**The Fuzzy Logic:**

Fuzzy match is designed to match two different authors or strings from different tables. Suppose we have “Bahl,A” in table A and “Bahl, A” in table B, we are trying to say that both “Bahl, A” are actually the same person.

So, the obvious way is to match the name strings and say if the authors are the same. String matching is a complicated area and there are two ways to match a string. One way is to ask if the strings are same and the second way is to ask how much are the two strings match?

We use the second way to match the authors. We are trying to identify the percentage match between the two strings. The main reason for that is the variation in author names in the two tables. A simple human error can result in a mismatch. Another problem we might face is the data entry format in each of the different datasets. One dataset may follow a Full LastName - Middle Initial - FirstName format and other may follow LastName - Middle Initial - First Initial format. That is one of the problems associated with string matching. There is a lot of preprocessing that we need to do in order to match the strings accurately. We first need to make sure that both the strings are in the same format first.

There are two parts to author disambiguation - string matching and author matching. Let’s talk about the logic behind string matching first. There is a simple ‘stringdist’ package in R which calculates various string distances for us. The most common are 'osa','lv','dl','hamming','lcs','qgram','cosine','jaccard','jw'. For our purposes, we will use the cosine distance. It's a pretty popular way of quantifying the similarity of sequences by treating them as vectors and calculating their cosine. The output of the cosine algorithm is a distance between 0 and 1. 0 denotes perfect match and 1 is a perfect mismatch (???). You can read more about cosine similarity [here](https://blog.nishtahir.com/2015/09/20/fuzzy-string-matching-using-cosine-similarity/). The cosine similarity match will form a base of our calculations. It is apt for our analysis since it does not just give out a binary output (0/1).

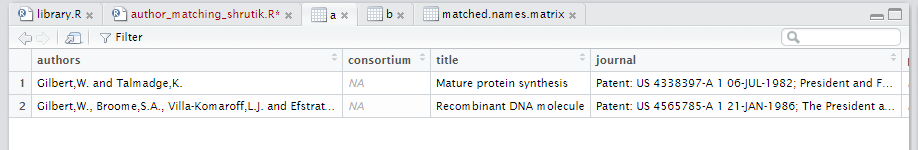
Next, we talk about the logic behind author matching. The most reliable way of matching our tables is using co-author matching. There are about 206,000 authors in the US patents alone. Out of those, only ~17% are unique. Problem arises when we try to match authors with common names. One way to solve this problem is to see if the co-authors match. The idea behind this technique is that pairs of authors collaborate more than once in different projects. To demonstrate this logic, I’d like to paste a snippet from another note which will help understand the working logic of the algorithm:

**Step 1:** Get the Author from Patent Data and same Author from Reference Data

Goal is to say if those two authors are the same.

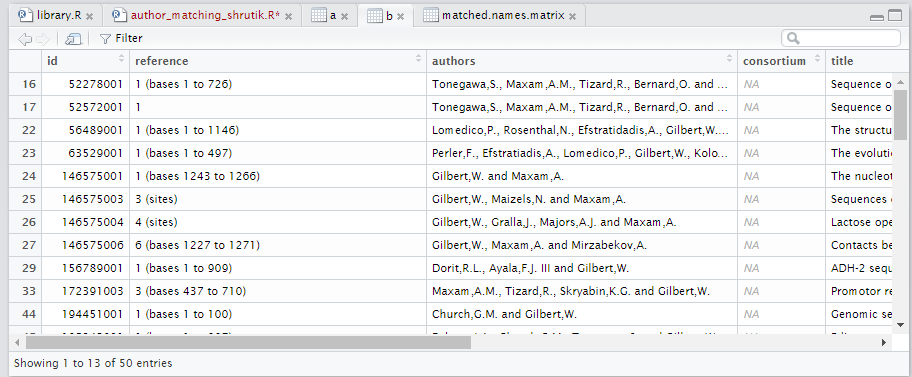
For that, we are going to see if their co-authors are same. \*Assumption\*

**a <- US\_unique[ grep("Gilbert,W", US\_unique$authors), ] # Author "Gilbert,W" from Patent table**



**b <- dbGetQuery(con,"select \* from Reference WHERE LOWER(authors) LIKE '%gilbert%';")**

**b = b[ grep("Gilbert,W",b$authors),] # Author "Gilbert,W" from Reference table**

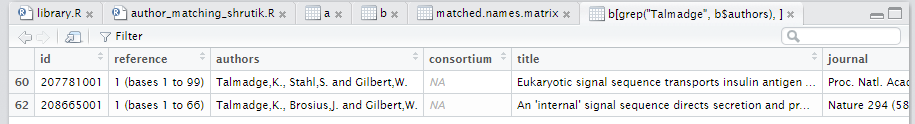


There are 2 Gilbert,W in the Patent Dataset and 50 in Reference...

