linearRegressionTutorial

March 2, 2020

0.1 Linear Regression

In this tutorial we will fit a line and then a curve to velocity vs time measurement of a F1 car. We use linear regression class from Pyrhon scikit-learn package. This class uses least square method to determine the curve parameters (the thetas in the presentation). If you are not familiar with Pandas and Numpy packages, there are some introductory commands in this tutorial.

Important : You should complete the code where ever there is a "Task" or a "Question".

0.2 Pandas

It is a Python library for data cleaning, manipulation and analysis. This package can read data in different formats such as text, csv, excel etc. The structured data are kept in data frame format in Pandas. You can apply aggregation, union, join etc operations on data. Pandas facilitates managing the null values. This package is compatible with Numpy.

[ ]:
```
# oepning the data
import pandas as pd
# Task : write your path to carF1.csv file
path =
df = pd.read_csv(path)
```

[ ]:
```
# Task : print the df contents :
```

[ ]:
```
# Task : print the following commands and write about their functionality

# type(df)

# df.dtypes

# df.columns
```

0.3 Data Cleaning

Before starting to manipulate or analysis the data, you should be sure that the data has no defects and bad values. You may have already realised that there are NaN (Not a Number) values for the velocity column. These values should be removed from the data.

[ ]:
```
# Drop the NaN Values :
df = df.dropna()
```

[ ]:
```
# Task : print df
```

[ ]:
```
# Task : print the following commands and write what they do

# df["Time"]

# df["Velocity"]

# df.iloc[5,1]

# df.iloc[5, :]

# df.iloc[0:6,1]

# df.iloc[0, :]

# df.iloc[:, 0]

# df.iloc[:, 1]

# df.iloc[:,-1]

# df.iloc[:,-2]
```

0.4 Matplotlib

It is a Python library for plotting the data. You can plot data points, curves, histograms, profiles, 3D plots etc by using this library.

[ ]:
```
# Plotting the data
from matplotlib import pylab as plt
plt.scatter(df["Time"], df["Velocity"])
plt.xlabel("Time (s)")
plt.ylabel("Velocity (km/h)")
plt.title("Car velocity curve")
plt.show()
```

[ ]:
```
# Question : after how many seconds the car velocity reaches 100 km/h ?
```

0.5 Numpy

This Python library is designed for scintific calculations. It contains n-dimentional arrays and matrices as well as high-level mathematical functions to manipulate them. We will use this library to perform linear algebra operations.

[ ]:
```
# Converting Pandas data frame to Numpy ndarray :
import numpy as np
data_array = df.values
```

[ ]:
```
# Task : run the following commands

# print the type of data_array

# data_array.shape

# data_array

# data_array[:,:]

# data_array[20,:]

# data_array[:,-1]

# data_array[:,:-1]
```

We split the data into a feature array (X) and an array of dependent variable (y) :

[ ]:
```
X = data_array[:,:-1]
y = data_array[:,-1]

# Task : print the shape of X and y
```

Attention : although both X and y are one-dimensional arrays but their printed shape is different. We have chosen in a way that for Python, X is a matrix (one0column matrix) and y is a vector.

[ ]:
```
# Task : print X and y
```

0.6 scikit-learn

It is a Machine Learning library for Python. It includes supervised and unsupervised classifiers such as Logistic Regression, MultiLayer Perceptrons, Random Forest, Gradient Boosting, K-means etc. We use this library for Linear Regression.

As you saw in the introductory presentation, in supervised learning, we select a trainig set with known features (array of independent variables X) and known measurements (array of dependent variable y) from the data. Furthermore, we select a test set from the same data, with known X and y, to verify the accuracy of the fit.

[ ]:
```
# selecting the trainig and test sets from the data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3,␣
↪random_state = 2)
```

```
# We have taken the 70% of the data to the training set and 30% to the test set.
# The random_state choose the random selection method. We put an integer to␣
 ↪have the
# same results for different users.
# Use help(train_test_split) to see the options
```

[ ]:
```
# Task : what are the type and shape of X_train, X_test, y_train and y_test ?
```

[ ]:
```
# Line fit
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_train, y_train)

# regressor is an instance of LinearRegression class. The method "fit" computes␣
 ↪the
# theta_0 and theta_1 by minimising the cost function by considering X_train␣
 ↪and y_train
```

[ ]:
```
# The fit of linear model can make a prediction for velocities of the test set
y_test_pred = reg.predict(X_test)
```

[ ]:
```
# Let's compare the model prediction for the test set to real data (y_test) :
plt.scatter(X_test, y_test)
plt.scatter(X_test, y_test_pred, c='r')
```

