Pollution

May 22, 2018

In [183]:
```
from pandas import read_csv
from matplotlib import pyplot
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from numpy import concatenate
```

In [184]:
```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

In [185]:
```
# load dataset
dataset = read_csv('pollution.csv', header=0, index_col=0)
values = dataset.values
# integer encode direction
```

```
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])

# ensure all data is float
values = values.astype('float32')

reframed = series_to_supervised(values, 1, 1)
#  normalize features - this will make it easier to interpret regression coefficnts
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
print(reframed.head())
```

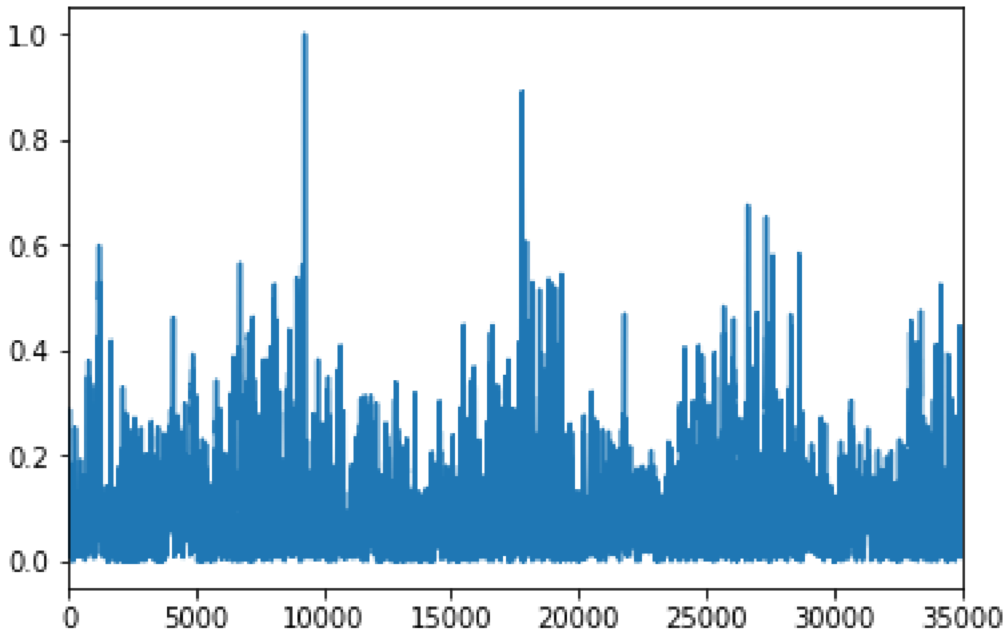
```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.129779   0.352941   0.245902   0.527273   0.666667   0.002290
2   0.148893   0.367647   0.245902   0.527273   0.666667   0.003811
3   0.159960   0.426471   0.229508   0.545454   0.666667   0.005332
4   0.182093   0.485294   0.229508   0.563637   0.666667   0.008391
5   0.138833   0.485294   0.229508   0.563637   0.666667   0.009912

   var7(t-1)  var8(t-1)   var1(t)
1   0.000000        0.0  0.148893
2   0.000000        0.0  0.159960
3   0.000000        0.0  0.182093
4   0.037037        0.0  0.138833
5   0.074074        0.0  0.109658
```

In [186]:
```
# split into train and test sets
values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(8760, 8) (8760,) (35039, 8) (35039,)
```

In [187]:
```
# plot the time series we are predicting
from pandas import Series
ts = Series(data=test_y) #index=pd.to_datetime(dates))
ts.plot()
```

Out[187]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7561d99e10>




```
In [188]: from sklearn.linear_model import Ridge
          clf = Ridge(alpha=0.001)
          clf.fit(train_X, train_y)
          #note that this is coming very close to just using the most recent measurement
          print(clf.coef_)
          yhat = clf.predict(test_X)
```

```
[ 0.92011267  0.01994498 -0.02873378 -0.01381463  0.00926529 -0.00564334
 -0.01413328 -0.02505873]
```

```
In [189]: #1 calculate RMSE
          mse = mean_squared_error(test_y, yhat)
          print('Test MSE: %.7f' % mse)
```

```
Test MSE: 0.0006973
```

```
In [190]: # If you want error in the original usits, you'd need to transform the data back
          # to calculate RMSE in original space
          # one coulde invert scaling for forecast,but it's just a constant scaling.
          #inv_yhat = scaler.inverse_transform(yhat)
```

