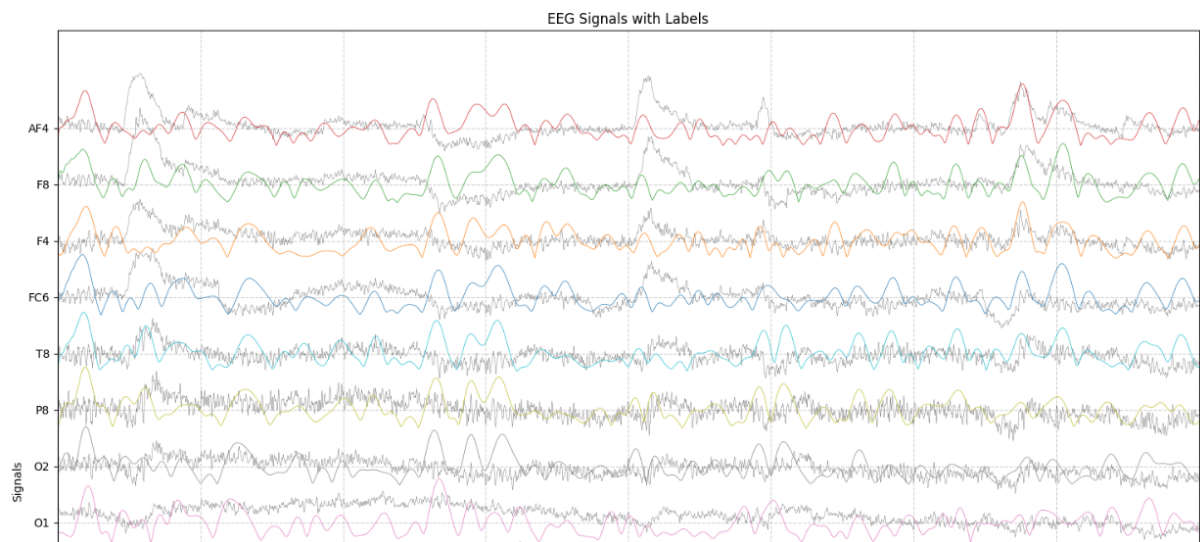
```
# now that data is clean, extract alpha waves magnitude from the clean signals

# filter the data between 8-12 Hz (note that data has been rescaled to original scale after filtering for comparable visualization)
b, a = signal.butter(6, [8 / Fs * 2, 12 / Fs * 2], btype='bandpass')
X_interp_clean_alpha = X_interp_clean.apply(lambda x: signal.filtfilt(b, a, x) / max(abs(signal.filtfilt(b, a, x))) * max(abs(x)), axis=0)

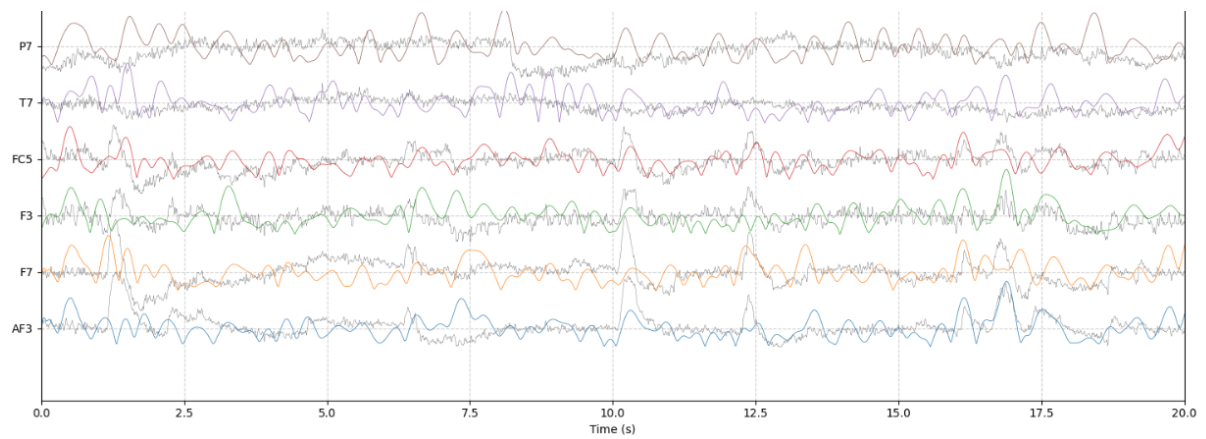
# extract envelope of the Alpha waves
X_interp_clean_alpha = X_interp_clean_alpha.apply(lambda x: np.abs(signal.hilbert(x)), axis=0)
X_interp_clean_alpha.columns = cols

plot_data([X_interp_clean, X_interp_clean_alpha], xlim=[0, 20])
```

Hình 1.16: Lọc tín hiệu bằng bộ lọc Butterworth



Hình 1.17: Tín hiệu sau khi lọc



Hình 1.18: Tín hiệu sau khi lọc

Xây dựng và đánh giá các mô hình học máy

Chương này tập trung vào việc xây dựng, huấn luyện, và đánh giá các mô hình học máy (K-Nearest Neighbors, Support Vector Machine, và Random Forest) để phân loại trạng thái mắt dựa trên tín hiệu EEG đã qua xử lý.

2.1 Phân chia dữ liệu

Thực hiện tách dữ liệu đã qua tiền xử lý thành 2 phần:

- **Tập huấn luyện:** Chiếm 80% dùng để huấn luyện mô hình.
- **Tập kiểm tra:** Chiếm 20% dùng để đánh giá hiệu suất của mô hình.

Sau đó, đưa dữ liệu về cùng thang đo nhằm đảm bảo các đặc trưng không bị ảnh hưởng bởi sự chênh lệch về giá trị. Ta sử dụng hàm `StandardScaler()` từ thư viện `sklearn.preprocessing` để chuẩn hóa dữ liệu.

```
# train an SVM to classify
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler

# split train test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=48, test_size=0.2, stratify=Y, shuffle=True)

# normalize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Hình 2.1: Phân chia dữ liệu và chuẩn hóa

2.2 Trích xuất các đặc trưng bằng Common Spatial Patterns-CSP

Common Spatial Patterns (CSP) có tác dụng chính trong việc trích xuất các đặc trưng không gian của tín hiệu, nhằm tối ưu hóa việc phân biệt các trạng thái hoặc lớp khác nhau.