[ ]:
```
# Task : make similar plots for the whole data (X and y). what do you conclude ?
```

[ ]:
```
# Question : the velocity values are given until time = 10 s. Can you predict␣
 ↪the velocity
# at t = 12s by using your linear model ?
# To answer this question we make a feature array containing the time value :
X_12 = np.array([12]).reshape(-1,1)
# Now you do the prediction :

# What is the velocity at t=2.82 s ?
```

[ ]:
```
# The score is a number that shows you the reliability of the fit. If it is␣
 ↪close to 1, the
# fit is to be considered as "good". If it is close to 0 it os not reliable.

reg.score(X_test, y_test)
```

[ ]:
```
# You can have access to the fit parametes (thetas) :

theta_0 = reg.intercept_
```

```
theta_1 = reg.coef_[0]

# Task : print theta_0 and theta_1
```

[ ]:
```
# Task : run and explaine this code :
t = np.linspace(0,11,500)
v = theta_0 + theta_1 * t
plt.scatter(df["Time"], df["Velocity"], c='b')
plt.scatter(t, v, marker='.', c='r', alpha=.1)
```

0.7 Polynomial Regression

So far we have considered the data with linear behaviours where the linear model gives the best data fit. We may have data with more sophisticated variations that are not well described by a linear model. in this case the cost function (qui-square) minimisation does not make the best fit score. This means to find the best curve fitting the data, we should apply non-linear models (E.g polynomial models) to the data.

Let's continue with the car velocity versus time example. We consider the effect of air resitance against the car movement. This effect can be more important in windy weather. In this case the velocity depends not only on time but also the square of time. This means velocity depends on the second degree of time : y = theta_0 + theta_1 * x + theta_2 * x^2 where y is the velocity and x is the time respectively.

[ ]:
```
# Task : open the carF1_wind.csv in a Pandas data frame
df_wind =
```

[ ]:
```
# Task : show the firt 4 lines of the df_wind
```

[ ]:
```
# Task : plot as follow
plt.scatter(df_wind["time"], df_wind["velocity"], c="b")
plt.scatter(df_wind["time"], df_wind["velocity_wind"], c="r")
plt.xlabel("Time (s)")
plt.ylabel("Velocity (km/h)")
plt.title("Car velocity with and without wind")
plt.show()
```

The blue points are the velocity vs time in the absence of the wind. You can see that the velocity depends on time perfectly linearly. In contrast, the red points define a curve rather than a line because the air resistance add another dependency on velocity that is proportional to time square.

We are going to find the parameters for the curve to be fitted to red points. Here we have two features (time and square of time) hence, there will be three parameters (theta_0, theta_1 and theta_2).

[ ]:
```
# Task : convert the df_wind to Numpy array
data2_array =
```

```
[ ]: # The first column of data_array contains the feature  (time)
     # Last column is the velocity in windy weather (y array)
     X = data2_array[:,:1]
     y = data2_array[:,-1]
```

We now that the velocity depends on higher degree of time (here time square). So, we should add this dependency as a new feature to the X-matrix :

```
[ ]: from sklearn.preprocessing import PolynomialFeatures
     # we define the instance by giving the polynomial degree
     poly_deg = PolynomialFeatures(degree=2)
     X_poly = poly_deg.fit_transform(X)
```

```
[ ]: # Task : print X_poly vbalues and explain what happend
```

```
[ ]: # Task : split the data into training and test sets
     # Attention : you should split X_poly instead of X
     from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test =
```

```
[ ]: # Task : define the linear regression instance and fit the X_train and y_train
     from sklearn.linear_model import LinearRegression
     reg3 =
```

```
[ ]: # Task : compute the score for the test set
```

```
[ ]: # Task : make a prediction for whole data
     y_pred =
```

```
[ ]: # Task : plot the curve on your data
     plt.scatter(X[:,0], y, c='r')
     plt.plot(X[:,0], y_pred, c='black')
     plt.show()
```

0.8 Linear Regression with Multiple Independent Variables (Features)

So far we had only one independent variable (one feature). It was the time. The velocity depended only on time. In general, the dependent variable can depend on multiple features. For example the price of an appartement can depend linearly on the surface, floor number and the district: y = theta_0 + theta_1 * x_1 + theta_2 * x_2 + theta_3 * x_3

```
[ ]: # Task : open the rentPriceNaN.csv as a Pandas dataframe :
     rent =
     # Task : print the rent dataframe, what are these data about ? :
```

```
[ ]: # Task : convert the data into numpy array :
     rent_array =
```

You may have noticed that there are nan values in “surface” column. This time we won’t drop the nan values. Instead we replace them by the mean value of the surface.

