Lab 5: Feature Scaling and Engineering, Linear Regression with Scikit-Learn

This lab consists of three parts:

1. Feature scaling and learning rate
2. Feature engineering and polynomial regression
3. Linear regression using Scikit-learn

Goals of Lab

In this lab, you will:

- run Gradient Descent on a data set with multiple features

- explore the impact of the learning rate alpha on gradient descent

- improve performance of gradient descent by feature scaling using z-score normalization

- Utilize scikit-learn to implement linear regression using Gradient Descent

Guidelines for Completing the Lab

In order to complete this lab, you need to run and understand the codes that are provided in this lab. Furthermore, the you need to complete Task 1 to Task 7 under part 1. For these tasks, the following deliverable(s) are required

1. Python file(s) to demonstrate your implementation
2. Screenshots of task wise outputs generated by your code. You have to place these screenshots in this lab under the task with title, for example, “Solution Task 5”

Part 1: Feature Scaling and Learning Rate

Goals

- Utilize the multiple variables routines developed in the previous lab
- run Gradient Descent on a data set with multiple features
- explore the impact of the learning rate alpha on gradient descent
- improve performance of gradient descent by feature scaling using z-score normalization
- explore feature engineering and polynomial regression

Problem Statement

As in the previous labs, you will use the motivating example of housing price prediction. The training data set contains many examples with 4 features (size, bedrooms, floors and age) shown in the table below. Note, in this lab, the Size feature is in sqft while earlier labs utilized 1000 sqft. This data set is larger than the previous lab.

We would like to build a linear regression model using these values so we can then predict the price for other houses - say, a house with 1200 sqft, 3 bedrooms, 1 floor, 40 years old.

1.1 Dataset

Sample code to open excel files and extracting columns

Here's a simple example of how you could read an Excel file and extract specific columns using pandas. First, make sure you have **pandas** and **openpyxl** installed in your environment, as openpyxl is needed to read Excel files (xlsx format). You can install them using pip if you haven't already:

```
pip install pandas openpyxl
```

```
import pandas as pd
```

```
# The path to your Excel file
```

```
excel_file_path = 'path_to_your_excel_file.xlsx'

# Load the Excel file
df = pd.read_excel(excel_file_path)

# Assuming you want to extract columns named 'Column1' and 'Column2'
extracted_columns = df[['Column1', 'Column2']]

# Show the extracted columns
print(extracted_columns)

# If you want to save the extracted columns to a new Excel file
extracted_columns.to_excel('extracted_columns.xlsx', index=False)
```

Replace 'path_to_your_excel_file.xlsx' with the actual path to your Excel file and 'Column1' and 'Column2' with the actual names of the columns you're interested in extracting.

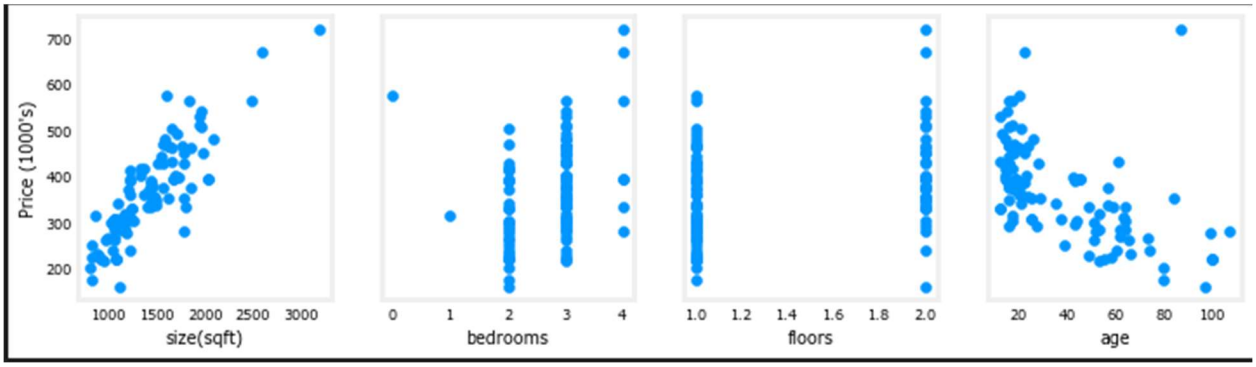
This code snippet does the following:

- Imports the necessary library (pandas).
- Reads the Excel file into a DataFrame, which is a 2-dimensional labeled data structure with columns of potentially different types.
- Extracts the columns you're interested in by specifying their names in a list.
- Prints the extracted columns to the console.
- Optionally, saves the extracted columns to a new Excel file named extracted_columns.xlsx.

Task 1*: Load the Housing dataset and extract the following features from it using the provided sample code. You can edit, modify this code as pre your requirements*.

1.2. View the Dataset

Task 2: Write a Python code to view the loaded dataset. Your code should output the following similar plots:



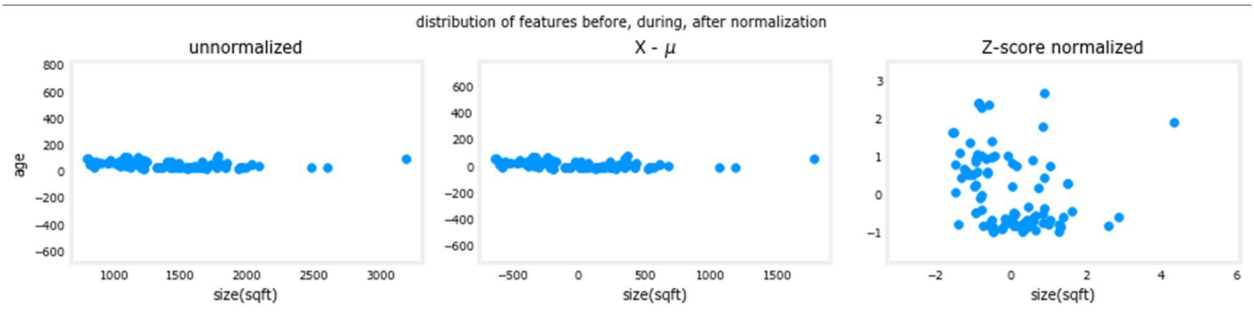
1.3. Learning Rate

Task 3: *Use your previously written Python codes to investigate the impact of learning rate α on the cost. Use the following three values of α and plot the graph between iterations vs cost.:*

a. $\alpha = 9.9e^{-7}$
b. $\alpha = 9e^{-7}$
c. $\alpha = 1e^{-7}$

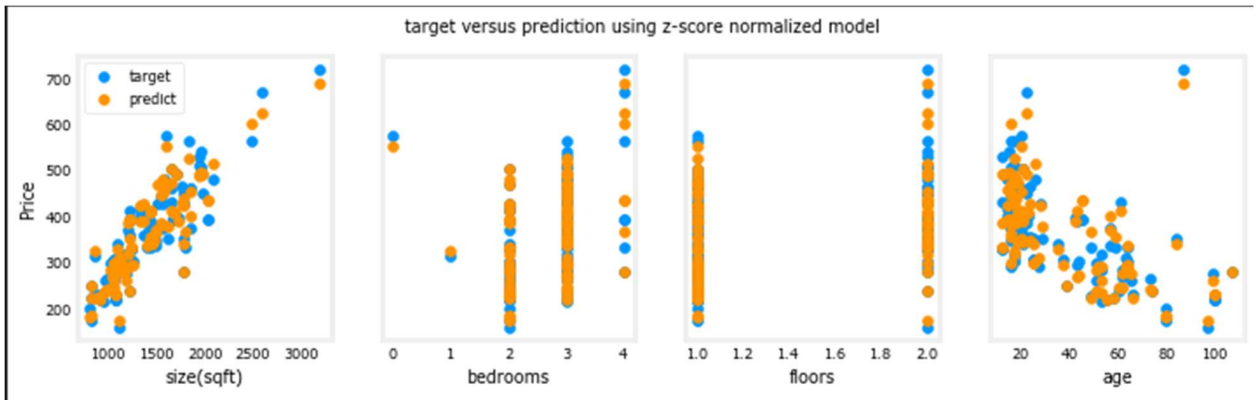
1.4. Feature Scaling

Task 4: *Using the dataset imported in part 1 of this task, write a Python code to implement feature scaling using the Z-score normalization. Your code should output the plots similar to the following figures:*



Task 5: *Re-run the gradient descent algorithm with the normalized features. Use* $\alpha = 1e^{-7}$ *to obtain the normalized parameters of the model and compare the speed of convergence between normalized and unnormalized data*.

