Class Overview

- Objective Functions
- Regularization
  - L1 Regularization
  - L2 Regularization
- Polynomial Regression
- Multi Variable Linear Regression
- Regression Model Evaluation
  - SSR
  - SST
  - R2 value

Polynomial Regression / Multi Variable Linear Regression

Relationship

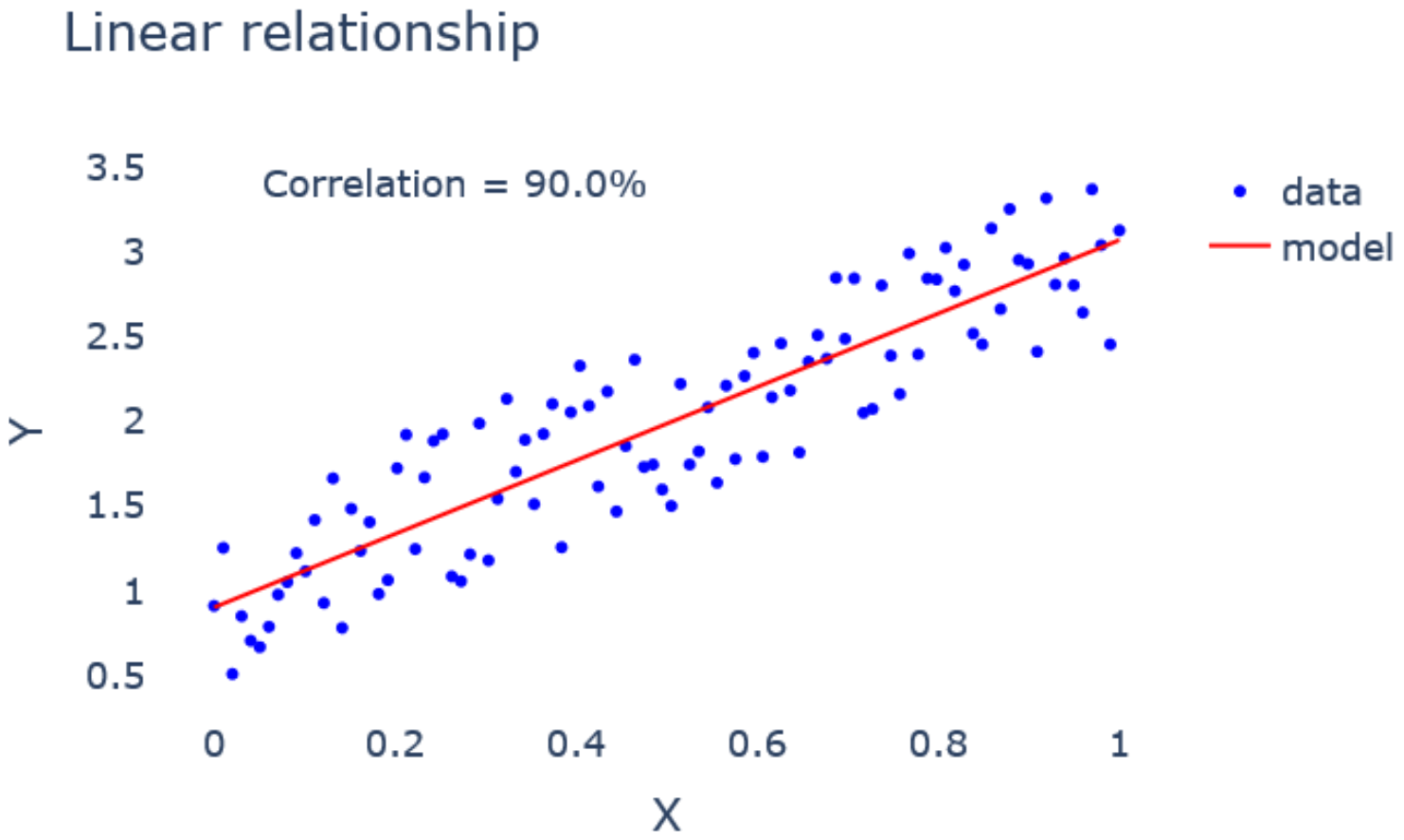

Fig: Correlation between X and Y

Relationship

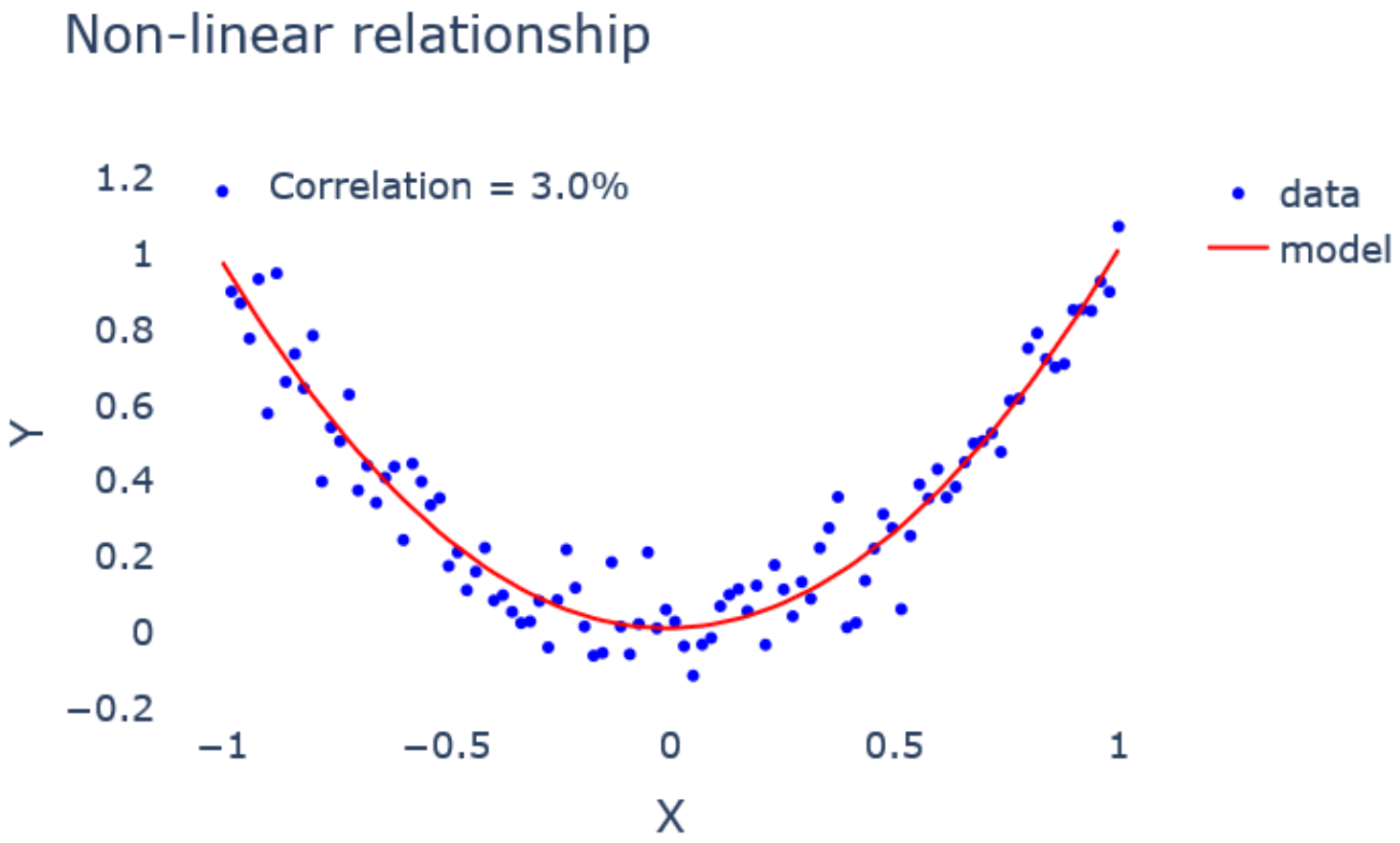

Fig: Correlation between X and Y

Correlation

To calculate the correlation coefficient between two variables X and Y, you can use Pearson's correlation coefficient formula.

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \cdot \sum(Y_i - \bar{Y})^2}}$$

Where:

- X_i and Y_i are individual data points for variables X and Y, respectively.
- $\bar{X}$ and $\bar{Y}$ are the means (averages) of variables X and Y, respectively.
- $\sum$ denotes the summation over all data points.

Correlation

Here's how you can calculate it step by step:

1. Calculate the mean ($\bar{X}$) and the mean ($\bar{Y}$) of variables X and Y, respectively.
2. For each data point, calculate the deviation from the mean for both X and Y. That is, subtract the mean from each data point: $X_i - \bar{X}$ and $Y_i - \bar{Y}$.
3. Multiply the deviations for each data point: $(X_i - \bar{X})(Y_i - \bar{Y})$.
4. Sum up all the products calculated in step 3.
5. Square the deviations for each data point for X: $(X_i - \bar{X})^2$, sum up all these squared deviations.
6. Square the deviations for each data point for Y: $(Y_i - \bar{Y})^2$, sum up all these squared deviations.
7. Take the square root of the product of the sums of squared deviations from steps 5 and 6.
8. Divide the result from step 4 by the result from step 7.

