4th Assignment: Wind Power modelling: Non-linear regression, Markov-Switching and Adaptive CTSM

David Ribberholt Ipsen (s164522), Kasper .... (s....), Michelle ... (s....)

December 2021

Contents

1. Introduction

This report focuses on modelling the wind power production of the wind farm in Klim. It will focus solely on the 1h ahead prediction horizon, but the models can easily be generalized for different horizons (and tied beneficially together using temporal hierarchies). This report sets out to apply many different models (for the sake of learning) instead of going very deep, comprehensivly into few models.

2. Explorative Data Analysis

First, let's load the data and do some explorative data analysis. The variable of particular interest is the power p.

Note, the strong time serial dependance in p. Particularly the AR(1)-term carries a lot of information.

Now let's analyze p in relation to the explanatory variables.

Plot Power vs (normalized) Ttime-of-Year

In order to utilize the time-of-year variable in a (linear and non-linear) regression setting, I normalize the time-of-year variable by

$$toy_{norm} = cos(2\pi \cdot \frac{toy}{365})$$

Thereby linking December and January close together at a value near 1, while in the summer time the value is close to -1.

```
# Define normalized time of year: Ratio between 0 and 1 indicating the extent of winter (end of year)
D$ntoy = cos(2*pi*D$toy/365)

plot(D$t, D$ntoy) # Utilise the time-of-year variable as follows, decsribing the degree of winter.
plot(D$ntoy, D$p)

loessfit_ntoy = loess(p ~ ntoy, dat=D, span=0.3, family='gaussian', degree=2)
lines(D$ntoy[!is.na(D$p)], predict(loessfit_ntoy, ntoy=list(D$ntoy)), col=3, lwd=2)
```

In order to be able to interpret the density of the points, I've added local polynomial regression smoothing using a Gaussian kernel of size 30 %. The figures indicates higher power productions in winter time.

Power vs. Wind Speed

Similarly, let's scatter the data and make a non-parametric fit for the forecasted wind speed (1h ahead).

```
library(ggplot2); library(RColorBrewer)
rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))
r <- rf(32)

mask = !is.na(D$Ws1) & !is.na(D$p) & D$Ws1<25 # Mask non-NaN

plot(D$Ws1[mask], D$p[mask])
points(D$Ws1[mask], predict(loess(p ~ Ws1, dat=D[mask,], span=0.3, family='gaussian', degree=2), Ws1=lis
```

```
#p <- ggplot(D, aes(Ws1,p))
# Add colouring and change bins
#library(RColorBrewer)
#h3 <- p + stat_bin2d(bins=100) + scale_fill_gradientn(colours=r) + ggtitle(sprintf("Wind speed forecas
#h3
```

There seem to be somewhat of an logistic effect of wind speed on the power, i.e. the power increases exponentially in the beginning but becomes rather saturated with more and more wind.

Power vs Wind Direction

```
# Wind direction
p <- ggplot(D, aes(Wd1,p))
h4 <- p + stat_bin2d(bins=100) + scale_fill_gradientn(colours=rf(32)) + ggtitle(sprintf("Wind direction
h4

mask = !is.na(D$Wd1) & !is.na(D$p)# Mask non-NaN
plot(D$Wd1[mask], D$p[mask])
loessfit_Wd1 = loess(p ~ Wd1, dat=D[mask,], span=0.3, family='gaussian', degree=2)
points(D$Wd1[mask], predict(loessfit_Wd1, Ws1=list(D$Wd1[mask])), col=3, lwd=0.7)
```

It seems to be favourable for the wind power production if the wind direction is around 200 or 300 degrees.

These findings will be utilised in the modelling section later.

