Text_Normalization

Contents

Introduction

The aim of this challenge is to "automate the process of developing text normalization grammars via machine learning" It is described as the process of transforming language into a specific, self-consistent grammar system with well-defined rules. Here, we aim to convert "written expressions into into appropriate 'spoken' forms", as described using the following example: "[. . .] convert 12:47 to 'twelve forty-seven' and $3.16 into 'three dollars, sixteen cents.' "

The data comes in the shape of two files: `../input/en_train.csv` and `../input/en_test.csv`. Each row contains a single language element (such as words, letters, or punctuation) together with its associated identifiers.

Load libraries and helper functions

```
# general visualisation
library('ggplot2') # visualisation
library('scales') # visualisation
library('grid') # visualisation
library('gridExtra') # visualisation
library('RColorBrewer') # visualisation
library('corrplot') # visualisation
library('ggforce') # visualisation
library('treemapify') # visualisation

# general data manipulation
library('dplyr') # data manipulation
library('readr') # input/output
library('data.table') # data manipulation
library('tibble') # data wrangling
library('tidyr') # data wrangling
library('stringr') # string manipulation
library('forcats') # factor manipulation

# Text / NLP
library('tidytext') # text analysis
library('tm') # text analysis
library('SnowballC') # text analysis
library('topicmodels') # text analysis
library('wordcloud') # test visualisation

# Extra Vis
library('plotly')

# Extras
library('babynames')
```

```
# Define multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
```

```r
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

Load data

We use *data.table's* fread function to speed up reading in the data. The training data file is rather extensive, with close to 10 million rows and 300 MB uncompressed size:

```r
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
train <- as.tibble(fread('./en_train.csv'))
test <- as.tibble(fread('./en_test.csv'))
```

File structure and content

We will have an overview of the data sets using the *summary* and *glimpse* tools. First the training data:

```
summary(train)
```

```
##   sentence_id        token_id         class              before
##  Min.   :     0   Min.   :  0.00   Length:9918441     Length:9918441
##  1st Qu.:192526   1st Qu.:  3.00   Class :character   Class :character
##  Median :379259   Median :  6.00   Mode  :character   Mode  :character
##  Mean   :377856   Mean   :  7.52
##  3rd Qu.:564189   3rd Qu.: 11.00
##  Max.   :748065   Max.   :255.00
##     after
##  Length:9918441
##  Class :character
##  Mode  :character
##
##
##
```

```
glimpse(train)
```

```
## Rows: 9,918,441
## Columns: 5
## $ sentence_id <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2~
## $ token_id    <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1~
## $ class       <chr> "PLAIN", "PLAIN", "PLAIN", "PLAIN", "PLAIN", "PLAIN", "PLA~
## $ before      <chr> "Brillantaisia", "is", "a", "genus", "of", "plant", "in", ~
## $ after       <chr> "Brillantaisia", "is", "a", "genus", "of", "plant", "in", ~
```

And then the test data:

```
summary(test)
```

```
##   sentence_id       token_id          before
##  Min.   :    0   Min.   :  0.000   Length:1088564
##  1st Qu.:17488   1st Qu.:  3.000   Class :character
##  Median :35028   Median :  7.000   Mode  :character
##  Mean   :35007   Mean   :  8.344
##  3rd Qu.:52522   3rd Qu.: 12.000
##  Max.   :69999   Max.   :248.000
```

```
glimpse(test)
```

```
## Rows: 1,088,564
## Columns: 3
## $ sentence_id <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1~
## $ token_id    <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 1, 2,~
## $ before      <chr> "Another", "religious", "family", "is", "of", "Hazrat", "S~
```

We find:

- *Sentence_id* is an integer identifying each individual sentence. These are the total number of sentences in the train and test data:

```
print(c(max(train$sentence_id), max(test$sentence_id)))
```

```
## [1] 748065  69999
```

- *Token_id*, in a similar way, is an integer counting the number of language elements within each sentence; e.g. words or punctuation. The maximum *token_id*, i.e. sentence length, is 255 in the training data and 248 in the test data.
- *Before* is the original language element and *after* is its normalised text. **The aim of this challenge is to predict the 'after' column for the test data set.**
- *Class* is indicating the type of the language token. This feature, together with *after*, is "intentionally omitted from the test set"

We combine the train and test data sets for comparison treatment:

```
combine <- bind_rows(train %>% mutate(set = "train"),test %>% mutate(set = "test")) %>%
  mutate(set = as.factor(set))
```

Missing values

There are no missing values in our data set:

```
sum(is.na(train))
```

```
## [1] 0
```

```
sum(is.na(test))
```

```
## [1] 0
```

Reformating features

We decide to turn *Class* into a factor for exploration purposes:

```
train <- train %>%
  mutate(class = factor(class))
```

Example sentences and overview visualisations

A first look at normalised sentences

Before diving into data summaries, let's begin by printing a few *before* and *after* sentences to get a feeling for the data. To make this task easier, we first define a short helper function to compare these sentences:

```
before_vs_after <- function(sent_id){

  bf <- train %>%
    filter(sentence_id == sent_id) %>%
    .$before %>%
    str_c(collapse = " ")

  af <- train %>%
    filter(sentence_id == sent_id) %>%
    .$after %>%
    str_c(collapse = " ")

  print(str_c("Before:", bf, sep = " ", collapse = " "))
  print(str_c("After :", af, sep = " ", collapse = " "))
}
```

Those are a few example sentences:

```
before_vs_after(11)
```

```
## [1] "Before: Retrieved April 10, 2013 ."
## [1] "After : Retrieved april tenth twenty thirteen ."
```

```
before_vs_after(99)
```

```
## [1] "Before: Retrieved 12 April 2015 ."
## [1] "After : Retrieved the twelfth of april twenty fifteen ."
```

We already notice a few things:

- The first two examples are very similar, but the normalisations manage to make the difference between “april tenth” and “tenth of april” depending on how the date is written.
- “2015” becomes “twenty fifteen” instead of “two thousand fifteen”. This should be easy to transform, though.
- “April” becomes “april”. Lower vs upper case should not be a problem, but it is noteworthy.
- Acronyms like “PMO” turn into their spoken version of “p m o”.

Summary visualisations

Now let’s look at overview visualisations. First, we examine the different token *classes* and their frequency. Here is a visual summary:

```
train %>%
  group_by(class) %>%
  count() %>%
  ungroup() %>%
  mutate(class = reorder(class, n)) %>%
  ggplot(aes(class,n, fill = class)) +
  geom_col() +
  scale_y_log10() +
  labs(y = "Frequency") +
  coord_flip() +
  theme(legend.position = "none")
```

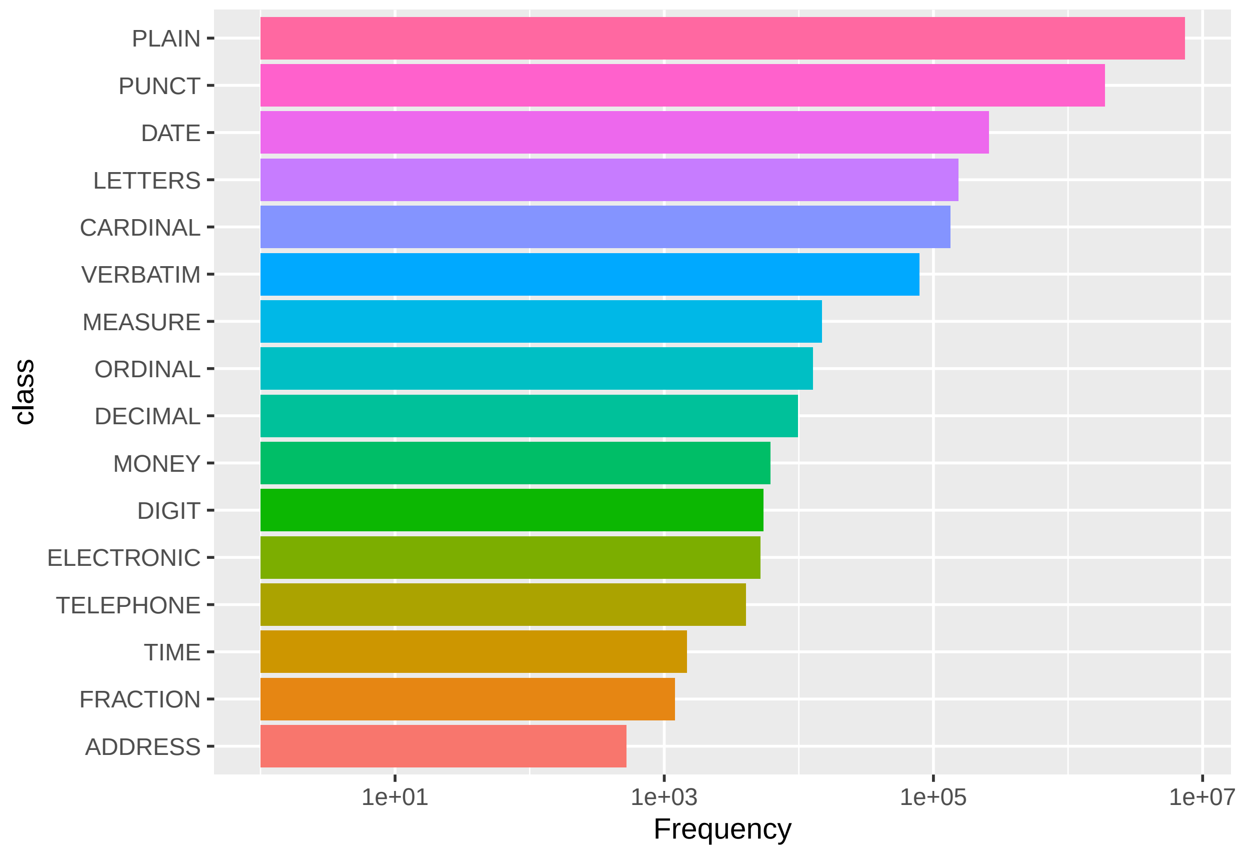

Figure 1: Fig. 1

Note the logarithmic x-axis.

We find:

- The "PLAIN" *class* is by far the most frequent, followed by "PUNCT" and "DATE".
- In total there are 16 *classes*, with "TIME", "FRACTION", and "ADDRESS" having the lowest number of occurrences (around/below 100 tokens each).

Next up, this is the histogram distribution of the sentence length for the training data:

```
train %>%
  group_by(sentence_id) %>%
  summarise(sentence_len = max(token_id)) %>%
  ggplot(aes(sentence_len)) +
  geom_histogram(bins = 50, fill = "red") +
  scale_y_sqrt() +
  scale_x_log10() +
  labs(x = "Sentence length")
```

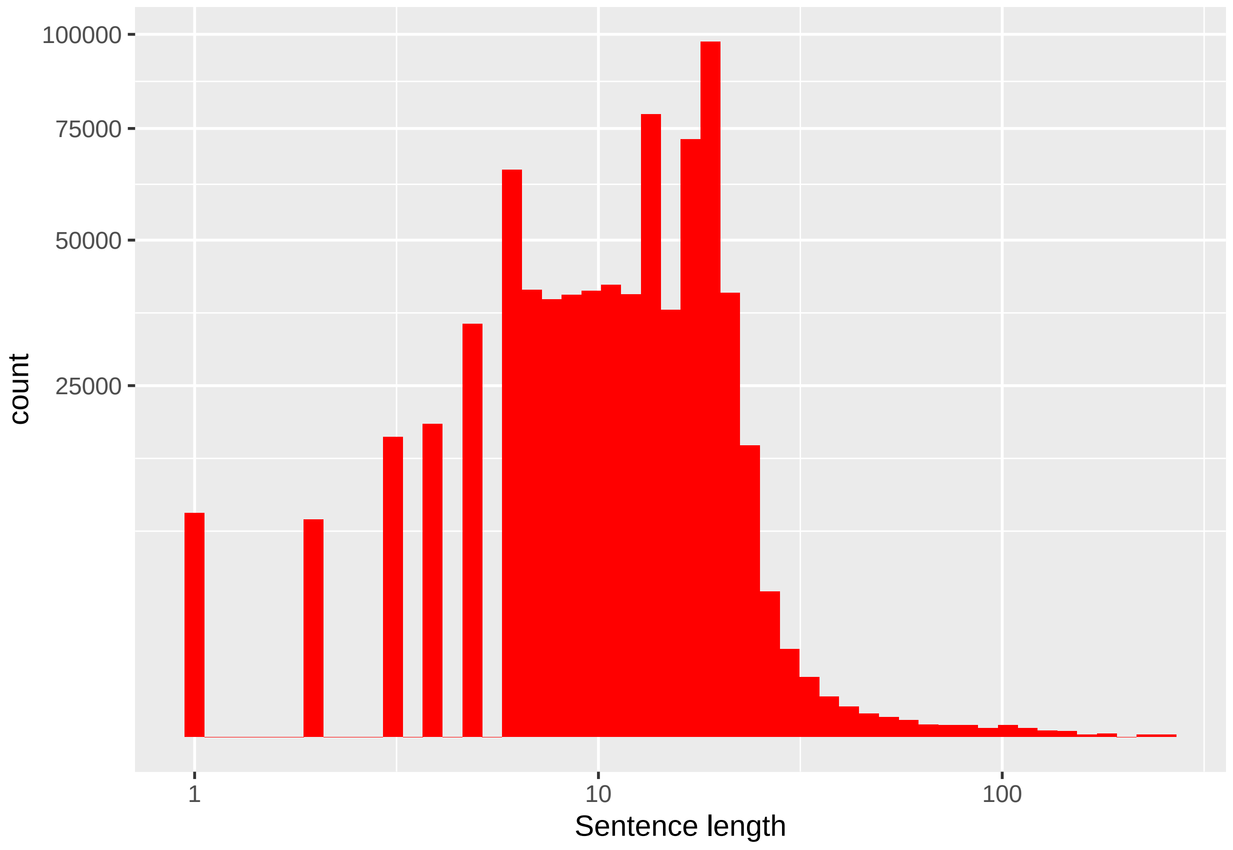

Figure 2: Fig. 2

We find that our sentences are typically up to 15-20 tokens long, after which the frequency drops quickly. Very long sentences (> 100 tokens) exist but are relatively rare. Note again the logarithmic x-axis and square-root y-axis.

Next, we will look at the *token_ids* of each *class* in their sentences:

```
train %>%
  ggplot(aes(reorder(class, token_id, FUN = median), token_id, col = class)) +
  geom_boxplot() +
  scale_y_log10() +
  theme(legend.position = "none", axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  labs(x = "Class", y = "Token ID")
```

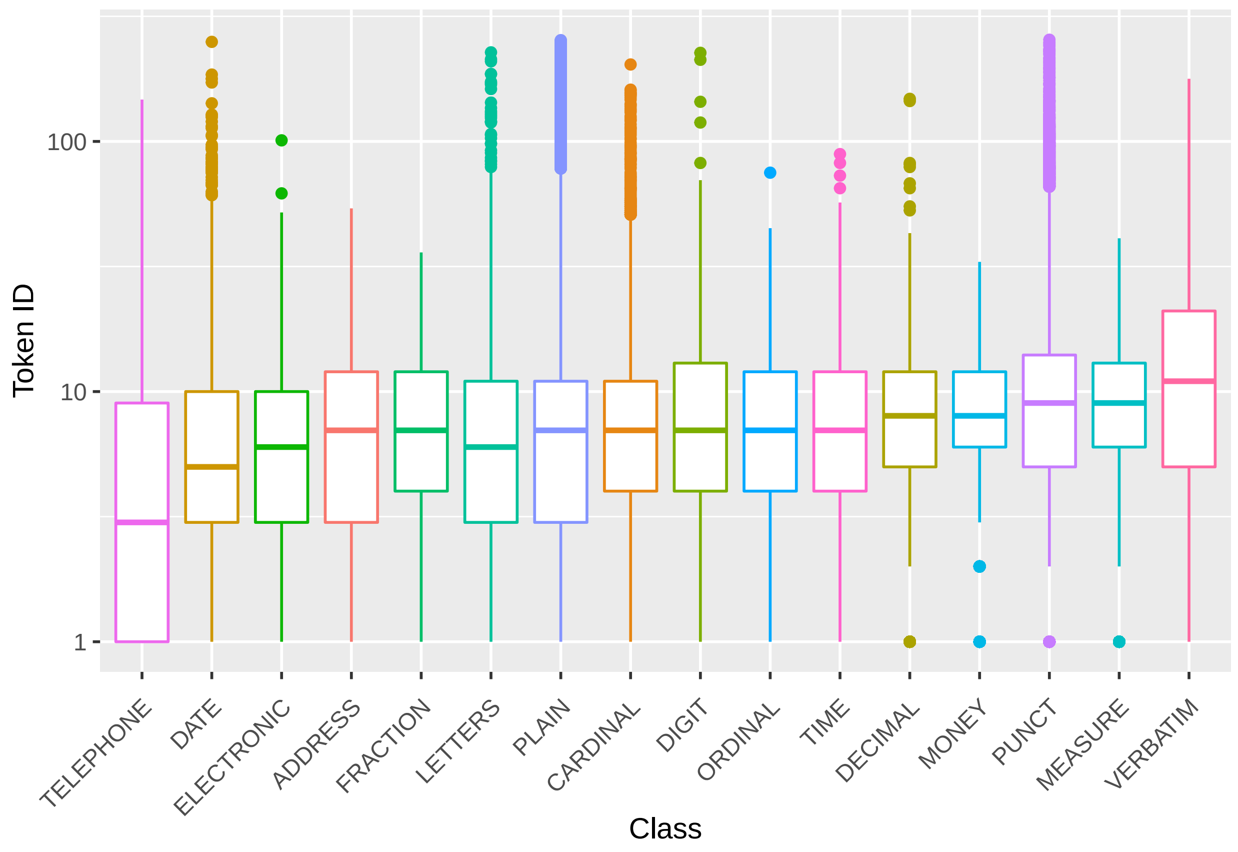

Figure 3: Fig. 4

We find:

- The "TELEPHONE" class appears predominantly at *token_ids* of less than 10.
- Above *token_ids* of 100 we find no occurences of the "ELECTRONICS", "ADDRESS", "FRACTION", "DIGIT", "ORDINAL", "TIME", "MONEY", and "MEASURE" *classes*. Of those, "FRACTION", "MONEY", and "MEASURE" barely appear above `token_id == 20`.
- The *classes* "DECIMAL", "MONEY", "PUNCT", and "MEASURE" are rarely found in the first token of a sentence.

We can take this analysis a step further by relating the *token_id* to the length of the sentence. Thereby, we will see at which relative position in a sentence a certain *class* is more likely to occur:

```
sen_len <- train %>%
  group_by(sentence_id) %>%
  summarise(sentence_len = max(token_id))

