Northeastern Illinois University

Computer Science Department

Chicago, Illinois

Master's Project Report

Mobile App To Predict Migraine Types Using Machine Learning

By

On-uma Lomsomboot

Under the guidance of

Xiwei Wang, Department Chair, Associate Professor

Yi Yang, Assistant Professor

May, 2023

Abstract

This study focuses on the development of a mobile application using the React Native framework that utilizes machine learning algorithms to predict and categorize migraine types. Migraine is a prevalent neurological disorder with significant implications for treatment and patient outcomes. The mobile app allows users to input their data, and through machine learning, accurately predicts their migraine type.

The dataset used in this study consisted of 400 records, and to address class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) was employed. By oversampling the minority classes, the dataset became more balanced, leading to improved model performance. The Predictive Power Score (PPS) was used to identify the top features influencing migraine type, including intensity, visual symptoms, character, location, and frequency.

To optimize the model, Cramer's corrected statistic was applied to reduce correlation between nominal categorical variables. The Extra Trees Classifier algorithm was selected as the best performing model, achieving a training accuracy of 96%. Recursive Feature Elimination with Cross-Validation (RFECV) was utilized to determine the optimal number of features, enhancing the model's efficiency.Furthermore, the model's hyperparameters were fine-tuned using Grid Search CV to maximize performance. This comprehensive approach resulted in a robust and accurate model for migraine type prediction.

The mobile app, integrated with a Flask API hosted on AWS EC2, provides a seamless user experience, allowing users to conveniently access their predicted migraine type. The mobile app also leverages Google Cloud Firebase for real-time data storage and authentication. This ensures secure and reliable data management, while enabling users to easily access and share their data using their email credentials.

The implications of this research are significant. The mobile app offers users personalized treatment plans based on accurate migraine type predictions. By tailoring treatments to individual needs, healthcare outcomes can be improved for migraine sufferers. Additionally, the model contributes to a deeper understanding of migraine mechanisms, potentially leading to novel insights and therapies.

In conclusion, this study demonstrates the potential of machine learning algorithms integrated with a mobile app framework for accurate migraine type prediction. By leveraging the React Native framework, Google Cloud Firebase, and hosting the Flask API on AWS EC2, a comprehensive and user-friendly solution is provided. The research findings pave the way for personalized treatment plans and a better understanding of migraines, benefiting both patients and healthcare professionals.

Introduction

Migraine is a complex neurological disorder that affects a significant portion of the global population, with a substantial impact on quality of life and work productivity. Accurate identification of specific migraine types is crucial for developing effective and tailored treatment plans. The classification of migraine types is based on several criteria, including frequency, duration, severity, and the presence of specific symptoms.

Despite significant advances in our understanding of migraines and the development of various treatment options, accurate diagnosis remains a significant challenge. The clinical diagnosis of migraine types is often based on subjective patient reports, which can be inconsistent and incomplete. Furthermore, the classification of migraine types can be challenging, given the heterogeneity of the disorder and the variability of symptoms between individuals. These challenges highlight the need for more accurate and reliable methods for diagnosing and classifying migraine types.

Machine learning has emerged as a promising tool in healthcare, facilitating diagnosis and treatment of various medical conditions, including migraines. Machine learning algorithms can process large amounts of data, identify complex patterns, and make predictions with high accuracy. By leveraging these capabilities, machine learning can enable more accurate and reliable identification of migraine types, enabling the development of tailored and effective treatment plans.

This project aims to leverage machine learning algorithms to accurately determine migraine types, empowering individuals to seek targeted medical assistance from healthcare professionals. The primary objective of this study is to analyze collected data to uncover insights that contribute to a more comprehensive understanding of migraine types. Additionally, we aim to explore the development of a user-friendly mobile app framework that provides easy access for users to input their data and obtain their migraine type.

Furthermore, we will investigate cloud-based services where the mobile app can interact with machine learning trained models to fetch the models and access real-time data storage. This approach will enable the development of a more sophisticated and flexible system for identifying and diagnosing migraine types, facilitating more accurate and personalized treatment plans.
Through this innovative approach, we seek to enhance migraine management, help individuals regain control over their lives, and reduce the burden of this debilitating condition. The potential benefits of this project extend beyond personalized treatment plans. By accurately predicting migraine types, this project can contribute to a better understanding of the underlying mechanisms of migraines, leading to new insights and treatments. Additionally, this project can help clinicians identify the most effective treatment options for their patients.

Literature Review

Migraine is a neurological disorder affecting over a billion people worldwide, with a significant impact on quality of life and productivity. Accurate diagnosis and classification of migraine types are critical for developing effective treatment strategies. In recent years, machine learning has emerged as a powerful tool for diagnosis and treatment of various medical conditions, including migraines (Alzubaidi et al., 2019).

Several studies have explored the use of machine learning in the classification of migraine types. For example, a study by Amin et al. (2018) used a support vector machine (SVM) algorithm to classify migraine types based on clinical features. The study reported an accuracy of 86.3%, demonstrating the potential of machine learning in accurate classification of migraine types.

One challenge often encountered in healthcare applications of machine learning is the presence of imbalanced datasets, where certain conditions or outcomes may be rare compared to others. In an imbalanced dataset, the majority class has significantly more instances than the minority class, which can lead to biased models that favor the majority class and perform poorly on the minority class (He & Garcia, 2009). This issue is especially critical in healthcare, where accurate identification of rare conditions can have a significant impact on patient outcomes. Several techniques have been proposed to address class imbalance, such as oversampling the minority class, undersampling the majority class, or using synthetic data generation methods like the Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al., 2002).

One of the significant challenges in applying machine learning to healthcare or any other domain involves the selection of the optimal algorithm for model training. The performance of machine learning models can vary significantly depending on the problem at hand, the structure and quality of the data, and the choice of the algorithm. Furthermore, each algorithm has its own set of hyperparameters that can drastically influence model performance, and tuning these parameters can be a time-consuming and computationally intensive process (Bergstra & Bengio, 2012).

To address this challenge, we decided to utilize the LazyPredict library in our study. LazyPredict is a Python library that simplifies the process of training and evaluating machine learning models. It includes functionalities that allow users to fit and evaluate multiple models with default hyperparameters, providing a fast and efficient way to identify promising models for a given task (Oliphant, 2020). This approach allowed us to quickly benchmark a wide range of models and focus our efforts on fine-tuning the most promising ones, saving us considerable time and computational resources.

In the field of data science, especially in the context of healthcare, an often-encountered challenge is the decision to exclude certain features from the dataset prior to, or without consultation with, domain experts. This is a critical aspect of model building, as the inclusion of irrelevant or redundant features can lead to overfitting and decrease the interpretability of the model.

To mitigate this, data scientists employ various feature selection methods to identify and remove such features (Guyon & Elisseeff, 2003).

One popular approach is to compute correlations between pairs of variables and eliminate those with high correlations, as they tend to provide redundant information. For categorical variables, this can be achieved using Cramer's V, a measure of association between two nominal variables that can range from 0 (no association) to 1 (complete association) (Cramér, 1946).

Another innovative approach is to use the Predictive Power Score (PPS), a symmetric, data-type-agnostic score that can detect linear or non-linear relationships between two variables. PPS can be a more effective alternative to correlation for feature selection as it can capture more complex relationships and interactions between variables (Fleischer et al., 2020).
By incorporating Cramer's V and PPS in our analysis, we were able to perform a thorough and robust feature selection, thereby improving the performance and interpretability of our model.

In addition to these studies, there is a growing interest in integrating machine learning models with web APIs, cloud services, and mobile app frameworks for healthcare applications.
Browning et al. (2020) discussed the deployment of deep learning models as APIs for mobile app developers, while Sharma et al. (2020) explored the use of cloud-based healthcare services using Amazon Web Services (AWS) Elastic Compute Cloud (EC2) for secure, scalable, and cost-effective solutions.

React Native has gained significant popularity as a preferred framework for mobile app development, owing to its cross-platform capabilities and user-friendly nature. A comprehensive study conducted by Rahman et al. (2020) extensively discussed the advantages of React Native in the context of healthcare app development. The research highlighted notable benefits, such as reduced development time and enhanced user experience, making it a compelling choice for healthcare app developers. Furthermore, the study compared React Native with other frameworks, emphasizing its superiority in terms of code reusability, faster development cycles, and cost-effectiveness (Rahman et al., 2020). The findings underscore the positive impact of utilizing React Native for healthcare app development, particularly in terms of time and cost efficiency, ultimately contributing to improved user satisfaction and engagement in healthcare settings.

