QBUS6840: Tutorial 5 – Linear Regression

Objectives

- Use `sklearn` library to train linear regression models
- Use linear regression and decomposition results for forecasting
- Using linear regression for modeling trend and seasonal simultaneously
- Analyze and evaluate suitability of linear regression models (optional)

In this tutorial, we will apply the basic linear regression to predict the beer sales. More specifically, we will use a linear regression model to generate the trend estimation and then combine this trend with seasonal index as forecasting results.

1. Linear regression model in `sklearn` library

In this task, you will learn how to use the python machine learning package sklearn to do linear regression for a time series.

Step 1: Load the Data and Visual Inspection

Create a new Python script called "`tutorial_05.py`" and download the "`beer.txt`" file from the QBUS6840 Canvas site.

Begin our script by importing necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

Then read the data file into the `beer_df` dataframe variable

```
beer_df = pd.read_csv('beer.txt')
```

Confirm that the data was loaded successfully by plotting the beer sales data.

```
X = np.linspace(1, len(beer_df), len(beer_df))
y = beer_df['Sales']

plt.figure()
plt.plot(X, y)
plt.title("Beer Sales")
```

Question: What is the datatype and size of `X` and `y`?

Step 2: Regress beer sales against time

To regress beer sales we can use the `scikit` learn `LinearRegression` object. It takes care of fitting the model and calculating predictions for us.

First we need to prepare variables for linear regression

```
X = np.reshape(X, (len(beer_df), 1))
y = y.values.reshape(len(beer_df),1)
```

We must convert the training data into to 2D shape (matrices). This is the fundamental data structure requirement of `sklearn` package.

In the above case, variable X's data type is a 1D array of float64. You may think of it as a series of values. However all the sklearn modeling methods take training data as 2D array. We use `X = np.reshape(X, (len(beer_df), 1))` clearly convert X into a one column shape in size `(len(beer_df), 1)`. Same to `y`.

Create a new `LinearRegression()` object and assign this object to a new variable:

```
lm = LinearRegression()
```

Generally, once you create a new variable, you can always find its name in the Variable Explorer. However, `LinearRegression()` object can't be visualized as specific numbers or strings. Therefore, you can't see `lm` variable in the Variable Explorer. In this case, if you want to know the details of this `lm` variable, you can type the variable name "`lm`" in the IPython Console and check the details of the object settings. Below is an example:

```
In[*]  : lm
Out[*] : LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=1, normalize=False)
```

The above message tells us that we have defined a linear regression model which will fit the intercept and make a copy of the data when fitting the model.

Train the model by using the `fit()` function

```
lm.fit(X, y)
```

`sklearn` hides the fitted parameters inside the model variable lm (in this case). We can extract these information by accessing models' relevant attributes. The following example shows how to print out parameters from our fitted model.

```
# The coefficients
print("Coefficients: {0}".format(lm.coef_))
# The intercept
print("Intercept: {0}".format(lm.intercept_))

print("Total model: y = {0} + {1} X".format(lm.intercept_,
lm.coef_[0]))
```

Calculate R-squared by feeding back the original data points into the model.

```
print("Variance score (R^2): {0:.2f}".format(lm.score(X, y)))
```

Question: what is the meaning of `"{0:.2f}"`?

Plot the predictions/trend from the model. You can use `predict()` function for forecasting the test data.

```
trend = lm.predict(X)

plt.figure()
plt.plot(X, y, label = "Beer Sales")
plt.plot(trend, label="Trend")
plt.legend()
plt.title("Beer Sales Trend from Linear Regression Model")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show(block=False)
```

Question: What is the meaning of `trend = lm.predict(X)`?

Calculate SSE or use the `._residues` attribute

```
sse1 = np.sum( np.power(y - trend,2), axis=0)
sse2 = lm._residues
```

For more details about `LinearRegression()` object in `sklearn` library, please refer to the following link:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Step 3: Predict the test data

We can use the raw data to predict our sales trend. This will give you a very rough and approximate prediction.

```
forecast = lm.predict(np.reshape(np.arange(72), (72,1)))

plt.figure()
plt.plot(X, y, label="Beer Sales")
plt.plot(trend, label="Trend")
plt.plot(forecast, linestyle='--', label="Forecast")
plt.legend()
plt.title("Beer Sales Forecast from Trend Only Linear Regression
Model")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show(block=False)
```

2. Using linear regression for forecasting

In the previous task, we have trained a linear regression model. One thing you should notice is that this linear regression could only be used to generate/forecast the trend component $\widehat{T}_t$ rather than the entire time-series sequence $\widehat{y}_t$.

Generally, in order to obtain a more accurate result, you need to decompose your original data into trend(-cycle) and seasonal index (as shown in tutorial 3 and 4), and then use forecasted trend component combined with seasonal index to do the forecasting.

