Here is the text-based representation of our conversation:

**Conversation with BLACKBOX AI**

**User:** No, Try to understand that If I need to predict that this particular person is having this MS Disease of not then how will I have to procced?

**BLACKBOX AI:** I understand your question now. You have trained the model and now you want to use it to predict whether a particular person has MS disease or not.

To do this, you need to provide the model with the input features of the person you want to predict. In your case, the input features are:

* **age**
* **EDSS**
* **orig\_MSSS**
* **updated\_MSSS**
* **inf\_act\_6m**
* **inf\_act\_12m**

You need to provide these values for the person you want to predict. Let's say you have a new person with the following values:

* **age**: 35
* **EDSS**: 2.5
* **orig\_MSSS**: 1.2
* **updated\_MSSS**: 1.5
* **inf\_act\_6m**: 0.8
* **inf\_act\_12m**: 0.9

You can create a new DataFrame with these values:

python

EditCopy code

1new\_person = pd.DataFrame({'age': [35], 'EDSS': [2.5], 'orig\_MSSS': [1.2], 'updated\_MSSS': [1.5], 'inf\_act\_6m': [0.8], 'inf\_act\_12m': [0.9]})

Then, you can use the trained model to make a prediction:

python

EditCopy code

1prediction = clf.predict(new\_person)

The **prediction** variable will contain the predicted class label (0 or 1) indicating whether the person has MS disease or not.

You can also use the **predict\_proba** method to get the probability of the person having MS disease:

python

EditCopy code

1probability = clf.predict\_proba(new\_person)

The **probability** variable will contain an array with two values: the probability of the person having MS disease (class label 1) and the probability of the person not having MS disease (class label 0).

Note that you need to make sure that the input values are in the same scale and format as the training data. If the input values are not in the same scale, you may need to normalize or scale them before making a prediction.

**User:** There should be something like it should be showing that whether this person is having this disease or not?

**BLACKBOX AI:** You have trained the model and made predictions using the testing data. Now, you want to use the model to predict whether a new person has MS disease or not.

You have created a new DataFrame **new\_person** with the input features for the new person:

python

EditCopy code

1new\_person = pd.DataFrame({'age': [35], 'EDSS': [2.5], 'orig\_MSSS': [1.2], 'updated\_MSSS': [1.5], 'inf\_act\_6m': [0.8], 'inf\_act\_12m': [0.9]})

And you have used the trained model to make a prediction:

python

EditCopy code

1prediction = clf.predict(new\_person)

However, you are not seeing the output of the prediction.

To see the output, you can print the **prediction** variable:

python

EditCopy code

1print(prediction)

This will print the predicted class label (0 or 1) indicating whether the person has MS disease or not.

Alternatively, you can use the **predict\_proba** method to get the probability of the person having MS disease:

python

EditCopy code

1probability = clf.predict\_proba(new\_person)

2print(probability)

This will print an array with two values: the probability of the person having MS disease (class label 1) and the probability of the person not having MS disease (class label 0).

You can also use the **print** statement to display a message indicating whether the person has MS disease or not based on the prediction:

python

EditCopy code

1if prediction == 1:

2 print("The person is likely to have MS disease.")

3else:

4 print("The person is unlikely to have MS disease.")

Make sure to add these print statements after the prediction code to see the output.

**User:** File Path: c:\Users\win11\Desktop\College-Final Year\Java Revision\MS\_Project.ipynb

File Content: ... (rest of the file content)

**BLACKBOX AI:** ... (rest of the response)

You can save this text-based representation of our conversation as a PDF file using your preferred method, such as copying and pasting the text into a word processor or text editor and then saving it as a PDF file.

