Data Science with Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

Regression Analysis

Learning outcomes:

- Regression Analysis
- Linear Regression
- Implementing Linear Regression
- Multiple Linear Regression
- Implementing Multiple Linear Regression
- Polynomial Regression
- Implementing Polynomial Regression

Regression Analysis

Regression analysis is a predictive modelling technique that analyses the relation between the target or dependent variable and independent variable in a dataset. The different **types of regression analysis** techniques get used when the target and independent variables show a linear or non-linear relationship between each other, and the target variable contains continuous values. The regression technique gets used mainly to determine the predictor strength, forecast trend, time series, and in case of cause & effect relation.

Regression Analysis

Regression analysis is the primary technique to solve the regression problems in machine learning using data modelling. It involves determining the best fit line, which is a line that passes through all the data points in such a way that distance of the line from each data point is minimized.
Regression analysis provides detailed insight that can be applied to further improve products and services.

Regression Analysis

Types of Regression Analysis Techniques:

There are many **types of regression analysis techniques**, and the use of each method depends upon the number of factors. These factors include the type of target variable, shape of the regression line, and the number of independent variables. We will cover here some important types of regression analysis techniques.

Linear Regression

Linear regression is one of the most basic **types of regression in machine learning**.
The linear regression model consists of a **predictor variable (independent variable)** and a **response** or **dependent variable** related linearly to each other. You make this kind of relationships in your head all the time, for example when you calculate the age of a child based on her height, you are assuming the older he/she is, the taller he/she will be. Linear regression is one of the most basic statistical models out there, its results can be interpreted by almost everyone.

Linear Regression

The general mathematical equation for a linear regression is −

$y = ax + b$

where

y is the response variable (Dependent variable).

x is the predictor variable (Independent variable).

a is the slope of the line.

b is y-intercept(the value of ***y*** when ***x*** = 0)

a and **b** are constants which are called the coefficients.

Linear Regression

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. If we plot the independent variable (x) on the x-axis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points, as shown in the figure below.

Linear Regression

Python has methods for finding a relationship between data-points and to draw a line of linear regression. I will show you how to use these methods instead of going through the mathematical formula.

Implementing Linear Regression

It’s time to start implementing linear regression in Python. Basically, all you should do is apply the proper packages and their functions and classes. The package **scikit-learn** is a widely used Python library for machine learning, built on top of NumPy and some other packages. It provides the means for pre-processing data, reducing dimensionality, implementing regression, classification, clustering, and more. Like NumPy, scikit-learn is also open source.

Implementing Linear Regression

Simple Linear Regression with scikit-learn:

Let’s start with the simplest case, which is simple linear regression. There are five basic steps when you’re implementing linear regression:

- **Import the packages you need.**
- **Provide data to work with and eventually do appropriate transformations.**
- **Create a regression model and fit it with existing data.**
- **Check the results of model fitting to know whether the model is satisfactory.**
- **Apply the model for predictions.**

These steps are more or less general for most of the regression approaches and implementations.

Implementing Linear Regression

Step 1: Import packages and classes
The first step is to import the package **numpy** and the class **LinearRegression** from **sklearn.linear_model**:

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

Now, you have all the functionalities you need to implement linear regression. The class **sklearn.linear_model.LinearRegression** will be used to perform linear and polynomial regression and make predictions accordingly.

Implementing Linear Regression

Step 2: Provide data

The second step is defining data to work with.

In the example below, the x-axis represents **height**, and the y-axis represents **weight**. We have registered the height and weight of 13 person. Let us see if the data we collected could be used in a linear regression:

Here we will predict the **weight** of the person based on his/her **height**.

Implementing Linear Regression

Step 2: Provide data

```
x =
np.array([150,155,160,157,162,165,168,170,140,1
45,175,178,180]).reshape((-1,1))
y  = np.array
([50,52,54,58,62,63,66,68,45,47,70,72,73])
```

Now, you have two arrays: the input x and output y. You should call **.reshape()** on x because this array is required to be **two-dimensional**, or to be more precise, to have **one column and as many rows as necessary**. That's exactly what the argument (-1, 1) of .reshape() specifies.

Implementing Linear Regression

Step 3: Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data.

