Air Passenger Forecasting using ARMA

TAF Individual Project



산업공학과

학번: 2018015414

이름: 박진혁

1. Data Loading and Preprocessing

The dataset is loaded from AirPassengers.csv which contains monthly data on the number of air passengers from 1949 to 1960.

The timestamp column is converted to datetime format and set as the index.

1. # Load the dataset

2. file\_path = 'AirPassengers.csv'

3. data = pd.read\_csv(file\_path, header=None)

4.

5. # Rename columns and convert 'timestamp' to datetime

6. data.columns = ['timestamp', 'Passengers']

7. data['timestamp'] = pd.to\_datetime(data['timestamp'], format='%Y-%m')

8. data.set\_index('timestamp', inplace=True)

9.

The number of passenger looks like there is some trend(increasing) and doesn't have structural break Thus it is reasonable to visually conclude that the number of passenger is not stationary(because of trend). Also it seems like there is volatility increasing depends on time(at diff data, second diff data too).

So it is reasonable to have some transform, i choose log transform.

텍스트, 폰트, 친필, 번호이(가) 표시된 사진

자동 생성된 설명

Fig 1: plot of the number of passenger, first difference, second difference

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

Fig 2: Plot of log transformed data and differences for log transformed data

It is reasonable to say first difference and second difference seems more stable than data before transform (visually conclude) then we will use ADF and KPSS test for stationary check first.

2. Stationary test (ADF, KPSS)

Before the stationary check, splits data to in-sample and out-sample data.

1. train = data[:'1958-12-01']

2. test = data['1959-01-01':]

Then all the test is on train, which is in-sample data

The code for ADF and KPSS are follows. Using statsmodels libraries.

1. from statsmodels.tsa.stattools import acf

2. from statsmodels.tsa.stattools import adfuller

3. from statsmodels.tsa.stattools import kpss

4. def adf\_test(dataframe):

5.   result = adfuller(dataframe)

6.   print(f'Statistics: {result[0]}')

7.   print(f'p-value: {result[1]}')

8.   print(f'Critical values: {result[4]}')

9. def kpss\_test(dataframe):

10.   result = kpss(dataframe)

11.   print(f'Statistics: {result[0]}')

12.   print(f'bounded p-value: {result[1]}')

13.   print(f'Critical values: {result[3]}')

The ADF test results are follows.

|  |  |  |
| --- | --- | --- |
| Data type | Statistics | p-value |
| Origin data | -0.7734607708969381 | 0.8267937485032447 |
| First differenced data | -2.1641431278047762 | 0.21951577637150677 |
| Second differenced data | -13.947363642065772 | 4.7704196840302815e-26 |
| First differenced  log-transformed data | 0.15822839568723085 | 0.15822839568723085 |
| Second differenced  log-transformed data | -7.633351316530781 | 1.981881694590538e-11 |

And KPSS test results are followed.

|  |  |  |
| --- | --- | --- |
| Data type | Statistics | p-value |
| Origin data | 1.7058124992217791 | 0.01 |
| First differenced data | 0.019023125650029386 | 0.1 |
| Second differenced data | 0.08275298336937394 | 0.1 |
| First differenced  log-transformed data | 0.029471641004343525 | 0.1 |
| Second differenced  log-transformed data | 0.409180392862506 | 0.1 |

The ADF test results show that the p-value is very high (0.82), indicating that the series is not stationary. We need to difference the data to achieve stationarity. If you seed the p-value you can find origin and first differences (also log transformed data) do not reject null. We can find second derivative (also log transformed data) reject the null, which means they are stationary.

The KPSS test suggests that the first differenced series (also log transformed) is stationary, but the ADF test is not as conclusive. which means trend stationary.

The KPSS test& ADF test suggests that the second differenced series (also log transformed) is stationary. we need to check the ACF and PACF plots for further insights.

3. Candidate model (ACF, PACF)

Checking the ACF, PACF give us the information for appropriate p, q for ARMA model(also for ARIMA model). We can easily plot ACF, PACF by using stats model libraries.

