ep01_linreg_analytic

April 4, 2022

[1]:
```
name = "Thiago Jose Benitez Pena"  # write YOUR NAME

honorPledge = "I affirm that I have not given or received any unauthorized " \
              "help on this assignment, and that this work is my own.\n"


print("\nName: ", name)
print("\nHonor pledge: ", honorPledge)
```

```
Name:  Thiago Jose Benitez Pena

Honor pledge:  I affirm that I have not given or received any unauthorized help
on this assignment, and that this work is my own.
```

1 MAC0460 / MAC5832 (2022)

2 EP2: Linear regression - analytic solution

2.0.1 Objectives:

- to implement and test the analytic solution for the linear regression task (see, for instance, Slides of Lecture 03 of *Learning from Data*)
- to understand the core idea (*optimization of a loss or cost function*) for parameter adjustment in machine learning

2.0.2 What to do:

- some cells of this notebook must be filled. Places to be filled are indicated as: # START OF YOUR CODE:

  # END OF YOUR CODE

3 Linear regression

Given a dataset $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$ with $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$, we would like to approximate the unknown function $f : \mathbb{R}^d \to \mathbb{R}$ (recall that $y^{(i)} = f(\mathbf{x}^{(i)})$) by means of a linear model

h:

$$h(\mathbf{x}^{(i)}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x}^{(i)} + b$$

Note that $h(\mathbf{x}^{(i)}; \mathbf{w}, b)$ is, in fact, an affine transformation of $\mathbf{x}^{(i)}$. As commonly done, we will use the term "linear" to refer to an affine transformation.

The output of h is a linear transformation of $\mathbf{x}^{(i)}$. We use the notation $h(\mathbf{x}^{(i)}; \mathbf{w}, b)$ to make clear that h is a parametric model, i.e., the transformation h is defined by the parameters $\mathbf{w}$ and b. We can view vector $\mathbf{w}$ as a *weight* vector that controls the effect of each *feature* in the prediction.

By adding one component with value equal to 1 to the observations $\mathbf{x}$ (an artificial coordinate), we have:

$$\tilde{\mathbf{x}} = (1, x_1, \ldots, x_d) \in \mathbb{R}^{1+d}$$

and then we can simplify the notation:

$$h(\mathbf{x}^{(i)}; \mathbf{w}) = \hat{y}^{(i)} = \mathbf{w}^\top \tilde{\mathbf{x}}^{(i)}$$

We would like to determine the optimal parameters $\mathbf{w}$ such that prediction $\hat{y}^{(i)}$ is as closest as possible to $y^{(i)}$ according to some error metric. Adopting the *mean square error* as such metric we have the following cost function:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \big(\hat{y}^{(i)} - y^{(i)}\big)^2 \tag{1}$$

Thus, the task of determining a function h that is closest to f is reduced to the task of finding the values $\mathbf{w}$ that minimize $J(\mathbf{w})$.

3.1 Some imports

[2]:
```
import numpy as np
import time
import matplotlib.pyplot as plt

%matplotlib inline
```

3.1.1 Auxiliary functions

The two auxiliary functions below are for generating simulated data and for plotting data.

[3]:
```
# An auxiliary function
def get_housing_prices_data(N, verbose=True):
    """
    Generates artificial linear data,
    where x = square meter, y = house price

    :param N: data set size
```

```
    :type N: int

    :param verbose: param to control print
    :type verbose: bool
    :return: design matrix, regression targets
    :rtype: np.array, np.array
    """
    cond = False
    while not cond:
        x = np.linspace(90, 1200, N)
        gamma = np.random.normal(30, 10, x.size)
        y = 50 * x + gamma * 400
        x = x.astype("float32")
        x = x.reshape((x.shape[0], 1))
        y = y.astype("float32")
        y = y.reshape((y.shape[0], 1))
        cond = min(y) > 0

    xmean, xsdt, xmax, xmin = np.mean(x), np.std(x), np.max(x), np.min(x)
    ymean, ysdt, ymax, ymin = np.mean(y), np.std(y), np.max(y), np.min(y)
    if verbose:
        print("\nX shape = {}".format(x.shape))
        print("y shape = {}\n".format(y.shape))
        print("X: mean {}, sdt {:.2f}, max {:.2f}, min {:.2f}".format(xmean,
                                                               xsdt,
                                                               xmax,
                                                               xmin))
        print("y: mean {:.2f}, sdt {:.2f}, max {:.2f}, min {:.2f}".format(ymean,
                                                                 ysdt,
                                                                 ymax,
                                                                 ymin))
    return x, y
```

