How to build a real-time influenza prediction model - DIY

Paul Schneider

2017-08-22

---

*This is a fully reproducable **R**-tutorial on how to build a (near) real-time prediction model for influenza in Germany, based on search query data from Google, page view statistics from Wikipedia and influenza incidence data from the Robert Koch Institute. Methods are inspired by Max Kuhn's R-package caret and his book Applied Predictive Modelling. For criticism, comments and questions: schneider.paulpeter@gmail.com*

1 Introduction

1.1 Digital Epidemiology

What is 'Digital Epidemiology'?

The idea behind the predictive influenza model is quite intuitive: We expect that people who get infected with influenza will tend to look up their symptoms, blablabla… Elaborate…

1.2 Milestones in Influenza prediction models

Some milestones: Google Flu Trend in 2008, Wikipedia predictions 2014, Twitter, etc…

1.3 Why Influenza?

In principle, the methods of surveillance, we use in this example, can be applied to any disease. However, some of the features of influenza make it easier to detect relevant predictors and monitor its activity. First, influenza has a strong seasonality (in most countries). Every year there is a start and an end, and during the summer months there are literally no cases of influenza. This strong annual pattern can be detected easily. In diseases, in which the incidence is more stable, there might be not enough variation to distinguish signal and noise. Second, during the flu season, between 5% and 20% of the population get sick. The online traces of rare diseases (say, achalasia) probably get lost in noise. And finally, influenza has distinct symptoms that most people will recognize as flu-symptoms, this means the symptoms are (somewhat) specific to influenza. In contrast, diseases with only subtle or very non-specific symptoms (say, fatigue) are certainly more problematic to track. Nevertheless, there is no reason to assume that other diseases can not be surveyed using similar methods. A few papers have already shown promising results in …Ebola, TUberculosis… [ADD LINK][LINK]. We would indeed be very interested to see how predictive models perform in further diseases, like allergic rhinitis, depression or even sunburns. **If you have good data on this or know where to get it, we invite you to follow this tutorial, build your own model(s) and share your results with us and the public**.

2 Tutorial: Building 'Influenza Nowcast'

2.1 Overview & data sources

We will build a model for predicting the influenza incidence (lab-confirmed cases) in Germany in near real-time ('Nowcasting') by using data from various sources. Outome data is taken from the Robert Koch Institute, predictor data from Google Correlate, Google Trends and Wikipedia via it's API and Wikishark. All analyses are performed using the statistical software R/R Studio, and plenty of its packages, particularly the caret package. Everything we use is available online for free- However, you have to have a Google account to access some of the Google Correlate data.

We will start with getting information on influenza incidence in Germany, since the availability of the outcome data determines the scope and time horizon of the model we build. We then download data on search queries and page view statistics from Google Correlate, Google Trends and Wikipedia, respectively. The data have to be re-fromatted a bit to be matched with each other and the data will be pre-processed before entering into the model. Subsequently, we will use lasso regression to select relevant predictors and build a prediction model. The objective of the model will be to estimate the current influenza incidence.

We would like to point out that when we speak of 'predictions', we use the term in its statistical sense. We do not mean 'forecasting, as in forecasting future influenza cases. We just observe what people were doing on the internet until today and infer from this information how many influenza cases there are about now - We like to call that 'Nowcasting'

If you want to transfer this approach to another setting, the only thing you need is a set of outcome data, i.e. good-quality data of the actual weekly incidence of the disease, over a sufficient time span (1 year +), in a specific country. The rest should work (more or less) automatically, unless you wish to add more data sources as predictors, which may require some more editing. Depending on the country in which you want to predict disease activity, the availability and quality of Google and Wikipedia data may differ substantially. For Wikipedia data, page view statistics can only be distinguished by language, but not by country. Language can be a good proxy in some cases (e.g. Japanese, Korean, Italian, etc.), but a pretty bad one in others (e.g. French, English, Spanish). You can look up how much of a country's Wikipedia traffic is in a specfific language, and how much Wikipedia traffic within a specific language is from a specfific country. Moreover, with weekly data, you should be aware that there are divergent opinions about what is the best starting day for a week (Saturday? Sunday? Monday?).

2.1 Getting started

Before building the model, we need to install and load the required R-packages:

```r
# Install and load all required packages
# This may take a while...
required_packages<-c("knitr","RCurl","ISOweek","jsonlite","ggplot2","prophet","dplyr","gtrendsR","wikipe

pft_packages <- function(package){
    for(i in 1:length(package)){
      if(eval(parse(text=paste("require(",package[i],")")))==0) {
        install.packages(package)}}
    return (eval(parse(text=paste("require(",package,")"))))}

pft_packages(required_packages)

#  gtrendR has to be the developer version:
#  devtools::install_github("PMassicotte/gtrendsR")
#  library(gtrendsR)
```

And we should specify a few key parameters:

```
# What is the outcome of interest? ('term' should correspond to a Wikipedia page)
term = "Influenza" # - Laboratory-confirmed cases of influenza

# For which country do we want to build the model?
country_of_interest = "DE" # Germany in ISO_3166-2 (See: https://en.wikipedia.org/wiki/ISO_3166-2)

# Which language is relevant?
language_of_interest = "de" # German in ISO_639-1 (See: https://en.wikipedia.org/wiki/List_of_ISO_639-1

# What the relevant time horizon (i.e. the time span we have data for)
from = as.Date("2010-07-31") # Start
to = as.Date("2017-07-31") # End

# How do we split the data into training and test data?
split.at = as.Date("2016-08-01")
# --> Training data:  2010-07-31 - 2016-08-01
# --> Test Data:      2016-08-02 - 2017-07-31
```

2.2 Outcome data

We download German influenza incidence data (cases per per 10,000) from the Robert Koch Institute, from August 2010 until August 2017. We uploaded a spreadhseet to github, so it is easier to access. However, you can also go to Survstat and customize your data query. The data do not come in the format we need, so we have to re-arrange it a bit.

