11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)

Machine learning methods can be used for classification and forecasting on time series problems.

Before exploring machine learning methods for time series, it is a good idea to ensure you have exhausted classical linear time series forecasting methods. Classical time series forecasting methods may be focused on linear relationships, nevertheless, they are sophisticated and perform well on a wide range of problems, assuming that your data is suitably prepared, and the method is well configured.

All code examples are in Python and use the Statsmodels library. The APIs for this library can be tricky for beginners (trust me!), so having a working code example as a starting point will greatly accelerate your progress.

Overview

This cheat sheet demonstrates 11 different classical time series forecasting methods; they are:

1. Autoregression (AR)
2. Moving Average (MA)
3. Autoregressive Moving Average (ARMA)
4. Autoregressive Integrated Moving Average (ARIMA)
5. Seasonal Autoregressive Integrated Moving-Average (SARIMA)
6. Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)
7. Vector Autoregression (VAR)
8. Vector Autoregression Moving-Average (VARMA)
9. Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)
10. Simple Exponential Smoothing (SES)
11. Holt Winter's Exponential Smoothing (HWES)

Each method is presented in a consistent manner.

This includes:

- **Description**. A short and precise description of the technique.
- **Python Code**. A short working example of fitting the model and making a prediction in Python.
- **More Information**. References for the API and the algorithm.

Each code example is demonstrated on a simple contrived dataset.

Autoregression (AR)

The autoregression (AR) method models the next step in the sequence as a linear function of the observations at prior time steps.

The notation for the model involves specifying the order of the model p as a parameter to the AR function, e.g. AR(p). For example, AR(1) is a first-order autoregression model.

The method is suitable for univariate time series without trend and seasonal components.

Python Code

```
# AR example

from statsmodels.tsa.ar_model import AutoReg

from random import random

# contrived dataset

data = [x + random() for x in range(1, 100)]

# fit model

model = AutoReg(data, lags=1)

model_fit = model.fit()

# make prediction

yhat = model_fit.predict(len(data), len(data))

print(yhat)
```

More Information

- statsmodels.tsa.ar_model.AutoReg API
- statsmodels.tsa.ar_model.AutoRegResults API

Moving Average (MA)

The moving average (MA) method models the next step in the sequence as a linear function of the residual errors from a mean process at prior time steps.

A moving average model is different from calculating the moving average of the time series.

The notation for the model involves specifying the order of the model q as a parameter to the MA function, e.g. MA(q). For example, MA(1) is a first-order moving average model.

The method is suitable for univariate time series without trend and seasonal components.

Python Code

We can use the ARIMA class to create an MA model and setting a zeroth-order AR model. We must specify the order of the MA model in the order argument.

```
# MA example

from statsmodels.tsa.arima.model import ARIMA

from random import random

# contrived dataset

data = [x + random() for x in range(1, 100)]

# fit model

model = ARIMA(data, order=(0, 0, 1))

model_fit = model.fit()

# make prediction

yhat = model_fit.predict(len(data), len(data))

print(yhat)
```

Autoregressive Moving Average (ARMA)

The Autoregressive Moving Average (ARMA) method models the next step in the sequence as a linear function of the observations and residual errors at prior time steps.

It combines both Autoregression (AR) and Moving Average (MA) models.

The notation for the model involves specifying the order for the AR(p) and MA(q) models as parameters to an ARMA function, e.g. ARMA(p, q). An ARIMA model can be used to develop AR or MA models.

The method is suitable for univariate time series without trend and seasonal components.

Python Code

```
# ARMA example

from statsmodels.tsa.arima.model import ARIMA

from random import random

# contrived dataset

data = [random() for x in range(1, 100)]

# fit model

model = ARIMA(data, order=(2, 0, 1))

model_fit = model.fit()

# make prediction

yhat = model_fit.predict(len(data), len(data))

print(yhat)
```

Autoregressive Integrated Moving Average (ARIMA)

The Autoregressive Integrated Moving Average (ARIMA) method models the next step in the sequence as a linear function of the differenced observations and residual errors at prior time steps.

It combines both Autoregression (AR) and Moving Average (MA) models as well as a differencing pre-processing step of the sequence to make the sequence stationary, called integration (I).

