1.0 Machine Learning – Scale

When your data has different values and measurements, it can be difficult to compare. What are kilos compared to meters? Or height compared to time?

The answer to this problem is scaling. We can scale data into new values that are easier to compare.

There are different methods for scaling data mainly normalisation or standardisation. In this tutorial, we will use standardisation. Look over Moodle for further information on both methods.

Open up cars.csv, It can be difficult to compare the volume 1.0 with the weight 790, but if we scale them both into comparable values, we can easily see how much one value is compared to the other.

The standardization method uses this formula:

z = (x - u) / s

Where z is the new value, x is the original value, u is the mean and s is the standard deviation.

If you take the weight column from the data set the first value is 790, and the scaled value will be:

(790 - 1292.23) / 238.74 = -2.1

If you take the volume column from the data set above, the first value is 1.0, and the scaled value will be:

1. - 1.61) / 0.38 = -1.59

Now you can compare -2.1 with -1.59 instead of comparing 790 with 1.0.

You do not have to do this manually, the Python sklearn module has a method called StandardScaler() which returns a Scaler object with methods for transforming data sets.

2.0 Python Scale Example

Download the cars.csv file from Moodle

import pandas as pd

from sklearn import linear\_model

from sklearn.preprocessing import StandardScaler

scale = StandardScaler()

dataset = pd.read\_csv (r'C:\Users\Paul home\Desktop\CO7426 - Machine Learning\Week 1\cars.csv')

X = dataset[['Weight', 'Volume']]

scaledX = scale.fit\_transform(X)

print(scaledX)

Result:

Text

Description automatically generated

Note that the first two values are -2.1 and -1.59, which corresponds to our calculations.

3.0 Predict CO2 Values

When the data set is scaled, you can use the scale when you predict values. Let’s predict the CO2 emissions from a 1.3-litre car that weighs 2300 kilograms.

import pandas as pd

from sklearn import linear\_model

from sklearn.preprocessing import StandardScaler

scale = StandardScaler()

dataset = pd.read\_csv (r'C:\Users\Paul home\Desktop\CO7426 - Machine Learning\Week 1\cars.csv')

X = dataset[['Weight', 'Volume']]

y = dataset['CO2']

scaledX = scale.fit\_transform(X)

regr = linear\_model.LinearRegression()

regr.fit(scaledX, y)

scaled = scale.transform([[2300, 1.3]])

predictedCO2 = regr.predict([scaled[0]])

print(predictedCO2)

Results:



3.1 Create a loop to find the values for all the cars in the data set

4.0 Machine Learning - Train/Test

In Machine Learning we create models to predict the outcome of certain events, like in the previous exercise where we predicted the CO2 emission of a car when we knew the weight and engine size. To measure if the model is a good fit, we can use a method called Train/Test.

Train/Test is a method to measure the accuracy of your model. It is called Train/Test because you split 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 to create the model.
* Test the model means to test the accuracy of the model.

Download the TrainTest.csv file from Moodle

import pandas as pd

import numpy

import matplotlib.pyplot as plt

x = numpy.random.normal(3, 1, 100)

y = numpy.random.normal(150, 40, 100) / x

# numpy.random.normal We are using random data here – your results may differ slightly to mine.

plt.scatter(x, y)

plt.show()

Result:

Chart, scatter chart

Description automatically generated

4.1 Can we come up with any conclusions from this data?

The training set should be a random selection of 80% of the original data and the testing set should be the remaining 20%. To do this we can add the below code:

train\_x = x[:80]

train\_y = y[:80]

test\_x = x[80:]

test\_y = y[80:]

4.2 Plot the training set. Does this look similar to the original data? Does this mean it is a fair selection?

4.3 Do the same for the test data.

5.0 Fit the Data Set

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.

NumPy has methods for finding a relationship between data points and drawing a line of polynomial regression.

mypolymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))

polyline = numpy.linspace(1, 7, 100)

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

plt.plot(polyline, mypolymodel(polyline))

plt.show()

Result:

Chart, scatter chart

Description automatically generated

4.4 What does the 4 in this code mean? What happens when we change this number?

“mypolymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))”

4.5 How would you decide on a number here?

5.0 R2 - R-squared

The result can back the suggestion of the data set fitting a polynomial regression well. The R-squared score is a good indicator of how well the data set is fitting the model.

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.

from sklearn.metrics import r2\_score

#Import the above library

mypolymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))

#method that lets us make a polynomial mode

polyline = numpy.linspace(0, 6)

#Then specify how the line will display, we start at position 1, and end at position 6:

r2 = r2\_score(train\_y, mypolymodel(train\_x))

print(r2)

**Result:**

Chart

Description automatically generated



A result of 0.9 shows a good relationship, this means the model is a good fit. The below image shows us what a bad fir may look like.

Chart, scatter chart

Description automatically generated

The above diagram had an r-squared value of 0.00995 this suggests that the polynomial regression model is not a good fit here.

**6.0 Testing set**

As we have made a model that seems to fit the data well, we can move on to testing the data with the testing data. This will allow us to see if we get the same result. Here we will use the [sklearn.metrics.r2\_score library](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html), using the below code:

r2testdata = r2\_score(test\_y, mypolymodel(test\_x))

print(r2testdata)

Result:

Text

Description automatically generated

The first value is different to the last task 0.901 because we are using random data samples. However, we can see it is still a good fit and the test data is 0.727. This shows that the model fits the testing set as well, and we are confident that we can use the model to predict future values.

**7.0 Predict Values**

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

How much money will a buying customer spend, if she or he stays in the shop for 4 minutes?

print(mypolymodel(4))

Result:

Text

Description automatically generated

What result do you get for:

1. 3 minutes
2. 4.54 minutes
3. 22.54 minutes

Do all these values seem to fit? Why or why not?

\*The results in these exercises will change every time as we are using random NumPy data. This is not ideal, but it shows you’re how the process works. Throughout this module, we will investigate models like this in more detail.