```
# Define a function to compute CSP filters
def compute_csp(X, y, n_components=14):
    # Separate EEG trials for each class
    class_0 = np.transpose(X[y == 0])
    class_1 = np.transpose(X[y == 1])

    # Compute class-wise covariance matrices
    cov_0 = np.dot(class_0, np.transpose(class_0)) / np.trace(np.dot(class_0, np.transpose(class_0)))
    cov_1 = np.dot(class_1, np.transpose(class_1)) / np.trace(np.dot(class_1, np.transpose(class_1)))

    # Compute generalized eigenvalues and eigenvectors
    eigenvalues, eigenvectors = eigh(cov_0, cov_0 + cov_1)

    # Sort eigenvectors based on eigenvalues
    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvectors = eigenvectors[:, sorted_indices]

    # Select top CSP filters
    csp_filters = sorted_eigenvectors[:, :n_components]

    return csp_filters

# Define a function to apply CSP filtering to EEG signals
def apply_csp(X, csp_filters):
    return np.dot(csp_filters.T, X.T).T
    # Transpose X before applying filters and transpose the result back

# Compute CSP filters
csp_filters = compute_csp(X_train, y_train)
# Apply CSP filtering to training and testing data
X_train_csp = apply_csp(X_train, csp_filters)
X_test_csp = apply_csp(X_test, csp_filters)
```

Hình 2.2: Trích xuất các đặc trưng bằng Common Spatial Patterns

2.3 Mô hình K-Nearest Neighbors(KNN)

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ

liệu mới. K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression.

Một cách ngắn gọn, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất (K-lân cận), không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu.

Ở đây chúng ta sử dụng 5 điểm lân cận gần nhất với khoảng cách Minkowski được sử dụng (khi $p = 2$, nó tương đương với khoảng cách Euclidean). Sử dụng dữ liệu huấn luyện đã qua CSP (X_{train_csp}) và nhãn (y_{train}) để huấn luyện mô hình.

```
# Initialize KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
# Train the KNN classifier on the CSP-filtered training data
knn_classifier.fit(X_train_csp, y_train)

# Make predictions on the CSP-filtered training data
y_train_pred = knn_classifier.predict(X_train_csp )

# Make predictions on the CSP-filtered test data
y_test_pred = knn_classifier.predict(X_test_csp)

# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
# Calculate test accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)
# Print classification report for test data
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))
# Calculate and print confusion matrix for test data
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Hình 2.3: Huấn luyện mô hình sử dụng KNN

```

Training Accuracy: 0.9994993324432577
Test Accuracy: 0.9966622162883845
Test Classification Report:

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1651
1	1.00	0.99	1.00	1345
accuracy			1.00	2996
macro avg	1.00	1.00	1.00	2996
weighted avg	1.00	1.00	1.00	2996

Hình 2.4: Kết quả của mô hình sử dụng KNN

Từ kết quả thu được ta có thể thấy tỉ lệ mẫu được dự đoán đúng trên tập huấn luyện (Training Accuracy) đạt 99.9%. Tỉ lệ này phản ánh khả năng của mô hình trong việc ghi nhớ và học mẫu từ dữ liệu huấn luyện là rất tốt. Tỷ lệ mẫu được dự đoán đúng trên tập kiểm tra (Test Accuracy) đạt 99.6%. Đây cũng là 1 tỉ lệ cao phản ánh khả năng của mô hình trong việc tổng quát hóa và xử lý dữ liệu mới là gần như chính xác.

Thực hiện vẽ ma trận nhầm lẫn (confusion matrix) để trực quan hóa hiệu suất của mô hình phân loại. Ma trận này biểu diễn số lượng dự đoán đúng và sai của mô hình cho từng lớp.

```

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

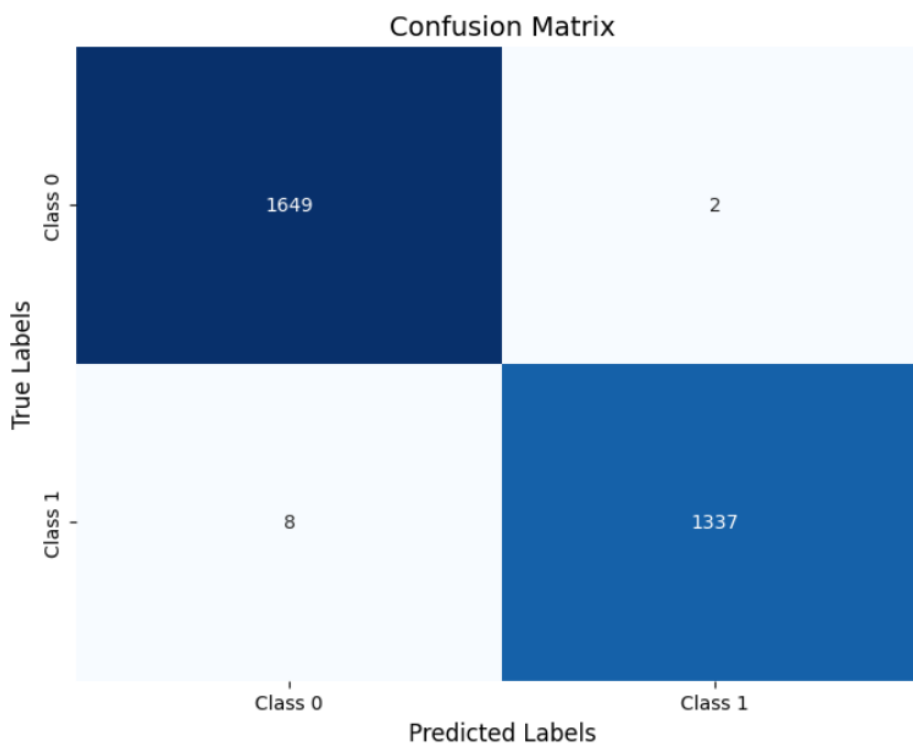
# Vẽ heatmap cho ma trận confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1']) # Đổi nhãn nếu cần

# Thêm nhãn và tiêu đề
plt.xlabel('Predicted Labels', fontsize=12)
plt.ylabel('True Labels', fontsize=12)
plt.title('Confusion Matrix', fontsize=14)
plt.show()

```

Hình 2.5: Thực thi ma trận nhầm lẫn của thuật toán KNN

```
Confusion Matrix:  
[[1649  2]  
 [  8 1337]]
```



Hình 2.6: Ma trận nhầm lẫn của thuật toán KNN

- **Số lượng dự đoán đúng :** Class 0 đạt 1649 dữ liệu. Class 1 đạt 1337 dữ liệu
- **Số lượng dự đoán sai:** Class 0 sai 2 dữ liệu. Class 1 sai 8 dữ liệu.

2.4 Mô hình Support Vector Machine (SVM)

Support Vector Machine là một thuật toán phân loại mạnh mẽ, dựa trên việc tìm siêu phẳng tối ưu để phân biệt các lớp. Nó được sử dụng rộng rãi cho cả phân loại tuyến tính và phi tuyến tính, cũng như các tác vụ phục hồi và phát hiện ngoại lệ.

Thực hiện huấn luyện mô hình Support Vector Machine (SVM) bằng cách tối ưu hóa các tham số thông qua Grid Search.

