**Regression Analysis**

**Linear Regression**

Linear regression is probably one of the most important and widely used regression techniques. It’s among the simplest regression methods. One of its main advantages is the ease of interpreting results.

Problem Formulation

When implementing linear regression of some dependent variable 𝑦 on the set of independent variables 𝐱 = (𝑥₁, …, 𝑥ᵣ), where 𝑟 is the number of predictors, you assume a linear relationship between 𝑦 and 𝐱: 𝑦 = 𝛽₀ + 𝛽₁𝑥₁ + ⋯ + 𝛽ᵣ𝑥ᵣ + 𝜀.

This equation is the **regression equation**. 𝛽₀, 𝛽₁, …, 𝛽ᵣ are the **regression coefficients**, and 𝜀 is the **random error**.

Linear regression calculates the **estimators** of the regression coefficients or simply the **predicted weights**, denoted with 𝑏₀, 𝑏₁, …, 𝑏ᵣ. They define the **estimated regression function** 𝑓(𝐱) = 𝑏₀ + 𝑏₁𝑥₁ + ⋯ + 𝑏ᵣ𝑥ᵣ. This function should capture the dependencies between the inputs and output sufficiently well.

The **estimated** or **predicted response**, 𝑓(𝐱ᵢ), for each observation 𝑖 = 1, …, 𝑛, should be as close as possible to the corresponding **actual response** 𝑦ᵢ. The differences 𝑦ᵢ - 𝑓(𝐱ᵢ) for all observations 𝑖 = 1, …, 𝑛, are called the **residuals**.

Regression is about determining the **best predicted weights**, that is the weights corresponding to the smallest residuals.

To get the best weights, you usually **minimize the sum of squared residuals** (SSR) for all observations 𝑖 = 1, …, 𝑛: SSR = Σᵢ(𝑦ᵢ - 𝑓(𝐱ᵢ))². This approach is called the **method of ordinary least squares**.

Regression Performance

The variation of actual responses 𝑦ᵢ, 𝑖 = 1, …, 𝑛, occurs partly due to the dependence on the predictors 𝐱ᵢ. However, there is also an additional inherent variance of the output. The **coefficient of determination**, denoted as 𝑅², tells you which amount of variation in 𝑦 can be explained by the dependence on 𝐱 using the particular regression model. Larger 𝑅² indicates a better fit and means that the model can better explain the variation of the output with different inputs.

The value 𝑅² = 1 corresponds to SSR = 0, that is to the **perfect fit** since the values of predicted and actual responses fit completely to each other.

Simple Linear Regression

Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, 𝐱 = 𝑥.

The following figure illustrates simple linear regression:



The estimated regression function (black line) has the equation 𝑓(𝑥) = 𝑏₀ + 𝑏₁𝑥. Your goal is to calculate the optimal values of the predicted weights 𝑏₀ and 𝑏₁ that minimize SSR and determine the estimated regression function. The value of 𝑏₀, also called the **intercept**, shows the point where the estimated regression line crosses the 𝑦 axis. It is the value of the estimated response 𝑓(𝑥) for 𝑥 = 0. The value of 𝑏₁ determines the **slope** of the estimated regression line.

The predicted responses (red squares) are the points on the regression line that correspond to the input values. For example, for the input 𝑥 = 5, the predicted response is 𝑓(5) = 8.33 (represented with the leftmost red square).

The residuals (vertical dashed gray lines) can be calculated as 𝑦ᵢ - 𝑓(𝐱ᵢ) = 𝑦ᵢ - 𝑏₀ - 𝑏₁𝑥ᵢ for 𝑖 = 1, …, 𝑛. They are the distances between the green circles and red squares. When you implement linear regression, you are actually trying to minimize these distances and make the red squares as close to the predefined green circles as possible.