Let's create an instance of the class **LinearRegression**, which will represent the regression model:

```
model = LinearRegression()
```

It's time to start using the model. First, you need to call **.fit()** on model:

```
model.fit(x, y)
```

Implementing Linear Regression

Step 3: Create a model and fit it

With **.fit(),** you calculate the optimal values of the weights, using the existing input and output (x and y) as the arguments. In other words, .fit() **fits the model**. It returns self, which is the variable model itself. That's why you can replace the last two statements with this one:

```
model = LinearRegression().fit(x, y)
```

Implementing Linear Regression

Step 4: Get results

Once you have your model fitted, you can get the results to check whether the model works satisfactorily and interpret it. The attributes of model are .intercept_ which represents the coefficient **b** and .coef_ which represents **a**:

```
print('intercept:', model.intercept_)
print('slope:', model.coef_)
```

Implementing Linear Regression

Step 5: Predict response

Once there is a satisfactory model, you can use it for predictions with either existing or new data. To obtain the predicted response, use **.predict()**:

```
y_pred = model.predict(x)
print('predicted response:', y_pred)
```

Let’s find the weight of the person with height of 166 cm

```
z = model.predict(np.array([[166]]))
Print(z)
```

Implementing Linear Regression

Step 5: Predict response

When applying .predict(), you pass the regressor as the argument and get the corresponding predicted response.

This is a nearly identical way to predict the response:

```
y_pred = model.intercept_ + model.coef_ * x
print('predicted response:', y_pred)
```

Implementing Linear Regression

r-value: It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship then the linear regression can not be used to predict anything. This relationship - **the coefficient of correlation - is called r.** The degree of association is measured by a correlation coefficient, denoted by **r.** Complete correlation between two variables is expressed by either + 1 or -1. When one variable increases as the other increases the correlation is positive; when one decreases as the other increases it is negative. Complete absence of correlation is represented by 0. The r value is **0** means **no relationship**, and **1** means **100% related.** A value of **-1.0** means there is a perfect negative relationship between the two variables.

Implementing Linear Regression

In **regression**, the R^2 **coefficient of determination** is a statistical measure of how well the **regression** predictions approximate the real data points. An R^2 of 1 indicates that the **regression** predictions perfectly fit the data. You can obtain the coefficient of determination (R^2) with .score() called on model:

```
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
```

The result 0.96 shows that there is a very good relationship.

Multiple Linear Regression

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

Multiple Linear Regression

We have performed linear regression involving two variables. Almost all the real-world problems that you are going to encounter will have more than two variables. Linear regression involving multiple variables is called “**multiple linear regression**” or **multivariate linear regression**.

Multiple regression is an extension of simple linear regression. It is used when we want to predict the value of one dependent variable based on the values of two or more independent variables.

Multiple Linear Regression

The steps to perform multiple linear regression are almost similar to that of simple linear regression. The difference lies in the evaluation. You can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other. For multiple regression, we assume that there is a linear relationship between the dependent variable and the independent variables.

Implementing Multiple Linear Regression

Example:

Take a look at the data that we are going to use, it contains some information about cars.

We can predict the CO2 emission of a car based on the size of the engine (volume), but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

Implementing Multiple Linear Regression

Example:

Start by importing the Pandas module.

```
import pandas as pd
```

The Pandas module allows us to read csv files and return a DataFrame object.

```
df = pd.read_csv("F:\Alison R Studio\cars.csv")

# take a look at the dataset
df.head(10)
```

Uplatz

Implementing Multiple Linear Regression

Example:

Then make a list of the independent values and call this variable x.

Put the dependent values in a variable called y.

```
x = df[['Weight', 'Volume']]
y = df['CO2']
```

Implementing Multiple Linear Regression

Example:

We will use some methods from the **sklearn** module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the `LinearRegression()` method to create a linear regression object.

Implementing Multiple Linear Regression

Example:

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()
regr.fit(x, y)
```

Implementing Multiple Linear Regression

Example:

Now we have a regression object that are ready to predict **CO2** values based on a car's **weight** and **volume**:
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm^3:

```
predictedCO2 = regr.predict([[2300, 1300]])
print(predictedCO2)
```

Implementing Multiple Linear Regression

Example:

Result: We have predicted that a car with 1.3 litre engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometre it drives.

Implementing Multiple Linear Regression

Example:

Coefficient:
The coefficient is a factor that describes the relationship with an unknown variable.
In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2.
The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

Implementing Multiple Linear Regression

Example:

Coefficient:

Print the coefficient values of the regression object:

```
print(regr.coef_)
```

Result:

```
Output: [0.00755095 0.00780526]
```

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095

Volume: 0.00780526

Implementing Multiple Linear Regression

Example:

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm^3, the CO2 emission increases by 0.00780526 g.

I think that is a fair guess, but let test it!

We have already predicted that if a car with a 1300cm^3 engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Implementing Multiple Linear Regression

Example:

```
predictedCO2 = regr.predict([[3300, 1300]])
print(predictedCO2)
```

Output: [114.75968007]

We have predicted that a car with 1.3 litre engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometre it drives.

Which shows that the coefficient of 0.00755095 is correct:

107.2087328 + (1000 * 0.00755095) = 114.75968

Implementing Multiple Linear Regression

Model Evaluation:
The goal of regression is to build a model to accurately predict an unknown case.
We have to perform regression evaluation after building the model.
We'll introduce and discuss two types of evaluation approaches that can be used to achieve this goal.
These approaches are:
train and test on the same dataset,
and train/test split.

Implementing Multiple Linear Regression

Model Evaluation:

We'll talk about what each of these are, as well as the pros and cons of using each of these models.
Also, we'll introduce some metrics for accuracy of regression models. Let's look at the first approach.
When considering evaluation models, we clearly want to choose the one that will give us the most accurate results.
So, the question is, how we can calculate the accuracy of our model? In other words, how much can we trust this model for prediction of an unknown sample, using a given a dataset and having built a model such as linear regression.
One of the solutions is to select a portion of our dataset for testing. For instance, assume that we have 10 records in our dataset.

Implementing Multiple Linear Regression

Model Evaluation:

We use the entire dataset for training, and we build a model using this training set. Now, we select a small portion of the dataset, such as row numbers 6 to 9, but without the labels. This set, is called a test set, which has the labels, but the labels are not used for prediction, and is used only as ground truth. The labels are called "Actual values" of the test set. Now, we pass the feature set of the testing portion to our built model, and predict the target values.

Finally, we compare the predicted values by our model with the actual values in the test set.

This indicates how accurate our model actually is.

Implementing Multiple Linear Regression

Model Evaluation:

There are different metrics to report the accuracy of the model, but most of them work generally, based on the similarity of the predicted and actual values.

Let's look at one of the simplest metrics to calculate the accuracy of our regression model. As mentioned, we just compare the actual values, **y**, with the predicted values, which is noted as **ŷ** for the testing set. The error of the model is calculated as the average difference between the predicted and actual values for all the rows. So, the first evaluation approach we just talked about is the simplest one: train and test on the **SAME dataset.** Essentially, the name of this approach says it all, you train the model on the

Implementing Multiple Linear Regression

Model Evaluation:

entire dataset, then you test it using a portion of the same dataset. In a general sense, when you test with a dataset in which you know the target value for each **data point, you're able to obtain a percentage of accurate predictions for the model.** This evaluation approach would most likely have a high "**training accuracy**" and a low "**out-of-sample accuracy**", since the model knows all of the testing data points from the training. What is training accuracy and out-of-sample accuracy?

We said that training and testing on the same dataset produces a high training accuracy, but what exactly is "training accuracy?"

Implementing Multiple Linear Regression

Model Evaluation:

Training accuracy is the percentage of correct predictions that the model makes when using the test dataset. However, a high training accuracy isn’t necessarily a good thing. For instance, having a high training accuracy may result in an ‘over-fit’ of the data. This means that the model is overly trained to the dataset, which may capture noise and produce a non-generalized model.

Out-of-sample accuracy is the percentage of correct predictions that the model makes on data that the model has NOT been trained on. Doing a “train and test” on the same dataset will most likely have low out-of-sample accuracy due to the likelihood of being over-fit.

Implementing Multiple Linear Regression

Model Evaluation:

It’s important that our models have high, out-of-sample accuracy, because the purpose of our model is, of course, to make correct predictions on unknown data.

So, how can we improve out-of-sample accuracy?

