Basic modelling

To start us off, in the initial blog we discussed elements of the dataset to consider. One of them was temporal, we reasoned that water levels are influenced *over time* by rainfall and temperature. This was why 2 weeks ago we wrote about creating lag features in our dataset. This week in the modelling we must thus work with time series based models.
In my journey of the past 2 weeks I've fallen into a classic trap of machine learning with time series which I will go into depth about. By the end of the blog we have overcome this issue.

First attempts

My first model contained as basic inputs:

- Lags of the outcome variable
- Lags of the predictive variables
- Seasonal indicators (year, quarter, month)

In the previous blog we only worked with the lags of the predictive variables and came up to an Rsquared of .33%, this time we include a few other variables and except to improve our prediction.

Including lags of the outcome variable is a common time-series technique to improve the prediction. In my financial economics class we learned about the concept of 'random-walks', this essentially is a time series that can't be predicted as it includes no pattern. This is, according to the study books, applicable to the day-to-day trading of currency markets. They say tomorrows exchange rates best prediction is todays exchange rate, noone can say any more then that. Random-walking aside, the point is that if we want to predict something for tomorrow, we better include all information on that phenomena we have to date in our model, including lags of our outcome variable.

The seasonal indicators are also added to allow for trends both in years, quarters and months. For example there might be a decline in water levels over time due to global warming, our year indicator can account for this effect.

The next general step is to preprocess our data, what we discussed in depth in the last blog, and set up our model for cross validation. Cross validation is used to hold out a part of the training data to validate the results. Often this is called the 'validation' set for that reason. I used a 10 fold cross validation, with 3 repeats.

Afterwards I set up my gridsearch for the gbm (Gradient Boosting Machine) model we will be using. I initially chose the gbm model as I am familiar with how it works, that honestly really was the only reason. It is smart to try multiple models and compare them later. It is also smart to run a simple linear model as a baseline comparison.

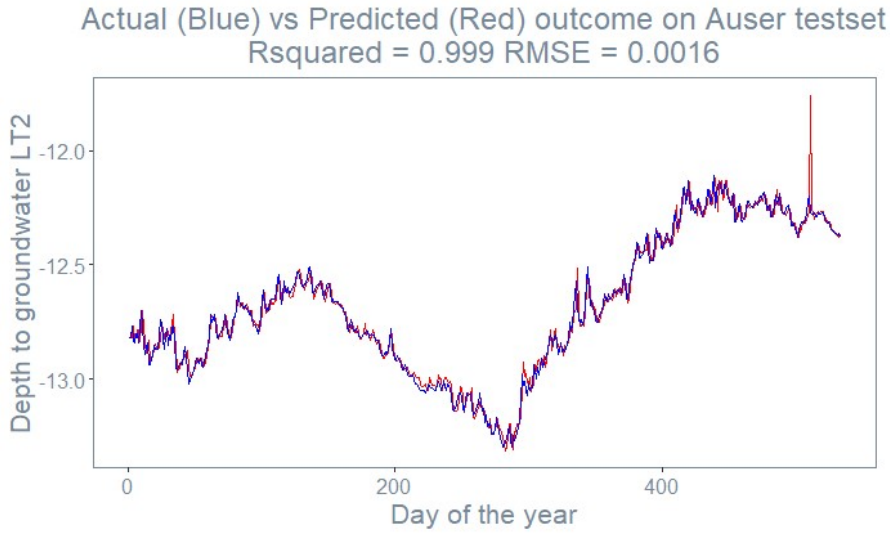
```
create_data_model <- function(data, features, outcome, lag, preprocess
= NULL, include_outcome = NULL, test = NULL){
    # data: This function allows for a generic data object,
    # features: some features that are column names in the data object
    # outcome: name of the otucome column
    # lag: Highest lag to be included in the model
    # preprocess : Whether the data is preprocessed before modelling
(Highly recommended!)
```

```
    # include_outcome:  Indicator if the outcome variable is included
with lags
    # test: Indicator if a test set is to be constructed (default to 1
year)

    data_model <- data[,c(outcome,'Date',features)]
    names(data_model)[which(names(data_model)== outcome)] <- 'outcome'

    # Feature addition
    # Laggs
    if(!is.null(include_outcome)){
        features <- c(features, 'outcome')
    }

    for(i in 1:length(features)){
        for(j in 1:lag){
            data_model$temp <- Lag(data_model[,features[i],+j])
            names(data_model)[which(names(data_model)=='temp')] <-
paste(features[i],j, sep = '_')
        }
    }

    # Inlude seasonality:
    data_model$year <- as.numeric(substr(data_model$Date,7,10))
    data_model$year <- data_model$year - min(data_model$year) + 1
    data_model$month <- as.numeric(substr(data_model$Date,4,5))
    data_model$quarter <- ifelse(data_model$month <= 3,1,
                                 ifelse(data_model$month >=4 &
data_model$month <= 6,2,
                                        ifelse(data_model$month >=7 &
data_model$month <= 9,3,
                                               ifelse(data_model$month
>9,4,NA))))

    # Data cleaning
    data_model <- data_model[complete.cases(data_model),]
# Remove all rows with missing values
    data_model <- data_model[which(data_model$outcome!= 0),]
# Remove all outlier measurements

    if(!is.null(test)){
        data_test <- data_model[which(as.Date(data_model$Date,format =
'%d/%m/%Y')>=as.Date(as.Date(max(data_model$Date), format = '%d/%m
/%Y')-365)),]
        data_model <- data_model[which(!as.Date(data_model$Date,format
= '%d/%m/%Y')>=as.Date(as.Date(max(data_model$Date), format = '%d/%m
/%Y')-365)),]
        data_test$Date <- NULL
    }

    data_model$Date <- NULL
```

```
    # Statistical preprocessing
    if(!is.null(preprocess)){
        temp <- data_model[,-1]
        nzv <- nearZeroVar(data_model)
# excluding variables with very low frequencies
        if(length(nzv)>0){temp <- temp[, -nzv]}
        i <- findCorrelation(cor(temp))
# excluding variables that are highly correlated with others
        if(length(i) > 0) temp <- temp[, -i]
        i <- findLinearCombos(temp)
# excluding variables that are a linear combination of others
        if(!is.null(i$remove)) temp <- temp[, -i$remove]
        data_model <- data_model[,c('outcome', names(temp))]
    }

    if(!is.null(test)){
        data_test <- data_test[,c('outcome',names(temp))]
    }

    # Modelling:
    fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 3,
                           verboseIter = T)

    gbmGrid <-  expand.grid(interaction.depth = c(1,2,4,8),
                          n.trees = 1:2000,
                          shrinkage = c(0.01,0.001),
                          n.minobsinnode = c(2,5))

    err <- try(load(paste(maindir,modeldir, paste('outcome
='',outcome,'lag = ',lag,
                                                  'preprocess =
',preprocess,'include_outcome = ',include_outcome,'test =
',test,'.RData', sep = ''),sep = '/')))
    if(err != 'train1'){
        train1 <- train(outcome ~ ., data= data_model, method = 'gbm',
trControl = fitControl, tuneGrid=gbmGrid)
        save(train1, file = paste(maindir,modeldir, paste('outcome
='',outcome,'lag = ',lag,'preprocess = ',preprocess,'include_outcome =
',include_outcome,
                                                          'test =
',test,'.RData', sep = ''),
                                    sep = '/'))
    }
    if(!is.null(test)){
        data_test$predict <- predict(train1, newdata = data_test)
        return(list(data_test, train1))
    }
```

```
    train1
}
```

Then something weird happened, as I hit an Rsquared of 0.999, this is a near perfect prediction! You can find my training results in the picture below. What turned out is that I fell into a classic timeseries trap. The great thing is that we can learn together what the difference is between a correct prediction and a naïve approach.



What went wrong?

The trap is as follows, by using a naïve cross validation of the dataset we have ignored the temporal aspects of the dataset. Specifically, we included future data points in our training set compared to our validation set, for example in a given training sample there might be a lot of data of the year 2018, while the validation set contains data of 2016. This obviously is not applicable if we want to use this model today to predict 'tomorrow'.

