

03b CSI online typing: Automatic answer classification

Kirsten Stark

3/30/2021

Load packages

```
rm(list = ls())

library(tidyr)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##   filter, lag

## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union

library(stringr)
library(stringdist)

##
## Attaching package: 'stringdist'

## The following object is masked from 'package:tidyr':
##
##   extract

options( "encoding" = "UTF-8" )
```

Load data

```
# input
input = "data_long_final.csv"
```

```

# input synonym/alternative naming list
alternatives = "naming_alternatives.csv"

# load data
df <- read.csv(here::here("data", "transient_data_files", input))

# load alternatives
alternatives <- read.csv(here::here("data", "supplementary_info", alternatives),
                        sep = ";")

```

Define functions

1) Function to delete of last character typed words if those are space or enter

This function checks whether the last character(s) of the word entries is a space or is the typed word “Enter”, and if so, deletes the last/the last five characters. If the last character(s) are neither of both, the word remains unchanged. The function can be used within dplyr’s mutate function. Additionally, the function has the option to delete an alternative ending, while keeping " " and “Enter” at the end of the word

```

delete_ending <- function(word, ending = NA) {
  if(is.na(ending)) {
    # if no custom ending is given, delete " " and "Enter" at the end
    case_when(str_ends(word, " ") ~ str_sub(word, end=str_length(word)-str_length(" ")),
              str_ends(word, "Enter") ~ str_sub(word, end =str_length(word)-str_length("Enter")),
              TRUE ~ word)
  } else{
    # else delete custom ending
    case_when(str_ends(word, ending) ~ str_sub(word, end=str_length(word)-str_length(ending)),
              TRUE ~ word)
  }
}

```

2) Replace special keys (e.g. backspace, shift, etc.) by other characters (e.g. numbers)

Function takes as entries the word entries, the keys to be changed, and the characters they should be replaced with.

```

replace_special_chars <- function(input, oldnames, newnames) {
  if(length(oldnames) != length(newnames)){
    print("Your oldname/newname vectors don't have the same length. Please correct!")
    stop()
  }
  for (i in 1:length(input)) {
    for (j in 1:length(oldnames)) {
      # loop through all input values and all special characters and replaces all special characters by
      input[i] <- str_replace_all(input[i], pattern = oldnames[j],
                                replacement = as.character(newnames[j]))
    }
    if( i == 1) {
      print(paste0("The pattern ", oldnames[j],
                  " has been replaced by the pattern ",
                  newnames[j], ".", sep = ""))}
  }
}

```

```

    }}
    return(input)
}

```

3) Function that computes the final words by applying all backspaces

The function takes as input the word entries and, optionally, the backspace identifier.

```

replace_backspace <- function(input, backspace = "Backspace") {
  for(i in 1:length(input)) {
    # loop through all word entries and count number of backspaces in the current word
    backspaces <- str_locate_all(input[i], backspace)[[1]]
    for(j in 1:nrow(backspaces)){
      # loop as many times as there are backspaces
      # for the current first backspace, delete backspace and the preceding character
      input[i] <- sub(str_c(".{1}",backspace), "", input[i])
    }
  }
  return(input)
}

```

4) Function that computes the fuzzy string matching

Calculate stringdistance between (backspace corrected) input word and item/alternative namings, and select the “best match”, i.e. the item/alternative with the lowest distance and the first letter being correct.

The default method is the Jaro distance (Jaro-Winkler distance (“jw”) with $p = 0$), but other methods, of the stringdist function (van der Loo, 2014) are theoretically possible as well, but further options of the stringdist function might be necessary to adapt as well.

```

# alternatives should be a dataframe with at least one column called item (same entry for as many alter

calculate_stringdist <- function(word, stims, alternatives =
                                alternatives, method = "jw", p = 0,
                                weight = c(1,1,1,1), q = 1,
                                firstlettercorrect = TRUE) {

  # input check
  if(length(word) != length(stims)){
    print("Your typed words and item vectors don't have the
          same length. Please correct!")
    stop()
  }

  # Compute string distance between word entry and item
  # using given method
  stringdistance <- stringdist(toupper(word),
                               toupper(stims), method = method,
                               p = p, weight = weight, q = q)

  # placeholders that will be filled in the for-loop
  bestmatch <- stims
  dist <- 100