**Part-02**

**Sure! Here is the conversation in the format you requested:**

**User: Ohh there are 4 types of values present.**

**15 m patient SPMS 1-1-2008 1-1-2003 False False True 35 4.5 4.17 5.26 8.06 False False apurito Ja 2x Everion 16 m patient RRMS 4-1-2017 4-1-2017 False False False 44 3 6.82 7.16 5.05 False False guayabo Ja 2x Everion, Nexus 5s 17 w patient RRMS 2-1-2018 1-1-2013 False False False 37 2 3.17 3.90 4.41 False False cerati Ja 2x Everion, Nexus 5s 18 w patient RRMS 1-1-2005 1-1-2004 False False True 39 3.5 3.18 4.29 6.78 False False boavista Ja 2x Everion, Nexus 5s 19 w patient PPMS 2-17-2020 9-1-2019 False False True 39 2.5 7.08 6.79 5.26 False False tucupita Ja 2x Everion, Nexus 5s 20 m patient RRMS 2-1-2017 1-1-2017 False False False 26 1.5 3.34 3.98 4.67 False True trinidad Ja 2x Everion, Nexus 5s 21 m patient encephalitis 70 tobago Nein 22 w patient RRMS 2-1-2020 2-1-2020 False False False 24 2.5 7.08 6.79 7.26 True True panama Ja 2x Everion, Nexus 5s 23 w patient PPMS 3-1-2009 2-1-2009 False False False 55 6.5 8.14 8.60 7.89 False False hural Ja Google Pixel 3a**

**You: Based on the provided dataset, there are four types of values present in the MS\_type column:**

1. **RRMS: Relapsing-Remitting Multiple Sclerosis**
2. **SPMS: Secondary Progressive Multiple Sclerosis**
3. **PPMS: Primary Progressive Multiple Sclerosis**
4. **encephalitis: A type of inflammation of the brain**

**The first three values represent different types of Multiple Sclerosis (MS) disease, while the last value, encephalitis, is not a type of MS but an inflammation of the brain that can sometimes be mistaken for MS.**

**User: If all these represents these 4 types of MS disease the how the model will identify that this is not a MS type? Because every MS\_typefield contains either of these 4 values.**

**IT Means the target column should not be MS\_TYPE as it is only stating the type of MS disease.**

**Kindly give the accurate answer to the point and which column should be representing the final predition that whether this persion is with MS Disease or not because right not it is only showing the type of disease.**

**You: You are correct that the MS\_type column only indicates the type of MS disease, not whether the person has MS disease or not.**

**To identify whether a person has MS disease or not, you need a separate column that indicates the presence or absence of MS disease. This column is typically called the "target" or "response" variable.**

**In your dataset, I couldn't find a column that explicitly indicates the presence or absence of MS disease. However, I noticed that there is a column called clinical\_active which might be related to the presence of MS disease. This column contains boolean values (True/False) which could indicate whether the person has active MS disease or not.**

**If this is the case, you could use the clinical\_active column as the target variable instead of MS\_type. This would allow the model to predict whether a person has MS disease or not, rather than just predicting the type of MS disease.**

**Alternatively, you could create a new column that combines information from multiple columns to indicate the presence or absence of MS disease. For example, you could create a new column called has\_MS that is True if the person has any of the following conditions:**

* **clinical\_active is True**
* **radiological\_active is True**
* **progressive is True**
* **MS\_type is not encephalitis (since this is not a type of MS disease)**

**You would need to carefully evaluate the relationships between these columns and the presence or absence of MS disease to determine the best approach.**

**In summary, the MS\_type column is not suitable as the target variable for predicting whether a person has MS disease or not. You need to identify or create a separate column that indicates the presence or absence of MS disease, and use that column as the target variable instead.**

# Project Report: Multiple Sclerosis Prediction using IoT Devices

## Table of Contents

1. Introduction

2. Code Analysis

2.1 Libraries Used

2.2 Data Preprocessing

2.3 Machine Learning Model

2.4 Model Evaluation

2.5 Prediction on New Data

3. IoT Devices and Data Collection

3.1 Smart Watches / Fitness Trackers

3.2 Smartphones

3.3 Specialized Devices

4. Data Integration Process

5. Building a Streamlit Interface

6. Mobile Data Collection

7. Conclusion

## 1. Introduction

This project aims to predict Multiple Sclerosis (MS) using data collected from various IoT devices. By leveraging machine learning techniques and real-time data from wearables and smartphones, we can potentially identify early signs of MS and assist in timely diagnosis.