Now we will take the Second author of the first row in the Patent Dataset,

Its "Talmagde,K"...Let's see if there is an author named "Talmagde,K" in the reference set

**View(b[grep("Talmadge",b$authors),])**

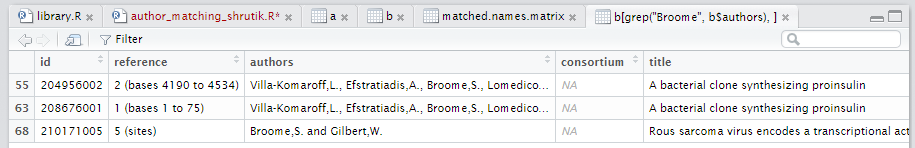


There is, in fact there are two such occurrences. Hence we can say that these Gilberts are the same.

What's more is that the Patent Year was 1982 and publication years were 1980 and 1981

Now, moving into second Gilbert,W. There are 3 co-authors in this case

**View(b[grep("Broome",b$authors),])**



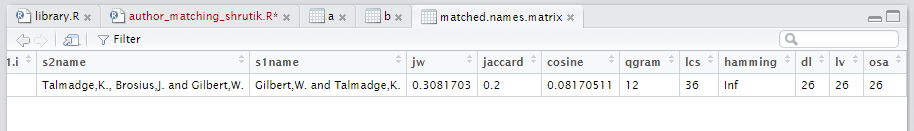
We now have 2 instances, the first one is definitely our Gilbert,W #2 since many co authors are same

In the second case, Gilbert,W is only working with Broome, so that strengthens our case.

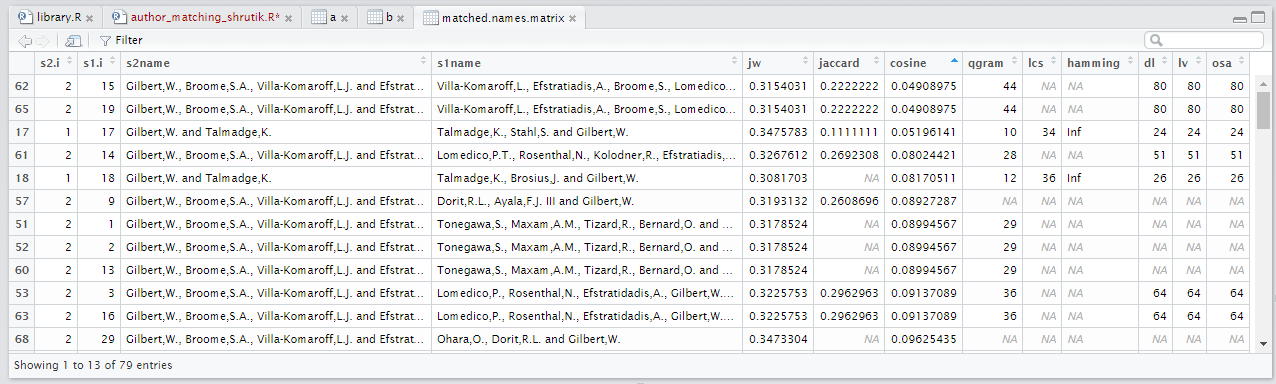
Now let's see if we can fuzzy match them..

**\*Some Fuzzy Code**\*

Case 1: Just for one instance. Check the scores for one entry in patent and compare it with one entry in ref



Case 1: Now we try it for all the instances of "Gilbert,W" in patent and compare it with all entries in ref



That is the basic outlying logic behind author disambiguation. The big change in the latest version of the algorithm is that instead of matching all the author strings, we match each author individually, this ensures that there is accuracy in what we are doing but the big downside to this is that the new program requires a lot of computational power and hence is considerably slow but more on that later.

Now I’d like to explain the code and after playing around for a bit you will be able to pass two dataframes through the algorithm and get the matched author list.