Correlation

This will give you the correlation coefficient r between variables X and Y. The value of r will lie between -1 and 1, where:

- $r = 1$ indicates a perfect positive linear relationship,
- $r = -1$ indicates a perfect negative linear relationship,
- $r = 0$ indicates no linear relationship.

Multiple Linear Regression:

- Multiple linear regression involves modeling the relationship between a dependent variable y and two or more independent variables $x_1, x_2, \ldots, x_k$.
- The general form of the multiple linear regression equation with k independent variables is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon$$

- Each x_i represents a different independent variable, and $\beta_0, \beta_1, \ldots, \beta_k$ are the coefficients to be estimated.

Polynomial Regression

Mathematical concepts

Multiple Linear Regression:

- Multiple linear regression involves modeling the relationship between a dependent variable y and two or more independent variables $x_1, x_2, \ldots, x_k$.
- The general form of the multiple linear regression equation with k independent variables is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon$$

- Each x_i represents a different independent variable, and $\beta_0, \beta_1, \ldots, \beta_k$ are the coefficients to be estimated.

Polynomial Regression:

- Polynomial regression involves modeling the relationship between a dependent variable y and an independent variable x using a polynomial equation.
- The general form of a polynomial regression equation of degree n is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \varepsilon$$

- Here, the independent variable x is raised to different powers up to n, allowing the model to capture non-linear relationships.

The "degree" of the polynomial refers to the **highest power of the input variable(s)** in the polynomial equation. The degree indicates the level of complexity of the model, with higher degrees allowing for more complex curves to fit the data.

- **Degree 1**:

  This represents a linear equation of the form $y = ax + b$, where a is the slope of the line and b is the y-intercept. A polynomial of degree 1 indicates a simple linear relationship between the input variable x and the output variable y.

- **Degree 2**:

  This represents a quadratic equation of the form $y = ax^2 + bx + c$. A polynomial of degree 2 can model parabolic curves, allowing for the representation of relationships where the rate of change is not constant and can accelerate or decelerate.

- **Degree 3**:

  This represents a cubic equation of the form $y = ax^3 + bx^2 + cx + d$. A cubic polynomial can model more complex curves that can change direction, allowing for an S-shaped curve or other non-linear patterns that a quadratic equation cannot capture.

- **Higher Degrees**: Polynomials of degree 4, 5, and beyond can model even more complex relationships with multiple inflection points and changes in direction. However, with increased degree, the polynomial model becomes more prone to overfitting, especially if the data does not support such complexity or if there is not enough data.

Polynomial Regression

Mathematical concepts

The formula to calculate the total number of features after applying polynomial features of degree *d* to an original set of *n* features is given by:

$$\text{Total features} = \binom{n+d}{d} = \frac{(n+d)!}{n!d!}$$

Where $\binom{n+d}{d}$ is a binomial coefficient representing the number of ways to choose d items from $n+d$ items without regard to the order, n is the original number of features, and d is the degree of the polynomial.

Polynomial Regression

Mathematical concepts

- For degree 2 (including original features, squares, and pairwise interactions):

$$\text{Total features} = \binom{3+2}{2} = \frac{(3+2)!}{3!2!}$$

- For degree 3 (including original features, squares, cubes, and all interactions up to third degree):

$$\text{Total features} = \binom{3+3}{3} = \frac{(3+3)!}{3!3!}$$

After performing polynomial regression:

- ❖ With a degree of 2, the feature count will increase to 10.
- ❖ With a degree of 3, the feature count will increase to 20.

Polynomial Regression

Pros and Cons

The "best" degree for polynomial regression depends on your specific dataset and the **complexity** of the **underlying relationship** that you're trying to **model between the features and the target variable**. Here are some considerations to help decide between a degree of 2 and a degree of 3, or any other degree:

Degree 2 (Quadratic):

- **Pros:**
  - Captures relationships that are not merely linear, such as parabolic trends.
  - Increases model complexity moderately, which can improve fitting for slightly more complex patterns without going too far into overfitting.

- **Cons:**
  - May still be too simple for some datasets, failing to capture higher-order relationships.

Degree 3 (Cubic):

- **Pros:**
  - Can model more complex patterns than quadratic, including S-shaped curves, which can be more representative of certain natural or economic phenomena.
  - Useful for datasets where the relationship between features and target changes direction more than once.

