Rapport - ft_linear_regression

Thibault Nguyen - thibnguy

Décembre 2024

1 Definition of a Linear Regression

Linear regression is a statistical method used to model the relationship between one dependent variable (Y) and one or more independent variables (X). It assumes that this relationship is linear, meaning that changes in X are directly proportional to changes in Y.
Key Elements of Linear Regression:

- **Equation of a Line**:
  - For simple linear regression (one independent variable), the model is:

    $$Y = mX + b + \epsilon$$

    where:
    - Y: Dependent variable (outcome or target variable).
    - m: Independent variable (predictor or feature).
    - X: Slope of the line, representing the rate of change in Y for a one-unit change in X.
    - b: Intercept, the value of Y when $X = 0$.
    - ϵ: Error term, accounting for the deviation of actual values from the predicted line.
- **Multiple Linear Regression**:

- If there are multiple independent variables (X_1, X_2, ..., X_n), the equation becomes:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n + \epsilon$$

  where each b_i represents the coefficient (effect) of the corresponding feature.

- **Assumptions**:
  - The relationship between X and Y is linear.
  - The residuals (errors) are normally distributed.
  - Homoscedasticity: The variance of residuals is constant across all values of X.
  - Independence: Observations are independent of one another.

- **Goal**:
  - The primary goal of linear regression is to find the best-fit line that minimizes the sum of squared residuals (differences between observed and predicted values).

- **Applications**:
  - Predicting continuous outcomes, such as house prices, temperatures, or sales.
  - Understanding relationships between variables, such as how advertising spend affects sales.

In summary, linear regression is a foundational technique in statistics and machine learning that models and predicts outcomes based on a linear relationship between variables.

2 How it works

Linear regression is a supervised learning algorithm that predicts a continuous target variable by finding the best-fit linear relationship between input features and the target. Here's how it works using the **four fundamentals**:

- **Dataset**
  - **Definition**: The dataset consists of examples with input features (X) and their corresponding target value (Y).
    - Example: Predicting house prices (Y) based on features like size (X_1) and number of bedrooms (X_2).
  - The dataset must be:
    - Clean and properly formatted.
    - Split into **training** and **testing** sets to evaluate the model's performance.

  By convention:

  - m: Number of examples (rows).
  - n: Number of features (columns, excluding the target).

Dataset (x, y)

m

Target	**Features**				
y	x_1	x_2	x_3	...	x_n
$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	...	$x_n^{(1)}$
$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$
$y^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	...	$x_n^{(3)}$
...	...	...	...	...	...
$y^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$	...	$x_n^{(m)}$

n

- **Model**
  - **Definition**: The model is a mathematical function that predicts Y (target) from X (features).

* For simple linear regression:

$$\hat{Y} = mX + b$$

  · m: Slope (how much Y changes per unit of X).

  · b: Intercept (value of Y when $X = 0$).

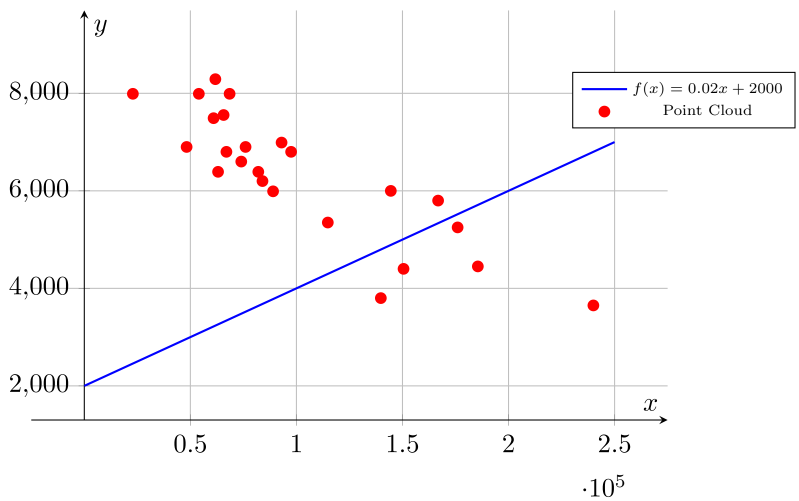
* For multiple linear regression:

$$\hat{Y} = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

  · b_0: Intercept.

  · b_i: Coefficient of each feature (X_i).

– **Purpose**: To represent the relationship between inputs (X) and output (Y) using a straight line (or hyperplane in multiple dimensions).



