2.Introduction-to-Forecasting---SES

2020年5月25日 星期一 下午2:13

_7925cda2
857e2aa...

At this point, we can look at a time series and, using software or by developing our own code, estimate the order of the process and estimate the coefficients describing autoregressive or moving average components. We also know how to judge the quality of a model.

To push further, we will explore ways to use our past data in order to say something intelligent about what values we are likely to observe in the future.

Simple Exponential Smoothing

In the handy reference *A Little Book of R for Time Series* by Avril Coghlan, the author reads in a classic data set describing the total annual rainfall recorded during the years 1813-1912, values measured in inches, for London, England (original data from Hipel and McLeod, 1994). You can access these data from the comfort and safety of your R script with the *scan()* command. The author's R code has been edited slightly and is presented below.

go to website - grab data - store in array

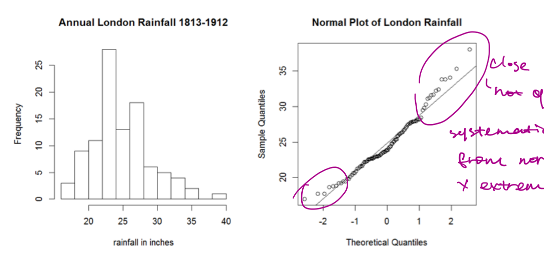
```
rm(list=ls(all=TRUE))rain.data <-
scan("http://robjhyndman.com/tsdldata/hurst/precip1.dat",skip=1)
rain.ts <- ts(rain.data, start=c(1813))
```

We'd like to produce some plots summarizing the nature of our data. The histogram shows data that are not quite symmetrical and mound shaped.

```
par( mfrow=c(1,2) )
hist(rain.data, main="Annual London Rainfall 1813-1912",
        xlab="rainfall in inches")
```

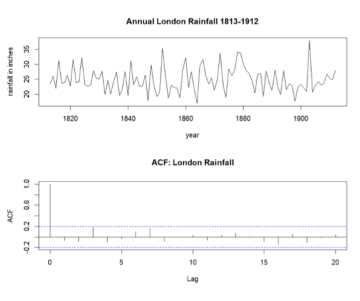
normal distributed or not

```
qqnorm(rain.data,main="Normal Plot of London Rainfall")
qqline(rain.data)
```



close
not quite normal
systematic departures from normality
✗ extreme

Of course we also plot the data and look at the ACF.



t.s. → sequence → ACF
dataset - noise
✗ structure

```
par( mfrow=c(2,1) )
plot.ts(rain.ts, main="Annual London Rainfall 1813-1912",
                              xlab="year", ylab="rainfall in inches")

acf(rain.ts, main="ACF: London Rainfall")
```

It's hard to look at these plots and think of an obvious model. The data themselves exhibit some skew and aren't terribly far from normally distributed. The autocorrelations seem pretty weak. Even *auto.arima()* kind of gives up on us.

```
library(forecast)
auto.arima(rain.ts)

Series: rain.ts
ARIMA(0,0,0) with non-zero mean

Coefficients:
         mean
         24.8239
s.e.     0.4193

sigma^2 estimated as 17.76:  log likelihood=-285.25
AIC=574.49   AICc=574.61   BIC=579.7
```

no fit model

✗ MA coef.

If you could hop in your Tardis and time travel back to 1912, would you be able to make a prediction about the 1913 rainfall given the data gathered in the years prior? How would you do it? (Make the prediction, not the time travel).

Naïve method

The simplest way we can think to make a forecast is to take the previous year's data and just assume that will happen again in the current year. In our data set, the last few years have annual rainfall as listed.

YEAR	1909	1910	1911	1912	1913
RAINFALL	26.75	25.36	24.79	27.88	???

In this naïve method, you would forecast that the rainfall in 1913 will be 27.88 inches. If you can find someone who will pay you to make predictions in this manner please let me know because I usually have to work harder for my money.

