Linear Models - Linear Regression

Guru Sarath Thangamani

February 11, 2024

Abstract

I have created this post to give you a comprehensive picture of the linear regression model. Being one of the most fundamental mathematical models, an in-depth understanding of linear regression is crucial to understanding more complex machine-learning / deep-learning models. I have tried my level best and collected information from multiple sources to prepare this content. Let us quickly deep dive into the topic.

1 Linear Regression

The linear regression model estimates a real-valued quantity (Y) based on a set of observed inputs (X).
Two ways to think about the linear regression model?

1. Fitting a straight line through the training dataset. We want to choose the line in such a way that it best predicts the output. The linear regression process finds the best-fit line.
2. Output is a weighted sum of observed input features (predictors) $\hat{y}_i = \sum_{j=0}^{p} x_{ij}\beta_j$. Features with higher weight are more important for predicting the output. The linear regression process finds the weights for each input feature.

Below are the types of linear regression based on the number of inputs and outputs -

1. **Simple Linear Regression**- One predictor (X) and output (Y).
2. **Multiple Linear Regression**- More than one predictor (X).
3. **Multivariate Linear Regression** - More than one output to predict (Y).

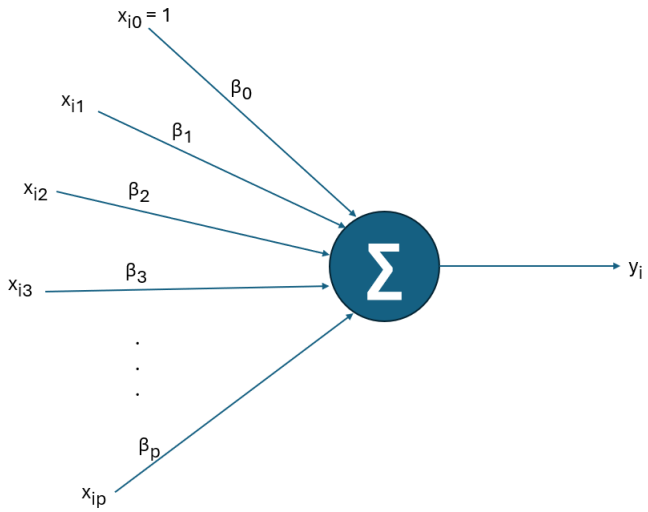

Figure 1: Linear Regression Model

2 Terminologies used in this post

1. X - Input feature matrix. x_{ij} - ith entry and jth feature in the training dataset.
2. y - Single output vector. y_i - output of ith entry in the training dataset.
3. β - weight vector. β_j - weight of the jth feature(input).

2.1 When is it a good idea to use Linear Regression ?

If you somehow know that the output (Y) is some linear combination of the inputs (X), the linear regression procedure solves the problem.
But, oftentimes, we do not know the underlying relationship between input and output. What do we do then? My Suggestion - Linear regression being one of the simplest predictor models, you should always try it first and validate the output. If the fitted linear regression model accurately predicts the output from the test dataset, then you could probably stick to it. A simpler model that performs well on the test dataset is always better.

2.2 Linear Regression - Model

$$f(\boldsymbol{x_i}) = \hat{y}_i = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j$$

Set ($x_{i0} = 1$) and absorb β_0 into the summation.

$$\boxed{f(\boldsymbol{x_i}) = \hat{y}_i = \sum_{j=0}^{p} x_{ij}\beta_j} \tag{1}$$

2.2.1 Solving Linear Regression (Error Metric)

In the gradient approach, we solve the linear regression by minimizing the residual sum of squares metric.

$$RSS(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\boxed{RSS(\boldsymbol{\beta}) = \sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{p} x_{ij}\beta_j\right)^2} \tag{2}$$

2.2.2 Solving Linear Regression (Gradient Descent)

Calculate the gradient of β (weights) and update the weights in the direction opposite to the gradient (Gradient points towards the maxima).