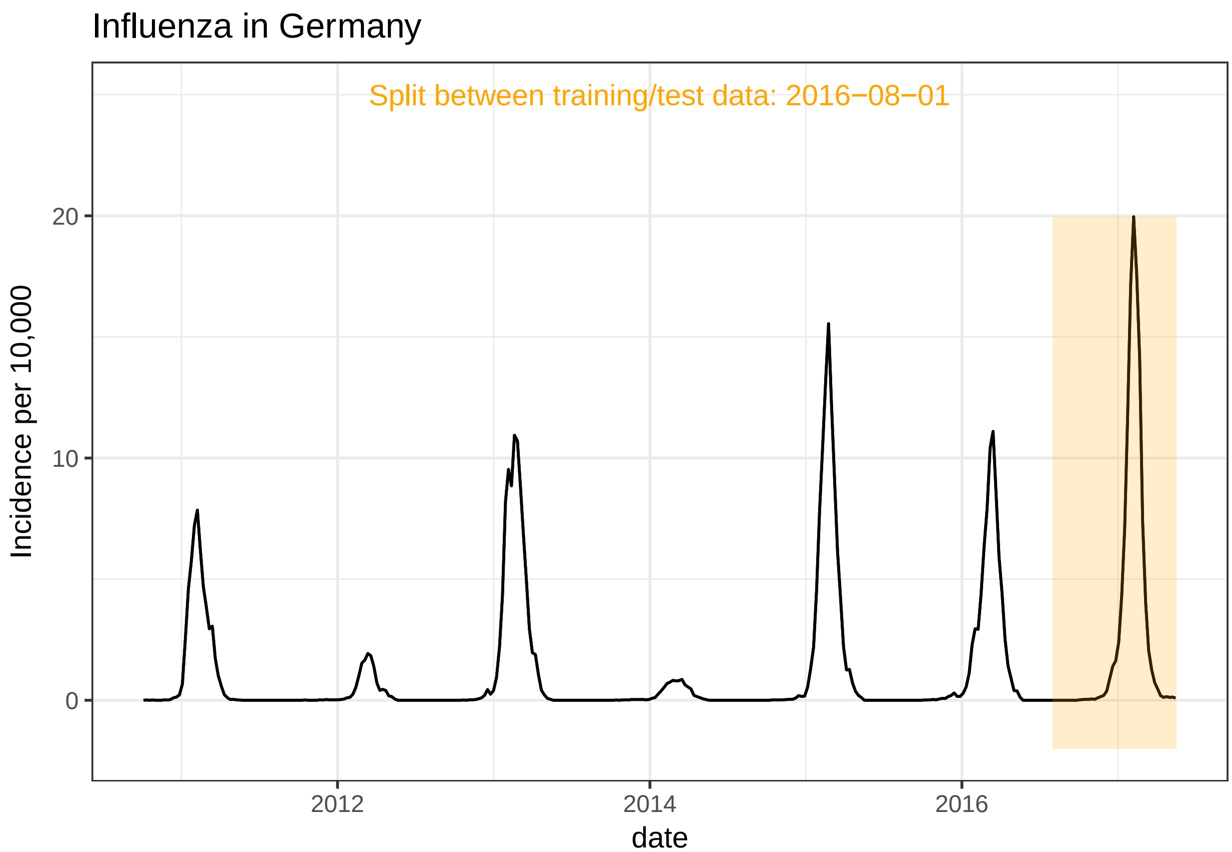
```
# Load data from github repository
influenza.de = read.csv("https://raw.githubusercontent.com/projectflutrend/pft.2/master/outcome%20data/I
names(influenza.de) = c("date","y")
head(influenza.de)
```

  date   y

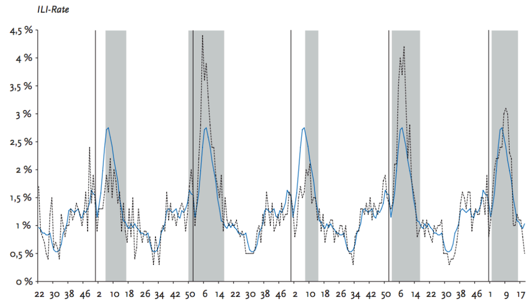
1 2009-w40 1.54 2 2009-w41 1.72 3 2009-w42 1.92 4 2009-w43 3.38 5 2009-w44 9.75 6 2009-w45 24.81

```
# Re-formatting 'date', and removing irrelevant data points
influenza.de$date = paste(sub("w","W", influenza.de$date),"-1", sep="")
influenza.de$date  = ISOweek2date(influenza.de$date)
influenza.de = influenza.de[influenza.de$date>as.Date("2010-07-31") & influenza.de$date<=as.Date("2017-0

# The RKI data does not report data during summer months, in which there are literally no influenza cas
reference =data.frame(date = seq(from=min(influenza.de$date),
                                 to=max(as.Date(influenza.de$date)),
                                 by=7))
influenza.de = merge(reference,influenza.de,by="date",all=T)
influenza.de$y[is.na(influenza.de$y)] = 0

# Plotting the data
ggplot(influenza.de,aes(x=date,y=y)) +
  geom_line() +
  theme_bw() +
  geom_line() +
  annotate("rect", xmin=split.at,
           xmax=max(influenza.de$date)+2,
           ymin=min(influenza.de[,2])-2,
```

```
          ymax=max(influenza.de$y),
          fill="orange",
          alpha=0.2) +
  annotate("text",
          x=median(influenza.de$date),
          y=25,
          label= paste("Split between training/test data:",split.at),
          size=4,col="orange") +
  ylab("Incidence per 10,000") +
  ggtitle("Influenza in Germany")
```



The figures shows the influenza seasons 2010/2011 to 2016/2017, in Germany. More specifically, it shows 'laboratory-confirmed cases'. It should be noted that this type of outcome is probably not ideal to use as a gold standard for training a digital surveillance system that is based on symptoms: If people feel like having the flu, they will look up influenza on Google and Wikipedia. Flu-like symptoms that occur during the summer months are, however, not caused by the influenza virus- this is why the Robert Koch Institute doesn't report any data for these months and why we have to assume a flat line at zero. Our prediction model has to deal with the challenge of distinguishing between people who have the real influenza and people who only have influenza-like symptoms.



As a comparison: pattern of influenza-like illness in Germany, as reported by the Robert Koch Institute. Influenza-like illness is probably more appropriate as an outcome for nowcasting models, as it is a better reflection of the incidence of influenza-like symptoms. However, for Germany, this data is not publicly available.