```

from sklearn.metrics import roc_auc_score

# train with grid search
svc = SVC()
parameters = {'gamma': [0.1, 1, 10], 'C': [0.1, 1, 10]}
clf = GridSearchCV(svc, parameters)
clf.fit(X_train_csp, y_train)

# predict labels
y_test_pred = clf.predict(X_test_csp)

# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)

# Calculate test accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)

# Print classification report for test data
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

# Calculate and print confusion matrix for test data
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)

```

Hình 2.7: Huấn luyện mô hình sử dụng Support Vector Machine

```

Training Accuracy: 1.0
Test Accuracy: 0.9976635514018691
Test Classification Report:

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1651
1	1.00	1.00	1.00	1345
accuracy			1.00	2996
macro avg	1.00	1.00	1.00	2996
weighted avg	1.00	1.00	1.00	2996

Hình 2.8: Kết quả mô hình sử dụng Support Vector Machine

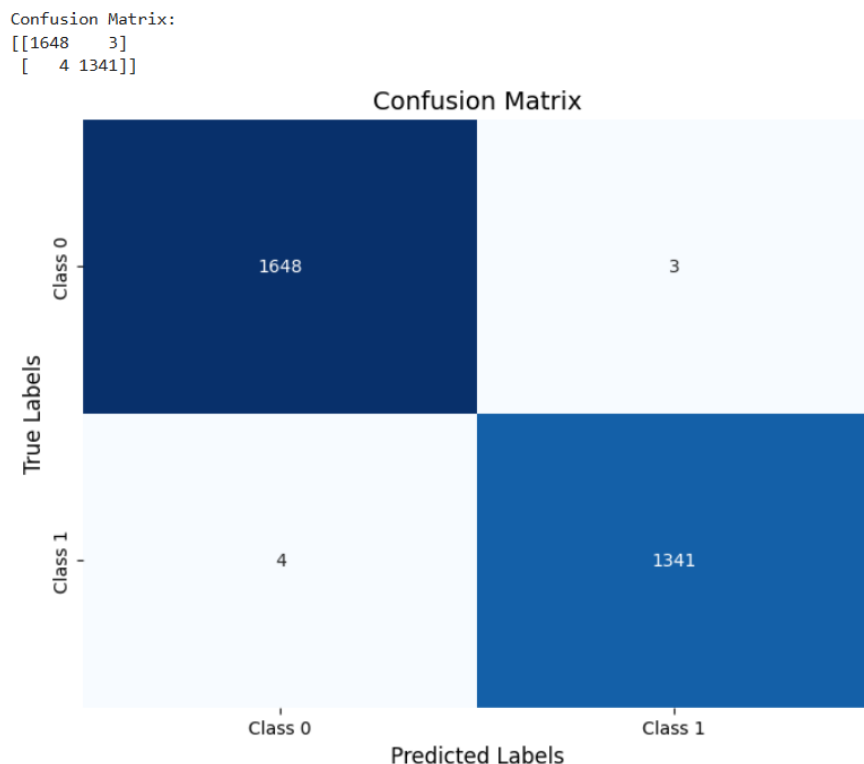
Tỉ lệ mẫu được dự đoán đúng trên tập huấn luyện (Training Accuracy) đạt 100%. Trong khi đó, tỷ lệ mẫu được dự đoán đúng trên tập kiểm tra (Test Accuracy) đạt 99.7%. Tỉ lệ Training Accuracy và Test Accuracy đạt kết quả tốt hơn so với mô hình sử dụng thuật toán KNN ở mục 3.3.

Thực hiện vẽ ma trận nhầm lẫn (confusion matrix) để trực quan hóa hiệu suất của mô hình phân loại. Ma trận này biểu diễn số lượng dự đoán đúng và sai của mô hình cho từng lớp.

```
# Vẽ heatmap cho ma trận confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1']) # Đổi nhãn nếu cần

# Thêm nhãn và tiêu đề
plt.xlabel('Predicted Labels', fontsize=12)
plt.ylabel('True Labels', fontsize=12)
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```

Hình 2.9: Thực thi ma trận nhầm lẫn của thuật toán SVM



Hình 2.10: Ma trận nhầm lẫn của thuật toán SVM

- **Số lượng dự đoán đúng :** Class 0 đạt 1648 dữ liệu. Class 1 đạt 1341 dữ liệu
- **Số lượng dự đoán sai:** Class 0 sai 3 dữ liệu. Class 1 sai 4 dữ liệu.

2.5 Mô hình Random Forest

Random Forest hoạt động bằng cách tạo ra một số quyết định trong giai đoạn đào tạo. Mỗi cây được xây dựng bằng cách sử dụng một tập hợp ngẫu nhiên của dữ liệu để đo một tập hợp các tính năng ngẫu nhiên trong mỗi phân vùng. Tính ngẫu nhiên này tạo ra sự thay đổi giữa các cây riêng biệt, giảm nguy cơ cơ chế phù hợp và cải thiện hiệu suất.

Trong dự đoán, thuật toán tổng hợp kết quả của tất cả các cây bằng cách bỏ phiếu (cho các loại nhiệm vụ) hoặc bằng cách tính trung bình (cho các nhiệm vụ hồi phục). Quá trình này đã được xác định hợp lý, được nhiều cây hỗ trợ để hiểu được độ sâu của chúng, cung cấp một ví dụ về kết quả ổn định và chính xác.