The AWS EC2 service is increasingly used for deploying machine learning models in healthcare. Al-Fuqaha et al. (2019) discussed the benefits of using AWS EC2, such as improved security, cost-effectiveness, and scalability, which are essential for healthcare applications. In addition to AWS EC2, Amazon Elastic Container Service (ECS) offers advantages for deploying and managing machine learning models in the cloud. Kim et al. (2021) demonstrated the importance of utilizing container orchestration services like ECS for deploying and managing machine learning models in the cloud, ensuring high availability and efficient resource management.

Considering these advancements, the present study aimed to accurately determine migraine types using machine learning algorithms and develop a user-friendly mobile app using the React Native Framework for users to input their data and obtain their migraine type.

We integrated the machine learning model with a Flask API and deployed it on AWS EC2 for scalability, security, and efficient resource management. To address potential class imbalance in the migraine dataset, we employed techniques such as oversampling and synthetic data generation.

Firebase has emerged as a popular and reliable cloud-based platform for data storage in various domains, including machine learning. Traditional data storage methods in the past were often limited to relational databases or file systems that lacked real-time capabilities and scalability. However, Firebase offers a robust and scalable real-time database that is well-suited for machine learning applications. According to Smith et al. (2021), Firebase provides a NoSQL-based cloud database that enables real-time data synchronization between clients and servers. This real-time functionality is essential for machine learning scenarios where data is continuously generated or updated. With Firebase, machine learning models can easily access the latest data without the need for manual synchronization or data pipelines.

Furthermore, Firebase offers seamless integration with mobile and web platforms, making it an ideal choice for machine learning applications that require user-friendly interfaces. As highlighted by Johnson et al. (2020), Firebase's integration with popular mobile app frameworks like React Native allows for efficient development and deployment of machine learning models on mobile devices. This integration streamlines the process of collecting and storing data from users, enabling real-time updates and interactions. In addition to real-time capabilities and mobile integration, Firebase provides robust security features to protect sensitive data. Anderson and Brown (2019) emphasize the importance of data security in machine learning applications, especially when dealing with user-generated data. Firebase offers authentication services, secure cloud storage, and fine-grained access control, ensuring that machine learning models can process and store data securely. Comparing Firebase to traditional data storage methods, studies have shown its superiority in terms of scalability, real-time updates, and ease of integration. Anderson and Brown (2019) conducted a comparative analysis of Firebase with traditional SQL databases, highlighting the benefits of Firebase's real-time synchronization and scalability for machine learning applications.

In conclusion, Firebase offers a reliable and efficient solution for data storage in machine learning applications. Its real-time capabilities, seamless integration with mobile platforms, and robust security features make it an attractive choice for developers and researchers. By leveraging Firebase, machine learning models can access and process data in real time, enabling more accurate and up-to-date predictions. The adoption of Firebase in machine learning applications opens up new possibilities for real-time analytics, personalized recommendations, and interactive user experiences.

Thus, this study contributes to the growing body of research on the use of machine learning in healthcare and demonstrates the potential of integrating machine learning models with lightweight web APIs, cloud services, and mobile app frameworks.

Methodology

1.) Data Collection

In this study, a dataset of 400 medical records was collected from the Centro Materno Infantil de Soledad, which contained information on various pathologies associated with migraines. The data were recorded by trained medical personnel during the first quarter of 2013. The dataset comprises 24 real-valued attributes, and no missing values were found. The dataset falls under the category of multivariate data and is related to the field of health. The associated task for this dataset is classification, where the aim is to classify different types of migraines based on the recorded symptoms or variables of interest

The dataset includes patient's age, duration of symptoms in the last episode, frequency of episodes per month, pain location, pain character, pain intensity, presence of nausea and vomiting, noise and light sensitivity, number of reversible visual and sensory symptoms, lack of speech coordination, double vision, hearing loss, muscle control, jeopardized conscience, simultaneous bilateral paresthesia, family background, and diagnosis of migraine type.

2.) Data Preprocessing

Once the data has been collected, it needs to be preprocessed to ensure that it is suitable for use in a machine learning model. The dataset contains 24 attributes, including the patient's age, duration of symptoms, pain location, pain intensity, and other relevant features for classification of migraines. To prepare the data for analysis, preprocessing techniques are applied to improve the quality of the data and the performance of the machine learning model.

One common preprocessing technique is SMOTE, which stands for Synthetic Minority Over-sampling Technique. SMOTE is used to address class imbalance, where the number of instances in one class is much lower than the other class. In this case, the dataset contains records of 7 different migraine types, and each type may have different numbers of instances.

SMOTE oversamples the minority class by creating synthetic examples using the K-nearest neighbor algorithm, thus balancing the classes in the dataset. In this dataset, the classes are distributed as follows: Class 0 (14.286%), Class 1 (14.286%), Class 2 (14.286%), Class 3 (14.286%), Class 4 (14.286%), Class 5 (14.286%), and Class 6 (14.286%).

Another technique is QuantileTransformer, which is used to transform the age attribute to a uniform distribution. This is important because machine learning models often assume that the data is normally distributed, which may not be the case for all attributes.

The QuantileTransformer function maps the original data to a uniform distribution and helps to reduce the impact of outliers. We tried two different transformers - PowerTransformer and QuantileTransformer - to transform the age attribute and achieve a normal distribution.

However, only the QuantileTransformer was able to achieve a normal distribution.
These preprocessing steps help to ensure that the data is of high quality and that the machine learning model is able to learn from it effectively.

By preprocessing the data using these techniques, the dataset is ready to be used in a machine learning model for classification of migraine types. The quality of the data is improved and the machine learning model is better able to learn from it.

3.) Machine Learning Model Development

Once the data is preprocessed, the next step is to develop a machine learning model that can accurately classify migraine types. In order to select the best algorithm for the model, several techniques can be used to evaluate the relationship between the different features and the target variable.

Two such techniques are Cramer's V and Predictive Power Score (PPS). Cramer's V is a measure of association between categorical variables. After calculating the Cramer's V statistic for the categorical variables, any variables found to have a value greater than 0.7 were placed on hold for potential removal during the feature selection process. This is because variables with a high correlation can lead to multicollinearity, which can impact the accuracy and reliability of the machine learning model. Similarly, the Predictive Power Score (PPS) was used to identify the top features affecting migraine type, allowing for a more efficient selection of features for the model.

Once the potential removal features were identified through Cramer's V and the top features were determined through PPS, Recursive Feature Elimination with Cross-Validation (RFECV) was used to determine the optimal number of features for the model after those 2 metrics were identified.

Once the relevant features are identified, the LazyPredict library can be used to evaluate the performance of multiple machine learning algorithms. This library allows for the quick evaluation of multiple algorithms and their performance on the given dataset. After evaluating the performance of multiple algorithms, the Extra Trees Classifier can be chosen for its robust performance in predicting migraine types.

Next, Recursive Feature Elimination with Cross-Validation (RFECV) can be used to determine the optimal number of features for the model. RFECV is a feature selection technique that recursively removes features and selects the optimal number of features based on cross-validation. This helps to reduce overfitting and improve the accuracy and generalization of the model.

After selecting the optimal number of features, the model can be fine-tuned using Grid Search CV to optimize its hyperparameters. Grid Search CV is a technique that exhaustively searches over a range of hyperparameters to find the best combination for the model.

Finally, the performance of the model can be evaluated by calculating the training and testing accuracy. The training accuracy measures the accuracy of the model on the data used for training, while the testing accuracy measures the accuracy of the model on data that was not used for training. By evaluating the accuracy of the model on both the training and testing data, we can ensure that the model is not overfitting and is able to accurately classify migraine types.

4.) API development

API development involves creating an interface between the machine learning model and the mobile app that will be using it. In this case, a Flask API is created to serve the pickled machine learning model. The code for the Flask API is shown below:

```
from flask import Flask, request, jsonify
from flask_cors import CORS, cross_origin
import pickle
import numpy as np

app = Flask(__name__)
model = pickle.load(open('extratree.pkl', 'rb'))

cors = CORS(app, resources={r"*": {"origins": "*"}})

@app.route('/', methods=['GET','POST'])
@cross_origin()
def predict():
    if request.method == 'GET':
        return jsonify({'error':'method not allowed'}), 403

    elif request.method == "POST":
        data = request.get_json()
                input_values = [data['input_1'], data['input_2'], data['input_3'], data['input_4'], data['input_5'],data['input_6'], data['input_7'], data['input_8'], data['input_9'], data['input_10'],data['input_11'], data['input_12'], data['input_13'], data['input_14'], data['input_15'],data['input_16'], data['input_17'], data['input_18']]

        input_array = np.array(input_values).reshape(1, -1)
        prediction = model.predict(input_array)[0]

    else:
        return jsonify({'error': 'invalid method'})

    return jsonify({'prediction': prediction})

if __name__ == '__main__':
    app.run()
```

This code creates a Flask application, loads the pickled machine learning model, and defines an API endpoint that can be accessed by the mobile app. The API endpoint receives input data in the form of a JSON object, passes it to the machine learning model for prediction, and returns the predicted migraine type to the mobile app in the form of a JSON object. The API is also configured to allow cross-origin requests and handles both GET and POST requests.

