02-Autoregression-AR

October 19, 2022

---

1 AR(p)

2 Autoregressive Model

In a moving average model as we saw with Holt-Winters, we forecast the variable of interest using a linear combination of predictors. In our example we forecasted numbers of airline passengers in thousands based on a set of level, trend and seasonal predictors.

In an autoregression model, we forecast using a linear combination of past values of the variable. The term autoregression describes a regression of the variable against itself. An autoregression is run against a set of lagged values of order p.

2.0.1 $y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$

where c is a constant, ϕ_1 and ϕ_2 are lag coefficients up to order p, and ε_t is white noise.

For example, an AR(1) model would follow the formula

$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$

whereas an AR(2) model would follow the formula

$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \varepsilon_t$

and so on.

Note that the lag coeffients are usually less than one, as we usually restrict autoregressive models to stationary data. Specifically, for an AR(1) model: $-1 < \phi_1 < 1$ and for an AR(2) model: $-1 < \phi_2 < 1,\ \phi_1 + \phi_2 < 1,\ \phi_2 - \phi_1 < 1$

Models AR(3) and higher become mathematically very complex. Fortunately statsmodels does all the heavy lifting for us.

Related Functions:

ar_model.AR(endog[, dates, freq, missing]) Autoregressive AR(p) model
ar_model.ARResults(model, params[, …]) Class to hold results from fitting an AR model

For Further Reading:

Forecasting: Principles and Practice Autoregressive models Wikipedia Autoregressive model

2.1 Perform standard imports and load datasets

For this exercise we'll look at monthly U.S. population estimates in thousands from January 2011 to December 2018 (96 records, 8 years of data). Population includes resident population plus armed forces overseas. The monthly estimate is the average of estimates for the first of the month and the first of the following month. Source: https://fred.stlouisfed.org/series/POPTHM

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline

# Load specific forecasting tools
from statsmodels.tsa.ar_model import AR,ARResults

# Load the U.S. Population dataset
df = pd.read_csv('../Data/uspopulation.csv',index_col='DATE',parse_dates=True)
df.index.freq = 'MS'
```

[2]:
```
df.head()
```

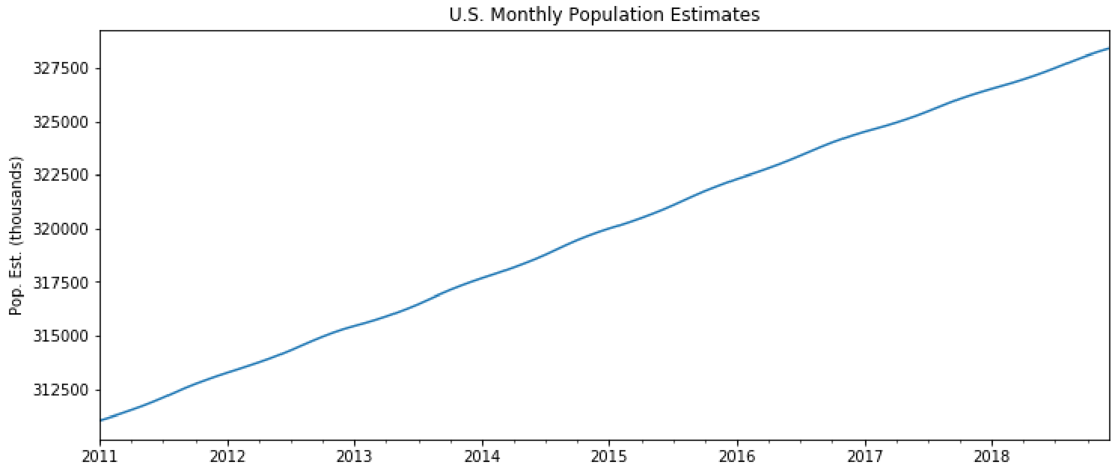
[2]:

	PopEst
DATE	
2011-01-01	311037
2011-02-01	311189
2011-03-01	311351
2011-04-01	311522
2011-05-01	311699

2.2 Plot the source data

[3]:
```
title='U.S. Monthly Population Estimates'
ylabel='Pop. Est. (thousands)'
xlabel='' # we don't really need a label here

ax = df['PopEst'].plot(figsize=(12,5),title=title);
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



2.3 Split the data into train/test sets

The goal in this section is to: * Split known data into a training set of records on which to fit the model * Use the remaining records for testing, to evaluate the model * Fit the model again on the full set of records * Predict a future set of values using the model

As a general rule you should set the length of your test set equal to your intended forecast size. That is, for a monthly dataset you might want to forecast out one more year. Therefore your test set should be one year long.

NOTE: For many training and testing applications we would use the train_test_split() function available from Python's scikit-learn library. This won't work here as train_test_split() takes random samples of data from the population.

