**Lab – 12: Machine Learning – Regression**

**Regression**

The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

**Linear Regression**

Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.

Chart, scatter chart

Description automatically generated

In Machine Learning, predicting the future is very important.

**How Does it Work?**

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

In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example

Start by drawing a scatter plot:

import matplotlib.pyplot as plt  
  
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]  
  
plt.scatter(x, y)  
plt.show()

Result:

Chart, scatter chart

Description automatically generated

Example

Import scipy and draw the line of Linear Regression:

import matplotlib.pyplot as plt  
from scipy import stats  
  
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]  
  
slope, intercept, r, p, std\_err = stats.linregress(x, y)  
  
def myfunc(x):  
  return slope \* x + intercept  
  
mymodel = list(map(myfunc, x))  
  
plt.scatter(x, y)  
plt.plot(x, mymodel)  
plt.show()

**Result:**

Chart, scatter chart

Description automatically generated

**Example Explained**

Import the modules you need.

You can learn about the Matplotlib module in our [Matplotlib Tutorial](https://www.w3schools.com/python/matplotlib_intro.asp).

You can learn about the SciPy module in our [SciPy Tutorial](https://www.w3schools.com/python/scipy_intro.asp).

import matplotlib.pyplot as plt  
from scipy import stats

Create the arrays that represent the values of the x and y axis:

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

Execute a method that returns some important key values of Linear Regression:

slope, intercept, r, p, std\_err = stats.linregress(x, y)

Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

def myfunc(x):  
  return slope \* x + intercept

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

mymodel = list(map(myfunc, x))

Draw the original scatter plot:

plt.scatter(x, y)

Draw the line of linear regression:

plt.plot(x, mymodel)

Display the diagram:

plt.show()

**R for Relationship**

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 the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called r.

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

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

Example

How well does my data fit in a linear regression?

from scipy import stats  
  
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]  
  
slope, intercept, r, p, std\_err = stats.linregress(x, y)  
  
print(r)

**Note:** The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

**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 10 years old car.

To do so, we need the same myfunc() function from the example above:

def myfunc(x):  
  return slope \* x + intercept

Example

Predict the speed of a 10 years old car:

from scipy import stats  
  
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]  
  
slope, intercept, r, p, std\_err = stats.linregress(x, y)  
  
def myfunc(x):  
  return slope \* x + intercept  
  
speed = myfunc(10)  
  
print(speed)

The example predicted a speed at 85.6, which we also could read from the diagram:

Chart, scatter chart

Description automatically generated

**Bad Fit?**

Let us create an example where linear regression would not be the best method to predict future values.

Example

These values for the x- and y-axis should result in a very bad fit for linear regression:

import matplotlib.pyplot as plt  
from scipy import stats  
  
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]  
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]  
  
slope, intercept, r, p, std\_err = stats.linregress(x, y)  
  
def myfunc(x):  
  return slope \* x + intercept  
  
mymodel = list(map(myfunc, x))  
  
plt.scatter(x, y)  
plt.plot(x, mymodel)  
plt.show()

Result:

Chart, scatter chart

Description automatically generated

And the r for relationship?

Example

You should get a very low r value.

import numpy  
from scipy import stats  
  
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]  
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]  
  
slope, intercept, r, p, std\_err = stats.linregress(x, y)  
  
print(r)

The result: 0.013 indicates a very bad relationship, and tells us that this data set is not suitable for linear regression.

**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.

Chart, scatter chart

Description automatically generated

**How Does it Work?**

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

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

Example

Start by drawing a scatter plot:

import matplotlib.pyplot as plt  
  
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]  
  
plt.scatter(x, y)  
plt.show()

Result:

Chart, scatter chart

Description automatically generated

Example

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

import numpy  
import matplotlib.pyplot as plt  
  
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()

Result:

Chart, scatter chart

Description automatically generated

**Example Explained**

Import the modules you need.

You can learn about the NumPy module in our [NumPy Tutorial](https://www.w3schools.com/python/numpy/default.asp).

You can learn about the SciPy module in our [SciPy Tutorial](https://www.w3schools.com/python/scipy_intro.asp).

import numpy  
import matplotlib.pyplot as plt

Create the arrays that represent the values of the x and y axis:

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]

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

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

Then specify how the line will display, we start at position 1, and end at position 22:

myline = numpy.linspace(1, 22, 100)

Draw the original scatter plot:

plt.scatter(x, y)

Draw the line of polynomial regression:

plt.plot(myline, mymodel(myline))

Display the diagram:

plt.show()

**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:

Example

How well does my data fit in a polynomial regression?

import numpy  
from sklearn.metrics import r2\_score  
  
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))  
  
print(r2\_score(y, mymodel(x)))

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

**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:

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

Example

Predict the speed of a car passing at 17 P.M:

import numpy  
from sklearn.metrics import r2\_score  
  
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))  
  
speed = mymodel(17)  
print(speed)

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

Chart

Description automatically generated

**Bad Fit?**

Let us create an example where polynomial regression would not be the best method to predict future values.