1. import matplotlib.pyplot as plt

2. from statsmodels.graphics.tsaplots import plot\_acf

3. from statsmodels.graphics.tsaplots import plot\_pacf

4. f, axes = plt.subplots(nrows=2, ncols=1, figsize=(8, 2\*4))

5. plot\_acf(train\_diff1, lags=24, ax=axes[0], title='Autocorrelations', color='black',vlines\_kwargs={'colors':'black','linewidth':5}, alpha=None)

6. axes[0].hlines(xmin=0,xmax=24,y=2\*np.sqrt(1/len(train\_diff1)),label='two standard deviations from zero',color='black',linewidth=1)

7. axes[0].hlines(xmin=0,xmax=24,y=-2\*np.sqrt(1/len(train\_diff1)),color='black',linewidth=1)

8. plot\_pacf(train\_diff1, lags=24, ax=axes[1], method='ols', title='PACF', color='gray',vlines\_kwargs={'colors':'gray','linewidth':5}, alpha=None)

9. axes[1].hlines(xmin=0,xmax=24,y=2\*np.sqrt(1/len(train\_diff1)),label='two standard deviations from zero',color='black',linewidth=1)

10. axes[1].hlines(xmin=0,xmax=24,y=-2\*np.sqrt(1/len(train\_diff1)),color='black',linewidth=1)

11. axes[1].legend()

12. plt.tight\_layout()

13. plt.show()

3-1 d=1 (ACF, PACF)

텍스트, 도표, 라인, 평행이(가) 표시된 사진

자동 생성된 설명

1. The ACF and PACF show us a cyclical(seasonal?) pattern(12), which seems like they has seasonality

2. The ACF does not cut to zero so that we can rule out a pure MA(q) process.

3. The ACF plot shows significant spikes at lag 1, 12, and 24.

There is a gradual decay in the autocorrelations. These patterns suggest the presence of seasonal components, given the regular spikes at higher lags.

4. PACF analysis

The PACF plot shows significant spikes at lag 1, 8, 10, 12. The initial spike at lag 1 indicates that an AR(1) component may be necessary. The PACF does not cut off quickly, which could indicate a mixture of autoregressive components and/or seasonal autoregressive components.

So for the given information suggested model are

1. since the series has been difference, d=1

2. significant spike at lag 1 in PACF suggests AR(1). p=1

3. significant spike at lag 1 in ACF suggests MA(1). so q=1 or 0, [1,4]

3-2 d=2 (ACF, PACF)

1. The ACF and PACF show us a cyclical pattern(12), which seems like they has seasonality

2. The ACF does not cut to zero so that we can rule out a pure MA(q) process.

3. ACF

The ACF plot shows significant spikes at lag 1 and 12, with the spikes decaying slowly over time. This suggests the presence of a seasonal component around lag 12. The spike at lag 24 is also notable and might suggest another seasonal cycle, but the primary focus should remain on the first noticeable seasonality at lag 12.

4. PACF

The PACF plot shows a significant spike at lag 1 and 2, indicating a possible AR(1) or AR(2) component. The significant spikes at lags 10,11,12 seasonal autoregressive components at these lags.

텍스트, 도표, 라인, 평행이(가) 표시된 사진

자동 생성된 설명

So for the given information suggested model are

1. since the series has been difference, d=2

2. significant spike at lag 1 in PACF suggests AR(1), AR(2). p=1, 2

3. significant spike at lag 1,2 in ACF suggests MA(1), MA(2). q= 0, 1, 2(I just test q=0 for sure)

3-3 d=1, log-transformed (ACF, PACF)

텍스트, 도표, 라인, 평행이(가) 표시된 사진

자동 생성된 설명

1. The ACF and PACF show us a cyclical pattern (12), which seems like they have seasonality

2. The ACF does not cut to zero so that we can rule out a pure MA(q) process.

3. ACF: Significant spikes at lags 1, 12, and 24.

4. PACF: Significant spikes at lag 1 and 4 also at 10~12

So for the given information suggested model are

1. since the series has been difference, d=1

2. significant spike at lag 1 in PACF suggests AR(1), AR([1,4])

3. significant spike at lag 1 ACF suggests MA(1), MA([1,4])

3-3 d=2, log-transformed (ACF, PACF)

텍스트, 도표, 라인, 평행이(가) 표시된 사진

자동 생성된 설명1. The ACF and PACF show us a cyclical pattern(12), which seems like they has seasonality

2. The ACF does not cut to zero so that we can rule out a pure MA(q) process.

3. ACF: Significant spike at 1,4 / 12

4. PACF: Significant spike at 1,2,4 and 10,11,12

So for the given information suggested model are </br>

1. since the series has been difference, d=2

2. significant spike at lag 1,2 in PACF suggests AR(1), AR(2)

3. significant spike at lag 1, 4 ACF suggests MA(1), MA([1,4])

4. Model Fitting and Scoring

First let’s fitted the data for original data. I test 9 models for original data.