```
# Initialize Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=1000, random_state=42)

# Train the Random Forest classifier on the CSP-filtered training data
rf_classifier.fit(X_train_csp, y_train)

# Make predictions on the CSP-filtered training data
y_train_pred = rf_classifier.predict(X_train_csp)

# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
# Make predictions on the CSP-filtered test data
y_test_pred = rf_classifier.predict(X_test_csp)
# Calculate testing accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Testing Accuracy:", test_accuracy)
print("Classification Report:")
print(classification_report(y_test, y_test_pred))
# Calculate and print confusion matrix for test data
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Hình 2.11: Huấn luyện mô hình sử dụng Random Forest

```

Training Accuracy: 1.0
Testing Accuracy: 0.9943257676902537
Classification Report:

```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1651
1	1.00	0.99	0.99	1345
accuracy			0.99	2996
macro avg	0.99	0.99	0.99	2996
weighted avg	0.99	0.99	0.99	2996

Hình 2.12: Kết quả mô hình sử dụng Random Forest

Tỉ lệ mẫu được dự đoán đúng trên tập huấn luyện (Training Accuracy) đạt 100%. Trong khi đó, tỷ lệ mẫu được dự đoán đúng trên tập kiểm tra (Test Accuracy) đạt 99.4%. Tỉ lệ Training Accuracy có xu hướng giống với mô hình sử dụng thuật toán Support Vector Machine. Tuy nhiên tỉ lệ Test Accuracy lại thấp hơn 2 mô hình sử dụng thuật toán K-Nearest Neighbors và Support Vector Machine.

Thực hiện vẽ ma trận nhầm lẫn (confusion matrix) để trực quan hóa hiệu suất của mô hình phân loại. Ma trận này biểu diễn số lượng dự đoán đúng và sai của mô hình cho từng lớp.

```

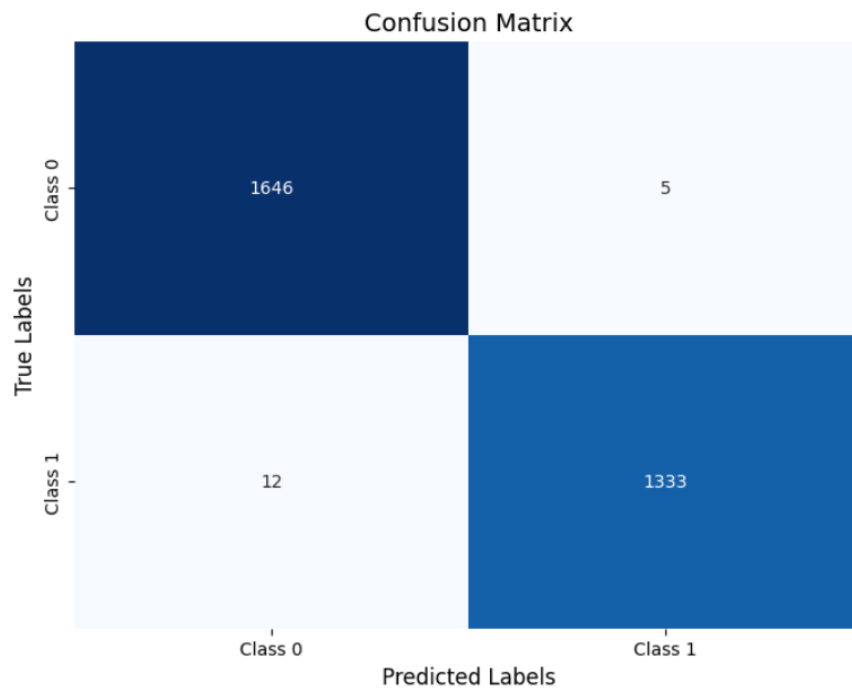
# Vẽ heatmap cho ma trận confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1']) # Đổi nhãn nếu cần

# Thêm nhãn và tiêu đề
plt.xlabel('Predicted Labels', fontsize=12)
plt.ylabel('True Labels', fontsize=12)
plt.title('Confusion Matrix', fontsize=14)
plt.show()

```

Hình 2.13: Ma trận nhầm lẫn của thuật toán Random Forest


```
Confusion Matrix:  
[[1646   5]  
 [  12 1333]]
```



Hình 2.14: Ma trận nhầm lẫn của thuật toán Random Forest

- **Số lượng dự đoán đúng :** Class 0 đạt 1646 dữ liệu. Class 1 đạt 1333 dữ liệu
- **Số lượng dự đoán sai:** Class 0 sai 5 dữ liệu. Class 1 sai 12 dữ liệu.

CHƯƠNG 3

Xây dựng ứng dụng

Từ phần xây dựng mô hình dự đoán ở trên, ta có thể thấy mô hình sử dụng thuật toán Support Vector Machine cho kết quả tốt nhất trong ba mô hình. Vì vậy, mô hình này sẽ được chọn và xây dựng ứng dụng dự đoán trạng thái của mắt.

3.1 Giao diện

Thực hiện tích hợp mô hình SVM đã training vào ứng dụng

```
# Tải mô hình SVM đã lưu
try:
    model = joblib.load('SVM_model.joblib') # Đảm bảo đường dẫn đúng tới mô hình
except FileNotFoundError:
    messagebox.showerror("Lỗi", "Mô hình SVM không tìm thấy! Vui lòng kiểm tra lại.")
    exit()
```

Hình 3.1: Tích hợp mô hình SVM

Giao diện ứng dụng gồm 14 ô để nhập dữ liệu, 1 ô hiển thị kết quả dự đoán và 3 nút thực hiện chức năng. Thực hiện tạo các ô nhập dữ liệu.

```
# Tạo cửa sổ ứng dụng Tkinter
root = tk.Tk()
root.title("EEG Detection")

# Tạo một khung chính với lề 100px
main_frame = tk.Frame(root, padx=100, pady=100)
main_frame.pack(fill="both", expand=True)

# Tạo các nhãn và ô nhập liệu cho mỗi đặc trưng
labels = ["AF3", "F7", "F3", "FC5", "T7", "P7", "O1", "O2", "P8", "T8", "FC6", "F4", "F8", "AF4"]
entries = []

for i, label in enumerate(labels):
    tk.Label(main_frame, text=f"{label}:").grid(row=i, column=0, sticky="w", padx=5, pady=5)
    entry = tk.Entry(main_frame)
    entry.grid(row=i, column=1, sticky="ew", padx=5, pady=5)
    entries.append(entry)
```

Hình 3.2: Tạo các ô nhập dữ liệu

Thực hiện tạo ô hiển thị kết quả

```
# Thêm một Label để hiển thị kết quả dự đoán
result_label = tk.Label(main_frame, text="", font=("Helvetica", 12))
result_label.grid(row=len(labels), column=0, columnspan=2, sticky="w", padx=5, pady=5)
```