```

```

# convert all entries to upper
word <- toupper(word)
stims <- toupper(stims)
alternatives$item <- toupper(alternatives$item)
alternatives$alternatives <- toupper(alternatives$alternatives)

# loop through all word entries
for(i in 1:length(word)){
  # loop only if string distance to item is not already perfect
  # and is distance is not NA (meaning the word is NA)
  if(stringdistance[i] != 0 & !is.na(stringdistance[i])) {
    # curritem <- alternatives %>% filter(item == stims[i]) %>%
    #   mutate(dist = stringdist(word[i], alternatives,
    #     method = method, p = p, q = q,
    #     weight = weight)) %>%
    #   filter(substring(word[i],1,1) ==
    #     substring(alternatives,1,1)) %>%
    #   slice(which.min(dist))
    # if(nrow(curritem) != 0 &
    #   curritem$dist[1] < stringdistance[i]){
    #   stringdistance[i] <- curritem$dist[1]
    #   bestmatch[i] <- curritem$alternatives[1]
    # }}}

    # filter "alternatives" df for alternatives of current item
    curritem <- alternatives %>% filter(item == stims[i])
    # check whether current alternative column is not empty
    if(nrow(curritem) != 0) {
      for(j in 1:nrow(curritem)) {
        currentalternative <- curritem$alternatives[j]
        dist <- stringdist(word[i], currentalternative,
          method = method, p = p, q = q, weight = weight)
        # compare the current string distance to the best
        # distance so far
        # for firstlettercorrect = TRUE
        if(dist < stringdistance[i] &
          firstlettercorrect == TRUE &
          substring(word[i],1,1) ==
          substring(currentalternative,1,1)) {
          stringdistance[i] <- dist
          bestmatch[i] <- currentalternative
          dist <- 100
        } else if (dist < stringdistance[i]) {
          # for firstlettercorrect = FALSE
          stringdistance[i] <- dist
          bestmatch[i] <- currentalternative
          dist <- 100
        }
      }
    }
  }
}

distancebest <- cbind(stringdistance, bestmatch)
return(distancebest)
}

```

5) Function that classifies the word entries

Function that classifies the word entries for correctness and different typing errors.

```
case_character_type <- function(word, item, wordcorrected,
                                distance, bestmatch, d) {
  case_when(

    # correct answers: participants typed exactly the correct word,
    # with space or enter at the end
    toupper(word) == toupper(item) |
    toupper(word) == toupper(str_c(item, " ")) |
    toupper(word) == toupper(str_c(item, "Enter")) ~ "correct",

    # correctedto correct: participants corrected their entry to the correct
    # word using "Backspace"
    (toupper(wordcorrected) == toupper(item) |
     toupper(wordcorrected) == toupper(str_c(item, " ")) |
     toupper(wordcorrected) == toupper(str_c(item, "Enter"))) &
     substring(wordcorrected,1,1) == substring(word,1,1) &
     substring(word,2, 10) != "Backspace" ~ "correctedtocorrect",

    # approx_correct: the approximately correct and best fitting word is the actual item
    # distance limits needs to be set
    (distance < d) & toupper(item) == toupper(bestmatch) &
     toupper(substring(wordcorrected,1,1)) ==
     toupper(substring(bestmatch,1,1)) &
     toupper(substring(word, 1,1)) ==
     toupper(substring(bestmatch,1,1)) &
     substring(word,2, 10) != "Backspace" ~ "approx_correct",

    # alternative: alternative was typed correctly
    (distance == 0) & (toupper(word) == toupper(wordcorrected) |
     toupper(str_c(word, " ")) == toupper(wordcorrected) |
     toupper(str_c(word, "Enter")) == toupper(wordcorrected)) &
     toupper(substring(word,1,1)) ==
     toupper(substring(bestmatch,1,1)) ~ "alternative",

    # alternative_corrected: alternative typed correctly after backspace correction
    (distance == 0) & toupper(word) != toupper(wordcorrected) &
     toupper(str_c(word, " ")) != toupper(wordcorrected) &
     toupper(str_c(word, "Enter")) != toupper(wordcorrected) &
     toupper(substring(word, 1,1)) ==
     toupper(substring(bestmatch,1,1)) &
     substring(word,2, 10) != "Backspace" ~
     "alternative_corrected",

    # approx_alternative: distance limit needs to be set
    (distance < d) & distance != 0 &
     toupper(substring(wordcorrected,1,1)) ==
     toupper(substring(bestmatch,1,1)) &
     toupper(substring(word, 1,1)) ==
     toupper(substring(bestmatch,1,1)) &
     substring(word,2, 10) != "Backspace" ~ "approx_alternative",
```

```

# backspace_space_enter: participants started by typing backspace, space,
# enter, or capslock
str_starts(word,"Backspace") |
  str_starts(word," ") |
  str_starts(word,"CapsLock") |
  str_starts(word,"Enter") ~ "backspace_space_enter",

# shift_start: participants started by pressing the shift key
str_starts(word,"Shift") ~ "shift_start",

# isna: participants didn't enter anything
is.na(word) ~ "isna",

# distance-based error
(distance >= d) &
  toupper(substring(wordcorrected,1,1)) ==
  toupper(substring(bestmatch,1,1)) &
  toupper(substring(word, 1,1)) ==
  toupper(substring(bestmatch,1,1)) &
  substring(word,2, 10) != "Backspace" ~ "distance_based_error",

# first letter-based error
(distance < d) &
  (toupper(substring(wordcorrected,1,1)) !=
  toupper(substring(bestmatch,1,1)) |
  toupper(substring(word, 1,1)) !=
  toupper(substring(bestmatch,1,1)) |
  substring(word,2, 10) == "Backspace") ~ "first_letter_error",

# are all answers classified
TRUE ~ "not_correct" )
}

