Time series forecasting

Machine learning methods

Julien JACQUES

Université Lumière Lyon 2

Neural network models

Other Machine Learning models

Neural network models

Neuron

A neuron is a *model*, with p features, which map the p *inputs* $x^1, \ldots, x^p$ to an *output* y :

$$y = g\left(\alpha_0 + \sum_{j=1}^{p} \alpha_j x^j\right)$$

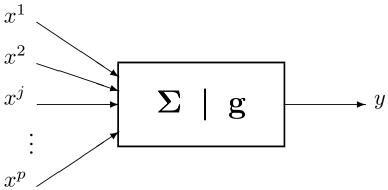


Figure 1: Neuron representation

- Σ: linear combination of inputs
- g: activation function

A specific neuron: linear model

One neuron with linear activation function $g(x) = x$ is the usual *linear model*:

$$y = \alpha_0 + \sum_{j=1}^{p} \alpha_j x^j$$

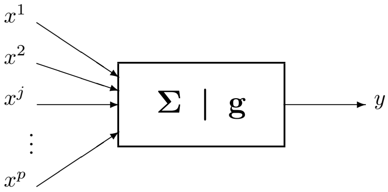


Figure 2: Neuron representation

Neural networks

A neural network is the association of several neurons, in a more or less complex graph, characterized by:

- its architecture (layer ...)
- its complexity (number of neurons, presence of loops)
- activation functions
- the objective: supervised or unsupervised learning ...

Multilayer perceptron

- A multilayer perceptron is made up of **layers**
- Layer: set of neurons without connection between them
- It has an input layer, an output layer, and **one or more hidden layers**
- The neurons are all connected at the input to each of the neurons of the previous layer and at the output to each of the neurons of the next layer

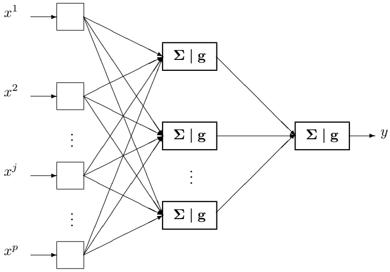


Figure 3: Multilayer perceptron with 1 hidden layer

Neural Network Auto-Regression (NNAR)

For **non seasonal** data:

- $NNAR_{p,k}$ model:
  - Inputs: lagged values of the time series $x_{t-1}, \ldots, x_{t-p}$
  - 1 hidden layer with k neurons
  - sigmoïd activation function
  - Rk: $NNAR_{p,0} = AR_p$

Neural Network Auto-Regression (NNAR)

For **non seasonal** data:

- $NNAR_{p,k}$ model:
  - Inputs: lagged values of the time series $x_{t-1}, \dots, x_{t-p}$
  - 1 hidden layer with k neurons
  - sigmoïd activation function
  - Rk: $NNAR_{p,0} = AR_p$

For **seasonal** data (of period T), we add lagged values from the same season as last observed values:

- $NNAR_{(p,P,k)_T}$ model:
  - Inputs: lagged values of the time series $x_{t-1}, x_{t-2}, \dots, x_{t-p}, x_{t-T}, x_{t-2T}, \dots, x_{t-PT}$
  - 1 hidden layer with k neurons
  - sigmoïd activation function
  - Rk: $NNAR_{(p,P,0)_T} = SARIMA_{(p,0,0)(P,0,0)_T}$

`nnetar` function

Estimation of an $NNAR_{(p,P,k)_T}$ with the `forecast` package:

```
nnetar(x, p, P, size=k)
```

- if `p` not specified, it is chosen automatically by minimizing AIC of a linear AR_p model
- if `P` not specified, $P = 1$ is chosen
- if `k` not specified, $k = (p + P + 1)/2$ is chosen

Other options:

- `xreg` allows to add external regressors
- `lambda` allows to use Box-Cox transformation

Neural Network Auto-Regression (NNAR)

- Advantage over a linear model (AR_p):
  - more flexible, modeling non-linear relation
- Dis-advantage over a linear model (AR_p):
  - none well-defined sochastic model -> prediction interval not direct (need boostrap simulations, option `PI=TRUE`)
  - not possible to *integrate* differecing

More Neural Network

More sophisticated neural networks for time series as Recurrent Neural Network (but also LSTM, GRU…).