One way is to use another evaluation approach called "Train/Test Split."

In this approach, we select a portion of our dataset for training, for example, rows 0 to 5.

And the rest is used for testing, for example, rows 6 to 9.

The model is built on the training set. Then, the test feature set is passed to the model for prediction. And finally, the predicted values for the test set are compared with the actual values of the testing set.

Implementing Multiple Linear Regression

Model Evaluation:

This second evaluation approach, is called "Train/Test Split."

Train/Test Split involves splitting the dataset into training and testing sets, respectively, which are mutually exclusive, after which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is NOT part of the dataset that has been used to train the data. It is more realistic for real world problems. This means that we know the outcome of each data point in this dataset, making it great to test with!

And since this data has not been used to train the model, the model has no knowledge of the outcome of these data

Implementing Multiple Linear Regression

Model Evaluation:

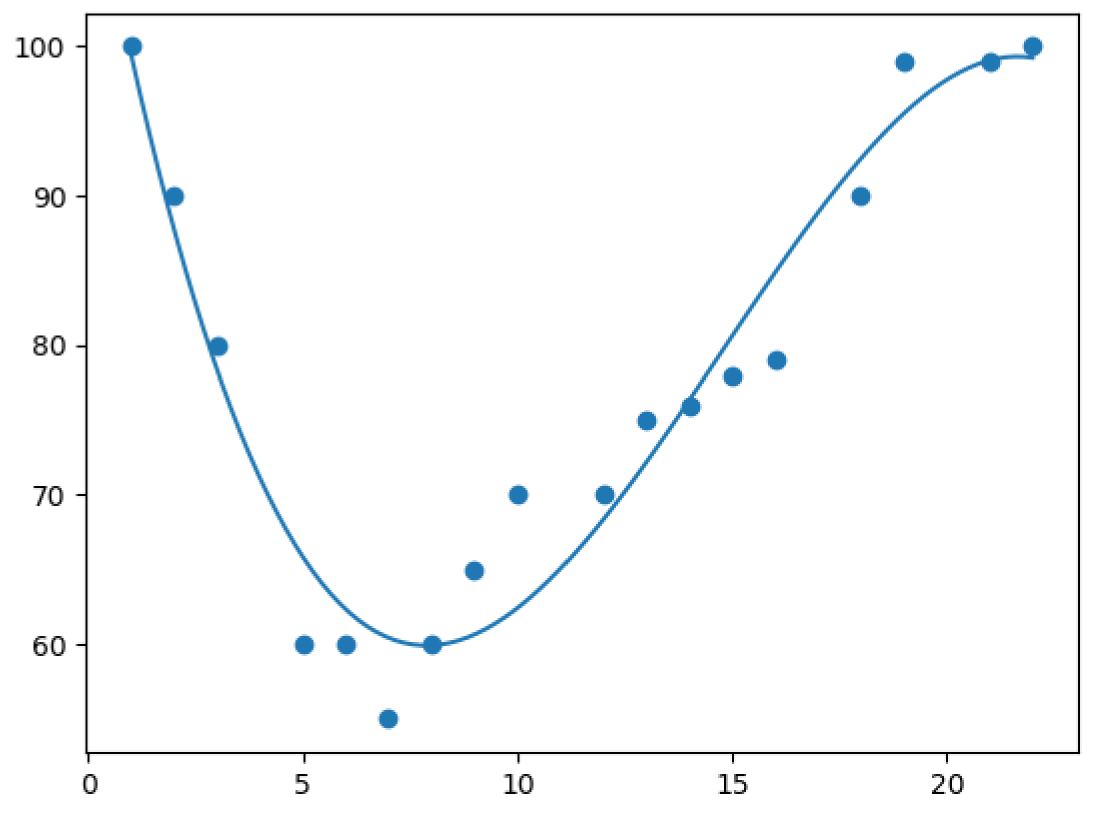
So, in essence, it’s truly out-of-sample testing.
However, please ensure that you train your model with the testing set afterwards, as you don’t want to lose potentially valuable data.

Let’s see how to practically implement "Train/Test Split."

Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for **polynomial regression**. Polynomial regression, like linear regression, uses the relationship between the variables **x** and **y** to find the best way to draw a line through the data points.

Polynomial Regression



Uplatz

Polynomial Regression

Uses of Polynomial Regression:

These are basically used to define or describe non-linear phenomenon such as:

Growth rate of tissues.

Progression of disease epidemics

Distribution of carbon isotopes in lake sediments

Advantages of using Polynomial Regression:

- Polynomial provides the best approximation of the relationship between the dependent and independent variable.
- A Broad range of function can be fit under it.
- Polynomial basically fits a wide range of curvature.

Implementing Polynomial Regression

Example:

Python has methods for finding a relationship between data-points and to draw a line of polynomial regression. I will show you how to use these methods instead of going through the mathematical formula.

In the example below, we have registered 18 cars as they were passing a certain tollbooth.

We have registered the car's speed, and the time of day (hour) the passing occurred.

Implementing Polynomial Regression

Example:
The x-axis represents the hours of the day and the y-axis represents the speed:

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]

y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

Implementing Polynomial Regression

Example: Import numpy and matplotlib then draw the line of Polynomial Regression:

NumPy has a method that lets us make a polynomial model:

mymodel = np.poly1d(np.polyfit(x, y, 3))

poly1d- A one-dimensional polynomial class.

Here 3 represent '**Degree of the fitting polynomial**'

```
import matplotlib.pyplot as plt
Import numpy as np
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(1, 22, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Implementing Polynomial Regression

R-Squared:

It is important to know how well the relationship between the values of the x- and y-axis is, if there are no relationship the polynomial regression can not be used to predict anything.

The relationship is measured with a value called the **r-squared**.

The r-squared value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Sklearn module will compute this value for you, all you have to do is feed it with the x and y arrays:

Implementing Polynomial Regression

R-Squared:
How well does my data fit in a polynomial regression?

```
from sklearn.metrics import r2_score
print(r2_score(y, mymodel(x)))
```

The result 0.94 shows that there is a very good relationship, and we can use polynomial regression in future predictions.

Implementing Polynomial Regression

Predict Future Values:

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a car that passes the tollbooth at around 17 P.M:

To do so, we need the same mymodel array from the example above:

```
speed = mymodel(17)
print(speed)
```

The example predicted a speed to be 88.87, which we also could read from the diagram.

Thank you

Uplatz