Blog

mikmak

13/12/2016

## Loading required package: DBI

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':  
##   
## date

# Mit BigData und AI die Pflanzen bewässern

## Idee

Ein Algorithmus muss vorhersagen können wieviel Wasser meine Balkonpflanzen in den nächsten 24 Stunden brauchen werden.

## Warum BigData und AI und nicht handelsübliches Steuergerät

Warum so komplizierte Anlage bauen und nicht einfach eine handelsübliches Gerät kaufen, der mit einem Feuchtigkeitssensor ausgestattet ist. Hier sind einige Argumente:

* eine sensorgesteuerte Anlage hat meistens keine Internet Anbindung. D.h. wenn ich längere Zeit weg bin und es passiert etwas Unvorgesehnbares, kann ich nicht wissen ob die Anlage noch mit Strom versorgt wird (meine Anlage ist an eine Batterie angeschlossen welche von einem Solar Pannel aufgeladen wird), ob der Feuchtigkeitssensor noch richtig positioniert ist und nicht vom Wind umgeworfen wurde und so falsche Daten an die Steuereinheit sendet. (Es ist mir früher schon passiert).
* Handelsübliche Anlagen sind so programmiert, dass diese zu einem bestimmten Zeitpunkt die Sensortwerte auswerten und entscheiden ob die Pumpe gestartet werden soll oder nicht. Z.B. um 17 Uhr meldet der Sensor, dass die Erde trocken ist und es Wasser braucht. Um 17:05 fängt es zu regnen an. So hat die Anlage umsonst das Wasser und Strom verbraucht.
* Skalleneffekte. Eine an die Cloud angeschlossene Anlage benötigt keinen hochqualitativen (und teueren) Sensor. Beim Bau jeder weieter Anlage sinken die Hardware- und somit die Gesamtkosten.

## Lösung im Detail

### Algorithm

Es gibt grundsätzlich zwei Arten von Algorythmen:

* Überwachtes Lernen: hier verwendet man historische Daten um den Rechner zu "trainieren" anhand der Input Daten den richtigen Output zu berechnen. z.B. Regressionsanalysen, Neueronale Netze
* Unüberwachtes Lernen: hier lässt man den Rechner die Daten analysieren und gewisse gesetzesmässigkeiten oder Muster zu erkennen. Beispiele sind: Korrelationsanalyse, Clustering. Für mein Vorhaben eignen sich die Algorithmen für überwachtes Lernen am Besten. Dabei habe ich dem Tipp eines Stanford Professors gefolgt und habe mit dem einfachsten von Allen begonnen, mit der linearer Regression.

### Lineare Regression

Alle Algorithmen vom Typ Überwachtes Lernen kann man in Form einer mathematischer Funktion darstellen, wo X eine oder mehrere Input Variablen sind und y ist die sogenannte Target Variable, also der Wert, den wir voraussagen möchten.

Die mathematische Funktion für die lineare Regression sieht so aus:

Auch hier steht X für Input Variablen, in meinem Fall sind es: Höchsttemperatur in C in den letzten 24 Stunden, durchschnittlicher Luftdruck in den letzten 24 Stunden usw. y ist die Wassermenge in ml, die eine Pflanze aktuell benötigt. Um y zu berechnen, muss man die Input Parameter X mit den Modell Parametern multiplizieren und die Resultate aufsummieren.

Tönt einfach, ist es auch, mann muss nur passende Parameter finden. Man nennt es das Modell trainieren. Dazu verwende ich die "Methode der kleinsten Quadrate".

1. Dabei vergibt man allen Parametern einen Initialwert (i.d.R. 0.1) und
2. man berechnet den Zielparameter für die historischen Daten. Die Summe der Differenzen zwischen dem errechneten Wert und dem wahren historischen Wert im Quadrat (deswegen Methode der kleinsten Quadrate) ist der korrektur Faktor.
3. Danach korrigiert man die Parameter um den Korrektur Faktor. Damit nicht alle Thetas um den selben Wert korrigiert werden, multipliziert man den Korrekturfaktor mit dem jeweiligen Wert . Anschliessend
4. wiederholt man die Schritte 1-3 x-fach bis der Korrekturfaktor sich nicht mehr wesentlich ändert. In meinem Fall haben 300 Iterationen gereicht.

OK, es ist doch eine zwar richtige, aber doch vereinfachte Erklärung der Methode der kleinsten Quadrate. Wer mehr wissen möchte, empfehle ich den Wikipedia Artikel.

