

Training Linear Models

Performing Linear Regression Using the Normal Equation

Norina Akhtar 19643

Table of Content

Introduction

Design

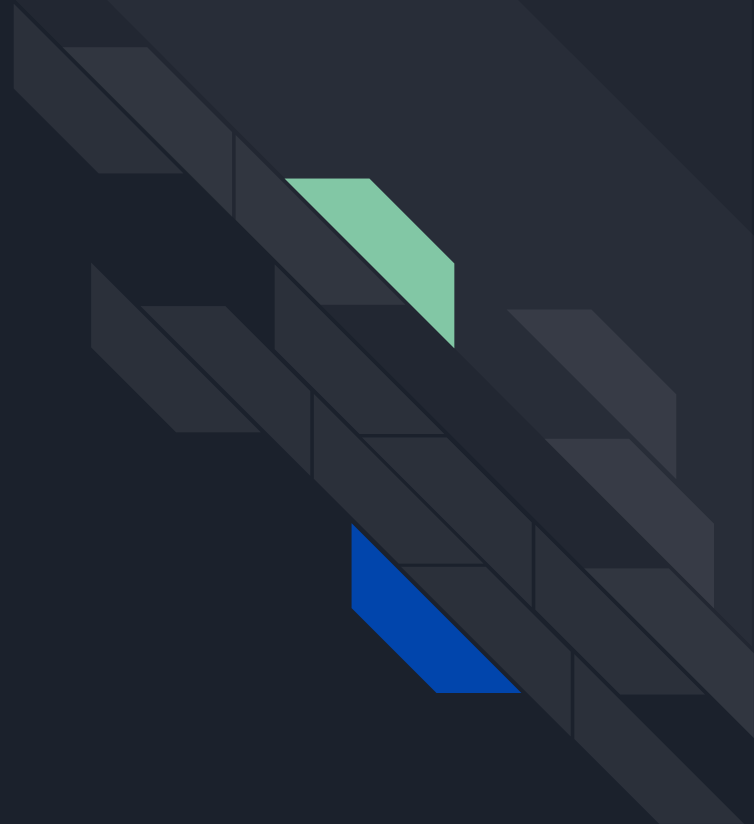
Implementation

Test

Enhancement Ideas

Conclusion

References





Introduction

Linear regression is one of the most important and popular predictive techniques in data analysis.

Its objective is to fit the best line (or a hyper-/plane) to the set of given points (observations) by calculating regression function parameters that minimize specific cost function (error), e.g. mean squared error (MSE).

As a reminder, below there is a linear regression equation in the expanded form.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

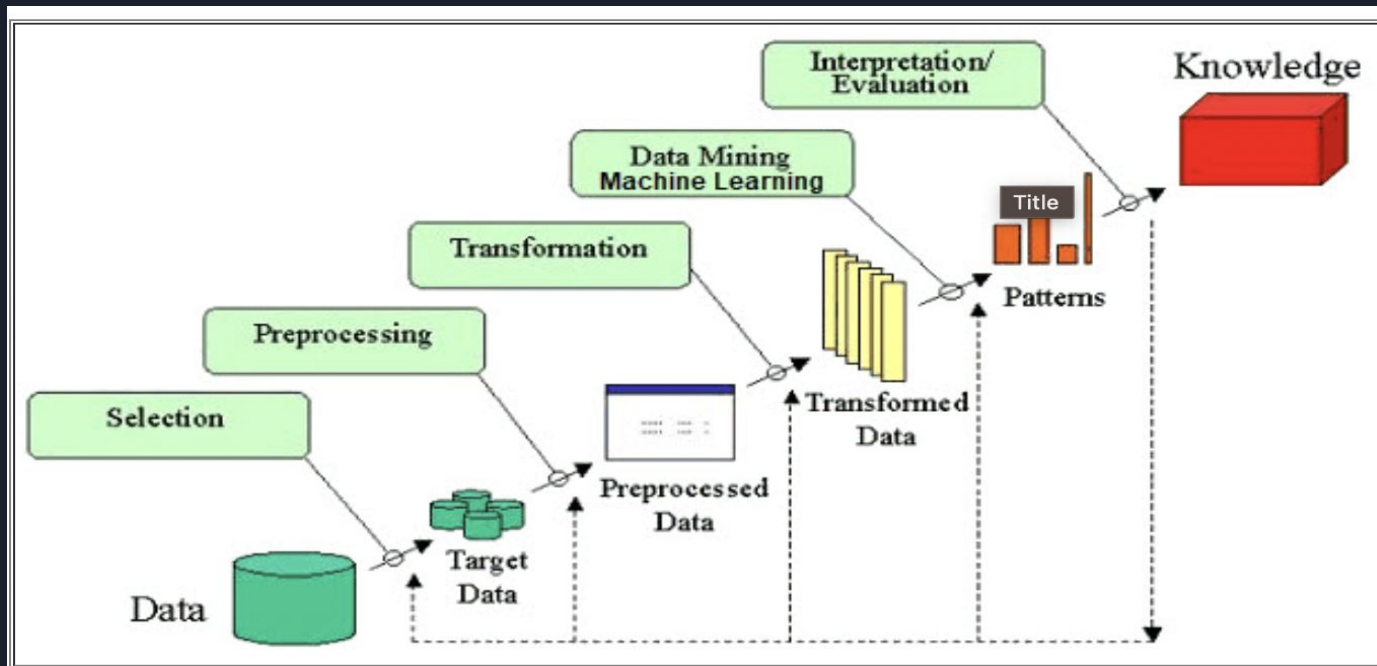
In a vectorized form it looks like that:

$$\hat{y} = \theta^T \cdot x$$

where θ is a vector of parameters weights.

Design

Except for "Preprocessing", this exercise involves all the steps described in the following diagram



Implementation

```
1 import numpy as np
2 import pandas as pd
3 from google.colab import files
4 uploaded = files.upload()
```

Choose Files abalone_train.csv

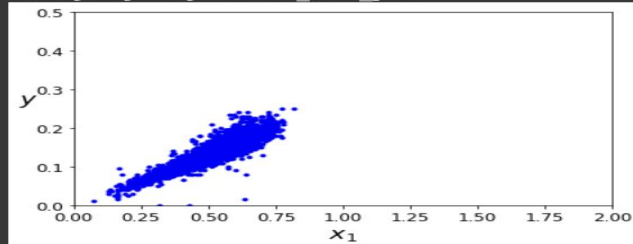
• abalone_train.csv(text/csv) - 145915 bytes, last modified: 1/31/2023 - 100% done
Saving abalone_train.csv to abalone_train (1).csv

```
[ ] 1 import io
2     abalone = pd.read_csv(
3         io.BytesIO(uploaded['abalone_train.csv'])),
4         names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
5              "Viscera weight", "Shell weight", "Age"])
```

```
[ ] 1 X1 = abalone["Length"]
2     X2 = np.array(X1)
3     X = X2.reshape(-1, 1)
4     y1 = abalone["Height"]
5     y2 = np.array(y1)
6     y = y2.reshape(-1, 1)
```

```
1 plt.plot(X, y, "b.")
2 plt.xlabel("$x_1$", fontsize=18)
3 plt.ylabel("$y$", rotation=0, fontsize=18)
4 plt.axis([0, 2, 0, 0.5])
5 save_fig("generated_data_plot")
6 plt.show()
```

Saving figure generated data plot

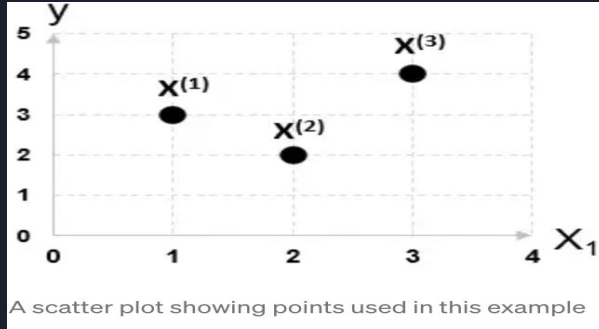


```
[ ] 1 X_b = np.c_[np.ones((len(y), 1)), X]  # add x0 = 1 to each instance
2     theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```
[ ] 1 theta_best
array([[ -0.0108267 ],
       [  0.28716253]])
```

Test

In this example, there are only three points (observations) with only one variable (X_1). On the graph, they look like below.