```
In [191]: # 2 what if we don't know the most recent polution level?
          clf = Ridge(alpha=0.001)
```

```
clf.fit(train_X[:,1:], train_y)
#note that this is coming close to just using the most recent measurement
print(clf.coef_)
yhat = clf.predict(test_X[:,1:])
#print(test_y.shape)
#print(yhat.shape)
mse = mean_squared_error(test_y, yhat)
print('Test MSE: %.7f' % mse)
# we get an order of magnitude higher error
```

```
[ 0.28931925 -0.3976796  -0.21497105  0.03484664 -0.09229524 -0.13636717
 -0.2073935 ]
Test MSE: 0.0073079
```

In [192]:
```
#3 naive forecast: future is the same as the most recent value
#print(test_y)
# lag by 1 to get y_hat
#print('test_y mean, min, max:',ts.mean(), ts.min(), ts.max())
#print(ts.head())
#print(yhat.shape)
#print(test_y.shape)
yhat = ts.shift()
mse = mean_squared_error(test_y[1:], yhat[1:])
print('Test MSE: %.7f' % mse)
# the same as the original model
```

```
Test MSE: 0.0007139
```

In [193]:
```
### 5 AR(1)
ts = Series(data=train_y)
df = concat([ts.shift()], axis=1).dropna()  #create lags for a single series
clf = Ridge(alpha=0.001)
train_X = df.values
clf.fit(train_X,train_y[1:])
#print(train_y[2:].shape)
#print(train_X.shape)
ts_test = Series(data=test_y)
df_test = concat([ts_test.shift()], axis=1).dropna()  #create lags for a single series
test_X = df_test.values
#print(test_X.shape)
yhat = clf.predict(test_X)
print(clf.coef_)
mse = mean_squared_error(test_y[1:], yhat)
print('Test MSE: %.7f' % mse)
```

```
[0.9432982]
Test MSE: 0.0007011
```

In [194]:
```
### 5 generate more lags AR(3)
ts = Series(data=train_y)
df = concat([ts.shift(), ts.shift(2), ts.shift(3)], axis=1).dropna()  #create lags for
clf = Ridge(alpha=0.001)
train_X = df.values
clf.fit(train_X,train_y[3:])
#print(train_y[2:].shape)
#print(train_X.shape)
ts_test = Series(data=test_y)
df_test = concat([ts_test.shift(), ts_test.shift(2), ts_test.shift(3)], axis=1).dropna
test_X = df_test.values
#print(test_X.shape)
yhat = clf.predict(test_X)
print(clf.coef_)
mse = mean_squared_error(test_y[3:], yhat)
print('Test MSE: %.7f' % mse)
```

```
[ 0.85808223  0.13635994 -0.04825178]
Test MSE: 0.0007144
```

In [195]:
```
### 5 generate more lags AR(5)
ts = Series(data=train_y)
df = concat([ts.shift(), ts.shift(2), ts.shift(3),ts.shift(4),ts.shift(5)], axis=1).dr
clf = Ridge(alpha=0.001)
train_X = df.values
clf.fit(train_X,train_y[5:])
#print(train_y[2:].shape)
#print(train_X.shape)
ts_test = Series(data=test_y)
df_test = concat([ts_test.shift(), ts_test.shift(2), ts_test.shift(3),ts_test.shift(4)
test_X = df_test.values
#print(test_X.shape)
yhat = clf.predict(test_X)
print(clf.coef_)
mse = mean_squared_error(test_y[5:], yhat)
print('Test MSE: %.7f' % mse)
```

```
[ 0.85691845  0.14046852 -0.02578499 -0.03351098  0.00707205]
Test MSE: 0.0007150
```

In [196]:
```
# Now do the same for temperature
# load dataset
dataset = read_csv('pollution.csv', header=0, index_col=0)
cols = list(dataset)
cols[2], cols[0] = cols[0], cols[2]
cols
```

```python
dataset=(dataset.ix[:,cols])



print(dataset)
values = dataset.values

# integer encode direction
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])

# ensure all data is float
values = values.astype('float32')

reframed = series_to_supervised(values, 1, 1)
#  normalize features - this will make it easier to interpret regression coefficnts
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
print(reframed.head())
```