- **Cost Function**

  – **Definition**: The cost function measures the error between the predicted values ($\hat{Y}$) and the actual target values (Y).

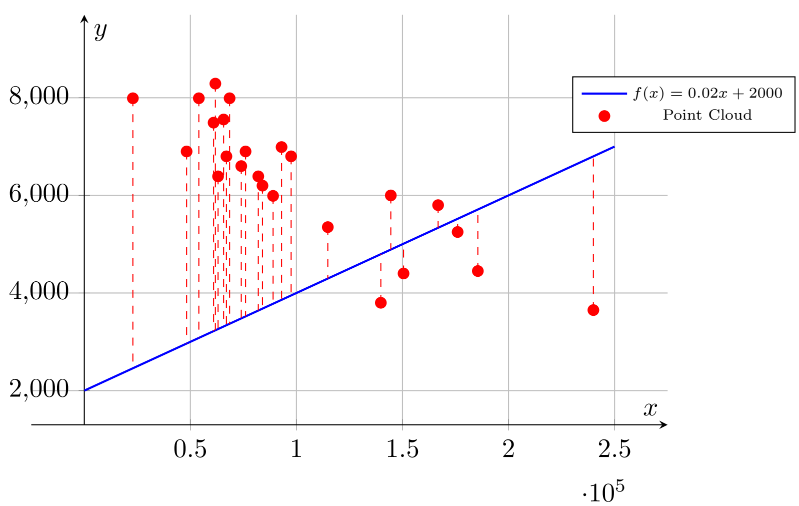
  * For linear regression, the most common cost function is the

Mean Squared Error (MSE):

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{Y}^{(i)} - Y^{(i)})^2$$

- $J(\theta)$: Cost function value.
- $\hat{Y}^{(i)}$: Predicted value for the i-th example.
- $Y^{(i)}$: Actual target value for the i-th example.
- m: Number of examples.

– **Purpose**: To provide a metric that the algorithm minimizes to improve predictions. Lower cost indicates better model performance.



- **Minimization Algorithm**

  – **Definition**: The minimization algorithm adjusts the model's parameters (coefficients m, b or b_0, b_1, ..., b_n) to reduce the cost function $J(\theta)$.

  – The most common algorithm used is **Gradient Descent**:

    ∗ Iteratively updates the parameters using the rule:

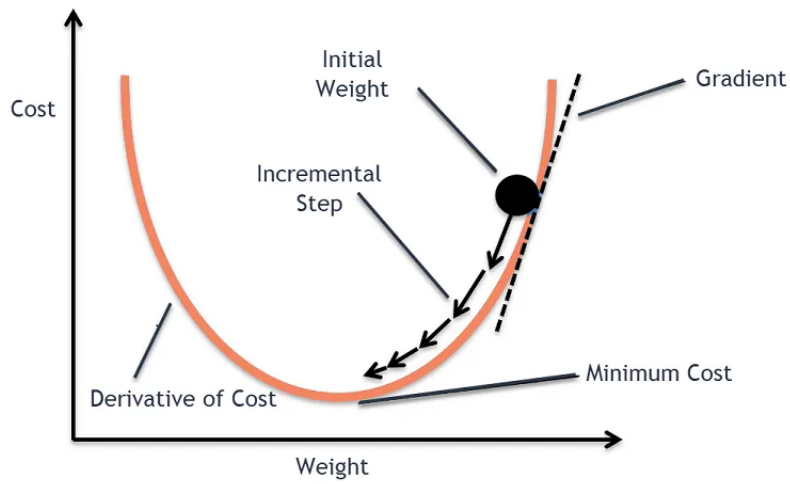
$$\theta = \theta - \alpha.\frac{\partial J(\theta)}{\partial \theta}$$

· α: Learning rate (step size).

· $\frac{\partial J(\theta)}{\partial \theta}$: Gradient of the cost function.

– **Steps in Gradient Descent**:

  * Compute the cost $J(\theta)$ for the current parameters.
  * Calculate the gradient (direction of steepest ascent).
  * Update the parameters in the opposite direction of the gradient (descent).
  * Repeat until the cost converges to its minimum.



Summary

– **Dataset**: Provides input-output pairs to learn from.

– **Model**: Represents the linear relationship between inputs (X) and output (Y).

– **Cost Function**: Quantifies prediction errors to guide the optimization.

– **Minimization Algorithm**: Adjusts the model's parameters to minimize errors, finding the best-fit line.

By combining these four steps, linear regression learns to make accurate predictions based on the data.