Gradient descent update rule -

$$\boldsymbol{\beta} = \boldsymbol{\beta} - learning\_rate * \nabla \boldsymbol{error}$$

Algorithm 1 Gradient Descent - Linera Regression

procedure RUN GRADIENT DESCENT($\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\beta}$, lr, max_epochs)
  for *epoch* $= 0$ to max_epochs **do**
    for $k = 0$ to p **do** ▷ Loop over all features
      $\frac{\partial RSS}{\partial \beta_k} \leftarrow \sum_{i=1}^{n} 2 * \left(y_i - \sum_{j=0}^{p} x_{ij}\beta_j\right) * x_{ik}$ ▷ Calculate the gradient
      $\beta_k \leftarrow \beta_k - lr * \frac{\partial RSS}{\partial \beta_k}$ ▷ Update weight
    end for
  end for
end procedure

2.3 Linear Regression - Model - Matrix Notation

n - Number of observations
p - Number of features

$\mathbf{y}$ (Expected Output - Shape: n x 1)

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$ (Predictors)(Input Observations)

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix}$$

$\boldsymbol{\beta} \in \mathbb{R}^{(p+1) \times 1}$ (Model weights)

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

$\hat{\boldsymbol{y}} \in \mathbb{R}^{n \times 1}$ (Prediction)

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} * \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} \hat{y_0} \\ \hat{y_1} \\ \vdots \\ \hat{y_n} \end{bmatrix}$$

$\boldsymbol{\epsilon}$ (Prediction error)

$$\hat{\boldsymbol{y}} - \boldsymbol{y} = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

2.3.1 Solving Linear Regression - Error Metric (Matrix Approach)

For a given $\boldsymbol{X}$ and $\boldsymbol{y}$, we need to minimize the error.
1 possible Error metric to minimize -
Residual Sum of Squares (RSS)

$$RSS(\boldsymbol{\beta}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

2.3.2 Analytical Solution (Matrix Approach) (Solve the Normal Eqn.)

Differentiating w.r.t. $\boldsymbol{\beta}$ and equating it to zero to minimize.

$$\frac{\partial RSS}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0$$

$$-2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0$$

$$\boxed{\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0} \tag{3}$$

$$\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = 0$$

$$\boxed{\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}} \tag{4}$$

Eq 4 is the Normal Equation. Solve the normal equation for β

$$\boxed{\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}} \tag{5}$$

The above unique solution for beta (5) is only solvable if $\mathbf{X}$ is full rank. If $\mathbf{X}$ is not full rank, $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ will not be invertible. This could happen when two inputs are perfectly correlated.

Equation (3) tells us that the solution is obtained by finding an error vector $\boldsymbol{\epsilon} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$ that is perpendicular to the column space of $\mathbf{X}$.

$$\boxed{\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta} = (\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)\boldsymbol{y}} \tag{6}$$

The prediction $\hat{\boldsymbol{y}}$ is the orthogonal projection of $\mathbf{y}$ onto the subspace spanned by $\mathbf{X}$. Recollect from basic linear algebra that $(\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)$ is the projection matrix for the subspace spanned by $\mathbf{X}$. Even when $\mathbf{X}$ is not full rank, the solution is obtained by projecting $\mathbf{y}$ on $\mathbf{X}$.

2.3.3 Pseudoinverse (X^+) Solution

If $\mathbf{X}$ is invertible $\boldsymbol{X}^+ = \boldsymbol{X}^{-1}$.
If $\mathbf{X}$ is m x n, then $\boldsymbol{X}^{-1}$ is n x m.
The pseudo inverse of $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ is $\boldsymbol{X}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T$.
Where -

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\boldsymbol{\Sigma}^+ = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{y}$$

$$\boxed{\boldsymbol{\beta} = \boldsymbol{X}^+\boldsymbol{y}} \tag{7}$$

2.3.4 Gram-Schmidt Solution

Gram-Schmidt procedure takes a set of independent vectors to an orthogonal set of vectors.
If the columns of $\mathbf{X}$ are independent, then orthogonalize the columns of $\mathbf{X}$ to get matrix $\mathbf{Q}$. Use Q to solve the normal equation -

$$\boldsymbol{\beta} = (\boldsymbol{Q}^T\boldsymbol{Q})^{-1}\boldsymbol{Q}^T\boldsymbol{y}$$

In the above equation. $\boldsymbol{Q}^T\boldsymbol{Q}$ is a Diagonal matrix. It is very easy to find the inverse of a diagonal matrix (invert the diagonal elements). At first, this might look like a better way to solve the normal equation, but the operation count is double compared to the regular normal equation. The main advantage of this method is numerical stability. Orthogonal matrix provides better numerical stability, especially when $\mathbf{X}$ is nearly singular.