Viele Programmiersprachen der 4. Generation (4GL) entwickelt für die statistische Datenverarbeitung wie R, MATLAB, SPSS usw. bieten die Funktionen zur optimierung der linearen Regressionen bereits in der Standardauslieferung.

In R verwendet man dazu die Funktion

lm(y ~ x + 1)

### Daten

Gemäss einer Studie von Stanford University sind die Trainings Daten wichtiger, als die Wahl des Algorythumus.

Die Daten kann man entweder selber sammeln, kaufen oder die Daten der öffentlichen Insitutionen verwenden (Stichwort "Open Data").

Für meinen Fall verwende ich - öffentliche Meteo Daten von der MeteoSuisse auf <http://www1.ncdc.noaa.gov/pub/data/noaa/> und - den von mir gesammelten Bewässerungslog (wie lange ich an welchem Tag die Pflanzen mit der Gardena Anlage bewässert habe)

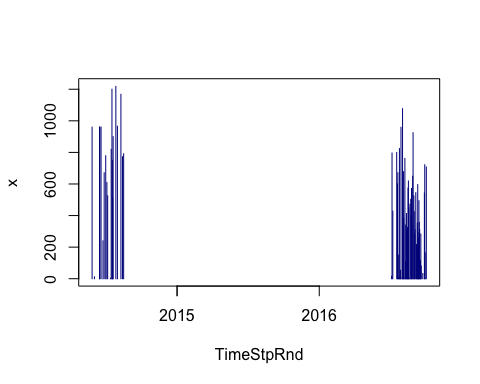
Bewässerungslog ist also die Variable

## Warning in .local(conn, statement, ...): unrecognized MySQL field type 7 in  
## column 2 imported as character

## Warning: Closing open result sets

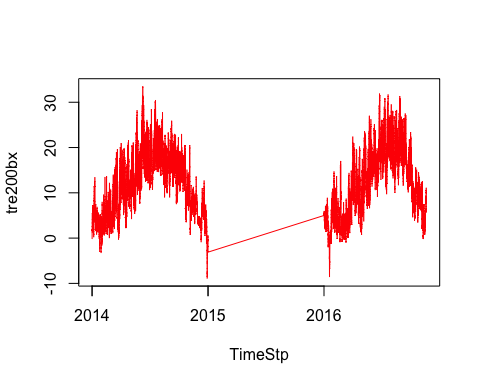
## [1] TRUE

e\_pumpLog[3] <- as.POSIXct(e\_pumpLog[,3] , origin="1970-01-01")  
t\_pumpLog <- e\_pumpLog[order(e\_pumpLog$TmStp),]  
  
t\_pumpLog <- cbind(t\_pumpLog, round(t\_pumpLog$TmStp, units = "hours"))  
names(t\_pumpLog)[ncol(t\_pumpLog)] <- "TimeStpRnd"  
  
  
  
t\_pumpLog <- cbind(t\_pumpLog, as.Date(t\_pumpLog$TimeStpRnd))  
names(t\_pumpLog)[ncol(t\_pumpLog)] <- "DateRnd"  
  
# Fill missing logs with zeroes in 2014 (For 2016 there are records if no pumping was neccesseary)  
minDate <- min(t\_pumpLog$DateRnd)  
maxDate <- max(t\_pumpLog$DateRnd[year(t\_pumpLog$DateRnd) == 2014])  
missingValues <- data.frame("DateRnd" = seq(from = as.Date(minDate), to = as.Date(maxDate), by = "days")  
 , "msgv1" = 0)  
t\_pumpLog <- rbind(t\_pumpLog[, c("DateRnd", "msgv1")], missingValues)  
  
# Calculate the sum by day and set pumping time to 07:00 PM  
t\_pumpLog <- aggregate(as.integer(t\_pumpLog$msgv1), by=list(DateRnd = t\_pumpLog$DateRnd), FUN=sum)  
t\_pumpLog <- cbind(t\_pumpLog, paste(t\_pumpLog$DateRnd, "19:00:00", sep = " "))  
names(t\_pumpLog)[ncol(t\_pumpLog)] <- "TimeStpRnd"  
t\_pumpLog$TimeStpRnd <- parse\_date\_time(t\_pumpLog$TimeStpRnd, c('ymd HMS'))  
  
p\_pumpLog <- t\_pumpLog[year(t\_pumpLog$DateRnd) == 2014,]  
p\_pumpLog <- t\_pumpLog[!t\_pumpLog$x == 0,]  
p\_pumpLog <- p\_pumpLog[!p\_pumpLog$x == 480,]  
#p\_pumpLog <- t\_pumpLog  
  
plot(p\_pumpLog[, c("TimeStpRnd", "x")], type = "h", col = "darkblue")



