IoT and Edge Computing (CS3100)
Project Report

Predicting Student Outcomes with Machine Learning and Deploying on Embedded Systems

Submitted by

Shreyas R

1RVU22CSE153

School of Computer Science

RV University

Submitted to

Chandramouleeswaran Sankaran Sir

Professor

School of Computer Science

RV University

Submission date 13/11/2024

IoT and Edge Computing (CS3100)
Project Report

1. Abstract

This project aims to create an IoT-enabled model deployed on an ESP32 microcontroller to predict the risk of academic dropout in higher education students. Using a dataset encompassing academic, demographic, and socio-economic factors, a machine learning model is developed, optimized, and deployed for real-time inference at the edge. The project successfully demonstrates the feasibility of on-device predictions to facilitate early interventions for at-risk students.

2. Introduction

Project Background and Relevance: With the growing capabilities of IoT and Edge Computing, deploying machine learning models directly on low-power devices such as the ESP32 microcontroller has become a viable solution for real-time inference. Edge-based models allow for localized data processing, reducing latency and enabling timely decision-making. This project leverages IoT and Edge Computing to monitor and predict student success rates in real-time, providing early alerts for academic dropout risks and enabling educational institutions to take proactive steps.

Objectives: The project aims to develop and deploy a machine learning model capable of predicting student dropout risks. The objectives include:

- Training a neural network model using a higher education dataset.
- Optimizing the model for inference on a resource-constrained ESP32 microcontroller.
- Enabling real-time predictions on the ESP32, allowing the device to classify students into categories of dropout risk and academic success.

3. System Overview

System Architecture: The architecture comprises several key components:

Data Preprocessing Module: The data preprocessing includes removing the rows where target column value was enrolled then moving on to encoding the target variable, separating features and target, splitting the data into training and testing sets, and standardizing the features for model training.

Neural Network Model: A lightweight neural network developed and trained in Python, optimized for embedded deployment.

Edge Deployment Module: The model, after conversion to TensorFlow Lite, is deployed on the ESP32 using EloquentTinyML for on-device inference.

4. **Details of the Dataset:**
   The dataset used in this project was developed to address academic dropout and failure in higher education by identifying at-risk students early. It includes 34 input features, such as demographic, socio-economic, and academic path variables. The model output is a two-category classification representing different academic outcomes:

- **Marital status:** Indicates whether the student is married or single at the time of enrollment.
- **Application mode:** Describes how the student applied (e.g., through a standard procedure, special admissions, etc.).
- **Application order:** The order in which the student applied in relation to others.
- **Course:** The degree program the student is enrolled in (e.g., Computer Science, Engineering).
- **Daytime/evening attendance:** Specifies if the student attends courses during the day or evening.
- **Previous qualification:** The highest qualification the student obtained before enrolling in higher education.
- **Nationality:** The student's nationality.
- **Mother's qualification:** The highest level of education attained by the student's mother.
- **Father's qualification**: The highest level of education attained by the student's father.
- **Mother's occupation:** The occupation of the student's mother.
- **Father's occupation:** The occupation of the student's father.
- **Displaced:** Indicates whether the student is a displaced person (e.g., due to migration or war).
- **Educational special needs:** Specifies if the student has any educational special needs (e.g., learning disabilities).
- **Debtor:** Indicates whether the student has financial debt or outstanding fees.
- **Tuition fees up to date:** Specifies if the student has paid their tuition fees up to the current semester.
- **Gender:** The gender of the student.
- **Scholarship holder:** Indicates if the student is receiving a scholarship.
- **Age at enrollment:** The student's age at the time of enrollment.
- **International:** Indicates whether the student is an international student.
- **Curricular units 1st sem (credited):** The number of curricular units (courses) credited in the first semester.
- **Curricular units 1st sem (enrolled):** The number of curricular units the student is enrolled in during the first semester.

