Gender Prediction from First Names case study

Antoine Tixier

July 12, 2015

Contents

In a nutshell

This document was dynamically produced with knitr and makes use of the R programming language.
The goal is never to reinvent the wheel, so every project should start with a review of the work that has been conducted on the subject. I decided not to adopt an approach based on simple counts in very large databases (like done here for instance or here) because (1) it does not rely on Machine Learning, and (2) does not adress the problem of classifying brand new, unseen observations.
Due to lack of time, I did not consider space and time data issues. These issues are that depending on the country or time period considered, gender-indicative features can vary. Similarly, some first names that are predominantly male in one geographical area or epoch can be mostly female in another one.
The approach I adopted is that of statistical learning. I decided to compare two algorithms that are standard and broadly used in text classification (and in other applications of ML), namely Random Forests and Support Vector Machines. A first step was to review the literature in search of gender-indicative features to extract from the first names and to pass to the models. Below is a quick summary of what I came across:

- this online text book reports the last and the last two letters to be gender-discriminatory features,
- this tutorial also uses the last and the last two letters as features, and whether the last letter is a vowel,
- this paper uses Support Vector Machines with all the abovementioned features, plus the length of the word and the number of sylllables,
- finally, a good discussion on the topic and the numerous challenges associated with it is provided here.

Packages needed

```
# set up working directory (e.g., wd="C:/Users/User1/Documents")
setwd(wd)
# select packages
libs=c("tm","stringr","wordcloud","qdap","repmis","randomForest","foreach","doParallel","parallel","akima","l
# install packages
lapply(libs,install.packages,repos="http://cran.cnr.Berkeley.edu")
# load packages
lapply(libs,library,character.only=TRUE)
```

Functions developed

extractor()

This function extracts the following features from a given word: (1) first letter, (2) last letter, (3) last two letters, (4) last three letters, (5) word length in number of letters, (6) number of syllables, and (7) whether the word ends with a vowel.

```
setwd(wd)

extractor <- function(word) {
    word.vector = as.vector(unlist(strsplit(word, split = "")))
    l = length(word.vector)
    first = word.vector[1]
    last = word.vector[l]
    last.two = paste0(word.vector[(l - 1):l], collapse = "")
    last.three = paste0(word.vector[(l - 2):l], collapse = "")
    length = length(word.vector)
    numb.syl = syllable_sum(word)
    end.vowel = last %in% c("a", "e", "i", "o", "u", "y")
    return(c(first, last, last.two, last.three, length, numb.syl, end.vowel))
}
dump("extractor", "extractor.R")
source("extractor.R")
```

Examples:

```
setwd(wd)
source("extractor.R")
extractor("fergus")

[1] "f"     "s"     "us"    "gus"   "6"     "2"     "FALSE"

extractor("mariette")

[1] "m"    "e"    "te"   "tte"  "8"    "2"    "TRUE"
```

extractor.clean.wrap()

This function takes a character vector as input, cleans it, and outputs a data frame with the features (as columns) extracted by the extractor() function previously introduced for each element (each row) of the character vector.

```
setwd(wd)

extractor.clean.wrap <- function(x) {
    x.c = Corpus(VectorSource(x))
    # convert to lower case
    x.c = tm_map(x.c, tolower)
    # remove trailing whitespace
    x.c = gsub("\\s+$", "", as.character(unlist(x.c)))
    # extract features and store the results in a data frame
```

```r
    features.x = t(as.data.frame(lapply(x.c, extractor)))
    colnames(features.x) = c("first", "last", "last.two", "last.three", "length",
        "numb.syl", "end.vowel")
    rownames(features.x) = NULL
    return(list(features = features.x))
}
dump("extractor.clean.wrap", "extractor.clean.wrap.R")
source("extractor.clean.wrap.R")
```

Example:

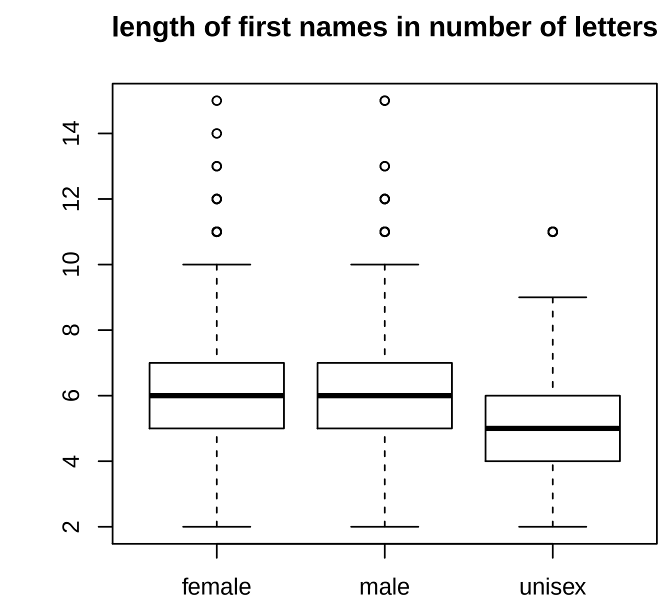
```r
setwd(wd)
invisible(lapply(c("extractor.R", "extractor.clean.wrap.R"), source))
extractor.clean.wrap(c("Romuald ", "Blanquette  "))$features

     first last last.two last.three length numb.syl end.vowel
[1,] "r"   "d"  "ld"     "ald"      "7"    "2"      "FALSE"
[2,] "b"   "e"  "te"     "tte"      "10"   "2"      "TRUE"
```

