I wanted to try using Vowpal Wabbit  
for now because it seems to be the perfect  
tool for when you’re dealing with what I call *big-ish* data: data that is not big data, and might  
fit in your RAM, but is still a PITA to deal with. It can be data that is large enough to take 30  
seconds to be imported into R, and then every operation on it lasts for minutes, and estimating/training  
a model on it might eat up all your RAM. Vowpal Wabbit avoids all this because it’s an online-learning  
system. Vowpal Wabbit is capable of training a model with data that it sees on the fly, which means  
VW can be used for real-time machine learning, but also for when the training data is very large.  
Each row of the data gets streamed into VW which updates the estimated parameters of the model  
(or weights) in real time. So no need to first import all the data into R!

The goal of this blog post is to get started with VW, and build a very simple logistic model  
to classify documents using the historical newspapers data from the National Library of Luxembourg,  
which you can download [here](https://data.bnl.lu/data/historical-newspapers/) (scroll down and  
download the *Text Analysis Pack*). The goal is not to build the best model, but *a* model. Several  
steps are needed for this: prepare the data, install VW and train a model using {RVowpalWabbit}.

**Step 1: Preparing the data**

The data is in a neat .xml format, and extracting what I need will be easy. However, the input  
format for VW is a bit unusual; it resembles *.psv* files (**P**ipe **S**eparated **V**alues) but  
allows for more flexibility. I will not dwell much into it, but for our purposes, the file must  
look like this:

1 | this is the first observation, which in our case will be free text

2 | this is another observation, its label, or class, equals 2

4 | this is another observation, of class 4

The first column, before the “|” is the target class we want to predict, and the second column  
contains free text.

The raw data looks like this:

Click if you want to see the raw data

2019-02-28T11:13:01

http://www.eluxemburgensia.lu/OAI

digitool-publish:3026998-DTL45

2019-02-28T11:13:01Z

https://persist.lu/ark:/70795/6gq1q1/articles/DTL45

newspaper/indeplux/1871-12-29\_01

L'indépendance luxembourgeoise

issue:newspaper/indeplux/1871-12-29\_01/article:DTL45

1871-12-29

Jean Joris

3026998

http://www.eluxemburgensia.lu/webclient/DeliveryManager?pid=3026998#panel:pp|issue:3026998|article:DTL45

CONSEIL COMMUNAL de la ville de Luxembourg. Séance du 23 décembre 1871. (Suite.) Art. 6. Glacière communale. M. le Bourgmcstr ¦ . Le collège échevinal propose un autro mode de se procurer de la glace. Nous avons dépensé 250 fr. cha- que année pour distribuer 30 kilos do glace; c’est une trop forte somme pour un résultat si minime. Nous aurions voulu nous aboucher avec des fabricants de bière ou autres industriels qui nous auraient fourni de la glace en cas de besoin. L’architecte qui été chargé de passer un contrat, a été trouver des négociants, mais ses démarches n’ont pas abouti.

CONSEIL COMMUNAL de la ville de Luxembourg. Séance du 23 décembre 1871. (Suite.)

ARTICLE

fr

863

I need several things from this file:

* The title of the newspaper: L'indépendance luxembourgeoise
* The type of the article: ARTICLE. Can be Article, Advertisement, Issue, Section or Other.
* The contents: CONSEIL COMMUNAL de la ville de Luxembourg. Séance du ....

I will only focus on newspapers in French, even though newspapers in German also had articles in French.  
This is because the tag fr is not always available. If it were, I could  
simply look for it and extract all the content in French easily, but unfortunately this is not the case.

