Learn To Make Prediction By Using Multiple Variables

**Introduction** :

The goal of the blogpost is to equip beginners with basics of Linear Regression algorithm having multiple features and quickly help them to build their first model. This is also known as multivariable Linear Regression. We will mainly focus on the modeling side of it . The data cleaning and preprocessing parts would be covered in detail in an upcoming post.

A multivariable model can be thought of as a model in which multiple variables are found on the right side of the model equation. This type of statistical model can be used to attempt to assess the relationship between a number of variables. A simple linear regression model has a continuous outcome and one predictor, whereas a multiple or multivariable linear regression model has a continuous outcome and multiple predictors (continuous or categorical). A simple linear regression model would have the form



a multivariable or multiple linear regression model would take the form



where y is a continuous dependent variable, x is a single predictor in the simple regression model, and x1, x2, …, xk are the predictors in the multivariable model.

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is estimated.

Multivariable linear regression can can model more complex relationship which comes from various features together. They should be used in cases where one particular variable is not evident enough to map the relationship between the independent and the dependent variable.

Let’s work on a case study to understand this better.

**Problem Statement :**

To predict the relative performance of a computer hardware given other associated attributes of the hardware.

**Data details**

Computer Hardware dataset  
===========================  
  
URL : <https://archive.ics.uci.edu/ml/datasets/Computer+Hardware>

1. Title: Relative CPU Performance Data   
  
2. Source Information  
 -- Creators: Phillip Ein-Dor and Jacob Feldmesser  
 -- Ein-Dor: Faculty of Management; Tel Aviv University; Ramat-Aviv;   
 Tel Aviv, 69978; Israel  
 -- Donor: David W. Aha (aha@ics.uci.edu) (714) 856-8779   
 -- Date: October, 1987  
  
3. Past Usage:  
 1. Ein-Dor and Feldmesser (CACM 4/87, pp 308-317)  
 -- Results:   
 -- linear regression prediction of relative cpu performance  
 -- Recorded 34% average deviation from actual values   
 2. Kibler,D. & Aha,D. (1988). Instance-Based Prediction of  
 Real-Valued Attributes. In Proceedings of the CSCSI (Canadian  
 AI) Conference.  
 -- Results:  
 -- instance-based prediction of relative cpu performance  
 -- similar results; no transformations required  
 - Predicted attribute: cpu relative performance (numeric)  
  
4. Relevant Information:  
 -- The estimated relative performance values were estimated by the authors  
 using a linear regression method. See their article (pp 308-313) for  
 more details on how the relative performance values were set.  
  
5. Number of Instances: 209   
  
6. Number of Attributes: 10 (6 predictive attributes, 2 non-predictive,   
 1 goal field, and the linear regression guess)  
  
7. Attribute Information:  
 1. vendor name: 30   
 (adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec,   
 dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson,   
 microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry,   
 sratus, wang)  
 2. Model Name: many unique symbols  
 3. MYCT: machine cycle time in nanoseconds (integer)  
 4. MMIN: minimum main memory in kilobytes (integer)  
 5. MMAX: maximum main memory in kilobytes (integer)  
 6. CACH: cache memory in kilobytes (integer)  
 7. CHMIN: minimum channels in units (integer)  
 8. CHMAX: maximum channels in units (integer)  
 9. PRP: published relative performance (integer)  
 10. ERP: estimated relative performance from the original article (integer)  
  
8. Missing Attribute Values: None  
  
9. Class Distribution: the class value (PRP) is continuously valued.  
 PRP Value Range: Number of Instances in Range:  
 0-20 31  
 21-100 121  
 101-200 27  
 201-300 13  
 301-400 7  
 401-500 4  
 501-600 2  
 above 600 4  
  
Summary Statistics:  
 Min Max Mean SD PRP Correlation  
 MCYT: 17 1500 203.8 260.3 -0.3071  
 MMIN: 64 32000 2868.0 3878.7 0.7949  
 MMAX: 64 64000 11796.1 11726.6 0.8630  
 CACH: 0 256 25.2 40.6 0.6626  
 CHMIN: 0 52 4.7 6.8 0.6089  
 CHMAX: 0 176 18.2 26.0 0.6052  
 PRP: 6 1150 105.6 160.8 1.0000  
 ERP: 15 1238 99.3 154.8 0.9665

**Tools used** :

Pandas , Numpy , Matplotlib , scikit-learn

**Python Implementation with code :**

**Import necessary libraries**

Import the necessary modules from specific libraries.

|  |
| --- |
| import numpy as np  import pandas as pd  %matplotlib inline  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn import datasets  from sklearn.metrics import mean\_squared\_error  from sklearn import linear\_model |

**Load the data set**

Use pandas module to read the taxi data from the file system. Check few records of the dataset.

|  |
| --- |
| names = ['VENDOR','MODEL\_NAME','MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN', 'CHMAX', 'PRP', 'ERP' ];  data = pd.read\_csv('data/computer-hardware/machine.data',names=names)  data.head()  VENDOR MODEL\_NAME MYCT MMIN MMAX CACH CHMIN CHMAX PRP ERP  0 adviser 32/60 125 256 6000 256 16 128 198 199  1 amdahl 470v/7 29 8000 32000 32 8 32 269 253  2 amdahl 470v/7a 29 8000 32000 32 8 32 220 253  3 amdahl 470v/7b 29 8000 32000 32 8 32 172 253  4 amdahl 470v/7c 29 8000 16000 32 8 16 132 132 |