**The first Version:**

**CODE STARTS HERE**

**# This code works without data since we are defining a dummy dataset**

# Packages:

install.packages('stringdist') **# Package which makes cosine work**

library(stringdist)

library(reshape2)

library(data.table)

###Fuzzy Function:

fuzzy1 <- function(a,b)

{

#a1 = as.character(a$authors[1])

#b1 = as.character(b$authors[grep("Talmadge",b$authors)][2])

a1 = as.character(a)

b1 = as.character(b)

dist.a1b1<-adist(a1,b1, partial = TRUE, ignore.case = TRUE)

distance.methods<-c('osa','lv','dl','hamming','lcs','qgram','cosine','jaccard','jw')

dist.methods<-list()

#Fuzz 1###########

for(m in 1:length(distance.methods))

{

dist.name.enh<-matrix(NA, ncol = length(b1),nrow = length(a1))

for(i in 1:length(b1)) {

for(j in 1:length(a1)) {

dist.name.enh[j,i]<-stringdist(tolower(b1[i]),tolower(a1[j]),method = distance.methods[m])

#adist.enhance(source2.devices[i,]$name,source1.devices[j,]$name)

}

}

dist.methods[[distance.methods[m]]]<-dist.name.enh

}

match.s1.s2.enh<-NULL

for(m in 1:length(dist.methods))

{

dist.matrix<-as.matrix(dist.methods[[distance.methods[m]]])

min.name.enh<-apply(dist.matrix, 1, base::min)

for(i in 1:nrow(dist.matrix))

{

s2.i<-match(min.name.enh[i],dist.matrix[i,])

s1.i<-i

match.s1.s2.enh<-rbind(data.frame(s2.i=s2.i,s1.i=s1.i,s2name=b1[s2.i],s1name=a1[s1.i], adist=min.name.enh[i],method=distance.methods[m]),match.s1.s2.enh)

}

}

# Let's have a look at the results

matched.names.matrix<-dcast(match.s1.s2.enh,s2.i+s1.i+s2name+s1name~method, value.var = "adist")

Fuzzy1 = matched.names.matrix

return(Fuzzy1)

}

**# Creating a dummy df:######**

**# Note: you can pass any dataframe here too, just make sure to use the same structure as these ones. Dataframe a should have the columns patent\_no, Lname, Fname and seq. Dataframe b should have unique\_id, Lname, Fname and seq. Also it is very important to use the same name nomenclature for both the dataframes. Either use Full Lname, Full Fname in both or Full Lname, Fname initial on both, this will define the accuracy of the algorithm.**

**#patentref**

a = data.frame( patent\_no = c(1,1,1,2,3,4,5,6)

,Lname = c("roberts","evans","george","yu","bahl","stevens","Alyssia","Cecelia")

,Fname = c("A","L","A","F","G","M","A","A")

,seq = c(1,2,3,1,1,1,1,1))

#reference

b = data.frame( unique\_id = c(1,1,2,3,4,5,6,7)

,Lname = c("Robert","Evans","micheal","watson","baxter","stevens","Elissa","Cecily")

,Fname = c("A","L","A","F","G","M","A","A")

,seq = c(1,2,1,1,1,1,1,1))

**# Then convert the dataframes to tables since tables have better processing speeds in R and have a better memory utilization. It may not matter with a small dataset but with over 10k rows, this little trick will come in very handy.**

a = data.table(a)

b = data.table(b)

**# Structure for final author list:**

merged\_author\_list\_temp = data.frame( patentAuthorLname = as.character(a$Lname[1])

,patentAuthorFname = as.character(a$Fname[1])

,refAuthorLname = as.character(b$Lname[1])

,refentAuthorFname = as.character(b$Fname[1])

,patentNumber = a$patent\_no[1]

,unique\_id = b$unique\_id[1]

)

**# Here we define the final list where we will put all the matched authors and this will form a connection bridge between the two tables.**

final\_list = merged\_author\_list\_temp[0,]

final\_list = data.table(final\_list)

**# The main loop:**

**# This loop controls the row flow of the main dataset. Suppose dataframe a has 100 rows then this loop will run 100 times.**

for(i in 1:length(a$patent\_no))

{

print(i) # Necessary only if you want to keep track of the progress.