Hình 3.3: Tạo ô hiển thị kết quả

Thực hiện tạo nút thực hiện dự đoán

```
# Tạo nút để dự đoán
predict_button = tk.Button(main_frame, text="Dự đoán", command=predict, bg="limegreen", fg="black", width=20)
predict_button.grid(row=len(labels) + 1, column=0, columnspan=2, sticky="ew", padx=5, pady=5)
```

Hình 3.4: Tạo nút thực hiện dự đoán

Thực hiện tạo nút xóa các ô đã nhập

```
# Tạo nút để dự đoán
predict_button = tk.Button(main_frame, text="Dự đoán", command=predict, bg="limegreen", fg="black", width=20)
predict_button.grid(row=len(labels) + 1, column=0, columnspan=2, sticky="ew", padx=5, pady=5)
```

Hình 3.5: Tạo nút xóa các ô đã nhập

Thực hiện tạo nút tải file dữ liệu cần dự đoán có định dạng excel

```
# Tạo nút để tải và dự đoán từ file Excel
def load_file():
    file_path = filedialog.askopenfilename(filetypes=[("Excel Files", "*.xlsx;*.xls")])
    if file_path:
        load_and_predict_from_excel(file_path)

load_button = tk.Button(main_frame, text="Tải dữ liệu cần dự đoán từ Excel", command=load_file, bg="gold", fg="black")
load_button.grid(row=len(labels) + 3, column=0, columnspan=2, sticky="ew", padx=5, pady=5)
```

Hình 3.6: Tạo nút tải file dữ liệu cần dự đoán

AF3:

F7:

F3:

FC5:

T7:

P7:

O1:

O2:

P8:

T8:

FC6:

F4:

F8:

AF4:

Dự đoán

Xóa

Tải dữ liệu cần dự đoán từ Excel

Hình 3.7: Tổng quan ứng dụng

3.2 Chức năng các hàm

Hàm `get_input_data()` có chức năng lấy dữ liệu từ các ô mà người dùng nhập, chuyển đổi thành một mảng. Thông báo nếu người dùng nhập dữ liệu không hợp lệ.

```
# Hàm lấy giá trị từ các ô nhập liệu
def get_input_data(entries):
    input_data = []
    for entry in entries:
        try:
            value = float(entry.get())
            input_data.append(value)
        except ValueError:
            messagebox.showerror("Lỗi", f"Vui lòng nhập giá trị hợp lệ cho {entry.get()}")
            return None
    return np.array(input_data).reshape(1, -1)
```

Hình 3.8: Hàm `get_input_data()`

Hàm `predict()` nhận dữ liệu từ 14 ô dữ liệu. Nó dùng mô hình đã tải để dự đoán trạng thái mắt nhắm hoặc mắt mở và trả về kết quả ở ô hiển thị kết quả 3.3

```
# Hàm dự đoán
def predict():
    input_data = get_input_data(entries)
    if input_data is None:
        return
    # Dự đoán với mô hình SVM
    prediction = model.predict(input_data)
    result = "Mắt nhắm" if prediction[0] == 0 else "Mắt mở"
    result_label.config(text=f"Dự đoán: {result}", font=("Helvetica", 12, "bold"))
```

Hình 3.9: Hàm `predict()`

Hàm `load_and_predict_from_excel()` đọc dữ liệu từ file excel và ghi lại kết quả dự đoán vào cột `eyeDetection` trong file

```

# Hàm đọc file Excel, gán nhãn vào cột 'eyeDetection' và lưu lại vào file Excel mới
def load_and_predict_from_excel(file_path):
    try:
        # Đọc dữ liệu từ file Excel
        df = pd.read_excel(file_path)

        # Gán nhãn vào cột 'eyeDetection' dựa trên mô hình
        features = df.iloc[:, :-1].values # Giả sử tất cả các cột trừ cột cuối là đặc trưng
        predictions = model.predict(features)

        # Thêm kết quả dự đoán vào cột 'eyeDetection'
        df['eyeDetection'] = predictions

        # Lưu lại dữ liệu đã gán nhãn vào file Excel mới
        save_path = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel Files", "*.xlsx")])
        if save_path:
            df.to_excel(save_path, index=False)
            messagebox.showinfo("Thành công", f"Đã lưu kết quả dự đoán vào file: {save_path}")
        else:
            messagebox.showerror("Lỗi", "Không lưu được file. Vui lòng chọn đường dẫn lưu.")

    except Exception as e:
        messagebox.showerror("Lỗi", f"Đã xảy ra lỗi khi xử lý file Excel: {e}")

```

Hình 3.10: Hàm load_and_predict_from_excel()

3.3 Kết quả

3.3.1 Dự đoán một trạng thái bất kì

Thực hiện nhập dữ liệu cần dự đoán tại 1 trạng thái bất kì.

The screenshot shows a window titled "EEG Detection" with a list of EEG channels and their corresponding numerical values. Below the list are three buttons: "Dự đoán" (green), "Xóa" (red), and "Tải dữ liệu cần dự đoán từ Excel" (yellow).

AF3:	-3.84901438803167
F7:	1.3421092768687
F3:	0.583266586835025
FC5:	1.23543442919397
T7:	0.786198265914215
P7:	-0.702534798828214
O1:	-0.745844099865561
O2:	6.04015960516605
P8:	-1.96245633470477
T8:	-1.24201216429396
FC6:	0.571860003581973
F4:	2.6406752656997
F8:	1.47796306786474
AF4:	-0.597484373181478

Buttons:

- Dự đoán
- Xóa
- Tải dữ liệu cần dự đoán từ Excel

Hình 3.11: Thực hiện nhập dữ liệu

Sau đó chọn nút "Dự đoán".

EEG Detection

AF3:

-3.84901438803167

F7:

1.3421092768687

F3:

0.583266586835025

FC5:

1.23543442919397

T7:

0.786198265914215

P7:

-0.702534798828214

O1:

-0.745844099865561

O2:

6.04015960516605

P8:

-1.96245633470477

T8:

-1.24201216429396

FC6:

0.571860003581973

F4:

2.6406752656997

F8:

1.47796306786474

AF4:

-0.597484373181478

Dự đoán: Mắt mở

Dự đoán

Xóa

Tải dữ liệu cần dự đoán từ Excel

Hình 3.12: Kết quả dự đoán

Kết quả dự đoán trạng thái mắt hiện tại là "Mở".

