Navigation

Start Here Blog Books About Contact

Search...

How to Make Out-of-Sample Forecasts with ARIMA in Python

by **Jason Brownlee** on March 24, 2017 in **Time Series**

Making out-of-sample forecasts can be confusing when getting started with time series data.

The statsmodels Python API provides functions for performing one-step and multi-step out-of-sample forecasts.

In this tutorial, you will clear up any confusion you have about making out-of-sample forecasts with time series data in Python.

After completing this tutorial, you will know:

- How to make a one-step out-of-sample forecast.
- How to make a multi-step out-of-sample forecast.
- The difference between the *forecast()* and *predict()* functions.

Let's get started.

How to Make Out-of-Sample Forecasts with ARIMA in Pytho
Photo by dziambel, some rights reserved.

Tutorial Overview

This tutorial is broken down into the following 5 steps:

1. Dataset Description
2. Split Dataset
3. Develop Model
4. One-Step Out-of-Sample Forecast
5. Multi-Step Out-of-Sample Forecast

Stop learning Time Series Forecasting the

1. Minimum Daily Temperatures Dataset

This dataset describes the minimum daily temperatures over 10 years (1981-1990) in the city of Melbourne, Australia.

The units are in degrees Celsius and there are 3,650 observations. The source of the data is credite

Learn more about the dataset on Data Market.

Download the Minimum Daily Temperatures dataset to your current working directory with the filenar

Note: The downloaded file contains some question mark ("?") characters that must be removed befc editor and remove the "?" characters. Also, remove any footer information in the file.

The example below loads the dataset as a Pandas Series.

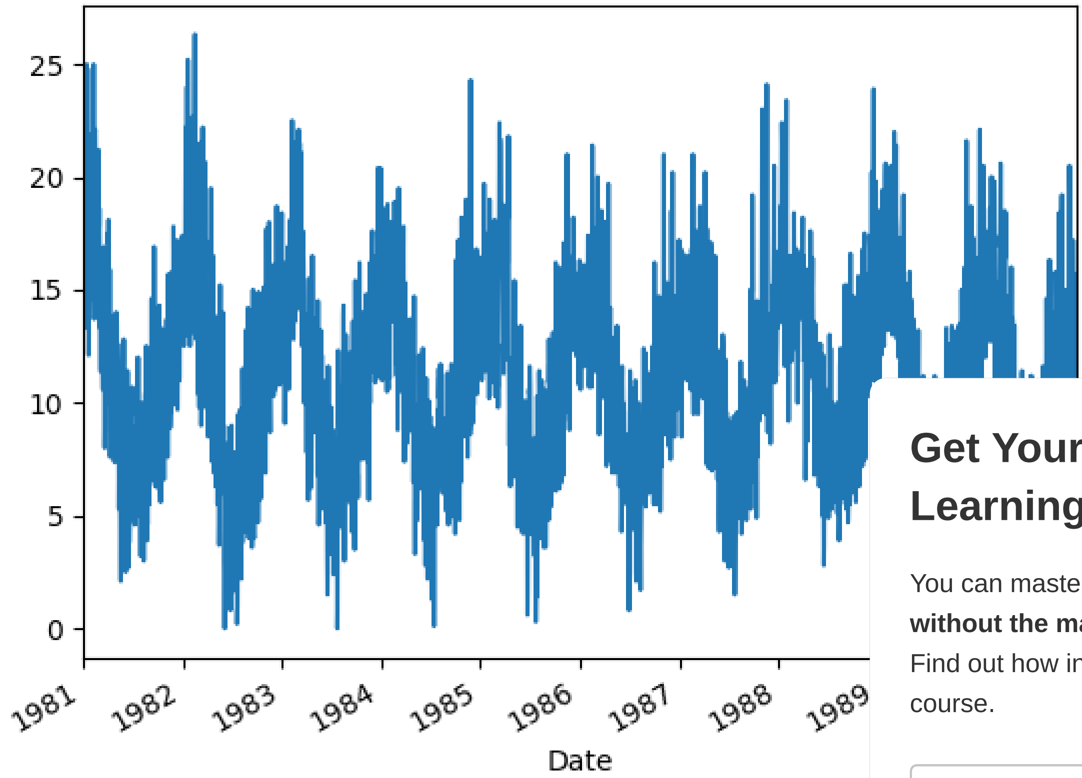

```
# line plot of time series
from pandas import Series
from matplotlib import pyplot
# load dataset
series = Series.from_csv('daily-minimum-temperatures.csv', header=0)
# display first few rows
print(series.head(20))
# line plot of dataset
series.plot()
pyplot.show()
```