[4]:
```
len(df)
```

[4]: 96

[5]:
```
# Set one year for testing
train = df.iloc[:84]
test = df.iloc[84:]
```

2.4 Fit an AR(1) Model

[6]:
```
# Ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")
```

[7]:
```
model = AR(train['PopEst'])
AR1fit = model.fit(maxlag=1,method='mle')
print(f'Lag: {AR1fit.k_ar}')
print(f'Coefficients:\n{AR1fit.params}')
```

```
Lag: 1
Coefficients:
const        132.578420
L1.PopEst      0.999583
dtype: float64
```

NOTE: There's a slight difference between the object returned by the Holt-Winters Exponential Smoothing .fit() method and that returned by AR. The Holt-Winters object uses .forecast() for predicted values, while AR uses .predict().

[8]:
```
# This is the general format for obtaining predictions
start=len(train)
end=len(train)+len(test)-1
predictions1 = AR1fit.predict(start=start, end=end, dynamic=False).
 ↪rename('AR(1) Predictions')
```

[9]:
```
predictions1
```

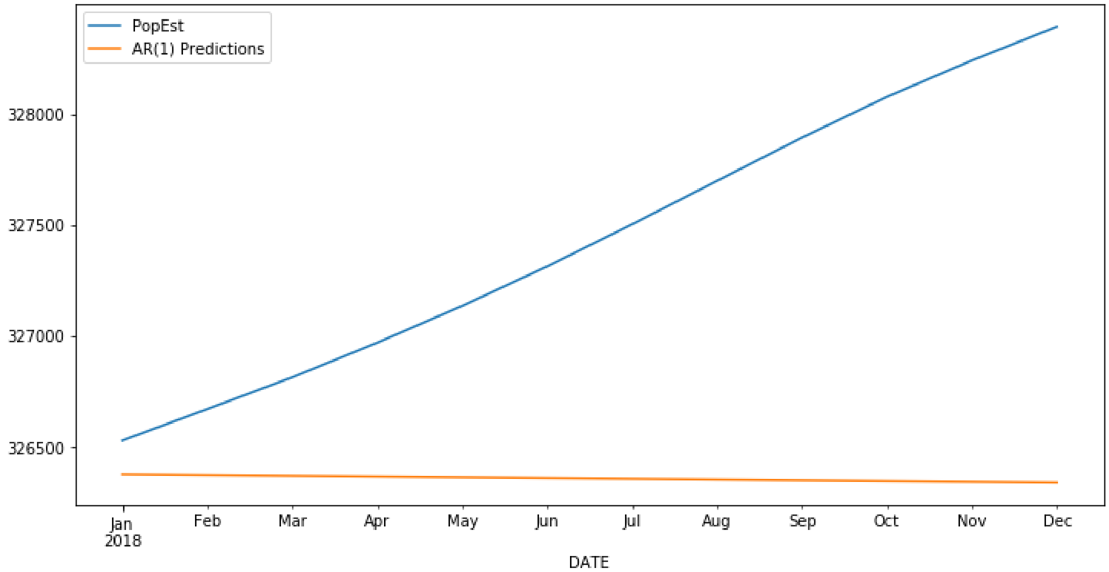
[9]:
```
2018-01-01    326374.598675
2018-02-01    326371.198767
2018-03-01    326367.800275
2018-04-01    326364.403200
2018-05-01    326361.007540
2018-06-01    326357.613294
2018-07-01    326354.220463
2018-08-01    326350.829045
2018-09-01    326347.439040
2018-10-01    326344.050448
2018-11-01    326340.663267
2018-12-01    326337.277498
Freq: MS, Name: AR(1) Predictions, dtype: float64
```

[10]:
```
# Comparing predictions to expected values
for i in range(len(predictions1)):
    print(f"predicted={predictions1[i]:<11.10}, expected={test['PopEst'][i]}")
```

```
predicted=326374.5987, expected=326527
predicted=326371.1988, expected=326669
predicted=326367.8003, expected=326812
predicted=326364.4032, expected=326968
predicted=326361.0075, expected=327134
predicted=326357.6133, expected=327312
predicted=326354.2205, expected=327502
```

```
predicted=326350.829 , expected=327698
predicted=326347.439 , expected=327893
predicted=326344.0504, expected=328077
predicted=326340.6633, expected=328241
predicted=326337.2775, expected=328393
```

[11]:
```
test['PopEst'].plot(legend=True)
predictions1.plot(legend=True,figsize=(12,6));
```