Example

These values for the x- and y-axis should result in a very bad fit for polynomial regression:

import numpy  
import matplotlib.pyplot as plt  
  
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]  
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]  
  
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))  
  
myline = numpy.linspace(2, 95, 100)  
  
plt.scatter(x, y)  
plt.plot(myline, mymodel(myline))  
plt.show()

Chart, scatter chart

Description automatically generatedResult:

And the r-squared value?

Example

You should get a very low r-squared value.

import numpy  
from sklearn.metrics import r2\_score  
  
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]  
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]  
  
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))  
  
print(r2\_score(y, mymodel(x)))

The result: 0.00995 indicates a very bad relationship, and tells us that this data set is not suitable for polynomial regression.

**Multiple Regression**

Multiple regression is like [linear regression](https://www.w3schools.com/python/python_ml_linear_regression.asp), but with more than one independent value, meaning that we try to predict a value based on **two or more** variables.

Take a look at the data set below, it contains some information about cars.

| Car | Model | Volume | Weight | CO2 |  |
| --- | --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Toyota | Aygo | 1000 | 790 | 99 |
| Mitsubishi | Space Star | 1200 | 1160 | 95 |
| Skoda | Citigo | 1000 | 929 | 95 |
| Fiat | 500 | 900 | 865 | 90 |
| Mini | Cooper | 1500 | 1140 | 105 |
| VW | Up! | 1000 | 929 | 105 |
| Skoda | Fabia | 1400 | 1109 | 90 |
| Mercedes | A-Class | 1500 | 1365 | 92 |
| Ford | Fiesta | 1500 | 1112 | 98 |
| Audi | A1 | 1600 | 1150 | 99 |
| Hyundai | I20 | 1100 | 980 | 99 |
| Suzuki | Swift | 1300 | 990 | 101 |
| Ford | Fiesta | 1000 | 1112 | 99 |
| Honda | Civic | 1600 | 1252 | 94 |
| Hundai | I30 | 1600 | 1326 | 97 |
| Opel | Astra | 1600 | 1330 | 97 |
| BMW | 1 | 1600 | 1365 | 99 |
| Mazda | 3 | 2200 | 1280 | 104 |
| Skoda | Rapid | 1600 | 1119 | 104 |
| Ford | Focus | 2000 | 1328 | 105 |
| Ford | Mondeo | 1600 | 1584 | 94 |
| Opel | Insignia | 2000 | 1428 | 99 |
| Mercedes | C-Class | 2100 | 1365 | 99 |
| Skoda | Octavia | 1600 | 1415 | 99 |
| Volvo | S60 | 2000 | 1415 | 99 |
| Mercedes | CLA | 1500 | 1465 | 102 |
| Audi | A4 | 2000 | 1490 | 104 |
| Audi | A6 | 2000 | 1725 | 114 |
| Volvo | V70 | 1600 | 1523 | 109 |
| BMW | 5 | 2000 | 1705 | 114 |
| Mercedes | E-Class | 2100 | 1605 | 115 |
| Volvo | XC70 | 2000 | 1746 | 117 |
| Ford | B-Max | 1600 | 1235 | 104 |
| BMW | 2 | 1600 | 1390 | 108 |
| Opel | Zafira | 1600 | 1405 | 109 |
| Mercedes | SLK | 2500 | 1395 | 120 |

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

**How Does it Work?**

In Python we have modules that will do the work for us. Start by importing the Pandas module.

import pandas

Learn about the Pandas module in our [Pandas Tutorial](https://www.w3schools.com/python/pandas_tutorial.asp).

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

The file is meant for testing purposes only, you can download it here: [cars.csv](https://www.w3schools.com/python/cars.csv)

df = pandas.read\_csv("cars.csv")

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']

**Tip:** It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

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.

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)

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 1300cm3:  
predictedCO2 = regr.predict([[2300, 1300]])

Example

See the whole example in action:

import pandas  
from sklearn import linear\_model  
  
df = pandas.read\_csv("cars.csv")  
  
X = df[['Weight', 'Volume']]  
y = df['CO2']  
  
regr = linear\_model.LinearRegression()  
regr.fit(X, y)  
  
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:  
predictedCO2 = regr.predict([[2300, 1300]])  
  
print(predictedCO2)

Result:

[107.2087328]

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

**Coefficient**

The coefficient is a factor that describes the relationship with an unknown variable.

Example: if x is a variable, then 2x is x two times. x is the unknown variable, and the number 2 is the coefficient.

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.

