CS 4840 Intro Machine Learning - Lab Assignment 1

End-to-End Machine Learning Pipeline: A Linear Regression Problem

This is only for undergraduate students in CS 4840

1. Overview

The learning objective of this lab assignment is for students to understand the end-to-end machine learning pipeline, including how to load the data from csv file to DataFrame, how to take a quick look at data structure and statistics, how to visualize the data to gain insights, how to deal with missing features and categorical features, and how to scale feature values. As we use diabetes data as an example, this lab assignment also includes how to build a linear regression model using scikit-learn and gradient descent (PyTorch) to predict the diabetes progression in one year, and how to evaluate the regression results.

Lecture notes.

Detailed coverage of these topics can be found in the following:

Terms and Principles of Machine Learning
Model Training using Gradient Descent
Data and Feature Examination
Feature Preprocessing

Code demonstrations.

Code 2024-09-09-M-Model Training using Gradient Descent-1.ipynb
Code 2024-09-11-W-Gradient Descent Variants.ipynb
Code 2024-09-11-W-Training Linear Regression using Scikit-Learn.ipynb
Code 2024-09-11-W-Training Linear Regression using PyTorch.ipynb
Code 2024-09-16-M-Data and Feature Processing.ipynb

2. Submission

You need to submit a detailed lab report with code, running results, and answers to the questions. If you submit a jupyter notebook ("Firstname-Lastname-4840-Lab1.ipynd"), please fill in this file directly and place the code, running results, and answers in order for each question. If you submit a PDF report ("Firstname-Lastname-4840-Lab1.pdf") with code file

("Firstname-Lastname-4840-Lab1.py"), please include the screenshots (code and running results) with answers for each question in the report.

3. Questions (50 points)

For this lab assignment, you will be using the `Diabetes dataset` to complete the following tasks and answer the questions. Diabetes dataset has different features and a target label. You will use these features to build a linear regression model to predict the diabetes disease progression one year. First, please place `diabetes.csv` and your notebook/python file in the same directory, and load the data into DataFrame.

In [ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Please place diabetes.csv and your notebook/python file in the same directory; oth
DATA_PATH = ""

def load_diabetes_data(diabetes_path=DATA_PATH):
    csv_path = os.path.join(diabetes_path, "diabetes.csv")
    return pd.read_csv(csv_path)

diabetes = load_diabetes_data()
```

Question 1 (3 points):

Please print out the information/statistics of diabetes DataFrame in function `answer_one( )` , and describe: how many records does the diabetes dataset have? Except for the target label `diabetes_progression_one_year` , how many features does the diabetes dataset have? What are these features? Among these features, which feature is non-numerical (categorical) and which feature has missing values?

In [ ]:
```
def answer_one():
    #Please complete print() using diabetes(DataFrame)'s information API
    print( )

#Run your function in the cell to return the result
answer_one()
```

Answer 1:

(Put your answers here)

Question 2 (5 points):

Please calculate feature correlation matrix and print out the feature correlation information for `diabetes_progression_one_year` in function `answer_three( )` , and describe: except for `diabetes_progression_one_year` itself, which feature has the strongest correlation with `diabetes_progression_one_year` , and which feature has the weakest correlation with `diabetes_progression_one_year` ? Why?

After that, please fill in feature name in the option `x=" "` of plot function in `answer_two( )` to plot the data distribution between the strongest/weakest feature and `diabetes_progression_one_year` . Please describe your observations on these two plots: how the data distribution looks like for each plot, and if the data distribution conforms to the obtained feature correlation?

In [ ]:
```
def answer_two():
    #Please complete the function using diabetes(DataFrame)'s correlation API
    corr_matrix =
    #Please print out the feature correlation information for diabetes_progression_
    print( )

    #First, run the above code and figure out the feature (strongest_feature) that
    #and the feature (weakest_feature) that has the weakest correlation with `diabe
    #Then, set option x=" " using the name of strongest_feature
    diabetes.plot(kind="scatter", x=" ", y="diabetes_progression_one_year", alpha=0
    plt.axis([0, 45, 0, 360])