Running the example prints the first 20 rows of the loaded dataset.

```
Date
```

```
1981-01-01    20.7
1981-01-02    17.9
1981-01-03    18.8
1981-01-04    14.6
1981-01-05    15.8
1981-01-06    15.8
1981-01-07    15.8
1981-01-08    17.4
1981-01-09    21.8
1981-01-10    20.0
1981-01-11    16.2
1981-01-12    13.3
1981-01-13    16.7
1981-01-14    21.5
1981-01-15    25.0
1981-01-16    20.7
1981-01-17    20.6
1981-01-18    24.8
1981-01-19    17.7
1981-01-20    15.5
```

A line plot of the time series is also created.



Minimum Daily Temperatures Dataset Line Plot

2. Split Dataset

We can split the dataset into two parts.

The first part is the training dataset that we will use to prepare an ARIMA model. The second part is the test dataset that we will pretend is not available. It is these time steps that we will treat as out of sample.

The dataset contains data from January 1st 1981 to December 31st 1990.

We will hold back the last 7 days of the dataset from December 1990 as the test dataset and treat th[redacted]

Specifically 1990-12-25 to 1990-12-31:

```
1990-12-25,12.9
1990-12-26,14.6
1990-12-27,14.0
1990-12-28,13.6
1990-12-29,13.5
1990-12-30,15.7
1990-12-31,13.0
```

The code below will load the dataset, split it into the training and validation datasets, and save them to files *dataset.csv* and *validation.csv* respectively.

```
# split the dataset
from pandas import Series
series = Series.from_csv('daily-minimum-temperatures.csv', header=0)
split_point = len(series) - 7
dataset, validation = series[0:split_point], series[split_point:]
print('Dataset %d, Validation %d' % (len(dataset), len(validation)))
dataset.to_csv('dataset.csv')
validation.to_csv('validation.csv')
```


Run the example and you should now have two files to work with.

The last observation in the dataset.csv is Christmas Eve 1990:

```
1990-12-24,10.0
```

That means Christmas Day 1990 and onwards are out-of-sample time steps for a model trained on c

3. Develop Model

In this section, we are going to make the data stationary and develop a simple ARIMA model.

The data has a strong seasonal component. We can neutralize this and make the data stationary by taking the seasonal difference. That is, we can take the observation for a day and subtract the observation from the same day one year ago.

This will result in a stationary dataset from which we can fit a model.

```
# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return numpy.array(diff)
```

We can invert this operation by adding the value of the observation one year ago. We will need to do this to any forecasts made by a model trained on the seasonally adjusted data.

```
# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]
```

We can fit an ARIMA model.

Fitting a strong ARIMA model to the data is not the focus of this post, so rather than going through th parameters, I will choose a simple ARIMA(7,0,7) configuration.

We can put all of this together as follows:

```
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
import numpy

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return numpy.array(diff)

# load dataset
series = Series.from_csv('dataset.csv', header=None)
# seasonal difference
X = series.values
days_in_year = 365
differenced = difference(X, days_in_year)
# fit model
model = ARIMA(differenced, order=(7,0,1))
model_fit = model.fit(disp=0)
# print summary of fit model
```

```
print(model_fit.summary())
```

Running the example loads the dataset, takes the seasonal difference, then fits an ARIMA(7,0,7) model and prints the summary of the fit model.

