WEEK -05: Polynomial and Logistic Regression

POLYNOMIAL REGRESSION

Some engineering data is poorly represented by a straight line. For these cases, a curve would be better suited to it these data. The method is to it polynomials to the data using polynomial regression. Polynomial Regression is a form of regression analysis in which the relationship between the independent variables and dependent variables are modeled in the nth degree polynomial. Polynomial Regression models are usually fit with the method of least squares. The **least square method** minimizes the variance of the coefficients, under the Gauss Markov Theorem. Polynomial Regression is a special case of Linear Regression where we fit the polynomial equation on the data with a curvilinear relationship between the dependent and independent variables. A Quadratic Equation is a Polynomial Equation of 2nd Degree. However, this degree can increase to nth values.

The least-squares procedure can be readily extended to it the data to a higher-order polynomial. For example, suppose that we it a second-order polynomial or quadratic:

$$y = a_0 + a_1 x + a_2 x^2 + e$$

Where x-independent variable, y-dependent variable, a0,a1 and a2 are coefficients, and e –error term. For this case the sum of the squares of the residuals is

$$S_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

we take the derivative with respect to each of the unknown coefficients of the polynomial, as in

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$(n) a_0 + \left(\sum x_i\right) a_1 + \left(\sum x_i^2\right) a_2 = \sum y_i$$

$$\left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 + \left(\sum x_i^3\right) a_2 = \sum x_i y_i$$

$$\left(\sum x_i^2\right) a_0 + \left(\sum x_i^3\right) a_1 + \left(\sum x_i^4\right) a_2 = \sum x_i^2 y_i$$

where all summations are from i = 1 through n. Note that the above three equations are linear and have three unknowns: a0 , a1 , and a2 . The coefficients of the unknowns can be calculated directly from the observed data. For this case, we see that the problem of determining a least-squares second-order polynomial is equivalent to solving a system of three simultaneous linear equations.

These equations can be rewritten in matrix form as follows:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \end{bmatrix}$$

Where:

- *n* is the number of data points.
- *xi* and *yi* are the values of the independent and dependent variables for the *i*-th data point.

The most common way to solve a system of linear equations is by using matrix algebra and methods like Gaussian elimination or matrix inversion. Here are the general steps to solve a system of linear equations:

Step 1: Write Down the System of Equations

Write the system of linear equations in standard form, where each equation has the form:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = b$$

Where:

- $a1,a2,\ldots,an$ are the coefficients of the variables $x1,x2,\ldots,xn$.
- b is the constant on the right-hand side of the equation.

You should have as many equations as there are variables.

Step 2: Create a Coefficient Matrix

Construct a coefficient matrix A by extracting the coefficients of the variables from the left side of each equation. This matrix will have dimensions $n \times n$ if you have n variables.

Step 3: Create a Right-Hand Side Vector

Create a right-hand side vector B by extracting the constants from the right side of each equation. This vector will have dimensions $n \times 1$.

Step 4: Solve the System

There are several methods to solve the system:

There are several methods to solve the system:

- **Matrix Inversion**: If A is invertible (non-singular), you can solve the system using the formula $X=A^{-1}B$, where X is the vector of solutions.
- **Gaussian Elimination**: Use row operations to transform the augmented matrix $[A|B]$ into row-echelon or reduced row-echelon form. Then, back-substitute to find the values of the variables.
- **Matrix Factorization**: Factorize A into LU or QR form and use the factorization to solve the system more efficiently.
- **Numerical Methods**: For large or ill-conditioned systems, numerical methods like the Gauss-Seidel method or the Conjugate Gradient method are used.

Here's a simple example using matrix inversion in Python:

```
import numpy as np
# Define the coefficient matrix A and the right-hand side vector B
A = np.array([[2, 1], [1, 3]])
B = np.array([4, 7])
# Solve for the variables X using matrix inversion
X = np.linalg.inv(A).dot(B)
print("Solution:")
print(X)
```

The above code solves the system $2x_1+x_2=4$ and $x_1+3x_2=7$ and prints the values of x_1 and x_2 as the solution.

Solution:

[1. 2.]

