BENGALURU CITY UNIVERSITY

ಬೆಂಗಳೂರು ನಗರ ವಿಶ್ವವಿದ್ಯಾಲಯ BENGALURU CITY UNIVERSITY ಆಗು ನೀ ಅನಿಕೇತನ BE BOUNDLESS

A project on

Diabetes Prediction Android Application Using Machine Learning and FastAPI

A mobile-friendly solution for early diabetes risk prediction leveraging a machine learning model and API integration

BACHELOR OF COMPUTER APPLICATION (BCA)

SUBMITTED BY HAMSHINI PREMA KR

(U18UZ22S0004)

DEPARTMENT OF COMPUTER SCIENCE

CENTRAL COLLEGE CAMPUS, DR AMBEDKAR VEEDHI ROAD BENGALURU-560001

ABSTRACT

Diabetes is a common health condition that, if left unchecked, leads to serious complications. This project focuses on building a simple, user-friendly Android application that helps users assess their risk of diabetes based on key health metrics like blood pressure, glucose levels, insulin levels, BMI, and age. By combining machine learning with mobile technology, this app provides a quick way for individuals to gain preliminary insights into their health and potentially take action sooner.

The application uses a machine learning model trained on health data to distinguish between diabetic and non-diabetic cases. This model is hosted on a FastAPI server and connects to the mobile app through an API. To make this API accessible to the Android app, the project uses ngrok, allowing the server to be publicly available. The app itself is created using MIT App Inventor, which enables an intuitive, drag-and-drop design, making it straightforward for users.

This project merges machine learning and mobile development to create a tool that brings health insights closer to users. While currently a prototype, this app paves the way for more accessible healthcare solutions, especially for individuals without easy access to medical facilities. Future updates include additional health metrics, improved model accuracy, and a cloud-based server for continuous availability. This project illustrates how technology makes healthcare more accessible, empowering people to monitor their health independently.

INTRODUCTION

Project Overview

This project introduces a mobile application designed to predict the likelihood of diabetes based on a user's health information. With the growing prevalence of diabetes, particularly in regions with limited healthcare access, this tool provides an accessible way for users to gain insights into their potential health risks. Using machine learning, the application processes health metrics such as glucose levels, blood pressure, insulin levels, BMI, age, and family history.

The project combines several technologies to create an interactive, user-friendly experience. A machine learning model, trained with scikit-learn, serves as the core predictive engine. This model is deployed using FastAPI to create a robust backend API that the Android app accesses. To enable public access, ngrok exposes the server to a unique URL, linking the mobile app and backend server seamlessly. The app itself is built on MIT App Inventor, a visual platform for developing Android applications with drag-and-drop features.

This diabetes prediction app aims to support users in proactive health management by offering a quick, accessible risk assessment tool. The project highlights the potential for machine learning applications in healthcare and showcases the integration of software tools to develop meaningful, user-centered technology.

Objective

This project aims to create an Android app that assesses diabetes risk based on health metrics, enabling users to gain early insights into their health. Using a machine learning model and integrating tools like FastAPI, ngrok, and MIT App Inventor, the app provides a simple, accessible way to promote proactive health management and demonstrate machine learning's potential in healthcare.

Motivation

Diabetes is increasingly common worldwide, especially in areas with limited healthcare access. Early detection is crucial for preventing complications, but regular check-ups and diagnostic tools can be hard to come by. This project aims to address this gap by developing a user-friendly mobile app that offers a quick, preliminary diabetes risk assessment using easily available health metrics. By leveraging machine learning and mobile technology, the app will make vital health insights accessible to more people, promoting better health outcomes and raising awareness about diabetes.

METHODOLOGY

Model Development

The diabetes prediction model for this project is developed using a supervised machine learning approach with scikit-learn. The steps in model development include data preprocessing, model selection, training, and evaluation, ensuring that the model is accurate and reliable for real-world predictions.

1. **Data Collection and Preprocessing**: The model is trained on a dataset containing information about individuals, with features such as the number of pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function, and age. To prepare the data for model training, all features are checked for missing values and scaled appropriately to ensure consistency.
2. **Model Selection**: For this project, after testing multiple algorithms, **Random Forest Classifier** is selected as the final model due to its high accuracy and ability to handle complex relationships within the dataset. Initially, logistic regression and support vector machine (SVM) models were tested; however, while they performed adequately, they did not achieve the level of accuracy needed for reliable predictions in this health-related application.
    - **Random Forest Classifier**: This algorithm is a robust choice for classification problems as it combines multiple decision trees, improving both prediction accuracy and stability. It is less prone to overfitting compared to individual decision trees, making it ideal for

datasets with varied and interdependent features, as seen in this diabetes dataset.