Chọn nút "Xóa" để xóa dữ liệu cũ và tiếp tục nhập dữ liệu mới.

EEG Detection

AF3:

-1.30630035775712

F7:

-0.342736085501873

F3:

0.914297262910801

FC5:

-0.986169124689759

T7:

-0.388857935638636

P7:

0.762299221503954

O1:

2.3090611565766

O2:

0.263767083963048

P8:

0.53896185395274

T8:

-0.716243298158755

FC6:

-0.313065926414935

F4:

0.0907273766170826

F8:

2.45641838258102

AF4:

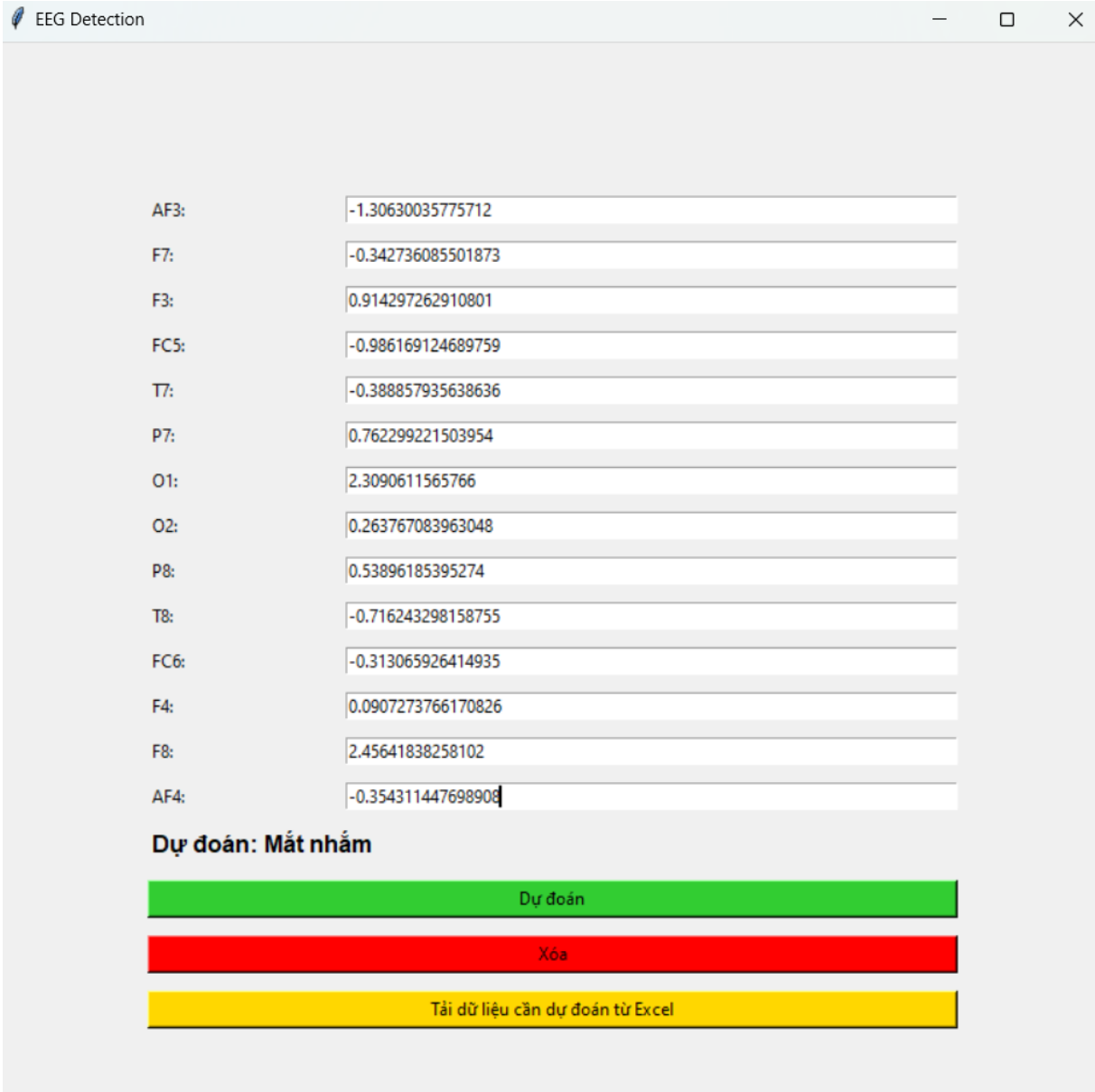
-0.354311447698908

Dự đoán

Xóa

Tải dữ liệu cần dự đoán từ Excel

Hình 3.13: Thực hiện nhập dữ liệu



Channel	Value
AF3:	-1.30630035775712
F7:	-0.342736085501873
F3:	0.914297262910801
FC5:	-0.986169124689759
T7:	-0.388857935638636
P7:	0.762299221503954
O1:	2.3090611565766
O2:	0.263767083963048
P8:	0.53896185395274
T8:	-0.716243298158755
FC6:	-0.313065926414935
F4:	0.0907273766170826
F8:	2.45641838258102
AF4:	-0.354311447698908