```
                     temp  dew  pollution   press wnd_dir  wnd_spd  snow  rain
date
2010-01-02 00:00:00  -4.0  -16      129.0  1020.0      SE     1.79     0     0
2010-01-02 01:00:00  -4.0  -15      148.0  1020.0      SE     2.68     0     0
2010-01-02 02:00:00  -5.0  -11      159.0  1021.0      SE     3.57     0     0
2010-01-02 03:00:00  -5.0   -7      181.0  1022.0      SE     5.36     1     0
2010-01-02 04:00:00  -5.0   -7      138.0  1022.0      SE     6.25     2     0
2010-01-02 05:00:00  -6.0   -7      109.0  1022.0      SE     7.14     3     0
2010-01-02 06:00:00  -6.0   -7      105.0  1023.0      SE     8.93     4     0
2010-01-02 07:00:00  -5.0   -7      124.0  1024.0      SE    10.72     0     0
2010-01-02 08:00:00  -6.0   -8      120.0  1024.0      SE    12.51     0     0
2010-01-02 09:00:00  -5.0   -7      132.0  1025.0      SE    14.30     0     0
2010-01-02 10:00:00  -5.0   -7      140.0  1026.0      SE    17.43     1     0
2010-01-02 11:00:00  -5.0   -8      152.0  1026.0      SE    20.56     0     0
2010-01-02 12:00:00  -5.0   -8      148.0  1026.0      SE    23.69     0     0
2010-01-02 13:00:00  -5.0   -8      164.0  1025.0      SE    27.71     0     0
2010-01-02 14:00:00  -5.0   -9      158.0  1025.0      SE    31.73     0     0
2010-01-02 15:00:00  -5.0   -9      154.0  1025.0      SE    35.75     0     0
2010-01-02 16:00:00  -5.0   -9      159.0  1026.0      SE    37.54     0     0
2010-01-02 17:00:00  -5.0   -8      164.0  1027.0      SE    39.33     0     0
2010-01-02 18:00:00  -5.0   -8      170.0  1027.0      SE    42.46     0     0
2010-01-02 19:00:00  -5.0   -8      149.0  1028.0      SE    44.25     0     0
2010-01-02 20:00:00  -5.0   -7      154.0  1028.0      SE    46.04     0     0
```

```
2010-01-02 21:00:00  -5.0   -7      164.0  1027.0      SE    49.17     1     0
2010-01-02 22:00:00  -6.0   -8      156.0  1028.0      SE    52.30     2     0
2010-01-02 23:00:00  -6.0   -8      126.0  1027.0      SE    55.43     3     0
2010-01-03 00:00:00  -6.0   -7       90.0  1027.0      SE    58.56     4     0
2010-01-03 01:00:00  -6.0   -8       63.0  1026.0      SE    61.69     5     0
2010-01-03 02:00:00  -7.0   -8       65.0  1026.0      SE    65.71     6     0
2010-01-03 03:00:00  -7.0   -8       55.0  1025.0      SE    68.84     7     0
2010-01-03 04:00:00  -7.0   -8       65.0  1024.0      SE    72.86     8     0
2010-01-03 05:00:00  -8.0   -9       83.0  1024.0      SE    76.88     9     0
...                   ...  ...        ...     ...     ...      ...   ...   ...
2014-12-30 18:00:00   2.0  -13       79.0  1020.0      NE     3.58     0     0
2014-12-30 19:00:00   6.0   -8       35.0  1021.0      NW     5.81     0     0
2014-12-30 20:00:00   5.0  -11       26.0  1022.0      NW    12.96     0     0
2014-12-30 21:00:00   4.0  -12       20.0  1023.0      NW    21.90     0     0
2014-12-30 22:00:00   2.0  -21        8.0  1025.0      NW    31.73     0     0
2014-12-30 23:00:00   0.0  -22       16.0  1026.0      NW    38.88     0     0
2014-12-31 00:00:00  -1.0  -19       10.0  1027.0      NW    51.84     0     0
2014-12-31 01:00:00  -1.0  -18       11.0  1028.0      NW    61.67     0     0
2014-12-31 02:00:00  -1.0  -17       20.0  1028.0      NW    70.61     0     0
2014-12-31 03:00:00  -1.0  -17        9.0  1029.0      NW    81.79     0     0
2014-12-31 04:00:00  -2.0  -19        8.0  1030.0      NW    94.75     0     0
2014-12-31 05:00:00  -3.0  -21        9.0  1030.0      NW   109.95     0     0
2014-12-31 06:00:00  -4.0  -23        8.0  1032.0      NW   130.07     0     0
2014-12-31 07:00:00  -5.0  -22        8.0  1034.0      NW   143.03     0     0
2014-12-31 08:00:00  -5.0  -22        8.0  1034.0      NW   150.18     0     0
2014-12-31 09:00:00  -3.0  -22        8.0  1034.0      NW   155.99     0     0
2014-12-31 10:00:00  -2.0  -22        7.0  1034.0      NW   163.14     0     0
2014-12-31 11:00:00  -2.0  -22       12.0  1034.0      NW   170.29     0     0
2014-12-31 12:00:00   0.0  -22       17.0  1033.0      NW   177.44     0     0
2014-12-31 13:00:00   0.0  -27       11.0  1032.0      NW   186.38     0     0
2014-12-31 14:00:00   1.0  -27        9.0  1032.0      NW   196.21     0     0
2014-12-31 15:00:00   1.0  -26       11.0  1032.0      NW   205.15     0     0
2014-12-31 16:00:00   0.0  -23        8.0  1032.0      NW   214.09     0     0
2014-12-31 17:00:00  -1.0  -22        9.0  1033.0      NW   221.24     0     0
2014-12-31 18:00:00  -2.0  -22       10.0  1033.0      NW   226.16     0     0
2014-12-31 19:00:00  -2.0  -23        8.0  1034.0      NW   231.97     0     0
2014-12-31 20:00:00  -3.0  -22       10.0  1034.0      NW   237.78     0     0
2014-12-31 21:00:00  -3.0  -22       10.0  1034.0      NW   242.70     0     0
2014-12-31 22:00:00  -4.0  -22        8.0  1034.0      NW   246.72     0     0
2014-12-31 23:00:00  -3.0  -21       12.0  1034.0      NW   249.85     0     0