## 2. Code Analysis

### 2.1 Libraries Used

- pandas: For data manipulation and analysis

- sklearn: For machine learning operations

- SimpleImputer: For handling missing values (though not used in the provided code)

### 2.2 Data Preprocessing

- Data is loaded from an Excel file ('Patient\_health\_info.xlsx')

- Rows with NaN values are removed using `data.dropna(inplace=True)`

- Features selected: 'age', 'EDSS', 'orig\_MSSS', 'updated\_MSSS', 'inf\_act\_6m', 'inf\_act\_12m'

- Target variable: 'MS\_type'

### 2.3 Machine Learning Model

- Algorithm: Random Forest Classifier

- The Random Forest algorithm is an ensemble learning method that constructs multiple decision trees and combines their outputs for prediction.

- Parameters:

- n\_estimators=100: Number of trees in the forest

- random\_state=42: Seed for reproducibility

### 2.4 Model Evaluation

- The dataset is split into training (80%) and testing (20%) sets

- Accuracy is used as the evaluation metric:

```python

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

```

### 2.5 Prediction on New Data

- A sample prediction is made on a new person's data:

```python

new\_person = pd.DataFrame({'age': [35], 'EDSS': [2.5], 'orig\_MSSS': [1.2], 'updated\_MSSS': [1.5], 'inf\_act\_6m': [0.8], 'inf\_act\_12m': [0.9]})

prediction = clf.predict(new\_person)

probability = clf.predict\_proba(new\_person)

```

### 2.6 Plotting and Graphs

While the provided code doesn't include plotting, we can enhance the project by adding visualizations:

1. Feature Importance Plot:

```python

import matplotlib.pyplot as plt

feature\_importance = clf.feature\_importances\_

plt.bar(features, feature\_importance)

plt.title('Feature Importance in MS Prediction')

plt.xlabel('Features')

plt.ylabel('Importance')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

```

2. Confusion Matrix:

```python

from sklearn.metrics import confusion\_matrix

import seaborn as sns

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

```

3. ROC Curve (for binary classification):

```python

from sklearn.metrics import roc\_curve, auc

fpr, tpr, \_ = roc\_curve(y\_test, clf.predict\_proba(X\_test)[:,1])

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

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(loc="lower right")

plt.show()

```

## 3. IoT Devices and Data Collection

### 3.1 Smart Watches / Fitness Trackers

- Devices: Apple Watch, Fitbit, Garmin, etc.

- Data collected:

- Heart Rate

- Heart Rate Variability

- Skin Temperature

- Galvanic Skin Response

- Energy Expenditure

- Steps

- Sleep Duration

Collection Process:

1. Develop an app for the smartwatch platform (e.g., watchOS for Apple Watch)

2. Use the device's health and fitness APIs to access sensor data

3. Set up background processes to collect data at regular intervals

4. Implement data buffering and batch sending to conserve battery life

### 3.2 Smartphones

- Devices: iPhones, Android phones

- Data collected:

- Phone Lock/Unlock frequency

- Tapping Task performance

Collection Process:

1. Develop a cross-platform mobile app (e.g., using React Native or Flutter)

2. For lock/unlock data:

- On Android: Use `UsageStatsManager` to track app usage

- On iOS: Use `ScreenTime API` (with user permission)

3. For tapping task:

- Implement a simple game within the app

- Record metrics like tapping speed, accuracy, and patterns

### 3.3 Specialized Devices

- Respiration Rate: Smart clothing (e.g., Hexoskin) or chest straps

- Blood Pulse Wave: Advanced pulse oximeters or photoplethysmography (PPG) sensors

Collection Process:

1. Integrate with device-specific SDKs or APIs

2. Set up Bluetooth Low Energy (BLE) connections for real-time data streaming

3. Implement data validation to ensure accuracy of specialized measurements

## 4. Data Integration Process

1. Central Data Hub:

- Set up a secure cloud server (e.g., AWS, Google Cloud) to receive and store data

- Implement API endpoints for each data type

2. Data Standardization:

- Create a common data format for all devices

- Implement data transformation functions to convert device-specific formats

3. Data Synchronization:

- Set up regular sync intervals for each device (e.g., hourly for smartwatches, daily for smartphones)