2.3 The 'null-model'

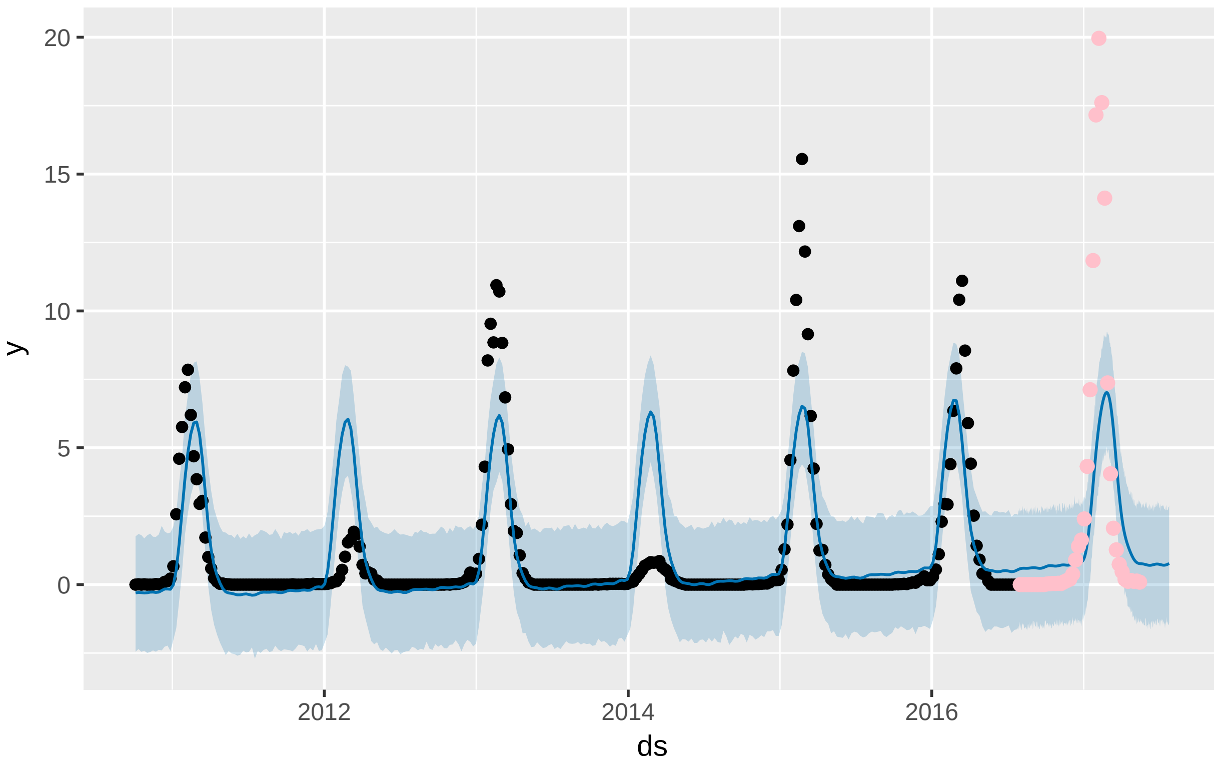
Before we go on to build a prediction model, we should quickly define what we will compare our model against. It would not be fair to compare a prediction model against a flat line at the mean, or at zero cases. This would be like a clinical trial with no comparator. So, let's compare against placebo: Our model should *at least* be better than a 'naiv' forecast, which only knows what has happend in the past, and extrapolates the pattern into the future (*a future that is the present to us*). To build this Null-Model, we will use the prophet package that was build by people at Facebook. It is supposed to automatically detect changes in trends, fit a piecewise linear growth curve and add a yearly seasonal component, modeled using Fourier series. There are probaly better models to forecast influenza, but it is still better than nothing.

For the present data, we will split the data set into training (seaons 2010/2011-2015/2016) and test data sets (season 2016/2017), we fit a forecast model using the training data and ask for a forecast for the next year. Subsequently, we can compare the predictions with the actual data from the test data set.

```r
# Spliting the data for the forecast model into train and test data set
train.influenza.de = influenza.de[influenza.de$date<split.at,]
test.influenza.de = influenza.de[influenza.de$date>=split.at,]

# 'Propheting'
m <- prophet(df=data.frame(ds = train.influenza.de$date,
                           y=train.influenza.de$y),
             growth = "linear",
             yearly.seasonality = T,
             weekly.seasonality = F)
```

```
## Initial log joint probability = -6.91857
## Optimization terminated normally:
##   Convergence detected: relative gradient magnitude is below tolerance
```

```r
future <- make_future_dataframe(m, periods = 365)
forecast <- predict(m, future)
p = plot(m, forecast) +
  geom_point(data=test.influenza.de, aes(x=date,y=y),col="pink",lty="dashed", size=2)
p
```



The prophet-forecast algorithm tries to find a pattern in the available data (2010/2011-2015/2016) and extrapolates it into the future (2016/2017). We can see that the 2016/2017 influenza season was unusually strong, compared to the last six seasons. The forecast model would have underpredicted the number of cases. Furthermore, the season started ealier than usual. So, the forecast model predicted the onset, peak and end of the season a few weeks earlier than they occured. Later, we can use this forecast and compare it to what our Nowcast-model predicts. The metric of interest will be the Root-mean-squared-error.

2.4 Google Correlate

Google Correlate is a tool that can identify search queries that are highly correlated with any time series data you upload (Even for randomly generated numbers it will find a good match). The algorithm was also a building block of [Google Flu Trend](https://googleblog.blogspot.co.uk/2011/05/mining-patterns-in-search-data-with.html. See this paper for more detailes on how Google Trends works, and this guide on how to use it.

What you can also do is to identify queries that are correlated with another query: So, for example, find queries that were related to "Influenza" queries. But we won't do this here.

Unfortunately, there is no Google Correlate API available for R, so you need to go to the website and upload your weekly outcome data manually. Google gives you 100 correlated search queries within the country you select (Not all countries are available) and you can download the results as a .csv file. What you need for that is a) A Google Account and b) A spreadsheet with your data in a specific format. Here, we don't want to leak any information from the test data into the training data- So we will ask for correlated queries only for the trained data set, withholding any patterns seen in the test data set. It's worth noting that there is a "shift series" button. It shifts your data one week in time, if you think there might be a delay between people using Google and the actual reporting of influenza cases (Only shift your data into the past, the predictors should be collected during the week the cases were reported or BEFORE the cases were reported, not afterwards).