[43800 rows x 8 columns]
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.245902   0.352941   0.129779   0.527273   0.666667   0.002290
2   0.245902   0.367647   0.148893   0.527273   0.666667   0.003811
3   0.229508   0.426471   0.159960   0.545454   0.666667   0.005332
4   0.229508   0.485294   0.182093   0.563637   0.666667   0.008391
5   0.229508   0.485294   0.138833   0.563637   0.666667   0.009912
```

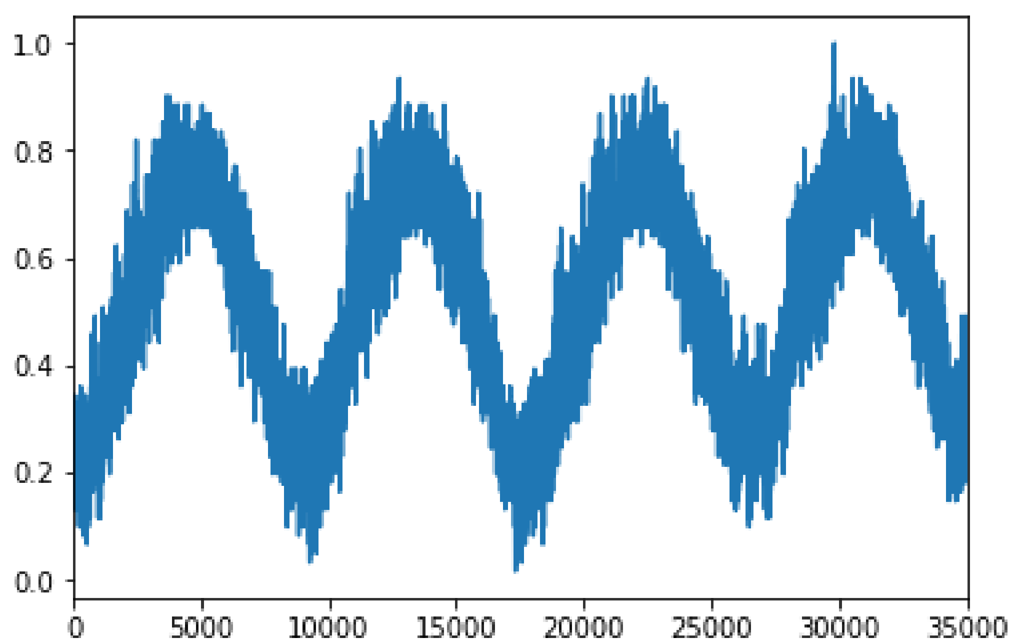
```
   var7(t-1)  var8(t-1)   var1(t)
1   0.000000        0.0  0.245902
2   0.000000        0.0  0.229508
3   0.000000        0.0  0.229508
4   0.037037        0.0  0.229508
5   0.074074        0.0  0.213115
```

In [197]:
```
# split into train and test sets
values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(8760, 8) (8760,) (35039, 8) (35039,)
```

In [198]:
```
# plot the time series we are predicting
from pandas import Series
ts = Series(data=test_y) #index=pd.to_datetime(dates))
ts.plot()
```

Out[198]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7562287e48>



In [199]:
```
from sklearn.linear_model import Ridge
clf = Ridge(alpha=0.001)
clf.fit(train_X, train_y)
#note that this is coming very close to just using the most recent measurement
print(clf.coef_)
yhat = clf.predict(test_X)
```