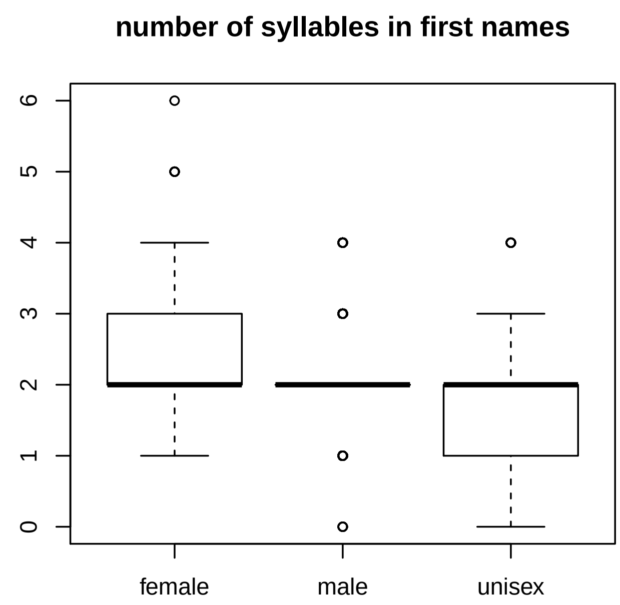
Diagnostics

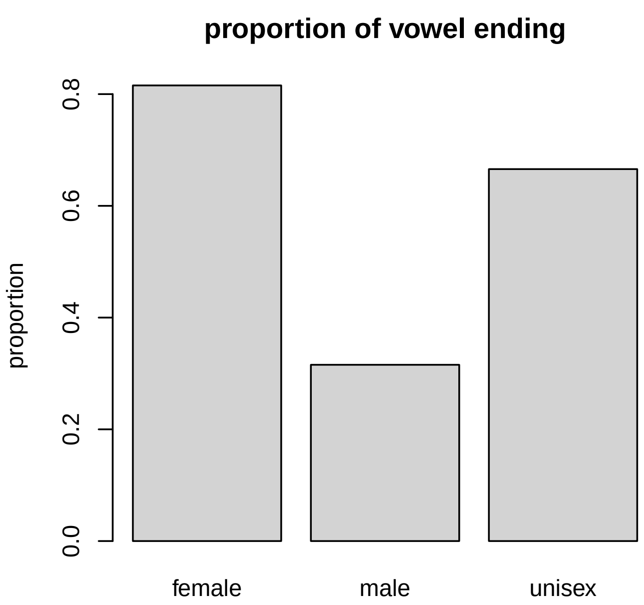
Data loading

```r
setwd(wd)
source("extractor.R")
source("extractor.clean.wrap.R")

# load data from dropbox
female = source_data("https://dl.dropboxusercontent.com/u/47464062/female.snips.csv")[,
    1]
male = source_data("https://dl.dropboxusercontent.com/u/47464062/male.snips.csv")[,
    1]

# check content
head(female)

[1] "Abagail" "Abbe"    "Abbey"   "Abbi"    "Abbie"   "Abby"

head(male)

[1] "Aaron"  "Abbey"  "Abbie"  "Abbot"  "Abbott" "Abby"

length(female)

[1] 5000

length(male)

[1] 2942

# we have quite some tricky cases
unisex = intersect(female, male)
length(unisex)

[1] 365

# to find strong gender discriminatory features, we remove the overlapping
# cases
female.strict = female[!female %in% unisex]
male.strict = male[!male %in% unisex]

# extract features for females
features.female = extractor.clean.wrap(female.strict)$features
# check that everything went OK
head(female.strict)
```