Step 1: Initially estimate the trend-cycle and obtain de-trend series

We first using a 12 x 2 moving average to extract the initial trend. And then use this initial trend to calculate the seasonal component.

```
T = beer_df.rolling(12, center = True).mean().rolling(2, center =
True).mean().shift(-1)

S_additive = beer_df['Sales'] - T['Sales']
```

Exam you `S_additive` variable, How many missing value in the beginning? What is the size of this variable?

Step 2: Extract the seasonal index and obtain the seasonal adjusted series

Since we have some missing value, we need to fill this nan element with 0. In addition, we need to concatenate several 0 in the end of `S_additive` variable so that the length will comes to 60 (which is 5 years).

```
safe_S = np.nan_to_num(S_additive)
monthly_S = np.reshape(np.concatenate( (safe_S, [0,0,0,0]), axis =
0), (5,12))
```

Then we need to calculate the seasonal index. You can refer to week04 tutorial material for the detailed explanation.

```
monthly_avg = np.mean(monthly_S[1:4,], axis=0)
mean_allmonth = monthly_avg.mean()
monthly_avg_normed = monthly_avg - mean_allmonth

tiled_avg = np.tile(monthly_avg_normed, 6)
seasonal_adjusted = beer_df['Sales'] - tiled_avg[:len(beer_df)]
```

Step 3: Re-estimate/forecast the trend-cycle using the linear regression

Using the seasonal adjusted data, we can model the trend component using the linear regression model.

```
y_trend = seasonal_adjusted.values.reshape(-1,1)
lm_trend = LinearRegression().fit(X, y_trend)
```

Step 4: Forecast

First, we forecast 16 values for the trend and then we combine the trend and seasonal index together as the forecasting results.

```
linear_trend = lm_trend.predict(np.reshape(np.arange(72), (72,1)))
linear_seasonal_forecast = linear_trend + tiled_avg.reshape(-1,1)

# Plot forecast results
plt.figure()
plt.plot(X, y, label="Original Data")
plt.plot(linear_trend, label="Linear Model trend")
plt.plot(linear_seasonal_forecast, label="Linear+Seasonal
Forecast")
plt.title("Beer Sales Forecast from Trend+Seasonal Linear
Regression Model")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show(block=False)
```

Congratulations, you have learned our first algorithm for forecasting the time-series data.

3. Using linear regression for modeling trend and seasonal simultaneously

In Lecture 4, we introduce a method of using dummy variables to fit some patterns in time series. For seasonal patterns, we can use the so-called dummy predictors.

In the session 1, we build a linear regression model for the trend-cycle component where the predictor was time t (from 1 to 56) and the model is

$$T_t = b_0 + b_1 t$$

In the following example, we will build a regression model directly which is defined as

$$Y_t = b_0 + b_1 t + b_2 d_{2,t} + b_3 d_{3,t} + b_4 d_{4,t} + b_5 d_{5,t} + b_6 d_{6,t} + b_7 d_{7,t} + b_8 d_{8,t} + b_9 d_{9,t} + b_{10} d_{10,t} + b_{11} d_{11,t} + b_{12} d_{12,t}$$

where $d_{j,t}$ are dummy predictors corresponding to a particular month (if we are working on monthly data) and b_j ($j \geq 2$) are something similar to seasonal index in the decomposition model.

The above model is a multiple linear regression model. To fit the model, the key is to organize the predictor values.

The values of dummy predictors $d_{j,t}$ are special values determined by the time t. In general, we shall have:

1. When t (such as t=1, 13, 27, etc) is a January, we assume
$$d_{2,t} = d_{3,t} = \cdots = d_{12,t} = 0$$
2. When t is a February, we assume:
$$d_{2,t} = 1 \ \ and\ others\ d = 0$$
3. When t is a March, we assume
$$d_{3,t} = 1 \ \ and\ others\ d = 0$$
4. ..
5. ..
6. When t is a December, we assume
$$d_{12,t} = 1 \ \ and\ others\ d = 0$$

For the regression, input data will look like

t	$d_{2,t}$	$d_{3,t}$	$d_{4,t}$	$d_{5,t}$	$d_{6,t}$	$d_{7,t}$	$d_{7,t}$	$d_{8,t}$	$d_{9,t}$	$d_{10,t}$	$d_{11,t}$	$d_{12,t}$
1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0
		…					….					
56	0	0	0	0	0	0	0	1	0	0	0	0

Each row corresponds to a time series value Y_t. Then do regression fitting. All these can be easily done in sklearn.

Step 1: Load and Prepare Initial Data

In here you may refer to the first step in Task 1 to load the dataset.
Check variables X and y in the variable explorer. Check their data types and shapes.

