System Requirements Specification Index

For

Pytorch fitness usecase L1

IIHT Pvt. Ltd.

fullstack@iiht.com

PYTORCH FITNESS USECASE

Objective:

This use case focuses on building a machine learning model to classify workout sessions based on the number of calories burned. Fitness tracking devices collect various data points such as heart rate, steps, and workout duration, but raw calorie values alone can be difficult to interpret. Categorizing calorie burn into levels like low, medium, and high helps users and trainers quickly assess workout intensity. we can provide personalized insights and improve the user experience in fitness applications.

You are given a CSV file (`fitness_data.csv`) containing workout session data. Your goal is to:

1. **Bin calorie values** into 3 levels (low, medium, high),
2. **Preprocess** the data,
3. **Train a classifier** using PyTorch,
4. **Evaluate** and **save** the model.

This exercise will test your knowledge of:

- Pandas data manipulation,
- PyTorch datasets, models, and training loops,
- Data preprocessing with Scikit-learn.

Part 1: Bin Calorie Values

Complete the function below to bin `calories_burned` into 3 categories:

- **0**: 0–200 calories (low)
- **1**: 200–350 calories (medium)
- **2**: 350–1000 calories (high)

Inside this function:

- Use `pandas.cut` to divide the `calories_burned` values into three bins:
  - Category **0** for values between 0 and 200 (inclusive of 0, exclusive of 200),
  - Category **1** for values between 200 and 350,
  - Category **2** for values between 350 and 1000.
- Use the bin edges `[0, 200, 350, 1000]` and the labels `[0, 1, 2]`.
- Ensure the result is returned as a `pandas.Series` of **integer values**.
- Return this series of category labels.

Task 2: Load and Preprocess Data

Write a function named **`load_data_from_csv(path='fitness_data.csv')`** that reads and processes the fitness data for training.

Inside this function:

1. Load the CSV file using `pandas.read_csv`.
2. Apply the `bin_calories` function to the `calories_burned` column to convert it into categorical labels.
3. Drop any rows where the binned values are missing (`NaN`), which may occur due to out-of-range values.
4. Separate the dataset into:
   - **Features (`X`)**: all columns except `calories_burned`
   - **Target (`y`)**: the binned labels you created
5. Use `StandardScaler` from `sklearn.preprocessing` to scale the features.
6. Use `StratifiedShuffleSplit` from `sklearn.model_selection` to split the data into training and testing sets, ensuring the split is **stratified by target class**, with:
   - `test_size = 0.2`
   - `n_splits = 1`
   - `random_state = 42`
7. Convert the resulting splits into PyTorch tensors:
   - Features should be of type `torch.float32`
   - Labels should be of type `torch.long`

Return value: A tuple of four PyTorch tensors in the following order:

- `X_train`: the training features (torch.float32 tensor)
- `y_train`: the training labels (torch.long tensor)
- `X_test`: the test features (torch.float32 tensor)
- `y_test`: the test labels (torch.long tensor)

Task 3: Create a Custom PyTorch Dataset

Create a class called **`FitnessDataset`** that inherits from `torch.utils.data.Dataset`.

Implement the following:

- An `__init__` method that accepts two arguments: `X` and `y`, and stores them internally.
- A `__len__` method that returns the number of samples in the dataset.
- A `__getitem__` method that takes an index `idx` and returns a tuple: `(X[idx], y[idx])`.

Return behavior: When indexing the dataset, it should return a tuple of one feature tensor and one label tensor corresponding to that sample.

Task 4: Build the Model

Write a function called **`build_model(input_size=5, num_classes=3)`** that builds and returns a PyTorch `nn.Sequential` model.

The architecture should include:

1. A fully connected (`Linear`) layer from `input_size` to 16 units,
2. A `ReLU` activation layer,
3. A `Dropout` layer with a dropout rate of 0.3,
4. A second `Linear` layer from 16 units to 8 units,
5. Another `ReLU` activation layer,
6. An output `Linear` layer from 8 units to `num_classes`.

Return value: A PyTorch `nn.Sequential` model.

Task 5: Train the Model

Write a function named **`train_model(model, dataloader, val_loader=None, epochs=15, lr=0.01)`** that trains the given model using the provided training data loader.

Within this function:

1. Use `CrossEntropyLoss` as the loss function.
2. Use the `Adam` optimizer with the specified learning rate.
3. For each epoch:
    - Set the model to training mode.
    - Loop over each batch:
        - Zero the optimizer gradients,
        - Perform a forward pass,
        - Compute the loss,
        - Perform a backward pass (backpropagation),
        - Update the model parameters.
4. If a validation data loader is provided (`val_loader` is not `None`), evaluate the model after each epoch and print the validation accuracy.