```

Preprocess data, applying functions

1) Clean word ending

By deleting the last character(s) of typed words if those are space or enter keys. (Alternatively, the function also takes custom endings that should be deleted.)

As entries, the `delete_ending` function takes the column with the word entries and, optionally, a custom ending. We can repeat applying this function if we want to keep deleting if Enter or space is repeated several times at the end of the word. The while loops stops as soon as none of the words has a space or Enter (or custom ending) at the end. (In our case, this changes only the ending of three words)

```

#df2 <- df %>% mutate(word.c = delete_ending(df$word))
isnotequal <- 1
df$word.c = currentupdate = df$word
while (isnotequal > 0) {
  df <- df %>% mutate(word.c = delete_ending(df$word.c))
  isnotequal <- sum(currentupdate != df$word.c, na.rm = TRUE)
  currentupdate <- df$word.c
}

```

```
# df2 <- df %>% mutate(word.c = delete_ending(df$word, ending = " "))
# df2 <- df %>% mutate(word.c = delete_ending(df$word.c, ending = "Enter"))
# sum(df$word.c != df2$word.c, na.rm = T)
# df2$word[df$word.endc != df2$word.c & !is.na(df$word)]
```

2) Replace special characters

Special characters such as Enter and Backspace are written as entire words. We want to replace these with identifiable numbers.

```
oldnames <- c("Enter", "CapsLock", "Shift", "ArrowLeft", "ArrowRight", "Backspace", "Control")
newnames <- c("1", "2", "3", "4", "5", "6", "7")
df$word.c <- replace_special_chars(input = df$word.c, oldnames = oldnames, newnames = newnames)
```

```
## [1] "The pattern Enter has been replaced by the pattern 1."
## [1] "The pattern CapsLock has been replaced by the pattern 2."
## [1] "The pattern Shift has been replaced by the pattern 3."
## [1] "The pattern ArrowLeft has been replaced by the pattern 4."
## [1] "The pattern ArrowRight has been replaced by the pattern 5."
## [1] "The pattern Backspace has been replaced by the pattern 6."
## [1] "The pattern Control has been replaced by the pattern 7."
```

3) Compute finally submitted words by applying all backspaces

Function takes as input the word entries and, optionally, the backspace identifier.

```
#df$word.c[1:200]
#df$word.cb <- replace_backspace(df$word.c, backspace = "Backspace")
df$word.c <- replace_backspace(df$word.c, backspace = "6")
#df$word.cb[1:200]
```