#### Wetterdaten einlesen und verarbeiten

e\_weatherData2014 <- read.csv("~/CloudStation/private/myProjects/plant.watering.PredictiveModel.R/data/shopdwhdata\_2YH\_WAE.csv", sep=";", stringsAsFactors=FALSE)  
e\_weatherData2016 <- read.csv("~/CloudStation/private/myProjects/plant.watering.PredictiveModel.R/data/shopdwhdata\_0YH\_WAE.csv", sep=";", stringsAsFactors=FALSE)  
  
t\_weatherData <- rbind(e\_weatherData2014, e\_weatherData2016)  
t\_weatherData <- cbind(t\_weatherData, paste(t\_weatherData$date, t\_weatherData$time, sep = " "))  
names(t\_weatherData)[ncol(t\_weatherData)] <- "TimeStp"  
  
t\_weatherData$TimeStp <- parse\_date\_time(t\_weatherData$TimeStp, c('dmy HM'))  
  
p\_weatherData <- t\_weatherData  
plot(p\_weatherData[, c("TimeStp", "tre200bx")], type = "l", col="red")



#### Datenset vorbereiten

* Input
  + Wetter vor 24 Stunden
  + Wetter vor 48 Stunden
  + Bewesserungstageszeit (Abend/Morgen, Nacht, Tag)
  + Wetterveränderung seit 24 Stunden (Temperatur, Luftdruck, Niederschlagsmenge)
* Output
  + Wasserkonsum in den nächsten 24 Stunden

# Create DataSet --------------------------------------------  
t\_dataSet <- p\_pumpLog  
lv\_pumpDurationInLast24h <- apply(t\_dataSet, 1, function(x) {  
 date <- as.Date(x["DateRnd"]) - days(1)  
  
 data <- p\_pumpLog[p\_pumpLog$DateRnd == date,"x"]  
   
 if(length(data) == 0) data = 0 else data = data  
   
 return(data)  
})  
  
t\_dataSet <- cbind(t\_dataSet, unlist(lv\_pumpDurationInLast24h))  
names(t\_dataSet)[ncol(t\_dataSet)] <- "PumpDurationLast24h"  
  
  
# a\_weatherData <- cbind(p\_weatherData, p\_weatherData$TimeStp + hours(0))  
a\_weatherData <- cbind(p\_weatherData, p\_weatherData$TimeStp)  
names(a\_weatherData)[ncol(a\_weatherData)] <- "TimeStp-24"  
a\_weatherData <- cbind(a\_weatherData, p\_weatherData$TimeStp + hours(24))  
names(a\_weatherData)[ncol(a\_weatherData)] <- "TimeStp-27"  
a\_weatherData <- cbind(a\_weatherData, p\_weatherData$TimeStp + hours(36))  
names(a\_weatherData)[ncol(a\_weatherData)] <- "TimeStp-30"  
a\_weatherData <- cbind(a\_weatherData, p\_weatherData$TimeStp + hours(42))  
names(a\_weatherData)[ncol(a\_weatherData)] <- "TimeStp-36"  
 a\_weatherData <- cbind(a\_weatherData, p\_weatherData$TimeStp + hours(48))  
 names(a\_weatherData)[ncol(a\_weatherData)] <- "TimeStp-42"  
   
 # Get Sums and Means  
   
 a\_sumsANDmeans <- apply(t\_dataSet, 1, function(x) {  
 lv\_date <- as.POSIXct(x["TimeStpRnd"]) - days(1)  
 lv\_date\_24h <- as.POSIXct(x["TimeStpRnd"]) - days(2)  
  
 lv\_d <- a\_weatherData[a\_weatherData$TimeStp >= lv\_date\_24h & a\_weatherData$TimeStp <= lv\_date,] # Get weather Data for last 48 hours  
   
 lv\_y <- mean(lv\_d$tre200b0)  
 lv\_y <- cbind(lv\_y, mean(lv\_d$tre200bn))  
 lv\_y <- cbind(lv\_y, mean(lv\_d$tre200bx))  
 lv\_y <- cbind(lv\_y, mean(lv\_d$ure200b0))  
 lv\_y <- cbind(lv\_y, sum(lv\_d$rre150b0))  
 lv\_y <- cbind(lv\_y, mean(lv\_d$prestab0))  
 lv\_y <- cbind(lv\_y, 1) #Add column with ones to replace it in the next step with dates  
 lv\_y[ncol(lv\_y)] <- as.POSIXct(lv\_date, origin="1970-01-01", tz="GMT" )  
  