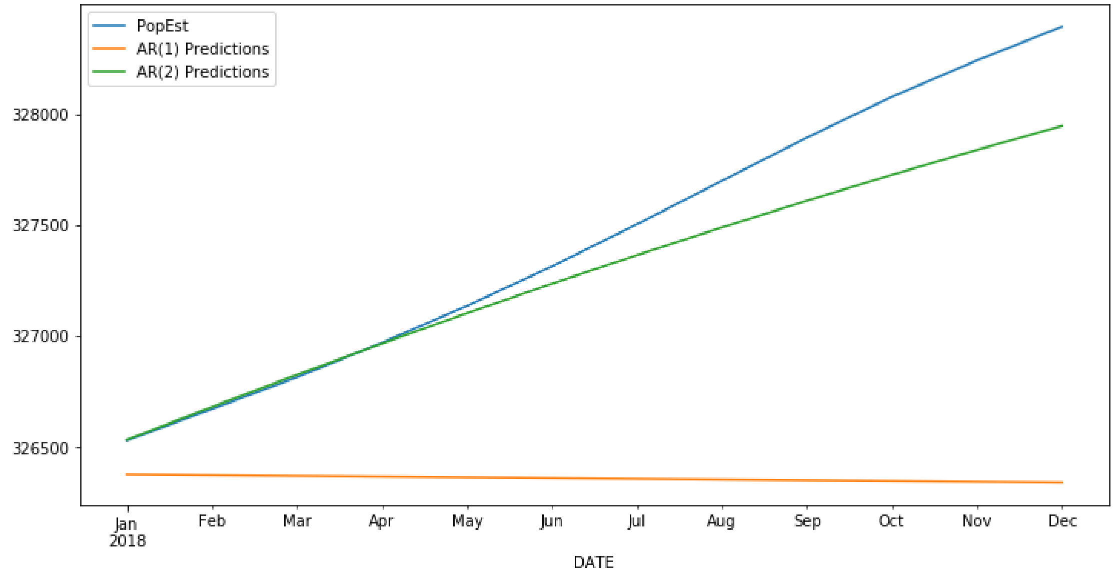
2.5 Fit an AR(2) Model

[12]:
```
# Recall that our model was already created above based on the training set
AR2fit = model.fit(maxlag=2,method='mle')
print(f'Lag: {AR2fit.k_ar}')
print(f'Coefficients:\n{AR2fit.params}')
```

```
Lag: 2
Coefficients:
const        135.359186
L1.PopEst      1.996616
L2.PopEst     -0.997041
dtype: float64
```

[13]:
```
start=len(train)
end=len(train)+len(test)-1
predictions2 = AR2fit.predict(start=start, end=end, dynamic=False).
 ↪rename('AR(2) Predictions')
```

[14]:
```python
test['PopEst'].plot(legend=True)
predictions1.plot(legend=True)
predictions2.plot(legend=True,figsize=(12,6));
```



2.6 Fit an AR(p) model where statsmodels chooses p

This time we'll omit the maxlag argument in AR.fit() and let statsmodels choose a p-value for us.

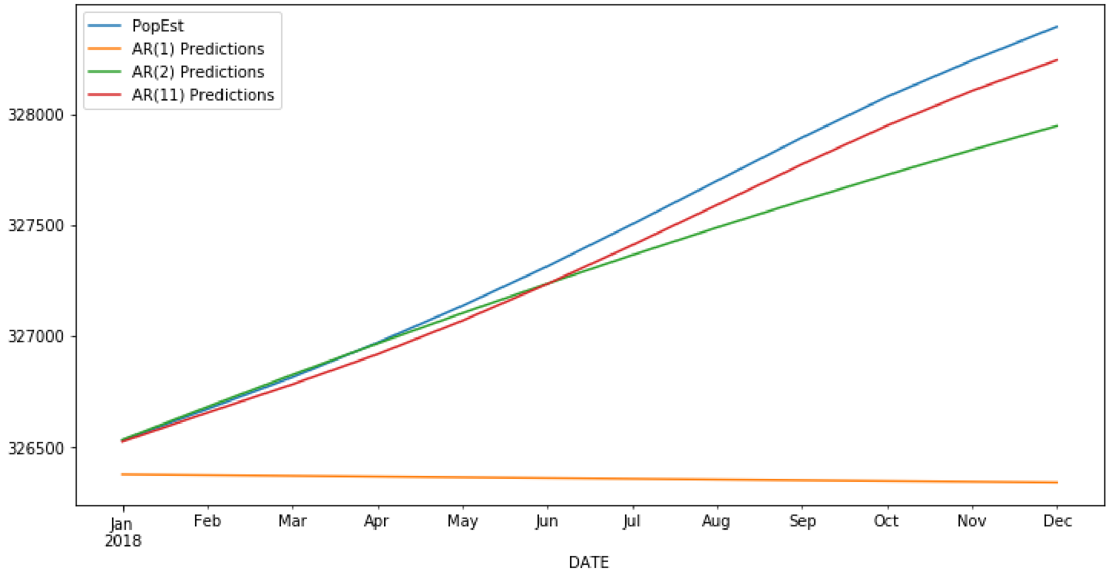
[15]:
```python
ARfit = model.fit(method='mle')
print(f'Lag: {ARfit.k_ar}')
print(f'Coefficients:\n{ARfit.params}')
```

```
Lag: 11
Coefficients:
const          96.145642
L1.PopEst       2.314345
L2.PopEst      -2.213026
L3.PopEst       1.761875
L4.PopEst      -1.430621
L5.PopEst       0.900533
L6.PopEst      -0.968998
L7.PopEst       0.965925
L8.PopEst      -0.264157
L9.PopEst       0.127500
L10.PopEst     -0.194535
L11.PopEst      0.000855
dtype: float64
```

```
[16]: start = len(train)
end = len(train)+len(test)-1
rename = f'AR(11) Predictions'

predictions11 = ARfit.predict(start=start,end=end,dynamic=False).rename(rename)
```

```
[17]: test['PopEst'].plot(legend=True)
predictions1.plot(legend=True)
predictions2.plot(legend=True)
predictions11.plot(legend=True,figsize=(12,6));
```