**Multivariate (Multiple) Linear Regression**

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables. If there are just two independent variables, the estimated regression function is 𝑓(𝑥₁, 𝑥₂) = 𝑏₀ + 𝑏₁𝑥₁ + 𝑏₂𝑥₂. It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights 𝑏₀, 𝑏₁, and 𝑏₂ such that this plane is as close as possible to the actual responses and yield the minimal SSR. The case of more than two independent variables is similar, but more general. The estimated regression function is 𝑓(𝑥₁, …, 𝑥ᵣ) = 𝑏₀ + 𝑏₁𝑥₁ + ⋯ +𝑏ᵣ𝑥ᵣ, and there are 𝑟 + 1 weights to be determined when the number of inputs is 𝑟.

Underfitting and Overfitting

One very important question that might arise when you’re implementing polynomial regression is related to **the choice of the optimal degree of the polynomial regression function**.There is no straightforward rule for doing this. It depends on the case. You should, however, be aware of twoproblems that might follow the choice of the degree: **underfitting** and **overfitting**.

**Underfitting** occurs when a model can’t accurately capture the dependencies among data, usually as a consequence of its own simplicity. It often yields a low 𝑅² with known data and bad generalization capabilities when applied with new data.

**Overfitting** happens when a model learns both dependencies among data and random fluctuations. In other words, a model learns the existing data too well. Complex models, which have many features or terms, are often prone to overfitting. When applied to known data, such models usually yield high 𝑅². However, they often don’t generalize well and have significantly lower 𝑅² when used with new data. The next figure illustrates the underfitted, well-fitted, and overfitted models:

The next figure illustrates the underfitted, well-fitted, and overfitted models:





The top left plot shows a linear regression line that has a low 𝑅². It might also be important that a straight line can’t take into account the fact that the actual response increases as 𝑥 moves away from 25 towards zero. This is likely an example of underfitting. The top right plot illustrates polynomial regression with the degree equal to 2. In this instance, this might be the optimal degree for modeling this data. The model has a value of 𝑅² that is satisfactory in many cases and shows trends nicely.

The bottom left plot presents polynomial regression with the degree equal to 3. The value of 𝑅² is higher than in the preceding cases. This model behaves better with known data than the previous ones. However, it shows some signs of overfitting, especially for the input values close to 60 where the line starts decreasing, although actual data don’t show that. Finally, on the bottom right plot, you can see the perfect fit: six points and the polynomial line of the degree 5 (or higher) yield 𝑅² = 1.



R-squared is possibly the most important measurement. R-squared is the measurement of how much of the dependent variable is explained by changes in our independent variables. In percentage terms, 0.862 would mean our model explains 86.2% of the change in our ‘y’ variable.



**Polynomial Regression: a. In Sample Testing – b. Out of Sample Testing**

### **Overfitting** is one of the most serious kinds of problems related to machine learning. It occurs when a model learns the training data too well. The model then learns not only the relationships among data but also the noise in the dataset. Overfitted models tend to have good performance with the data used to fit them (the training data), but they behave poorly with unseen data (or test data, which is data not used to fit the model)



**Simple Linear Regression and Graphical Representation**

**(with 1 independent variable x)**

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Load the dataset

data = pd.read\_csv("C:\\Kostas\\python-3.10.6-embed-amd64\\python\_code2\\ml\\RS1L.csv")

# Use only one feature

X = data['Open']

# Dependent variable Y

Y = data['Close']

# divide the data into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=0)

# Split the data into training/testing sets

X\_train2 = np.array(X\_train).reshape((-1, 1))

X\_test2 = np.array(X\_test).reshape((-1, 1))

# Create linear regression object

regr = linear\_model.LinearRegression()

# Train the model using the training sets

regr.fit(X\_train2, Y\_train)

# Make predictions using the testing set

Y\_pred = regr.predict(X\_test2)

# The coefficients

print("Coefficients: \n", regr.coef\_)

# The mean squared error

print("Mean squared error: %.2f" % mean\_squared\_error(Y\_test, Y\_pred))