- **Model Performance**: During testing, the Random Forest model demonstrated superior performance, accurately identifying diabetic and non-diabetic cases with minimal error. Hyperparameter tuning further improved its performance, adjusting parameters like the number of trees in the forest to balance accuracy and processing speed.

The final Random Forest model showed the best balance between precision, recall, and overall accuracy, making it the most reliable choice for this project's diabetes prediction application. This model is then saved using pickle to be deployed in the FastAPI backend.

3. **Model Training**: The selected model is trained using 80% of the dataset, leaving 20% for testing. During training, the model learns patterns in the data, identifying relationships between the health metrics and the presence or absence of diabetes. Hyperparameter tuning is applied to optimize model performance.

4. **Model Evaluation**: To ensure accuracy, the model's performance is evaluated using metrics like accuracy, precision, recall, and F1 score. This evaluation helps in understanding how well the model distinguishes between diabetic and non-diabetic cases. After calculating these metrics, the model demonstrates robust performance across all four indicators, with particularly high recall and F1 scores. This indicates that the Random Forest model is not only accurate but also reliable in identifying both diabetic and non-diabetic cases, making it well-suited for the app's diabetes prediction feature.

By following these steps, the model is optimized for accurate diabetes risk prediction, providing a reliable back-end for the mobile application.

API Development

The FastAPI framework is used to develop a RESTful API that enables our diabetes prediction model to communicate with the mobile app, providing a seamless user experience. This API serves as the bridge between the machine learning model and the Android app interface, allowing users to input health metrics and receive predictions in real time.

1. **Setting Up FastAPI**: The API development begins by setting up a FastAPI application, a modern, fast web framework well-suited for building APIs with Python. FastAPI supports asynchronous programming and provides efficient handling of incoming requests, making it an ideal choice for this project, which needs to process input data and return predictions rapidly.
2. **Enabling Cross-Origin Resource Sharing (CORS)**: To allow the Android app to interact with the FastAPI backend without restrictions, CORS (Cross-Origin Resource Sharing) is enabled. Since the mobile app and backend server operate on different domains (especially when using ngrok), configuring CORS is essential to ensure that the frontend app can make requests to the backend API seamlessly.
3. **Defining Input Data Schema**: The API requires a structured input for predicting diabetes, which includes various health metrics such as pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, and age. To ensure structured data input, a schema is defined using Pydantic's BaseModel. This schema enforces the data types of each parameter, helping to prevent invalid data from being submitted.
4. **Loading the Trained Model**: The diabetes prediction model, trained as a Random Forest Classifier and saved using pickle, is loaded in the API. This allows the FastAPI app to access the model's prediction function whenever it receives input from the Android app.
5. **Creating the Prediction Endpoint**: The prediction endpoint is created using the POST method, which allows users to send health data to the API. The endpoint receives the input data, parses it, and prepares it for model prediction. Here's how the endpoint functions:
    - The input data is received as JSON and converted to a dictionary format for easier access to each feature.
    - Each input feature is extracted and arranged in the order expected by the model.
    - The model then processes this list of features and outputs a prediction.
6. **Deploying the API with ngrok**: To make the API accessible on a mobile device, ngrok is used to generate a public URL that exposes the FastAPI server running locally on port 8000. By running the ngrok command and using an authtoken, a secure, temporary public URL is generated. This URL is shared with the MIT App Inventor project, allowing the mobile app to communicate with the FastAPI backend remotely.

7. **Starting the API Server:** To start the API, uvicorn is used, which runs the FastAPI application on port 8000, allowing it to listen for requests. In this notebook environment, nest_asyncio is also applied to avoid any runtime conflicts.

By following these steps, the API is fully set up and operational, allowing the Android application to send user input, receive model predictions, and provide diabetes risk assessments in real-time. This deployment strategy makes use of FastAPI's efficiency, ngrok's secure tunneling capabilities, and MIT App Inventor's user-friendly interface, delivering a comprehensive and accessible diabetes prediction tool.

Mobile App Development (MIT App Inventor)

The mobile app is developed using MIT App Inventor, an accessible, drag-and-drop tool that allows for quick Android app creation without coding. Here's an overview of how each component of the app was designed to interact with the FastAPI model prediction API:

1. **User Interface (UI)**:
    - The UI of the app is designed with simplicity in mind, allowing users to input personal health details like *Pregnancies*, *Glucose*, *Blood Pressure*, *Skin Thickness*, *Insulin Levels*, *BMI*, *Diabetes Pedigree Function*, and *Age*.
    - Each input field is created as a text box, enabling easy data entry, while a single "Predict" button submits the data.
2. **API Integration**:
    - Using MIT App Inventor's Web component, the app connects to the FastAPI model endpoint via the public URL generated by ngrok.
    - When users press the "Predict" button, the app sends a request to the FastAPI server with all the input data in JSON format. The server processes this data and returns a prediction response: either "The person is diabetic" or "The person is not diabetic."
3. **Data Handling**:
    - The app captures user input, formats it as JSON, and sends it to the ngrok URL. The Web component handles both sending the data and receiving the response.