 return(lv\_y)  
})  
 a\_sumsANDmeans <- as.data.frame(t(unlist(a\_sumsANDmeans)))  
 a\_sumsANDmeans[,ncol(a\_sumsANDmeans)] <- as.POSIXct(a\_sumsANDmeans[,ncol(a\_sumsANDmeans)], origin = "1970-01-01", tz = "GMT") + hours(2)  
 names(a\_sumsANDmeans) <- c("tre200b0Mean", "tre200bnMean", "tre200bxMean", "ure200b0Mean", "rre150b0Sum", "prestab0Mean", "V7")  
  
#t\_dataSet <- merge(t\_dataSet, a\_weatherData, by.y = "TimeStp-24", by.x = "TimeStpRnd", suffixes = c(".m",".24"))  
t\_dataSet <- merge(t\_dataSet, a\_weatherData, by.y = "TimeStp-24", by.x = "TimeStpRnd")  
t\_dataSet <- merge(t\_dataSet, a\_weatherData, by.y = "TimeStp-27", by.x = "TimeStpRnd")  
  
t\_dataSet <- merge(x = t\_dataSet, y = a\_sumsANDmeans, by.y = "V7", by.x = "TimeStp-24", all.x = TRUE, all.y = TRUE)  
  
# Drop unneccessairy variables --------------------------------------------  
  
#columnsToDrop <- names(t\_dataSet) %in% grep("[.y]$", names(t\_dataSet), value=TRUE)  
#t\_dataSet <- t\_dataSet[,!columnsToDrop]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% c("TimeStpRnd"))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% grep("^time\*", names(t\_dataSet), value=TRUE))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% grep("^date\*", names(t\_dataSet), value=TRUE))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% grep("^stn\*", names(t\_dataSet), value=TRUE))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% grep("^fu\*", names(t\_dataSet), value=TRUE))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% grep("^TimeStp\*", names(t\_dataSet), value=TRUE))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% grep("^dkl\*", names(t\_dataSet), value=TRUE))]  
t\_dataSet <- t\_dataSet[, !(names(t\_dataSet) %in% c("DateRnd"))]  
  
# Prepare DataSet for machine learning --------------------------------------------  
t\_dataSet <- na.omit(t\_dataSet)  
  
a\_dataSet <- data.matrix(t\_dataSet)  
a\_dataSet <- as.data.frame(a\_dataSet)  
  
a\_dataSet <- polyFeatures(a\_dataSet,3)  
a\_dataSet <- a\_dataSet[, !(names(t\_dataSet) %in% grep("^x\*2", names(a\_dataSet), value=TRUE))] #No X's in dataset!!!  
  
 # Train models --------------------------------------------  
 model <- lm(`x` ~ . - `x\*3` - `x\*3.1` + 1, data = a\_dataSet)  
 #model <- lm(`x` ~ prestab0Mean +1, data = a\_dataSet)  
  
summary(model)