First of all, let’s get the data into R:

library("tidyverse")

library("xml2")

library("furrr")

files <- list.files(path = "export01-newspapers1841-1878/", all.files = TRUE, recursive = TRUE)

This results in a character vector with the path to all the files:

head(files)

[1] "000/1400000/1400000-ADVERTISEMENT-DTL78.xml" "000/1400000/1400000-ADVERTISEMENT-DTL79.xml"

[3] "000/1400000/1400000-ADVERTISEMENT-DTL80.xml" "000/1400000/1400000-ADVERTISEMENT-DTL81.xml"

[5] "000/1400000/1400000-MODSMD\_ARTICLE1-DTL34.xml" "000/1400000/1400000-MODSMD\_ARTICLE2-DTL35.xml"

Now I write a function that does the needed data preparation steps. I describe what the function  
does in the comments inside:

to\_vw <- function(xml\_file){

# read in the xml file

file <- read\_xml(paste0("export01-newspapers1841-1878/", xml\_file))

# Get the newspaper

newspaper <- xml\_find\_all(file, ".//dcterms:isPartOf") %>% xml\_text()

# Only keep the newspapers written in French

if(!(newspaper %in% c("L'UNION.",

"L'indépendance luxembourgeoise",

"COURRIER DU GRAND-DUCHÉ DE LUXEMBOURG.",

"JOURNAL DE LUXEMBOURG.",

"L'AVENIR",

"L’Arlequin",

"La Gazette du Grand-Duché de Luxembourg",

"L'AVENIR DE LUXEMBOURG",

"L'AVENIR DU GRAND-DUCHE DE LUXEMBOURG.",

"L'AVENIR DU GRAND-DUCHÉ DE LUXEMBOURG.",

"Le gratis luxembourgeois",

"Luxemburger Zeitung – Journal de Luxembourg",

"Recueil des mémoires et des travaux publiés par la Société de Botanique du Grand-Duché de Luxembourg"))){

return(NULL)

} else {

# Get the type of the content. Can be article, advert, issue, section or other

type <- xml\_find\_all(file, ".//dc:type") %>% xml\_text()

type <- case\_when(type == "ARTICLE" ~ "1",

type == "ADVERTISEMENT" ~ "2",

type == "ISSUE" ~ "3",

type == "SECTION" ~ "4",

TRUE ~ "5"

)

# Get the content itself. Only keep alphanumeric characters, and remove any line returns or

# carriage returns

description <- xml\_find\_all(file, ".//dc:description") %>%

xml\_text() %>%

str\_replace\_all(pattern = "[^[:alnum:][:space:]]", "") %>%

str\_to\_lower() %>%

str\_replace\_all("\r?\n|\r|\n", " ")

# Return the final object: one line that looks like this

# 1 | bla bla

paste(type, "|", description)

}

}

I can now run this code to parse all the files, and I do so in parallel, thanks to the {furrr} package:

plan(multiprocess, workers = 12)

text\_fr <- files %>%

future\_map(to\_vw)

text\_fr <- text\_fr %>%

discard(is.null)

write\_lines(text\_fr, "text\_fr.txt")

**Step 2: Install Vowpal Wabbit**

To easiest way to install VW must be using Anaconda, and more specifically the conda package manager.  
Anaconda is a Python (and R) distribution for scientific computing and it comes with a package manager  
called conda which makes installing Python (or R) packages very easy. While VW is a standalone  
piece of software, it can also be installed by conda or pip. Instead of installing the full Anaconda distribution,  
you can install Miniconda, which only comes with the bare minimum: a Python executable and the  
conda package manager. You can find Miniconda [here](https://docs.conda.io/en/latest/miniconda.html) and once it’s installed, you can install VW with:

conda install -c gwerbin vowpal-wabbit

It is also possible to install VW with pip, as detailed [here](https://pypi.org/project/vowpalwabbit/),  
but in my experience, managing Python packages with pip is not super. It is better to manage your  
Python distribution through conda, because it creates environments in your home folder which are  
independent of the system’s Python installation, which is often out-of-date.

**Step 3: Building *a* model**

Vowpal Wabbit can be used from the command line, but there are interfaces for Python and since a  
few weeks, for R. The R interface is quite crude for now, as it’s still in very early stages. I’m  
sure it will evolve, and perhaps a Vowpal Wabbit engine will be added to {parsnip}, which would  
make modeling with VW really easy.