For some reason, Google Correlate won't provide data for the entire time span. It only gives data points untill 2017-03-12 (Which is strange...). Anyway, we need to extract the keywords and download the datapoints from google trends.

```r
# We prepare the German Influenza data and save it in the right format
g.cor.influenza.training.upload = influenza.de[influenza.de$date<split.at,]

# Adjusting week format: making Sunday the first day of the week
g.cor.influenza.training.upload$date = g.cor.influenza.training.upload$date-1
# Saving the file in your default folder
write.table( g.cor.influenza.training.upload, col.names=FALSE,row.names = FALSE,sep=",")
```

```
## 2010-10-03,0
## 2010-10-10,0.01
## 2010-10-17,0
## 2010-10-24,0.01
## 2010-10-31,0
## 2010-11-07,0
## 2010-11-14,0
## 2010-11-21,0.02
## 2010-11-28,0.01
## 2010-12-05,0.03
## 2010-12-12,0.1
## 2010-12-19,0.13
## 2010-12-26,0.22
## 2011-01-02,0.67
## 2011-01-09,2.57
## 2011-01-16,4.6
## 2011-01-23,5.76
## 2011-01-30,7.21
## 2011-02-06,7.85
## 2011-02-13,6.2
## 2011-02-20,4.69
## 2011-02-27,3.85
## 2011-03-06,2.95
## 2011-03-13,3.06
## 2011-03-20,1.72
## 2011-03-27,1.01
## 2011-04-03,0.59
## 2011-04-10,0.23
## 2011-04-17,0.11
## 2011-04-24,0.03
## 2011-05-01,0.04
## 2011-05-08,0.02
## 2011-05-15,0.01
## 2011-05-22,0
## 2011-05-29,0
## 2011-06-05,0
## 2011-06-12,0
## 2011-06-19,0
## 2011-06-26,0
## 2011-07-03,0
## 2011-07-10,0
## 2011-07-17,0
## 2011-07-24,0
## 2011-07-31,0
## 2011-08-07,0
```

```
## 2011-08-14,0
## 2011-08-21,0
## 2011-08-28,0
## 2011-09-04,0
## 2011-09-11,0
## 2011-09-18,0
## 2011-09-25,0
## 2011-10-02,0
## 2011-10-09,0
## 2011-10-16,0.01
## 2011-10-23,0
## 2011-10-30,0
## 2011-11-06,0
## 2011-11-13,0
## 2011-11-20,0.02
## 2011-11-27,0.01
## 2011-12-04,0.03
## 2011-12-11,0.02
## 2011-12-18,0.02
## 2011-12-25,0.02
## 2012-01-01,0.02
## 2012-01-08,0.03
## 2012-01-15,0.05
## 2012-01-22,0.1
## 2012-01-29,0.12
## 2012-02-05,0.25
## 2012-02-12,0.54
## 2012-02-19,1.02
## 2012-02-26,1.54
## 2012-03-04,1.66
## 2012-03-11,1.93
## 2012-03-18,1.84
## 2012-03-25,1.39
## 2012-04-01,0.72
## 2012-04-08,0.41
## 2012-04-15,0.45
## 2012-04-22,0.4
## 2012-04-29,0.18
## 2012-05-06,0.15
## 2012-05-13,0.05
## 2012-05-20,0
## 2012-05-27,0
## 2012-06-03,0
## 2012-06-10,0
## 2012-06-17,0
## 2012-06-24,0
## 2012-07-01,0
## 2012-07-08,0
## 2012-07-15,0
## 2012-07-22,0
## 2012-07-29,0
## 2012-08-05,0
## 2012-08-12,0
## 2012-08-19,0
```

```
## 2012-08-26,0
## 2012-09-02,0
## 2012-09-09,0
## 2012-09-16,0
## 2012-09-23,0
## 2012-09-30,0
## 2012-10-07,0
## 2012-10-14,0
## 2012-10-21,0.01
## 2012-10-28,0
## 2012-11-04,0.02
## 2012-11-11,0.02
## 2012-11-18,0.03
## 2012-11-25,0.06
## 2012-12-02,0.1
## 2012-12-09,0.2
## 2012-12-16,0.44
## 2012-12-23,0.25
## 2012-12-30,0.4
## 2013-01-06,0.94
## 2013-01-13,2.19
## 2013-01-20,4.31
## 2013-01-27,8.19
## 2013-02-03,9.53
## 2013-02-10,8.85
## 2013-02-17,10.94
## 2013-02-24,10.71
## 2013-03-03,8.83
## 2013-03-10,6.84
## 2013-03-17,4.94
## 2013-03-24,2.94
## 2013-03-31,1.96
## 2013-04-07,1.89
## 2013-04-14,1.07
## 2013-04-21,0.42
## 2013-04-28,0.22
## 2013-05-05,0.08
## 2013-05-12,0.04
## 2013-05-19,0
## 2013-05-26,0
## 2013-06-02,0
## 2013-06-09,0
## 2013-06-16,0
## 2013-06-23,0
## 2013-06-30,0
## 2013-07-07,0
## 2013-07-14,0
## 2013-07-21,0
## 2013-07-28,0
## 2013-08-04,0
## 2013-08-11,0
## 2013-08-18,0
## 2013-08-25,0
## 2013-09-01,0
```

