System Requirements Specification Index

Tensor flow Iris Detection project

Version 1.0

Objective

You are working as a junior data scientist at a botanical research lab where the team is developing an AI system to assist field researchers in identifying different plant species. One particular species of interest, **Iris Setosa**, needs to be detected automatically due to its distinct characteristics and research relevance. To achieve this, you are provided with the classic **Iris dataset**, which includes measurements of sepal length, sepal width, petal length, and petal width for three types of iris flowers. Your task is to build a **binary classification model** using **TensorFlow and Keras** that determines whether a given flower is Setosa or not. The target output should be a probability score along with a classification decision. This model, once trained and validated, will be integrated into a mobile application used by botanists in the field, helping automate the identification process and ensuring faster, more accurate data collection in botanical studies.

We will use the built-in **Iris dataset** from `sklearn.datasets.load_iris`.

Each record in the dataset contains the following **4 input features** (all are numeric):

Feature Name	Description
Sepal Length (cm)	Length of the outer petal
Sepal Width (cm)	Width of the outer petal
Petal Length (cm)	Length of the inner petal
Petal Width (cm)	Width of the inner petal

The **target label** (`y`) contains one of 3 values:

- `0` → Setosa
- `1` → Versicolor
- `2` → Virginica

For this task, we will convert this into a **binary classification problem**:

- `1` → if the flower is **Setosa**
- `0` → otherwise

What You Have to Build

You need to complete a Python program consisting of six functions. Each function should implement one step of the ML workflow.

Start from the given skeleton code where only the dataset function is implemented.

Task Breakdown by Function

- **To load the data which we are trying to find out if this flower is Setosa flower or not**

`get_prediction_sample()`

Purpose:
This function returns a known sample (Setosa flower) for testing purposes. It is used by both the `predict_sample()` function

Instructions:

- Return a 2D NumPy array with shape `(1, 4)`
- Example:

```
return np.array([[5.1, 3.5, 1.4, 0.2]])
```

1. `load_and_preprocess()`

Purpose:
Load the dataset, convert it into a binary problem, split it, and scale it.

Instructions:

- Load the Iris dataset using `sklearn.datasets.load_iris()`
- Convert the target labels:
  - 1 if Setosa (i.e., `target == 0`)
  - 0 otherwise
- Split the data into training and testing sets using `train_test_split()` (80/20)
- Normalize the input features using `StandardScaler`
  - `fit_transform()` for training data
  - `transform()` for test data

Returns:
`X_train`, `X_test`, `y_train`, `y_test`, `scaler`

2. `build_model()`

You must implement this function.

- Use `tf.keras.Sequential()` to build the model
- Add the following layers:
  - Dense layer with 10 units, `activation='relu'`, `input_shape=(4,)`
  - Output Dense layer with 1 unit, `activation='sigmoid'`
- Compile the model using:
  - `optimizer='adam'`
  - `loss='binary_crossentropy'`
  - `metrics=['accuracy']`

Return:

- The compiled model

3. `train_model(model, X_train, y_train)`

You must implement this function.

- Train the model using `model.fit()` with:
  - `epochs=50`
  - `batch_size=8`
  - `verbose=0` (suppress training logs)

Return:

- The trained model

---

4. `evaluate_model(model, X_test, y_test)`

You must implement this function.

- Evaluate the model on test data using `model.evaluate()`
- Print the result in this exact format:

```
Test Accuracy: 0.9821
```

---

5. `predict_sample(model, scaler)`

You must implement this function.

- Use the following sample input (likely Setosa):

```
[[5.1, 3.5, 1.4, 0.2]]
```

- Use the `scaler` to transform the input
- Call `model.predict()` and print:

```
Predicted probability of being Setosa: 0.XXXX
```

Round to 4 decimal places for display.

---

6. `main()`

You must implement this function.

- Call each of the above functions in order:
  - Load data
  - Build model
  - Train model
  - Evaluate model
  - Predict a sample

Execution Steps to Follow:

● All actions like build, compile, running application, running test cases will be through

Command Terminal.

● To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

● This editor Auto Saves the code

● If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) .

● These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

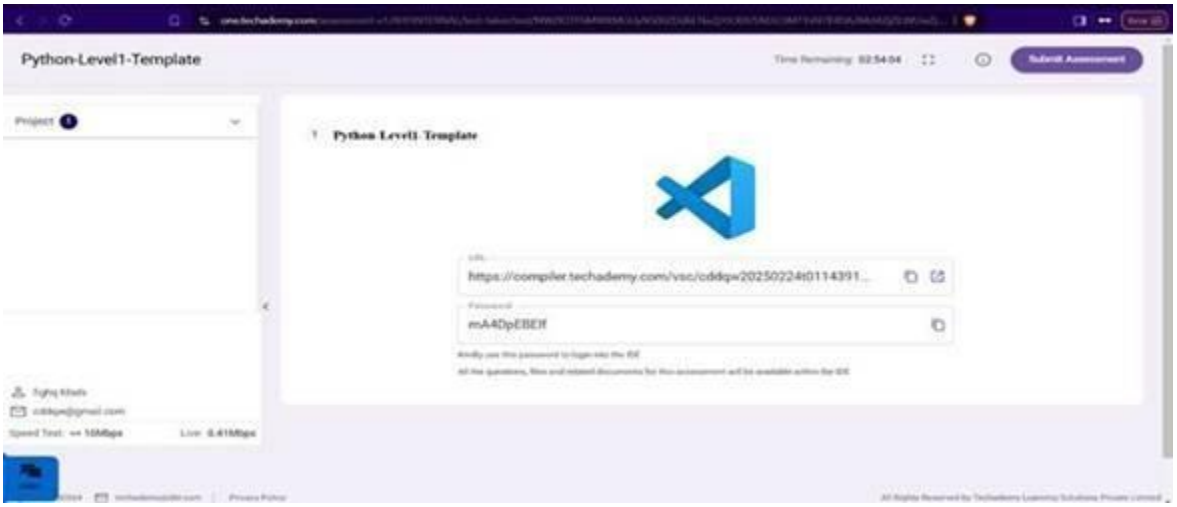
- To launch application: python3 filename.py
- To run Test cases: python3 -m unittest

Screen shot to run the program

To run the application

python3 filename.py

To run the testcase python3 -m unittest



● Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.