Linear and Logistic Regression

quantra

Table of Contents

Linear Regression

Linear Regression is one of the most well-known and understood algorithms in machine learning and statistics. Here, you will be given a brief overview on how it works and how you can use it in your machine learning problems.

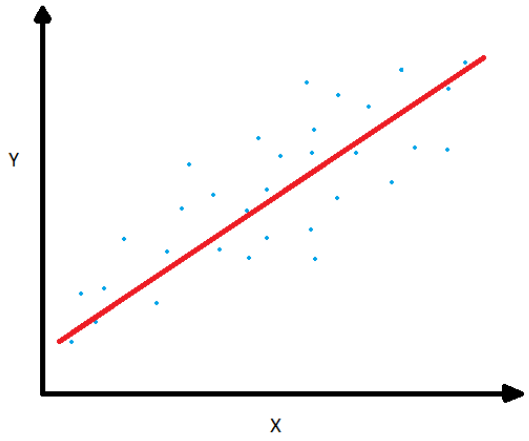
Linear Regression was developed under the field of statistics to study the relationship between input and output numerical variables, but has been borrowed by machine learning to make predictions based on a linear regression equation.

The mathematical representation of Linear Regression is a linear equation that combines a specific set of input data (x) to predict the output value (y) for that set of input values. The linear equation assigns a factor to each set of input values, which are called the coefficients represented by the Greek letter Beta (β). The equation mentioned below represents a linear regression model with two sets of input values, x_1 and x_2. Y represents the output of the model, β_0, β_1 and β_2 are the coefficients of the linear equation.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

The above equation represents a plane, also called a hyperplane. When there is only one input variable, the linear equation represents a straight line. For simplicity, consider β_2 to be equal to zero, this would imply that the variable x_2 will not influence the output of the linear regression model. In this case the linear regression will represent a straight line, the equation and plot are shown below.

$$y = \beta_0 + \beta_1 x_1$$



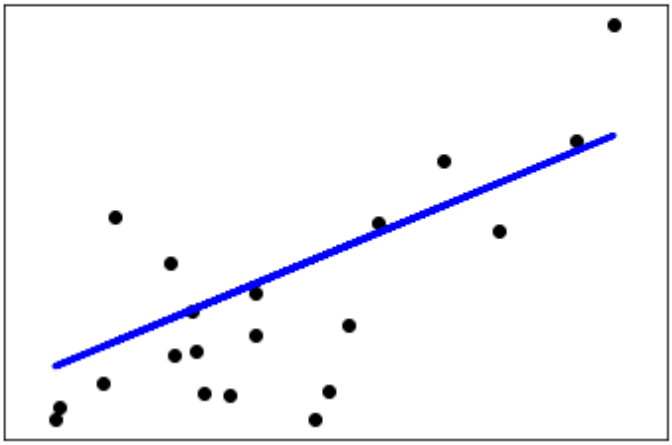
The blue dots in the above plot represents the input - output data (x, y) and the red line represents the linear regression equation which tries to plot the relationship between the input and output variables. Given below is a basic example of a Linear Regression Algorithm.

Linear Regression Algorithm:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model

# Load the diabetes dataset
diabetes = datasets.load_diabetes()


# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % np.mean((regr.predict(diabetes_X_test) - diabetes_y_test) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(diabetes_X_test, diabetes_y_test))
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, regr.predict(diabetes_X_test), color='blue',
         linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Output of Linear Regression Model:

```
('Coefficients: \n', array([ 938.23786125]))
Mean squared error: 2548.07
Variance score: 0.47
```



Logistic Regression

In linear regression, our aim was to predict continuous valued quantities as a linear function of the input values. In cases when the aim is to predict a discrete variable such as predicting whether a grid of pixel intensities represents 0 or 1, we want to classify based on the input values. Logistic Regression is a simple classification algorithm for learning to make such decisions.

Logistic Regression is a model that is used when the dependent variable is categorical. A few cases where logistic regression can be used are mentioned below:

- Image Segmentation and Categorization
- Geographic Image recognition
- Handwriting recognition
- Determining whether a person is depressed based on the words of his social media posts
- Predicting the probability of a person voting for a candidate in an election

In logistic regression, we try to predict the probability that a given example belongs to the '1' class versus the probability that it belongs to the '0' class. Logistic regression is mathematically represented by the equation given below:

$$P(y = 1 \mid x) = h_\theta(x) = 1 / (1 + \exp(-\theta^T x)) = \sigma(\theta^T x)$$

$$P(y = 0 \mid x) = 1 - P(y = 1 \mid x) = 1 - h_\theta(x)$$

The function (z) = 1 / (1 +exp(-z)) is called the sigmoid or logistic function. It is an S-shaped function that restricts the range of values of z into the range [0,1], so that we can interpret hθ(x) as a probability. Given below is a logistic regression model coded in Python:

Logistic Regression Algorithm:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

h = .02  # step size in the mesh

logreg = linear_model.LogisticRegression(C=1e5)

# we create an instance of Neighbours Classifier and fit the data.
logreg.fit(X, Y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

Output of Logistic Regression Model: