# Benchmark based on the WiLI dataset

The *fastText* language identification pre-trained models support currently 176 languages. The following character vector shows the available *language isocodes*.

fasttext\_supported\_languages = c('af', 'als', 'am', 'an', 'ar', 'arz', 'as', 'ast', 'av',

'bg', 'bh', 'bn',

'cbk', 'ce', 'ceb',

'de', 'diq', 'dsb',

'es', 'et', 'eu',

'gd', 'gl', 'gn',

'hr', 'hsb', 'ht',

'io', 'is', 'it',

'kn', 'ko', 'krc',

'lez', 'li', 'lmo',

'mhr', 'min', 'mk',

'mwl', 'my', 'myv',

'new', 'nl', 'nn', 'no',

'pl', 'pms', 'pnb',

'rue', 'sa', 'sah',

'sk', 'sl', 'so',

'te', 'tg', 'th', 'tk',

'ur', 'uz', 'vec',

'wuu', 'xal', 'xmf',

'az', 'azb', 'ba', 'bar', 'bcl', 'be',

'bo', 'bpy', 'br', 'bs', 'bxr', 'ca',

'ckb', 'co', 'cs', 'cv', 'cy', 'da',

'dty', 'dv', 'el', 'eml', 'en', 'eo',

'fa', 'fi', 'fr', 'frr', 'fy', 'ga',

'gom', 'gu', 'gv', 'he', 'hi', 'hif',

'hu', 'hy', 'ia', 'id', 'ie', 'ilo',

'ja', 'jbo', 'jv', 'ka', 'kk', 'km',

'ku', 'kv', 'kw', 'ky', 'la', 'lb',

'lo', 'lrc', 'lt', 'lv', 'mai', 'mg',

'ml', 'mn', 'mr', 'mrj', 'ms', 'mt',

'mzn', 'nah', 'nap', 'nds', 'ne',

'oc', 'or', 'os', 'pa', 'pam', 'pfl',

'ps', 'pt', 'qu', 'rm', 'ro', 'ru',

'sc', 'scn', 'sco', 'sd', 'sh', 'si',

'sq', 'sr', 'su', 'sv', 'sw', 'ta',

'tl', 'tr', 'tt', 'tyv', 'ug', 'uk',

'vep', 'vi', 'vls', 'vo', 'wa', 'war', 'yi', 'yo', 'yue', 'zh')

For illustration purposes we’ll subset the *WiLI dataset* to the 2-letter isocodes of the supported fastText languages. For this purpose we’ll use the ISOcodes R package and especially the **ISO\_639\_2** function which includes the required 2- and 3-letter isocodes and also the available full names of the languages,

isocodes = ISOcodes::ISO\_639\_2 # head(isocodes)

comp\_cases = complete.cases(isocodes$Alpha\_2) isocodes\_fasttext = isocodes[comp\_cases, ]

# dim(isocodes\_fasttext)

idx\_keep\_fasttext = which(isocodes\_fasttext$Alpha\_2 %in% fasttext\_supported\_languages)

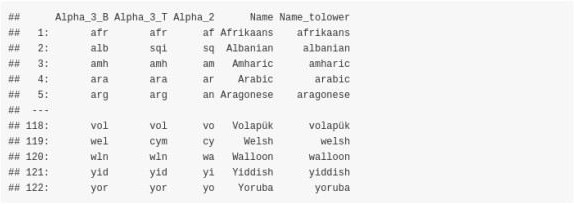
isocodes\_fasttext = isocodes\_fasttext[idx\_keep\_fasttext, ] isocodes\_fasttext = data.table::data.table(isocodes\_fasttext) # isocodes\_fasttext

lower\_nams = tolower(isocodes\_fasttext$Name)

lower\_nams = trimws(as.vector(unlist(lapply(strsplit(lower\_nams, "[;,

]"), function(x) x[1]))), which = 'both') # remove second or third naming of the country name

isocodes\_fasttext$Name\_tolower = lower\_nams isocodes\_fasttext



The next function will be used to compute and print the accuracy in all cases,

print\_accuracy = function(size\_input\_data,

true\_data, preds\_data, method) {

cat(glue::glue("Total Rows: {size\_input\_data}"), '\n') rnd\_2 = round(length(preds\_data)/size\_input\_data, 4)

msg\_2 ="Predicted Rows: {length(preds\_data)} ({rnd\_2 \* 100}% predicted)"

cat(glue::glue(msg\_2), '\n') cat(glue::glue("Missing Values: {size\_input\_data -

length(preds\_data)}"), '\n')

rnd\_4 = round(sum(true\_data == preds\_data) / length(preds\_data), 4) msg\_4 = "Accuracy on 'Predicted Rows' using '{method}': {rnd\_4 \*