    #Please set option x=" " using the name of weakest_feature
    diabetes.plot(kind="scatter", x=" ", y="diabetes_progression_one_year", alpha=0
    plt.axis([0, 250, 0, 360])

    plt.show()

#Run your function in the cell to return the result
answer_two()
```

Answer 2:

(Put your answers here)

Devide the DataFrame into features `diabetes_features` and labels `diabetes_labels`

In [ ]:
```
diabetes_labels = diabetes["diabetes_progression_one_year"].copy() # use diabete_pr
diabetes_features = diabetes.drop("diabetes_progression_one_year", axis=1) # drop d
```

Question 3 (3 points):

As `total_cholesterol` has missing values, please use its median number to set those missing feature values in function `answer_three( )` .

In [ ]:
```
def answer_three():
    #Please operate on diabetes_features to set the missing "total_cholesterol" val
    median =
    diabetes_features.

#Run your function in the cell to return the result
answer_three()
```

In [ ]:
```
#Check if there are still records with null feature values
#Empty DataFrame means no record with null feature values
print(diabetes_features[diabetes_features.isnull().any(axis=1)].head())
```

Devide features into numerical part `diabetes_num` and categorical part `diabetes_cat`

In [ ]:
```
#diabetes_num only includes the numerical features without gender
diabetes_num = diabetes_features.drop("gender", axis=1)

#diabetes_cat only includes the categorical feature gender
diabetes_cat = diabetes_features[["gender"]]
print(diabetes_cat["gender"].value_counts())
```

Question 4 (3 points):

As gender contains categories without order, please convert these categoties into numbers directly using `OrdinalEncoder` in function `answer_four( )` , and print out gender values after conversion.

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder

def answer_four():
    ordinal_encoder =
    #Please complete the code line using fit_transform API to operate on diabetes_c
    diabetes_cat_encoded =

    return diabetes_cat_encoded

#Run your function in the cell to return the result
diabetes_cat_encoded = answer_four()
print(diabetes_cat_encoded)
```

Question 5 (3 points):

As numerical features have very different scales, please get all the features to have the same scale using `StandardScaler` in function `answer_five( )`, and print out feature values after scaling.

In [ ]:
```
from sklearn.preprocessing import StandardScaler

def answer_five():
    std_scaler =
    #Please complete the code line using fit_transform API to operate on diabetes_n
    diabetes_num_scaled =

    return diabetes_num_scaled

#Run your function in the cell to return the result
diabetes_num_scaled = answer_five()
print(diabetes_num_scaled)
```

Concatenate `diabetes_num_scaled` and `diabetes_cat_encoded` into the final features `X`

In [ ]:
```
#Diabetes features
X = np.concatenate((diabetes_num_scaled, diabetes_cat_encoded), axis=1)

#Diabetes labels
y = diabetes_labels.to_numpy()

print(X.shape)
print(y.shape)
```

Question 6 (5 points):

Please use `train_test_split` to split `X` and `y` into training and test sets `(X_train, X_test, y_train, and y_test)` in function `answer_six()`, and describe the shape of `X_train`, `X_test`, `y_train`, and `y_test`, respectively.

Set `random_state=42` and `test_size=0.2` in `train_test_split` to make sure 80% of your dataset is used for training and 20% for testing

In [ ]:
```
from sklearn.model_selection import train_test_split

def answer_six():
    #Please complete the code line using train_test_split
    X_train, X_test, y_train, y_test =

    return X_train, X_test, y_train, y_test
```

```
#Run your function in the cell to return the result
X_train, X_test, y_train, y_test = answer_six()
```

Answer 6:

(Put your answers here)

Question 7 (10 points):

Please first use `LinearRegression` from scikit-learn to build a linear regression model with `X_train` and `y_train` in function `answer_seven( )` , and then evaluate the trained linear regression model by calculating root mean square error (RMSE) and mean absolute error (MAE) between the true labels `y_test` and predictions `y_predict` , and describe the results of RMSE and MAE.

According to RMSE and MAE, describe if your linear regression model trained on `X_train` and `y_train` using scikit-learn makes good predictions and explain why.

