Comprehensive Documentation for Fashion MNIST Classification with ANN and CNN

Introduction

This documentation describes a Python code for classifying the Fashion MNIST dataset using Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN) with TensorFlow and Keras. It includes information on installation, usage, and important details for users.

Installation

Prerequisites
Before using this code, ensure the following are installed:

Python 3.11
NumPy
Pandas
Matplotlib
TensorFlow
Keras

Usage

Data Loading
The code loads the Fashion MNIST dataset

ANN Model
The Artificial Neural Network (ANN) model is defined as follows:

A sequential model with fully connected layers.
CNN Model
The Convolutional Neural Network (CNN) model is defined as follows:

A sequential model with convolutional layers followed by fully connected layers.
Model Training
Both ANN and CNN models are trained using the model.fit method:

Result Visualization
The code provides visualization of training and validation accuracy and loss using Matplotlib.

Important Details
The Fashion MNIST dataset consists of grayscale images (28x28 pixels) representing 10 different fashion categories.
The ANN model consists of fully connected layers, while the CNN model includes convolutional layers to capture spatial information.
Users can modify the model architectures and hyperparameters to experiment with different configurations.

Conclusion
This documentation provides an overview of the code for Fashion MNIST classification using ANN and CNN models. Users can follow the installation and usage instructions, modify the models, and explore different configurations for image classification tasks.

Comprehensive Documentation for Task Management System

This documentation describes a Python script that implements a simple Task Management System using SQLite for data storage. The system allows users to perform operations such as adding tasks, listing tasks, updating tasks, and deleting tasks. Below is detailed information about the code, including installation, usage, and important details for users.

Installation

Prerequisites
Before using this code, ensure you have the following installed:

Python 3.11
SQLite

Run the Code: Execute the script using a Python IDE or terminal:

bash
Copy code
```
python your_script.py
```

Usage

Database Initialization

The code creates or connects to a SQLite database named tasks.db and defines a tasks table with columns for task information: id, task_name, task_description, and task_status.

Functionality

The code provides the following functionalities:

Add Task: Allows users to add a task with a name and description. The task is assigned a status of "Pending" by default.

List Tasks: Lists all tasks in the database, displaying their ID, name, description, and status.

Update Task: Allows users to update a task by specifying its ID, and then providing a new name, description, and status.

Delete Task: Permits the deletion of a task by specifying its ID.

Exit: Exits the Task Management System.

Important Details

The database file (tasks.db) will be created in the same directory as the script.
When adding or updating tasks, the user must provide valid inputs.
Task IDs are unique and automatically generated when adding tasks.
The system displays an error message for invalid user inputs.
The Task Management System operates in a loop until the user chooses to exit.

Conclusion

This documentation provides an overview of the Task Management System code. Users can follow the installation and usage instructions to manage tasks effectively. The code offers a simple yet functional tool for managing tasks using a SQLite database as the backend storage.

Machine Learning Model Integration with Google API

A complete example of integrating a machine learning model with a Google API using Google Cloud Functions. The code demonstrates how to deploy a Flask-based Python web application that serves as an HTTP endpoint, enabling users to send input data for predictions using a pre-trained machine learning model.

Capabilities

The code in this repository offers the following capabilities:

Integration of a machine learning model into a serverless Google Cloud Function.
Acceptance of input data via HTTP POST requests for real-time predictions.
Loading and utilization of a pre-trained machine learning model (Random Forest Regressor) for regression tasks.
Error handling and JSON response for smooth interaction with the API.

Installation

To run this code and deploy the machine learning model integration with Google API, follow these steps:

Clone the Repository:

Clone this repository to a local machine using the following command:

bash
Copy code
```
git clone https://github.com/yourusername/machine-learning-google-api.git
```
Install Dependencies:

Navigate to the project directory and install the required Python dependencies using pip:

bash
Copy code
```
cd machine-learning-google-api
pip install -r requirements.txt
```
Deploy to Google Cloud Functions:

Set up a Google Cloud Platform (GCP) project.
Enable the Google Cloud Functions API for your project in the GCP Console.
Use the Google Cloud SDK to deploy the Flask app as a Google Cloud Function.
Model Loading:

Ensure that the pre-trained machine learning model (Random Forest Regressor) is saved as 'marriage_age_predictor.ml' in the project directory. There need to adapt the code if you are using a different model format.

API Endpoint:

Once deployed, you will receive an API endpoint URL for your Google Cloud Function. This URL will be used to send input data for predictions.

4. Usage
Here's how to use the machine learning model integration with the Google API:

Send Input Data for Prediction:

Use an HTTP POST request to the API endpoint URL provided after deploying the Google Cloud Function.
Send input data in JSON format with the required features for prediction. For example:
Json

```
{
  "features":
{
    "gender": "Female",
    "religion": "Hindu",
    "caste": "Brahmin",
    "mother_tongue": "English",
    "country": "India",
    "height": "5'11\""
  }
}
```

Receive Predictions:
The API will return predictions based on the input data, in JSON format.

Error Handling:
The API includes error handling to ensure smooth interaction. If there are any issues, you will receive an error message in JSON format.

Conclusion:

In conclusion, this comprehensive documentation provides users with a clear understanding of the capabilities, installation process, and usage of the machine learning model integration with a Google API. By following the outlined steps, users can successfully deploy a Flask-based Python web application as a Google Cloud Function, allowing them to make real-time predictions using a pre-trained machine learning model.

This code serves as a valuable resource for data scientists and developers looking to integrate machine learning models into production environments, particularly when interacting with Google Cloud Platform services. The documentation also encourages contributions and feedback to improve the codebase and its functionality.

By following the instructions and guidelines provided in this documentation, users can leverage the power of machine learning models through a user-friendly API, opening up possibilities for various applications and use cases.