3 Train the model

- **Feature Normalization**

```
def normalize_features(x)
```

  - **Purpose**: Normalize the input features (x) to have a mean of 0 and a standard deviation of 1. This is important to:
    - Improve the performance of gradient descent (features with large scales can slow convergence).
    - Make the model parameters more interpretable.
  - **How it works**:

$$x' = \frac{x - mean(x)}{std(x)}$$

  - Example: If x represents car mileage ranging from 0 to 200,000, normalization ensures that this large range doesn't dominate the training process.

```
def denormalize_theta(theta, mean_x, std_x)
```

  - **Purpose**: Convert the normalized slope and intercept back to the original scale for interpretability.
  - **How it works**:
    - The slope is scaled by dividing by the standard deviation (std_x).
    - The intercept is adjusted by subtracting the influence of the slope on the mean of x.

- **Model**

```
def model(X, theta)
```

  - **Purpose**: Predict the output ($\hat{y}$) using the linear regression equation:

$$\hat{y} = X.\theta$$

  - X includes the features and an intercept term (bias).

  - θ represents the model parameters: slope (θ_1) and intercept (θ_0).

- **Cost Function**

```python
def cost(X, y, theta)
```

  - **Purpose**: Compute the **Mean Squared Error (MSE)** cost function to measure how well the model fits the data:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

    * m: Number of training examples.
    * Using $\frac{1}{2m}$ ensures the gradient is consistent in scale with other terms in the optimization process, making the training process smoother.
      The $\frac{1}{2}$ factor doesn't affect the optimization itself because it is a constant scaling factor.
      Minimizing $J(\theta)$ with $\frac{1}{2m}$ leads to the same parameter values θ as minimizing $J(\theta)$ with $\frac{1}{m}$.
      In summary, $\frac{1}{2m}$ is used instead of $\frac{1}{m}$ to make gradient calculations more elegant and computationally simpler without altering the results of the optimization.

- **Gradient Calculation**

```python
def gradient(X, y, theta)
```

  - **Purpose**: Compute the gradients of the cost function with respect to θ_0 (intercept) and θ_1 (slope):

    * Gradient for θ_0:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})$$

* Gradient for θ_1:

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)}).x^{(i)}$$

- These gradients indicate the direction and magnitude of parameter updates.

- **Gradient Descent**

```
def gradient_descent(X, y, theta, learning_rate, n_iterations)
```

- **Purpose**: Optimize the parameters θ_0 and θ_1 by iteratively updating them in the direction of the negative gradient.
- **How it works**:
  * Update rule:

$$\theta = \theta - \alpha.\nabla J(\theta)$$

    · α: Learning rate, controlling the step size.
    · $\nabla J(\theta)$: Gradient of the cost function.
  * Cost is computed and stored in each iteration for tracking convergence.
- The formulas of the **subject** are:

$$tmp\theta_0 = \alpha.\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})$$

$$tmp\theta_1 = \alpha.\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)}).x^{(i)}$$

where:

  * α: Learning rate.
  * $\hat{y}^{(i)}$: estimatePrice(mileage[i]).
  * $y^{(i)}$: price[i].
  * $x^{(i)}$: mileage[i].

- **Evaluation Metrics**

```python
def coef_determination(y, pred)
```

- **Formula**:

$$R^2 = 1 - \frac{Sum of Squared Residuals(SSR)}{Total Sum of Squares(SST)}$$

- **Purpose**:
  * R^2 measures the proportion of variance in the target variable explained by the model.
  * $R^2 = 1$: Perfect fit.
  * $R^2 = 0$: No better than predicting the mean.

```python
def mean_squared_error(y_actual, y_predicted)
```

- **Formula**:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

- **Purpose**:
  * MSE measures the average of the squared differences between actual and predicted values.
  * It penalizes larger errors more heavily than smaller ones due to the squaring.
  * It's useful for comparing model performance but not as interpretable because its units are the square of the target variable's units.

```python
def mean_absolute_error(y_actual, y_predicted)
```

- **Formula**:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y^{(i)} - \hat{y}^{(i)}|$$

- **Purpose**:
  * MAE calculates the average of the absolute differences between actual and predicted values.