- **Cons:**
  - Increases the risk of overfitting, especially if the dataset is not very large, as the model becomes significantly more complex.
  - Requires more data to train effectively compared to quadratic models.

Polynomial Regression

General Considerations:

• **Model Complexity vs. Overfitting**: Higher degrees can capture more complex relationships but also increase the risk of overfitting. Overfitting happens when the model learns the noise in the training data instead of the actual underlying pattern, leading to poor performance on unseen data.

• **Cross-Validation**: To find the best degree for your polynomial regression, you can use cross-validation. This involves training the model with different degrees and evaluating their performance on a validation set or through k-fold cross-validation.

• **Domain Knowledge**: Sometimes, domain knowledge can guide the choice of degree. If you know the underlying relationship should have a specific shape or form, you can choose the degree accordingly.

Regression Model Evaluation

Regression Performances

The coefficient of determination

R2 is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s) in a regression model. Mathematically, it is defined as follows:

R-squared (R^2):

- Calculate R^2 using the formula: $R^2 = 1 - \frac{SSR}{SST}$.
- R^2 will be between 0 and 1.

Here's a breakdown of the components in this equation:

- m is the number of data points.
- y_i is the actual (observed) value of the dependent variable for the i-th data point.
- $\hat{y}_i$ is the predicted value of the dependent variable for the i-th data point according to the regression model.
- $\bar{y}$ is the mean of the observed values of the dependent variable.

Residual Sum of Squares (SSR):

- Use your regression model to predict $\hat{y}_i$ for each data point.
- For each y_i, calculate the squared difference between the observed and predicted values: $(y_i - \hat{y}_i)^2$.
- Sum all these squared differences. This sum represents the unexplained variability and is called the Residual Sum of Squares (SSR).

Total Sum of Squares (SST):

- Calculate the mean of the observed Y values, denoted as $\bar{y}$.
- For each y_i, calculate the squared difference from the mean: $(y_i - \bar{y})^2$.
- Sum all these squared differences. This sum represents the total variability in Y and is called the Total Sum of Squares (SST).

Objective Functions

The objective function, or the cost function or loss function, is a mathematical function that a machine learning algorithm seeks to minimize during the training process. It represents a measure of the error or discrepancy between the predicted values of the model and the actual target values in the training data.

The primary goal of a machine learning algorithm is to learn a set of parameters that minimizes this objective function, thereby improving the model's ability to make accurate predictions on new, unseen data.

The objective function guides the optimization algorithm, which iteratively adjusts the model parameters to find the optimal values. or loss function is a mathematical function that a machine learning algorithm seeks to minimize during training loss function, is a mathematical function that a machine learning algorithm seeks to minimize during training loss function, is a mathematical function that a machine learning algorithm seeks to minimize during training

Objective Function

Types / Situations

- **Regression Problems:** The mean squared error (MSE) is a common objective function. It measures the average squared difference between the predicted and actual values.
- **Classification Problems:** The cross-entropy loss is often used. It quantifies the dissimilarity between the true class labels and the predicted probabilities.

The objective function is typically denoted as $J(\theta)$, where θ represents the parameters of the model. The learning algorithm aims to find the values of θ that minimize $J(\theta)$. The process of minimizing the objective function is often referred to as training the model.

Objective Function

The Mean Squared Error

The Mean Squared Error (MSE) is a commonly used objective function for regression problems. It quantifies the average squared difference between the predicted values of a model and the actual target values in the training data. The formula for MSE is as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Here:

- n is the number of data points in the training set.
- y_i is the actual target value for the i-th data point.
- $\hat{y}_i$ is the predicted value for the i-th data point.

The MSE formula calculates the squared difference between each predicted value and its corresponding actual value, sums up these squared differences across all data points, and then divides by the number of data points (n) to obtain the average squared difference.

Objective Function

Binary Cross-Entropy Loss

Binary Cross-Entropy Loss:

For binary classification problems (two classes: 0 and 1), the binary cross-entropy loss is given by:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Here,

- N is the number of data points.
- y_i is the true label for the i-th data point (either 0 or 1).
- $\hat{y}_i$ is the predicted probability that the instance belongs to class 1.

Objective Function

Categorical Cross-Entropy Loss

Categorical Cross-Entropy Loss:

For multi-class classification problems (more than two classes), the categorical cross-entropy loss is given by:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

Here,

- N is the number of data points.
- C is the number of classes.
- y_{ij} is an indicator function equal to 1 if the true class of the i-th data point is j and 0 otherwise.
- $\hat{y}_{ij}$ is the predicted probability (or score) that the i-th data point belongs to class j.