**# First, we will take the main author of the Patent in dataframe a, and match it’s last name with all the last names pn dataframe b.**

argument1 = a$Lname[i]

argument2 = b$Lname[i]

result = fuzzy1(argument1,argument2) # First result

df = result[0,] # For copying the df structure

**# For loop to check the matches between L\_name in 1st table with all L\_names in 2nd table**

**# Here, we will loop through the entire length of dataframe b and try to match the last names with the current last name in dataframe a.**

for(j in 1:length(b$unique\_id)) # Loops through b

{

argument2 = b$Lname[j]

result = fuzzy1(argument1,argument2) # Stores all the cosine values of all the matches

df = rbind(df,result)

}

**# Now, since we only want similar last names:**

**# Once we have all the values, we will filter them out to keep only the similar last names. Due to the errors in names and other problems we will set a threshold of 0.2 in this case. This means that we will pass all the author names which are somewhat similar too. For example, a threshold value of 0.2 will pass George and georg as same authors. You can change the value depending on the requirement.**

current\_df = df[df$cosine<0.2,] # Filtering process.

**# Now, lets check if the first initial matches:**

**# once we know that the last names are similar, we can move on to the first names/initials. We follow the same process to match the first initials.**

firstInitial\_table1 = a$Fname[a$Lname == as.character(current\_df$s1name[1])]

firstInitial\_table2 = b$Fname[b$Lname == as.character(current\_df$s2name[1])]

# Fuzzy match for Fnames having the same Lnames

result\_firstInitial\_1 = fuzzy1(firstInitial\_table1,firstInitial\_table2)

**# Now, if the match is good enough and there is just one entry in the table, we can say that the two authors from diff table are the same. Only one entry in the table means that there is an unique match and we do not need to match for co-authors. Here’s how we specify that in R:**

if(!is.na(result\_firstInitial\_1$cosine[1])&result\_firstInitial\_1$cosine[1]<0.2& length(result\_firstInitial\_1$cosine)==1)

{

**# Let's create a final merged author list, all the final matched authors will go into this list**

**# Make sure to get the patent number and unique\_id directly from the original dataframes. In fact, that is the only way to retrieve the ids, since we are matching just the strings, we need to use that to trace back and get the unique\_id and patent number.**

merged\_author\_list = data.frame( patentAuthorLname = as.character(current\_df$s1name[1])

,patentAuthorFname = as.character(result\_firstInitial\_1$s1name[1])

,refAuthorLname = as.character(current\_df$s2name[1])

,refentAuthorFname = as.character(result\_firstInitial\_1$s2name[1])

,patentNumber = a$patent\_no[a$Fname==as.character(result\_firstInitial\_1$s1name[1]) &

a$Lname==as.character(current\_df$s1name[1])]

,unique\_id = b$unique\_id[b$Fname==as.character(result\_firstInitial\_1$s2name[1]) &

b$Lname==as.character(current\_df$s2name[1])]

)

l = list(final\_list,merged\_author\_list)

final\_list = rbindlist(l)

**# Another important trick. Be sure to use list and rbindlist() for joining the rows since it is much faster than the original rbind(). Also, we are producing the final\_list on the go using the rbindlist(), this means that we can reuse the same variables for each loop. For example, in the previous lines we use merged\_author\_list, which we can use for all the iterations, just reallocating the same memory for every use, this saves a ton of space.**

}

**# Now we get into the co-authors. This else represents that there are more than one author(s) with a similar lastname/first initial combination, so we move on to the co-authors.**

else **#else if the same combination appears twice, we check for co-authors**

{

**# To get the co-author list, we simply take the authors with the same patent numbers or unique\_ids**

**# To get all the authors from table1 with same patent number**

currentPatentNumber = a$patent\_no[a$Lname == as.character(argument1)]

argument1\_df = a[a$patent\_no == currentPatentNumber,]

argument1 = argument1\_df$Lname[argument1\_df$Lname != as.character(argument1)][1]

**# Same with table 2**

currentUnique\_id = b$unique\_id[b$Lname == as.character(argument2)]

argument2\_df = b[b$unique\_id == currentUnique\_id,]

argument2 = argument2\_df$Lname[argument2\_df$Lname != as.character(argument2)][1]

result1 = fuzzy1(argument1,argument2)

df1 = result1[0,] **# Again, For copying the df structure**

**# Next step is similar to the previous ones. We match the lastnames, then firstnames and if those are similar and have more than one entry then we move onto the third author match and so on**

for(k in 1:length(argument2\_df$unique\_id))