ARIMA(1,1,0), ARIMA(1,1,1), ARIMA(1,1,[1,4]), ARIMA (1,2,0), ARIMA (1,2,1), ARIMA (1,2,2), ARIMA (2,2,0), ARIMA (2,2,1), ARIMA(2,2,2)]

1. lag\_list = [(1,1,0), (1,1,1), (1,1,[1,4]), (1,2,0), (1,2,1), (1,2,2), (2,2,0), (2,2,1), (2,2,2)]

2.

3. summary\_table = dict()

4.

5. idx=0

6. num\_of\_obs = len(train[4:])

7.

8. for lag in lag\_list:

9.     # SSE i.e SSR, AIC, SBC, Ljung–Box Q-statistics of the residual autocorrelations for lag:={4, 8, 12}.

10.     temp\_perf\_dict = {key: key for key in ['SSE','AIC','SBC','Q(4)','Q(8)','Q(12)']}

11.

12.     # Get the maximum value(s) from each element in the set

13.     max\_values = [get\_max\_value(elem) for elem in lag]

14.     max\_element = max(max\_values)

15.     # We use spread[4-max\_element:] to estimate each equation over the 1961Q4-2012Q4.

16.     res = ARIMA(endog = train[4-max\_element:], order=lag).fit() # Use Durbin–Levinson algorithm. You can also use other estimation method.

17.     temp\_perf\_dict['SSE'] = round(res.sse,2)

18.

19.     # Note that since we assume the model errors are IID according to a normal distribution, the BIC and AIC formula is bit different from a general form.

20.     temp\_perf\_dict['AIC'] = round(num\_of\_obs\*np.log(res.sse) + 2\*len(res.params),2)

21.     temp\_perf\_dict['SBC'] = round(num\_of\_obs\*np.log(res.sse) + len(res.params)\*np.log(num\_of\_obs),2)

22.

23.     # Ljung-box Q-statistics for lag 4,8,12

24.     q\_statistics = res.test\_serial\_correlation(method='ljungbox',lags=12)[0]

25.

26.     temp\_perf\_dict['Q(4)'] = {'q\_stats' : round(q\_statistics[0][3],2), 'p\_val': round(q\_statistics[1][3],2)}

27.     temp\_perf\_dict['Q(8)'] = {'q\_stats' : round(q\_statistics[0][7],2), 'p\_val': round(q\_statistics[1][7],2)}

28.     temp\_perf\_dict['Q(12)'] = {'q\_stats' : round(q\_statistics[0][11],2), 'p\_val': round(q\_statistics[1][11],2)}

29.

30.     for param\_name, param in zip(res.params.index, res.params):

31.         temp\_perf\_dict[param\_name] = {'coef':round(param,2), 't\_stats':round(res.tvalues[param\_name],2)}

32.

33.     hashable\_order = tuple([tuple(order) if isinstance(order,list) == True else order for order in res.specification['order']]) # make res.specification['order'] hashable.

34.     summary\_table[hashable\_order] = temp\_perf\_dict

35.

For the result of 9 models is below.

스크린샷, 텍스트, 흑백, 블랙이(가) 표시된 사진

자동 생성된 설명

Note that Ljung–Box Q-statistics can serve as a check to see if the residuals from an estimated ARMA(p, q) and ARIMA(p,d,q) model behave as a white-noise process. (The t-statistics at 5% significance level is 1.93.)

From result of Sum of Squared Errors (SSE), Akaike Information Criterion (AIC), Schwarz Bayesian Criterion (SBC), I choose

라인, 스크린샷, 도표, 텍스트이(가) 표시된 사진

자동 생성된 설명

Second the code for fitting Log-transformed data is as blow.

텍스트, 스크린샷, 흑백, 블랙이(가) 표시된 사진

자동 생성된 설명

스크린샷, 텍스트, 라인, 도표이(가) 표시된 사진

자동 생성된 설명