train %>%
  left_join(sen_len, by = "sentence_id") %>%
  mutate(token_rel = token_id/sentence_len) %>%
  ggplot(aes(reorder(class, token_rel, FUN = median), token_rel, col = class)) +
  geom_boxplot() +
  #scale_y_log10() +
  theme(legend.position = "none", axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  labs(x = "Class", y = "Relative token ID")
```

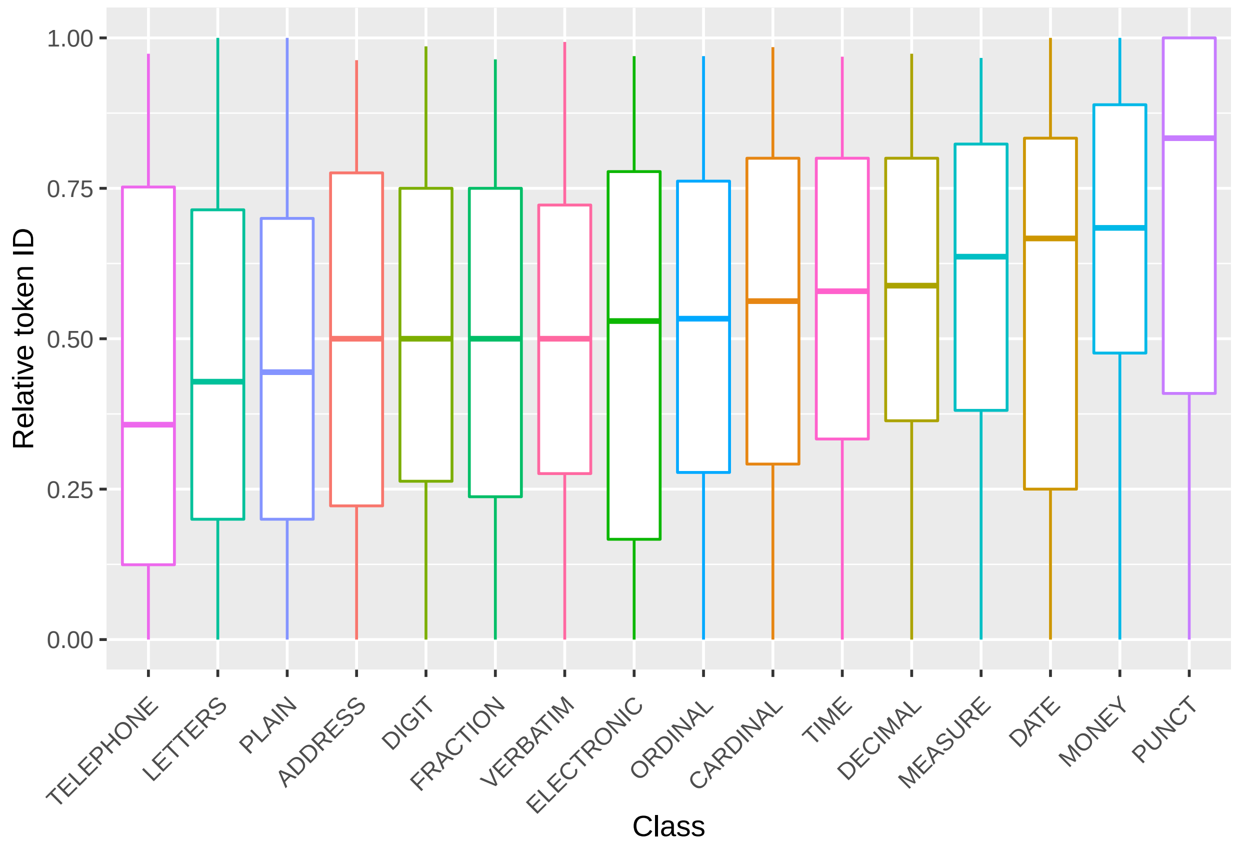

Figure 4: Fig. 5

We find:

- As suggest above, "TELEPHONE" tokens are more likely to occur early in a sentence. A similar observation holds for "LETTERS" and "PLAIN".
- Unsurprisingly, the "PUNCT" *class* can be found more frequently towards the end of a sentence. Similarly, "MONEY" tokens occur relatively late.
- In general, there is a certain trend among the *classes*, with medians ranging from about 0.4 to 0.8. However, interquartile ranges are wide and there is a large amount of overlap between the different *classes*.

The impact of the normalisation

Now we will include the effects of the text normalisation in our study by analysing the changes it introduced in the training data.

To begin, we define the new feature *transformed* to indicate those tokens that changed from *before* to *after*:

```
train <- train %>%
  mutate(transformed = (before != after))
```

In total, only about 7% of tokens in the training data, or about 660k objects in total, were changed during the process of text normalisation:

```
train %>%
  group_by(transformed) %>%
  count() %>%
  mutate(freq = n/nrow(train))
```

```
## # A tibble: 2 x 3
## # Groups:   transformed [2]
##   transformed       n   freq
##   <lgl>         <int>  <dbl>
## 1 FALSE       9258648 0.933
## 2 TRUE         659793 0.0665
```

By comparing the fraction of tokens that changed from *before* to *after* the text normalisation we can visualise which *classes* are most affected by this process:

```
train %>%
  ggplot(aes(class, fill = transformed)) +
  geom_bar(position = "fill") +
  theme(axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  labs(x = "Class", y = "Transformed fraction [%]")
```

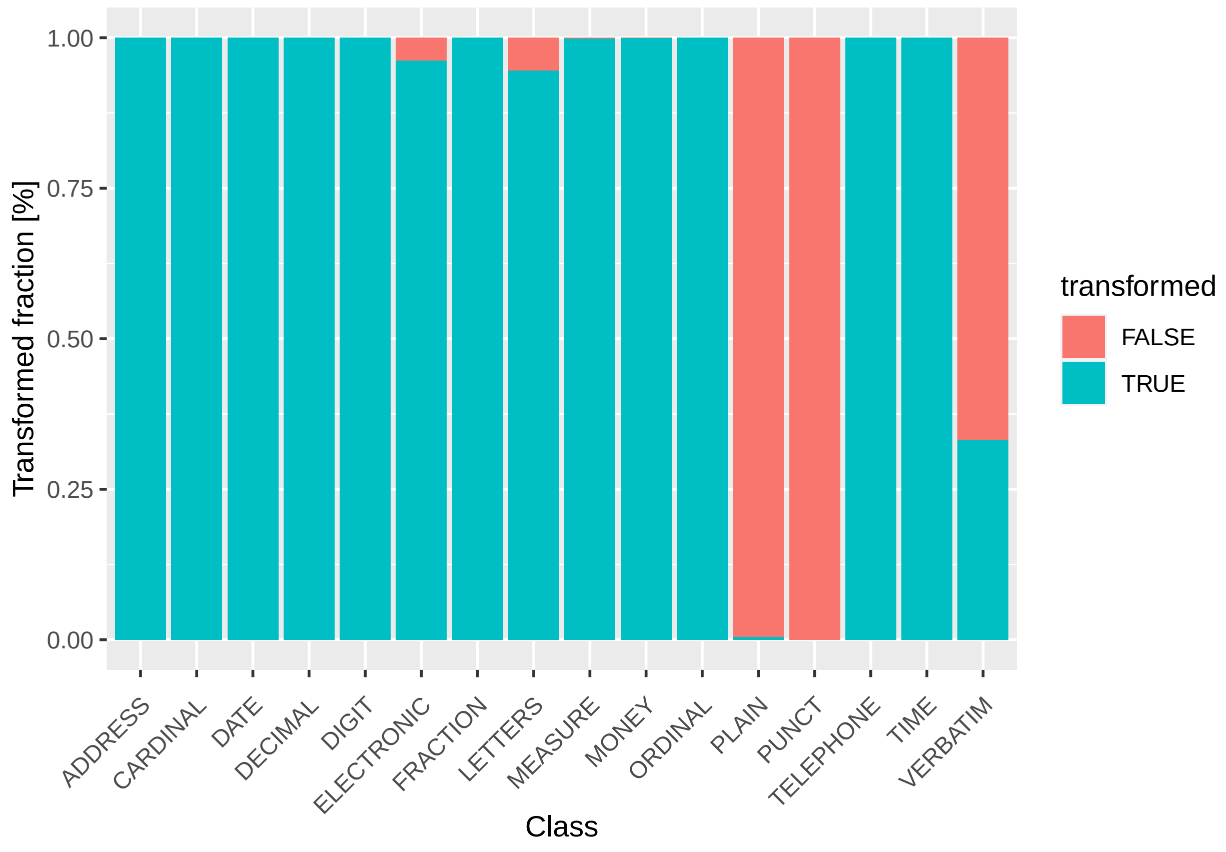

Figure 5: Fig. 6

We find:

- Quite a few “ELECTRONIC” and “LETTERS” tokens did not change in *after* vs *before*.
- A noteable fraction of “PLAIN” *class* text elements **did change**.
- The majority of “VERBATIM”-*class* tokens remained identical during the normalisation.

This is a breakdown of the number and fraction how many tokens per *class* remained identical in *before* vs *after*. Here we only list the *classes* that feature at least one unchanged token:

```
train %>%
  group_by(class, transformed) %>%
  count() %>%
  spread(key = transformed, value = n) %>%
  mutate(`TRUE` = ifelse(is.na(`TRUE`),0,`TRUE`),
         `FALSE` = ifelse(is.na(`FALSE`),0,`FALSE`)) %>%
  mutate(frac = `FALSE`/(`TRUE`+`FALSE`)*100) %>%
  filter(frac>1e-5) %>%
  arrange(desc(frac)) %>%
  rename(unchanged = `FALSE`, changed = `TRUE`, unchanged_percentage = frac)
```

```
## # A tibble: 7 x 4
## # Groups:   class [7]
##   class      unchanged changed unchanged_percentage
##   <fct>          <dbl>   <dbl>                <dbl>
## 1 PUNCT        1880507       0             100
## 2 PLAIN        7317221   36472              99.5
## 3 VERBATIM       52271   25837              66.9
## 4 LETTERS         8426  144369               5.51
## 5 ELECTRONIC       198    4964               3.84
## 6 MEASURE           22   14761               0.149
## 7 MONEY              3    6125               0.0490
```

Token classes and their text normalisation

In order to explore the meaning of these classes for our normalisation task we modify our helper function to include the class name. Here we also remove punctuation:

```
before_vs_after_class <- function(sent_id){

  bf <- train %>%
    filter(sentence_id == sent_id & class != "PUNCT") %>%
    .$before %>%
    str_pad(30) %>%
    str_c(collapse = " ")

  af <- train %>%
    filter(sentence_id == sent_id & class != "PUNCT") %>%
    .$after %>%
    str_pad(30) %>%
    str_c(collapse = " ")

  cl <- train %>%
    filter(sentence_id == sent_id & class != "PUNCT") %>%
    .$class %>%
    str_pad(30) %>%
    str_c(collapse = " ")

  print(str_c("[Class]:", cl, sep = " ", collapse = " "))
  print(str_c("Before :", bf, sep = " ", collapse = " "))
  print(str_c("After  :", af, sep = " ", collapse = " "))
}
```

Using the example sentence from earlier, we get the following output structure, indicating the combination of a “PLAIN” and a “DATE” token:

```
before_vs_after_class(11)
```

```
## [1] "[Class]:                          PLAIN                          DATE"
## [1] "Before :                      Retrieved                April 10, 2013"
## [1] "After  :                      Retrieved    april tenth twenty thirteen"
```

In the following segment we will explore the different token *classes* available and provide a few examples for each.

Plain/simple word class: "PLAIN"

As we saw above, most "PLAIN" tokens remained unchanged. However, about 0.5% were transformed, which still amounts to about 36k tokens:

```
train %>%
  filter(class == "PLAIN") %>%
  group_by(transformed) %>%
  count() %>%
  spread(key = transformed, value = n) %>%
  mutate(frac = `FALSE`/(`TRUE`+`FALSE`)*100) %>%
  filter(!is.na(frac)) %>%
  arrange(desc(frac)) %>%
  rename(unchanged = `FALSE`, changed = `TRUE`, unchanged_percentage = frac)
```

```
## # A tibble: 1 x 3
##   unchanged changed unchanged_percentage
##       <int>   <int>                <dbl>
## 1   7317221   36472                 99.5
```

Few transformed examples:

```
set.seed(1234)
train %>%
  filter(transformed == TRUE & class == "PLAIN") %>%
  sample_n(10)
```

```
## # A tibble: 10 x 6
##    sentence_id token_id class before    after    transformed
##          <int>    <int> <fct> <chr>     <chr>    <lgl>
##  1      322886        0 PLAIN st        saint    TRUE
##  2      692875       10 PLAIN st        saint    TRUE
##  3      732353        7 PLAIN -         to       TRUE
##  4      367611        3 PLAIN favourite favorite TRUE
##  5      322388        0 PLAIN st        saint    TRUE
##  6      411373        1 PLAIN -         to       TRUE
##  7       58061        0 PLAIN st        saint    TRUE
##  8       66159        4 PLAIN OK        okay     TRUE
##  9      366032        6 PLAIN Synagogue synagog  TRUE
## 10      685106        6 PLAIN st        saint    TRUE
```

We see a few typical changes, such as "-" to "to" and the adjustment from British to American spelling.

Punctuation class: "PUNCT"

It is worth to briefly cross-check the "PUNCT" class. Intuitively, punctuation should not be affected by text normalisation unless it is associated to other structures such as numbers or dates. Here are a few punctuation examples:

```
set.seed(1234)
train %>%
  filter(class == "PUNCT") %>%
  sample_n(10)
```

```
## # A tibble: 10 x 6
##    sentence_id token_id class before after transformed
##          <int>    <int> <fct> <chr>  <chr> <lgl>
##  1      720720       18 PUNCT )      )     FALSE
##  2      669170       18 PUNCT .      .     FALSE
##  3      124454        4 PUNCT ,      ,     FALSE
##  4      374523       19 PUNCT .      .     FALSE
##  5      121056        3 PUNCT .      .     FALSE
##  6      408524        2 PUNCT (      (     FALSE
##  7       52291        3 PUNCT .      .     FALSE
##  8      571840       12 PUNCT .      .     FALSE
##  9      141063        2 PUNCT ,      ,     FALSE
## 10      348724        2 PUNCT ,      ,     FALSE
```

As expected, those tokens are identical *before* and *after*:

```
train %>%
  filter(class == "PUNCT") %>%
  mutate(test = (before == after)) %>%
  group_by(test) %>%
  count()
```

```
## # A tibble: 1 x 2
## # Groups:   test [1]
##   test        n
##   <lgl>   <int>
## 1 TRUE  1880507
```

Numerical *classes*

By far the most diverse *classes* (and normalisation treatment) involve numbers in one way or another. Those classes are the following: “DATE”, CARDINAL“,”MEASURE“,”ORDINAL“,”DECIMAL“,”MONEY“, DIGIT”, “TELEPHONE”, “TIME”, “FRACTION”, and “ADDRESS”. Here we will look at a few examples for each of them.

“DATE”

We have already seen how dates can be transformed differently depending on their formatting. Those are some more examples:

```
before_vs_after_class(8)
```

```
## [1] "[Class]:                          PLAIN                           DATE"
## [1] "Before :                      Retrieved                   4 March 2014"
## [1] "After  :                      Retrieved the fourth of march twenty fourteen"
```