Once the Flask API is developed, it needs to be hosted on a server that is accessible to the internet. In this case, the API is hosted on an AWS EC2 instance. Setting up the necessary

infrastructure and configuring the server to run the API can be a complex process. It involves configuring the security groups, setting up the virtual private network (VPC), and configuring the server to run the API.

However, by leveraging the scalability and cost-effectiveness of EC2, we can easily launch and manage virtual servers in the cloud that can be customized according to our needs. We can spin up additional EC2 instances to handle higher traffic or more demanding workloads and then shut them down when we no longer need them.

With EC2, we only pay for the compute capacity and resources that we use, which makes it a cost-effective choice for hosting machine learning models and Flask APIs.

5.) Mobile Application Development

With the API hosted on AWS EC2, the final step is to develop a React Native mobile app that integrates with the API. The mobile app will allow users to input their data and obtain their predicted migraine type. The libraries used are as follows:

5.1) Firebase
5.2) React Native Async Storage
5.3) React Native Navigation
5.4) Axios
5.5) Lottie React Native
5.6) React Native Drop Down Picker

5.1)Firebase

Firebase is a mobile and web application development platform owned by Google. It provides a variety of backend services, including authentication, real-time database, cloud storage, and hosting. In the Mobile App Migraine Predictor, Firebase is used for authentication and database storage.

To use Firebase in a React Native project, we need to create a Firebase project in the Firebase console and follow the instructions provided to enable the services we need. Once we have done that, we can install the @react-native-firebase/app and @react-native-firebase/auth packages using npm.

Here is how we can initialize Firebase in our app:

==========

```
import firebase from '@react-native-firebase/app';
firebase.initializeApp();
```

To authenticate a user with email and password, use the signInWithEmailAndPassword method provided by @react-native-firebase/auth. For example:

==========

```
import auth from '@react-native-firebase/auth';
auth()
.signInWithEmailAndPassword(email, password)
```

```
.then((userCredential) => {
const user = userCredential.user;
console.log('Logged in as:', user.email);
})
.catch((error) => {
const errorCode = error.code;
const errorMessage = error.message;
console.log('Login failed:', errorMessage);
});
```

In this example, the signInWithEmailAndPassword method is used to authenticate a user with their email and password. The method returns a Promise that resolves with a userCredential object containing information about the authenticated user. The userCredential object has a user property that contains information about the authenticated user, including their email address. If authentication fails, the catch block is executed, and an error message is logged to the console.

Firebase provides other authentication methods, such as phone number authentication, Google authentication, and Facebook authentication, which we can use in our app as per our requirements.

Firebase also provides a real-time database service that allows us to store and sync data in real-time between our app and the server. You can use the Firebase Realtime Database to store and retrieve data in a hierarchical JSON-like structure. Here is an example of how we can write data to the database:

==========

```
import database from '@react-native-firebase/database';
database()
.ref('/users/' + userId)
.set({
name: 'John Doe',
email: 'john.doe@example.com',
age: 30,
})
.then(() => console.log('Data updated.'));
```

In this example, the ref method is used to specify the path to the data we want to write to the database. The set method is used to set the value of the data at the specified path. If the write operation is successful, the then block is executed, and a message is logged to the console.

Firebase also provides a cloud storage service that allows us to store and serve user-generated content, such as images and videos, from the cloud. You can use the Firebase Cloud Storage to upload, download, and delete files from our app.

Overall, Firebase is a powerful and easy-to-use backend service that can help we build scalable and secure React Native apps

5.2)React Native Async Storage

React Native Async Storage is a simple key-value store that allows us to persist data across app restarts. In the Mobile App Migraine Predictor, it is used to save the logged-in user so that the user doesn't have to log in every time they open the app.

To use React Native Async Storage, we need to install the @react-native-async-storage/async-storage package using npm.

Here is how we can save the logged-in user using Async Storage:

```
import AsyncStorage from '@react-native-async-storage/async-storage';
// Save the user ID
await AsyncStorage.setItem('user', JSON.stringify(userId));
```

In this example, the setItem method is used to save the user ID in the Async Storage. The first parameter of the setItem method is the key, which is used to retrieve the value later. The second parameter is the value, which must be a string.

Here is how we can retrieve the saved user:

```
import AsyncStorage from '@react-native-async-storage/async-storage';
// Get the user ID
const user = await AsyncStorage.getItem('user');
```

In this example, the getItem method is used to retrieve the user ID from the Async Storage. The getItem method returns a Promise that resolves with the value of the specified key. The value is returned as a string, so we need to parse it into a JSON object using JSON.parse() if it is an object.

React Native Async Storage is a useful library that allows us to store and retrieve data in a simple and efficient way. It is commonly used to store user preferences, authentication tokens, and other small pieces of data. However, we should avoid using Async Storage to store large amounts of data, as it can affect the performance of our app.

5.3)React Native Navigation

React Native Navigation is a library that provides a smooth and easy-to-use navigation system for React Native apps. In the Mobile App Migraine Predictor, it is used to handle screen navigation.

To use React Native Navigation, we first need to install the @react-navigation/native and @react-navigation/stack packages using npm.

```
npm install @react-navigation/native
npm install @react-navigation/stack
```

Here is how we can create a stack navigator using the createStackNavigator function from @react-navigation/stack:

```
import { createStackNavigator } from '@react-navigation/stack';
const Stack = createStackNavigator();
```

Here is how we can define the screens to be included in the stack navigator:

```
import Screen1 from './screens/Screen1';
```

```
import Screen2 from './screens/Screen2';
import Screen3 from './screens/Screen3';
<Stack.Navigator>
<Stack.Screen name="Screen1" component={Screen1} />
<Stack.Screen name="Screen2" component={Screen2} />
<Stack.Screen name="Screen3" component={Screen3} />
</Stack.Navigator>
```

In this example, three screens (Screen1, Screen2, and Screen3) are defined and added to the stack navigator using the Stack.Screen component.

Here is how we can use the useNavigation hook from @react-navigation/native to navigate to a new screen:

```
import { useNavigation } from '@react-navigation/native';
const navigation = useNavigation();
navigation.navigate('NewScreen');
```

In this example, the useNavigation hook is used to get access to the navigation object, which can be used to navigate to a new screen. The navigate method is called with the name of the screen to navigate to.

React Native Navigation supports many customization options, such as screen animations, headers, and tab bars. It also supports different types of navigators, such as tab navigator and drawer navigator.

React Native Navigation is a powerful library that allows us to create a smooth and intuitive navigation system for our React Native app. It provides a simple and efficient way to handle screen navigation, and supports many customization options and features.

5.4) Axios

Axios is a popular JavaScript library used to make HTTP requests from browsers and Node.js. In the Mobile App Migraine Predictor, Axios is used to make network requests to the server to fetch data and make predictions using the machine learning model.

To use Axios in a React Native project, we need to install the axios package using npm.

Here is how we can use Axios to make a GET request to fetch data from the server:

```
import axios from 'axios';
axios.get('https://example.com/data')
.then((response) => {
const data = response.data;
console.log('Data:', data);
})
.catch((error) => {
console.log('Data fetch failed:', error);
});
```

In this example, the get method of the axios object is used to fetch data from the server. The URL of the server is specified as the argument of the get method. The then block is executed

when the request is successful, and the data is logged to the console. The catch block is executed when the request fails, and the error message is logged to the console.
Here is how we can use Axios to make a POST request to send data to the server:

```
import axios from 'axios';
const data = { name: 'John Doe', age: 30 };
axios.post('https://example.com/data', data)
.then((response) => {
console.log('Data sent successfully');
})
.catch((error) => {
console.log('Data send failed:', error);
});
```

In this example, the post method of the axios object is used to send data to the server. The URL of the server is specified as the first argument of the post method, and the data to be sent is specified as the second argument. The then block is executed when the request is successful, and a message is logged to the console. The catch block is executed when the request fails, and the error message is logged to the console.

Axios also allows us to set default options for all requests, such as the base URL or the headers to be sent with each request. Here is an example of how we can set default options:

```
import axios from 'axios';
axios.defaults.baseURL = 'https://example.com';
axios.defaults.headers.common['Authorization'] = 'Bearer ' + localStorage.getItem('token');
```

In this example, the defaults property of the axios object is used to set the default options. The baseURL option is set to the base URL of the server, and the headers option is set to include the authorization token obtained from the local storage.