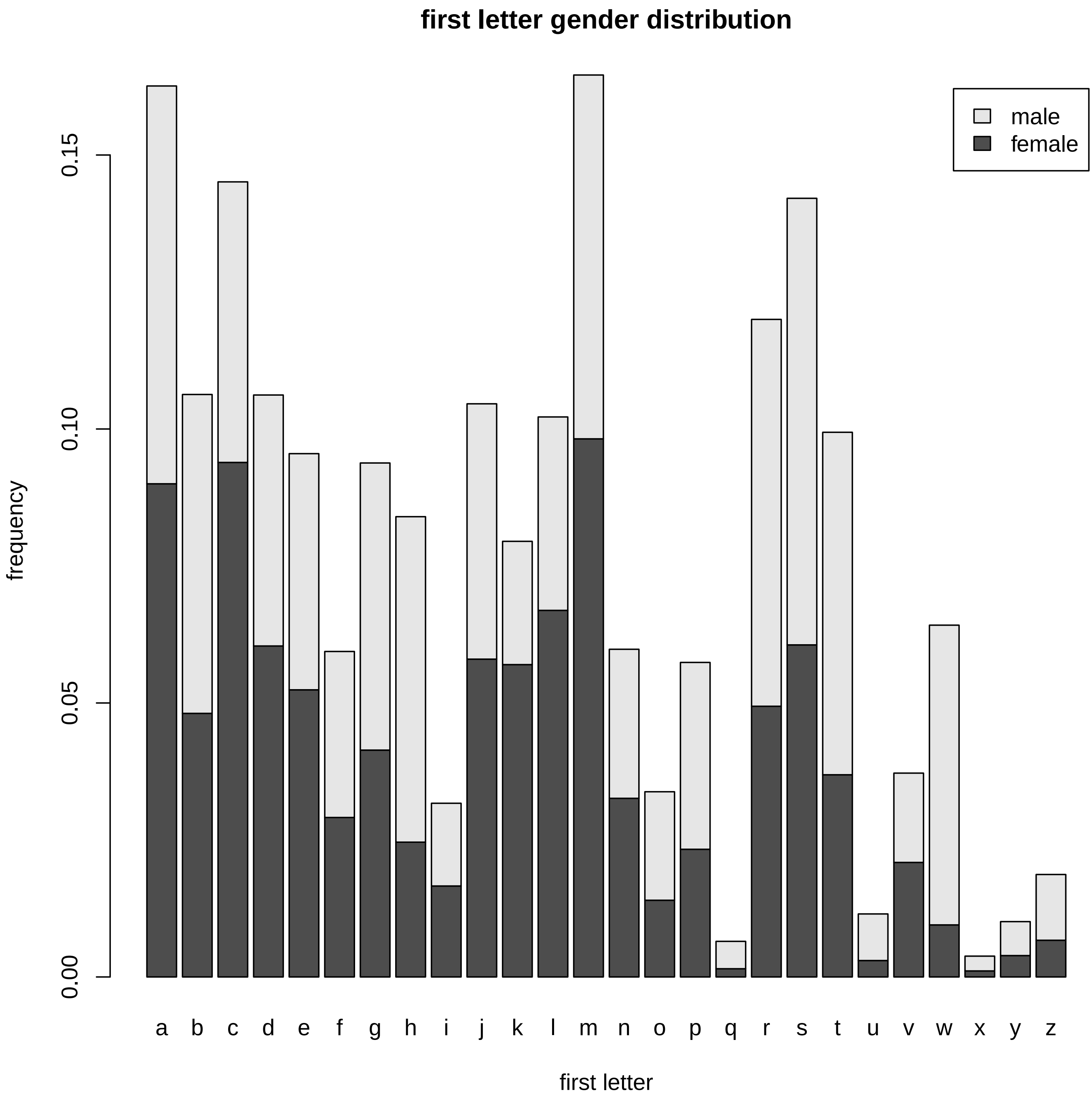
```
[1] "Abagail" "Abbe"    "Abbi"    "Abigael" "Abigail" "Abigale"

head(features.female)

     first last last.two last.three length numb.syl end.vowel
[1,] "a"   "l"  "il"     "ail"      "7"    "3"      "FALSE"
[2,] "a"   "e"  "be"     "bbe"      "4"    "2"      "TRUE"
[3,] "a"   "i"  "bi"     "bbi"      "4"    "2"      "TRUE"
[4,] "a"   "l"  "el"     "ael"      "7"    "3"      "FALSE"
[5,] "a"   "l"  "il"     "ail"      "7"    "3"      "FALSE"
[6,] "a"   "e"  "le"     "ale"      "7"    "3"      "TRUE"

# extract features for males
features.male = extractor.clean.wrap(male.strict)$features
# extract features for unisex names
features.unisex = extractor.clean.wrap(unisex)$features

par(mfrow = c(2, 2))
# comparison length of first names
lengthes = cbind(as.numeric(features.female[, "length"]), as.numeric(features.male[,
    "length"]), as.numeric(features.unisex[, "length"]))
colnames(lengthes) = c("female", "male", "unisex")
boxplot(lengthes)
title(main = c("length of first names in number of letters"))

# comparison number of syllables
n.syll = cbind(as.numeric(features.female[, "numb.syl"]), as.numeric(features.male[,
    "numb.syl"]), as.numeric(features.unisex[, "numb.syl"]))
colnames(n.syll) = c("female", "male", "unisex")
boxplot(n.syll)
title(main = c("number of syllables in first names"))

barplot(cbind(length(which(features.female[, "end.vowel"] == TRUE))/length(female.strict),
    length(which(features.male[, "end.vowel"] == TRUE))/length(male.strict),
    length(which(features.unisex[, "end.vowel"] == TRUE))/length(unisex)), names.arg = c("female",
    "male", "unisex"), col = "light grey", main = "")
title(main = "proportion of vowel ending")
title(ylab = "proportion")

# we extract frequencies for letter-based features
wordclouds.female = apply(features.female[, 1:4], 2, function(x) {
    round(summary(as.factor(x))/length(female.strict), 4)
})

wordclouds.male = apply(features.male[, 1:4], 2, function(x) {
    round(summary(as.factor(x))/length(male.strict), 4)
})

wordclouds.unisex = apply(features.unisex[, 1:4], 2, function(x) {
    round(summary(as.factor(x))/length(unisex), 4)
})

# the first letter does not seem to be a very discriminatory feature on its
# own. But, to be safe, we keep it as it may interact in a subtle way with
# other features to create strong gender signatures. And, in any way, Random
# Forest, and (to a lesser extent) Support Vector Machines, are robust to
# the inclusion of irrelevant predictors.

par(mfrow = c(1, 1))
```



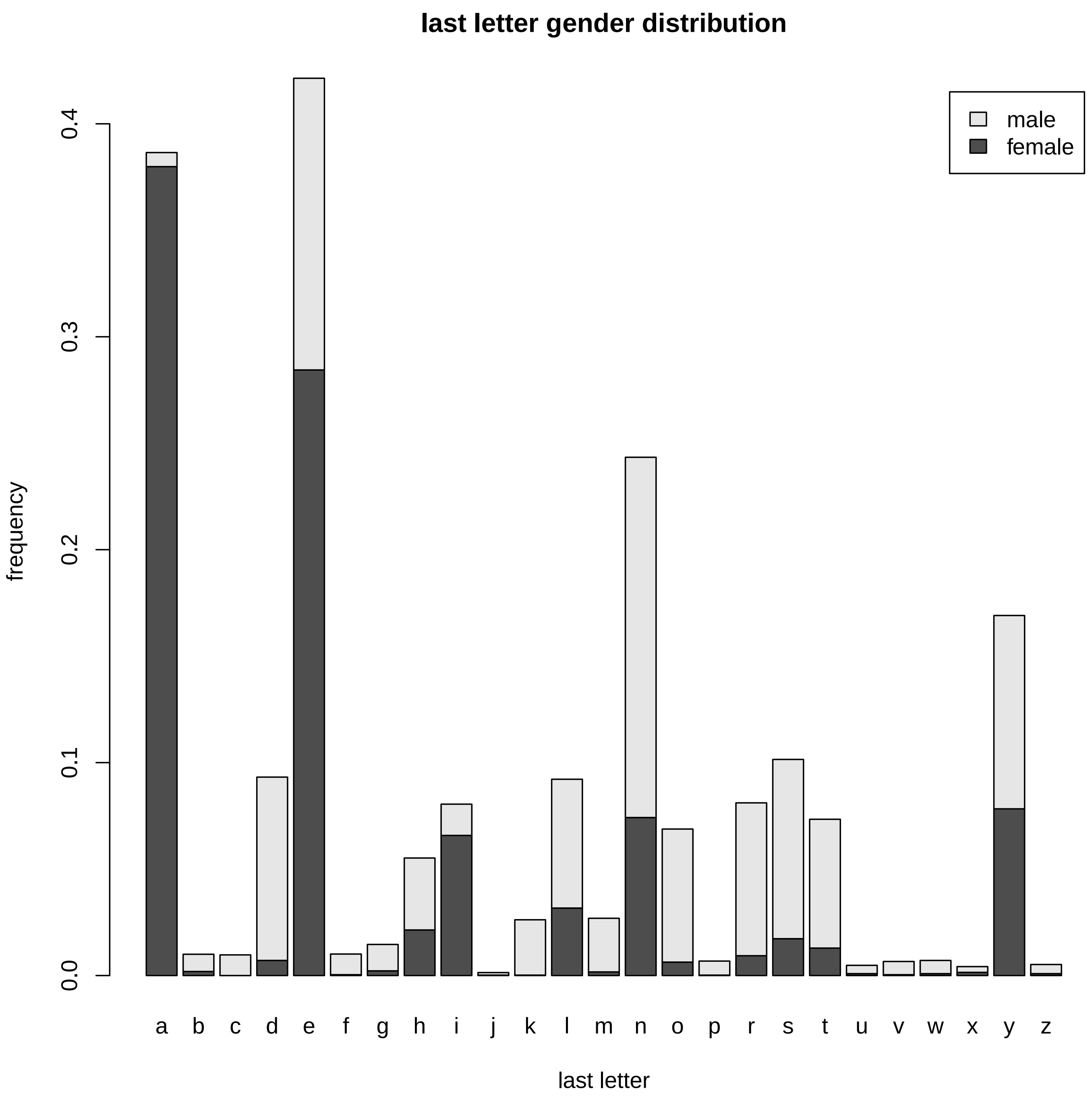





```
barplot(rbind(wordclouds.female[[1]], wordclouds.male[[1]]), legend.text = c("female",
    "male"), ylab = "frequency", xlab = "first letter")
title(main = "first letter gender distribution")
```




```r
# we need to add the 'c' for female first names to be able to compare last
# letters
wordclouds.female[[2]]["c"] = 0
wordclouds.female[[2]] = wordclouds.female[[2]][order(names(wordclouds.female[[2]]))]

barplot(rbind(wordclouds.female[[2]], wordclouds.male[[2]]), legend.text = c("female",
    "male"), ylab = "frequency", xlab = "last letter")
title(main = "last letter gender distribution")
```

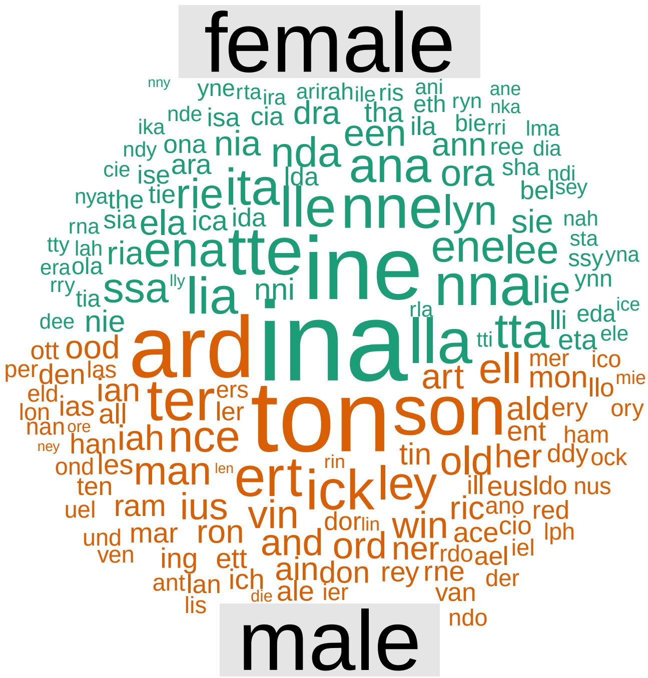



```
# the last letter seems to be strongly indicative of gender

# for the rest of the letter-based features, wordclouds are better suited
# than barplots due to the high number of categories

# we create a term document matrix where each row is a letter-based feature,
# there is one column for male and one for female, and the values in the
# cells are the frequencies of appearance of each feature

rownames.compare.last.two = unique(c(names(wordclouds.female[[3]]), names(wordclouds.male[[3]])))
last.two.tdm = matrix(nrow = length(rownames.compare.last.two), ncol = 2)
rownames(last.two.tdm) = rownames.compare.last.two
colnames(last.two.tdm) = c("female", "male")

# we now fill the tdm matrix
for (i in 1:nrow(last.two.tdm)) {
    feature = rownames(last.two.tdm)[i]
    last.two.tdm[i, "female"] = as.numeric(wordclouds.female[[3]][feature])
    last.two.tdm[i, "male"] = as.numeric(wordclouds.male[[3]][feature])
```

```r
}

# replace NA's by frequency of zero
last.two.tdm[is.na(last.two.tdm)] = 0
# remove 'Other' row
last.two.tdm = last.two.tdm[-which(rownames(last.two.tdm) == "(Other)"), ]

# we repeat the exact same process for the 'last three letters' features,
# again we can see that we have some signal here
rownames.compare.last.three = unique(c(names(wordclouds.female[[4]]), names(wordclouds.male[[4]])))
last.three.tdm = matrix(nrow = length(rownames.compare.last.three), ncol = 2)
rownames(last.three.tdm) = rownames.compare.last.three
colnames(last.three.tdm) = c("female", "male")

for (i in 1:nrow(last.three.tdm)) {
    feature = rownames(last.three.tdm)[i]
    last.three.tdm[i, "female"] = as.numeric(wordclouds.female[[4]][feature])
    last.three.tdm[i, "male"] = as.numeric(wordclouds.male[[4]][feature])
}

last.three.tdm[is.na(last.three.tdm)] = 0
last.three.tdm = last.three.tdm[-which(rownames(last.three.tdm) == "(Other)"),
    ]

par(mfrow = c(1, 2))
comparison.cloud(last.two.tdm, scale = c(4, 0.5), max.words = 300, random.order = FALSE,
    rot.per = 0)
title(main = "last two letters")

comparison.cloud(last.three.tdm, scale = c(4, 0.5), max.words = 300, random.order = FALSE,
    rot.per = 0)
title(main = "last three letters")
```

last two letters

last three letters



Prediction

Preparatory work

```
colnames.attributes = c(paste0("first=", names(wordclouds.female[[1]])), paste0("last=",
    names(wordclouds.female[[1]])), paste0("last.two=", c(names(wordclouds.female[[3]])[1:99],
    names(wordclouds.male[[3]][1:99]), names(wordclouds.unisex[[3]])[1:99])),
    paste0("last.three=", c(names(wordclouds.female[[4]])[1:99], names(wordclouds.male[[4]][1:99]),
        names(wordclouds.unisex[[4]])[1:99])), paste0("length=", as.character(2:15)),
    paste0("numb.syl=", as.character(1:6)), "end.vowel", "gender")

colnames.attributes = unique(colnames.attributes)

# we end up with 467 binary features and one target (gender)
length(colnames.attributes)

attributes = matrix(nrow = length(female.strict) + length(male.strict) + length(unisex),
```

```
    ncol = length(colnames.attributes))
colnames(attributes) = colnames.attributes
rownames(attributes) = c(female.strict, male.strict, unisex)

# populate the matrix
row.features = vector(length = 6)
for (i in 1:nrow(attributes)) {
    if (i <= length(female.strict)) {
        row = features.female[i, ]
        attributes[i, "gender"] = "female"
    } else if ((i > length(female.strict)) & (i <= (length(female.strict) + length(male.strict)))) {
        row = features.male[(i - length(female.strict)), ]
        attributes[i, "gender"] = "male"
    } else if (i > (length(female.strict) + length(male.strict))) {
        row = features.unisex[(i - (length(female.strict) + length(male.strict))),
            ]
        attributes[i, "gender"] = "unisex"
    }
    for (j in 1:length(row.features)) {
        row.features[j] = paste0(names(row)[j], "=", row[j])
    }
    index = which(colnames.attributes %in% row.features)
    # put 1 for the features that have been detected
    attributes[i, index] = 1
    # put 0 for the features that are not present
    attributes[i, setdiff(1:(ncol(attributes) - 1), index)] = 0
    if (row[7] == TRUE) {
        attributes[i, "end.vowel"] = 1
    } else {
        attributes[i, "end.vowel"] = 0
    }
}

attributes = as.data.frame(attributes)

# save data frame to save the reader some time because the loop above takes
# a couple of minutes to run
write.csv(attributes, "attributes.snips.csv")
```

```
# load the attribute data (previously created) from dropbox
attributes = source_data("https://dl.dropboxusercontent.com/u/47464062/attributes.snips.csv")

gender = as.factor(attributes[, "gender"])

attributes = attributes[, -c(1, 468)]

# we turn the columns into factors
attributes = as.data.frame(apply(attributes, 2, function(x) {
    x = factor(x, levels = c("1", "0"), labels = c("yes", "no"), ordered = FALSE)
}))

# inspect result
attributes[1:6, 458:466]

  length=14 length=15 numb.syl=1 numb.syl=2 numb.syl=3 numb.syl=4
1        no        no         no        yes         no         no
2        no        no         no        yes         no         no
3        no        no        yes         no         no         no
4        no        no         no        yes         no         no
5        no        no         no         no        yes         no
6        no        no         no        yes         no         no
  numb.syl=5 numb.syl=6 end.vowel
1         no         no       yes
2         no         no       yes
```

```
3          no          no         no
4          no          no         no
5          no          no        yes
6          no          no         no

attributes[5000:5006, 24:30]

     first=x first=y first=z last=a last=b last=c last=d
5000      no      no      no     no     no     no     no
5001      no      no      no     no     no     no     no
5002      no      no      no    yes     no     no     no
5003      no      no      no     no     no     no     no
5004      no      no      no     no     no     no     no
5005      no      no      no    yes     no     no     no
5006      no      no      no     no     no     no     no
```

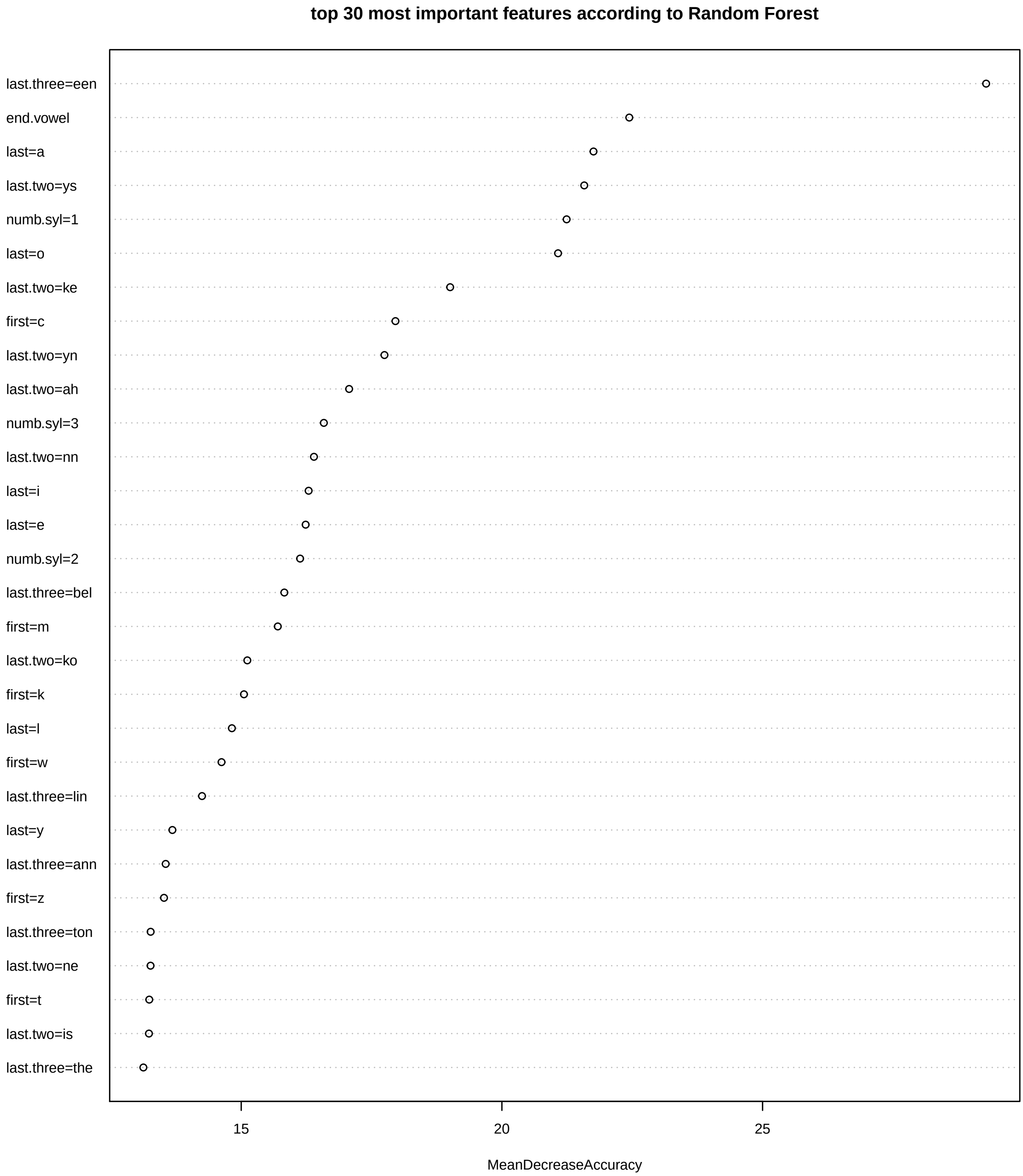
Random Forest

```
# determine best values of the n.tree and mtry parameters with a grid search
# and 8 runs of 40% out cross-validation

# we limit the grid search to affordable values in this case study due to
# time and computing resources issues

# define grid
error = expand.grid(ntree = seq(from = 201, to = 801, by = 200), mtry = c(20,
    40, 60))
error[, c("male", "female", "unisex")] = rep((1:nrow(error)) * 0, 3)

# 8 runs of leave-40%-out CV
nsim = 8

# we parallelize the loop with foreach() to speed up the process detect
# number of logical cores of the machine
nc = detectCores()

nobs = nrow(attributes)
index = 1:nobs
N40 = round(0.4 * nobs)

for (j in 1:nrow(error)) {

    # distribute the 8 runs of CV among the workers
    cl = makeCluster(nc)
    registerDoParallel(cl)

    temp = foreach(i = 1:nsim, .packages = c("randomForest"), .export = c("attributes",
        "error", "nobs", "index", "N40")) %dopar% {

        # leave 40% of the observations out as a test set
        drop = sample(index, N40, replace = FALSE)
        # keep the rest to train the model
        keep = setdiff(index, drop)

        data.train = attributes[keep, ]
        data.test = attributes[drop, ]
        Y = gender[keep]
        obs = gender[drop]

        # train the model on the kept observations
        rf = randomForest(Y ~ ., data = data.train, ntree = error[j, "ntree"],
            mtry = error[j, "mtry"], nodesize = 1)

        # test the model on the left out observations
```

```
        pred = as.character(predict(rf, newdata = data.test))

        # compute misclassification error rate for each class

        # index of new observations correctly classified
        index.correct = which(pred == obs)
        # index of new observations wrongly classified
        missed.obs = obs[setdiff(1:length(obs), index.correct)]
        # the ratios give the error rates
        round(summary(missed.obs)/summary(obs), 3)

    }  # end foreach loop

    # take the mean of the error rates for the 8 runs (and for each class) and
    # store it
    error[j, 3:5] = apply(data.frame(matrix(unlist(temp), nrow = length(temp),
        byrow = TRUE), stringsAsFactors = FALSE), 2, mean)

    # stop clusters
    stopCluster(cl)
    # monitor progress
    print(j)

}  # end outter simple loop

# despite parallelization, the above loop may still be a bit slow.
# Therefore, I am again saving the output to dropbox so the reader can
# directly load it in what follows does not have to actuallly run the above
# loop

write.csv(error, "error.RF.snips.csv")
```

```
error = source_data("https://dl.dropboxusercontent.com/u/47464062/error.RF.snips.csv")[,
    2:6]
error[, 6] = apply(error[, 3:5], 1, mean)
error[order(error[, 6], decreasing = FALSE), ]

   ntree mtry     male   female   unisex        V6
9    201   60 0.115875 0.238500 0.971625 0.4420000
11   601   60 0.112750 0.234750 0.980875 0.4427917
10   401   60 0.116875 0.231875 0.980875 0.4432083
12   801   60 0.110625 0.250000 0.979375 0.4466667
6    401   40 0.106250 0.241875 0.993375 0.4471667
7    601   40 0.103875 0.247125 0.991500 0.4475000
8    801   40 0.109250 0.246750 0.987500 0.4478333
5    201   40 0.112125 0.241125 0.991125 0.4481250
2    401   20 0.114875 0.251750 1.000000 0.4555417
1    201   20 0.106750 0.263000 0.997500 0.4557500
4    801   20 0.108500 0.259250 1.000000 0.4559167
3    601   20 0.111125 0.258500 1.000000 0.4565417

# the smallest error is attained for 201 trees and 60 variables randomly
# tried at each split
error

   ntree mtry     male   female   unisex        V6
1    201   20 0.106750 0.263000 0.997500 0.4557500
2    401   20 0.114875 0.251750 1.000000 0.4555417
3    601   20 0.111125 0.258500 1.000000 0.4565417
4    801   20 0.108500 0.259250 1.000000 0.4559167
5    201   40 0.112125 0.241125 0.991125 0.4481250
6    401   40 0.106250 0.241875 0.993375 0.4471667
7    601   40 0.103875 0.247125 0.991500 0.4475000
8    801   40 0.109250 0.246750 0.987500 0.4478333
9    201   60 0.115875 0.238500 0.971625 0.4420000
```

```
10   401   60 0.116875 0.231875 0.980875 0.4432083
11   601   60 0.112750 0.234750 0.980875 0.4427917
12   801   60 0.110625 0.250000 0.979375 0.4466667

# store values of optimal RF parameters
best.ntree = error[order(error[, 6], decreasing = FALSE), ][1, "ntree"]
best.mtry = error[order(error[, 6], decreasing = FALSE), ][1, "mtry"]

attach(attributes)

# train best RF model on full training set
Y = gender
rf = randomForest(Y ~ ., data = attributes, ntree = best.ntree, mtry = best.mtry,
    importance = TRUE)

# returns the predictions (probabilistic forecasts) for the out-of-bag
# observations
pred = predict(rf, type = "prob")

round(head(pred), 3)

  female  male unisex
1  1.000 0.000  0.000
2  1.000 0.000  0.000
3  0.173 0.765  0.062
4  0.000 1.000  0.000
5  1.000 0.000  0.000
6  0.200 0.800  0.000

as.character(head(Y))

[1] "female" "female" "male"   "male"   "female" "male"

# we can see that while the performance for male/female is decent, the
# performance for unisex names is very bad
round(head(pred[Y == "unisex", ]), 3)

    female  male unisex
16   0.662 0.247  0.091
29   0.016 0.984  0.000
72   1.000 0.000  0.000
81   0.519 0.377  0.104
102  0.338 0.649  0.014
152  0.714 0.243  0.043

as.character(head(Y[Y == "unisex"]))

[1] "unisex" "unisex" "unisex" "unisex" "unisex" "unisex"

# a very nice feature of RFs is the ability to compute variable importance
# metrics
varImpPlot(rf, type = 1, cex = 0.7, n.var = 30, main = "top 30 most important features according to Random Fo
```




```
# we see that these rankings correlate what we observed previously in the
# diagnostics section. It is also in accordance with the literature.
```