- Once the prediction result is received, it is displayed to the user immediately in a label, making the user experience seamless results.

4. **User Experience (UX)**:
    - By leveraging the simple drag-and-drop functionality of MIT App Inventor, the app is designed to be highly intuitive and requires minimal technical knowledge from the user.
    - The response from the API is instant, providing the user with a quick assessment of their diabetes risk, based on the input values.

Implementation

Backend Code Explanation:

1. **Importing Required Libraries:**
    - Essential libraries like FastAPI for creating the API, pydantic for defining data models, and pickle for loading the machine learning model are imported.
    - uvicorn is used to start the FastAPI server, while pyngrok helps create a public URL for API access. The CORS middleware is also configured to allow the mobile app to communicate with the backend.

The Code:

```
from fastapi import FastAPI

from pydantic import BaseModel

import pickle

import json

import uvicorn

from pyngrok import ngrok

from fastapi.middleware.cors import CORSMiddleware

import nest_asyncio
```

2. **Loading the Machine Learning Model**:
    - The diabetes prediction model is pre-trained and saved in a .sav file. It is loaded into memory using pickle, enabling quick access to predictions.

THE CODE:

```
diabetes_model = pickle.load(open('diabetes_model.sav', 'rb'))
```

3. **Setting Up the FastAPI Application**:
    - An instance of FastAPI is created to define and manage the API. CORS middleware is added, allowing requests from any origin, so the mobile app can connect without issues.

THE CODE:

```
app = FastAPI()

origins = ["*"]

app.add_middleware(

   CORSMiddleware,

   allow_origins=origins,

   allow_credentials=True,

   allow_methods=["*"],

   allow_headers=["*"],

)
```

4. **Defining the Input Model**:
    - A model_input class is created using pydantic. This class specifies the required input fields like Pregnancies, Glucose, BloodPressure, etc., and their data types. This input structure ensures data consistency and makes error handling simpler.

THE CODE:

```
class model_input(BaseModel):
    Pregnancies: int
    Glucose: int
    BloodPressure: int
    SkinThickness: int
    Insulin: int
    BMI: float
    DiabetesPedigreeFunction: float
    Age: int
```

5. **Creating the Prediction Endpoint**:
   - The /diabetes_prediction endpoint is defined to receive user input as JSON, process it, and return the prediction result.
   - Within the function, the input is converted from JSON format to a dictionary, extracting each feature value.
   - The input data is structured into a list format and passed to the predict method of the machine learning model. Based on the output (0 or 1), a message indicating whether the person is diabetic or not is returned.

THE CODE:

```
@app.post('/diabetes_prediction')
def diabetes_predd(input_parameters: model_input):
    input_data = input_parameters.json()
    input_dictionary = json.loads(input_data)


    preg = input_dictionary['Pregnancies']
```

```
glu = input_dictionary['Glucose']
bp = input_dictionary['BloodPressure']
skin = input_dictionary['SkinThickness']
insulin = input_dictionary['Insulin']
bmi = input_dictionary['BMI']
dpf = input_dictionary['DiabetesPedigreeFunction']
age = input_dictionary['Age']

input_list = [preg, glu, bp, skin, insulin, bmi, dpf, age]
prediction = diabetes_model.predict([input_list])

if (prediction[0] == 0):
    return 'The person is not diabetic'
else:
    return 'The person is diabetic'
```

6. **Starting the API Server**:
    - The API server is run locally on port 8000 using uvicorn. Since this runs within a Jupyter Notebook, nest_asyncio is applied to ensure smooth execution without conflicts.

THE CODE:

```
nest_asyncio.apply()
uvicorn.run(app, port=8000)
```

7. **ngrok Integration**:
    - To expose the local server to the internet, ngrok generates a public URL for the API endpoint. This URL is shared with the MIT App Inventor app, allowing it to send requests to the API.

THE CODE:

```
ngrok_tunnel = ngrok.connect(8000)

print('Public URL:', ngrok_tunnel.public_url)
```

In summary, the backend code creates an API that accepts user input, processes it using the trained model, and returns a prediction. Each part is set up to support smooth integration with the mobile app for real-time diabetes risk assessment.