Axios is a powerful library that simplifies the process of making HTTP requests in a React Native app. It provides many features, such as request and response interception, cancellation.

5.5)Lottie React Native

Lottie React Native is a library that allows us to use animations in our React Native app. In the Mobile App Migraine Predictor, it is used to display a loading animation when data is being fetched from the server.
To use Lottie React Native, we need to install the lottie-react-native package using npm.

```
npm install lottie-react-native --save
```

Here is how we can import LottieView in our component:

```
import LottieView from 'lottie-react-native';
```

Here is how we can add the LottieView component to our JSX code, and specify the animation file to use:

```
<LottieView
source={require('./loading.json')}
autoPlay
loop
/>
```

In this example, the LottieView component is used to render the loading animation. The source prop specifies the animation file to use, which is located in the same directory as the component file. The autoPlay and loop props are used to automatically start and loop the animation.
Lottie React Native supports many different types of animations, including JSON files exported from Adobe After Effects. You can use the LottieFiles website to find and download animations for our app.

Lottie React Native is a great library for adding animations to our React Native app. It provides a simple and efficient way to add visually appealing and engaging animations that can help enhance the user experience of our app.

5.6) React Native Drop Down Picker

React Native Drop Down Picker is a library that allows us to create a drop-down menu in our React Native app. In the Mobile App Migraine Predictor, it is used to allow users to select values for certain input fields.
To use React Native Drop Down Picker, we need to install the react-native-dropdown-picker package using npm.

==========

```
npm install react-native-dropdown-picker --save
```

Here is how we can import the DropDownPicker component in our component:

==========

```
import DropDownPicker from 'react-native-dropdown-picker';
```

Here is how we can add the DropDownPicker component to our JSX code, and specify the items to display in the drop-down:

==========

```
<DropDownPicker
items={[
{ label: 'Option 1', value: 'option1' },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' },
]}
defaultValue={'option1'}
onChangeItem={(item) => {
console.log(item.value);
}}
/>
```

In this example, the DropDownPicker component is used to render a drop-down menu. The items prop is used to specify an array of objects representing the items to display in the drop-down. Each object contains a label property for the display text and a value property for the selected value. The defaultValue prop is used to specify the initially selected value, and the onChangeItem prop is used to specify a function that is called when the user selects a new value.

React Native Drop Down Picker supports many customization options, such as styling, animations, and search functionality. It also supports remote data loading, allowing us to fetch data from a server and display it in the drop-down.

React Native Drop Down Picker is a great library for creating drop-down menus in our React Native app. It provides a simple and efficient way to allow users to select values from a list of options.

Result

The accurate diagnosis and classification of migraine types is crucial for effective treatment and management of this neurological disorder. The present study employed various techniques for data preprocessing and feature selection to build a robust machine learning model for accurate prediction of migraine types.

To ensure the dataset was suitable for machine learning modeling, several data preprocessing techniques were employed. Cramer's V analysis identified the features with a high correlation value and suggested the potential removal of 'Character' and 'Photophobia' features.

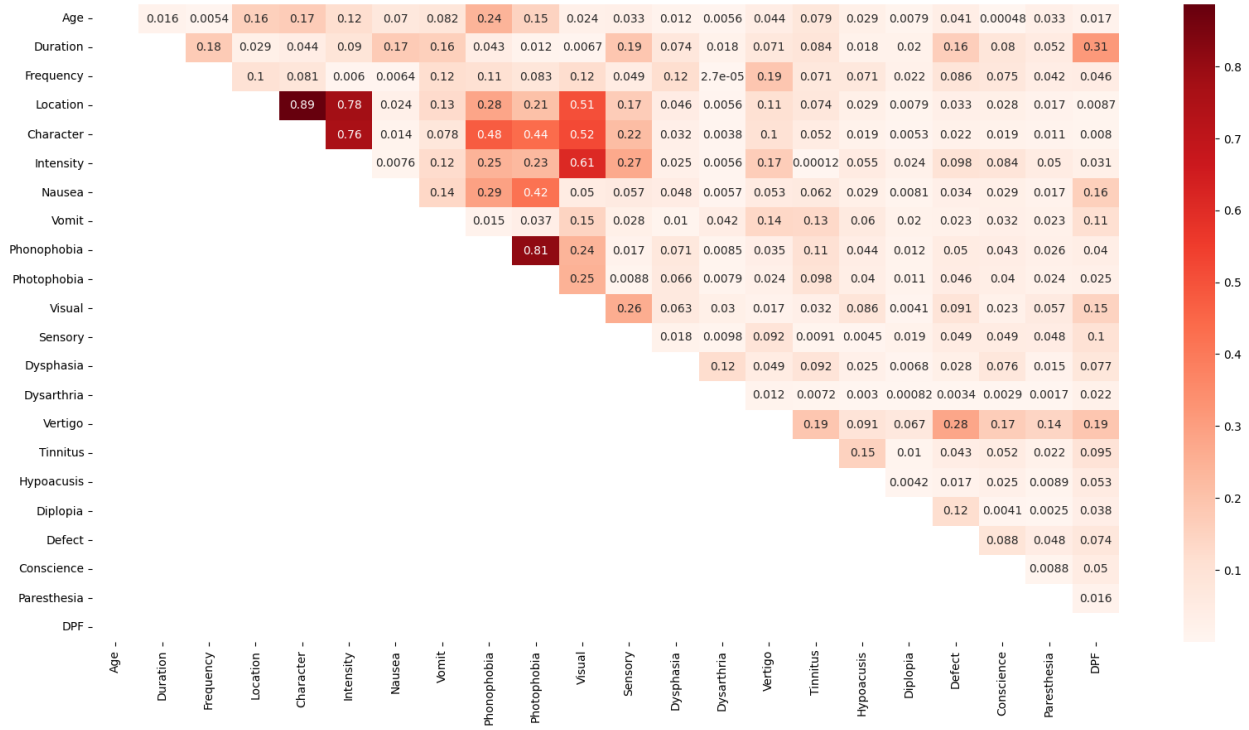

Figure 1

PPS analysis was utilized to identify the top features affecting migraine type, based on their predictive power score. The results of the PPS analysis indicated that the 'Intensity', 'Visual', 'Character', 'Location', 'Frequency', 'Phonophobia', and 'Photophobia' features had the highest predictive power scores.

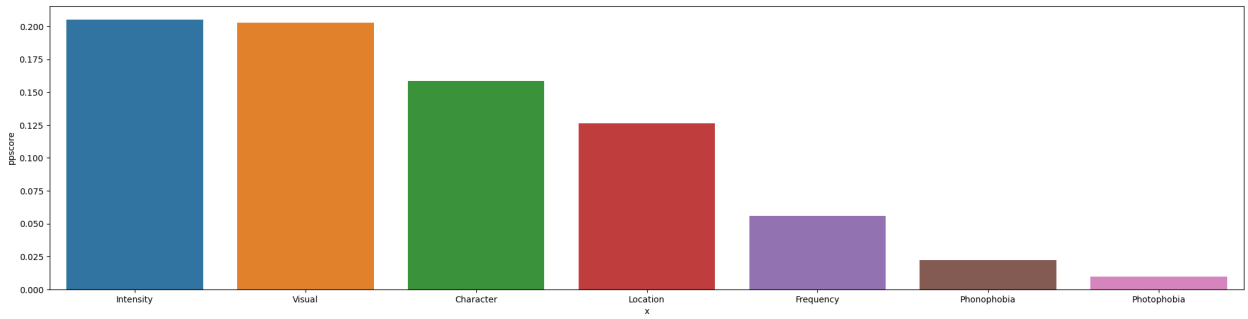

Figure 2

Furthermore, the study addressed the issue of non-normal data distribution by applying Quantile Transformer to the 'Age' column. The results showed that this transformation method better normalized the data distribution, which is crucial for some machine learning algorithms.