Dự đoán: Mắt nhắm

Dự đoán

Xóa

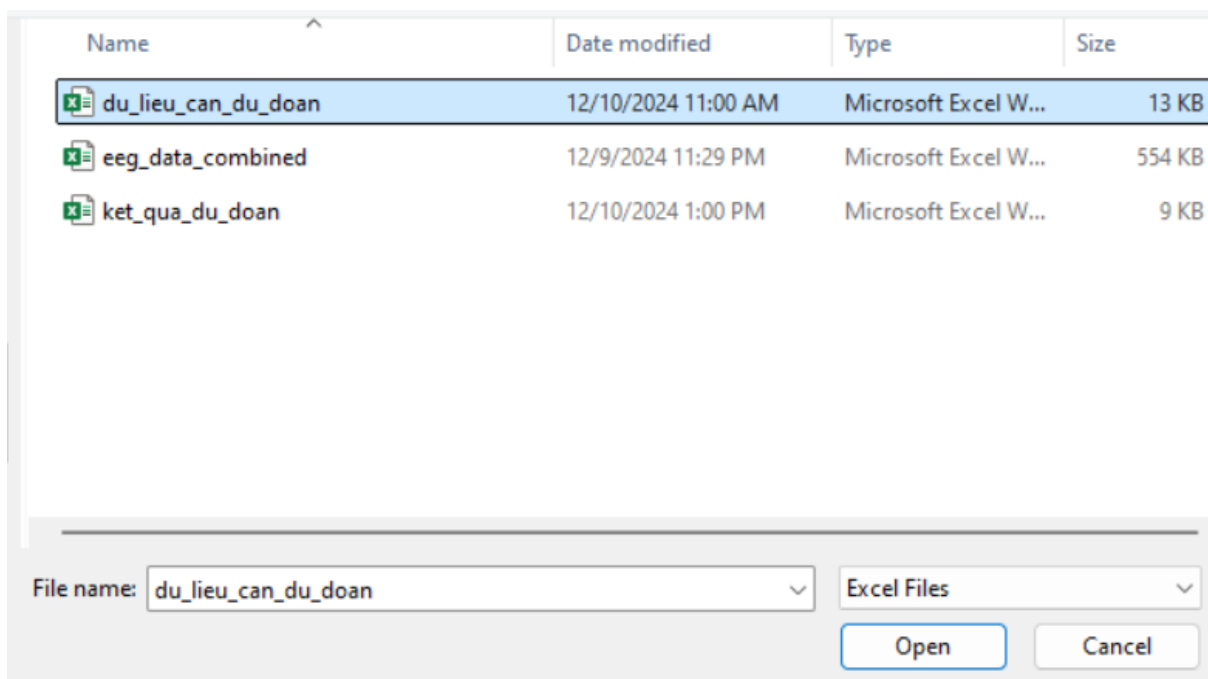
Tải dữ liệu cần dự đoán từ Excel

Hình 3.14: Kết quả dự đoán

Kết quả dự đoán trạng thái mắt hiện tại là "Nhắm"

3.3.2 Dự đoán hàng loạt các dữ liệu từ file excel

Thực hiện chọn nút "Tải dữ liệu cần dự đoán từ Excel" và chọn file cần dự đoán

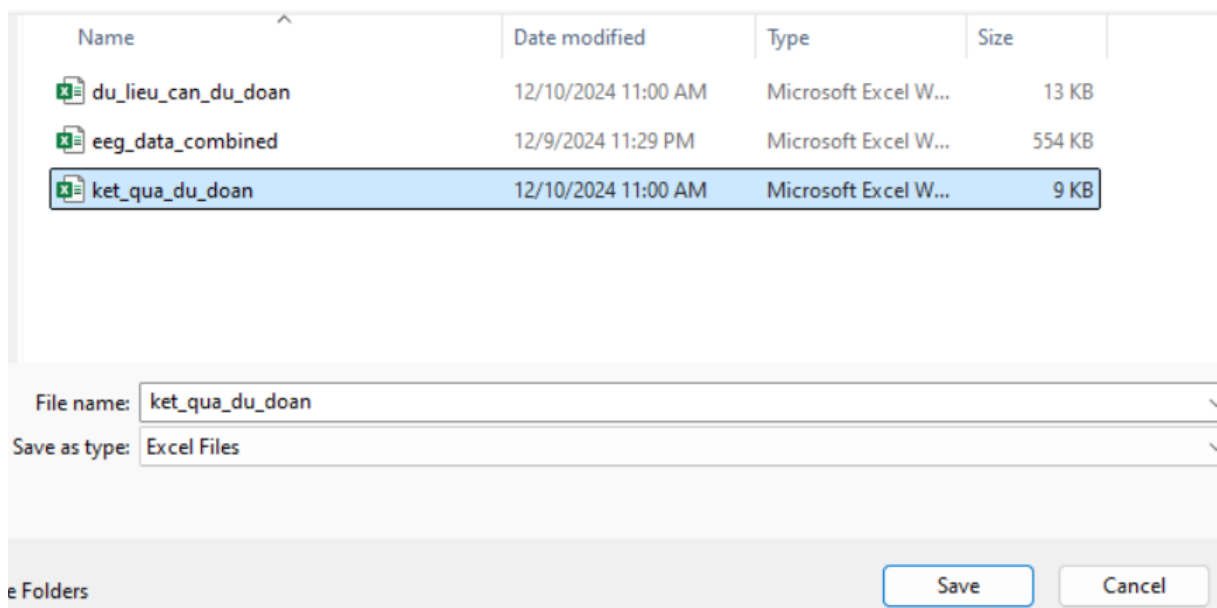


Hình 3.15: File dữ liệu excel cần dự đoán

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	eyeDetection
2	-3.84901	1.342109	0.583267	1.235434	0.786198	-0.70253	-0.74584	6.04016	-1.96246	-1.24201	0.57186	2.640675	1.477963	-0.59748	
3	-0.07187	-1.39369	-0.96957	-5.32532	2.448078	-5.04332	-0.73848	4.56574	3.965793	1.852594	2.493854	1.118504	0.424542	0.840116	
4	-0.553	-0.66049	-2.03014	-1.2962	-3.63728	0.493063	-1.37235	-3.15563	-0.2478	-1.40475	0.75425	2.182514	0.600194	0.465928	
5	-1.3063	-0.34274	0.914297	-0.98617	-0.38886	0.762299	2.309061	0.263767	0.538962	-0.71624	-0.31307	0.090727	2.456418	-0.35431	
6	-4.91877	-0.1875	1.192626	-4.90267	3.520002	1.817659	-4.09871	4.490628	0.008493	0.471587	1.090139	1.497477	-4.54411	-2.24985	
7	4.904868	-4.11379	0.641682	-1.15575	0.443362	5.54213	-1.43712	-1.63715	-4.92041	2.978692	-4.47354	-0.24863	-1.59932	0.971199	
8	-0.59517	-0.25391	2.285663	3.227375	0.510309	-4.56711	-0.52728	0.468668	0.047627	-1.03694	2.072067	-0.77565	-2.65073	0.365625	
9	1.600377	-2.69766	0.534297	-6.8663	1.115963	-0.86902	-1.86263	-0.05558	2.226632	-0.12259	-2.41831	0.841902	-0.11249	-1.9319	
10	0.166159	-0.29563	5.155386	1.803155	-0.58709	-2.3439	-2.01407	-0.61977	-2.42919	3.947392	-0.37245	-0.86076	0.334676	-2.42469	
11	-1.1499	3.193444	-3.23607	-1.51026	-2.17316	1.208454	0.09567	-0.5586	0.155028	2.396948	-1.81376	-1.56249	1.27962	-1.04838	
12	0.114196	1.918758	-1.47608	-1.8008	-2.67256	-2.87672	0.850222	4.538651	1.798193	-0.25579	1.403319	1.521509	-1.57579	-1.87695	
13	0.518041	-5.14003	0.254298	0.170384	-0.54236	2.56069	-7.07826	1.724002	-4.35444	6.478343	-7.57689	-5.78547	14.57698	6.010322	
14	1.059483	0.120201	0.417633	-0.82248	-0.68931	-4.07923	-2.6577	-4.53849	3.483077	1.717908	-0.72067	1.294669	0.726732	1.418821	
15	1.769161	-5.064	-0.67359	-5.61203	-1.37525	0.256292	-3.06937	2.682624	0.162213	-3.96138	0.153087	-2.53875	3.766729	2.042881	
16	2.058613	1.293498	3.327614	-0.98122	0.108098	-2.34588	1.123614	3.564759	1.996716	2.467783	3.496258	5.12812	-0.44793	-3.83097	
17	1.801539	1.353161	-5.40484	-2.02023	0.300507	0.163679	-2.16063	-0.96767	1.764359	1.316904	1.047176	1.193752	-0.92665	-0.55486	
18	0.417381	0.027459	-1.96266	0.525088	-3.46948	1.165312	-2.59411	-2.7309	1.210074	0.14455	0.13222	0.405113	-5.15329	-2.72872	
19	-3.8063	-2.11195	-1.52288	-0.28838	3.158002	-1.01335	5.254051	-0.94016	-0.99196	7.166939	3.112855	1.628742	2.3528	-0.80138	
20	-3.01955	1.138485	4.084786	1.447255	-1.26832	5.841587	-5.03028	2.318517	2.781189	2.967992	0.696466	0.231599	0.327517	-1.73731	
21	-0.48635	-0.98458	-0.31488	4.361256	-6.45129	-0.29103	-0.47802	4.945649	2.712895	-1.66895	-6.13088	4.98161	-10.5537	6.030532	