In both cases, the goal during training is to minimize the cross-entropy loss by adjusting the model parameters. The optimization process aims to find the parameter values that make the predicted probabilities as close as possible to the true class labels.

Objective Function

Categorical Cross-Entropy Loss

```
L = -Σ[y_i * log(p_i)]
```

where:

- `L` is the categorical cross-entropy loss
- `y_i` is the true probability of the i-th class
- `p_i` is the predicted probability of the i-th class

```
y_true = [1, 0]
y_pred = [0.7, 0.3]

loss = -sum(y_true * np.log(y_pred))
print(loss)
```

Regularization (L1 & L2)

Regularization is a technique used in machine learning to prevent overfitting and improve the generalization of a model. Overfitting occurs when a model learns the training data too well, including its noise and outliers, to the extent that it performs poorly on new, unseen data.

Regularization methods add a penalty term to the model's loss function, discouraging the learning algorithm from assigning excessive importance to any one feature or allowing the model to become too complex.

- L1 Regularization (Lasso)
- L2 Regularization (Ridge)

L1 Regularization (Lasso):

- In L1 regularization, a penalty is added to the loss function that is proportional to the absolute values of the model parameters (weights). The regularization term is the sum of the absolute values of the weights, multiplied by a regularization parameter (lambda or alpha).
- The L1 regularization term is expressed as: $\lambda \sum_{i=1}^{n} |w_i|$, where λ is the regularization parameter and w_i are the model parameters.
- L1 regularization tends to produce sparse weight vectors, meaning that many of the feature weights are exactly zero. This can be useful for feature selection, as it effectively sets some features to be irrelevant.

Regularization

Lasso / L1 with Linear Regression

L1 regularization, also known as Lasso (Least Absolute Shrinkage and Selection Operator) regularization, has the property of inducing sparsity in models. When you apply L1 regularization to linear regression (or other linear models), it adds a penalty term to the cost function that is proportional to the absolute values of the coefficients. This penalty encourages the optimization process to drive some of the coefficients to exactly zero. The L1 regularization term is typically added to the mean squared error (MSE) cost function for linear regression, resulting in the Lasso cost function::

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{m} \sum_{j=1}^{n} |\theta_j|$$

Here:

- $J(\theta)$ is the cost function.
- m is the number of training examples.
- $h_\theta(x^{(i)})$ is the predicted output for the i-th training example.
- $y^{(i)}$ is the true output for the i-th training example.
- θ_j are the coefficients (weights) of the linear regression model.
- λ is the regularization parameter, controlling the strength of the L1 penalty.
- n is the number of features.

L2 Regularization (Ridge):

- In L2 regularization, a penalty is added to the loss function that is proportional to the square of the model parameters. The regularization term is the sum of the squared values of the weights, multiplied by a regularization parameter (lambda or alpha).
- The L2 regularization term is expressed as: $\lambda \sum_{i=1}^{n} w_i^2$, where λ is the regularization parameter and w_i are the model parameters.
- L2 regularization tends to produce weight vectors with small, non-zero values. It is effective at preventing any one feature from having an extremely large influence on the model.

Regularization

Ridge / L2 with Linear Regression

Linear regression with an L2 regularization term is often referred to as Ridge regression. The L2 regularization term adds the sum of the squared values of the coefficients to the linear regression cost function. The complete cost function for Ridge regression is given by:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Here:

- $J(\theta)$ is the cost function.
- m is the number of training examples.
- $h_\theta(x^{(i)})$ is the predicted output for the i-th training example.
- $y^{(i)}$ is the true output for the i-th training example.
- θ_j are the coefficients (weights) of the linear regression model.
- λ is the regularization parameter, controlling the strength of the L2 penalty.
- n is the number of features.

- **Use Lasso (L1 Regularization) When:**
  - **Feature Selection:** If you want to perform feature selection and exclude irrelevant features.
  - **Sparse Solutions:** If you prefer a model with fewer non-zero coefficients for interpretability.
  - **High-Dimensional Data:** Effective for high-dimensional datasets with many features.
  - **Handling Multicollinearity:** Can handle multicollinearity by automatically selecting one feature from correlated groups.

- **Use Ridge (L2 Regularization) When:**
  - **Multicollinearity Concerns:** If your features are highly correlated, Ridge can mitigate multicollinearity.
  - **All Features Relevant:** If you believe all features are potentially relevant and should be retained.
  - **Preventing Overfitting:** Useful for preventing overfitting by penalizing large coefficients.
  - **Balancing Simplicity and Performance:** Provides a balance between model simplicity and predictive performance.

Let’s do it with Python