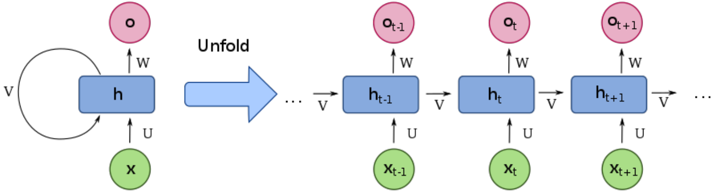


Figure 4: Neuron representation

RNN in R

RNN are implemented in the Keras lib. for Python.

We can used it through the `keras` R package

```
library(keras)
```

The idea is to split the whole time series into sub-series.

For instance for weekly data:

- $x_7 = f(x_6, \ldots, x_1)$
- $x_{14} = f(x_{13}, \ldots, x_8)$
- and so on
- $x_{n-7} = f(x_{n-8}, \ldots, x_{n-13})$

A neural network is learned to estimate f, and then used to forecast x_{n+1}

- $\hat{x}_{n+1} = \hat{f}(x_n, \ldots, x_{n-6})$

More Neural Network

To my experience, such models:

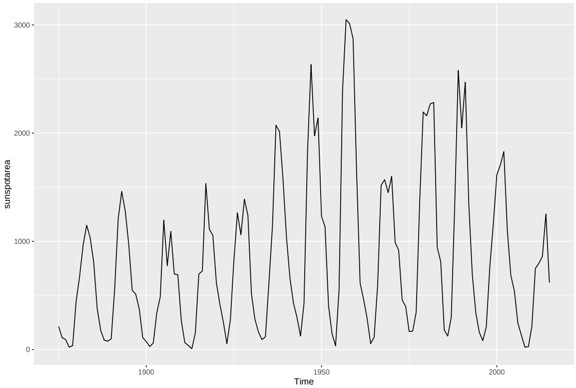
- are efficient when the series is hard to forecast, with no evident model behind and when usual model are not efficient
- need time series of large sizes

Note that you can use RNN directly from R thanks to the `keras` package.

If you want to make your own opinion, let have a look for instance to: https://www.r-bloggers.com/2020/05/time-series-with-arima-and-rnn-models/

Example: sunspots

```
autoplot(sunspotarea)
```



No seasonal but **cyclic pattern** $\Rightarrow$ can not be modelized by usual linear *SARIMA* models

Example: sunspots

$NNAR_{p,k}$ model estimation, with automatic choice of p and k:

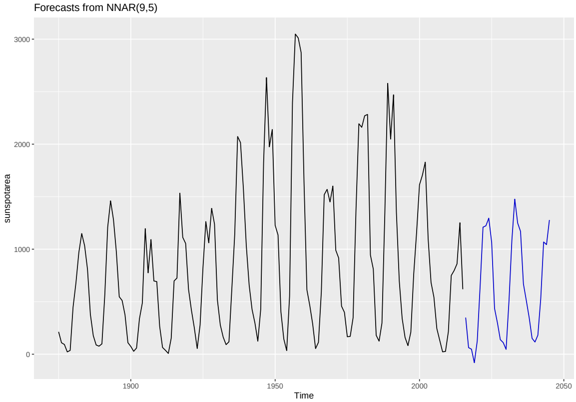
```
fit=nnetar(sunspotarea)
print(fit)
```

```
## Series: sunspotarea
## Model:  NNAR(9,5)
## Call:   nnetar(y = sunspotarea)
##
## Average of 20 networks, each of which is
## a 9-5-1 network with 56 weights
## options were - linear output units
##
## sigma^2 estimated as 10769
```

Example: sunspots

Forecasting for next 30 years:

```
autoplot(forecast(fit,h=30))
```



- asymetric cyclicity as been modelled well

Exercice: San Francisco precipitation

San Fransisco precipitation from 1932 to 1966 are available here:
http://eric.univ-lyon2.fr/jjacques/Download/DataSet/sanfran.dat

- Try to improve your forecasts obtained with exponential smoothing and SARIMA models with neural network models

Exercices:

Try to improve all your previous forecast with NN ! . . . good luck ;-)

Other Machine Learning models

Data prepration

- any machine learning model can be tune once we correctly split the time series $x_1, \ldots, x_n$ into input|output data set:

$$
\begin{array}{ccc|c}
 & \textit{inputs} & & \textit{output} \\
x_1 & \cdots & x_T & x_{T+1} \\
 & & & \\
x_{n-T-1} & \cdots & x_{n-2} & x_{n-1} \\
x_{n-T} & \cdots & x_{n-1} & x_n
\end{array}
$$

- then, we just have to learn any function f:

$$x_n = f(x_{n-T}, \ldots, x_{n-1})$$

- f can be Random Forest, SVM...