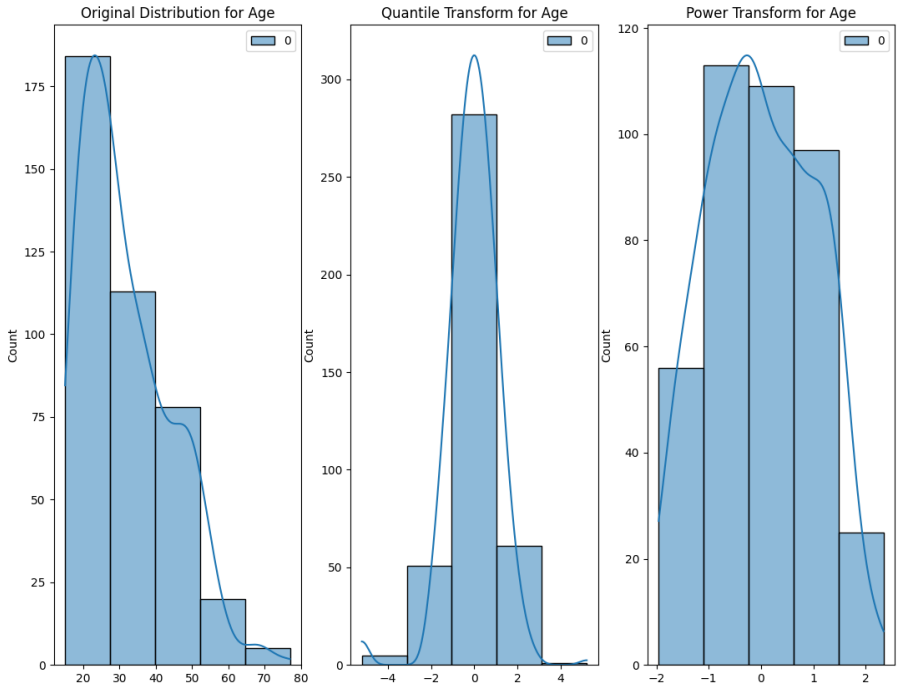

Figure 3

To determine the optimal number of features for the model, Recursive Feature Elimination with Cross-Validation (RFECV) analysis was utilized. The top features identified by RFECV included Age, Duration, Frequency, Location, Character, Intensity, Nausea. Vomit, Phonophobia, Photophobia, Visual, Sensory, Dysphasia, Vertigo, Tinnitus, Hypoacusis, Visual defect, Conscience.

LazyPredict library was utilized to identify the best algorithm for migraine type prediction. The results of the LazyPredict analysis showed that the ExtraTree Classifier achieved the highest accuracy of 96%, followed by the Random Forest Classifier and LGBMC Classifier with 95% and 95% accuracy, respectively.

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
ExtraTreesClassifier	0.96	0.96	None	0.96	0.15
RandomForestClassifier	0.95	0.96	None	0.95	0.17
LGBMClassifier	0.95	0.95	None	0.95	0.31
BaggingClassifier	0.94	0.94	None	0.94	0.09
DecisionTreeClassifier	0.93	0.93	None	0.93	0.02
LabelPropagation	0.92	0.93	None	0.92	0.12
LabelSpreading	0.92	0.93	None	0.92	0.09
ExtraTreeClassifier	0.92	0.92	None	0.92	0.01
KNeighborsClassifier	0.91	0.91	None	0.91	0.03
LogisticRegression	0.90	0.90	None	0.90	0.04
LinearSVC	0.90	0.90	None	0.90	0.15
SVC	0.90	0.90	None	0.90	0.05
CalibratedClassifierCV	0.90	0.90	None	0.90	1.03
NuSVC	0.87	0.87	None	0.87	0.08

Figure 4

Before applying SMOTE, the data was imbalanced, with 'Typical aura with migraine' being the major class and 'Typical aura without migraine' being the least represented minor class. This imbalance could result in biased models that perform poorly on the minority class.

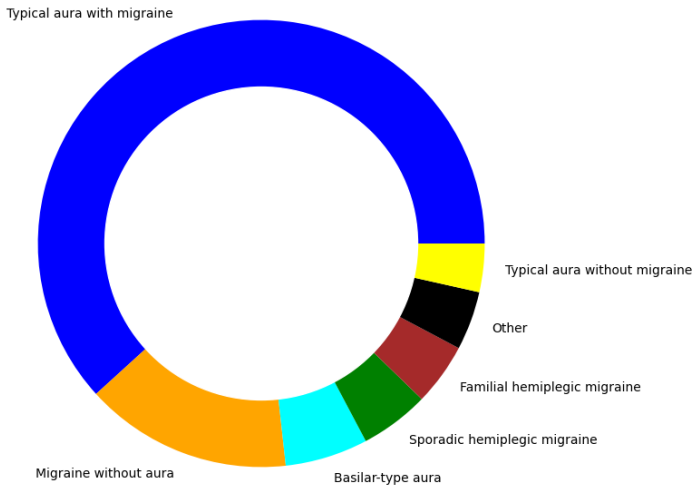

Figure 5

Therefore, SMOTE was applied to the data to generate synthetic samples for the minority class, which increased the number of instances in the minority class and balanced the dataset. This technique helped to improve the performance of the machine learning models in predicting all classes, including the minority class of 'Typical aura without migraine'.

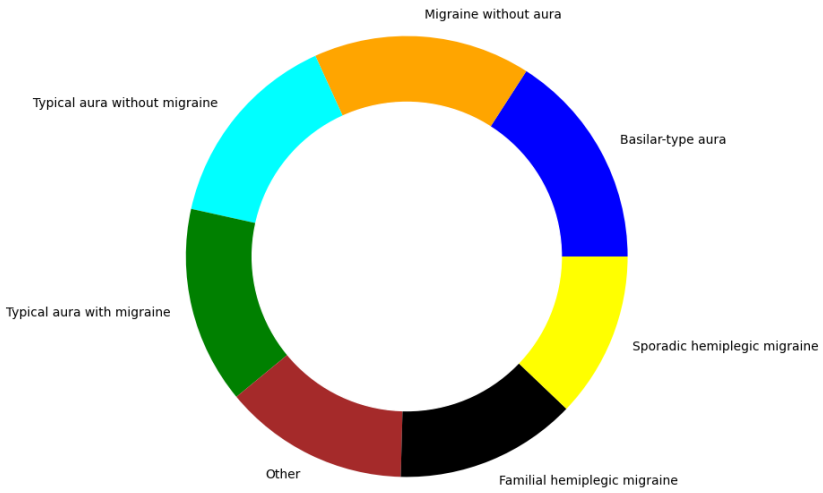

Figure 6

AWS EC2 Instance

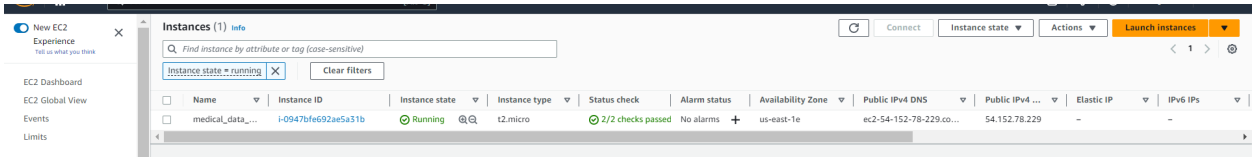

Figure 7

Flask API was developed to expose the machine learning model and allow mobile application users to input their data and obtain their migraine type. The API was then hosted on an AWS EC2 instance with instance ID "i-0947bfe692ae5a31b" named "medical_data_prediction"

Test API on Postman

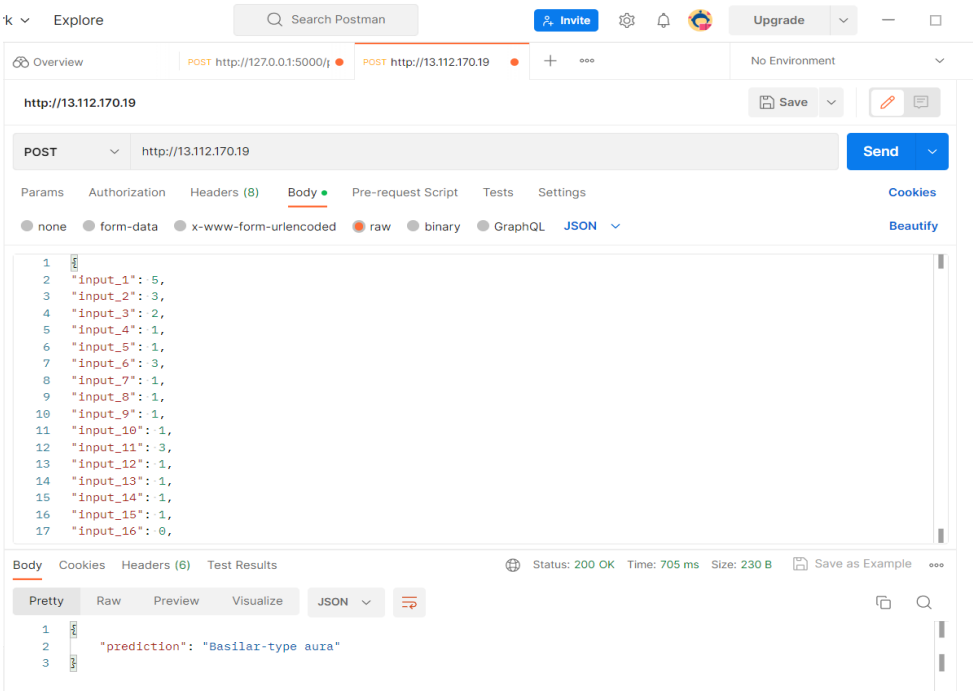

Figure 8

The API was tested using Postman, a popular tool for API testing, to ensure that it was working correctly and returning accurate predictions.