Return value: This function doesn't need to return anything unless you wish to return the trained model explicitly.

Task 6: Evaluate the Model

Write a function named **`evaluate_model(model, dataloader)`** that calculates the accuracy of the model on a given dataset.

Inside this function:

1. Set the model to evaluation mode.
2. Disable gradient tracking using `torch.no_grad()`.
3. Loop over the data loader:
    - Perform a forward pass,

   - Get predicted classes using `torch.max` on the output logits,
   - Count how many predictions match the true labels.
4. Compute and return the **accuracy** as a float (e.g., `0.87` for 87%).

Return value: A float representing classification accuracy (between 0 and 1).

Task 7: Save the Trained Model

Write a function named **`save_model(model, path='fitness_model_class.pth')`** that saves the model's state dictionary to the given file path using `torch.save`.

Return value: This function does not return anything.

Task 8: Load a Saved Model

Write a function named **`load_model(path='fitness_model_class.pth', input_size=5, num_classes=3)`** that:

1. Builds a new model using `build_model()`,
2. Loads the saved state dictionary from the file using `torch.load`,
3. Sets the model to evaluation mode.

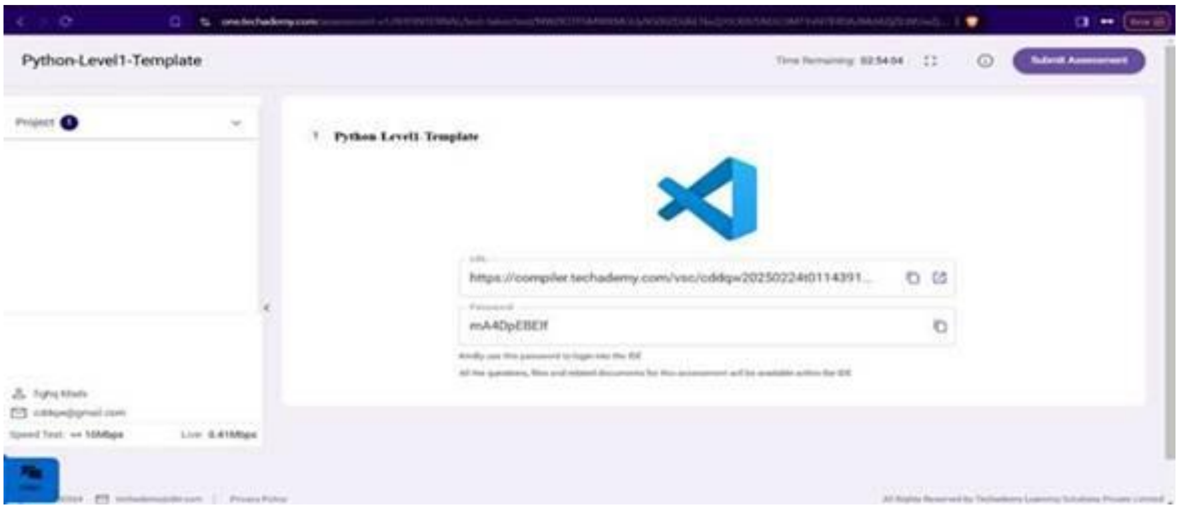
Return value: The loaded model (PyTorch `nn.Sequential` instance), ready for inference.

Task 9: Main Pipeline Execution

In the `__main__` block of your script, execute the full training and evaluation pipeline in this order:

1. Load the data using `load_data_from_csv()`.
2. Wrap the training and test data in `FitnessDataset` instances.
3. Create PyTorch `DataLoader`s for training and testing:
   - Use `batch_size=4` and `shuffle=True` for training.
4. Build the model using `build_model()` and correct `input_size`.
5. Train the model using `train_model()`.
6. Evaluate the model using `evaluate_model()` and print the accuracy.
7. Save the trained model to a file using `save_model()`.

Execution Steps to Follow:

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor Auto Saves the code
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. 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.
6. To launch application: python3 filename.py
7. To run Test cases: python3 -m unittest
8. Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.
9. Screen shot to run the program
10. To run the application
11. Python3 filename.py
12. To run the testcase   python -m unittest
13. 

Python-Level1-Template

Time Remaining: 02:54:04

Submit Assessment

Project

1 Python Level1 Template

URL: https://compiler.techademy.com/vsc/cddqw20250224t0114391...

Password: mA4DpEBEIf

14. 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 the code.