Task 6: *Using the normalized features and parameters, predict the target variable. In addition, your code should also return the plots similar to the following figures.*



1.5. Prediction

Task 7: *The point of generating our model is to use it to predict housing prices that are not in the data set. Write a Python code to predict the price of a house with 1200 sqft, 3 bedrooms, 1 floor, 40 years old. Recall, that you must normalize the data with the mean and standard deviation derived when the training data was normalized.*

Part 2: Feature Engineering and Polynomial Regression

Goals

- explore feature engineering and polynomial regression which allows you to use the machinery of linear regression to fit very complicated, even very non-linear functions.

Tools

You will utilize the function developed in previous labs as well as matplotlib and NumPy.

```
import numpy as np
import matplotlib.pyplot as plt
from lab_utils_multi import zscore_normalize_features, run_gradient_descent_feng
np.set_printoptions(precision=2)  # reduced display precision on numpy arrays
```

2. Polynomial Features

Let's try using what we know so far to fit a non-linear curve. We'll start with a simple quadratic:

$$y = 1 + x^2$$

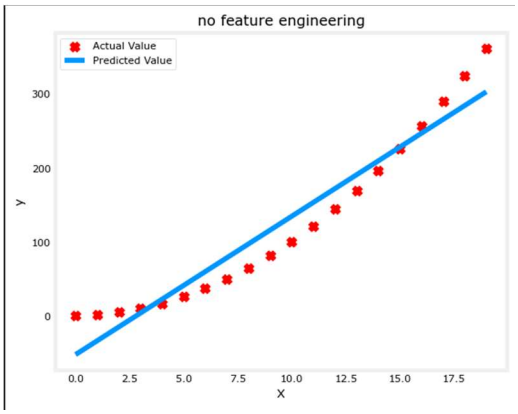
You're familiar with all the routines we're using. They are available in the lab_utils.py file for review. We'll use [`np.c_[..]`](https://numpy.org/doc/stable/reference/generated/numpy.c_.html) which is a NumPy routine to concatenate along the column boundary.

```
# create target data
x = np.arange(0, 20, 1)
y = 1 + x**2
X = x.reshape(-1, 1)
```

```
model_w,model_b = run_gradient_descent_feng(X,y,iterations=1000, alpha = 1e-2)


plt.scatter(x, y, marker='x', c='r', label="Actual Value"); plt.title("no feature engineering")
plt.plot(x,X@model_w + model_b, label="Predicted Value"); plt.xlabel("X"); plt.ylabel("y"); plt.legend(); plt.show()
```

Run the above code. It should produce the following output plot:



Well, as expected, not a great fit. What is needed is something like a polynomial feature.

To accomplish this, you can modify the input data to engineer the needed features. If you swap the original data with a version that squares the x value, then you can achieve $y = w_0 x_0^2 + b$. Let's try it. Swap `X` for `X**2` below:

```
# create target data
x = np.arange(0, 20, 1)
y = 1 + x**2

# Engineer features
X = x**2      #<-- added engineered feature
```