[4]:

```
# Another auxiliary function
def plot_points_regression(x,
                           y,
                           title,
                           xlabel,
                           ylabel,
                           prediction=None,
                           legend=False,
                           r_squared=None,
                           position=(90, 100)):
    """
    Plots the data points and the prediction,
    if there is one.
```

```
    :param x: design matrix
    :type x: np.array
    :param y: regression targets
    :type y: np.array
    :param title: plot's title
    :type title: str
    :param xlabel: x axis label
    :type xlabel: str
    :param ylabel: y axis label
    :type ylabel: str
    :param prediction: model's prediction
    :type prediction: np.array
    :param legend: param to control print legends
    :type legend: bool
    :param r_squared: r^2 value
    :type r_squared: float
    :param position: text position
    :type position: tuple
    """
    fig, ax = plt.subplots(1, 1, figsize=(8, 8))
    line1, = ax.plot(x, y, 'bo', label='Real data')
    if prediction is not None:
        line2, = ax.plot(x, prediction, 'r', label='Predicted data')
        if legend:
            plt.legend(handles=[line1, line2], loc=2)
        ax.set_title(title,
                 fontsize=20,
                 fontweight='bold')
    if r_squared is not None:
        bbox_props = dict(boxstyle="square,pad=0.3",
                          fc="white", ec="black", lw=0.2)
        t = ax.text(position[0], position[1], "$R^2 ={:.4f}$".format(r_squared),
                    size=15, bbox=bbox_props)

    ax.set_xlabel(xlabel, fontsize=20)
    ax.set_ylabel(ylabel, fontsize=20)
    plt.show()
```

3.1.2 The dataset and the task

The first dataset we will use is a toy dataset. We will generate $N = 100$ observations with only one *feature* and a real value associated to each of them. We can view these observations as being pairs *(area of a real state in square meters, price of the real state)*. Our task is to construct a model that is able to predict the price of a real state, given its area.

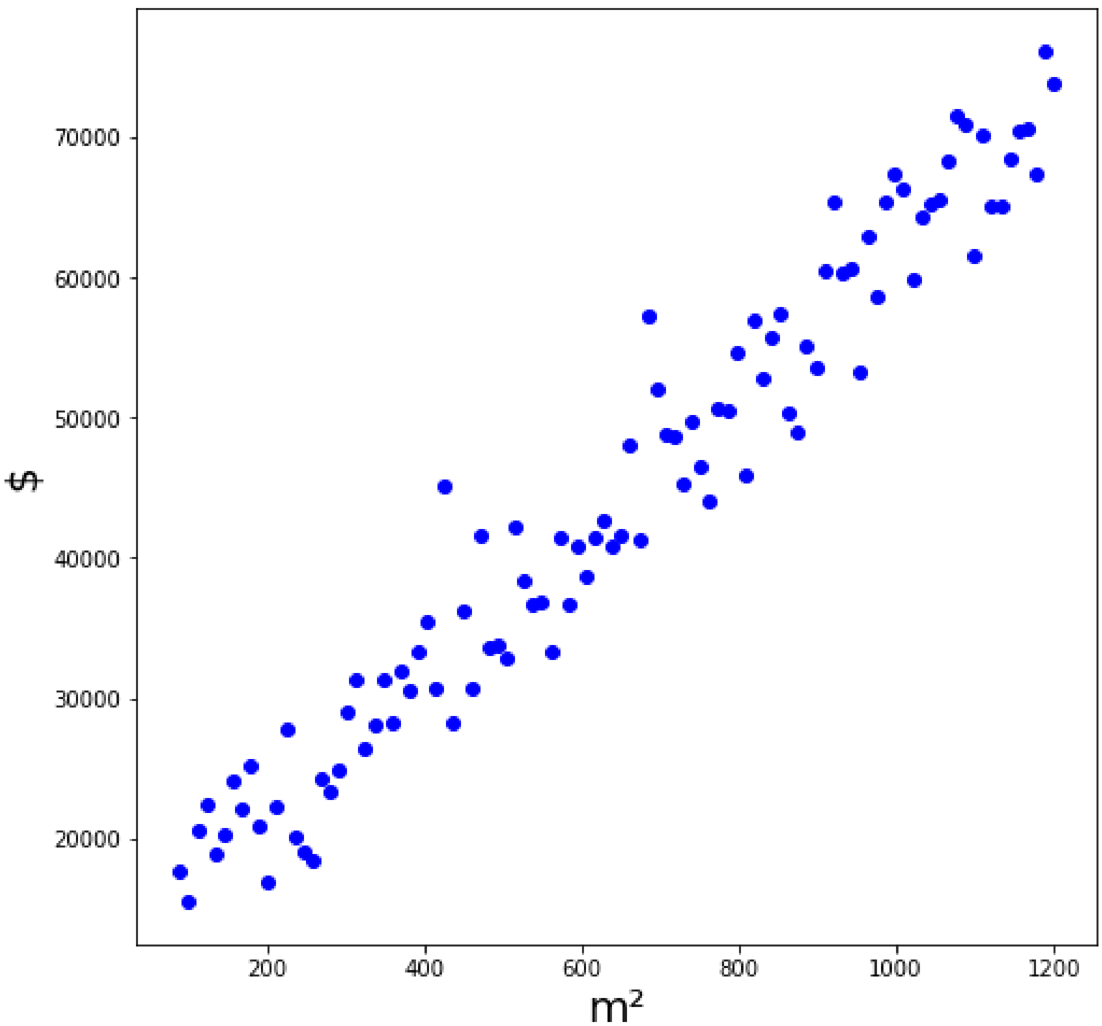
[5]:
```
X, y = get_housing_prices_data(N=100)
```

```
X shape = (100, 1)
y shape = (100, 1)

X: mean 645.0, sdt 323.65, max 1200.00, min 90.00
y: mean 44241.01, sdt 16764.85, max 76210.89, min 15484.04
```