# The mean absolute error

print("Mean absolute error: %.2f" % mean\_absolute\_error(Y\_test, Y\_pred))

# The coefficient of determination: 1 is perfect prediction

print("Coefficient of determination: %.2f" % r2\_score(Y\_test, Y\_pred))

# Create a dataframe

df = pd.DataFrame({'Real Values':Y\_test, 'Predicted Values':Y\_pred})

# Save the real values and the predicted values in a csv file

df.to\_csv("C:\Kostas\python-3.10.6-embed-amd64\python\_code2\ml\\Output.csv", encoding='utf-8', index=False)

df

# Plot outputs

plt.scatter(X\_test2, Y\_test, color="black")

plt.plot(X\_test2, Y\_pred, color="blue", linewidth=3)

plt.xticks(())

plt.yticks(())

plt.show()

------------------------------------------------------------------------

**Multiple Linear Regression and Graphical Representation**

**(with 2 independent variables x1 and x2)**

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.linear\_model import LinearRegression

# Load the dataset

data = pd.read\_csv("C:\\Kostas\\python-3.10.6-embed-amd64\\python\_code2\\ml\\RS1L.csv")

# Dependent variable Y

Y = data['Close']

print(Y)

# Independent variables X1 and X2

X = data[['Open', 'Volume']].to\_numpy()

X1 = data['Open']

print(X1)

X2 = data['Volume']

print(X2)

# create training and test splits

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.20)

X\_train2 = np.array(X\_train).reshape((-1, 2))

X\_test2 = np.array(X\_test).reshape((-1, 2))

X1\_test = X\_test2[:, 0]

X2\_test = X\_test2[:, 1]

x1\_pred = np.linspace(80, 1300, 30) # range of open values np.linspace(min, max, datapoints)

x2\_pred = np.linspace(10000, 3000000, 30) # range of volume values

xx1\_pred, xx2\_pred = np.meshgrid(x1\_pred, x2\_pred)

# model\_viz = np.array([xx1\_pred.flatten(), xx2\_pred.flatten()]).T

# Create a model and fit it

ols = linear\_model.LinearRegression()

model = ols.fit(X\_train2, Y\_train)

# Use the model for making Predictions

Y\_pred = model.predict(X\_test2)

print('predicted response:', Y\_pred, sep='\n')

# Create a dataframe

df = pd.DataFrame({'Real Values':Y\_test, 'Predicted Values':Y\_pred})

# Save the real values and the predicted values in a csv file

df.to\_csv("C:\Kostas\python-3.10.6-embed-amd64\python\_code2\ml\\Output2.csv", encoding='utf-8', index=False)

df

# Get results

r2 = model.score(X\_test2, Y\_pred)

print('coefficient of determination:', r2)

print('intercept:', model.intercept\_)

print('coefficients:', model.coef\_)

plt.style.use('default')

fig = plt.figure(figsize=(12, 4))

ax1 = fig.add\_subplot(131, projection='3d')

ax2 = fig.add\_subplot(132, projection='3d')

ax3 = fig.add\_subplot(133, projection='3d')

axes = [ax1, ax2, ax3]

for ax in axes:

 ax.plot(X1\_test, X2\_test, Y\_test, color='k', zorder=15, linestyle='none', marker='o', alpha=0.5)

 ax.scatter(X1\_test, X2\_test, Y\_pred, facecolor=(0,0,0,0), s=20, edgecolor='#70b3f0')

 ax.set\_xlabel('X1', fontsize=12)

 ax.set\_ylabel('X2', fontsize=12)

 ax.set\_zlabel('Y', fontsize=12)

 ax.locator\_params(nbins=4, axis='x')

 ax.locator\_params(nbins=5, axis='x')

ax1.view\_init(elev=28, azim=120)

ax2.view\_init(elev=4, azim=114)

ax3.view\_init(elev=60, azim=165)

fig.suptitle('$R^2 = %.2f$' % r2, fontsize=20)

fig.tight\_layout()