Average method

We should pause and develop a little notation. Let's assume that we have a time series of *n* points, listed in the usual way as $x_1, x_2, \ldots, x_n$. Even though it's fun to watch people fight about the differences between prediction and forecasting, but we'll just loosely *consider a forecast to be a prediction about a future value*. We will then make a forecast about the future value *h* lags away based upon data currently observed at the n[th] time, denoted as x_{n+h}^n.

$$x_{n+h}^n = \text{forecast at time n + h made from observations available at time n}$$

$$x_{n+h}^{n} = x_{\text{time for which you'd like a forecast}}^{\text{time for which you actually have data}}$$

time step n
data using for forecast
daily data
how many periods into future
Tues → Next Tues h=7

In the current case, the naïve method would give a one step ahead forecast as

observed value at time period n
time step
today
forecast

$$x_{n+1}^n = x_n \quad (naïve\ method)$$

A slight increase in complexity, especially if you believe there is some seasonality, is to predict the next future value based upon the corresponding value at the previous season.

$$x_{n+1}^n = x_{n+1-S} \quad (seasonal\ naïve\ method)$$

next period n+1
1 season ago.
weeks → S=7

Just a little bit fancier, another very simple forecasting method is to predict the upcoming value as the average of *all* of the past values, weighted equally

$$x_{n+1}^n = \frac{\sum_{i=1}^n x_i}{n} \quad (average\ method)$$

In this average method, you would forecast that the rainfall in 1913 will be 24.8239 inches. Frankly, I still don't think you are earning your salary.

Simple Exponential Smoothing

If you'd like to make a forecast based upon past values, but would like to give a greater weighting to values that are more recent, consider *Simple Exponential Smoothing* (*SES*). All we need is a decaying sequence of weights. The geometric series is a pretty obvious choice since they decrease at a constant ratio. We scale to make the weights add to 1.

Our commonly used convergence result is that

$$\sum_{k=0}^{\infty}(1-\alpha)^k = \frac{1}{\alpha}$$

For example, if we let $\alpha = \frac{1}{2}$ we'd say that

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 2$$

Multiply both sides of the sum by α and we can say that

$$\sum_{k=0}^{\infty}\alpha(1-\alpha)^k = 1 = \alpha + \alpha(1-\alpha) + \alpha(1-\alpha)^2 + \cdots + \alpha(1-\alpha)^k + \cdots$$

We can use these values as our weights. In this forecasting framework we would write

$$x_{n+1}^n = \alpha x_n + \alpha(1-\alpha)x_{n-1} + \alpha(1-\alpha)^2 x_{n-2} + \cdots + \alpha(1-\alpha)^k x_{n-k} + \cdots \quad (SES)$$

We form our weights and apply them looking back in time. You may complain that only a mathematician would create a weighting method involving an infinitely long series. Fair enough, but we can express this in so called ***recurrence form*** by creating the forecast at time *n+1* as an

update to the forecast at time *n* (which is itself based upon data up until *n-1*), etc. So, start with the forecast for time n based upon the preceding available data up to time $n-1$

$$x_n^{n-1} = \alpha x_{n-1} + \alpha(1-\alpha)x_{n-2} + \alpha(1-\alpha)^2 x_{n-3} + \cdots$$

A simple trick here:

$$(1-\alpha)\, x_n^{n-1} = \alpha(1-\alpha)x_{n-1} + \alpha(1-\alpha)^2 x_{n-2} + \alpha(1-\alpha)^3 x_{n-3} + \cdots$$

Let's use this to obtain the forecast for time $n+1$ based upon data up until time n.

$$x_{n+1}^n = \alpha x_n + \alpha(1-\alpha)x_{n-1} + \alpha(1-\alpha)^2 x_{n-2} + \cdots + \alpha(1-\alpha)^k x_{n-k} + \cdots$$

$$x_{n+1}^n = \alpha x_n + (1-\alpha)\, x_n^{n-1}$$

You can see that we form a new forecast by weighting the previous forecast x_n^{n-1} and updating with the fresh data point x_n. We need to start somewhere, so define

$$x_2^1 = x_1$$

and then proceed inductively.

$$x_3^2 = \alpha\, x_2 + (1-\alpha)x_2^1$$

$$x_4^3 = \alpha\, x_3 + (1-\alpha)x_3^2$$

$$\vdots$$

Not to belabor the point, but you can write this as

$$\text{new forecast at time } n+1 = \alpha \cdot \text{data at time } n + (1-\alpha) \cdot \text{ old forecast at time } n$$

