Simple Regression Problem

January 26, 2017

1 Overview

This is a typical albeit oversimplfied data science workflow. We look for insights in historical data, use those data to build a model which we then want to apply to future data. In this trivial example we have been given 5 time series sensor values and asked if we can predict the value for sensor 62 based on the values of the other four sensors. We decide to try a simple linear regression model.

First we need to ingest the data signals from their raw format, in this case two column csv files (TimeStamp, Value) one for each sensor over a 2 year period 2013-14 with a reading every 5 minutes. We read these into memory using the data science favourite pandas:

```
In [1]: import pandas as pd
```

```
In [2]: df_59 = pd.read_csv('regression_data/59.csv')
        df_60 = pd.read_csv('regression_data/60.csv')
        df_61 = pd.read_csv('regression_data/61.csv')
        df_62 = pd.read_csv('regression_data/62.csv')
        df_63 = pd.read_csv('regression_data/63.csv')

        #merge into a single data frame
        df_master = pd.concat([df_59['Value'], df_60['Value'], df_61['Value'],
                               df_62['Value'], df_63['Value']], axis=1)
        df_master.columns = ['val59','val60','val61','val62','val63']
        df_master
```

Out[2]:

	val59	val60	val61	val62	val63
0	16.107043	12.189274	10.761665	16.223043	55.467606
1	16.099436	12.207525	10.763932	16.219148	109.283522
2	16.103656	12.197332	10.829815	16.204840	109.293267
3	16.088023	12.224396	10.810413	16.198673	109.224787
4	16.056496	12.241735	10.826908	16.185203	109.356033
5	16.031169	12.236260	10.885047	16.144411	109.280784
6	16.064469	12.247371	10.870325	16.137030	109.313956
7	16.071302	12.219455	10.871628	16.130944	109.247048
8	16.070241	12.216692	10.841967	16.137975	109.219159
9	16.051955	12.193931	10.851947	16.149161	109.196147
10	16.029528	12.187809	10.888565	16.119112	109.209395
11	16.041749	12.182543	10.861268	16.108069	109.155831

12	16.016852	12.190664	10.834459	16.107514	109.165706
13	15.998481	12.168954	10.847254	16.089623	109.150211
14	15.994811	12.161150	10.834749	16.074165	109.164861
15	16.001709	12.159081	10.832419	16.062841	109.054025
16	16.021725	12.138906	10.830707	16.083527	109.036083
17	16.005508	12.136718	10.851341	16.090044	109.151662
18	15.981607	12.151525	10.833069	16.065807	109.051114
19	15.957761	12.141891	10.802820	16.044322	108.981237
20	15.940294	12.148204	10.803782	16.021136	108.883116
21	15.886960	12.166979	10.810548	16.018541	108.857433
22	15.902787	12.187023	10.821789	16.011570	108.906452
23	15.893399	12.197994	10.837707	16.010774	109.061612
24	15.848732	12.211411	10.835005	16.019167	108.981630
25	15.828734	12.192749	10.820144	16.032889	108.863926
26	15.809561	12.165388	10.837333	15.982124	108.767811
27	15.833828	12.153324	10.843030	16.012088	108.854755
28	15.816231	12.149347	10.819841	15.968956	108.788767
29	15.800765	12.168950	10.839499	15.980703	108.721034
...	...	...	...	...	...
210209	22.758929	19.304658	5.647323	24.926287	126.680626
210210	22.734814	19.260953	5.628436	24.901666	126.584623
210211	22.747625	19.242826	5.648806	24.914006	126.655916
210212	22.780584	19.255667	5.613513	24.919381	126.542663
210213	22.794182	19.246557	5.602674	24.888173	126.515810
210214	22.812912	19.255592	5.606056	24.907579	126.584765
210215	22.801160	19.231027	5.623659	24.889446	126.537889
210216	22.830027	19.246387	5.620311	24.912811	126.666393
210217	22.824237	19.199441	5.546067	24.927435	126.562677
210218	22.805882	19.199382	5.542644	24.951225	126.411191
210219	22.834666	19.200980	5.532143	24.939630	126.499107
210220	22.844289	19.219189	5.515989	24.934549	126.512486
210221	22.863307	19.206392	5.521974	24.888095	126.529550
210222	22.876360	19.180068	5.477316	24.886017	126.419606
210223	22.897409	19.182863	5.504722	24.877546	126.412507
210224	22.924011	19.212796	5.529496	24.872314	126.471051
210225	22.951730	19.226608	5.517308	24.887357	126.569907
210226	22.962569	19.219637	5.531897	24.896951	126.597471
210227	22.983255	19.208982	5.532762	24.902938	126.642819
210228	22.954334	19.232322	5.525280	24.900792	126.616434
210229	22.971257	19.196115	5.543716	24.900852	126.567699
210230	22.977711	19.171425	5.519678	24.892266	126.516628
210231	22.966350	19.172331	5.499619	24.882280	126.552986
210232	22.979586	19.185902	5.504332	24.883647	126.571484
210233	22.990389	19.171426	5.504297	24.882469	126.550471
210234	23.008548	19.160931	5.497235	24.870872	126.482443
210235	23.033286	19.171047	5.518670	24.882311	126.530720
210236	23.047332	19.166235	5.508800	24.904723	126.570327
210237	23.030544	19.197149	5.531400	24.924798	126.622914