3.1.3 Ploting the data

[6]:
```
plot_points_regression(X,
                       y,
                       title='Real estate prices prediction',
                       xlabel="m\u00b2",
                       ylabel='$')
```



3.1.4 The solution

Given $f : \mathbb{R}^{N\times M} \to \mathbb{R}$ and $\mathbf{A} \in \mathbb{R}^{N\times M}$, we define the gradient of f with respect to $\mathbf{A}$ as:

$$\nabla_{\mathbf{A}} f = \frac{\partial f}{\partial \mathbf{A}} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{A}_{1,1}} & \cdots & \frac{\partial f}{\partial \mathbf{A}_{1,m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial \mathbf{A}_{n,1}} & \cdots & \frac{\partial f}{\partial \mathbf{A}_{n,m}} \end{bmatrix}$$

Let $\mathbf{X} \in \mathbb{R}^{N\times(1+d)}$ be a matrix (sometimes also called the *design matrix*) whose rows are the extended observations of the dataset and let $\mathbf{y} \in \mathbb{R}^N$ be the vector consisting of all values $y^{(i)}$ (i.e., $\mathbf{X}^{(i,:)} = \tilde{\mathbf{x}}^{(i)}$ and $\mathbf{y}^{(i)} = y^{(i)}$). It can be verified that:

$$J(\mathbf{w}) = \frac{1}{N}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \tag{2}$$

Using basic matrix derivative concepts we can compute the gradient of $J(\mathbf{w})$ with respect to $\mathbf{w}$:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{2}{N}(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}) \tag{3}$$

Thus, when $\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$ we have

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y} \tag{4}$$

Hence,

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{5}$$

Note that this solution has a high computational cost. As the number of variables (*features*) increases, the cost for matrix inversion becomes prohibitive. See this text for more details.

3.2 NumPy

Quoted from NumPy documentation: "NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more."

A quick introduction to this library can be found here. Particularly useful for this EP (and this course) are the "array math" related tools.

4 Exercise 1

The objective of this exercise is to apply the solution just described on the dataset above created.

Using only **NumPy**, complete the two functions below. Recall that $\mathbf{X} \in \mathbb{R}^{N \times d}$; thus you will need to add a component of value 1 to each of the observations in $\mathbf{X}$ before performing the computation described above.