In this example, there are only three points (observations) with only one variable (X_1). On the graph, they look like below.

$$\hat{y} = \theta_0 + \theta_1 x_1$$

Eq. 4: A linear regression equation for this example

Test using manual solution

Features (X) and labels (y) are:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad y = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix}$$

Default bias, x_0 x_1


Features and Labels matrices

Note that we add a default bias term of 1 — it will be updated during our calculations. Not adding this term will lead to a wrong solution.

Step1 : This is a relatively simple task — rows become new columns.

$$X^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

Step 2: Multiplication on the transposed matrix and matrix X


$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$$
$$X^T \cdot X = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

Step 3: Inversion of a resultant matrix

To inverse a simple 2x2 matrix we can use the formula:

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Therefore , we get

$$(X^T \cdot X)^{-1} = \begin{bmatrix} 7/3 & -1 \\ -1 & 1/2 \end{bmatrix}$$

Step 4: Multiplication of the inverted matrix with X transposed

$$(X^T \cdot X)^{-1} \cdot X^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4/3 & 1/3 & -2/3 \\ -1/2 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} 2/3 & -1 \\ -1 & 1/2 \end{bmatrix}$$

Step 5: Final multiplication to obtain the vector of best parameters

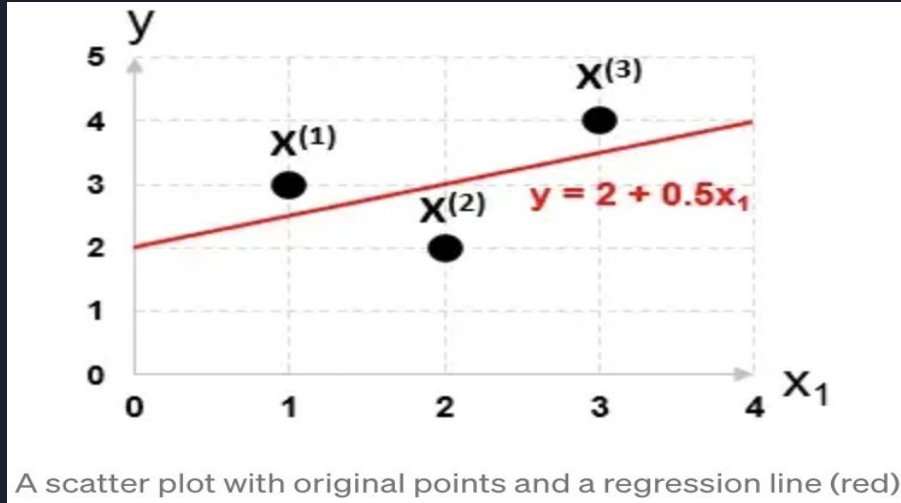
$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 4/3 & 1/3 & -2/3 \\ -1/3 & 0 & 1/2 \end{bmatrix}$$

Final linear regression equation forms like this:

$$\hat{y} = 2 + 0.5x_1$$

Eq. 5: A linear regression equation with the best weights

Plotting this line onto a previous graph looks like below.





Enhancement Ideas

The accuracy of model can be improved following these points:

- Creating simple new features
- Transforming target variable
- Clustering common data points
- Use of boosting algorithms
- Hyperparameter tuning



Conclusion

This approach is calculated just in one step and you don't have to choose the learning rate parameter. Additionally, in terms of memory usage this approach is linear $O(m)$ meaning it stores huge datasets effectively if they only fit into your computer's memory

The problem is in its numerical complexity. Solving this equation requires inverting a matrix and this is a computationally expensive operation — depending on the implementation, in a big O notation it is $O(n^3)$ or slightly less. This means it scales up horribly, practically meaning that when you double number of features, the computational times increases by $2^3 = 8$ times



References

<https://towardsdatascience.com/performing-linear-regression-using-the-normal-equation-6372ed3c57>

https://colab.research.google.com/github/ageron/handson-ml2/blob/master/04_training_linear_models.ipynb#scrollTo=Mqq0UYPxY6yR

https://hc.labnet.sfbu.edu/~henry/sfbu/course/hands_on_ml_with_schikit_2nd/train_model/slide/exercise_train_model.html#colab

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/load_data/csv.ipynb#scrollTo=IZVExo9DKoNz