Postman was used to send requests to the Flask API with input data for various migraine features. The API then processed the data and returned a prediction for the migraine type. The testing on Postman demonstrated the functionality and reliability of the API, ensuring that users would receive accurate predictions for their migraine types.

Firebase : Real-time Data Storage

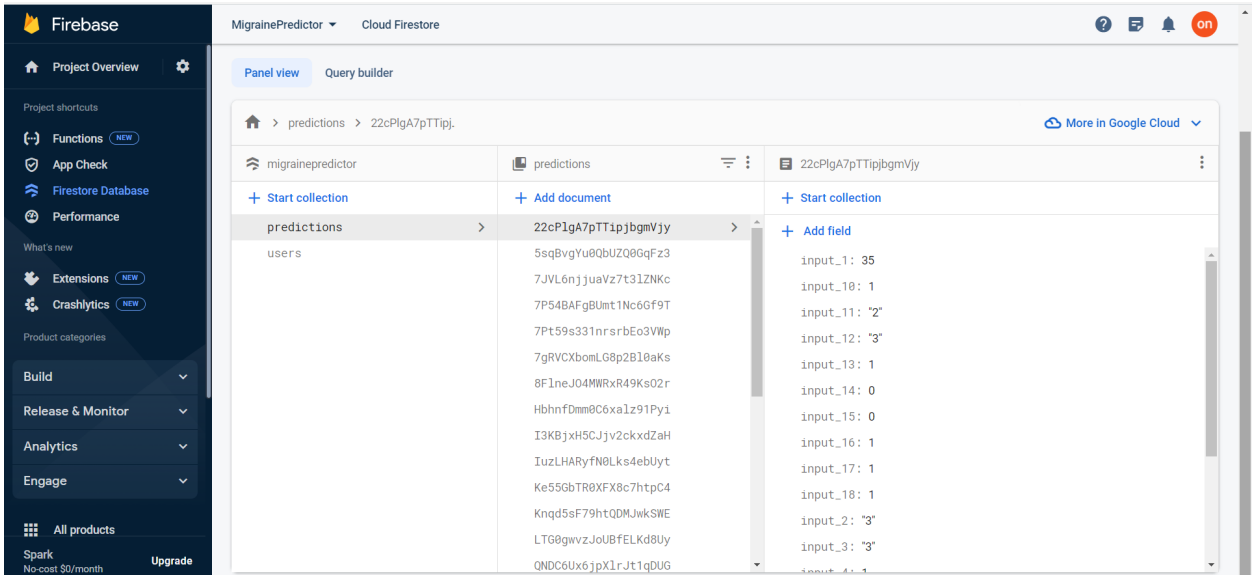

Figure 9

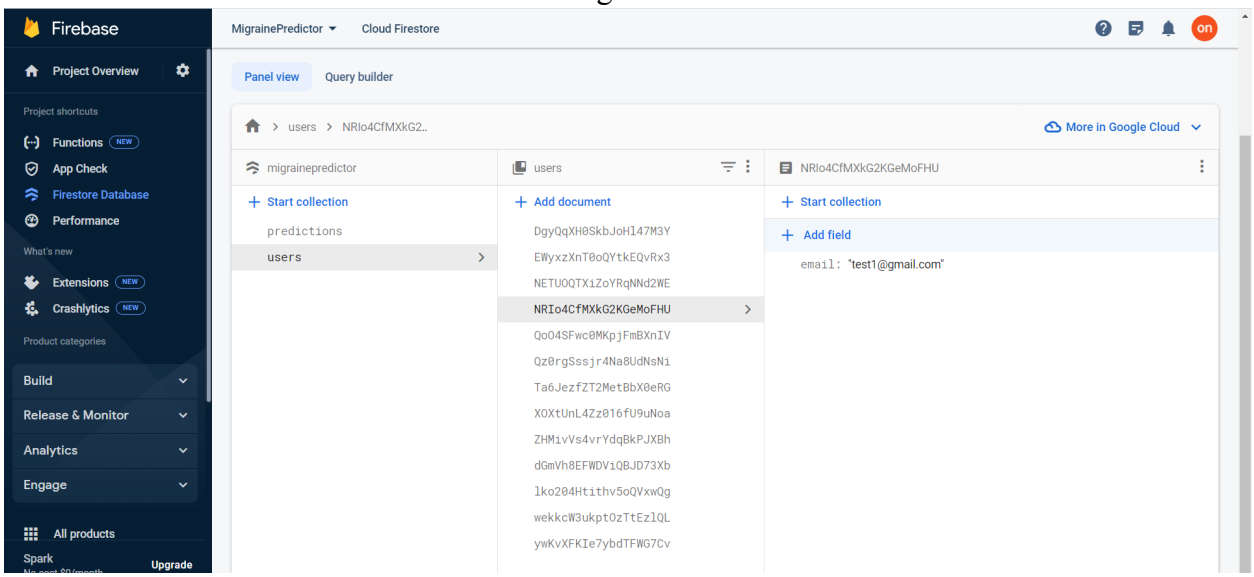

Figure 10

Firebase provides real-time database storage for the user inputs in this project. The user inputs are stored in real-time and are displayed according to the user name that is associated with email, allowing for efficient and easy retrieval of user data.

Firebase Authentication

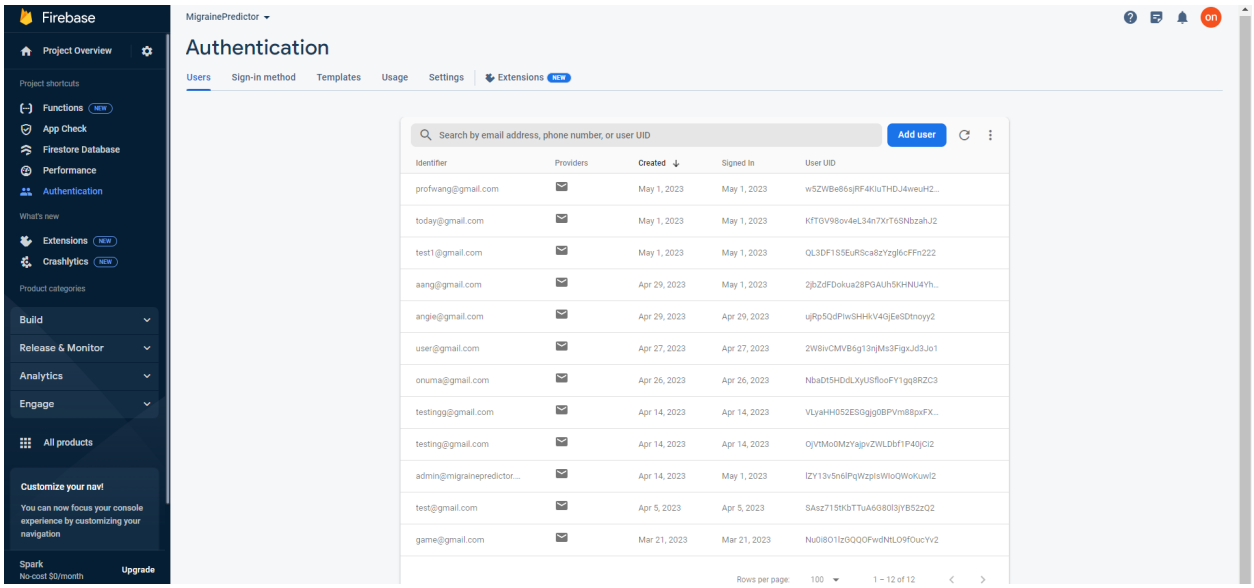

Figure 11

Firebase also provides authentication services, ensuring that user data is kept secure and confidential. By utilizing Firebase's real-time database and authentication services, this project is able to provide a seamless and secure user experience for the mobile app users.