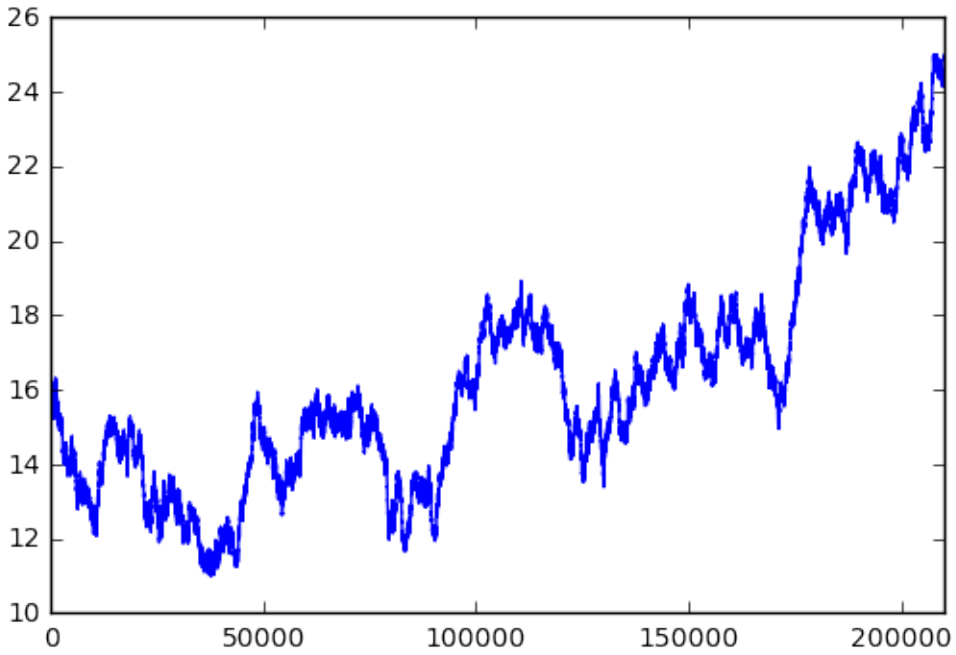
```
210238  23.041559  19.202965   5.523794  24.953947  126.745179

[210239 rows x 5 columns]
```

At this point we might want to do some basic visualisation, this is pretty straight forward for simple inspection but if we wanted to navigate the data in specific time periods, etc. there would be a lot of code bloat to do that with the usual data science plotting tools. We might end up dumping multiple plots to a folder then putting those into a presentation format for further inspection:

```
In [3]: %matplotlib inline
        df_master['val62'].plot()

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x119aba6d8>
```



Now we think about training and testing our model, there are a number of ways to split the data but as this is time series we choose to split down the middle:

```
In [5]: df_train=df_master.iloc[:100500,:]
        df_test = df_master.iloc[100500:,:]
```

Now we define the X and the Y features for the regression model for both the training and the testing data:

```
In [6]: train_X = df_train[['val59','val60','val61','val63']].as_matrix()
        train_Y = df_train['val62'].values

        test_X = df_test[['val59','val60','val61','val63']].as_matrix()
        test_Y = df_test['val62'].values
```

We train a simple linear regression model using every data scientist's favourite package scikit-learn:

```
In [7]: from sklearn import linear_model, metrics

        regr = linear_model.LinearRegression()
        regr.fit(train_X,train_Y)

Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False
```

Now we can check how well our model does on predicting the next batch of data by comparing the predicted and actual values using a typical metric like the coefficient of determination:?

```
In [8]: pred_Y = regr.predict(test_X)
        metrics.r2_score(test_Y,pred_Y)

Out[8]: 0.99538448535996171
```

We see the model makes very good predictions! (A little too good but this is just basic simulated data...) As our model is so good we would like to save it and the apply it to any further data we might be presented with.

```
In [9]: import pickle

In [10]: pickle.dump( regr, open( "regr.p", "wb" ) )

In [11]: import_model = pickle.load( open( "regr.p", "rb" ) )
         new_pred_Y = import_model.predict(test_X)
         metrics.r2_score(test_Y,new_pred_Y)

Out[11]: 0.99538448535996171
```

Single line prediction:

```
In [11]: regr.predict([[23.041559,19.202965,5.523794,126.745179]])

Out[11]: array([ 24.61394289])

In [ ]:
```