- **Curricular units 1st sem (evaluations):** The number of evaluations (exams, assignments, etc.) taken by the student for the first semester courses.
- **Curricular units 1st sem (approved):** The number of curricular units approved by the student in the first semester.
- **Curricular units 1st sem (grade):** The average grade achieved by the student in the first semester courses.
- **Curricular units 1st sem (without evaluations)**: The number of first semester courses for which the student did not take any evaluations.
- **Curricular units 2nd sem (credited)**: The number of curricular units credited in the second semester.
- **Curricular units 2nd sem (enrolled):** The number of curricular units the student is enrolled in during the second semester.
- **Curricular units 2nd sem (evaluations)**: The number of evaluations (exams, assignments, etc.) taken by the student for the second semester courses.
- **Curricular units 2nd sem (approved):** The number of curricular units approved by the student in the second semester.
- **Curricular units 2nd sem (grade):** The average grade achieved by the student in the second semester courses.
- **Curricular units 2nd sem (without evaluations)**: The number of second semester courses for which the student did not take any evaluations.
- **Unemployment rate:** The unemployment rate at the time of the student's enrollment, likely in the region or country.
- **Inflation rate:** The inflation rate at the time of the student's enrollment, likely in the region or country.
- **GDP:** The Gross Domestic Product (GDP) at the time of the student's enrollment, likely in the region or country.
- **Target:** The target category representing the student's status at the end of the normal duration of the course (dropout or graduate).

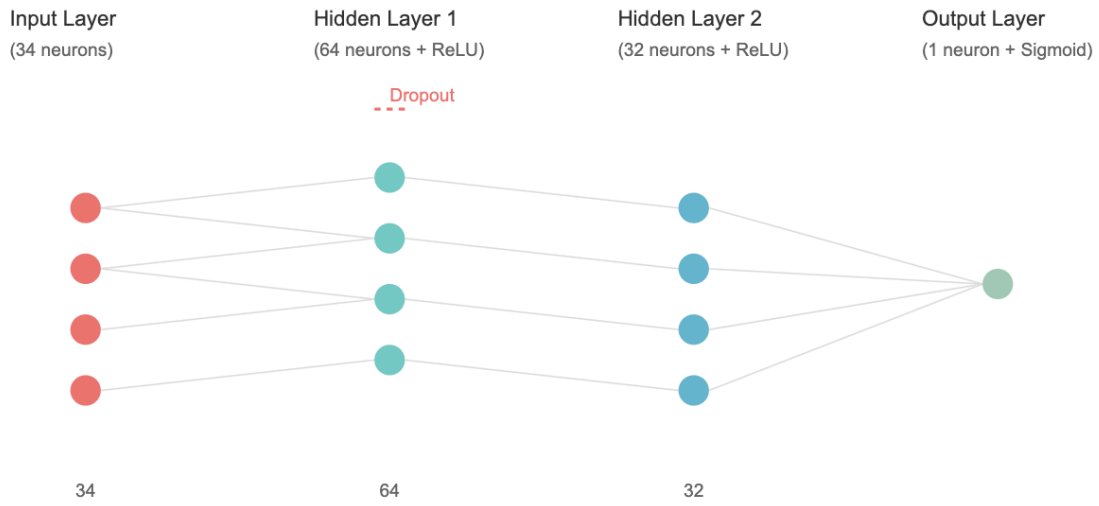
Dropout: Students at high risk of discontinuing their studies.
Graduate: Students likely to succeed academically.

5. Model Design:

The neural network architecture for this project includes:

Input Layer: Accepts 34 input features.
Hidden Layer 1: 64 neurons with ReLU activation and dropout for regularization.
Hidden Layer 2: 32 neurons with ReLU activation.
Output Layer: 1 neurons with sigmoid activation for binary classification (Dropout/Successful)

IoT and Edge Computing (CS3100)
Project Report



6. Requirements

Hardware: ESP32,
Software: TensorFlow, Numpy, Scikit-Learn for model training and preprocessing.