The notation for the model involves specifying the order for the AR(p), I(d), and MA(q) models as parameters to an ARIMA function, e.g. ARIMA(p, d, q). An ARIMA model can also be used to develop AR, MA, and ARMA models.

The method is suitable for univariate time series with trend and without seasonal components.

Python Code

```
# ARIMA example

from statsmodels.tsa.arima.model import ARIMA

from random import random

# contrived dataset

data = [x + random() for x in range(1, 100)]

# fit model

model = ARIMA(data, order=(1, 1, 1))

model_fit = model.fit()

# make prediction

yhat = model_fit.predict(len(data), len(data), typ='levels')

print(yhat)
```

Seasonal Autoregressive Integrated Moving-Average (SARIMA)

The Seasonal Autoregressive Integrated Moving Average (SARIMA) method models the next step in the sequence as a linear function of the differenced observations, errors, differenced seasonal observations, and seasonal errors at prior time steps.

It combines the ARIMA model with the ability to perform the same autoregression, differencing, and moving average modelling at the seasonal level.

The notation for the model involves specifying the order for the AR(p), I(d), and MA(q) models as parameters to an ARIMA function and AR(P), I(D), MA(Q) and m parameters at the seasonal level, e.g. SARIMA(p, d, q)(P, D, Q)m where “m” is the number of time steps in each season (the seasonal period). A SARIMA model can be used to develop AR, MA, ARMA and ARIMA models.

The method is suitable for univariate time series with trend and/or seasonal components.

Python Code

```
# SARIMA example

from statsmodels.tsa.statespace.sarimax import SARIMAX

from random import random

# contrived dataset

data = [x + random() for x in range(1, 100)]

# fit model

model = SARIMAX(data, order=(1, 1, 1), seasonal_order=(0, 0, 0, 0))

model_fit = model.fit(disp=False)

# make prediction

yhat = model_fit.predict(len(data), len(data))

print(yhat)
```

More Information

- statsmodels.tsa.statespace.sarimax.SARIMAX API
- statsmodels.tsa.statespace.sarimax.SARIMAXResults API

Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)

The Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX) is an extension of the SARIMA model that also includes the modelling of exogenous variables.

Exogenous variables are also called covariates and can be thought of as parallel input sequences that have observations at the same time steps as the original series. The primary series may be referred to as endogenous data to contrast it from the exogenous sequence(s). The observations for exogenous variables are included in the model directly at each time step and are not modelled in the same way as the primary endogenous sequence (e.g. as an AR, MA, etc. process).

The SARIMAX method can also be used to model the subsumed models with exogenous variables, such as ARX, MAX, ARMAX, and ARIMAX.

The method is suitable for univariate time series with trend and/or seasonal components and exogenous variables.

Python Code

```
# SARIMAX example

from statsmodels.tsa.statespace.sarimax import SARIMAX

from random import random

# contrived dataset

data1 = [x + random() for x in range(1, 100)]
```

```
data2 = [x + random() for x in range(101, 200)]

# fit model

model = SARIMAX(data1, exog=data2, order=(1, 1, 1), seasonal_order=(0, 0, 0, 0))

model_fit = model.fit(disp=False)

# make prediction

exog2 = [200 + random()]

yhat = model_fit.predict(len(data1), len(data1), exog=[exog2])

print(yhat)
```

More Information

- statsmodels.tsa.statespace.sarimax.SARIMAX API
- statsmodels.tsa.statespace.sarimax.SARIMAXResults API

Vector Autoregression (VAR)

The Vector Autoregression (VAR) method models the next step in each time series using an AR model. It is the generalization of AR to multiple parallel time series, e.g. multivariate time series.

The notation for the model involves specifying the order for the AR(p) model as parameters to a VAR function, e.g. VAR(p).

The method is suitable for multivariate time series without trend and seasonal components.