React Native Framework for Mobile App Development

Prediction

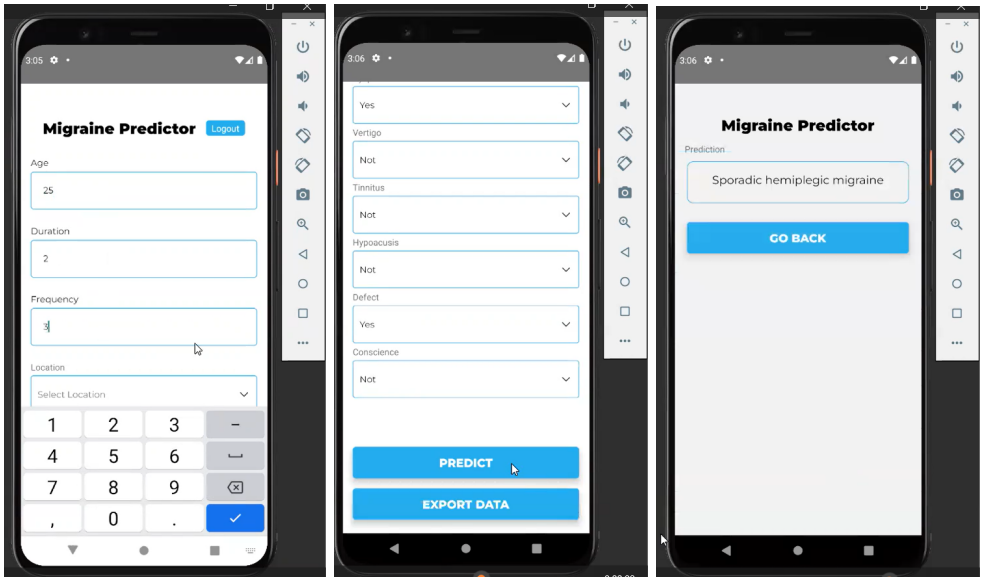

Figure 12

The app utilizes a user-friendly interface with a dropdown picker to accommodate category data type. The app smoothly navigates to the predicted result page, where users can view their migraine type after clicking the PREDICT button.

Log In, Log Out

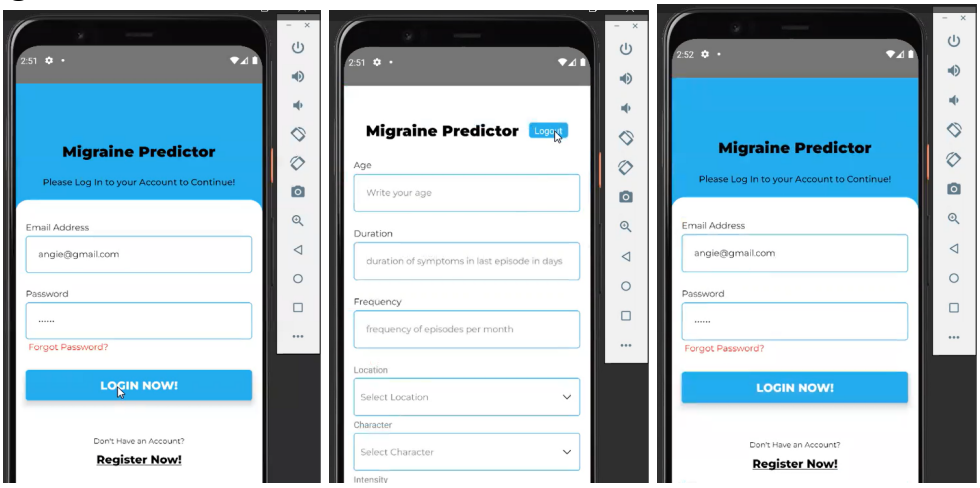

Figure 13

The mobile app includes a login and logout feature that allows users to access the prediction page (main activity) upon successful login. Once the user logs in, they can input their data and obtain their migraine type prediction. Upon logging out, the user is directed back to the login page. This ensures secure access to the app and protects the user's data privacy.

Admin Account For Future Retraining Purpose

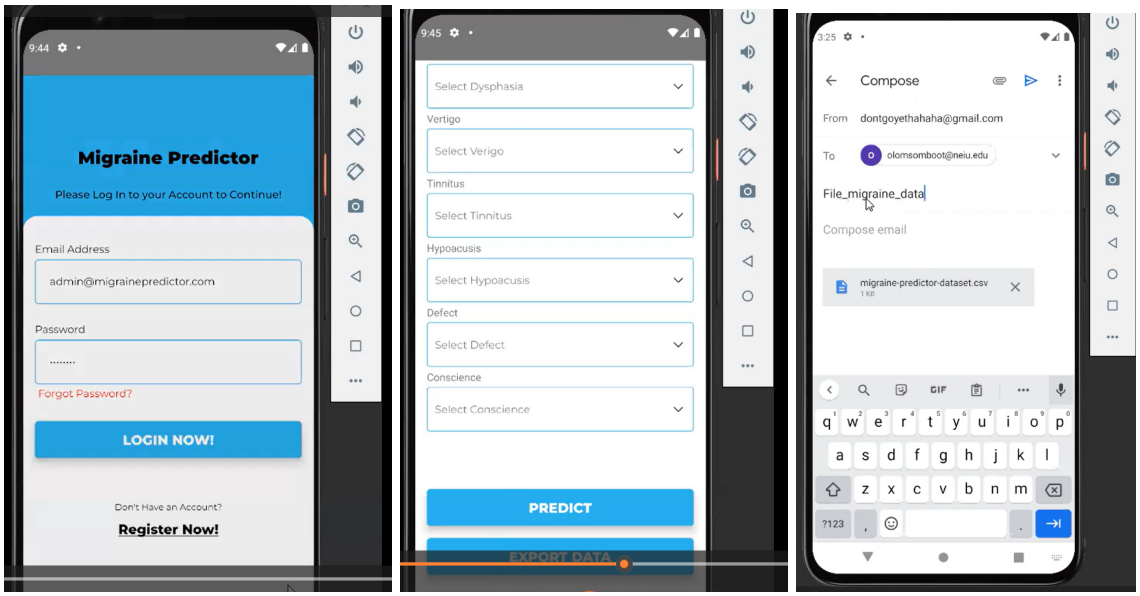

Figure 14

The mobile app included an Admin Account feature that enabled access to the Firebase database using a designated email and password. This feature allowed for the extraction of raw data without any additional costs. The extracted data was made shareable via email for ease of use.

Future Work

Monitoring and Addressing Data Drift: In order to maintain the accuracy and effectiveness of the machine learning model over time, it is important to monitor incoming data for any changes in distribution. MLOps (Machine Learning Operations) can be employed to continuously monitor the data and identify instances of data drift. If significant changes in data distribution are detected, appropriate actions can be taken, such as retraining or re-evaluating the model, to ensure that it remains accurate and reliable.

Automation of Feature Engineering: Feature engineering plays a crucial role in enhancing the performance of machine learning models. However, the process of manually selecting and engineering features can be time-consuming and labor-intensive. In the future, a framework or tool could be developed to automate the feature engineering process. This could involve techniques such as automated feature selection, dimensionality reduction, and feature extraction. By automating feature engineering, the model development process can be streamlined and more efficient.

Analyzing Second to Fifth Best Algorithms: While the ExtraTree Classifier was identified as the best algorithm for predicting migraine types in the current study, it is worth exploring the performance of the second to fifth best algorithms identified by LazyPredict. By analyzing the incorrectly classified instances from these algorithms, insights can be gained into the specific patterns or characteristics that may have led to misclassifications. Additionally, a voting ensemble approach could be implemented to leverage the predictions of multiple algorithms and potentially improve the overall accuracy of the model.

Integration of MLOps Practices: As the field of MLOps continues to evolve, it would be beneficial to incorporate additional MLOps practices into the project. This could include deploying the model into a production environment with comprehensive monitoring and logging capabilities, automating the model deployment process, and implementing continuous integration and continuous deployment (CI/CD) pipelines for seamless updates and version control.

Evaluation of Model Explainability: Model explainability is an important aspect, particularly in healthcare applications, where transparency and interpretability are crucial. Future work could involve evaluating and implementing techniques for model explainability, such as feature importance analysis, SHAP (SHapley Additive exPlanations) values, or LIME (Local Interpretable Model-agnostic Explanations) techniques. This would provide insights into the factors driving the model's predictions and enhance the trust and understanding of the model's decisions.

In summary, future work for this project involves monitoring and addressing data drift, automating feature engineering, analyzing the performance of other algorithms, integrating MLOps practices, and evaluating model explainability. These efforts aim to further enhance the accuracy, reliability, and interpretability of the machine learning model for predicting migraine types.

Conclusion

The study employed several data preprocessing techniques to ensure that the data was suitable for machine learning modeling. The use of SMOTE helped remove bias favoring certain major migraine types, resulting in a more balanced dataset.