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 3278
Model:                     ARMA(7, 1)   Log Likelihood               -8673.748
Method:                       css-mle   S.D. of innovations              3.411
Date:                Mon, 20 Feb 2017   AIC                          17367.497
Time:                        10:28:38   BIC                          17428.447
Sample:                             0   HQIC                         17389.322

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0132      0.132      0.100      0.921      -0.246       0.273
ar.L1.y        1.1424      0.287      3.976      0.000       0.579       1.706
ar.L2.y       -0.4346      0.154     -2.829      0.005      -0.736      -0.133
ar.L3.y        0.0961      0.042      2.289      0.022       0.014       0.178
ar.L4.y        0.0125      0.029      0.434      0.664      -0.044       0.069
ar.L5.y       -0.0101      0.029     -0.343      0.732      -0.068       0.047
ar.L6.y        0.0119      0.027      0.448      0.654      -0.040       0.064
ar.L7.y        0.0089      0.024      0.368      0.713      -0.038       0.056
ma.L1.y       -0.6157      0.287     -2.146      0.032      -1.178      -0.053
                                    Roots
=============================================================================
                 Real           Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           1.2234           -0.0000j            1.2234           -0.0000
AR.2           1.2561           -1.0676j            1.6485           -0.1121
AR.3           1.2561           +1.0676j            1.6485            0.1121
AR.4           0.0349           -2.0160j            2.0163           -0.2472
AR.5           0.0349           +2.0160j            2.0163            0.2472
AR.6          -2.5770           -1.3110j            2.8913           -0.4251
AR.7          -2.5770           +1.3110j            2.8913            0.4251
MA.1           1.6242           +0.0000j            1.6242            0.0000
-----------------------------------------------------------------------------
```


We are now ready to explore making out-of-sample forecasts with the model.

4. One-Step Out-of-Sample Forecast

ARIMA models are great for one-step forecasts.

A one-step forecast is a forecast of the very next time step in the sequence from the available data [redacted]

In this case, we are interested in a one-step forecast of Christmas Day 1990:

```
1990-12-25
```

Forecast Function

The statsmodel ARIMAResults object provides a *forecast()* function for making predictions.

By default, this function makes a single step out-of-sample forecast. As such, we can call it directly and make our forecast. The result of the *forecast()* function is an array containing the forecast value, the standard error of the forecast, and the confidence interval information. Now, we are only interested in the first element of this forecast, as follows.

```
# one-step out-of sample forecast
forecast = model_fit.forecast()[0]
```

Once made, we can invert the seasonal difference and convert the value back into the original scale

```
# invert the differenced forecast to something usable
forecast = inverse_difference(X, forecast, days_in_year)
```

The complete example is listed below:

```
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
import numpy

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return numpy.array(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load dataset
```

```
series = Series.from_csv('dataset.csv', header=None)
# seasonal difference
X = series.values
days_in_year = 365
differenced = difference(X, days_in_year)
# fit model
model = ARIMA(differenced, order=(7,0,1))
model_fit = model.fit(disp=0)
# one-step out-of sample forecast
forecast = model_fit.forecast()[0]
# invert the differenced forecast to something usable
forecast = inverse_difference(X, forecast, days_in_year)
print('Forecast: %f' % forecast)
```

Running the example prints 14.8 degrees, which is close to the expected 12.9 degrees in the *validation.csv* file.

```
Forecast: 14.861669
```


Predict Function

The statsmodel ARIMAResults object also provides a *predict()* function for making forecasts.

The predict function can be used to predict arbitrary in-sample and out-of-sample time steps, includi

The predict function requires a start and an end to be specified, these can be the indexes of the time a used to fit the model, for example:

```
# one-step out of sample forecast
start_index = len(differenced)
end_index = len(differenced)
forecast = model_fit.predict(start=start_index, end=end_index)
```

The start and end can also be a datetime string or a “datetime” type; for example:

```
start_index = '1990-12-25'
end_index = '1990-12-25'
forecast = model_fit.predict(start=start_index, end=end_index)
```

and

```
from pandas import datetime
start_index = datetime(1990, 12, 25)
```

```
end_index = datetime(1990, 12, 26)
forecast = model_fit.predict(start=start_index, end=end_index)
```

Using anything other than the time step indexes results in an error on my system, as follows:

```
AttributeError: 'NoneType' object has no attribute 'get_loc'
```

Perhaps you will have more luck; for now, I am sticking with the time step indexes.

The complete example is listed below:

```
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
import numpy
from pandas import datetime

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return numpy.array(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load dataset
series = Series.from_csv('dataset.csv', header=None)
# seasonal difference
X = series.values
days_in_year = 365
differenced = difference(X, days_in_year)
# fit model
model = ARIMA(differenced, order=(7,0,1))
model_fit = model.fit(disp=0)
# one-step out of sample forecast
start_index = len(differenced)
end_index = len(differenced)
forecast = model_fit.predict(start=start_index, end=end_index)
# invert the differenced forecast to something usable
forecast = inverse_difference(X, forecast, days_in_year)
print('Forecast: %f' % forecast)
```

Running the example prints the same forecast as above when using the *forecast()* function.

```
Forecast: 14.861669
```

You can see that the predict function is more flexible. You can specify any point or contiguous forecast interval in or out of sample.

Now that we know how to make a one-step forecast, we can now make some multi-step forecasts.

5. Multi-Step Out-of-Sample Forecast

We can also make multi-step forecasts using the *forecast()* and *predict()* functions.

It is common with weather data to make one week (7-day) forecasts, so in this section we will look at predicting the minimum daily temperature for the next 7 out-of-sample time steps.

Forecast Function

The *forecast()* function has an argument called *steps* that allows you to specify the number of time s

By default, this argument is set to 1 for a one-step out-of-sample forecast. We can set it to 7 to get a

```
# multi-step out-of-sample forecast
forecast = model_fit.forecast(steps=7)[0]
```

We can then invert each forecasted time step, one at a time and print the values. Note that to invert
forecast value for t+1. Here, we add them to the end of a list called history for use when calling *inver*

```
# invert the differenced forecast to something usable
history = [x for x in X]
day = 1
for yhat in forecast:
    inverted = inverse_difference(history, yhat, days_in_year)
    print('Day %d: %f' % (day, inverted))
    history.append(inverted)
    day += 1
```

The complete example is listed below:

```
from pandas import Series
```

```
from statsmodels.tsa.arima_model import ARIMA
import numpy

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return numpy.array(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load dataset
series = Series.from_csv('dataset.csv', header=None)
# seasonal difference
X = series.values
days_in_year = 365
differenced = difference(X, days_in_year)
# fit model
model = ARIMA(differenced, order=(7,0,1))
model_fit = model.fit(disp=0)
# multi-step out-of-sample forecast
forecast = model_fit.forecast(steps=7)[0]
# invert the differenced forecast to something usable
history = [x for x in X]
day = 1
for yhat in forecast:
    inverted = inverse_difference(history, yhat, days_in_year)
    print('Day %d: %f' % (day, inverted))
    history.append(inverted)
    day += 1
```


Running the example prints the forecast for the next 7 days.

```
Day 1: 14.861669
Day 2: 15.628784
Day 3: 13.331349
Day 4: 11.722413
Day 5: 10.421523
Day 6: 14.415549
Day 7: 12.674711
```

Predict Function

The *predict()* function can also forecast the next 7 out-of-sample time steps.

Using time step indexes, we can specify the end index as 6 more time steps in the future; for example:

```
# multi-step out-of-sample forecast
start_index = len(differenced)
end_index = start_index + 6
forecast = model_fit.predict(start=start_index, end=end_index)
```

The complete example is listed below.

```
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
import numpy

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return numpy.array(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load dataset
series = Series.from_csv('dataset.csv', header=None)
# seasonal difference
X = series.values
days_in_year = 365
differenced = difference(X, days_in_year)
# fit model
model = ARIMA(differenced, order=(7,0,1))
model_fit = model.fit(disp=0)
# multi-step out-of-sample forecast
start_index = len(differenced)
end_index = start_index + 6
forecast = model_fit.predict(start=start_index, end=end_index)
# invert the differenced forecast to something usable
history = [x for x in X]
day = 1
```

```
for yhat in forecast:
    inverted = inverse_difference(history, yhat, days_in_year)
    print('Day %d: %f' % (day, inverted))
    history.append(inverted)
    day += 1
```

Running the example produces the same results as calling the *forecast()* function in the previous section, as you would expect.