NOTE: Although the dataset above has data of dimension $d = 1$, your code must be generic (it should work for $d \geq 1$)

4.1 1.1. Weight computation function

[7]:
```
def normal_equation_weights(X, y):
    """
    Calculates the weights of a linear function using the normal equation␣
 ↪method.
    You should add into X a new column with 1s.

    :param X: design matrix
    :type X: np.ndarray(shape=(N, d))
    :param y: regression targets
    :type y: np.ndarray(shape=(N, 1))
    :return: weight vector
    :rtype: np.ndarray(shape=(1+d, 1))
    """

    # START OF YOUR CODE:
    X1 = np.c_[ np.ones(len(X)), X]
    X2 = np.dot(X1.T, X1)
    X2_inverse = np.linalg.inv(X2)
    return X2_inverse.dot(X1.T).dot(y)
    # END OF YOUR CODE
```

[8]:
```
# test of function normal_equation_weights()

w = 0  # this is not necessary
w = normal_equation_weights(X, y)
print("Estimated w =\n", w)
```

```
Estimated w =
 [[11678.36513322]
 [   50.48471778]]
```

4.2 1.2. Prediction function

[9]:
```python
def normal_equation_prediction(X, w):
    """
    Calculates the prediction over a set of observations X using the linear␣
 ↪function
    characterized by the weight vector w.
    You should add into X a new column with 1s.

    :param X: design matrix
    :type X: np.ndarray(shape=(N, d))
    :param w: weight vector
    :type w: np.ndarray(shape=(1+d, 1))
    :param y: regression prediction
    :type y: np.ndarray(shape=(N, 1))
    """

    # START OF YOUR CODE:
    X1 = np.c_[ np.ones(len(X)), X]
    return np.sum(w.T*X1, axis=1)
    # END OF YOUR CODE
```

4.3 1.3. Coefficient of determination

We can use the R^2 metric (Coefficient of determination) to evaluate how well the linear model fits the data.

Which ^{2} value would you expect to observe ?

[10]:
```python
from sklearn.metrics import r2_score

# test of function normal_equation_prediction()
prediction = normal_equation_prediction(X, w)

# compute the R2 score using the r2_score function from sklearn
# Replace 0 with an appropriate call of the function

# START OF YOUR CODE:
r_2 = r2_score(y, prediction)
# END OF YOUR CODE

plot_points_regression(X,
                       y,
                       title='Real estate prices prediction',
                       xlabel="m\u00b2",
                       ylabel='$',
                       prediction=prediction,
                       legend=True,
```

```
                                r_squared=r_2)
```

$R^2 = 0.9499$

4.4 Additional tests

Let us compute a prediction for $x = 650$

[11]:
```
# Let us use the prediction function
x = np.asarray([650]).reshape(1,1)
prediction = normal_equation_prediction(x, w)
print("Area = %.2f  Predicted price = %.4f" %(x[0], prediction))
```

```
Area = 650.00  Predicted price = 44493.4317
```

4.5 1.4. Processing time

Experiment with different number of samples N and observe how processing time varies.

Be careful not to use a too large value; it may make jupyter freeze …

[12]:
```
# Add other values for N
# START OF YOUR CODE:
N = [100, 1000, 10**4, 10**5, 10**6]
# END OF YOUR CODE

for i in N:
    init = time.time()
    X, y = get_housing_prices_data(N=i)
    #init = time.time()
    w = normal_equation_weights(X, y)
    prediction = normal_equation_prediction(X,w)
    init = time.time() - init

    print("\nExecution time = {:.8f}(s)\n".format(init))
```

```
X shape = (100, 1)
y shape = (100, 1)

X: mean 645.0, sdt 323.65, max 1200.00, min 90.00
y: mean 43846.25, sdt 16629.38, max 76922.16, min 13135.80

Execution time = 0.00113177(s)


X shape = (1000, 1)
y shape = (1000, 1)

X: mean 645.0, sdt 320.75, max 1200.00, min 90.00
y: mean 44170.61, sdt 16694.57, max 83422.53, min 10312.74

Execution time = 0.01615429(s)


X shape = (10000, 1)
y shape = (10000, 1)

X: mean 645.0000610351562, sdt 320.46, max 1200.00, min 90.00
y: mean 44215.14, sdt 16498.52, max 81260.94, min 2438.79

Execution time = 0.01682615(s)
```

```
X shape = (100000, 1)
y shape = (100000, 1)

X: mean 645.0000610351562, sdt 320.43, max 1200.00, min 90.00
y: mean 44262.67, sdt 16514.24, max 83998.42, min 4475.41

Execution time = 0.11640668(s)


X shape = (1000000, 1)
y shape = (1000000, 1)

X: mean 645.0000610351562, sdt 320.43, max 1200.00, min 90.00
y: mean 44248.29, sdt 16516.98, max 86534.19, min 1082.16

Execution time = 2.62375259(s)
```

5 Exercise 2

In this exercise, the goal is to play with the data we have collected in our first class. Download the data file from here (or directly from e-disciplinas, seção Tarefas). We will also try to explore cases where $d > 1$.

Note that there might be some invalid data entries. It is up to you how you will handle those data. Note that if you decide to do some pre-processing of the dataset, it should be done in this notebook (you are not allowed to edit the CSV datasheet). Feel free to added new cells if that helps to better organize your code.

5.1 Reading the dataset

[13]:
```python
import pandas as pd

# load the dataset
df = pd.read_csv('dataMAC0460_5832.csv')
df.head()
```

[13]:

	Sex	Age	Height	Weight	Shoe number	Trouser number
0	Female	53	154	59	36	40
1	Male	23	170	56	40	38
2	Female	23	167	63	37	40
3	Male	21	178	78	40	40
4	Female	25	153	58	36	38

[14]:
```python
df.describe()
```

[14]:

	Age	Height	Weight	Shoe number
count	202.000000	202.000000	202.000000	202.000000
mean	28.133663	171.084158	72.004950	39.777228
std	11.934604	12.808496	17.093392	2.857281
min	3.000000	65.000000	15.000000	24.000000
25%	21.000000	166.250000	61.000000	38.000000
50%	23.000000	172.500000	70.000000	40.000000
75%	29.750000	178.000000	81.750000	42.000000
max	67.000000	194.000000	159.000000	46.000000

5.1.1 Let's establish 'Weight' as the target variable

[15]:
```
# Our target variable is the Weight
y = df['Weight']
y
```

[15]:
```
0      59
1      56
2      63
3      78
4      58
       ..
197    57
198    68
199    65
200    51
201    62
Name: Weight, Length: 202, dtype: int64
```

5.2 2.1. One feature ($d = 1$)

We will use 'Height' as the input feature and predict the weight

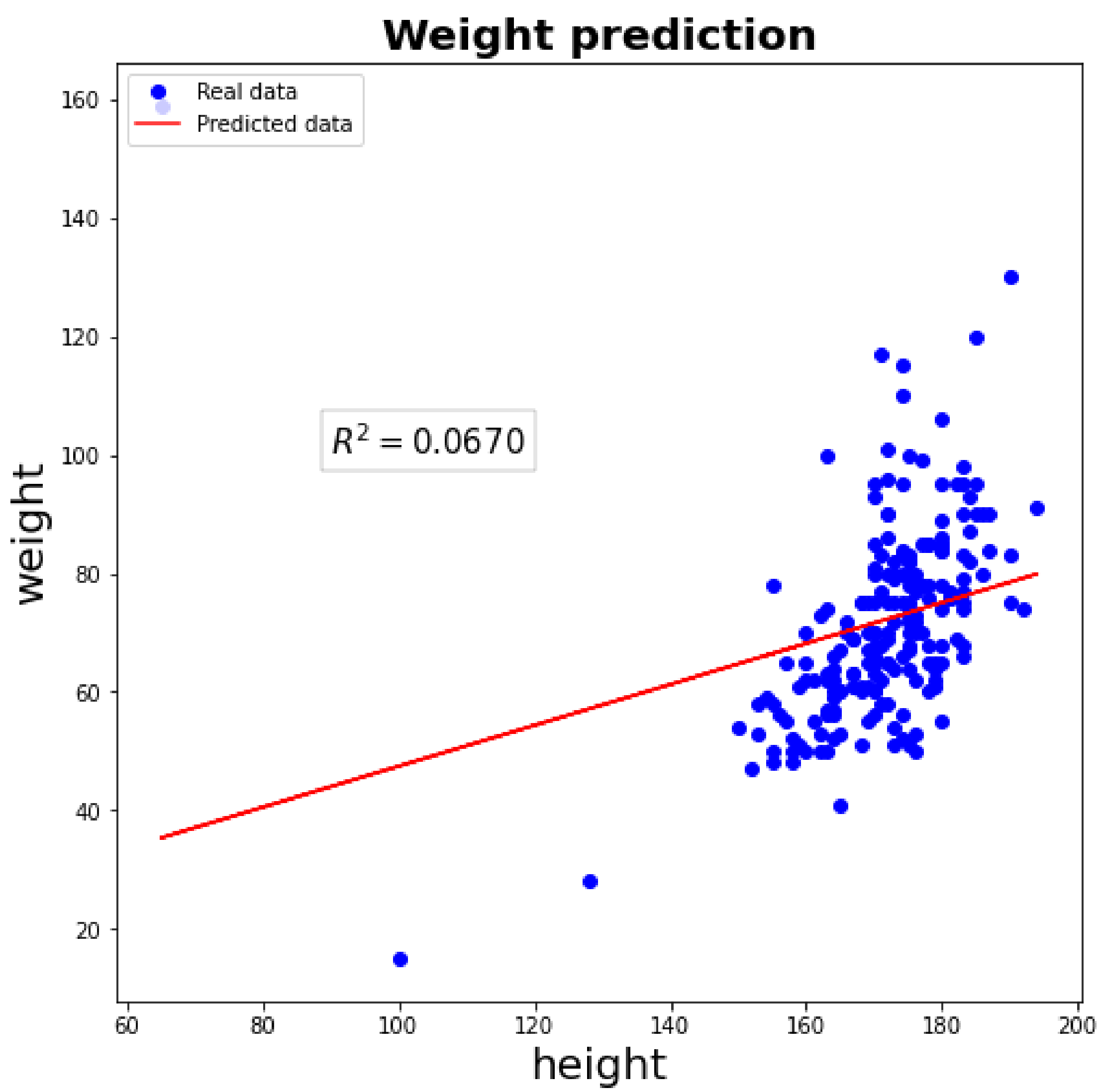
[16]:
```
feature_cols = ['Height']
X = df.loc[:, feature_cols]
X.shape
```