Data prepration for San Francisco data set

```
data=scan(file="data/sanfran.csv",skip=1)
sanfran<-ts(data,start=c(1932,1),end=c(1966,12),freq=12)
library(forecast)
sanfran_train=window(sanfran,start=c(1932,1),end=c(1963,12))
sanfran_test=window(sanfran,start=c(1964,1),end=c(1966,12))
data=as.vector(sanfran_train)[1:13]
for (i in 1:(length(as.vector(sanfran_train))-13)){
data=rbind(data,as.vector(sanfran_train)[(i+1):(i+13)])
}
print(head(sanfran))
```

```
##        Jan   Feb   Mar   Apr   May   Jun
## 1932 16.26 29.46 18.03 24.13 22.35 22.10
```

```
print(data[1:2,1:6])
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
## data 16.26 29.46 18.03 24.13 22.35 22.10
##      29.46 18.03 24.13 22.35 22.10 12.95
```

Random Forest

We fit the model

```
library(randomForest)
fitRF=randomForest(x=data[,-13], y=data[,13])
```

And then sequentially forecast the next 36 values

```
pred=rep(NULL,36)
newdata=tail(sanfran_train,12)
for (t in 1:36){
  pred[t]=predict(fitRF,newdata=newdata)
  newdata=c(newdata[-1],pred[t])
}
prevRF=ts(pred,start=c(1964,1),end=c(1966,12),frequency = 1
```

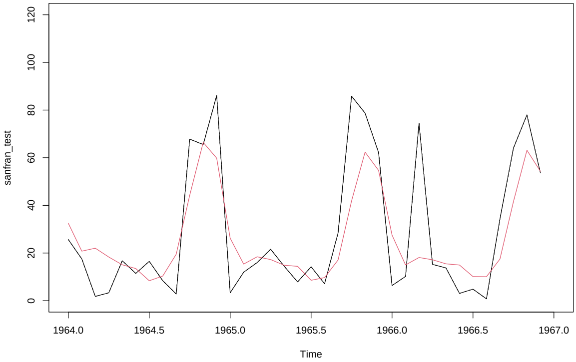
Random Forest

Forecasting results

```
print(sqrt(mean((prevRF-sanfran_test)^2)))
```

```
## [1] 16.86585
```

```
plot(sanfran_test,xlim=c(1964,1967),ylim=c(0,120))
lines(sanfran_test,lty=2)
lines(prevRF,col=2)
```



SVM

Model fitting and sequential forecasting

```
library(e1071)
fitSVM=svm(x=data[,-13], y=data[,13])
pred=rep(NULL,36)
newdata=tail(sanfran_train,12)
for (t in 1:36){
  pred[t]=predict(fitSVM,newdata=matrix(newdata,1,12))
  newdata=c(newdata[-1],pred[t])
}
prevSVM=ts(pred,start=c(1964,1),end=c(1966,12),frequency =
```

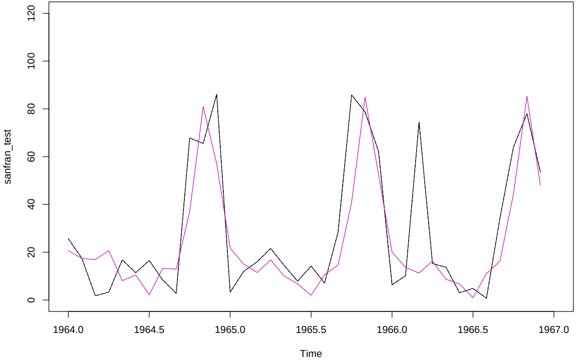
SVM

Forecasting results

```
print(sqrt(mean((prevSVM-sanfran_test)^2)))
```

```
## [1] 17.53055
```

```
plot(sanfran_test,xlim=c(1964,1967),ylim=c(0,120))
lines(sanfran_test,lty=2)
lines(prevSVM,col=6)
```