2. Simple models

For benchmarking, some simple models are fitted.

```
# Mean Model
fit1 = lm(p ~ 1, data=D); summary(fit1); logLik(fit1)

# AR(1)
fit2 = lm(p[2:n] ~ 0 + p[1:(n-1)], data=D); summary(fit2); logLik(fit2)

# Linear Regression
fit3a = lm(p ~ Ws1 + Wd1 + T1 + ntoy, data=D) ; summary(fit3a); logLik(fit3a) # Chosen var
fit3b = lm(p ~ ., data=D[,-1]) ; summary(fit3b); logLik(fit3b) # All var

# AR(1)X
fit4a = lm(p ~ Ws1 + Wd1 + T1 + ntoy + D$p[-n], data=D[2:n,-1]); summary(fit4a); logLik(fit4a) # Chosen
fit4b = lm(p ~ . + D$p[-n], data=D[2:n,-1]); summary(fit4b); logLik(fit4b) # All var
```

Clearly, it has a *very* large improvement in log-likelihood when an AR(1)-term is present in the model.

Let's try expanding the model with the non-linearities as found in the explorative data analysis. This means:

1. The wind speed seem to affect the power in a s-curve, i.e. use a logistic function on the wind speed, i.e. $p = ... + b \cdot \frac{e^{a \cdot WindSpeed}}{1+e^{a \cdot WindSpeed}}$

2. The normalized time-of-year had a non-linear affect on the power. Use the loess-fit linearly to predict power, i.e. $p = ... + c \cdot \hat{f}(ntoy)$, where $\hat{f}(ntoy)$ is the local regression smoother of power as a function of normalized time-of-year.

3. For the wind direction, utilise the loess-fit the same way as for normalized time-of-year.

Let's start with the logistic-fit in order to be able to see the benefit of such.

```
# Initially test nls()
#fit5 = nls(p ~ a0 + a1*Ws1 + a2*Wd1 + a3*T1 + a4*ntoy, start = list(a0=0, a1=0.5, a2=0.5, a3=0.1, a4=1,
  # Nice, exact same results as for lm()

# OK let's actually do some non-linear fit: Using findings from the descriptive section
fit6 = nls(p ~ a0  + a11*logit(a1*Ws1) + a2*Wd1 + a3*T1 + a4*ntoy,
           start = list(a0=0, a1=0.1, a11=10, a2=0.5, a3=0.1, a4=1),
           data=D); summary(fit6); logLik(fit6);
```

I.e. only a slight improvement in likelihood from the simple linear regression model (fit3a).

Now let's expand the models further with the non-linear fit of wind direction and time of year.

```
# Let's use the non-parametric fit of Wind Direction
fit7 = nls(p ~ a0  + a11*logit(a1*Ws1) + a2*predict(loessfit_Wd1, Wd1) + a3*T1 + a4*predict(loessfit_nt
           start = list(a0=0, a1=0.1, a11=10, a2=0.5, a3=0.1, a4=0.1),
           data=D); summary(fit7); logLik(fit7);
```

Again, a small improvement in likelihood. However, this complex with three non-linearities is still beaten by the AR(1)-model. So before moving on to another class of models, let's try adding the AR(1)-term.

```
plag1 = D$p[1:(n-1)]
# Let's use the non-parametric fit of Wind Direction
fit8 = nls(p ~ a0  + a11*logit(a1*Ws1) + a2*predict(loessfit_Wd1, Wd1) + a3*T1 + a4*predict(loessfit_nt
           start = list(a0=0, a1=0.1, a11=10, a2=0.5, a3=0.1, a4=0.1, a5=1),
           data=D[2:n,]); summary(fit8); logLik(fit8);
```

We now see a large improvement in likelihood. Let's compare it with the corresponding model AR(1)X model with the same variables but without non-linearities:

```
AIC(fit4a)
AIC(fit8)
```

I.e. a significant improvement.

3. Autoregressive Markov-Switching Model

Looking at the power over time, it is possible to identify some different states, e.g. periods of constant power production, periods of high volatility etc. This indicates that a state space model might be suitable for the problem. As inspired by (**A Markov-Switching model for building occupant activity estimation**, Wolf, Møller et. al., 2018), I will try a generalization of the Hidden Markov Model where the states determine the autoregressive coefficients on the observations and linear coefficients on the input data specific to each state, i.e. state-specific AR- and linear coefficients. I will utilise the implementation found on **https://cran.r-project.org/web/packages/MSwM/**.