```
[ 9.6679091e-01  3.1889014e-02 -9.1553042e-03  5.0978060e-04
 -4.9584783e-03 -1.1957677e-02 -1.0098002e-02 -1.9364167e-02]
```

In [200]:
```
#1 calculate RMSE
mse = mean_squared_error(test_y, yhat)
print('Test MSE: %.7f' % mse)
```

```
Test MSE: 0.0005973
```

In [201]:
```
# 2 what if we don't know the most recent temperature level?
clf = Ridge(alpha=0.001)
clf.fit(train_X[:,1:], train_y)
#note that this is coming close to just using the most recent measurement
print(clf.coef_)
yhat = clf.predict(test_X[:,1:])
#print(test_y.shape)
#print(yhat.shape)
mse = mean_squared_error(test_y, yhat)
print('Test MSE: %.7f' % mse)
# we get an order of magnitude higher error
```

```
[ 0.67140615 -0.4604669  -0.37646142  0.00159733  0.06496359 -0.31914526
 -0.26446584]
Test MSE: 0.0086624
```

In [202]:
```
yhat = ts.shift()
mse = mean_squared_error(test_y[1:], yhat[1:])
print('Test MSE: %.7f' % mse)
# the same as the original model
```

```
Test MSE: 0.0006171
```

In [203]:
```
### 5 AR(1)
ts = Series(data=train_y)
df = concat([ts.shift()], axis=1).dropna()  #create lags for a single series
```

```
clf = Ridge(alpha=0.001)
train_X = df.values
clf.fit(train_X,train_y[1:])
#print(train_y[2:].shape)
#print(train_X.shape)
ts_test = Series(data=test_y)
df_test = concat([ts_test.shift()], axis=1).dropna()  #create lags for a single series
test_X = df_test.values
#print(test_X.shape)
yhat = clf.predict(test_X)
print(clf.coef_)
mse = mean_squared_error(test_y[1:], yhat)
print('Test MSE: %.7f' % mse)
```

```
[0.9937631]
Test MSE: 0.0006148
```

In [204]:
```
### 5 generate more lags AR(5)
ts = Series(data=train_y)
df = concat([ts.shift(), ts.shift(2), ts.shift(3),ts.shift(4),ts.shift(5)], axis=1).dr
clf = Ridge(alpha=0.001)
train_X = df.values
clf.fit(train_X,train_y[5:])
#print(train_y[2:].shape)
#print(train_X.shape)
ts_test = Series(data=test_y)
df_test = concat([ts_test.shift(), ts_test.shift(2), ts_test.shift(3),ts_test.shift(4)
test_X = df_test.values
#print(test_X.shape)
yhat = clf.predict(test_X)
print(clf.coef_)
mse = mean_squared_error(test_y[5:], yhat)
print('Test MSE: %.7f' % mse)
```

```
[ 1.2271309   0.00189858 -0.15569393 -0.11352163  0.02916911]
Test MSE: 0.0005101
```

In [205]:
```
### 5 generate more lags AR(3)
ts = Series(data=train_y)
df = concat([ts.shift(), ts.shift(2), ts.shift(3)], axis=1).dropna()  #create lags for
clf = Ridge(alpha=0.001)
train_X = df.values
clf.fit(train_X,train_y[3:])
#print(train_y[2:].shape)
#print(train_X.shape)
ts_test = Series(data=test_y)
df_test = concat([ts_test.shift(), ts_test.shift(2), ts_test.shift(3)], axis=1).dropna
```

```python
test_X = df_test.values
#print(test_X.shape)
yhat = clf.predict(test_X)
print(clf.coef_)
mse = mean_squared_error(test_y[3:], yhat)
print('Test MSE: %.7f' % mse)
```

```
[ 1.2446258  -0.00264128 -0.25250247]
Test MSE: 0.0005134
```