```
Day 1: 14.861669
Day 2: 15.628784
Day 3: 13.331349
Day 4: 11.722413
Day 5: 10.421523
Day 6: 14.415549
Day 7: 12.674711
```

Summary

In this tutorial, you discovered how to make out-of-sample forecasts in Python using statsmodels.

Specifically, you learned:

- How to make a one-step out-of-sample forecast.
- How to make a 7-day multi-step out-of-sample forecast.
- How to use both the *forecast()* and *predict()* functions when forecasting.

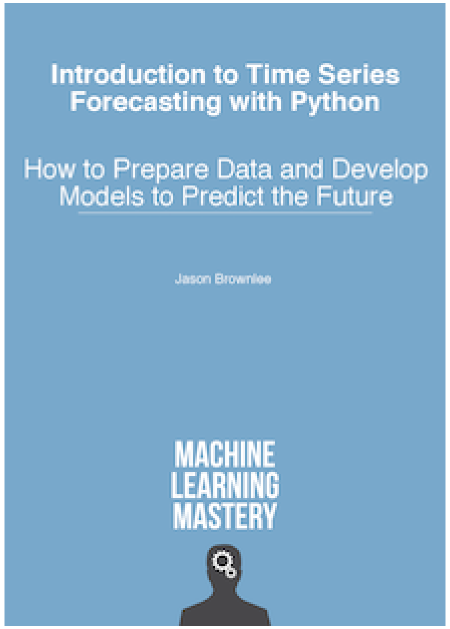
Do you have any questions about out-of-sample forecasts, or about this post? Ask your questions in

Want to Develop Time Series Forecasts with Python?

Develop Your Own Forecasts in Minutes

...with just a few lines of python code

Discover how in my new Ebook:
Introduction to Time Series Forecasting With Python



It covers **self-study tutorials** and **end-to-**[redacted]
Loading data, visualization, modeling, algorithm tuning, and much more...

Finally Bring Time Series Forecasting to Your Own Projects

Skip the Academics. Just Results.

Click to learn more.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional devel
to helping developers get started and get good at applied machine learning. Learn more.
View all posts by Jason Brownlee →

< Time Series Forecasting with Python 7-Day Mini-Course

Sensitivity Analysis of History Size to Forecast Skill with ARIMA in Python >

Get Your Start in Machine Learning

36 Responses to *How to Make Out-of-Sample Forecasts with ARIMA in Python*

Steve March 24, 2017 at 10:44 pm # REPLY

Your tutorials are the most helpful machine learning resources I have found on the Internet and have been hugely helpful in work and personal side projects. I don't know if you take requests but I'd love to see a series of posts on recommender systems one of these days!

Jason Brownlee March 25, 2017 at 7:36 am #

Thanks Steve, and great suggestion! Thanks.

Tim April 27, 2017 at 12:43 pm #

Hi,

This is a really nice example. Do you know if the ARIMA class allows to define the specification of the m
say I have parameters that were estimated using a dataset that I no longer have but I still want to produc

Thanks

Jason Brownlee April 28, 2017 at 7:32 am # REPLY

I expect you can set the coefficients explicitly within the ARIMA model.

Sorry I do not have an example, this post may be relevant:
http://machinelearningmastery.com/make-manual-predictions-arima-models-python/

masum May 11, 2017 at 8:32 pm # REPLY

sir,

would it be possible to do the same using LSTM RNN ?

if it is would you please come up with a blog?

Thanking you

Jason Brownlee May 12, 2017 at 7:41 am # REPLY

Yes.

Any of my LSTM tutorials show how to make out of sample forecasts. For example:
http://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-netw

masum May 12, 2017 at 8:29 pm #

I tried to run the above example without any seasonal difference with given below code.

from pandas import Series
from matplotlib import pyplot
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
# load dataset
series = Series.from_csv('daily-minimum-temperatures.csv', header=0)
print(series.head(20))
series.plot()
pyplot.show()

split_point = len(series) – 7
dataset, validation = series[0:split_point], series[split_point:]