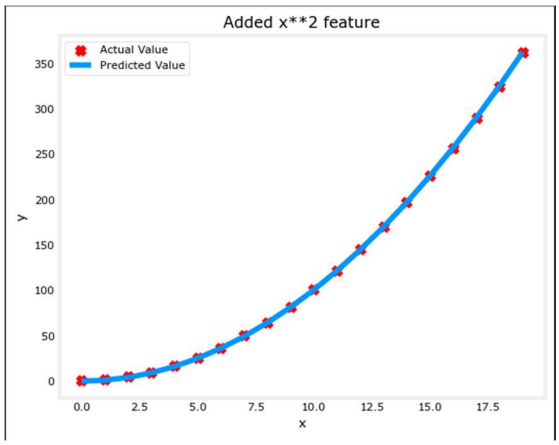
```
X = X.reshape(-1, 1)  #X should be a 2-D Matrix

model_w,model_b = run_gradient_descent_feng(X, y, iterations=10000,
alpha = 1e-5)


plt.scatter(x, y, marker='x', c='r', label="Actual Value");
plt.title("Added x**2 feature")

plt.plot(x, np.dot(X,model_w) + model_b, label="Predicted Value");
plt.xlabel("x"); plt.ylabel("y"); plt.legend(); plt.show()
```

After running the above code, the following output graph will be produced:



Part 3: Linear Regression Using Scikit-Learn

Goals

- Utilize scikit-learn to implement linear regression using Gradient Descent

Tools

You will utilize functions from scikit-learn as well as matplotlib and NumPy.

```
import numpy as np
np.set_printoptions(precision=2)
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
dlblue = '#0096ff'; dlorange = '#FF9300'; dldarkred='#C00000'; dlmagenta='#FF40FF'; dlpurple='#7030A0';
plt.style.use('deeplearning.mplstyle')
```

3.1. Gradient Descent

Scikit-learn has a gradient descent regression model [sklearn.linear_model.SGDRegressor] (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#examples-using-sklearn-linear-model-sgdregressor).

Like your previous implementation of gradient descent, this model performs best with normalized inputs. [sklearn.preprocessing.StandardScaler]

(https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler) will perform z-score normalization as in a previous lab. Here it is referred to as 'standard score'.

3.2. Load the data set

#Load the dataset utilized in part 1 and define the X_train, y_train from the Housing dataset

```
X_features = ['size(sqft)','bedrooms','floors','age']
```

3.3. Scale/normalize the training data

```
scaler = StandardScaler()
X_norm = scaler.fit_transform(X_train)
print(f"Peak to Peak range by column in Raw X:{np.ptp(X_train,axis=0)}")
print(f"Peak to Peak range by column in Normalized X:{np.ptp(X_norm,axis=0)}")
```

3.4. Create and fit the regression model

```
sgdr = SGDRegressor(max_iter=1000)
sgdr.fit(X_norm, y_train)
print(sgdr)
print(f"number of iterations completed: {sgdr.n_iter_}, number of weight updates: {sgdr.t_}")
```

3.5. View parameters

Note, the parameters are associated with the *normalized* input data. The fit parameters are very close to those found in the previous lab with this data.

```
b_norm = sgdr.intercept_
w_norm = sgdr.coef_
print(f"model parameters:                   w: {w_norm}, b:{b_norm}")
print(f"model parameters from previous lab: w: [110.56 -21.27 -32.71 -37.97], b: 363.16")
```

3.6. Make predictions

Predict the targets of the training data. Use both the `predict` routine and compute using w and b.

```
# make a prediction using sgdr.predict()
y_pred_sgd = sgdr.predict(X_norm)
# make a prediction using w,b.
y_pred = np.dot(X_norm, w_norm) + b_norm
print(f"prediction using np.dot() and sgdr.predict match: {(y_pred == y_pred_sgd).all()}")


print(f"Prediction on training set:\n{y_pred[:4]}" )
print(f"Target values \n{y_train[:4]}")
```

3.7. Plot Results

Let's plot the predictions versus the target values.

```
# plot predictions and targets vs original features
fig,ax=plt.subplots(1,4,figsize=(12,3),sharey=True)
for i in range(len(ax)):
    ax[i].scatter(X_train[:,i],y_train, label = 'target')
    ax[i].set_xlabel(X_features[i])
    ax[i].scatter(X_train[:,i],y_pred,color=dlorange,      label      = 'predict')
ax[0].set_ylabel("Price"); ax[0].legend();
fig.suptitle("target versus prediction using z-score normalized model")
plt.show()
```