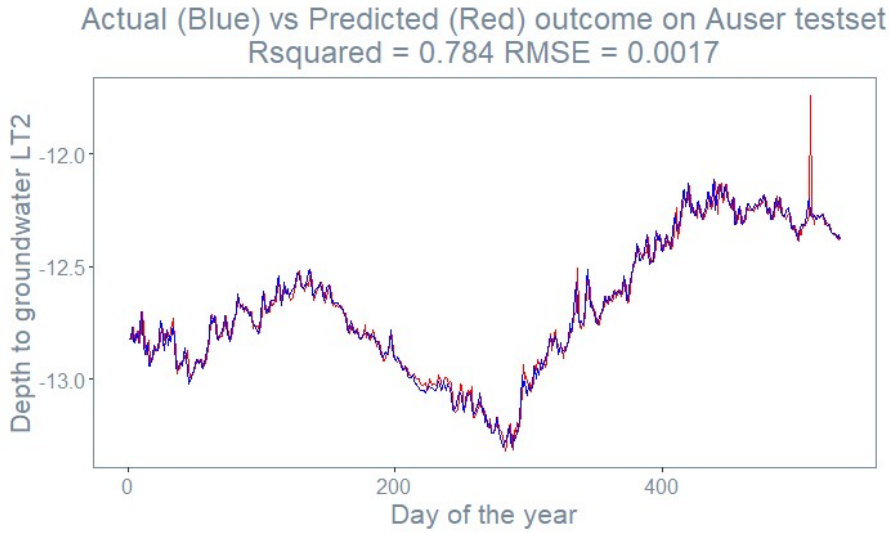
There is a lot of literature on cross validation with time-series, a specific person to look for is Rob Hyndman (3.4 Evaluating forecast accuracy | Forecasting: Principles and Practice (2nd ed) (otexts.com)), who introduced an ultimately more advanced method that I started to work on next. What we need to ensure in our training and validation data is that the temporal aspect is respected, and thus any training data needs to be before the validation data in time. Luckily R has such a rich community that the caret package that I've been using for my modelling has specific settings in the traincontrol setting to address this issue. Hyndmans own forecast package also has options for this modelling procedure.

We are going to make use of the 'timeslice' method of traincontrol, this method allows you to set up a window of time, measured out in rows of your dataset. So a dataset that has a day for each row might have a window of 365 to represent 1 year. Next is a horizon argument, this is the part of the window that is used for validation, a logical approach might be to set the horizon to 30 as in 1 month. Next there is a skip argument, this can be used to control the movement of the window of time, if its set to 30 in our above example, then window 1 is day 1:365, and window 2 is day 31:395. Lastly we can set the fixed window argument, this is for telling the timeslice method if the window moves along the dataset in time, or expands every skip. The relevant code compared to the block above is shown below:

```
  fitControl <- trainControl(method = "timeslice",
                             initialWindow = 500,
                             horizon = 150,
```

```
                    fixedWindow = FALSE,
                    skip = 149,
                    verboseIter = T)
```

Having set all these arguments we rerun the model with proper cross validation and find an Rsquared of 0.78, still high but not borderline unrealistic. Below you can find my training results in more detail:



What can we conclude?

It is actually pretty great to see the exact difference between a naïve modelling approach and one tailored to the issue at hand, I didn't know much about time series modelling and dove straight in. If I had not checked myself on what I was doing and why, the model would have been introduced in a completely overfitted manner.

When looking at the testset we find that the MSE is practically identical for the model training with specific time series cross-validation folds and the overfitted variant. I would personally have to dive deeper into the literature to explain the exact extensive difference, but my gut feeling already tells me that showing up with the overfitted model is a mistake. We can use the specifically designed method and put more trust in its results in a variety of situations and show that on untrained data it performs identically.

To summarize, the new model has slightly less predictive power but is ultimately designed for the environment it has to work for in a production environment.