Hình 3.16: Dữ liệu trong file excel cần dự đoán

Sau đó chọn file để lưu kết quả. Ta có thể ghi đè lên file dữ liệu gốc hoặc chọn 1 file mới



Hình 3.17: Chọn file lưu kết quả

Kết quả dự đoán đã được lưu vào cột eyeDetection.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	eyeDetection
2	-3.84901	1.342109	0.583267	1.235434	0.786198	-0.70253	-0.74584	6.04016	-1.96246	-1.24201	0.57186	2.640675	1.477963	-0.59748	1
3	-0.07187	-1.39369	-0.96957	-5.32532	2.448078	-5.04332	-0.73848	4.56574	3.965793	1.852594	2.493854	1.118504	0.424542	0.840116	1
4	-0.553	-0.66049	-2.03014	-1.2962	-3.63728	0.493063	-1.37235	-3.15563	-0.2478	-1.40475	0.75425	2.182514	0.600194	0.465928	1
5	-1.3063	-0.34274	0.914297	-0.98617	-0.38886	0.762299	2.309061	0.263767	0.538962	-0.71624	-0.31307	0.090727	2.456418	-0.35431	0
6	-4.91877	-0.1875	1.192626	-4.90267	3.520002	1.817659	-4.09871	4.490628	0.008493	0.471587	1.090139	1.497477	-4.54411	-2.24985	1
7	4.904868	-4.11379	0.641682	-1.15575	0.443362	5.54213	-1.43712	-1.63715	-4.92041	2.978692	-4.47354	-0.24863	-1.59932	0.971199	0
8	-0.59517	-0.25391	2.285663	3.227375	0.510309	-4.56711	-0.52728	0.468668	0.047627	-1.03694	2.072067	-0.77565	-2.65073	0.365625	1
9	1.600377	-2.69766	0.534297	-6.8663	1.115963	-0.86902	-1.86263	-0.05558	2.226632	-0.12259	-2.41831	0.841902	-0.11249	-1.9319	0
10	0.166159	-0.29563	5.155386	1.803155	-0.58709	-2.3439	-2.01407	-0.61977	-2.42919	3.947392	-0.37245	-0.86076	0.334676	-2.42469	1
11	-1.1499	3.193444	-3.23607	-1.51026	-2.17316	1.208454	0.09567	-0.5586	0.155028	2.396948	-1.81376	-1.56249	1.27962	-1.04838	0
12	0.114196	1.918758	-1.47608	-1.8008	-2.67256	-2.87672	0.850222	4.538651	1.798193	-0.25579	1.403319	1.521509	-1.57579	-1.87695	0
13	0.518041	-5.14003	0.254298	0.170384	-0.54236	2.56069	-7.07826	1.724002	-4.35444	6.478343	-7.57689	-5.78547	14.57698	6.010322	1
14	1.059483	0.120201	0.417633	-0.82248	-0.68931	-4.07923	-2.6577	-4.53849	3.483077	1.717908	-0.72067	1.294669	0.726732	1.418821	0
15	1.769161	-5.064	-0.67359	-5.61203	-1.37525	0.256292	-3.06937	2.682624	0.162213	-3.96138	0.153087	-2.53875	3.766729	2.042881	0
16	2.058613	1.293498	3.327614	-0.98122	0.108098	-2.34588	1.123614	3.564759	1.996716	2.467783	3.496258	5.12812	-0.44793	-3.83097	0
17	1.801539	1.353161	-5.40484	-2.02023	0.300507	0.163679	-2.16063	-0.96767	1.764359	1.316904	1.047176	1.193752	-0.92665	-0.55486	0
18	0.417381	0.027459	-1.96266	0.525088	-3.46948	1.165312	-2.59411	-2.7309	1.210074	0.14455	0.13222	0.405113	-5.15329	-2.72872	1
19	-3.8063	-2.11195	-1.52288	-0.28838	3.158002	-1.01335	5.254051	-0.94016	-0.99196	7.166939	3.112855	1.628742	2.3528	-0.80138	0
20	-3.01955	1.138485	4.084786	1.447255	-1.26832	5.841587	-5.03028	2.318517	2.781189	2.967992	0.696466	0.231599	0.327517	-1.73731	0
21	-0.48635	-0.98458	-0.31488	4.361256	-6.45129	-0.29103	-0.47802	4.945649	2.712895	-1.66895	-6.13088	4.98161	-10.5537	6.030532	1

Hình 3.18: Kết quả dự đoán

Giá trị "1" biểu thị trạng thái mắt Mở.

Giá trị "0" biểu thị trạng thái mắt Nhắm.