Python Code

```
# VAR example

from statsmodels.tsa.vector_ar.var_model import VAR

from random import random

# contrived dataset with dependency

data = list()

for i in range(100):

   v1 = i + random()

   v2 = v1 + random()

   row = [v1, v2]

   data.append(row)

# fit model

model = VAR(data)
```

```
model_fit = model.fit()

# make prediction

yhat = model_fit.forecast(model_fit.y, steps=1)

print(yhat)
```

More Information

- statsmodels.tsa.vector_ar.var_model.VAR API
- statsmodels.tsa.vector_ar.var_model.VARResults API

Vector Autoregression Moving-Average (VARMA)

The Vector Autoregression Moving-Average (VARMA) method models the next step in each time series using an ARMA model. It is the generalization of ARMA to multiple parallel time series, e.g. multivariate time series.

The notation for the model involves specifying the order for the AR(p) and MA(q) models as parameters to a VARMA function, e.g. VARMA(p, q). A VARMA model can also be used to develop VAR or VMA models.

The method is suitable for multivariate time series without trend and seasonal components.

Python Code

```
# VARMA example

from statsmodels.tsa.statespace.varmax import VARMAX

from random import random

# contrived dataset with dependency

data = list()

for i in range(100):

   v1 = random()

   v2 = v1 + random()

   row = [v1, v2]

   data.append(row)

# fit model

model = VARMAX(data, order=(1, 1))

model_fit = model.fit(disp=False)

# make prediction

yhat = model_fit.forecast()

print(yhat)
```

More Information

- statsmodels.tsa.statespace.varmax.VARMAX API
- statsmodels.tsa.statespace.varmax.VARMAXResults

Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)

The Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX) is an extension of the VARMA model that also includes the modeling of exogenous variables. It is a multivariate version of the ARMAX method.

Exogenous variables are also called covariates and can be thought of as parallel input sequences that have observations at the same time steps as the original series. The primary series(es) are referred to as endogenous data to contrast it from the exogenous sequence(s). The observations for exogenous variables are included in the model directly at each time step and are not modeled in the same way as the primary endogenous sequence (e.g. as an AR, MA, etc. process).

The VARMAX method can also be used to model the subsumed models with exogenous variables, such as VARX and VMAX.

The method is suitable for multivariate time series without trend and seasonal components with exogenous variables.

Python Code

```
# VARMAX example

from statsmodels.tsa.statespace.varmax import VARMAX

from random import random

# contrived dataset with dependency

data = list()

for i in range(100):

    v1 = random()

    v2 = v1 + random()

    row = [v1, v2]

    data.append(row)

data_exog = [x + random() for x in range(100)]

# fit model

model = VARMAX(data, exog=data_exog, order=(1, 1))

model_fit = model.fit(disp=False)

# make prediction

data_exog2 = [[100]]
```

```
yhat = model_fit.forecast(exog=data_exog2)

print(yhat)
```

More Information

- statsmodels.tsa.statespace.varmax.VARMAX API
- statsmodels.tsa.statespace.varmax.VARMAXResults

Simple Exponential Smoothing (SES)

The Simple Exponential Smoothing (SES) method models the next time step as an exponentially weighted linear function of observations at prior time steps.

The method is suitable for univariate time series without trend and seasonal components.

Python Code

```
# SES example

from statsmodels.tsa.holtwinters import SimpleExpSmoothing

from random import random

# contrived dataset

data = [x + random() for x in range(1, 100)]

# fit model

model = SimpleExpSmoothing(data)

model_fit = model.fit()

# make prediction

yhat = model_fit.predict(len(data), len(data))

print(yhat)
```

More Information

- statsmodels.tsa.holtwinters.SimpleExpSmoothing API
- statsmodels.tsa.holtwinters.HoltWintersResults API

Holt Winter’s Exponential Smoothing (HWES)

The Holt Winter’s Exponential Smoothing (HWES) also called the Triple Exponential Smoothing method models the next time step as an exponentially weighted linear function of observations at prior time steps, taking trends and seasonality into account.

The method is suitable for univariate time series with trend and/or seasonal components.

Python Code

```
# HWES example
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = ExponentialSmoothing(data)
model_fit = model.fit()
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

More Information

- statsmodels.tsa.holtwinters.ExponentialSmoothing API
- statsmodels.tsa.holtwinters.HoltWintersResults API

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Statsmodels: Time Series analysis API
- Statsmodels: Time Series Analysis by State Space Methods