4) Compute stringdist between word entries and items/alternatives

Compute Jaro distance

```
#stringdist(toupper(df$word.c2[1:200]), toupper(df$item[1:200]), method = "jw")
tictoc::tic()
output <- calculate_stringdist(word = df$word.c, stims = df$item,
                              alternatives = alternatives,
                              method = "jw", p = 0,
                              firstlettercorrect = TRUE)
tictoc::toc()
```

```
## 1.674 sec elapsed
```

```
df$jaro <- output[,1]
df$bestmatch_jaro <- output[,2]
#df$jaro[1:200]
```

5) Classify word entries

```
df2 <- df %>%
  mutate(answer_auto_jaro = case_character_type(word, item,
    word.c, jaro, bestmatch_jaro, d = 0.3))
```

```
df2 <- df2 %>%
  mutate(correct_auto_jaro = case_when(
    answer_auto_jaro == "correct" ~ 1,
    answer_auto_jaro == "correctedtocorrect" ~ 1,
    answer_auto_jaro == "approx_correct" ~ 1,
    answer_auto_jaro == "alternative" ~ 1,
    answer_auto_jaro == "alternative_corrected" ~ 1,
    answer_auto_jaro == "approx_alternative" ~ 1,
    TRUE ~ 0)) %>%
  mutate(correct_manual = case_when(correct == 1 ~ 1,
    is.na(correct) ~ 0))
```

Inspect results

```
new_correct <- df2 %>%
  filter(correct_manual == 0 &
    (correct_auto_jaro == 1)) %>%
  select(item, word, word.c, bestmatch_jaro, answercode, answer_auto_jaro)

new_incorrect <- df2 %>%
  filter(correct_manual == 1 &
    (correct_auto_jaro == 0)) %>%
  select(item, word, word.c, bestmatch_jaro, answercode, answer_auto_jaro)

both_incorrect <- df2 %>%
  filter(correct_manual == 1 & correct_auto_jaro == 1) %>%
  select(item, word, word.c, bestmatch_jaro, jaro, answercode, answer_auto_jaro)

print("Jaro vs. manual: ");
```

```
## [1] "Jaro vs. manual: "
```

```
table(df2$correct_auto_jaro, df2$correct_manual);
```

```
##
##      0      1
## 0 500    21
## 1      8 4271
```

Comparison manual and automatic classification (Jaro distance)


```
table(df2$correct_auto_jaro, df2$correct_manual)
```

```
##
##      0      1
## 0 500    21
## 1   8 4271
```

```
print("The new correct trials:")
```

```
## [1] "The new correct trials:"
```

```
(new_correct <- df2 %>% filter(correct_manual == 0 & correct_auto_jaro == 1) %>%
  select(item, word, word.c, bestmatch_jaro, jaro,
         answer_auto_jaro, answercode))
```

```
##      item
## 1   flasche
## 2    kelle
## 3    geige
## 4  schublade
## 5    u-boot
## 6 geschirrspüler
## 7  daunenweste
## 8 kaffeemaschine
```

```
##
```

```
## 1
```

```
## 2
```

```
## 3 GITARRBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceGE
```

```
## 4
```

```
## 5
```

```
## 6
```

```
## 7
```

```
## 8
```

```
##      word.c bestmatch_jaro      jaro  answer_auto_jaro
## 1    GLAS    GLASFLASCHE 0.21212121212121212 approx_alternative
## 2   KESSEL      kelle 0.26111111111111111 approx_correct
## 3    GEIGE      geige      0 correctedto correct
## 4   SCHUB    schublade 0.14814814814814814 approx_correct
## 5      U      u-boot 0.27777777777777778 approx_correct
## 6  GESCHIRR geschirrspüler 0.142857142857143 approx_correct
## 7 DAUNENJACKE daunenweste 0.242424242424243 approx_correct
## 8   KAFFEE  KAFFEEKOCHER 0.166666666666667 approx_alternative
```

```
##      answercode
```

```
## 1 semantic_relation
```

```
## 2 unrelated_other
```

```
## 3 semantic_relation
```

```
## 4 unrelated_other
```

```
## 5 unrelated_other
```

```
## 6 semantic_relation
```

```
## 7 first_letter_incorrect
```

```
## 8 semantic_relation
```