LM

Model fitting and sequential forecasting

```
data=data.frame(data)
colnames(data)=c('x1','x2','x3','x4','x5','x6','x7','x8','x
fitLM=lm(y~.,data=data)
pred=rep(NULL,36)
newdata=data.frame(matrix(tail(sanfran_train,12),1,12))
colnames(newdata)=c('x1','x2','x3','x4','x5','x6','x7','x8
for (t in 1:36){
  pred[t]=predict(fitLM,newdata=newdata)
  newdata=data.frame(c(newdata[-1],pred[t]))
  colnames(newdata)=c('x1','x2','x3','x4','x5','x6','x7','x
}
prevLM=ts(pred,start=c(1964,1),end=c(1966,12),frequency = 1
```

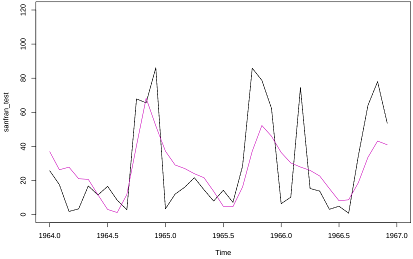
LM

Forecasting results

```
print(sqrt(mean((prevLM-sanfran_test)^2)))
```

```
## [1] 20.49199
```

```
plot(sanfran_test,xlim=c(1964,1967),ylim=c(0,120))
lines(sanfran_test,lty=2)
lines(prevLM,col=6)
```



To go further

- forecasting is done sequentially : $\hat{x}_{n+2}$ uses $\hat{x}_{n+1}$.
- improvement would be to simultaneously forecast $(\hat{x}_{n+1}, \hat{x}_{n+2})$
- for this machine learning model with multivariate output have to be used

Multivariate LM

Let's now implement a LM model with multivariate output. We consider 1 year (12 obs) as output, and 2 years as inputs (24 obs).

```
data=as.vector(sanfran_train)[1:36]
for (i in 1:(length(as.vector(sanfran_train))-36)){
  data=rbind(data,as.vector(sanfran_train)[(i+1):(i+36)])
}
data=data.frame(data)
for (i in 1:36) colnames(data)[i]=paste('x',i,sep='')
fitLM=lm(cbind(x36,x35,x34,x33,x32,x31,x30,x29,x28,x27,x26,x25)~.,data=data)
```

Multivariate LM

Sequential forecasting by year

```
pred=rep(NULL,36)
newdata=data.frame(matrix(tail(sanfran_train,24),1,24))
for (i in 1:24) colnames(newdata)[i]=paste('x',i,sep='')
pred[1:12]=predict(fitLM,newdata=newdata)
for (j in 1:2){
  newdata=data.frame(c(newdata[-(1:12)],pred[(12*(j-1)+1):(12*j)]))
  for (i in 1:24) colnames(newdata)[i]=paste('x',i,sep='')
  pred[(12*(j)+1):(12*(j+1))]=predict(fitLM,newdata=newdata)
}
prevMLM=ts(pred,start=c(1964,1),end=c(1966,12),frequency = 12)
```

Multivariate LM

Forecasting results

```
print(sqrt(mean((prevMLM-sanfran_test)^2)))
```

```
## [1] 26.66568
```

```
plot(sanfran_test,xlim=c(1964,1967),ylim=c(0,120))
lines(sanfran_test,lty=2)
lines(prevMLM,col=6)
```

Multivariate LM

We can change by sequentially forecasting months by months and not year by year

```
pred=rep(NULL,36)
newdata=data.frame(matrix(tail(sanfran_train,24),1,24))
for (i in 1:24) colnames(newdata)[i]=paste('x',i,sep='')
pred[1:12]=predict(fitLM,newdata=newdata)
for (j in 1:35){
  newdata=data.frame(c(newdata[-1],pred[j]))
  for (i in 1:24) colnames(newdata)[i]=paste('x',i,sep='')
  pred[j+1]=predict(fitLM,newdata=newdata)[1]
}
prevMLM=ts(pred,start=c(1964,1),end=c(1966,12),frequency = 12)
```

Multivariate LM

Forecasting results

```
print(sqrt(mean((prevMLM-sanfran_test)^2)))
```

```
## [1] 33.72657
```

```
plot(sanfran_test,xlim=c(1964,1967),ylim=c(0,120))
lines(sanfran_test,lty=2)
lines(prevMLM,col=6)
```