Polynomial regression is a type of regression analysis in which the relationship between the independent variable (X) and the dependent variable (Y) is modeled as an nth-degree polynomial. In Python, you can perform polynomial regression using libraries like NumPy and scikit-learn. Here's a basic example of polynomial regression using scikit-learn:

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
Y = 4 + 3 * X + np.random.randn(100, 1)

# Fit a polynomial regression model
degree = 2  # You can change the degree as needed
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)

model = LinearRegression()
model.fit(X_poly, Y)
```

```
# Make predictions
X_new = np.linspace(0, 2, 100).reshape(-1, 1)
X_new_poly = poly_features.transform(X_new)
Y_new = model.predict(X_new_poly)

# Plot the original data and the polynomial regression curve
plt.scatter(X, Y, label='Original Data')
plt.plot(X_new, Y_new, 'r-', label='Polynomial Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
# The coefficients of the multivariate polynomial regression model
coefficients = model.coef_
intercept = model.intercept_
print("Coefficients:")
print(coefficients)
print("Intercept:")
print(intercept)
```

In this example:

1. We generate some sample data points with random noise.
2. We use **PolynomialFeatures** from scikit-learn to transform our input features **X** into polynomial features up to a specified degree.
3. We then fit a linear regression model to the polynomial features.
4. Finally, we make predictions using the model and plot the original data along with the polynomial regression curve.

You can adjust the **degree** variable to control the degree of the polynomial you want to fit to your data. Higher-degree polynomials can capture more complex relationships but may also lead to overfitting, so be cautious when choosing the degree.

LOGISTIC REGRESSION

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class or not. It is a kind of statistical algorithm, which analyze the relationship between a set of independent variables and the dependent binary variables. It is a powerful tool for decision-making. For example email spam or not.

It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a sigmoid function to estimate the probability for the given class. The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

It is used for predicting the categorical dependent variable using a given set of independent variables.

- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1. o The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form.
- The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

1. **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
2. **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
3. **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Terminologies involved in Logistic Regression:

Here are some common terms involved in logistic regression:

- **Independent variables:** The input characteristics or predictor factors applied to the dependent variable's predictions.
- **Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.
- **Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.
- **Odds:** It is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur.
- **Log-odds:** The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.
- **Coefficient:** The logistic regression model's estimated parameters, show how the independent and dependent variables relate to one another.
- **Intercept:** A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.
- **Maximum likelihood estimation:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model.

How does Logistic Regression work?

The logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Let the independent input features be

$$X = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ x_{21} & \dots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e. 0 or 1.

$$Y = \begin{cases} 0 & \text{if } Class\ 1 \\ 1 & \text{if } Class\ 2 \end{cases}$$

then apply the multi-linear function to the input variables X

$$z = (\sum_{i=1}^{n} w_i x_i) + b$$

x_i $w_i = [w_1, w_2, w_3, \cdots, w_m]$

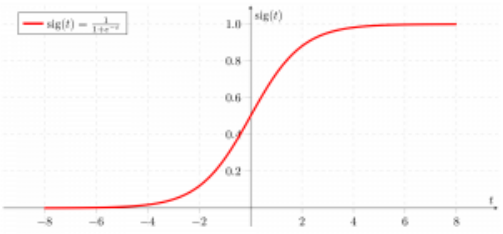
Here is the ith observation of X, is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

$$z = w \cdot X + b$$

Sigmoid Function

Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e predicted y.

$$\sigma(z) = \frac{1}{1-e^{-z}}$$



Sigmoid function

As shown above, the figure sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.

- $\sigma(z)$ $z \to \infty$ tends towards 1 as
- $\sigma(z)$ $z \to -\infty$ tends towards 0 as
- $\sigma(z)$ is always bounded between 0 and 1

where the probability of being a class can be measured as:

$$P(y = 1) = \sigma(z)$$
$$P(y = 0) = 1 - \sigma(z)$$

Logistic Regression Equation

The odd is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur. so odd will be

$$\frac{p(x)}{1-p(x)} = e^z$$

Applying natural log on odd. then log odd will be

$$\log\left[\frac{p(x)}{1-p(x)}\right] = z$$