- Implement retry mechanisms for failed syncs

4. Data Validation and Cleaning:

- Set up automated checks for data integrity and completeness

- Implement algorithms to handle missing data (e.g., interpolation)

5. Feature Extraction:

- Process raw data to extract relevant features (e.g., calculate daily averages, identify trends)

- Align extracted features with the ML model's input requirements

## 5. Building a Streamlit Interface

1. Set up Streamlit:

```

pip install streamlit

```

2. Create a new file `app.py`:

```python

import streamlit as st

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

import joblib

# Load the trained model

model = joblib.load('ms\_prediction\_model.joblib')

st.title('MS Prediction App')

# Create input fields

age = st.number\_input('Age', min\_value=0, max\_value=120)

edss = st.number\_input('EDSS Score', min\_value=0.0, max\_value=10.0)

orig\_msss = st.number\_input('Original MSSS', min\_value=0.0)

updated\_msss = st.number\_input('Updated MSSS', min\_value=0.0)

inf\_act\_6m = st.number\_input('Inflammatory Activity (6 months)', min\_value=0.0)

inf\_act\_12m = st.number\_input('Inflammatory Activity (12 months)', min\_value=0.0)

if st.button('Predict'):

# Create a DataFrame with the input data

input\_data = pd.DataFrame({

'age': [age],

'EDSS': [edss],

'orig\_MSSS': [orig\_msss],

'updated\_MSSS': [updated\_msss],

'inf\_act\_6m': [inf\_act\_6m],

'inf\_act\_12m': [inf\_act\_12m]

})

# Make prediction

prediction = model.predict(input\_data)

probability = model.predict\_proba(input\_data)

st.write(f"Prediction: {prediction[0]}")

st.write(f"Probability: {probability[0]}")

# Add data visualization

if st.checkbox('Show recent data'):

st.write("Fetching recent data...")

# Here you would fetch and display recent data from your database

# For example:

# recent\_data = fetch\_recent\_data() # Implement this function

# st.dataframe(recent\_data)

# Add a section for uploading new data

uploaded\_file = st.file\_uploader("Choose a CSV file to update the dataset", type="csv")

if uploaded\_file is not None:

data = pd.read\_csv(uploaded\_file)

st.write(data)

# Here you would add code to process and integrate this new data

# update\_dataset(data) # Implement this function

```

3. Run the Streamlit app:

```

streamlit run app.py

```

## 6. Mobile Data Collection

To collect data from mobile phones:

1. Develop a mobile app (iOS and Android):

- Use React Native or Flutter for cross-platform development

- Implement background services for continuous data collection

2. Implement the following features:

- User authentication and profile management

- Background data collection for phone usage patterns

- Scheduled prompts for user input (e.g., symptom logging)

- Tapping task games for motor function assessment

3. Data collection specifics:

- Phone usage:

- Android: Use `UsageStatsManager`

- iOS: Use `ScreenTime API` (requires user permission)

- Accelerometer data:

- Android: `SensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER)`

- iOS: `CoreMotion` framework

- Location data (if relevant):

- Use GPS data to track mobility patterns

- Ensure compliance with privacy regulations

4. Data transmission:

- Implement secure API calls to your central server

- Use batch processing to minimize battery drain

- Implement retry mechanisms for failed data uploads

5. User engagement:

- Implement push notifications for reminders

- Provide visualizations of collected data to users

- Gamify data collection to encourage regular use

6. Privacy and security:

- Implement end-to-end encryption for data transmission

- Provide clear privacy policies and data usage information

- Allow users to control data sharing preferences

7. Integration with wearables:

- Implement HealthKit (iOS) and Google Fit (Android) integrations

- Sync data from connected smartwatches and fitness trackers

## 7. Conclusion

This project demonstrates the potential of using IoT devices and machine learning for early detection and monitoring of Multiple Sclerosis. By collecting diverse data points from everyday devices, we can provide valuable insights to healthcare professionals and improve patient outcomes. Future work could focus on improving the accuracy of the prediction model, expanding the range of data sources, and conducting clinical trials to validate the effectiveness of this approach.