GLASE

KESSELE

GESCH

SCHUBE

GESCH

DAUNENJACKEE

KAFFEE

```
print("Amount of trials additionally considered as correct: ");
```

```
## [1] "Amount of trials additionally considered as correct: "
```

```
sum(df2$correct_manual == 0 & df2$correct_auto_jaro == 1);
```

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

```
print("In percent: ");
```

```
## [1] "In percent: "
```

```
round(sum(df2$correct_manual == 0 & df2$correct_auto_jaro == 1)/  
      nrow(df2)*100,2)
```

```
## [1] 0.17
```

```
print("The new incorrect trials:")
```

```
## [1] "The new incorrect trials:"
```

```
(new_incorrect <- df2 %>% filter(correct_manual == 1 & correct_auto_jaro == 0) %>%  
  select(item, word, word.c, bestmatch_jaro, jaro,  
         answer_auto_jaro, answercode))
```

```
##           item  
## 1  schornstein  
## 2  daunenweste  
## 3  pelzmantel  
## 4  goldfisch  
## 5  pelzmantel  
## 6    feile  
## 7    feile  
## 8    couch  
## 9  goldfisch  
## 10   burg  
## 11 pelzmantel  
## 12   schloss  
## 13    feile  
## 14 luftballon  
## 15  zigarette  
## 16   kuchen  
## 17  goldfisch  
## 18    feile  
## 19    feile  
## 20    fuss  
## 21   glocke  
##  
## 1
```

```

## 2 WESTEBackspaceBackspaceBackspaceBackspace
## 3
## 4 FISCBackspaceBackspaceBackspaceBackspaceG
## 5 MANTBackspaceBackspaceBackspaceBackspaceEL
## 6
## 7
## 8 SOFBa
## 9
## 10 SCHB
## 11 MANTELBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBackspaceBa
## 12 BURBack
## 13
## 14 BALLBackspa
## 15 ZOIGBackspaceBacksp
## 16
## 17
## 18
## 19
## 20 FPBackspaceUDeadDeadBackspaceBackspace?=Backspace
## 21

## word.c bestmatch_jaro jaro answer_auto_jaro
## 1 SCHORNSTEIN schornstein 0 first_letter_error
## 2 DAUNENWESTE daunenweste 0 first_letter_error
## 3 PELZMANTEL pelzmantel 0 first_letter_error
## 4 6FISCH FISCH 0.0555555555555556 first_letter_error
## 5 PELZMANTEL pelzmantel 0 first_letter_error
## 6 PFEILER feile 0.0952380952380952 first_letter_error
## 7 FEILE feile 0 first_letter_error
## 8 6COUCH couch 0.0555555555555556 first_letter_error
## 9 6GOLDFISCH goldfisch 0.0333333333333334 first_letter_error
## 10 6BURG burg 0.0666666666666668 first_letter_error
## 11 PELZ PELZ 0 first_letter_error
## 12 6SCHLOSS schloss 0.0416666666666667 first_letter_error
## 13 PFEILE feile 0.0555555555555556 first_letter_error
## 14 LUFTBALLON luftballon 0 first_letter_error
## 15 6ZIGARETTE zigarette 0.0333333333333334 first_letter_error
## 16 TORTE TORTE 0 first_letter_error
## 17 GOLDFISCH goldfisch 0 first_letter_error
## 18 PFEILE feile 0.0555555555555556 first_letter_error
## 19 PFEILE feile 0.0555555555555556 first_letter_error
## 20 FUDeadDeDeadSS FUß 0.396825396825397 distance_based_error
## 21 CLOCKE glocke 0.1111111111111111 first_letter_error

## answercode
## 1 almostcorrect
## 2 almostcorrect
## 3 almostcorrect
## 4 almostcorrect
## 5 almostcorrect
## 6 almostcorrect
## 7 almostcorrect
## 8 almostcorrect
## 9 almostcorrect
## 10 almostcorrect
## 11 almostcorrect