IDE: Arduino IDE for flashing the model onto ESP32.

7. Methodology

Dataset Preparation: Describe the initial steps where you load and preprocess the dataset, including encoding the target variable and normalizing feature data. Mention that the target classes ('Graduate' and 'Dropout') were encoded as binary values (1 and 0) to simplify the model's classification task.

Model Architecture: Briefly outline the neural network architecture with layers. The model has two hidden layers with ReLU activation and a final output layer with sigmoid activation to predict probabilities.

Training Process: Explain the model's compilation, including the use of Adam as the optimizer and Binary_Crossentropy as the loss function. Highlight that the model was trained with a validation split, providing real-time feedback on performance.

Model Conversion: Describe how the trained Keras model was saved and converted into a TensorFlow Lite (TFLite) format. This step was necessary to deploy the model on the ESP32.

IoT and Edge Computing (CS3100)
Project Report

Setup:

- **Hardware Configuration**: ESP32 microcontroller, a compact IoT device with WiFi and Bluetooth capabilities, was chosen for its processing power and small form factor, ideal for low-latency machine learning in embedded applications.
- **Software Configuration:** Python libraries like Pandas, Scikit-learn, TensorFlow, and Keras for training, along with TensorFlow Lite for model conversion. Mention that Arduino libraries, like EloquentTinyML, are used to deploy the model on the ESP32, allowing the device to perform predictions without needing a powerful server.

Code Summary: Key Functions or Modules of the Python Code, Focusing on ESP32 Deployment

Python Script:

- **label_encoder:** Encodes target labels for model training.

```
# Encode the target variable (convert 'Graduate'/'Dropout' to 1/0)

label_encoder = LabelEncoder()

data['Target'] = label_encoder.fit_transform(data['Target'])
```

- **StandardScaler:** Scales features for normalized training.

```
# Standardize the feature data

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

- **Sequential Model:** Configures and trains a neural network.

```
model = Sequential([

   Dense(64, input_shape=(X_train.shape[1],), activation='relu'),    # First
hidden layer

   Dense(32, activation='relu'),                                     # Second
hidden layer

   Dense(1, activation='sigmoid')                                    # Output
layer with sigmoid for binary classification

])
```

- **hex_to_c_array:** Converts the TFLite model binary to a C array for embedding in ESP32 firmware.

```
import time



def hex_to_c_array(hex_data, var_name="student_model_data"):

   """Convert hex data to a C array representation."""



   c_str = '#ifndef ' + var_name.upper() + '_H\n'

   c_str += '#define ' + var_name.upper() + '_H\n\n'

   c_str += '/* Generated from TensorFlow Lite model */\n'



   localtime = time.asctime(time.localtime(time.time()))

   c_str += f"// Generated on {localtime}\n\n"



   c_str += f"const unsigned int {var_name}_len = {len(hex_data)};\n"
```

```
c_str += f"const unsigned char {var_name}[] = {{\n"



 hex_lines = []

for i, val in enumerate(hex_data):

    hex_str = f'0x{val:02x}'

    if i < len(hex_data) - 1:

        hex_str += ','

    if (i + 1) % 12 == 0:

        hex_str += '\n'

    hex_lines.append(hex_str)

c_str += '\n'.join(hex_lines) + '\n};\n\n'



# Close header guard

c_str += '#endif // ' + var_name.upper() + '_H\n'



return c_str
```

IoT and Edge Computing (CS3100)
Project Report

ESP32 Code:

classifyAndDisplayResult: Core function for input prediction on ESP32, comparing predicted and expected classes.