100}%"

cat(glue::glue(msg\_4), '\n')

}

The datasets are attached on the repository as .ZIP file.

list.files(dir\_wili\_2018)



For this benchmark we’ll use only the **test** data (‘x\_test.txt’ and ‘y\_test.txt’ files) and we’ll keep only the *WiLI-isocodes* that intersect with the *fastText isocodes*,



# fastText based on the smaller pre-trained model ‘lid.176.ftz’ (approx. 917 kB)

First, we’ll use the smaller pre-trained dataset,

file\_ftz = system.file("language\_identification/lid.176.ftz", package = "fastText")

dtbl\_res\_in = fastText::language\_identification(input\_obj = wili\_test\_x$V1,

pre\_trained\_language\_model\_path = file\_ftz,

dtbl\_res\_in$true\_label = wili\_test\_y$V1 # dtbl\_res\_in

isocodes\_fasttext\_subs = isocodes\_fasttext[, c(1,3)] # merge the predicted labels with the 3-letter isocodes

merg\_labels = merge(dtbl\_res\_in, isocodes\_fasttext\_subs, by.x = 'iso\_lang\_1', by.y = 'Alpha\_2')

# as.vector(colSums(is.na(merg\_labels)))

print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = merg\_labels$true\_label, preds\_data = merg\_labels$Alpha\_3\_B,

method = 'fastText (.ftz pre-trained model)')



The **accuracy** of the model was **83.05%** (on **50211** out of **50500** text extracts)

# fastText based on the bigger pre-trained model ‘lid.176.bin’ (approx. 126 MB)

The parameter setting of the **fastText::language\_identification()** function is the same as before, and the only thing that changes is the **pre\_trained\_language\_model\_path** parameter which is set to **lid.176.bin**. Assuming this file is downloaded and extracted in the **dir\_wili\_2018** directory then,

file\_bin = file.path(dir\_wili\_2018, 'lid.176.bin')

dtbl\_res\_in = fastText::language\_identification(input\_obj = wili\_test\_x$V1,

pre\_trained\_language\_model\_path = file\_bin,

dtbl\_res\_in$true\_label = wili\_test\_y$V1 # dtbl\_res\_in

isocodes\_fasttext\_subs = isocodes\_fasttext[, c(1,3)] # merge the predicted labels with the 3-letter isocodes

merg\_labels = merge(dtbl\_res\_in, isocodes\_fasttext\_subs, by.x = 'iso\_lang\_1', by.y = 'Alpha\_2')

# as.vector(colSums(is.na(merg\_labels)))

print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = merg\_labels$true\_label, preds\_data = merg\_labels$Alpha\_3\_B,

method = 'fastText (.ftz pre-trained model)')



The **accuracy** based on the bigger model was increased to **86.55%** (on **50168** out of **50500** text extracts)

# ggplot visualization of the bigger .bin model

The following plot shows the confusion matrix of the bigger .bin model. The main diagonal is dominated by the dark green color indicating higher accuracy rates,

tbl = table(merg\_labels$true\_label, merg\_labels$Alpha\_3\_B)

df = as.data.frame.table(tbl)

colnames(df) = c('country\_vert', 'country\_horiz', 'Freq') # head(df)

require(magrittr) require(dplyr) require(ggplot2)

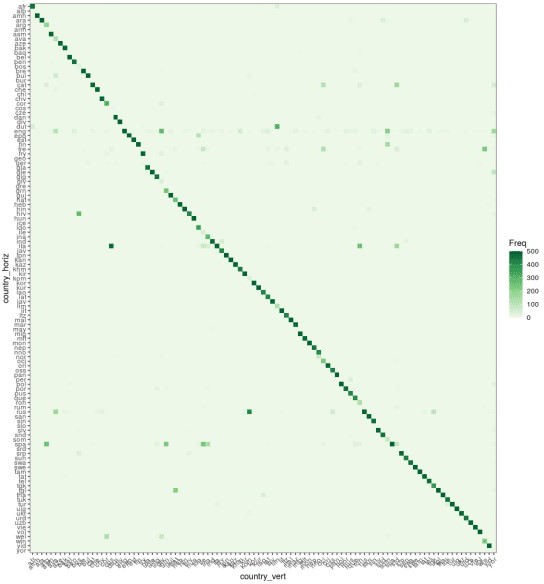
df <- df %>%

mutate(country\_vert = factor(country\_vert), # alphabetical order by default

country\_horiz = factor(country\_horiz, levels = rev(unique(country\_horiz))))

plt\_tbl = ggplot(df, aes(x=country\_vert, y=country\_horiz, fill=Freq)) + geom\_tile() + theme\_bw() + coord\_equal() + scale\_fill\_distiller(palette="Greens", direction=1) + ggplot2::theme(axis.text.x = element\_text(angle = 45, vjust = 1.0,

hjust = 1.0)) plt\_tbl



# ‘cld2’ language recognition package

The **Google’s Compact Language Detector 2** (CLD2) “ … probabilistically detects over 80 languages in Unicode UTF-8 text, either plain text or HTML/XML. For mixed-language input, CLD2 returns the top three languages found and their approximate percentages of the total text bytes (e.g. 80% English and 20% French out of 1000 bytes) …”. Based on the R package documentation, “The function ‘detect\_language()’ is vectorised and guesses the language of

each string in text or returns NA if the language could not reliably be determined.”

require(cld2)

t\_start = proc.time()

cld2\_vec = cld2::detect\_language(text = wili\_test\_x$V1, plain\_text = TRUE, lang\_code = TRUE)

cld2\_dtbl = data.table::setDT(list(Alpha\_2 = cld2\_vec)) cld2\_dtbl$true\_label = wili\_test\_y$V1

merg\_labels\_cld2 = merge(cld2\_dtbl, isocodes\_fasttext\_subs, by = 'Alpha\_2')

# as.vector(colSums(is.na(merg\_labels\_cld2)))

print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = merg\_labels\_cld2$true\_label, preds\_data = merg\_labels\_cld2$Alpha\_3\_B, method = 'cld2')



The **accuracy** of the **cld2** package is **83.13%** (on **34254** out of **50500** text extracts)

# ‘cld3’ language recognition package

The “ … Google’s Compact Language Detector 3 is a neural network model for language identification and the successor of CLD2 (available from) CRAN. This version is still experimental and uses a novell algorithm with different properties and outcomes. Based on the R package documentation, “The function detect\_language() is vectorised and guesses the language of each string in text or returns NA if the language could not reliably be determined.”

require(cld3)

t\_start = proc.time()

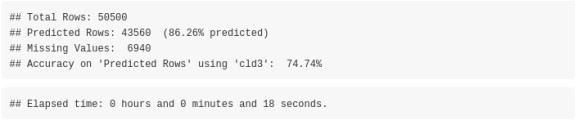
cld3\_vec = cld3::detect\_language(text = wili\_test\_x$V1)

cld3\_dtbl = data.table::setDT(list(Alpha\_2 = cld3\_vec)) cld3\_dtbl$true\_label = wili\_test\_y$V1

merg\_labels\_cld3 = merge(cld3\_dtbl, isocodes\_fasttext\_subs, by = 'Alpha\_2')

# as.vector(colSums(is.na(merg\_labels\_cld3)))

print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = merg\_labels\_cld3$true\_label, preds\_data = merg\_labels\_cld3$Alpha\_3\_B, method = 'cld3')



The **accuracy** of the **cld3** package is **74.74%** (on **43560** out of **50500** text extracts)

# Language recognition using the ‘textcat’ R package

The ‘textcat’ R package performs ‘text categorization based on n-grams’. The documentation of the package mentions: “… TextCat (https://[www.let.rug.nl/vannoord/TextCat/)](http://www.let.rug.nl/vannoord/TextCat/)) is a Perl implementation of the Cavnarand Trenkle ‘N-Gram-Based Text Categorization’ technique by Gertjan van Noord which was subsequently integrated into SpamAssassin. It provides byte

n-gram profiles for 74 ‘languages’ (more precisely, language/encoding combinations). The C library reimplementation libtextcat (https://software.wise-guys.nl/libtextcat/) adds one more non- empty profile.

‘TC\_byte\_profiles’ provides these byte profiles.

‘TC\_char\_profiles’ provides a subset of 56 character profiles obtained by converting the byte sequences to UTF-8 strings where possible.

The category ids are unchanged from the original, and give the full (English) name of the language, optionally combined the name of the encoding script. Note that ‘scots’ indicates Scots, the Germanic language variety historically spoken in Lowland Scotland and parts of Ulster, to be distinguished from Scottish Gaelic (named ‘scots\_gaelic’ in the profiles), the Celtic language variety spoken in most of the western Highlands and in the Hebrides (see https://en.wikipedia.org/wiki/Scots\_language) …”

Apart from the previous 2 mentioned TC-profiles the **ECIMCI\_profiles** (26 profiles) also exists. In my benchmark I’ll use only the *‘TC\_byte\_profile’* (75 profiles) and *‘TC\_char\_profiles’* (56 profiles) as input to the **textcat()** function to compute the country names, which by default the *textcat()* function returns. I’ll wrap the function to **parallel::mclapply()** because I observed it returns the results faster using multiple threads (in my benchmark I used 8 threads).

## using the ‘TC\_byte\_profiles’

Before proceeding lets have a look to the available profiles,

threads = parallel::detectCores() require(textcat)

names(textcat::TC\_byte\_profiles)



What we want is that the initial *lowercase isocodes* intersect with the processed

*TC\_byte\_profiles* so that the computation of the accuracy is correct,



t\_start = proc.time()

textc = as.vector(unlist(parallel::mclapply(1:length(wili\_test\_x$V1), function(x) {

textcat(x = wili\_test\_x$V1[x], p = textcat::TC\_byte\_profiles, method

= "CT")

}, mc.cores = threads)))

textc = as.vector(unlist(lapply(strsplit(textc, '-'), function(x) x[1])))

textc = trimws(textc, which = 'both')

unique(textc)



textc\_dtbl = data.table::setDT(list(Name\_tolower = textc)) textc\_dtbl$true\_label = wili\_test\_y$V1

fasttext\_isoc\_name = isocodes\_fasttext[, c(1,5)] merg\_labels\_textc = merge(textc\_dtbl, fasttext\_isoc\_name, by = 'Name\_tolower')

# as.vector(colSums(is.na(merg\_labels\_cld2)))

print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = merg\_labels\_textc$true\_label, preds\_data = merg\_labels\_textc$Alpha\_3\_B, method = 'textcat ( TC\_byte\_profiles )')



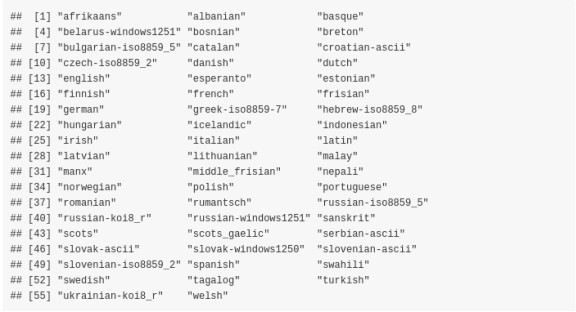
The **accuracy** of the **textcat** package using the **TC\_byte\_profiles** is **29.91%** (on **47324** out of

**50500** text extracts)

## using the ‘TC\_char\_profiles’

Again, as previously we can have a look to the available profiles,

names(textcat::TC\_char\_profiles)



What we want is that the initial *lowercase isocodes* intersect with the processed

*TC\_char\_profiles* so that the computation of the accuracy is correct,



t\_start = proc.time()

textc = as.vector(unlist(parallel::mclapply(1:length(wili\_test\_x$V1), function(x) {

textcat(x = wili\_test\_x$V1[x], p = textcat::TC\_char\_profiles, method

= "CT")

}, mc.cores = threads)))

textc = as.vector(unlist(lapply(strsplit(textc, '-'), function(x) x[1])))

textc = trimws(textc, which = 'both') unique(textc)



textc\_dtbl = data.table::setDT(list(Name\_tolower = textc)) textc\_dtbl$true\_label = wili\_test\_y$V1

fasttext\_isoc\_name = isocodes\_fasttext[, c(1,5)] merg\_labels\_textc = merge(textc\_dtbl, fasttext\_isoc\_name, by = 'Name\_tolower')

# as.vector(colSums(is.na(merg\_labels\_cld2)))

print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = merg\_labels\_textc$true\_label, preds\_data = merg\_labels\_textc$Alpha\_3\_B, method = 'textcat ( TC\_char\_profiles )')



The **accuracy** of the **textcat** package using the **TC\_char\_profiles** is **31.10%** (on **43265** out of

**50500** text extracts)

# Language recognition using the ‘franc’ R package

The R package port of Franc has no external dependencies and supports *310 languages*. All languages spoken by more than one million speakers.

The **franc()** function expects a text extract, therefore we will wrap the function with **parallel::mclapply()** as we’ve done with the *textcat* package to reduce the computation time. Moreover, we’ll set the **min\_speakers** parameter to **0** to include **all** languages known by franc (increasing the *max\_length* parameter to *4096* does not improve the accuracy for this specific data / text extracts),

require(franc)

t\_start = proc.time()

franc\_res = as.vector(unlist(parallel::mclapply(1:length(wili\_test\_x$ V1), function(x)

franc(text = wili\_test\_x$V1[x], min\_speakers = 0, min\_length = 10, max\_length = 2048

}, mc.cores = threads)))

franc\_dtbl = data.table::setDT(list(franc = franc\_res, true\_label = wili\_test\_y$V1))

# as.vector(colSums(is.na(franc\_dtbl)))

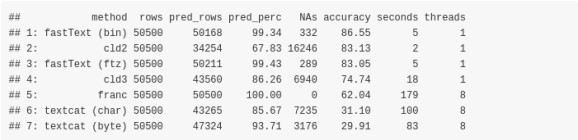
print\_accuracy(size\_input\_data = nrow(wili\_test\_y), true\_data = franc\_dtbl$true\_label, preds\_data = franc\_dtbl$franc, method = 'franc')



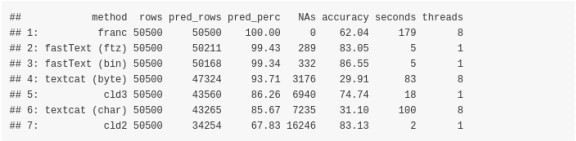
The **accuracy** of the **franc** package is **62.04%** (on **50500** out of **50500** text extracts)

# Overview datatable of all methods

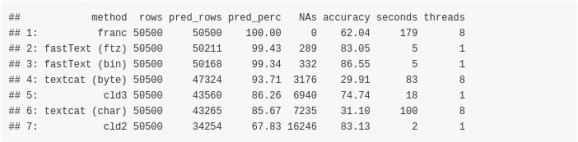
Sorted by **Accuracy** (highest better),



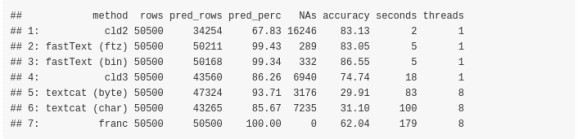
Sorted by **predicted percentage of text extracts** (highest better),



Sorted by **missing values** (lowest better),



Sorted by **computation time** (lowest better),



# Benchmark based on the Human Rights Declaration files

We can test the mentioned functions also using the **Declaration of Human Rights** text files, which are smaller in size and can give hints on potential misclassifications. As I mentioned at the beginning of this blog post the data can be downloaded from two different internet sources.

Assuming the **.zip** file is downloaded and extracted in the **dir\_wili\_2018** directory and the folder name that includes the files is named as **declaration\_human\_rights** then,

dir\_files = file.path(dir\_wili\_2018, 'declaration\_human\_rights')

lst\_files = list.files(dir\_files, full.names = T, pattern = '.pdf') decl\_dat = lapply(1:length(lst\_files), function(x) {

iter\_dat = pdftools::pdf\_text(pdf = lst\_files[x]) lang = trimws(unlist(strsplit(gsub('.pdf', '',

basename(lst\_files[x])), '\_')), which = 'both') lang = lang[length(lang)]

vec\_txt = as.vector(unlist(trimws(iter\_dat, which = 'both'))) vec\_txt = as.vector(sapply(vec\_txt, function(x) gsub('\n', '', x)))

idx\_lang = which(isocodes\_fasttext$Name\_tolower == lang) isocode\_3\_language = rep(isocodes\_fasttext$Alpha\_3\_B[idx\_lang],

length(vec\_txt))

isocode\_2\_language = rep(isocodes\_fasttext$Alpha\_2[idx\_lang], length(vec\_txt))

language = rep(lang, length(vec\_txt))

dtbl = data.table::setDT(list(isocode\_3\_language = isocode\_3\_language,

isocode\_2\_language,

dtbl

isocode\_2\_language =

language = language, text = vec\_txt))

})

decl\_dat = data.table::rbindlist(decl\_dat)



The output *data.table* includes besides the *language* also the language *isocodes* (consisting of 2 and 3 letters) and the *text extracts*. We can start to identify the language of these extracts using

the **fastText** R package and utilizing the *small* pre-trained *‘lid.176.ftz’* model,

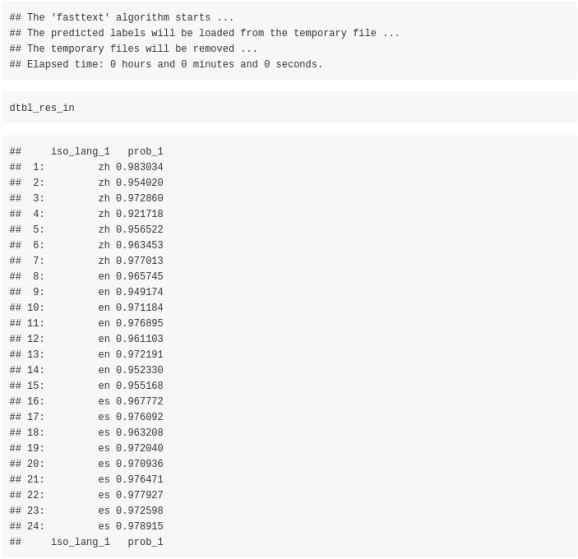
dtbl\_res\_in = fastText::language\_identification(input\_obj = decl\_dat$text,

pre\_trained\_language\_model\_path = file\_ftz,

k = 1,

th = 0.0,

threads = 1, verbose = TRUE)



To validate the results we will use the **isocode\_2\_language** column of the previous computed

**decl\_dat** data.table,

print\_accuracy(size\_input\_data = length(dtbl\_res\_in$iso\_lang\_1), true\_data = decl\_dat$isocode\_2\_language, preds\_data = dtbl\_res\_in$iso\_lang\_1,

method = 'fastText (.ftz pre-trained model)')



There are no misclassifications for the 24 input text extracts using the *fastText* algorithm. We can move to the **cld2** R package and the corresponding language identification function,

cld2\_vec = cld2::detect\_language(text = decl\_dat$text,

plain\_text = TRUE, lang\_code = TRUE)

cld2\_vec



print\_accuracy(size\_input\_data = nrow(decl\_dat),

true\_data = decl\_dat$isocode\_2\_language, preds\_data = cld2\_vec,

method = 'cld2')



There are no misclassifications for the **cld2** algorithm too. We’ll test also **cld3**,

cld3\_vec = cld3::detect\_language(text = decl\_dat$text) cld3\_vec



print\_accuracy(size\_input\_data = nrow(decl\_dat),

true\_data = decl\_dat$isocode\_2\_language, preds\_data = cld3\_vec,

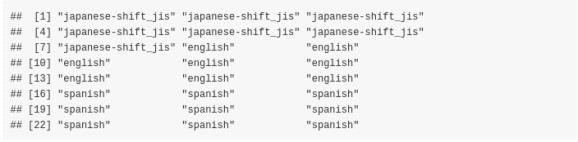
method = 'cld3')



There are no misclassifications for the **cld3** algorithm, so we move to the **textcat** R package. The **‘TC\_byte\_profiles’** include the **‘chinese-gb2312’** language characters therefore we’ll use these profiles in the **textcat** function,

textc = textcat(x = decl\_dat$text, p = textcat::TC\_byte\_profiles, method = "CT")

textc



textc = as.vector(unlist(lapply(strsplit(textc, '-'), function(x) x[1])))

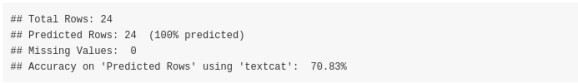
textc = trimws(textc, which = 'both') textc



print\_accuracy(size\_input\_data = nrow(decl\_dat),

true\_data = decl\_dat$language, preds\_data = textc,

method = 'textcat')



The **textcat** package misclassifies the *chinese text extracts* as *‘japanese-shift\_jis’*, therefore the accuracy *drops to approx. 70%*. Finally, we’ll test the **franc** package,

franc\_vec = as.vector(sapply(decl\_dat$text, function(x) { franc(text = x, min\_length = 10, max\_length = 2048)

}))

franc\_vec



print\_accuracy(size\_input\_data = nrow(decl\_dat),

true\_data = decl\_dat$isocode\_3\_language, preds\_data = franc\_vec,

method = 'franc')



The **franc** function identified the *chinese* text excerpts as *mandarin* chinese, therefore I personally would not consider these as misclassifications (as *mandarin* is a dialect of the chinese language). We can have an overview of the results of the different methods by illustrating the outputs in a single data.table,

dtbl\_out = decl\_dat[, 1:3]

colnames(dtbl\_out) = c('true\_y\_iso\_3', 'true\_y\_iso\_2', 'true\_y\_language')

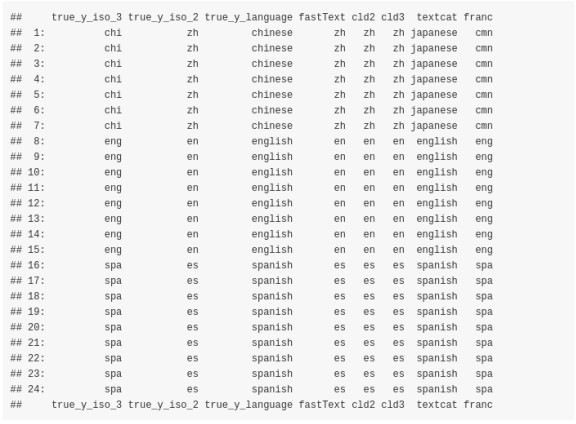
# dtbl\_out

dtbl\_preds = data.table::setDT(list(fastText = dtbl\_res\_in$iso\_lang\_1,

cld2 = cld2\_vec, cld3 = cld3\_vec, textcat = textc, franc = franc\_vec))

# dtbl\_preds

dtbl\_out = cbind(dtbl\_out, dtbl\_preds) dtbl\_out



# Comparison between ‘fastText’, ‘cl2’, ‘cl3’ and ‘franc’ for Multilingual output

Finally, we can observe the output of **fastText**, **cl2**, **cl3** and **franc** for Multilingual output (I’ll exclude the **textcat::textcat()** function, because it expects a single language per character string in the input vector).

We will first tokenize all three **Declaration of Human Rights** text files, then we will sample a specific number of words of the tokenized output and build a sentence that will be classified using the mentioned algorithms.

In order to verify the results and see how each algorithm performs we will pick **100 words** of each declaration file. Due to the fact that the **chinese** language has **ambiguous word boundaries** we will use the **stringi::stri\_split\_boundaries()** function of the **stringi** R package to extract the words of the chinese text file. The following function shows the pre-processing steps to come to the multilingual sentence,

lst\_files = list.files(dir\_files, full.names = F, pattern = '.pdf')

min\_letters\_en\_es = 3 # min. number of characters for the 'en' and 'es' languages

sample\_words = 100 # sample that many words from each tokenized file

decl\_dat = lapply(1:length(lst\_files), function(x) {

iter\_dat = pdftools::pdf\_text(pdf = file.path(dir\_files, lst\_files[x]))

dat\_txt = sapply(iter\_dat, function(y) {

if (lst\_files[x] == 'declaration\_human\_rights\_chinese.pdf') { res\_spl\_lang = stringi::stri\_split\_boundaries(str = y,

type = 'word', skip\_word\_none =

TRUE, TRUE,

TRUE)

