XGBoost Guided Exercise

XGBoost (Extreme Gradient Boosting) is a powerful and efficient implementation of gradient boosting that is widely used in machine learning competitions and real-world applications. It is optimized for speed and performance, supporting parallelization, distributed computing, and regularization.

Key Features of XGBoost:

- Highly efficient, optimized for speed
- Regularization to prevent overfitting
- Parallelized computation
- Built-in cross-validation
- Ability to handle missing values

Installation

To install XGBoost, use the following command:

```
pip install xgboost
```

You will also need `numpy` and `pandas` for handling data:

```
pip install numpy pandas
```

1. Importing Required Libraries

Fill in the missing parts below to import the necessary libraries.

In [1]:
```
import xgboost as xgb
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

2. Loading and Preparing Data

Fill in the missing parts to load the dataset and split it into training and testing sets.

In [2]:
```
from sklearn.datasets import load_iris

data = load_iris()
X = data.data
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

3. Converting Data into DMatrix

XGBoost uses a special data structure called `DMatrix`, which optimizes memory efficiency. Complete the code below:

In [3]:
```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

4. Setting Hyperparameters

Define the hyperparameters for the XGBoost model by filling in the missing values:

In [4]:
```
params = {
    'objective': 'multi:softmax',
    'num_class': 3,
    'max_depth': 4,
    'eta': 0.1,
    'eval_metric': 'mlogloss'
}
```

5. Training the Model

Complete the following code to train the model:

In [5]:
```
num_round = 80
bst = xgb.train(params, dtrain, num_round)
```

6. Making Predictions

Fill in the missing parts to make predictions:

In [6]:
```
y_pred = bst.predict(dtest).astype(int)
```

7. Evaluating Performance

Complete the following code to evaluate the accuracy of the model:

In [7]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 100.00%
```

8. Using XGBoost with Scikit-Learn API

XGBoost provides a Scikit-Learn compatible API for easier integration. Fill in the missing parts:

In [8]:
```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
/Users/igorsu/.pyenv/versions/3.12.12/lib/python3.12/site-packages/xgboost/t
raining.py:200: UserWarning: [16:39:51] WARNING: /Users/runner/work/xgboost/
xgboost/src/learner.cc:782:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Accuracy: 100.00%
```