Frontend Explanation

The frontend of this project is an Android application developed using MIT App Inventor.

1. **User Interface Design**:
    - MIT App Inventor is used to create the UI, utilizing a drag-and-drop interface to make it user-friendly and simple to navigate.
    - Input fields are created for each health parameter: *Pregnancies*, *Glucose Level*, *Blood Pressure*, *Skin Thickness*, *Insulin Level*, *BMI*, *Diabetes Pedigree Function*, and *Age*.
    - A button labeled "Predict" is added, which users can tap to send their data to the backend API and receive a prediction.
    - The app also includes a label or text box to display the prediction result, either "The person is diabetic" or "The person is not diabetic," based on the backend response.
2. **Connecting to the Backend API**:
    - The app uses the **Web component** in MIT App Inventor to establish communication with the backend FastAPI server.

- The URL for the API, generated by **ngrok**, is set in the Web component. This allows the app to interact with the FastAPI server, hosted locally but exposed to the internet via ngrok.
- When the "Predict" button is pressed, the app collects the data entered by the user, formats it as JSON, and sends it as a POST request to the backend API endpoint.

3. **Data Preparation**:
    - All input fields are retrieved, and their values are assigned to the JSON structure that matches the backend input model.
    - The Web component in MIT App Inventor formats this data and ensures that the JSON format is correctly structured, so the backend can process it seamlessly.
4. **Receiving and Displaying Predictions**:
    - After the POST request is sent, the Web component receives the prediction response from the API.
    - Based on the response (either 0 for non-diabetic or 1 for diabetic), the app displays the result text in the designated label, providing immediate feedback to the user on their diabetes risk.
5. **User Experience Enhancements**:
    - The app is designed to be intuitive, with clear labels for each input and real-time prediction feedback.
    - By utilizing drag-and-drop design, MIT App Inventor allows the app to be visually appealing and accessible to users without technical knowledge.

Through these steps, the frontend app collects input data, communicates with the backend API to receive predictions, and displays the results.

Results

1. **Model Accuracy and Evaluation:**
    - The machine learning model, trained on diabetes-related health data, demonstrates strong predictive performance in identifying diabetes risk. Model evaluation metrics indicate high accuracy, precision, and recall values, which collectively show the model's effectiveness in

handling test cases and predicting diabetic versus non-diabetic outcomes.

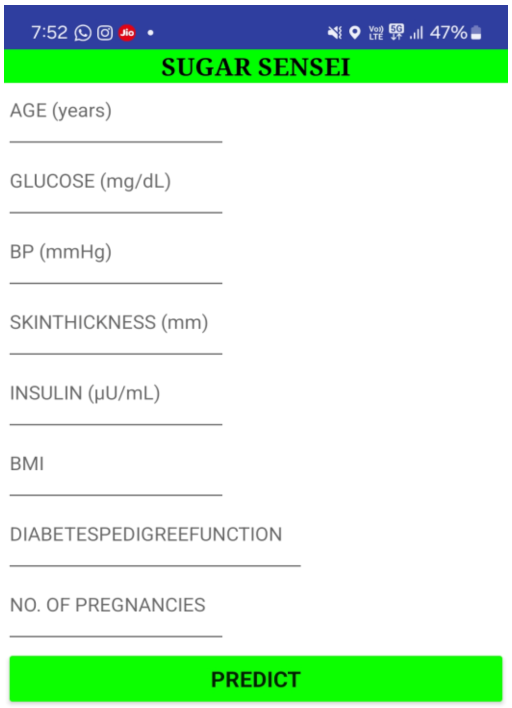
- This model performance highlights the accuracy in detecting patterns within the input health data, providing reliable results during API testing and app integration. The evaluation validates that the model is well-suited for real-world application, meeting the objectives set for accurate diabetes prediction.

2. **App Performance:**
   - The app delivers a smooth user experience, efficiently collecting input, processing data, and displaying predictions quickly. Testing on Android devices reveals quick response times for prediction results after submitting health data.
   - Integration with the FastAPI backend ensures stable and efficient communication, allowing real-time predictions without noticeable lag. The app effectively handles both valid inputs and potential errors, displaying informative messages and results, which contribute to an intuitive user experience.
   - Overall, the app maintains reliable performance, providing accurate predictions and an accessible, user-friendly interface for diabetes risk assessment.