2.7 Evaluate the Model

It helps to have a means of comparison between two or more models. One common method is to compute the Mean Squared Error (MSE), available from scikit-learn.

```
[18]: from sklearn.metrics import mean_squared_error

labels = ['AR(1)','AR(2)','AR(11)']
preds = [predictions1, predictions2, predictions11]  # these are variables, not␣
 ↪strings!

for i in range(3):
    error = mean_squared_error(test['PopEst'], preds[i])
    print(f'{labels[i]} Error: {error:11.10}')
```

```
AR(1) Error: 1545278.379
AR(2) Error: 53231.37611
AR(11) Error: 9032.545012
```

We see right away how well AR(11) outperformed the other two models.

Another method is the Akaike information criterion (AIC), which does a better job of evaluating models by avoiding overfitting. Fortunately this is available directly from the fit model object.

[19]:
```python
modls = [AR1fit,AR2fit,ARfit]

for i in range(3):
    print(f'{labels[i]} AIC: {modls[i].aic:6.5}')
```

```
AR(1) AIC: 3.4385
AR(2) AIC: 3.4623
AR(11) AIC: 3.6766
```

2.8 Forecasting

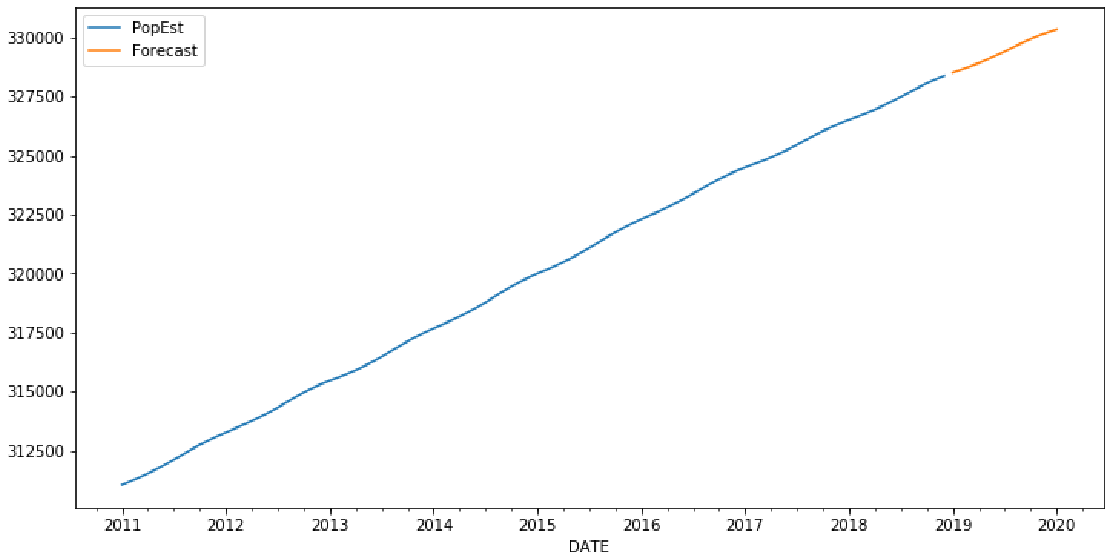
Now we're ready to train our best model on the greatest amount of data, and fit it to future dates.

[20]:
```python
# First, retrain the model on the full dataset
model = AR(df['PopEst'])

# Next, fit the model
ARfit = model.fit(maxlag=11,method='mle')

# Make predictions
fcast = ARfit.predict(start=len(df), end=len(df)+12, dynamic=False).
 ↪rename('Forecast')

# Plot the results
df['PopEst'].plot(legend=True)
fcast.plot(legend=True,figsize=(12,6));
```


PopEst
Forecast
330000
327500
325000
322500
320000
317500
315000
312500
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
DATE

2.9 Great job!