}

else {

res\_spl\_lang = stringi::stri\_split(str = y,

skip\_word\_letter = skip\_word\_number =

regex = '[ \n,]', omit\_empty = TRUE, tokens\_only = TRUE)

}

res\_spl\_lang = trimws(res\_spl\_lang[[1]], which = 'both') idx\_empty = which(res\_spl\_lang == "")

if (length(idx\_empty) > 0) {

res\_spl\_lang = res\_spl\_lang[-idx\_empty]

}

if (!is.null(min\_letters\_en\_es) & lst\_files[x] != 'declaration\_human\_rights\_chinese.pdf') {

nchars = nchar(res\_spl\_lang)

idx\_chars = which(nchars >= min\_letters\_en\_es) if (length(idx\_chars) > 0) {

res\_spl\_lang = res\_spl\_lang[idx\_chars]

}

}

res\_spl\_lang

})

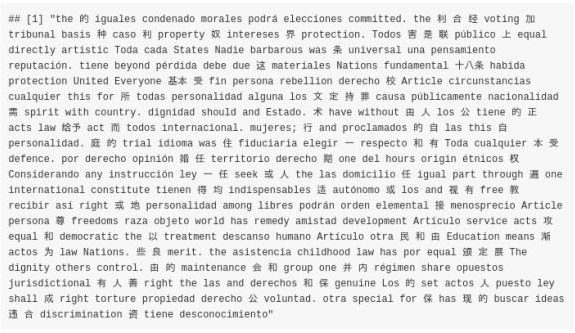
dat\_txt = as.vector(unlist(dat\_txt)) set.seed(1)

sample\_words = sample(dat\_txt, sample\_words) sample\_words

})

decl\_dat = as.vector(unlist(decl\_dat))

decl\_dat = decl\_dat[sample(1:length(decl\_dat), length(decl\_dat))] multilingual\_sentence = paste(decl\_dat, collapse = ' ' multilingual\_sentence



We deliberately mixed the words by first sampling the vector and then concatenating the tokens to a sentence. The purpose of the multilingual identification is to find out if each algorithm detects the *correct languages* assuming the *number of languages* in the text are *known beforehand*.

Imagine, you have 3 people having a conversation in a room where interchangeably a different language is spoken and this conversation is recorded by a fourth person.

num\_languages = 3

fastText Multilingual

dtbl\_multiling = fastText::language\_identification(input\_obj = multilingual\_sentence

pre\_trained\_language\_model\_path = file\_ftz,

dtbl\_multiling

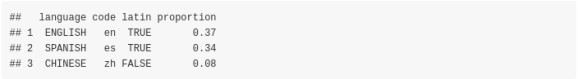
k = num\_languages, th = 0.0,

threads = 1, verbose = FALSE)



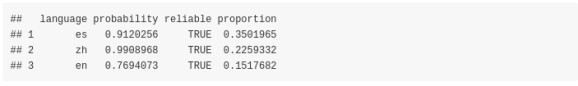
cld2 Multilingual

cld2::detect\_language\_mixed(text = multilingual\_sentence, plain\_text = TRUE)$classification



cld3 Multilingual

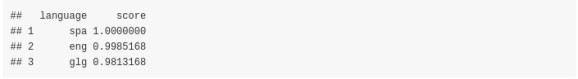
cld3::detect\_language\_mixed(text = multilingual\_sentence, size = num\_languages)



franc Multilingual

# we could use the 'whitelist' parameter but the purpose is to identify languages from unknown text

franc::franc\_all(text = multilingual\_sentence, max\_length = nchar(multilingual\_sentence) + 1)[1:num\_languages, ]



From the results one can come to the following **conclusions**:

the **cld2** *detect\_language\_mixed()* function detects the correct languages without even specifying how many languages are in the text

the **cld3** *detect\_language\_mixed()* function detects the correct languages (as *cld2*) but with the *limitation* that we have to specify the number of languages beforehand

the **fastText** function, detects 2 out of the 3 languages and the false detected one (*japanese*) seems to receive a higher probability than *chinese* (*english* is not detected at all)

the **franc** *franc\_all()* function detects correctly 2 out of the 3 languages (*english* and

*spanish*) but not chinese (the third language based on score is *Galician*)