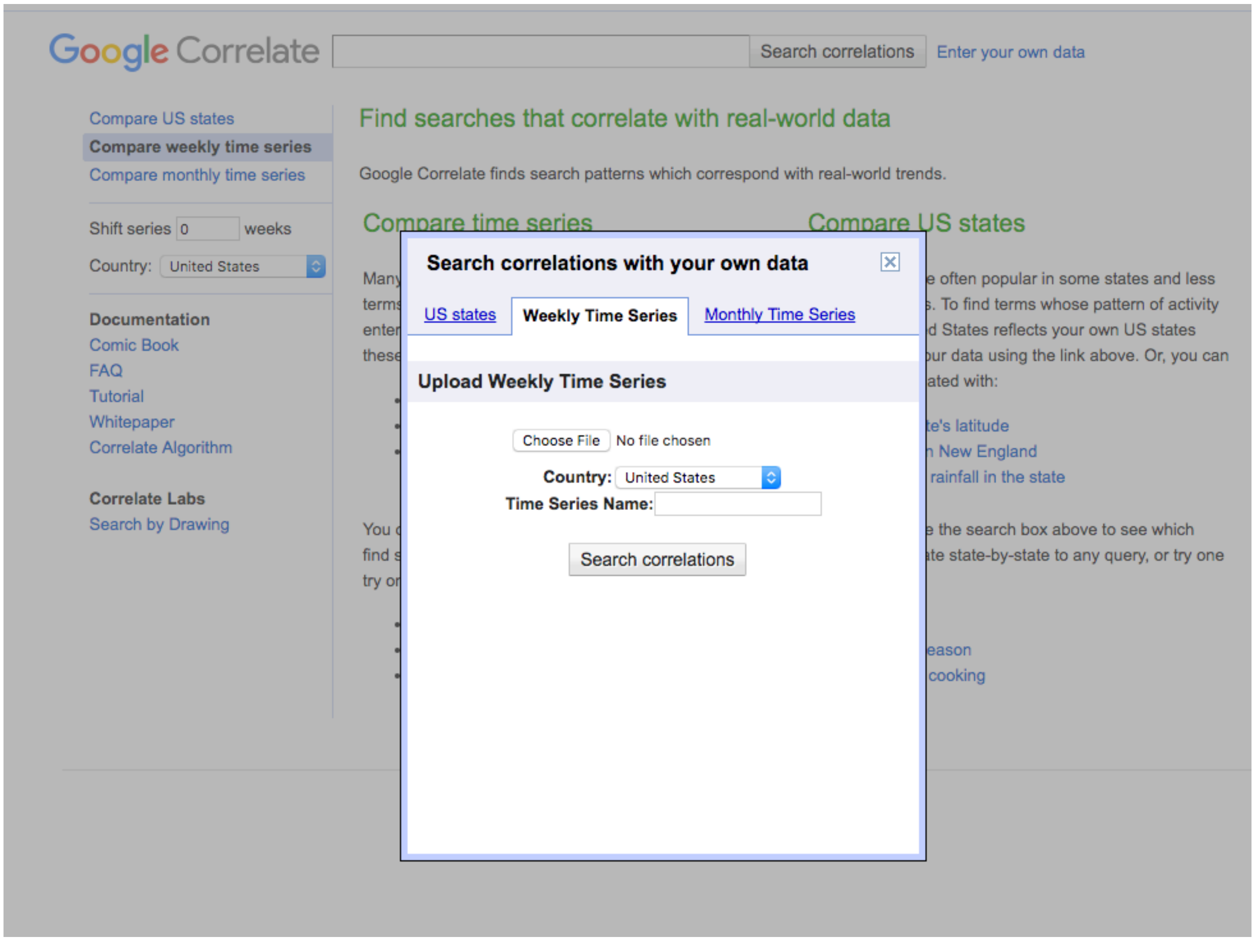
```
## 2013-09-08,0
## 2013-09-15,0
## 2013-09-22,0
## 2013-09-29,0
## 2013-10-06,0
## 2013-10-13,0.01
## 2013-10-20,0
## 2013-10-27,0.01
## 2013-11-03,0.02
## 2013-11-10,0.01
## 2013-11-17,0.03
## 2013-11-24,0.03
## 2013-12-01,0.03
## 2013-12-08,0.03
## 2013-12-15,0.03
## 2013-12-22,0.02
## 2013-12-29,0.03
## 2014-01-05,0.08
## 2014-01-12,0.11
## 2014-01-19,0.25
## 2014-01-26,0.38
## 2014-02-02,0.53
## 2014-02-09,0.69
## 2014-02-16,0.75
## 2014-02-23,0.82
## 2014-03-02,0.8
## 2014-03-09,0.81
## 2014-03-16,0.86
## 2014-03-23,0.64
## 2014-03-30,0.55
## 2014-04-06,0.47
## 2014-04-13,0.2
## 2014-04-20,0.15
## 2014-04-27,0.11
## 2014-05-04,0.06
## 2014-05-11,0.03
## 2014-05-18,0
## 2014-05-25,0
## 2014-06-01,0
## 2014-06-08,0
## 2014-06-15,0
## 2014-06-22,0
## 2014-06-29,0
## 2014-07-06,0
## 2014-07-13,0
## 2014-07-20,0
## 2014-07-27,0
## 2014-08-03,0
## 2014-08-10,0
## 2014-08-17,0
## 2014-08-24,0
## 2014-08-31,0
## 2014-09-07,0
## 2014-09-14,0
```

```
## 2014-09-21,0
## 2014-09-28,0
## 2014-10-05,0
## 2014-10-12,0.01
## 2014-10-19,0.02
## 2014-10-26,0.01
## 2014-11-02,0.02
## 2014-11-09,0.02
## 2014-11-16,0.03
## 2014-11-23,0.04
## 2014-11-30,0.04
## 2014-12-07,0.09
## 2014-12-14,0.19
## 2014-12-21,0.16
## 2014-12-28,0.17
## 2015-01-04,0.54
## 2015-01-11,1.29
## 2015-01-18,2.2
## 2015-01-25,4.55
## 2015-02-01,7.82
## 2015-02-08,10.4
## 2015-02-15,13.1
## 2015-02-22,15.55
## 2015-03-01,12.17
## 2015-03-08,9.15
## 2015-03-15,6.16
## 2015-03-22,4.24
## 2015-03-29,2.22
## 2015-04-05,1.25
## 2015-04-12,1.27
## 2015-04-19,0.72
## 2015-04-26,0.37
## 2015-05-03,0.2
## 2015-05-10,0.11
## 2015-05-17,0
## 2015-05-24,0
## 2015-05-31,0
## 2015-06-07,0
## 2015-06-14,0
## 2015-06-21,0
## 2015-06-28,0
## 2015-07-05,0
## 2015-07-12,0
## 2015-07-19,0
## 2015-07-26,0
## 2015-08-02,0
## 2015-08-09,0
## 2015-08-16,0
## 2015-08-23,0
## 2015-08-30,0
## 2015-09-06,0
## 2015-09-13,0
## 2015-09-20,0
## 2015-09-27,0
```

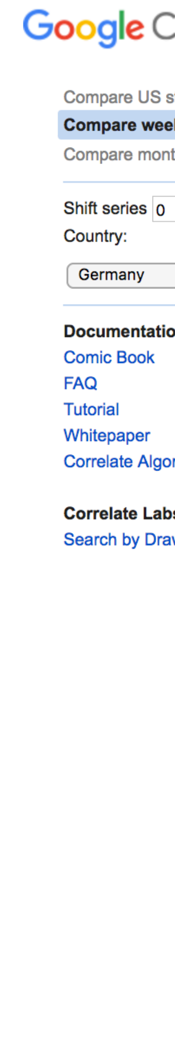
```
## 2015-10-04,0.01
## 2015-10-11,0.01
## 2015-10-18,0.02
## 2015-10-25,0.03
## 2015-11-01,0.02
## 2015-11-08,0.05
## 2015-11-15,0.08
## 2015-11-22,0.07
## 2015-11-29,0.15
## 2015-12-06,0.2
## 2015-12-13,0.3
## 2015-12-20,0.16
## 2015-12-27,0.16
## 2016-01-03,0.29
## 2016-01-10,0.55
## 2016-01-17,1.11
## 2016-01-24,2.3
## 2016-01-31,2.95
## 2016-02-07,2.93
## 2016-02-14,4.4
## 2016-02-21,6.35
## 2016-02-28,7.9
## 2016-03-06,10.41
## 2016-03-13,11.1
## 2016-03-20,8.55
## 2016-03-27,5.9
## 2016-04-03,4.42
## 2016-04-10,2.52
## 2016-04-17,1.42
## 2016-04-24,0.91
## 2016-05-01,0.4
## 2016-05-08,0.39
## 2016-05-15,0.13
## 2016-05-22,0
## 2016-05-29,0
## 2016-06-05,0
## 2016-06-12,0
## 2016-06-19,0
## 2016-06-26,0
## 2016-07-03,0
## 2016-07-10,0
## 2016-07-17,0
## 2016-07-24,0
```

```
### --> Now, go to https://www.google.com/trends/correlate/
###     Login, upload the spreadhseet, and download results
```