```
void classifyAndDisplayResult(float *input, const char *label, const char *expected_class) {

   float fResult[NUMBER_OF_OUTPUTS];

   initfResult(fResult);

   ml.predict(input, fResult);




   bool predicted_class = fResult[0] > 0.5;



   Serial.print("\nClassification for ");

   Serial.println(label);

   Serial.print("Expected class: ");

   Serial.println(expected_class);

   Serial.print("Predicted class: ");

   if (predicted_class) {

       Serial.println("Graduated");

   } else {

       Serial.println("Dropout");

   }
```

```
    displayOutput(fResult);

}
```

displayOutput: Outputs the prediction probability for class 'Graduated'.

```
void displayOutput(float *fResult) {

    Serial.print("Class probability (Graduated): ");

    Serial.println(fResult[0]);

}
```

8. Implementation and Testing

Core Code Excerpts: Key snippets and explanation

Core Code Excerpts: Key snippets and explanations

Neural Network Model Training (Python code)

- **Model Architecture:** The neural network was built using Keras and TensorFlow, with 2 hidden layers and a sigmoid activation function for binary classification. The code was responsible for training the model on the dataset and evaluating its performance. Here's a key snippet from the Python code:

```
model = Sequential([

    Dense(64, input_shape=(X_train.shape[1],), activation='relu'),    # First
hidden layer

    Dense(32, activation='relu'),                                     # Second
hidden layer

    Dense(1, activation='sigmoid')                                    # Output
layer with sigmoid for binary classification

])
```

Model Training and Evaluation: After defining the model, it was compiled with the Adam optimizer and binary cross-entropy loss function. The model was then trained on the dataset using the `fit()` method, and predictions were made using the `predict()` method. Accuracy, classification report, and confusion matrix were used to assess performance.

```
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
y_pred = (model.predict(X_test) > 0.5).astype(int).flatten()
```

TensorFlow Lite Conversion: After training, the model was converted to TensorFlow Lite format for use in the embedded ESP32 environment.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()
```

ESP32 Deployment (Arduino Code)

- **Model Integration**: The converted TensorFlow Lite model was loaded onto the ESP32 using the `TinyML` library. The model data was embedded in a header file for easy access on the ESP32.

```
#include <EloquentTinyML.h>

#include "student_model_data101.h"

#define NUMBER_OF_INPUTS 34

#define NUMBER_OF_OUTPUTS 1

#define TENSOR_ARENA_SIZE 20 * 1024

Eloquent::TinyML::TfLite<NUMBER_OF_INPUTS, NUMBER_OF_OUTPUTS,
TENSOR_ARENA_SIZE> ml;
```

- **Model Prediction:** The input data was passed to the model for prediction, and the result was printed on the serial monitor. The code checks the predicted class (Graduate/Dropout) based on the output probability.

```
ml.predict(input, fResult);

bool predicted_class = fResult[0] > 0.5;
```

Testing:

Validation Accuracy on the trained Model:

```
val_accuracy: 0.9037
```

9. Results

Data Output:
Performance Notes: N/A
Provide the screenshots of the inputs and outputs:

- Both on the Python (after the training)

accuracy			0.89	1089
macro avg	0.89	0.88	0.88	1089
weighted avg	0.89	0.89	0.89	1089

- On ESP32 Serial monitor output (inferencing on the ESP32)

```
Classification for Student 1
Expected class: Dropout
Predicted class: Dropout
Class probability (Graduated): 0.00

Classification for Student 2
Expected class: Graduated
Predicted class: Graduated
Class probability (Graduated): 1.00
```

10. Conclusion

This project involved developing a neural network to predict student outcomes (Graduated or Dropout) using Python and TensorFlow. The model was trained and then converted into a TensorFlow Lite format for deployment on an ESP32 microcontroller. Integration with Arduino code enabled real-time predictions on the embedded system. The project provided hands-on experience with machine learning model deployment on resource-constrained devices. Key learnings included model optimization for embedded systems and the end-to-end process of building and deploying machine learning models.

IoT and Edge Computing (CS3100)
Project Report

****

IoT and Edge Computing (CS3100)
Project Report