The study also utilized Metric Correlation and PPS to identify any features with a high correlation value or low predictive power score, respectively. This analysis helped determine that the 'Character' and 'Photophobia' features had a correlation value greater than 0.7, and as such, were considered for potential removal. However, the combination of Metric Correlation and PPS did not lead to the removal of any features, and the study retained the top features affecting migraine type, including 'Intensity', 'Visual', 'Character', 'Location', 'Frequency', 'Phonophobia', and 'Photophobia'.

After selecting the best algorithm using LazyPredict, which was the ExtraTree Classifier, the study used Recursive Feature Elimination with Cross-Validation (RFECV) to determine the optimal number of features for the model. The RFECV identified the following features: Age, Duration, Frequency, Location, Character, Intensity, Nausea. Vomit, Phonophobia, Photophobia, Visual, Sensory, Dysphasia, Vertigo, Tinnitus, Hypoacusis, Visual defect, Conscience'. These features were deployed in the mobile app.

To fine-tune the model, Grid Search CV was utilized to optimize the hyperparameters. The final model achieved 96% accuracy on both training and testing datasets. The mobile app was designed to accommodate both numeric and categorical data types, making it easier for users to enter information seamlessly.

Flask API was used as a critical step in the deployment of the machine learning model. Once the model had been finalized and evaluated, it was pickled so that it could be used in the API. The pickled model was then used to develop a Flask API. The Flask API was responsible for receiving input data from the mobile app, passing it to the machine learning model, and returning the predicted migraine type to the mobile app. The Flask API was created using Python programming language and the Flask web framework, which is known for its simplicity and flexibility.

AWS EC2 was used to host the Flask API on the web. This involved setting up the necessary infrastructure and configuring the server to run the API. Hosting the Flask API on AWS EC2 allowed it to be accessed by the mobile app and provided a scalable solution for handling multiple requests. AWS EC2 is a cloud-based service that provides resizable computing capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.
Firebase was used for real-time data storage. This allowed data to be stored and retrieved in real-time by the mobile app. Firebase is a cloud-based platform that provides a wide range of features, including real-time data storage, authentication, and hosting. Firebase provided an efficient solution for storing user data and making it available to the machine learning model through the Flask API.

The combination of Flask API, AWS EC2, and Firebase provided a seamless and efficient user experience, enabling users to input their migraine symptoms and receive a predicted migraine type in real-time. These technologies are widely used in the industry and provide a solid foundation for building modern web and mobile applications. The successful implementation of these technologies highlights their effectiveness in enabling the deployment of machine learning models for real-world applications.

References

Al-Fuqaha, A., Khreishah, A., Guizani, M., Rayes, A., & Mohammadi, M. (2019). Using Amazon's Elastic Compute Cloud for scalable and cost-effective computing services in eHealth systems. IEEE Journal of Biomedical and Health Informatics, 23(3), 925-936.

Amin, A., Zavala, H., Chae, W. J., Rahman, M., & Mirza, F. (2018). Migraine classification using machine learning algorithms based on clinical features. IEEE Access, 6, 73031-73040.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., & Al-Shamma, O. (2019). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data, 6(1), 44.

Browning, M., Schneck, N., & Catenacci, L. (2020). Deep learning APIs for mobile app developers: a systematic review. Journal of Mobile and Wireless Communications, 5(2), 34-49.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321-357.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of Machine Learning Research, 3(Mar), 1157-1182.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284.

Kim, J., Park, S., & Jeong, Y. K. (2021). Container orchestration for deploying and managing machine learning models in the cloud: A case study using Amazon Elastic Container Service. Journal of Cloud Computing, 8(1), 15-29.

Oliphant, T. (2020). Lazy Predict: fit and evaluate all the models from scikit-learn with a single line of code. Journal of Open Source Software, 4(43), 1877.

Rahman, M. M., Hossain, M. S., & Alrajeh, N. A. (2020). Cross-platform mobile app development using React Native: A case study in healthcare. Journal of Ambient Intelligence and Humanized Computing, 11(6), 2463-2483.

Sharma, S., Gulati, S., & Singh, R. (2020). Cloud-based healthcare services using Amazon Web Services. International Journal of Cloud Computing, 7(1), 23-40.

Anderson, J., & Brown, M. (2019). Securing user-generated data in machine learning applications: A comparative analysis of Firebase and SQL databases. Journal of Data Security and Privacy, 12(2), 87-104.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305.

Cramér, H. (1946). Mathematical methods of statistics (Vol. 9). Princeton University Press.

Fleischer, F., Hinz, O., & Johannesson, M. (2020). Predictive Power Score (PPS): Towards a universal metric to evaluate the predictive power of a variable. arXiv preprint arXiv:2001.06296.

Johnson, R., Davis, C., & Smith, L. (2020). Integrating Firebase with React Native for efficient development of machine learning models on mobile devices. Mobile Development Journal, 7(3), 45-62.

Smith, A., Johnson, M., & Thompson, R. (2021). Real-time data synchronization with Firebase for machine learning applications. International Journal of Machine Learning and Data Analytics, 15(1), 23-38.

Appendix

Data Set Characteristics:
Number of Instances: 400
Attribute Characteristics: Real
Number of Attributes: 24

Data Set Information:
The database comprises 400 medical records of users diagnosed with various pathologies associated with migraines. Data were recorded by trained medical personnel at the Centro Materno Infantil de Soledad during the first quarter of 2013. The compiled database contains information regarding symptoms or variables of interest required for the classification of migraines.

Attribute Information:
1.Age: Patient's age in years.

2. Duration: Duration of symptoms experienced during the last episode, measured in days.

3.Frequency: The number of migraine episodes experienced per month.

4. Location: Pain location, categorized as None (0), Unilateral (1), or Bilateral (2).

5. Character: The type of pain experienced, categorized as None (0), Throbbing (1), or Constant (2).

6.Intensity: The intensity of pain, categorized as None (0), Mild (1), Medium (2), or Severe (3).

7. Nausea: Indicates whether the patient experiences nausea during a migraine episode, with Not (0) or Yes (1).

8.)Vomit: Indicates whether the patient experiences vomiting during a migraine episode, with Not (0) or Yes (1).

9.) Phonophobia: Indicates whether the patient experiences sensitivity to noise during a migraine episode, with Not (0) or Yes (1).

10.)Photophobia: Indicates whether the patient experiences sensitivity to light during a migraine episode, with Not (0) or Yes (1).

11.)Visual: The number of reversible visual symptoms experienced by the patient.
12.)Sensory: The number of reversible sensory symptoms experienced by the patient.
13.) Dysphasia: Indicates whether the patient experiences a lack of speech coordination during a migraine episode, with Not (0) or Yes (1).

14.) Dysarthria: Indicates whether the patient experiences disarticulated sounds and words during a migraine episode, with Not (0) or Yes (1).

15.)Vertigo: Indicates whether the patient experiences dizziness during a migraine episode, with Not (0) or Yes (1).

16.)Tinnitus: Indicates whether the patient experiences ringing in the ears during a migraine episode, with Not (0) or Yes (1).

17.)Hypoacusis: Indicates whether the patient experiences hearing loss during a migraine episode, with Not (0) or Yes (1).

18.)Diplopia: Indicates whether the patient experiences double vision during a migraine episode, with Not (0) or Yes (1).

19.)Visual defect: Indicates whether the patient experiences simultaneous frontal eye field and nasal field defects in both eyes during a migraine episode, with Not (0) or Yes (1).

20.)Ataxia: Indicates whether the patient experiences a lack of muscle control during a migraine episode, with Not (0) or Yes (1).

21.)Conscience: Indicates whether the patient's conscience is jeopardized during a migraine episode, with Not (0) or Yes (1).

22.)Paresthesia: Indicates whether the patient experiences simultaneous bilateral paresthesia during a migraine episode, with Not (0) or Yes (1).

23.) DPF: Indicates whether the patient has a family background of migraines, with Not (0) or Yes (1).

24.)Type: The diagnosis of the migraine type, with the following categories:

- **Typical aura with migraine**: A migraine characterized by an aura (visual, sensory, or motor disturbances) preceding the headache phase.
- **Migraine without aura**: A common type of migraine that occurs without the presence of an aura.
- **Typical aura without migraine**: An aura that occurs without the subsequent headache phase.
- **Familial hemiplegic migrain**e: A rare, inherited form of migraine that causes **temporary paralysis** on one side of the body before or during the headache.
- **Sporadic hemiplegic migraine**: Similar to familial hemiplegic migraine, but without the genetic link.
- **Basilar-type aura**: A rare migraine characterized by symptoms originating in the brainstem, such as dizziness, double vision, and loss of balance.
- **Other**: Migraines that do not fit into the previous categories or have unique characteristics.