Support Vector Machines

```
# reload attributes because this time we need numerical value and not
# factors anymore
attributes = source_data("https://dl.dropboxusercontent.com/u/47464062/attributes.snips.csv")[,
    2:467]
attributes = apply(attributes, 2, function(x) {
    x = as.numeric(x)
})
```

```r
# remove the features that are always at zero (this was not necessary with
# RFs, as the irrelevant features are left aside by definition of the CART
# algorithm)
index.remove = which(apply(attributes, 2, sum) == 0)
attributes = attributes[, -index.remove]
```

```r
# after some trials and errors it seems that a Radial Basis Kernel is the
# best option

# due to time issues, we use the automatic parameter optimization
# functionality of the ksvm function, so no need for parameter tuning. It
# returns a good approximation of the best model.

# so, we just do 8 runs of leave-40%-out cross-validation to assess the
# predictive accuracy of the best model and compare with RF

cl = makeCluster(nc)
registerDoParallel(cl)

error = foreach(i = 1:nsim, .packages = c("kernlab"), .export = c("attributes",
    "index", "N40")) %dopar% {

    # leave 40% out
    drop = sample(index, N40, replace = FALSE)
    # keep the rest
    keep = setdiff(index, drop)

    data.train = attributes[keep, ]
    data.test = attributes[drop, ]
    Y = gender[keep]
    obs = gender[drop]

    # train
    svm = ksvm(Y ~ ., data = data.train, type = "spoc-svc", kernel = "rbfdot")

    # test
    pred = as.character(predict(svm, newdata = data.test))

    # error
    index.correct = which(pred == obs)
    missed.obs = obs[setdiff(1:length(obs), index.correct)]
    round(summary(missed.obs)/summary(obs), 3)
}

stopCluster(cl)

# again as a courtesy the output has been saved to dropbox
write.csv(data.frame(matrix(unlist(error), nrow = length(error), byrow = TRUE),
    stringsAsFactors = FALSE), "error.SVM.snips.csv")
```

```r
error.svm = source_data("https://dl.dropboxusercontent.com/u/47464062/error.SVM.snips.csv")[,
    2:4]
colnames(error.svm) = c("female", "male", "unisex")

error.svm

  female  male unisex
1  0.183 0.145  1.000
2  0.092 0.260  1.000
3  0.107 0.230  0.987
4  0.116 0.224  0.986
5  0.112 0.229  0.980
6  0.106 0.240  0.987
7  0.133 0.195  0.993
```

```
8  0.107 0.250  0.993

# we can see that SVMs do a slightly better job than RF here (notably with
# classifying female), but the performance for unisex remains very low
apply(error.svm, 2, mean)

  female     male   unisex
0.119500 0.221625 0.990750

# train best SVM on full training set
Y = gender
svm = ksvm(Y ~ ., data = attributes, type = "spoc-svc", kernel = "rbfdot")

Using automatic sigma estimation (sigest) for RBF or laplace kernel

# unfortunately no probabilistic forecast is available, just class
# predictions
pred = as.character(predict(svm, newdata = head(attributes)))

head(pred)

[1] "female" "female" "male"   "male"   "female" "male"

as.character(head(gender))

[1] "female" "female" "male"   "male"   "female" "male"
```