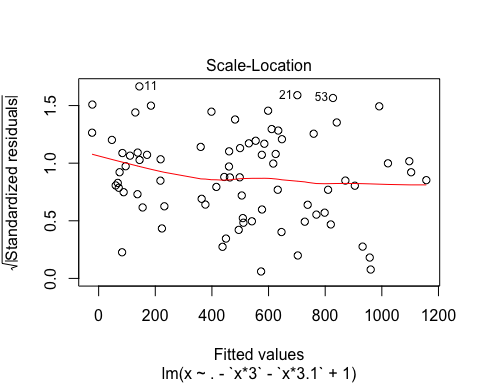
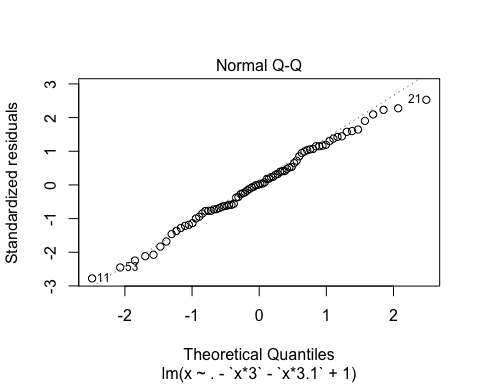
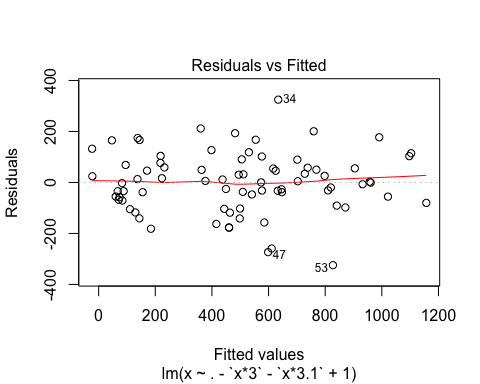
##   
## Call:  
## lm(formula = x ~ . - `x\*3` - `x\*3.1` + 1, data = a\_dataSet)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -324.43 -65.47 1.57 66.16 324.77   
##   
## Coefficients: (3 not defined because of singularities)  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) -8.202e+06 9.174e+06 -0.894 0.390  
## PumpDurationLast24h -3.480e+00 2.423e+00 -1.436 0.179  
## tre200b0.x 7.540e+03 2.055e+04 0.367 0.721  
## tre200bn.x -1.285e+03 1.033e+04 -0.124 0.903  
## tre200bx.x -8.261e+03 1.099e+04 -0.751 0.468  
## ure200b0.x -1.912e+01 5.343e+02 -0.036 0.972  
## rre150b0.x -1.032e+03 1.545e+03 -0.668 0.518  
## sre000b0.x -7.470e+00 6.785e+01 -0.110 0.914  
## gre000b0.x 4.516e+00 2.912e+01 0.155 0.880  
## prestab0.x -2.516e+03 1.212e+04 -0.208 0.839  
## tre200b0.y 3.221e+03 3.949e+04 0.082 0.936  
## tre200bn.y 4.524e+03 2.075e+04 0.218 0.831  
## tre200bx.y -6.484e+03 2.054e+04 -0.316 0.758  
## ure200b0.y -8.977e+00 7.516e+02 -0.012 0.991  
## rre150b0.y -9.822e+02 1.673e+03 -0.587 0.569  
## sre000b0.y -6.592e+00 6.351e+01 -0.104 0.919  
## gre000b0.y 2.729e+01 4.463e+01 0.611 0.553  
## prestab0.y 5.473e+03 1.868e+04 0.293 0.775  
## tre200b0Mean 1.681e+05 2.189e+05 0.768 0.459  
## tre200bnMean -1.175e+05 1.251e+05 -0.940 0.368  
## tre200bxMean -5.557e+04 1.008e+05 -0.551 0.592  
## ure200b0Mean -4.369e+03 4.435e+03 -0.985 0.346  
## rre150b0Sum 1.933e+01 7.825e+01 0.247 0.809  
## prestab0Mean 1.438e+04 2.025e+04 0.710 0.492  
## `PumpDurationLast24h\*3` 6.368e-03 5.797e-03 1.098 0.295  
## `PumpDurationLast24h\*3.1` -3.066e-06 3.744e-06 -0.819 0.430  
## `tre200b0.x\*3` -3.456e+02 1.061e+03 -0.326 0.751  
## `tre200b0.x\*3.1` 5.649e+00 1.809e+01 0.312 0.761  
## `tre200bn.x\*3` 5.560e+01 5.600e+02 0.099 0.923  
## `tre200bn.x\*3.1` -1.094e+00 1.008e+01 -0.108 0.916  
## `tre200bx.x\*3` 3.891e+02 5.393e+02 0.721 0.486  
## `tre200bx.x\*3.1` -6.114e+00 8.713e+00 -0.702 0.497  
## `ure200b0.x\*3` -6.199e-01 8.284e+00 -0.075 0.942  
## `ure200b0.x\*3.1` 6.757e-03 4.216e-02 0.160 0.876  
## `rre150b0.x\*3` 5.922e+02 8.398e+02 0.705 0.495  
## `rre150b0.x\*3.1` -8.495e+01 1.166e+02 -0.729 0.481  
## `sre000b0.x\*3` -1.010e+00 3.414e+00 -0.296 0.773  
## `sre000b0.x\*3.1` 1.692e-02 4.108e-02 0.412 0.688  
## `gre000b0.x\*3` -6.962e-02 8.068e-01 -0.086 0.933  
## `gre000b0.x\*3.1` -3.217e-04 6.483e-03 -0.050 0.961  
## `prestab0.x\*3` 1.310e+00 6.304e+00 0.208 0.839  
## `prestab0.x\*3.1` NA NA NA NA  
## `tre200b0.y\*3` -1.354e+02 1.994e+03 -0.068 0.947  
## `tre200b0.y\*3.1` 1.960e+00 3.309e+01 0.059 0.954  
## `tre200bn.y\*3` -2.089e+02 1.110e+03 -0.188 0.854  
## `tre200bn.y\*3.1` 3.130e+00 1.941e+01 0.161 0.875  
## `tre200bx.y\*3` 2.782e+02 9.794e+02 0.284 0.782  
## `tre200bx.y\*3.1` -3.896e+00 1.529e+01 -0.255 0.804  
## `ure200b0.y\*3` -5.766e-01 1.035e+01 -0.056 0.957  
## `ure200b0.y\*3.1` 6.541e-03 4.676e-02 0.140 0.891  
## `rre150b0.y\*3` 3.662e+02 8.844e+02 0.414 0.687  
## `rre150b0.y\*3.1` -2.607e+01 1.181e+02 -0.221 0.829  
## `sre000b0.y\*3` -2.201e-01 2.925e+00 -0.075 0.941  
## `sre000b0.y\*3.1` 1.207e-02 3.792e-02 0.318 0.756  
## `gre000b0.y\*3` -1.502e+00 1.533e+00 -0.980 0.348  
## `gre000b0.y\*3.1` 1.145e-02 1.284e-02 0.892 0.392  
## `prestab0.y\*3` -2.826e+00 9.738e+00 -0.290 0.777  
## `prestab0.y\*3.1` NA NA NA NA  
## `tre200b0Mean\*3` -9.562e+03 1.229e+04 -0.778 0.453  
## `tre200b0Mean\*3.1` 1.752e+02 2.266e+02 0.773 0.456  
## `tre200bnMean\*3` 6.810e+03 7.251e+03 0.939 0.368  
## `tre200bnMean\*3.1` -1.276e+02 1.378e+02 -0.926 0.374  
## `tre200bxMean\*3` 3.129e+03 5.482e+03 0.571 0.580  
## `tre200bxMean\*3.1` -5.652e+01 9.819e+01 -0.576 0.576  
## `ure200b0Mean\*3` 6.027e+01 6.109e+01 0.987 0.345  
## `ure200b0Mean\*3.1` -2.776e-01 2.794e-01 -0.994 0.342  
## `rre150b0Sum\*3` -2.983e+00 6.237e+00 -0.478 0.642  
## `rre150b0Sum\*3.1` 5.920e-02 1.099e-01 0.539 0.601  
## `prestab0Mean\*3` -7.497e+00 1.056e+01 -0.710 0.493  
## `prestab0Mean\*3.1` NA NA NA NA  
##   
## Residual standard error: 311.8 on 11 degrees of freedom  
## Multiple R-squared: 0.8718, Adjusted R-squared: 0.1024   
## F-statistic: 1.133 on 66 and 11 DF, p-value: 0.4386