Upload data to Google Correlate



Google Correl




```r
# We have put the Google Correlate data for our data on Github:
github.url = "https://raw.githubusercontent.com/projectflutrend/pft.2/master/input%20data/correlate-g_c
# The first 10 lines are text, so we want to skip this part
g.cor.results = read.csv(skip=10,github.url)
# extracting names, except 'date' and 'y'
g.cor.keywords = names(g.cor.results)[-c(1,2)]
# R inserted "." for spaces, we have to undo this:
g.cor.keywords = gsub("\\."," ",g.cor.keywords)

g.cor.keywords[1:20] # Showing the first 20 keywords
```

```
##  [1] "influenza a"                 "influenza"
##  [3] "virusgrippe"                 "j11 1"
##  [5] "grippe fieber"               "grippe verlauf"
##  [7] "influenza schnelltest"       "influenza inkubationszeit"
##  [9] "echte grippe"                "grippe bei kindern"
## [11] "grippe husten"               "grippe influenza"
```

```
## [13] "ist grippe ansteckend"        "influenza grippe"
## [15] "symptome grippe"              "j11 1 g"
## [17] "grippe wie lange"             "symptome influenza"
## [19] "wie lange dauert eine grippe" "verlauf grippe"
```

If you wonder what "J11 1" might be - It's the ICD-10 code for Influenza. Doctors print it on the letters for sick leaves. It seems as if some patients were curious what the doctor had diagnosed them with. Other terms refer to [5] fever, [10] Flu in Children, [11] influenza coughing or [17] Influenza how long

2.5 Google trends

Google Trends is "a public web facility of Google Inc., based on Google Search, that shows how often a particular search-term is entered relative to the total search-volume across various regions of the world, and in various languages" - (from Wikipedia). It can also be used to identify search queries that are related ("People who searched for this, also searched for this" - notice the difference between Google Trends and Correlate!)

First, we are going to download the data for the 100 keywords we have identified using Google Correlate. Moreover, while Google Correlate will most often provide you with good predictors, there might be additional value in taking into account predictors from Google Trends as well, even if only to control for confounding or media-generated over-prediction. Thus, we will download search query statistics for 'influenza', and for 15 related queries, as well as 'influenza'-queries in Google News. If you like, you can expand your requests and increase the numbers of predictors by also retrieving related keywords from related keyword, feed related keywords into Google Correlate, etc.

```r
# We have already set these parameters:
# term = "influenza"; country_of_interest="DE"; from="2010-07-31"; to="2017-07-31"; status= 1

# Identifying 15 related search queries
  google_primer = gtrends(keyword=term,
                          geo=country_of_interest,
                          time=paste(from,to),
                          gprop ="web")

  google_related = google_primer$related_queries$value[google_primer$related_queries$related_queries=="
  g.trends.keywords = c(term,google_related)
  print(g.trends.keywords)
```

```
##  [1] "Influenza"                        "grippe influenza"
##  [3] "virus influenza"                  "grippe"
##  [5] "rki influenza"                    "rki"
##  [7] "symptome influenza"               "influenza impfung"
##  [9] "symptome"                         "inkubationszeit"
## [11] "influenza inkubationszeit"        "hämophilus influenza"
## [13] "influenza 2017"                   "influenza 2016"
## [15] "influenza 2015"                   "fieber"
## [17] "influenza ansteckend"             "haemophilus influenza"
## [19] "influenza impfung pferd"          "influenza 2013"
## [21] "influenza dauer"                  "influenza wie lange ansteckend"
## [23] "influenza impfstoff"              "influenza deutschland"
## [25] "influenza viren"                  "was ist influenza"
```

Noteworthy: 5-year time span restriction: Google Correlate only gives you weekly data for time spans < 5 years. As a possible work-around, you could split your time frame into smaller chunks and download data separately. But because of the way the data is generated, it does not match 100%, even after adjusting the scale. Also, it is not clear whether results will be reproducible, as Google might show slightly different figures in new queries.

The loops to download the search queries are messy and rather long. We will just load them from github, where you can have a closer look at them, if you want.

NOTE: FUNCTIONS DO ONLY WORK FOR THE TIME SPAN SET IN THIS EXMAPLE, NEEDS TO BE REVISED **TRAINING TESTING SPLIT NECCESSARY???**

```
# Loading functions from github:
google.function <- getURL("https://raw.githubusercontent.com/projectflutrend/pft.2/master/quickfunctions
eval(parse(text = google.function))

# Functions loaded are:
# pft_ask_google(keyword,country_of_interest="DE",from="2010-07-31",to="2017-07-31",status= 1,prefix="g
# To split the time span and download and merge query statistics from Google

# Rescale.gtrends(df.t1,df.t2)
# In order to rescale, we look at the overlapping time span and try to find the best mutliplicative sca


# Download query statistics for
  # a) google.correlate queries:
g.cor.input =  pft_ask_google(g.cor.keywords[1:3],country_of_interest="DE",from="2010-07-31",to="2017-0
```

```
## asking Google for statistics for influenza a  -  33.3 %
## asking Google for statistics for influenza  -  66.7 %
## asking Google for statistics for virusgrippe  -  100 %
```

```
 # b) trends queries:
g.trends.input =  pft_ask_google(g.trends.keywords[1:3],country_of_interest="DE",from="2010-07-31",to="2
```

```
## asking Google for statistics for Influenza  -  33.3 %
## asking Google for statistics for grippe influenza  -  66.7 %
## asking Google for statistics for virus influenza  -  100 %
```

```
 # c) news on influenza as a potentially relevant (negative) predictor
g.news.input =  pft_ask_google("influenza",country_of_interest="DE",from="2010-07-31",to="2017-07-31",s
```

```
## asking Google for statistics for influenza  -  100 %
```

```
google.input.data = merge(g.cor.input,g.trends.input,by="date",all=T)
google.input.data = merge(google.input.data,g.news.input,by="date",all=T)

dim(google.input.data)
```

```
## [1] 366   8
```

```
head(google.input.data[,1:5])
```

```
##       date g.cor.influenza.a g.cor.influenza g.cor.virusgrippe
## 1 2010-W30               2.9               5               5.1
## 2 2010-W31               5.0               4               5.1
## 3 2010-W32               2.9               6               5.1
## 4 2010-W33               3.6               6               7.2
## 5 2010-W34               5.7               7               7.2
## 6 2010-W35               5.7               8               4.1
##   g.trends.Influenza
## 1                  5
```

```
## 2                 4
## 3                 6
## 4                 6
## 5                 7
## 6                 8
```

If you want to reproduce this example and avoid long downloading times, you can download the data [here] ***FIX LINK***

2.6 Wikipedia data

McIver & Brownstein made a strong case for the utility of Wikipedia page view data in influenza prediction models. They showed that "Wikipedia usage accurately estimated the week of peak ILI activity 17% more often than Google Flu Trends data and was often more accurate in its measure of ILI intensity". Combining Google data with Wikipedia may have further advantges.

