QBUS6840: Tutorial 5 – Linear Regression

Objectives

- Use sklearn library to train linear regression models
- Analyze and evaluate suitability of linear regression models
- Use linear regression and decomposition result to forecast the time-series data

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

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 blackboard.

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

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 visualize 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)
```

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

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

Print parameters from our 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)
```

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: Evaluate the model

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) 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 the same at all times.

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

```
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)
```

Step 4: 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(1,72),
(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 learning 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 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, and then use forecasted trend component combined with seasonal index to do the forecasting.

Step 1: Predict the Sales Trend using trend and seasonal components

A more accurate method is to decompose into trend and seasonal

components. Then we can add the seasonal components back in for a more accurate model.

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()

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

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)
```

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

Basically, in here you need to analyse your residual. If the residual is not stationary, means your extracted seasonal index and trend-cycle component is not accurate enough. Since we have analysed the residual in the previous task, in here we just skip these actions.

We use trained linear regression model to re-estimate the trend. In here, our `X` ranges from 1 to 72, which is 6 years.

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

Step 4: Forecast

Finally, we combine the trend and seasonal index together as the forecasting results.

```
linear_seasonal_forecast = linear_trend + tiled_avg

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.