$$\log\left[\frac{p(x)}{1-p(x)}\right] = w \cdot X + b$$

then the final logistic regression equation will be:

$$p(X;b,w) = \frac{e^{w\cdot X+b}}{1+e^{w\cdot X+b}} = \frac{1}{1+e^{-w\cdot X+b}}$$

Likelihood function for Logistic Regression

The predicted probabilities will p(X;b,w) = p(x) for y=1 and for y = 0 predicted probabilities will 1-p(X;b,w) = 1-p(x)

$$L(b,w) = \prod_{i=1} np(x_i)^{y_i}(1-p(x_i))^{1-y_i}$$

Taking natural logs on both sides

$$\begin{aligned} l(b,w) = \log(L(b,w)) &= \sum_{i=1}^{n} y_i \log p(x_i) \; + \; (1-y_i)\log(1-p(x_i)) \\ &= \sum_{i=1}^{n} y_i \log p(x_i) + \log(1-p(x_i)) - y_i \log(1-p(x_i)) \\ &= \sum_{i=1}^{n} \log(1-p(x_i)) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1-p(x_i} \\ &= \sum_{i=1}^{n} -\log 1 - e^{-(w\cdot x_i+b)} + \sum_{i=1}^{n} y_i(w\cdot x_i+b) \\ &= \sum_{i=1}^{n} -\log 1 + e^{w\cdot x_i+b} + \sum_{i=1}^{n} y_i(w\cdot x_i+b) \end{aligned}$$

Gradient of the log-likelihood function

To find the maximum likelihood estimates, we differentiate w.r.t w,

$$\begin{aligned} \frac{\partial J(l(b,w)}{\partial w_j} &= -\sum_{i=n}^{n} \frac{1}{1+e^{w\cdot x_i+b}} e^{w\cdot x_i+b} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= -\sum_{i=n}^{n} p(x_i;b,w) x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=n}^{n} (y_i - p(x_i;b,w)) x_{ij} \end{aligned}$$

Assumptions for Logistic Regression

The assumptions for Logistic regression are as follows:

- **Independent observations:** Each observation is independent of the other. meaning there is no correlation between any input variables.
- **Binary dependent variables:** It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories softmax functions are used.
- **Linearity relationship between independent variables and log odds:** The relationship between the independent variables and the log odds of the dependent variable should be linear.
- **No outliers:** There should be no outliers in the dataset.
- **Large sample size:** The sample size is sufficiently large

Logistic regression is a commonly used algorithm for binary classification problems. Here's a Python program for logistic regression with an example using the popular Iris dataset for classification:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # We'll use only the first two features for simplicity
y = (iris.target != 0) * 1  # Convert target labels to binary (0 or 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the feature data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a logistic regression model
model = LogisticRegression(solver='liblinear')

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdBu)
plt.xlabel('Sepal Length (standardized)')
plt.ylabel('Sepal Width (standardized)')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

In this program:

1. We load the Iris dataset and select only the first two features for binary classification.

2. We split the data into training and testing sets using **train_test_split**.
3. We standardize the feature data using **StandardScaler** to have zero mean and unit variance.
4. We create a logistic regression model using **LogisticRegression** from scikit-learn.
5. We train the model on the training data.
6. We make predictions on the test data and evaluate the model's performance using a confusion matrix and a classification report.
7. Finally, we plot the decision boundary of the logistic regression model to visualize how it separates the two classes.

You can adjust the dataset, features, and model parameters as needed for your specific classification problem.

Questions

1. Suppose you have a gold/silver price dataset with a single independent variable (X) and a dependent variable (Y). You want to fit a polynomial regression model to this data. Implement the process of selecting the appropriate degree for the polynomial (e.g., linear, quadratic, cubic) based on the dataset using Python.

3. Suppose you have a gold/silver price dataset with a single independent variable (X) and a dependent variable (Y). You want to fit a logistic regression model to this data. Develop an example code snippet in Python.

3. Imagine you have a gold and silver price dataset with two independent variables (X1 and X2) and a dependent variable (Y). Implement in python, how you can perform multivariate polynomial regression to model the relationship between the independent variables and the dependent variable.

4. Imagine you have a gold and silver price dataset with two independent variables (X1 and X2) and a dependent variable (Y). Implement in python, how you can perform the logistic regression to model the relationship between the independent variables and the dependent variable

Additional Questions

Consider the gold and/or silver price dataset and to evaluate a logistic regression model using ROC and AUC:

1. Calculate predicted probabilities for each instance in the test set.
2. Plot the ROC curve using the True Positive Rate (Sensitivity) and False Positive Rate.
3. Calculate the AUC to summarize the model's performance.