2.4 Code Linear Regression (sci-kit learn, pytorch)

Linear Regression using sklearn

```
from sklearn import linear_model
import numpy as np

# Load a csv dataset
X_Y_dataset = np.genfromtxt("small_dataset.csv", delimiter=",", skip_header=True) #
                                                Remove the first row as it containes column
                                                 header text
print(X_Y_dataset)
X = X_Y_dataset[:,:-1] # All rows, discard last column
Y = X_Y_dataset[:,-1] # All rows, only last column
# Create the model object
lin_reg_model = linear_model.LinearRegression()
# Fit the model
lin_reg_model.fit(X,Y)
# Print the coeffs
print(lin_reg_model.coef_)
# Pint the intercept
print(lin_reg_model.intercept_)
```

Linear Regression using pytorch

```
import numpy as np
import torch
import torch.nn as nn

class LinearRegression(nn.Module):

    def __init__(self, in_size, out_size=1):
        super().__init__()

        self.linear_layer = nn.Linear(in_size, out_size)

    def forward(self, x):

        return self.linear_layer(x)


# Load a csv dataset
X_Y_dataset = np.genfromtxt("small_dataset.csv", delimiter=",", skip_header=True) #
                                                Remove the first row as it containes column
                                                 header text
print(X_Y_dataset)
X = torch.tensor(X_Y_dataset[:,:-1], dtype=torch.float) # All rows, discard last
                                                column and convert to torch tensor
Y = torch.tensor(X_Y_dataset[:,-2:-1], dtype=torch.float) # All rows, only last column
                                                 and convert to torch tensor

num_features = X.shape[-1] # Num columns
num_training_data = X.shape[0] # Num Rows
num_outputs = Y.shape[-1] # Num colums

# Create the torch model
lin_model = LinearRegression(in_size=num_features, out_size=num_outputs)

# MSE loss function
loss_fn = nn.MSELoss()

#Optimizer
optimz = torch.optim.Adam(lin_model.parameters(), lr=0.001)

epochs = 5000
lin_model.train()
for epoch in range(epochs):

    preds = lin_model(X)

    loss = loss_fn(preds, Y)
```

```
    loss.backward()
    optimz.step()
    optimz.zero_grad()

    print('Weights = ', lin_model.linear_layer.state_dict()['weight'])
    print('Biases = ',lin_model.linear_layer.state_dict()['bias'])
```

2.5 Why minimize RSS? Why not any other error metric?

1. **Gauss-Markov Theorem**: Lest squares estimates of the parameters β have the smallest variance among all the linear unbiased estimates.
2. Minimizing least-squares(RSS) is equivalent to obtaining the Maximum likelihood estimation fit.
3. Minimizing quadratic function is easier since the derivate is a linear function.

2.6 Issues with outliers in the data

2.7 How to mitigate the issues caused by outliers?

1. Remove the outliers before fitting the straight line. Below are two ways to remove outliers -
   (a) Sort the data and remove top and bottom 5% of the data from the dataset (**X, y**)
   (b) Use some anomaly detection techniques to remove outliers.
2. Use shrinkage techniques below -
   (a) Ridge Regression - Uses an L2 regularization term to control the value of the weights.
   (b) Lasso Regression - Uses an L1 regularization term to penalize large weights heavily. This technique produces a space output (Most of the weights will be close to zero).

2.8 PCA vs Linear Regression

2.9 Ridge Regression

2.10 Lasso Regression

2.11 Feature selection using lasso regression