{

argument2 = argument2\_df$Lname[k]

result1 = fuzzy1(argument1,argument2)

df1 = rbind(df1,result1)

}

current\_df = df1[df1$cosine<0.1,]

firstInitial\_table1=argument1\_df$Fname[argument1\_df$Lname==as.character(current\_df$s1name[1])]

firstInitial\_table2=argument2\_df$Fname[argument2\_df$Lname==as.character(current\_df$s2name[1])]

result\_firstInitial\_1 = fuzzy1(firstInitial\_table1,firstInitial\_table2)

if(!is.na(result\_firstInitial\_1$cosine[1])&result\_firstInitial\_1$cosine[1]<0.1 &

length(result\_firstInitial\_1$cosine) == 1)

{

#print ("SECOND IF!!!") # for checking purposes

merged\_author\_list1 = data.frame( patentAuthorLname = as.character(current\_df$s1name[1])

,patentAuthorFname = as.character(result\_firstInitial\_1$s1name[1])

,refAuthorLname = as.character(current\_df$s2name[1])

,refentAuthorFname = as.character(result\_firstInitial\_1$s2name[1])

,patentNumber = a$patent\_no[a$Fname==as.character(result\_firstInitial\_1$s1name[1]) &

a$Lname==as.character(current\_df$s1name[1])]

,unique\_id = b$unique\_id[b$Fname==as.character(result\_firstInitial\_1$s2name[1]) &

b$Lname==as.character(current\_df$s2name[1])]

)

l1 = list(final\_list,merged\_author\_list1)

final\_list = rbindlist(l1)

}

else

{

print("TBD") # this is where the third author match is done, which was included in the later versions.

}

}**#else end**

} **# Main for loop end**

**CODE ENDS HERE**

That was the first iteration of code which works but the problem with that is that it takes long to run. So a few bottlenecks identified were defining a separate fuzzy function and calculating all the distances when we only need cosine. Also, we convert the data frames to data tables for faster computation, then we use rbindlist instead of rbind cause of memory efficiency. So, in the second version, I put everything into functions.

There are three main functions in the second version. The first function is a small one and is used just to copy the data structure and get input data for the next function. The second function matches the Last names and the third and final function matches for co-authors. Ideally, we wanted to do all of this using tapply but using tapply with nested loops is a tricky area. To add on to that, we wanted to parse data from one function to another and tapply doesn’t support that.

Versions 2 and 3 of the algorithm are fairly easy to understand if you get the first one, I won’t explain it here but the code is up on github with comments.

Useful R Tricks:

Now, I’ll try to list down all the little and big chunks of useful code which I feel will help you understand the data better. Having worked with the same data for so long, you tend to underestimate even some of the complex tasks.

**Starting with pulling the data from the MySQL workbench:**

library(dbConnect)

con <- dbConnect(MySQL(),

user="**YOUR\_USERNAME**",

password="**YOUR\_PASSWORD**",

dbname="genbank",

host="metadatalab.syr.edu")

**#then simple use a select query and populate your dataframe**

journal\_field <- dbGetQuery(con,"select \* from ReferencePatent LIMIT 200;")

**# You can also change your database with an additional dbname parameter in dbConnect()**

con\_write <- dbConnect(MySQL(),

user="**YOUR\_USERNAME**",

password="**YOUR\_PASSWORD**",

host="metadatalab.syr.edu",

dbname="test")

**#To write data in a MySql table:**

dbWriteTable(con\_write, "**TABLENAME\_IN\_MySQL**", **YOUR\_DF\_IN\_R**)

**# A way to see all tables directly from R, this way you don’t have to go to the workbench every time.**

table\_names = dbGetQuery(con\_write,"SHOW TABLES FROM test;")

**Separating the Journal Field in ReferencePatent:**

It was a bit tricky to separate the journal field, the main logic here was to replace the first space with some unique identifier like “A!&%DHBsdbA” and then use that as a delimiter to separate out columns. You can find the documentation [here](https://www.evernote.com/shard/s229/sh/3a8dcd25-23b3-42b4-b2ba-da228dccefb3/a0fcfa6a9c9c919da35934c4a612d5a1) and the code [here](https://github.com/jnqin86/collabnetwork/blob/genbank_2016_shrutik/seperate_journals.R).

**If you want to remove an object from memory which is eating up a lot of space but you want to keep all the other objects and dataframes:**

rm(OBJECT)