View(cbind(predict(model,a\_dataSet),a\_dataSet$x))

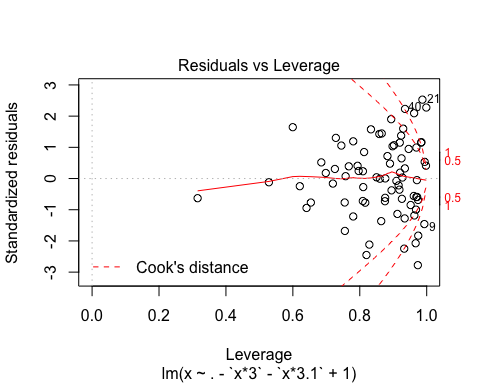
## Warning in predict.lm(model, a\_dataSet): prediction from a rank-deficient  
## fit may be misleading

#### Analyse des Models

plot(model)



## Warning in sqrt(crit \* p \* (1 - hh)/hh): NaNs produced  
  
## Warning in sqrt(crit \* p \* (1 - hh)/hh): NaNs produced



#### Wie die Daten wirklich passen

plot(predict(model,a\_dataSet),a\_dataSet$x,pch = 16, cex = 0.8, col = "blue")

## Warning in predict.lm(model, a\_dataSet): prediction from a rank-deficient  
## fit may be misleading

abline(lm(a\_dataSet$x ~ predict(model)), col="red")