Attention : replacing the missing data with statistical values are not generally recommanded specially when you have lots of missing values in a column.

[ ]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy = "mean")
# we calculate the mean values for the corresponding columns :
rent_array[:, 0:1] = np.round(imputer.fit_transform(rent_array[:, 0:1]))
```

[ ]:
```
# Task : print the rent_array and verfy that the nan values are replaced :
```

[ ]:
```
# Task : similar to the last section,
#        make the feature matrix and the dependent variable vector :
X =
y =
```

The third column (index=2) of the X contains the district name which is a categorical variable. There are two districts in the data : “A” and “B”. The categorical variables has string type and can not be used by the mathematical ooperations. We hence convert them to digits :

[ ]:
```
from sklearn.preprocessing import LabelEncoder
labcode = LabelEncoder()
X[:,2] = labcode.fit_transform(X[:,2])
```

[ ]:
```
# Task : print the feature matrix and say what happend to district column
```

The district name are now digits with value of either 0 or 1. They are ordinal numbers. We can not say that a district has a larger value than the other! To solve this problem one should split the district column into two columns of dummy variables :

[ ]:
```
from sklearn.preprocessing import OneHotEncoder
onehotencoder = OneHotEncoder(categorical_features=[2])
X = onehotencoder.fit_transform(X).toarray()
```

[ ]:
```
# Task : print the X matrix and say how the dsitricts are splitted into two␣
↪columns :
```

You may have realised that the two splitted columns are not independent. If one district in spilitted to values 0, 1 the other district has certaonly 1, 0 values. So we can remove one of the splitted columns :

[ ]:
```
X = X[:, 1:]
```

When you have multiple features you may need to re-scale them. This is neccesary when features differ by order of magnitudes from each other. E.g in our example the floor spans from 1 to 3 while surface spanse from 15 to 30 square meter. This means our features can be different by a factor of 10. This factor can have very larger values than our example.

This is a common case in multiple regression. Re-scaling means that we normalise all features so they all span the same numerical interval (e.g from -1 to +1). Without re-scaling, If some feutures spans very larger values than the others, then during the fit the features with smaller values will have a small weight parameter (theta) and will be effectively discarded from the fit.

One solution is that for each feature we subtract all its values from the mean value and divide them by the standard deviation. This is done by StandardScaler class of Scikit-learn.

[ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_sc = sc.fit_transform(X) # X_sc is the scaled copy of X
```

[ ]:
```
# Task : explaine the effect of scaling by running the following code
print("Min/Max for Time : {}, {}".format( X[:,0].min(), X[:,0].max() ) )
print("Min/Max for scaled Time : {}, {}".format( X_sc[:,0].min(), X_sc[:,0].
 ↪max() ) )
print("---------------------------------")
print("Min/Max for Time square : {}, {}".format( X[:,1].min(), X[:,1].max() ) )
print("Min/Max for scaled Time swuare : {}, {}".format( X_sc[:,1].min(), X_sc[:
 ↪,1].max() ) )
```

[ ]:
```
# Task : make your training and test sets similar to the previous section.
# Attention : you should use scaled feature matrix instead of X
```

[ ]:
```
# Task : make linear regression instance and fit your training set
reg3 =
```

[ ]:
```
# Task : compare the scores for the test and the training sets
```

[ ]:
```
# Task : compute the price of an appartement with surface of 21.5 m^2, at␣
 ↪second floor
#        located at 'A' dsitrict.
```

0.9 Exercise 1

This is an example for predicting the missed data.

From data frame df_wind select only two columns time and velocity and remove the velocity data for 4 s $<$ time $<$ 6 s.

Fit the linear model.

Plot the data and the fitted line.

0.10 Exercise 2

This is an example for traning set manipulation.

From data frame df_wind select only two columns time and velocity.

Replace the velocity data for 4 s $<$ time $<$ 6 s by the mean velocity computed from the other lines.

Plot the velocity vs. time.

Make a linear fit to these columns.

Plot the data and the fitted line.

What do you conclude ?

0.11 Exercise 3

This is an example of a training set with low statistics (over fitting).

Consider carF1.csv and take only 10% of the data for training.

Make the fit and compute the score for both the training and the test sets.

Plot the data and the fitted line together.

What do you conclude ?

0.12 Exercise 4

This is an example for a bad model usage where the features are not well selected (under fitting).

Consider the data carF1_wind.csv. Take only these columns : time and velocity_wind.

Fit a straight line to these columns and compute the score and compare it to the polynomial fit score.

Plot the data and the fitted line together.

What do you conclude ?