**Feature Selection :**

Let’s select only the numerical fields for model fitting.

|  |
| --- |
| data.info()  <class 'pandas.core.frame.DataFrame'> RangeIndex: 209 entries, 0 to 208 Data columns (total 10 columns): VENDOR 209 non-null object MODEL\_NAME 209 non-null object MYCT 209 non-null int64 MMIN 209 non-null int64 MMAX 209 non-null int64 CACH 209 non-null int64 CHMIN 209 non-null int64 CHMAX 209 non-null int64 PRP 209 non-null int64 ERP 209 non-null int64 dtypes: int64(8), object(2) |

We can see that barring first two variables rest of them are numeric in nature. Let’s only pick the numeric fields.

|  |
| --- |
| categorical\_ = data.iloc[:,:2]  numerical\_ = data.iloc[:,2:]  numerical\_.head()  MYCT MMIN MMAX CACH CHMIN CHMAX PRP ERP  0 125 256 6000 256 16 128 198 199  1 29 8000 32000 32 8 32 269 253  2 29 8000 32000 32 8 32 220 253  3 29 8000 32000 32 8 32 172 253  4 29 8000 16000 32 8 16 132 132 |

**Select the predictor and target variables**

|  |
| --- |
| X = data.iloc[:,:-1]  y = data.iloc[:,-1] |

**Train test split :**

|  |
| --- |
| x\_training\_set, x\_test\_set, y\_training\_set, y\_test\_set = train\_test\_split(X,y,test\_size=0.10,  random\_state=42,  shuffle=True) |

**Normalize the data :**

Before we do the fitting , let’s normalize the data so that the data is centered around the mean and has unit standard deviation.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  # Fit on training set only.  scaler.fit(x\_training\_set)  # Apply transform to both the training set and the test set.  x\_training\_set = scaler.transform(x\_training\_set)  x\_test\_set = scaler.transform(x\_test\_set) |

|  |
| --- |
| y\_training\_set = y\_training\_set.values.reshape(-1, 1)  y\_test\_set = y\_test\_set.values.reshape(-1, 1)  y\_scaler = StandardScaler()  # Fit on training set only.  y\_scaler.fit(y\_training\_set)  # Apply transform to both the training set and the test set.  y\_training\_set = y\_scaler.transform(y\_training\_set)  y\_test\_set = y\_scaler.transform(y\_test\_set) |

**Training / model fitting :**

Fit the model to selected supervised data

|  |
| --- |
| model = linear\_model.LinearRegression()  model.fit(x\_training\_set,y\_training\_set) |

**Model parameters study :**

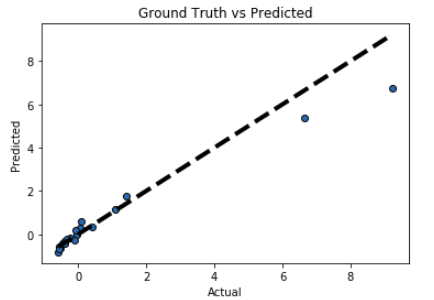
The coefficient R^2 is defined as (1 - u/v), where u is the residual sum of squares ((y\_true - y\_pred) \*\* 2).sum() and v is the total sum of squares ((y\_true - y\_true.mean()) \*\* 2).sum().

|  |
| --- |
| from sklearn.metrics import mean\_squared\_error, r2\_score  model\_score = model.score(x\_training\_set,y\_training\_set)  # Have a look at R sq to give an idea of the fit ,  # Explained variance score: 1 is perfect prediction  print(“ coefficient of determination R^2 of the prediction.: ',model\_score)  y\_predicted = model.predict(x\_test\_set)  # The mean squared error  print("Mean squared error: %.2f"% mean\_squared\_error(y\_test\_set, y\_predicted))  # Explained variance score: 1 is perfect prediction  print('Test Variance score: %.2f' % r2\_score(y\_test\_set, y\_predicted))  Coefficient of determination R^2 of the prediction : 0.9583846753218253 Mean squared error: 0.39 Test Variance score: 0.93 |

**Accuracy report with test data :**

Let’s visualise the goodness of the fit with the predictions being visualised by a line

|  |
| --- |
| # So let's run the model against the test data  from sklearn.model\_selection import cross\_val\_predict  fig, ax = plt.subplots()  ax.scatter(y\_test\_set, y\_predicted, edgecolors=(0, 0, 0))  ax.plot([y\_test\_set.min(), y\_test\_set.max()], [y\_test\_set.min(), y\_test\_set.max()], 'k--', lw=4)  ax.set\_xlabel('Actual')  ax.set\_ylabel('Predicted')  ax.set\_title("Ground Truth vs Predicted")  plt.show() |



**Conclusion** :

We can see that our R2 score and MSE are both very good. This means that we have found a best fitting model to predict the median price value of a house. There can be further improvement to the metric by doing some preprocessing before fitting the data. However the task for the post was to provide you sufficient knowledge to implement your first model. You can build over the existing pipeline and report your accuracies.