Final product

```
setwd(wd)
source("extractor.R")
source("extractor.clean.wrap.R")

# we select the SVM model over the RF model as it gave slightly best results
# the goal here is to show how the tool could be used in practice we create
# a simple function wrapping everything up:

final.product = function(words) {

    features = extractor.clean.wrap(words)$features
    newdata = matrix(nrow = nrow(features), ncol = length(colnames(attributes)))

    for (i in 1:nrow(features)) {
        row = features[i, ]
        row.features = vector()
        for (j in 1:ncol(features)) {
            row.features[j] = paste0(names(row)[j], "=", row[j])
        }
        index = which(colnames(attributes) %in% row.features)
        newdata[i, index] = 1
        newdata[i, setdiff((1:ncol(newdata)), index)] = 0
    }

    pred = as.character(predict(svm, newdata = newdata))

    for (i in 1:length(words)) {
        cat(words[i], "is", pred[i], "\n")
    }
}

dump("final.product", "final.product.R")
source("final.product.R")
```

```r
# example
final.product(c("Michael", "Matthew", "Keith", "Gus", "Pamela", "Francine",
    "Charlotte", "Alex", "Andy", "Andrew", "Jack", "Paul", "Rand"))
```

```
Michael is male
Matthew is male
Keith is female
Gus is male
Pamela is female
Francine is female
Charlotte is female
Alex is male
Andy is female
Andrew is male
Jack is male
Paul is male
Rand is male
```

Follow-up

antoine.tixier-1@colorado.edu