In the following section, I will for simplicity only use Ws1, Wd1, T1 and ntoy as input variables (corresponding to fit3a). Additionally, I will only use an AR(1)-term as was found to be of far greatest importance as seen in the pacf. Instead I will vary the number of states. Note that the models are nested, so for model comparison likelihood-ratio test is appropriate.

```
#install.packages('MSwM')
par(mfrow = c(1,1))
library(parallel)
library(MSwM)

# Simple Linear Regression
fit2b = lm(p ~ Ws1 + Wd1 + T1 + ntoy, data=D) ; summary(fit2b); logLik(fit2b)

# Fit MSM
fit_msm2 = msmFit(fit2b, k=2, p=1, sw=rep(TRUE,7), control = list(parallel = TRUE))
summary(fit_msm2)
#plotProb(fit_msm, which=1)
#plotProb(fit_msm, which=2)
#plotProb(fit_msm, which=3)
#plot(msmResid(fit_msm))
#plotReg(fit_msm)

# It's impossible to grasp the fit for the ~40000 observations.
# To assess the fit, fit and visualise the same model on a small subset of the data
Dsub = D[1:3000, ]
fit2bsub = lm(p ~ Ws1 + Wd1 + T1 + ntoy, data=Dsub)
fit_msm2_sub = msmFit(fit2bsub, k=2, p=1, sw=rep(TRUE,7), control = list(parallel = TRUE))
plotProb(fit_msm2_sub, which=2)
```

We see a very large improvement in likelihood and thus also AIC when compared to the best model in the previous section.

Note that in order to visualise the states, I've fitted the same model on a subset of the data. The global decoding, i.e. the most probable regime at time step t as seen above, seem to correspond to periods with constant wind power (regime 1) and periods with changing wind power (regime 2). This corresponds well with what we could expect of a state-dependent AR-coefficient: That one regime would correspond to keeping things constant.

Now it seems obvious to add another state, hopefully separating dynamics of the wind power when in decline / increasing.

```
# Fit MSM-AR with k=3 states
fit_msm3 = msmFit(fit2b, k=3, p=1, sw=rep(TRUE,7), control = list(parallel = TRUE))
summary(fit_msm3)


# Plot subset
fit_msm3_sub = msmFit(fit2bsub, k=3, p=1, sw=rep(TRUE,7), control = list(parallel = TRUE))
for (i in 2:4){
  plotProb(fit_msm3_sub, which=i)
}
```

Again, a large improvement in likelihood with the extra state. Is the likelihood gain worth the extra parameters? I will assess that using likelihood ratio test in the end of this section.

Now regime 1 corresponds to a constant period, while regime 2 and 3 seem to correspond to periods high and low power production respectively.

Let's try add another state

```
# Fit MSM-AR for k=4 states
fit_msm4 = msmFit(fit2b, k=4, p=1, sw=rep(TRUE,7), control = list(parallel = FALSE))


# Plot subset
fit_msm4_sub = msmFit(fit2bsub, k=4, p=1, sw=rep(TRUE,7), control = list(parallel = FALSE))
for (i in 2:5){
  plotProb(fit_msm4_sub, which=i)
}
```

However, this kills the kernel on my computer every time. This might be due to running out of memory (https://www.r-bloggers.com/2019/02/switching-regressions-cluster-time-series-data-and-understand-your-development/). To solve this issue, one might subset the data. However, this makes the likelihood values incompatible. For this assignment, a 3-state Markov-Switching AR model is sufficient.

Likelihood ratio tests

```
L1 = logLik(fit4a)[1]; df1 = 7 # 1-state MSM-AR / AR(1)X
L2 = -42051.17; df2 = 7*2+(2*2)-2; # 2-state
L3 = -35995.38; df3 = 7*(3*3)-3; # 3-state

sprintf('Likelihood ratio test of 2-state MSM-AR vs. non-linear 1-state regression: p-value = %.3f', 1-p

sprintf('Likelihood ratio test of 2-state MSM-AR vs. non-linear 1-state regression: p-value = %.3f', 1-p

sprintf('AIC of 3-state MSM-AR:  %2.f', 72026.76)
sprintf('AIC of best benchmark model:  %3.f', AIC(fit8))
```

I.e. the Markov-Switching model is a much better fit - despite the extra parameters.