Step 2: Prepare Dummy Predictors

```
# we start from January
seasons = []
for i in range(y.size):
    if i % 12 == 0:
        seasons = np.append(seasons, 'Jan')
    if i % 12 == 1:
        seasons = np.append(seasons, 'Feb')
    if i % 12 == 2:
        seasons = np.append(seasons, 'Mar')
    if i % 12 == 3:
```

```
        seasons = np.append(seasons, 'Apr')
    if i % 12 == 4:
        seasons = np.append(seasons, 'May')
    if i % 12 == 5:
        seasons = np.append(seasons, 'Jun')
    if i % 12 == 6:
        seasons = np.append(seasons, 'Jul')
    if i % 12 == 7:
        seasons = np.append(seasons, 'Aug')
    if i % 12 == 8:
        seasons = np.append(seasons, 'Sep')
    if i % 12 == 9:
        seasons = np.append(seasons, 'Oct')
    if i % 12 == 10:
        seasons = np.append(seasons, 'Nov')
    if i % 12 == 11:
        seasons = np.append(seasons, 'Dec')
```

Check variables `seasons` in the variable explorer. Check its data types and shapes. What are the values over there?

You note that the variable seasons contain string values such as 'Jan', …, 'Dec'. They are so-called categorical values.

Now copy the following code into your program

```
dummies = pd.get_dummies(seasons, drop_first=True)
```

Check variables `dummies` in the variable explorer. Check its data types and shapes. What are the values over there?

You may note that the fourth row (corresponding to April) contains all 0. This is because Python orders month names in alphabetic order. Write down all the value in the first row which corresponding to Jan. What are values?

It really does not matter which month's dummy predictor values are all 0. The important thing is that each month has its own dummy predictor values.

Step 3: Combining dummy predictors with our time predictor t

```
#Please note dummier is a pandas DataFrame, we shall take values out
by
dummies = dummies.values

# make sure the size of X and dummies match
# If you are using numpy version below than 1.10, you need to
uncomment the following statement
# X = X[:, np.newaxis]

# Now we add these dummy features into feature, stacking along the
column
Xnew = np.hstack((X,dummies))
```

Check variables Xnew in the variable explorer. Check its data types and shapes. What are the values over there?

Step 4: Doing regression

```
# Create linear regression object (model)
regr = LinearRegression()

# Train the model using the training sets
regr.fit(Xnew, y)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The intercept
print('Intercept: \n', regr.intercept_)
```

Step 5: Showing Results

```
Ypred = regr.predict(Xnew)

plt.plot(y, label='Observed')
plt.plot(Ypred, '-r', label='Predicted')
plt.legend()
```

Regression is powerful. If we have found patterns and define appropriate predictors which describe those patterns, then we can do a good predictive model.

Think about other patterns identified in the lecture and the way to define predictors in python.

4. Evaluate a linear regression model (Optional)

With time series data it is highly likely that the value of a variable observed in the current time period will be influenced by its value in the previous period, or even the period before that, and so on.

When fitting a regression model to time series data, it is very common to find autocorrelation in the residuals, which violates the assumption of no autocorrelation in the errors.

Some information left over should be utilized in order to obtain better forecasts. Therefore, we need to apply the autocorrelation function (ACF) [Note: We will talk about ACF later in lecture. Here we simply learn how to use them.] on the residual. Suppose, e_t denote the residual of a time series at time t. The ACF of the series gives correlations between e_t and e_{t-i} for $i = 1,2,3$, etc. Theoretically, the autocorrelation between e_t and e_{t-i} equals:

$$\frac{\text{Covariance }(e_t, e_{t-i})}{\text{Std}(e_t)\text{Std}(e_{t-i})} = \frac{\text{Covariance }(e_t, e_{t-i})}{\text{Variance }(e_t)}$$

Note that the denominator in the second formula occurs because the standard deviation of a stationary series is always the same .

Let's plot and inspect the ACF. This will show us the autocorrelation. In here we set the interval $i = 1,2, \dots, 15$

```
beer_df['residuals'] = y - trend

acf_vals = [beer_df['residuals'].autocorr(i) for i in
range(1,15) ]

plt.figure()
plt.bar(np.arange(1,15), acf_vals)
plt.title("ACF of Beer Sales")
plt.xlabel("Month delay/lag")
plt.ylabel("Correlation Score")
plt.show(block=False)
```

Question:

(1) What phenomenon you could observe?
(2) Since our data appears in yearly repeatable pattern, what will happen if we extend the lag interval from 15 to 25?

We also need to inspect the residual plot (X vs residual). A random distribution means the model has captured the trend accurately.

```
plt.figure()
plt.title("Residual plot")
plt.scatter(X, trend - y)
plt.xlabel("Month")
plt.ylabel("Residuals")
plt.show(block=False)
```