Challenges and Limitations

Developing this project presents a few challenges and limitations. Integrating the backend API with the mobile app requires precise data formatting and consistent testing to ensure seamless communication, especially when using ngrok for external connectivity. Model accuracy depends on the quality and diversity of training data, so predictions may vary with real-world data that deviates from the training set. Additionally, limitations in MIT App Inventor’s customization options restrict certain interface enhancements. Despite these challenges, the app effectively demonstrates predictive capabilities and offers a functional solution for diabetes risk assessment.

MOBILE VIEW:



7:52 47%

SUGAR SENSEI

AGE (years)

GLUCOSE (mg/dL)

BP (mmHg)

SKINTHICKNESS (mm)

INSULIN (μU/mL)

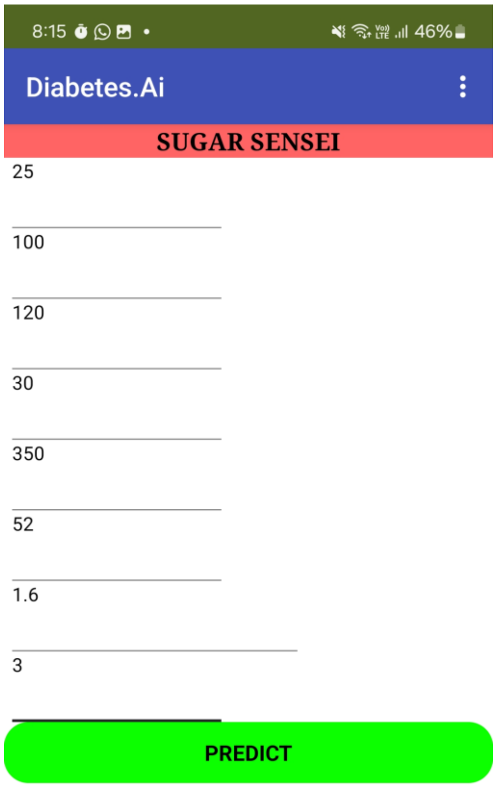
BMI

DIABETESPEDIGREEFUNCTION

NO. OF PREGNANCIES

PREDICT

TEST RESULTS:



8:15 46%

Diabetes.Ai

SUGAR SENSEI

25

100

120

30

350

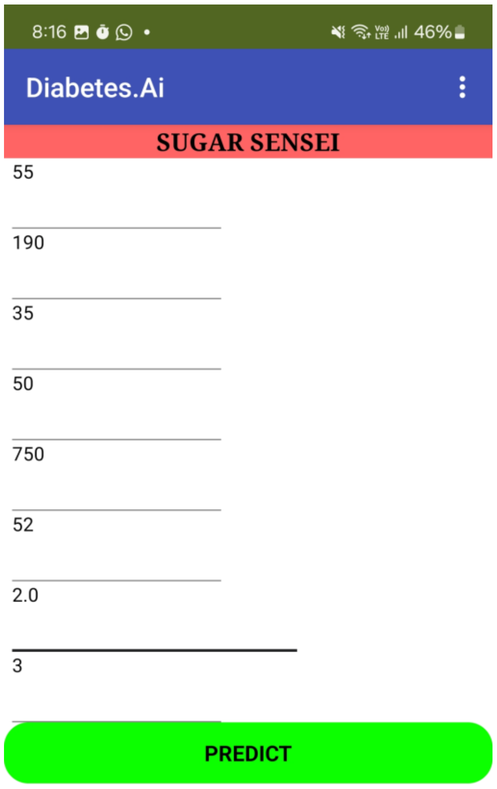
52

1.6

3

PREDICT

"The person is not diabetic"



8:16 46%

Diabetes.Ai

SUGAR SENSEI

55

190

35

50

750

52

2.0

3

PREDICT

"The person is diabetic"

Conclusion and Future Scope:

This project successfully combines machine learning with mobile app development to create an accessible diabetes prediction tool, enabling users to assess their risk quickly through an intuitive Android application. By leveraging a trained model hosted on a FastAPI backend, the app efficiently processes user health data to deliver reliable predictions, offering a practical application of machine learning in healthcare.

In terms of future scope, there are several potential enhancements. Expanding the model's training data with larger, more diverse datasets could improve its accuracy across different demographics. Additionally, integrating more complex machine learning algorithms may refine prediction precision. Further app development using platforms with greater customization, or converting it into a cross-platform application, could expand accessibility and improve user experience.

Appendix:

The input data ranges based on the dataset used:

- **Pregnancies**: 0 to 17
- **Glucose**: 0 to 199
- **Blood Pressure**: 0 to 122
- **Skin Thickness**: 0 to 99
- **Insulin**: 0 to 846
- **BMI**: 0.0 to 67.1
- **Diabetes Pedigree Function**: 0.078 to 2.42
- **Age**: 21 to 81