print('Dataset %d, Validation %d' % (len(dataset), len(validation)))
dataset.to_csv('dataset.csv')
validation.to_csv('validation.csv')

series = Series.from_csv('dataset.csv', header=None)
model = ARIMA(series, order=(7,0,1))
model_fit = model.fit(disp=0)

forecast = model_fit.forecast(steps=7)[0]
print('Forecast: %f' % forecast)

for the code i am getting an error:

TypeError: only length-1 arrays can be converted to Python scalars

how can i solve this? it does well for single step forecast

Jason Brownlee May 13, 2017 at 6:13 am #

I would recommend double checking your data, make sure any footer information was dele

Hans June 1, 2017 at 12:58 am #

What does 'seasonal difference' mean?

And what are the details of:

'Once made, we can invert the seasonal difference and convert the value back into the original scale.'

Is it worth to test this code with non-seasonal data or is there another ARIMA-tutorial for non-seasonal approaches on this site?

Jason Brownlee June 2, 2017 at 12:51 pm #

See this post:
http://machinelearningmastery.com/seasonal-persistence-forecasting-python/

And this post:
http://machinelearningmastery.com/time-series-seasonality-with-python/

Please use the search feature of the blog.

Hans June 15, 2017 at 11:27 am # REPLY

If I pretend data in test-partition is not given, does this tutorial do the same except of the seasonal cleaning?

http://machinelearningmastery.com/tune-arima-parameters-python/

Hans June 15, 2017 at 11:29 am #

Can I obtain a train RMSE from this example. Is training involved?

Jason Brownlee June 16, 2017 at 7:47 am #

The model is trained, then the trained model is used to make a forecast.

Consider reading and working through the tutorial.

Hans June 16, 2017 at 12:16 pm # REPLY

I did so several times.
How can I obtain a train RMSE from the model?

Jason Brownlee June 17, 2017 at 7:20 am #

REPLY

See this post on how to estimate the skill of a model prior to using it to make out of sample predictions:
http://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

See this post to understand the difference between evaluating a model and using a final model to make predictions:
http://machinelearningmastery.com/train-final-machine-learning-model/

Hans June 19, 2017 at 5:35 am #

I actually meant obtain a train RMSE from the model in the example.
As I understand the model was trained before making an out of sample prediction.
If we place a

print(model_fit.summary())

right after fitting/training it prints some information's, but no train RMSE.

A)
Is there a way to use the summery-information to obtain a train RMSE?
B)
Is there a way in Python to obtain all properties and methods from the model_fit object-

Jason Brownlee June 19, 2017 at 8:47 am #

Yes, this tutorial assumes you have already estimated the skill of your model and are now ready to use it to make forecasts.

Estimating the skill of the model is a different task. You can do this using walk forward validation or a train/test split evaluation.

Hans June 16, 2017 at 3:06 pm #

REPLY

Is this the line where the training happens?

model = ARIMA(differenced, order=(7,0,1))

Jason Brownlee June 17, 2017 at 7:22 am #

REPLY

No here:

```
model_fit = model.fit(disp=0)
```

Hans June 25, 2017 at 12:29 pm #

Yes I know. I actually thought there could be a direct answer to A) and B). I would use it for archiving.

Hans June 15, 2017 at 12:40 pm #

If I write: 'split_point = len(series) – 0' while my last datapoint in dataset is from today.

Would I have a valid forecast for tomorrow?

M.Swefy June 22, 2017 at 12:39 am #

REPLY

thanks a lot for the nice detailed article, i followed all steps and they all seem working properly, i seek your support Dr. to help me organize my project.

i have a raw data for temperature readings for some nodes (hourly readings), i selected the training set [redacted]
i used ARINA model to train and test and i got Test MSE: 3.716.

now i need to expose the mass raw data to the trained model, then get the forecased values vs. the actual values in the same csv file.

what should i do

M.Swefy June 22, 2017 at 12:41 am # REPLY

*ARIMA

Jason Brownlee June 22, 2017 at 6:09 am #

I'm not sure I follow. Consider this post on how to evaluate a time series model:
http://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

AMU June 23, 2017 at 5:33 am #

Thank you Jason for this wonderful post… It is very detailed and easy to understand..

Do you also have something similar for LSTM Neural Network algorithm as well? something like – How t
Python.

If not, will you write one blog like this with detail explanation? I am sure there are lot of people have the same question.

Jason Brownlee June 23, 2017 at 6:45 am # REPLY

Almost every post I have on LSTMs shows how to make out of sample forecasts. The code

Franklin July 1, 2017 at 1:09 am # REPLY

Hi Jason,

Thanks a lot for this lesson. It was pretty straightforward and easy to follow. It would have been a nice bonus to show how to evaluate the forecasts though with standard metrics. We separated the validation set out and forecasted values for that week, but didn't compare to see how accurate the forecast was.

On that note, I want to ask, does it make sense to use R^2 to score a time series forecast against test data? I'm trying to create absolute benchmarks for a time series that I'm analyzing and want to report unit-independent metrics, i.e. not standard RMSE that is necessarily expressed in the problem's unit scale. What about standardizing the data using zero mean and unit variance, fitting ARIMA, forecasting, and reporting that RMSE? I've been doing this and taking the R^2 and the results are pretty interpretable. RMSE: 0.149 / R^2: 0.8732, but I'm just wondering if doing things this way doesn't invalidate something along the way. Just want to be correct in my process.

Thanks!

Jason Brownlee July 1, 2017 at 6:37 am #

We do that in other posts. Tens of other posts in fact.

This post was laser focused on "how do I make a prediction when I don't know the real answer".

Yes, if R^2 is meaningful to you, that you can interpret it in your domain.

Generally, I recommend inverting all transforms on the prediction and then evaluating model skill at l
apples. This may be less of a concern for an R^2.

Vishanth July 19, 2017 at 6:56 am # REPLY

Seriously amazing. Thanks a lot professor

Jason Brownlee July 19, 2017 at 8:30 am # REPLY

Thanks. Also, I'm not a professor.

Kirui July 20, 2017 at 5:15 pm # REPLY

I get this error from your code

Traceback (most recent call last):
File "..", line 22, in
differenced = difference(X, days_in_year)
File "..", line 9, in difference
value = dataset[i] – dataset[i – interval]
TypeError: unsupported operand type(s) for -: 'str' and 'str'

Cant tell where the problem is.

Jason Brownlee July 21, 2017 at 9:31 am #

Perhaps check that you have loaded your data correct (as real values) and that you have c
space.

Antoine August 23, 2017 at 1:00 am # REPLY

Hi Jason,
Thanks for this detailled explanation. Very clear.

Do you know if it is possible to use the fitted parameters of an ARMA model (ARMAResults.params) and apply it on an other data set ?

I have an online process that compute a forecasting and I would like to have only one learning process [redacted] would like to applied the previously found parameters to the data.

Thanks in advance !

Jason Brownlee August 23, 2017 at 6:56 am #

REPLY

Yes, you can use a grid search:
https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/

Bob October 6, 2017 at 11:53 pm #

Ciao Jason,
Thanks for this tutorial and all the time series related ones. There is always a sense of order in how you
I'm by the way still confused about something which is probably more conceptual about ARIMA.
The ARIMA parameters specify the lag which it uses to forecast.
In your case you used p=7 for example so that you would take into consideration the previous week.
A first silly question is why do I need to fit an entire year of data if Im only looking at my window/lags ?
The second question is that fitting my model I get an error which is really minimal even if I use a short tra
first point.
What am I missing?
Thanks

Jason Brownlee October 7, 2017 at 5:56 am #

REPLY

The model needs lots of examples in order to generalize to new cases.

More data is often better, to a point of diminishing returns in terms of model skill.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Get Good at Time Series Forecasting

Get Your Start in Machine Learning

POPULAR


Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
JULY 21, 2016


Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016


Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016

Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras

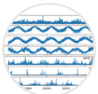

JULY 26, 2016

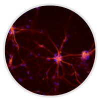
How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017


Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017


Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016


Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016


Multivariate Time Series Forecasting with LSTMs in Keras
AUGUST 14, 2017


How to Implement the Backpropagation Algorithm From Scratch In Python
NOVEMBER 7, 2016

Privacy | Contact | About

Get Your Start in Machine Learning