If you let $\alpha = 1$ you are of course back to naïve forecasting. Large values of α (i.e. those values of $\alpha \approx 1$) will cause the series to decay rapidly and put great emphasis on "near" observations. Smaller values of α cause the series to decay more slowly and thus afford a larger weighting to data points further in the past. We can easily write a for-loop to implement this procedure. It's instructive to implement SES yourself before calling a package.


```
alpha=.2                      #increase alpha for more rapid decay
forecast.values = NULL        #establish array to store forecast values

n = length(rain.data)

#naive first forecast
forecast.values [1] = rain.data[1]

#loop to create all forecast values
for( i in 1:n ) {
        forecast.values [i+1] = alpha*rain.data[i] + (1-alpha)* forecast.values [i]
}

paste("forecast for time",n+1," = ", forecast.values [n+1])
```

The *paste()* command may be new to us, but it's pretty obvious and allows us to concatenate strings.

```
"forecast at time:  101  =  25.3094062064236"
```

Moving Towards a Least Squares Approach

The choice of $\alpha = 0.2$ was totally unmotivated in the preceding. To find the optimal α value for a particular time series we can look at our forecast errors. Since we produce forecasts at every time step we can keep a running total of how we are doing. Define the prediction error at time n as

$$e_n \equiv \text{data value at time } n - \text{forecast value for time } n$$

$$e_n \equiv x_n - x_n^{n-1}$$

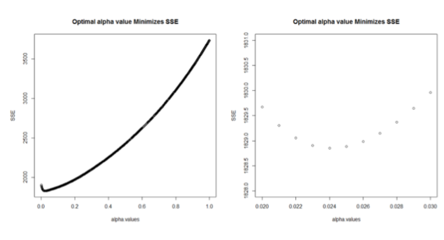
A measure of quality is then

$$\mathrm{SSE}(\alpha) = \sum_{i=1}^{n}(x_n - x_n^{n-1})^2$$

We can calculate SSE(α) for various values between 0 and 1 and see which returns the smallest aggregate error. I did this by placing the previous code in another for-loop. As usual, the code is meant to be transparently obvious, not display coding efficiency. (Excuses, excuses…)


```
SSE=NULL
n = length(rain.data)
alpha.values = seq( .001, .999, by=0.001)
number.alphas = length(alpha.values)

for( k in 1:number.alphas ) {
 forecast.values=NULL
 alpha = alpha.values[k]
 forecast.values[1] = rain.data[1]
 for( i in 1:n ) {
   forecast.values[i+1] = alpha*rain.data[i] + (1-alpha)*forecast.values[i]
 }
 SSE[k] = sum( (rain.data - forecast.values[1:n])^2 )
}
plot(SSE~alpha.values, main="Optimal alpha value Minimizes SSE")
```



Our best α in terms of minimizing the aggregate squared error is around 0.024 (found by zooming or by "interrogating" the SSE array to find which SSE value is the smallest, then retrieving the corresponding α value.

```
index.of.smallest.SSE = which.min(SSE)      #returns position 24

alpha.values[which.min(SSE)]                #returns 0.024
```


If you run with $\alpha = 0.024$ you will make a prediction

```
"forecast at time:  101  =  24.6771392918524"
```

In the next lecture we learn about Holt-Winters as a forecasting procedure. For now, we will use the R command *HoltWinters()* to perform the SES for us by setting a couple of parameters equal to zero, and observing our results. Our forecasts differ slightly because our alpha is very slightly less accurate (but not bad for just zooming!) If you are patient and have a reasonably fast machine, you can get a little closer with a finer grid.

```
HoltWinters(rain.ts, beta=FALSE, gamma=FALSE)

Holt-Winters exponential smoothing without trend and without seasonal component.

Call:
HoltWinters(x = rain.ts, beta = FALSE, gamma = FALSE)

Smoothing parameters:
 alpha:       0.02412151
 beta :       FALSE
 gamma:       FALSE

Coefficients: a 24.67819
```