```

```
## 12 almostcorrect
## 13 almostcorrect
## 14 almostcorrect
## 15 almostcorrect
## 16 almostcorrect
## 17 almostcorrect
## 18 almostcorrect
## 19 almostcorrect
## 20 almostcorrect
## 21 almostcorrect
```

```
print("Amount of trials additionally considered as incorrect: ");
```

```
## [1] "Amount of trials additionally considered as incorrect: "
```

```
sum(df2$correct_manual == 1 & df2$correct_auto_jaro == 0);
```

```
## [1] 21
```

```
print("In percent: ");
```

```
## [1] "In percent: "
```

```
round(sum(df2$correct_manual == 1 & df2$correct_auto_jaro == 0)/
      nrow(df2)*100,2)
```

```
## [1] 0.44
```

Total amount of trials classified differently (in percent):

```
round(((sum(df2$correct_manual == 0 & df2$correct_auto_jaro == 1)+ sum(df2$correct_manual == 1 & df2$correct_auto_jaro == 0))/
      nrow(df2))*100,2)
```

```
## [1] 0.6
```

Correlation between manual and automatic classification:

```
round(cor(df2$correct_manual, df2$correct_auto_jaro),2)
```

```
## [1] 0.97
```

Typing error analyses based on automatic classification

Amount of trials classified as correct and incorrect

```
print("totaltrials:")
```

```
## [1] "totaltrials:"
```

```
nrow(df2)
```

```
## [1] 4800
```

```
print("correct:")
```

```
## [1] "correct:"
```

```
(correct = sum(df2$correct_auto_jaro == 1))
```

```
## [1] 4279
```

```
print("incorrect:")
```

```
## [1] "incorrect:"
```

```
(incorrect = sum(df2$correct_auto_jaro == 0))
```

```
## [1] 521
```

Percentage of incorrect trials

```
# incorrect/nrow(df2)*100  
# incorrect/30/160*100  
incorrect_per_subject <-  
  as.data.frame(table(df2$subject, df2$correct_auto_jaro)) %>%  
  filter(Var2 == 0) %>% select(Var1, Freq) %>%  
  dplyr::rename(subject = Var1, perct_incorrect = Freq) %>%  
  mutate(perct_incorrect = perct_incorrect/160)  
  
print("Mean:")
```

```
## [1] "Mean:"
```

```
round(mean(incorrect_per_subject$perct_incorrect)*100,2)
```

```
## [1] 10.85
```

```
print("SD:")
```

```
## [1] "SD:"
```

```
round(sd(incorrect_per_subject$perct_incorrect)*100,2)
```

```
## [1] 4.73
```

```
print("Range:")
```

```
## [1] "Range:"
```

```
round(range(incorrect_per_subject$perct_incorrect)*100,2)
```

```
## [1] 3.12 19.38
```

Correct/incorrect trials per participant:

```
print(as.data.frame(table(
  df2$subject, df2$correct_auto_jaro == 1)) %>%
  filter(Var2 == TRUE) %>%
  dplyr::rename(subject = Var1, totaltrials = Var2,
    correct_auto = Freq) %>%
  mutate(totaltrials = 160) %>%
  mutate(percentagecorrect = correct_auto/totaltrials))
```

##	subject	totaltrials	correct_auto	percentagecorrect
## 1	1	160	155	0.96875
## 2	2	160	150	0.93750
## 3	3	160	149	0.93125
## 4	4	160	141	0.88125
## 5	5	160	141	0.88125
## 6	6	160	141	0.88125
## 7	7	160	153	0.95625
## 8	8	160	133	0.83125
## 9	9	160	129	0.80625
## 10	10	160	137	0.85625
## 11	11	160	148	0.92500
## 12	12	160	153	0.95625
## 13	13	160	129	0.80625
## 14	14	160	134	0.83750
## 15	15	160	153	0.95625
## 16	16	160	145	0.90625
## 17	17	160	154	0.96250
## 18	18	160	142	0.88750
## 19	19	160	142	0.88750
## 20	20	160	140	0.87500
## 21	21	160	139	0.86875
## 22	22	160	141	0.88125
## 23	23	160	149	0.93125
## 24	24	160	146	0.91250
## 25	25	160	139	0.86875
## 26	26	160	135	0.84375
## 27	27	160	145	0.90625
## 28	28	160	137	0.85625
## 29	29	160	131	0.81875
## 30	30	160	148	0.92500

Write data file for statistical analyses

```
write.csv(df2, here::here("data", "transient_data_files", "data_long_final.csv"))
```