Example

Print the coefficient values of the regression object:

import pandas  
from sklearn import linear\_model  
  
df = pandas.read\_csv("cars.csv")  
  
X = df[['Weight', 'Volume']]  
y = df['CO2']  
  
regr = linear\_model.LinearRegression()  
regr.fit(X, y)  
  
print(regr.coef\_)

Result:

[0.00755095 0.00780526]

**Result Explained**

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

Weight: 0.00755095  
Volume: 0.00780526

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 cm3, 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 1300cm3 engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Example

Copy the example from before, but change the weight from 2300 to 3300:

import pandas  
from sklearn import linear\_model  
  
df = pandas.read\_csv("cars.csv")  
  
X = df[['Weight', 'Volume']]  
y = df['CO2']  
  
regr = linear\_model.LinearRegression()  
regr.fit(X, y)  
  
predictedCO2 = regr.predict([[3300, 1300]])  
  
print(predictedCO2)

Result:

[114.75968007]

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

Which shows that the coefficient of 0.00755095 is correct:

107.2087328 + (1000 \* 0.00755095) = 114.75968

**Machine Learning - Train/Test**

**Evaluate Your Model**

In Machine Learning we create models to predict the outcome of certain events, like in the previous chapter where we predicted the CO2 emission of a car when we knew the weight and engine size.

To measure if the model is good enough, we can use a method called Train/Test.

**What is Train/Test**

Train/Test is a method to measure the accuracy of your model.

It is called Train/Test because you split the the data set into two sets: a training set and a testing set.

80% for training, and 20% for testing.

You *train* the model using the training set.

You *test* the model using the testing set.

*Train* the model means *create* the model.

*Test* the model means test the accuracy of the model.

**Start With a Data Set**

Start with a data set you want to test.

Our data set illustrates 100 customers in a shop, and their shopping habits.

Example

import numpy  
import matplotlib.pyplot as plt  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
plt.scatter(x, y)  
plt.show()

Result:

The x axis represents the number of minutes before making a purchase.

The y axis represents the amount of money spent on the purchase.

Chart, scatter chart

Description automatically generated

**Split Into Train/Test**

The *training* set should be a random selection of 80% of the original data.

The *testing* set should be the remaining 20%.

train\_x = x[:80]  
train\_y = y[:80]  
  
test\_x = x[80:]  
test\_y = y[80:]

**Display the Training Set**

Display the same scatter plot with the training set:

Example

plt.scatter(train\_x, train\_y)  
plt.show()

Result:

It looks like the original data set, so it seems to be a fair selection:

Chart, scatter chart

Description automatically generated

**Display the Testing Set**

To make sure the testing set is not completely different, we will take a look at the testing set as well.

Example

plt.scatter(test\_x, test\_y)  
plt.show()

Result:

The testing set also looks like the original data set:

Chart, scatter chart

Description automatically generated

**Fit the Data Set**

What does the data set look like? In my opinion I think the best fit would be a [polynomial regression](https://www.w3schools.com/python/python_ml_polynomial_regression.asp), so let us draw a line of polynomial regression.

To draw a line through the data points, we use the plot() method of the matplotlib module:

Example

Draw a polynomial regression line through the data points:

import numpy  
import matplotlib.pyplot as plt  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
train\_x = x[:80]  
train\_y = y[:80]  
  
test\_x = x[80:]  
test\_y = y[80:]  
  
mymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))  
  
myline = numpy.linspace(0, 6, 100)  
  
plt.scatter(train\_x, train\_y)  
plt.plot(myline, mymodel(myline))  
plt.show()

Result:

Chart

Description automatically generated

The result can back my suggestion of the data set fitting a polynomial regression, even though it would give us some weird results if we try to predict values outside of the data set. Example: the line indicates that a customer spending 6 minutes in the shop would make a purchase worth 200. That is probably a sign of overfitting.

But what about the R-squared score? The R-squared score is a good indicator of how well my data set is fitting the model.

**R2**

Remember R2, also known as R-squared?

It measures the relationship between the x axis and the y axis, and the value ranges from 0 to 1, where 0 means no relationship, and 1 means totally related.

The sklearn module has a method called r2\_score() that will help us find this relationship.

In this case we would like to measure the relationship between the minutes a customer stays in the shop and how much money they spend.

Example

How well does my training data fit in a polynomial regression?

import numpy  
from sklearn.metrics import r2\_score  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
train\_x = x[:80]  
train\_y = y[:80]  
  
test\_x = x[80:]  
test\_y = y[80:]  
  
mymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))  
  
r2 = r2\_score(train\_y, mymodel(train\_x))  
  
print(r2)

**Note:** The result 0.799 shows that there is a OK relationship.

**Bring in the Testing Set**

Now we have made a model that is OK, at least when it comes to training data.

Now we want to test the model with the testing data as well, to see if gives us the same result.

Example

Let us find the R2 score when using testing data:

import numpy  
from sklearn.metrics import r2\_score  
numpy.random.seed(2)  
  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
train\_x = x[:80]  
train\_y = y[:80]  
  
test\_x = x[80:]  
test\_y = y[80:]  
  
mymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))  
  
r2 = r2\_score(test\_y, mymodel(test\_x))  
  
print(r2)

**Note:** The result 0.809 shows that the model fits the testing set as well, and we are confident that we can use the model to predict future values.

**Predict Values**

Now that we have established that our model is OK, we can start predicting new values.

Example

How much money will a buying customer spend, if she or he stays in the shop for 5 minutes?

print(mymodel(5))

The example predicted the customer to spend 22.88 dollars, as seems to correspond to the diagram:

Chart

Description automatically generated