We can access Wikipedia page view statistics via it's API (for data from October 2015 until today) and Wikishark (for data from January 2008 until December 2016)

- **On a side note**: On Wikipedoa, pages with similar content are linked with each others across different languages. The Wikipediatrend package offers a convenient way to identify these corresponding pages: For 'Influenza', there are about 80 similar pages. While finding the German word for influenza (it's 'influenza'), is not too impressive, finding Wikipedia disease pages in Arabic and Japanese can be quite useful. Btw, Wikipedia refers to languages in ISO 639-1 code.

```r
# The 'Influenza' page in other languges
wikipediatrend::wp_linked_pages( page= "Influenza",lang="en")[1:10,2:3]
```

```
##    lang     title
## 1  af       Griep
## 2  als      Influenza
## 3  ar
## 4  an       Gripe
## 5  roa-rup  Aremi
## 6  as
## 7  ast      Gripe
## 8  gn       Mba'asyparar ...
## 9  ay       Jurma_usu
## 10 az       Qrip
```

To download Wikipedia page view data, we have to turn to two sifferent sources of data. Wikipedia has its own API, which allows fast and convenient open access. However, the API is only able to retrieve data for > October 2015. Statistics for prior dates are stored in huge files, showing page views per hour in every language. Fortunately, there is a private project, called wikishark, which we can quickly use to download relevant data per week (There used to be another server with an API: stats.grok.se, but it appears to be down since July 2017). For the two time periods, Wikipedia has used different metrics to count page views and to detmerine individual page visitors. Thus, there might be some discrepancies. It is also important to note that Wikipedia is not static. New pages are created and old pages may be changed or merged, i.e.for some pages that do exist today, we won't find any meaningful page view statistics in 2010 - So, expect (and ignore) some errors when downloading the data.

We start with downloading page view statistics for the main influenza page. Then, we extract all links on this page which refer to other wikipedia pages (On the Influenza Wikipedia page there are links to pages baout Aspirin, bronchitis, Allergy etc.). In addition, we can extract statistics for all pages which link to the influenza page (Most mention and link to influenza only en passant). Of you like, you can also add Wikipedia pages manually, if you think people who get flu will look them up (or people who don't have the flu, but visit the infuenza page, will look them up).

Again, the functions to identify linked pages and to download the statistics are rather long and messy. We cut it short and recommend just loading the functions from github, without going through the code. If you want to have a look at it, click here.

NOTE: FUNCTIONS DO ONLY WORK FOR THE TIME SPAN SET IN THIS EXMAPLE, NEEDS TO BE REVISED

```
# Loading functions from github:
wiki.functions <- getURL("https://raw.githubusercontent.com/projectflutrend/pft.2/master/quickfunctions/
eval(parse(text = wiki.functions))

# pft_wiki_lp(term = "Influenza", language_of_interest = "de", backlinked = 1 ,manual.pages=c("Halsschm

# pft_ask_wikipedia(pages = wiki.pages,language_of_interest =  "de", from = as.Date("2010-01-01"),to =

# Retrive potentially relevant Wikipedia pages
wiki.pages = pft_wiki_lp(term = "Influenza",
                         language_of_interest = "de",
                         backlinked = 1,
                         manual.pages=c("Halsschmerzen","Hausmittel"))
str(wiki.pages)
```

```
##  chr [1:604] "Influenza" "Acetylsalicylsäure" ...
```

```
# We have identified the main influenza page, 149 pages that are links on the influenza page, and 451 p

# Now, we can download their page voew statistics (Depending on the amount of relevant pages, this may
wikipedia.input.data = pft_ask_wikipedia(pages = wiki.pages[1:3],
                                         language_of_interest =  "de",
                                         from = as.Date("2010-01-01"),
                                         to = as.Date("2017-07-31"),
                                         status = 1)
```

```
## Downloading data for page 1 of 3  -  33.33333 %
## Downloading data for page 2 of 3  -  66.66667 %
## Downloading data for page 3 of 3  -  100 %
```

```
head(wikipedia.input.data[,1:4])
```

```
##       date wiki.Influenza wiki.Acetylsalicylsäure
## 1 2009-W53             NA                      NA
## 2 2010-W01     -1.1099757                11.40327
## 3 2010-W02     -0.5476942                17.00074
## 4 2010-W03     -1.3283038                11.03771
## 5 2010-W04     -1.3317693                12.65428
## 6 2010-W05     -1.3456314                17.82682
##   wiki.Adolf_Mayer_.Agronom.
## 1                         NA
## 2                 -0.1218000
## 3                  4.1631075
## 4                 -2.8485592
## 5                  0.6572741
## 6                  1.0468112
```

If you want to reproduce this example and avoid long downloading times, you can download the data [here] ***FIX LINK***

2.4.4 Putting the data together and pre-processing

Now, that we have all the data we wanted, we merge the files into one dataframe, and split it again into predictors and outcome data, as well as training and testing/evaluation data sets.

Preprocessing: Scaling, Centering. Also, many predictors are hughly correlated. Especially in the Google Correlate data set (see figure below)

```
# Combining outcome, wikipedia, google trends and google correlate
influenza.de$date = ISOweek(influenza.de$date )

# Merging by week (avoiding any Monday/Sunday or other day issues)
df.full = merge(influenza.de,google.input.data, by="date")
df.full = merge(df.full,wikipedia.input.data, by="date")

# Setting date back to a date
df.full$date = ISOweek2date(paste(df.full$date,"-1",sep="")) #
dim(df.full)
```

```
## [1] 346  12
```

```
# save(df.full, file="/users/waqr/desktop/df.full.de.2.rdata")
```

CHECK DIMS! 337 rows (=weeks),548 columns (date,outcome,546 potential predictors)

Splitting the data and pre-processing

First, we split the data into outcome/predictors, as well as training/testing data sets. From now on, we try not to leak any kind of information from the test data set to the training data set.

```
split = which(df.full$date<split.at)

df.train = df.full[split,-c(1,2)] # Predictor training data set
y.train = df.full[split,c(2)] # Outcome for training data set
date.train = df.full[split,c(1)] # Date, not a predictor but useful for plotting

df.test  = df.full[-split,-c(1,2)] # Predictors for testing/evaluation data set
y.test = df.full[-split,c(2)] # Outcome for testing data set
date.test = df.full[-split,c(1)] # date for test data set
```

Second, we will scale and center all predictors - Google data comes scaled and centered already, so in this example, we will do this with Wikipedia data. This improves the performance for some of the modelling techniques and it makes it easier to comapre predictors. However, we loose information about how many visits in absolut numbers a Wikipedia page had.