Finally, let's illustrate the last 200 (in-)samples of the fit:

```
res = msmResid(fit_msm3)
n1 = length(res)
fittedp = na.omit(D)$p + res[4:(n1-4)]

plot(na.omit(D)$p[(n1-200):n1], type='l', ylim=c(-3,20), ylab="Power")
lines(fittedp[(n1-200):n1], type='l', col=3)
legend("topleft",legend=c('True power', 'Fitted power'), lty=c(1,1), col=c(1,3))

par(mfrow=c(2,1))
acf(res)
pacf(res)
```

We see a reasonable fit, although there is a high variance in the predictions even resulting in negative (!) predictions of power. Also, there is still plenty more information in the residuals that should be modelled.

evt. Discrete State Space Form / Extended Kalman Filter (with adaptivity) (MANGLER)

$X_{t+1} = ...\theta_t X_t ...$ (unobserved) $\theta_{t+1} = \theta_t + ... + \epsilon_{t+1}$

$p_t = ...X + e_t$ (observed)

Continous-Descrete State Space Model (with adaptivity) (MANGLER)

```
#install.packages('pkgbuild')
#install.packages("ctsmr", repo = "http://ctsm.info/repo/dev")
library('ctsmr')
ctsm1 <- function(data){
  # Remove NaN
  mask = !is.na(data$Ws1) & !is.na(data$p)
  data = data[mask,]

  # Generate a new object of class ctsm
  model = ctsm()

  # Add a system equation and thereby also a state
  model$addSystem(dX ~ (phi*Ws1)*dt + exp(p11)*dw1)
  model$addSystem(dphi ~ 0 * dt + exp(p12)*dw2)

  # Set the names of the inputs
  model$addInput(Ws1)
  # Set the observation equation:
  model$addObs(p ~ X)
  # Set the variance of the measurement error
  model$setVariance(p ~ exp(e11))
  ##----------------------------------------------------------------
  # Set the initial value (for the optimization) of the value of the state at the starting time point
  model$setParameter(X = c(init = 5, lb = 0, ub = 25))
  model$setParameter(phi = c(init = 0, lb = -5, ub = 5))
  ##----------------------------------------------------------------
  # Set the initial value for the optimization
  #model$setParameter(Ci = c(init = 1, lb = 1E-5, ub = 1E5))
  #model$setParameter(Cm = c(init = 1000, lb = 1E-5, ub = 1E5))
  #model$setParameter(Ria = c(init = 20, lb = 1E-4, ub = 1E5))
  #model$setParameter(Rim = c(init = 20, lb = 1E-4, ub = 1E5))
  #model$setParameter(Aw = c(init = 6, lb = 1E-2, ub = 7.5+4.8+5))
  model$setParameter(p11 = c(init = 1, lb = -30, ub = 10))
  model$setParameter(p12 = c(init = 1, lb = -30, ub = 10))
  model$setParameter(e11 = c(init = 1, lb = -50, ub = 10))
  #model$setParameter(a1 = c(init = 0.2, lb = -50, ub = 100))
  #model$setParameter(a2 = c(init = 4, lb = -500, ub = 1000))
  #model$setParameter(a3 = c(init = 4, lb = -500, ub = 1000))
  #model$setParameter(a4 = c(init = 4, lb = -500, ub = 1000))
  #model$setParameter(a5 = c(init = 4, lb = -500, ub = 1000))
```

```
  ##----------------------------------------------------------------

  # Run the parameter optimization
  fit = model$estimate(data,firstorder = TRUE)
  return(fit)
}
Dsub = D[1:400,]
fit = ctsm1(Dsub)
summary(fit, extended = F)
fit$loglik
```

Kræver vel-installeret compiler.