For now, let’s only use 10000 lines for prototyping purposes before running the model on the whole file. Because  
the data is quite large, I do not want to import it into R. So I use command line tools to manipulate  
this data directly from my hard drive:

# Prepare data

system2("shuf", args = "-n 10000 text\_fr.txt > small.txt")

shuf is a Unix command, and as such the above code should work on GNU/Linux systems, and most  
likely macOS too. shuf generates random permutations of a given file to standard output. I use >  
to direct this output to another file, which I called small.txt. The -n 10000 options simply  
means that I want 10000 lines.

I then split this small file into a training and a testing set:

# Adapted from http://bitsearch.blogspot.com/2009/03/bash-script-to-split-train-and-test.html

# The command below counts the lines in small.txt. This is not really needed, since I know that the

# file only has 10000 lines, but I kept it here for future reference

# notice the stdout = TRUE option. This is needed because the output simply gets shown in R's

# command line and does get saved into a variable.

nb\_lines <- system2("cat", args = "small.txt | wc -l", stdout = TRUE)

system2("split", args = paste0("-l", as.numeric(nb\_lines)\*0.99, " small.txt data\_split/"))

split is the Unix command that does the splitting. I keep 99% of the lines in the training set and  
1% in the test set. This creates two files, aa and ab. I rename them using the mv Unix command:

system2("mv", args = "data\_split/aa data\_split/train.txt")

system2("mv", args = "data\_split/ab data\_split/test.txt")

Ok, now let’s run a model using the VW command line utility from R, using system2():

oaa\_fit <- system2("~/miniconda3/bin/vw", args = "--oaa 5 -d data\_split/train.txt -f oaa.model", stderr = TRUE)

I need to point system2() to the vw executable, and then add some options. --oaa stands for  
*one against all* and is a way of doing multiclass classification; first, one class gets classified  
by a logistic classifier against all the others, then the other class against all the others, then  
the other…. The 5 in the option means that there are 5 classes.

-d data\_split/train.txt specifies the path to the training data. -f means “final regressor”  
and specifies where you want to save the trained model.

This is the output that get’s captured and saved into oaa\_fit:

[1] "final\_regressor = oaa.model"

[2] "Num weight bits = 18"

[3] "learning rate = 0.5"

[4] "initial\_t = 0"

[5] "power\_t = 0.5"

[6] "using no cache"

[7] "Reading datafile = data\_split/train.txt"

[8] "num sources = 1"

[9] "average since example example current current current"

[10] "loss last counter weight label predict features"

[11] "1.000000 1.000000 1 1.0 3 1 87"

[12] "1.000000 1.000000 2 2.0 1 3 2951"

[13] "1.000000 1.000000 4 4.0 1 3 506"

[14] "0.625000 0.250000 8 8.0 1 1 262"

[15] "0.625000 0.625000 16 16.0 1 2 926"

[16] "0.500000 0.375000 32 32.0 4 1 3"

[17] "0.375000 0.250000 64 64.0 1 1 436"

[18] "0.296875 0.218750 128 128.0 2 2 277"

[19] "0.238281 0.179688 256 256.0 2 2 118"

[20] "0.158203 0.078125 512 512.0 2 2 61"

[21] "0.125000 0.091797 1024 1024.0 2 2 258"

[22] "0.096191 0.067383 2048 2048.0 1 1 45"

[23] "0.085205 0.074219 4096 4096.0 1 1 318"

[24] "0.076172 0.067139 8192 8192.0 2 1 523"

[25] ""

[26] "finished run"

[27] "number of examples = 9900"

[28] "weighted example sum = 9900.000000"

[29] "weighted label sum = 0.000000"

[30] "average loss = 0.073434"

[31] "total feature number = 4456798"

Now, when I try to run the same model using RVowpalWabbit::vw() I get the following error:

oaa\_class <- c("--oaa", "5",

"-d", "data\_split/train.txt",

"-f", "vw\_models/oaa.model")

result <- vw(oaa\_class)

Error in Rvw(args) : unrecognised option '--oaa'

I think the problem might be because I installed Vowpal Wabbit using conda, and the package  
cannot find the executable. I’ll open an issue with reproducible code and we’ll see.

In any case, that’s it for now! In the next blog post, we’ll see how to get the accuracy of this  
very simple model, and see how to improve it!