Third, we remove predictors that have too many (Say, >10%) missing values (See 'wiki.Geflügel.Aufstallungsverordung' = translatation?!) in figure below). This method we will also apply t the test data set. If we wouldn't, we could end up with test-predictors that have too many, or only missing values. Then, we would need to re-run the whole model again without the missing test-predictor. So, we want to make sure that sufficient cases are available in both, training and test data set. When less than 10% of cases are missing, we are going to impute the missing values by "k-nearest neighbor imputation"[Add a link, description], for the training and test data set separately.

Fourth, we also want to remove predictors that have near zero variance. This means they have few unique values. Take for example the Wikipedia page on "Samuel Warren Abott": The page was created only in 2015, so in the time before, the page had mostly 0 page views (Only rarely someone tries to access a Wikipedia pages that are non-existent). In addition, this page also has some missing values. Prediction models can be adversely affected by these near zero variance predictors, and they don't really add any value to the models.

You could also consider removing predictors that have a very low activity, because results could be unstable if models put weight in them.

NOT RUNNING THE FOLLOWING CODE IN SHORTEND VERSION OF THE CODE

```
# Removing features with >10% NAs
# in training
sum.NA.train = as.numeric(lapply(df.train,function(x){sum(is.na(x))}))
sum.NA.train = sum.NA.train > length(df.train[,1]) * 0.1
if(sum(sum.NA.train)>0){
df.train = df.train[-which(sum.NA.train)]
df.test = df.test[which(colnames(df.test) %in% colnames(df.train))]}
# and test data separately
sum.NA.test = as.numeric(lapply(df.test,function(x){sum(is.na(x))}))
sum.NA.test = sum.NA.test > length(df.test[,1]) * 0.1
if(sum(sum.NA.test)>0){
df.test = df.test[-which(sum.NA.test)]
df.train = df.train[which(colnames(df.train) %in% colnames(df.test))]}

# Removing features with near zero variance
# identify near zero-variance predictors [only in df.train!]
nearZeroVar = nearZeroVar(df.train,freqCut = 95/5 , uniqueCut = 25)
if(sum(nearZeroVar)>0){
df.train = df.train[,-nearZeroVar]
df.test = df.test[which(colnames(df.test) %in% colnames(df.train))]}


# Scaling, centering, and imputing remaining NAs
preprocess.df.train = preProcess(df.train, method=c("scale","center","knnImpute")) # why knnimpute erro

df.train = predict(preprocess.df.train, newdata = df.train)
df.test = predict(preprocess.df.train,newdata = df.test)
```

NOT RUNNING THE FOLLOWING CODE IN SHORTEND VERSION OF THE CODE

We can also consider principal component analysis, assome methods might work better when we reduce multi-colinerity. Especially Google Correlate data is highly correlated -> We can plot an example with just 10 predictors, but google correlate gives us 100 of these kind!

2.5 Building a preditcion model

For the purpouse of this tutorial, we will only use lasso-regression to build and train our model. It's a technique that automatically selects and regularizes predictors - See Tibshirani for a technical or this website and video for a more practical explanation. For a real application, we can try and evaluate many other methods, e.g. elatic net, multivariate adaptive regression splines, rule-based regression and other, available in the caret package. The predictive accuracy of the different models can be compared in terms of root-mean-squared-error, using cross-validation (see below). We can then choose whatever model is most accurate, regardless of it's form, complexity or the amount or types of predictors it uses. It turns out, though, that lasso regression seems to be always among the top performing methods in the type of model we are building. Also, compared to many other techniques, it is computationally very fast.

Cross-validation

Cross validation, rolling forward

- explain what cv is
- explain why split-cv doesn't work
- explain what rolling forward cv is
- Talk about an appropriate horizon
  - ideal: cv and 1-week forward rolling evaluation plot!?
- whats a grid?

```
# control object for cross validation: rolling forward cv with fixed origin
controlObject <- trainControl(method = "timeslice",
                              initialWindow = 52,
                              horizon = 52, # 1-52 # Smaller horizon better
                              fixedWindow = FALSE,
                              allowParallel = TRUE)

# setting grids
glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),.lambda = seq(.01, 1, length = 40))
# cubistGrid <- expand.grid(.committees = c(1, 5, 10, 50, 75, 100),.neighbors=c(0,1,3,5,7,9))
```

```
# paralel computing
no_cores <- detectCores() - 1
cl <- makeCluster(no_cores, type="FORK")
registerDoParallel(cl)
```

Model Specifications

```
# formula list
formula.list = list(glmnet.mod = list(y= y.train ,
                                      x = df.train,
                                      method = "glmnet",
                                      family = "gaussian",
                                      tuneGrid = glmnGrid,
                                      trControl = controlObject))
```

```
# A function to evaluate the models
eval.function <- getURL("https://raw.githubusercontent.com/projectflutrend/pft.2/master/quickfunctions/p
eval(parse(text = eval.function))

if(1==0){
```

```r
# A loop to build and evalute the model
models.de = list(result.list = list(), eval.list = list())
for(i in 1:length(formula.list)){
  cat("Building a model:",formula.list[[i]]$method,"\n")
  tryCatch({
    models.de$result.list[[i]] = do.call("train",formula.list[[i]])
    names(models.de$result.list)[i] = names(formula.list)[i]

    models.de$eval.list[[i]] = pft_eval_model(models.de$result.list[[i]])
    names(models.de$eval.list)[i] = names(formula.list)[i]

    cat(formula.list[[i]]$method,"evaluation done! \n")},
    error=function(e) {cat("error in",formula.list[[i]]$method,"\n")})
}


}
```

619 Evaluate the models

```r
#### evaluation plots
#models.de$eval.list[[1]]$plots$pred.plot

#models.de$eval.list[[2]]$plots$pred.plot
```

620 Conclusing remarks