[16]:
```
(202, 1)
```

Write the code for computing the following - compute the regression weights using **X** and **y** - compute the prediction - compute the R^2 value - plot the regression graph (use appropriate values for the parameters of function plot_points_regression())

[17]:
```
# START OF YOUR CODE:
y = y.values
w = normal_equation_weights(X, y)
prediction = normal_equation_prediction(X, w)
```

```
r_2 = r2_score(y, prediction)
plot_points_regression(X,
                       y,
                       title='Weight prediction',
                       xlabel="height",
                       ylabel='weight',
                       prediction=prediction,
                       legend=True,
                       r_squared=r_2)
# END OF YOUR CODE
```



5.3 2.2 - Two input features ($d = 2$)

Now repeat the exercise using as input the features 'Height' and 'Shoe number'

- compute the regression weights using **X** and **y**
- compute the prediction
- compute and print the R^2 value

Note that our plotting function can not be used for this dataset. Here tehre is no need to do the plotting.

[18]:
```python
# START OF YOUR CODE:
feature_cols = ['Height', 'Shoe number']
df2 = df[df['Height'] > 100] # removing outliers
y = df2['Weight'].values
X = df2.loc[:, feature_cols]
w = normal_equation_weights(X, y)
prediction = normal_equation_prediction(X, w)
r_2 = r2_score(y, prediction)
print(r_2)
# END OF YOUR CODE
```

```
0.38966657035500896
```

5.4 2.3 - Three input features $(d = 3)$

Now try with three features. There is no need to do plotting here. - compute the regression weights using **X** and **y** - compute the prediction - compute and print the R^2 value

[19]:
```python
# START OF YOUR CODE:
# Cleaning rows with non numeric values
df2['trouserNumberIsNumeric'] = df2['Trouser number'].apply(lambda x: x.
 ↪isnumeric())
df3 = df2[df2['trouserNumberIsNumeric'] == True].copy()
df3['Trouser number'] = df3['Trouser number'].astype(str).astype(int)


feature_cols = ['Height', 'Shoe number', 'Trouser number']
y = df3['Weight'].values
X = df3.loc[:, feature_cols]
w = normal_equation_weights(X, y)
prediction = normal_equation_prediction(X, w)
r_2 = r2_score(y, prediction)
print(r_2)
# END OF YOUR CODE
```

```
0.6124212350728968
```

```
/tmp/ipykernel_18850/511209252.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df2['trouserNumberIsNumeric'] = df2['Trouser number'].apply(lambda x:
x.isnumeric())
```

6 3. Your comments

Write any comments about your implementation or about the results you observed.

===> Notei que a primeira implementação teve um resultado bastante ruim de r^2=0.0670. A mesma implementação, removendo as linhas com valores de altura menores ou iguais a 100, obtive r^2 = 0.3482.

Na segunda implementação houve uma melhora, obtive r^2=0.3897, com a remoção de outliers.

Para a terceira implementação, tive que remover também as linhas com valores com letra ('P', 'M', 'G'), com isso, obtive uma melhoria substancial em relação à implementação anterior, com r^2= 0.6124.