* It provides an intuitive measure of error in the same unit as the target variable.
* Less sensitive to outliers compared to MSE since it doesn't square the errors.

```
def root_mean_squared_error(y_actual, y_predicted)
```

- **Formula**:

$$RMSE = \sqrt{MSE}$$

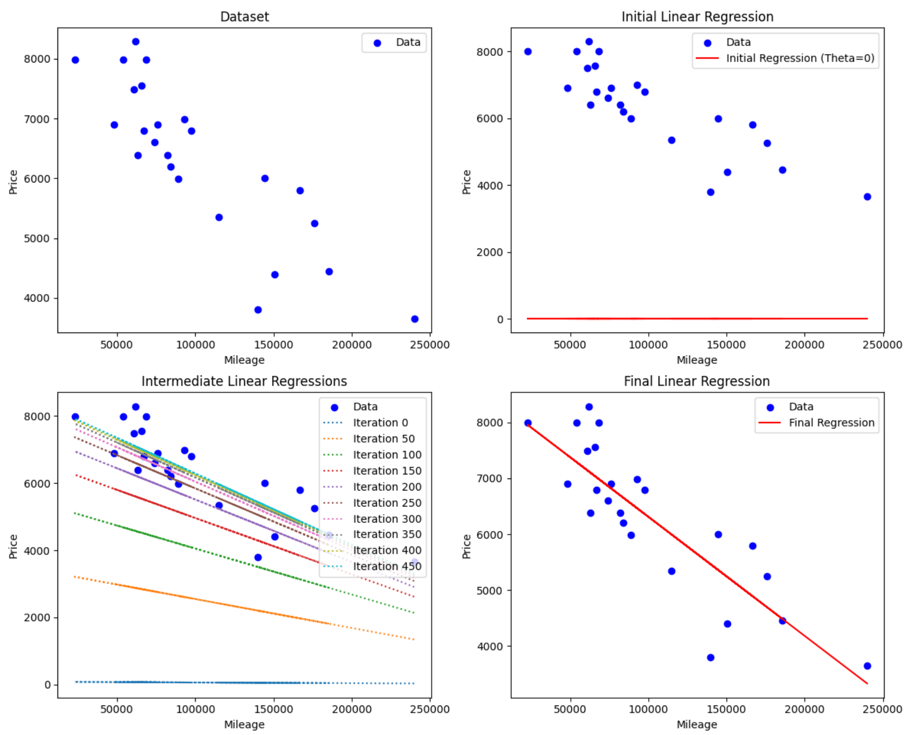
- **Purpose**:
  * RMSE is the square root of MSE, which brings the error back to the same unit as the target variable.
  * It combines the advantages of MSE (emphasizing larger errors) with interpretability in the target variable's units.
  * Often used in practice to compare model performance.

$Metric$	$PenalizesLargeErrors?$	$Unit$	$Interpretability$	$SensitivitytoOutliers$
MSE	Yes	$Squared$	Low	$High$
MAE	No	$Sameastarget$	$High$	Low
$RMSE$	Yes	$Sameastarget$	$Medium$	$Medium$
R^2	No	$Dimensionless$	$High(VarianceExplained)$	N/A

- **How It All Works Together**
  - **Normalize the features** to ensure faster and more stable gradient descent.
  - **Initialize** θ_0 **and** θ_1, and compute the initial cost.
  - Use **gradient descent** to iteratively update θ and minimize the cost.
  - Once training is complete:
    * Denormalize θ to interpret the slope and intercept in the original scale.
    * Evaluate the model's performance using R^2, MSE, MAE, and $RMSE$.

– **Visualize** the results:

* Plot the data points and the best-fit line.



4 Predict the model

```
def estimate_price(mileage, theta0, theta1)
```

- **Formula**:

$$Price = \theta_0 + (\theta_1.Mileage)$$

- **Purpose**:

  – The estimated price function is called with:

    * The user-provided mileage.
    * The loaded model parameters (θ_0 and θ_1).
    * θ_0: The baseline price.

    * θ_1: The price decrease per km.

- **Example**:
  - Assume the trained model get the values θ_0=8443.75 and θ_1=-0.0213
  - User enters a mileage of 50,000 km.
  - The program calculates:

$$Price = 8443.75 + (-0.0213 * 50000) = 8443.75 - 1065 = \$7378.75$$

Suppose that we get these values:

- **Mean Squared Error (MSE)**: 447428.88
  The squared error might look large, but it's in squared units and mainly useful for model comparisons.
- **Mean Absolute Error (MAE)**: 559.61
  On average, the predicted price is off by about $559.61.
- **Root Mean Squared Error ($RMSE$)**: 668.90
  Typical prediction error is around $668.90, but this value is more affected by outliers.
- **Coefficient of Determination (R^2)**: 73.19%
  Measures how well your model explains the variance in the target variable. Here, 73.19% of the variance in car prices is explained by the model (mileage as the feature).