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error
from sklearn.metrics import mean_absolute_error

def answer_seven():
    #Please create the model
    lin_reg =

    #Please train the model using fit API
    lin_reg.

    #Make prediction on the X_test using the trained model
    y_predict =

    #Please calculate root mean square error using root_mean_squared_error API
    rmse =

    #Please calculate mean absolute error using mean_absolute_error API
    mae =

    return rmse, mae

#Run your function in the cell to return the result
rmse_sklearn, mae_sklearn = answer_seven()
print(rmse_sklearn, mae_sklearn)
```

Answer 7:

(Put your answers here)

Pre-define a set of functions used by PyTorch model training

In [ ]:
```
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F

np.random.seed(42)
torch.manual_seed(42)

#Data convertion class
class ConvertDataset(Dataset):
    def __init__(self, X, y, transform=None):
        self.X = X
        self.y = y
        self.transform = transform

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        feature = self.X[idx]
        label = self.y[idx]
        sample = {'feature': feature, 'label': label}

        if self.transform:
            sample = self.transform(sample)

        return sample

#ToTensor function
class ToTensor(object):
    def __call__(self, sample):
        feature, label = sample['feature'], sample['label']
        label = np.array(label)
        return {'feature': torch.from_numpy(feature).float(),
                'label': torch.from_numpy(label).float()}

#Training function
def train(epoch, model, train_dataloader, optimizer):
    model.train()

    train_loss = 0.0

    for i, data in enumerate(train_dataloader):
        X, y = data['feature'], data['label']

        optimizer.zero_grad()

        predictions = model(X).squeeze()

        loss = lossfunction(predictions, y)
```

```
        loss.backward()
        optimizer.step()

        #print statistics
        train_loss += loss.item()

    print("epoch (%d): Train loss: %.3f" % (epoch, train_loss/10000))
```

Question 8 (3 points):

Please choose a batch size of your choice, convert diabetes training data `X_train` and diabetes labels `y_train` into tensors used by PyTorch, and then load the converted training data into Dataloader in function `answer_eight( )` .

In [ ]:
```
b_size =

def answer_eight(b_size, X_train, y_train):
    #Please convert X_train and y_train
    train_dataset =

    #Load the converted training data into DataLoader: pass b_size you choose to th
    train_dataloader =

    return train_dataloader

#Run your function in the cell to return the result
train_dataloader = answer_eight(b_size, X_train, y_train)
```

Question 9 (15 points):

Please first choose epochs, learning rate, and loss function of your choice. In function `answer_nine( )` , create a linear regression class, and then instantiate an object from that class to serve as the model. After that, specify `optimizer` using `Adam` , and then train the model. After the model is trained, calculate root mean square error (RMSE) and mean absolute error (MAE) between the true labels `y_test` and predictions `y_predict` , and describe the results of RMSE and MAE.

According to RMSE and MAE, describe if your linear regression model trained on `X_train` and `y_train` using PyTorch makes good predictions and explain why.

Also, compare to the results in Question 7, which one is better, and explain the potential reason why this difference happens.

In [ ]:
```
epochs =       #Consider the input data is very non-linear, please choose a large e
learning_rate =
lossfunction =

def answer_nine(train_dataloader):
```

```
    class LinearRegression(nn.Module):
        def __init__(self):
            super(LinearRegression, self).__init__()
            #The first parameter should be the feature dimension: refer to the answ
            #The second parameter should be the number of regression output, which
            self.fc =

        def forward(self, x):
            #Define the calculation from x to y
            y =
            return y

    #Instantizte an object from the class as the model
    model =
    #Define optimizer using Adam
    optimizer =

    #Train the model
    for epoch in range(1, epochs + 1):
        #Run train() function

    #Make prediction on the X_test using the trained model
    y_predict = model(torch.FloatTensor(X_test)).detach().numpy()

    #Please calculate root mean square error using root_mean_squared_error API
    rmse =

    #Please calculate mean absolute error using mean_absolute_error API
    mae =

    return rmse, mae

#Run your function in the cell to return the result
rmse_pytorch, mae_pytorch = answer_nine(train_dataloader)
print(rmse_pytorch, mae_pytorch)
```

Answer 9:

(Put your answers here)