```
before_vs_after_class(12)
```

```
## [1] "[Class]:                          PLAIN                          PLAIN
## [1] "Before :                     Downloaded                             on                  7 Augus
## [1] "After  :                     Downloaded                             on the seventh of august tw
```

Some cardinal numbers. Interestingly, these includes Roman numerals:

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "CARDINAL") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class    before after
##         <int> <fct>    <chr>  <chr>
## 1      594242 CARDINAL 354    three hundred fifty four
## 2      239411 CARDINAL 3      three
## 3      194392 CARDINAL 7      seven
## 4      703762 CARDINAL 100    one hundred
## 5      672429 CARDINAL 4      four
```

```
before_vs_after(471926)
```

```
## [1] "Before: Listed as Grade II by English Heritage ."
## [1] "After : Listed as Grade two by English Heritage ."
```

"MEASURE"

Those are mainly percentages and physical measurements:

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "MEASURE") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class   before      after
##         <int> <fct>   <chr>       <chr>
## 1      363560 MEASURE "3,000 rpm" three thousand revolutions per minute
## 2      393209 MEASURE "5 ft"      five feet
## 3      349545 MEASURE "81m"       eighty one meters
## 4      398094 MEASURE "2\"\""     two inches
## 5      355127 MEASURE "20 MA"     twenty mega amperes
```

```
before_vs_after(200476)
```

```
## [1] "Before: As of 2008 , the population was 50.3% male and 49.7% female ."
## [1] "After : As of two thousand eight , the population was fifty point three percent male and forty
```

"ORDINAL"

Also ordinal numbers can include Roman numerals, as in the example of Queen Elizabeth I:

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "ORDINAL") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class   before after
##         <int> <fct>   <chr>  <chr>
## 1      460373 ORDINAL 14th   fourteenth
## 2      493459 ORDINAL 18th   eighteenth
## 3      441931 ORDINAL 18th   eighteenth
## 4      497894 ORDINAL 30th   thirtieth
## 5      448833 ORDINAL 2nd    second
```

```
before_vs_after(478452)
```

```
## [1] "Before: Maid of honour to Elizabeth I ( 1576 until 1583 ) ."
## [1] "After : Maid of honor to Elizabeth the first ( fifteen seventy six until fifteen eighty three )
```

"DECIMAL"

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "DECIMAL") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class   before       after
##         <int> <fct>   <chr>        <chr>
## 1      555972 DECIMAL 2.58         two point five eight
## 2      600577 DECIMAL 40.4 million forty point four million
## 3      534320 DECIMAL .13          point one three
## 4      605133 DECIMAL 21.5         twenty one point five
## 5      541897 DECIMAL 2.71         two point seven one
```

```
before_vs_after(443910)
```

```
## [1] "Before: There were 331 housing units at an average density of 199.4 per square mile ( 77.0/km2 
## [1] "After : There were three hundred thirty one housing units at an average density of one hundred 
```

"MONEY"

Money tokens come with currency symbols like "$" or plain text names like "yuan":

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "MONEY") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class before      after
##         <int> <fct> <chr>       <chr>
## 1      125696 MONEY $31,019     thirty one thousand nineteen dollars
## 2       83805 MONEY $31,339     thirty one thousand three hundred thirty nine d~
## 3      329845 MONEY $27,459     twenty seven thousand four hundred fifty nine d~
## 4      119518 MONEY $1 million  one million dollars
## 5      552160 MONEY $15 million fifteen million dollars
```

```
before_vs_after(90430)
```

```
## [1] "Before: It issued 369 citations at that time , assessing $10.8 million in penalties ."
## [1] "After : It issued three hundred sixty nine citations at that time , assessing ten point eight mi
```

"DIGIT"

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "DIGIT") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class before   after
##         <int> <fct> <chr>    <chr>
## 1      149090 DIGIT 5-       five
## 2       91654 DIGIT 14467515 one four four six seven five one five
## 3      388890 DIGIT 1986     one nine eight six
## 4      136554 DIGIT 2008     two o o eight
## 5      617402 DIGIT 2009     two o o nine
```

```
before_vs_after(481444)
```

```
## [1] "Before: Mintz , Sidney W. 1996 a Tasting Food , Tasting Freedom : Excursions into Eating , Cultu
## [1] "After : Mintz , Sidney w one nine nine six a Tasting Food , Tasting Freedom : Excursions into Ea
```

"TELEPHONE"

```
set.seed(1234)
train %>%
  filter(class == "TELEPHONE") %>%
  select(-token_id, -class, -transformed) %>%
  sample_n(5)
```

```
## # A tibble: 5 x 3
##   sentence_id before            after
##         <int> <chr>             <chr>
## 1      626030 1-55704-371       one sil five five seven o four sil three seven ~
## 2      729191 978-9955-34-009-6 nine seven eight sil nine nine five five sil th~
## 3      575852 (2014) 38         two o one four sil three eight
## 4      740920 0-8020-9587-9     o sil eight o two o sil nine five eight seven s~
## 5      596101 2-6-2             two sil six sil two
```

```
before_vs_after(88957)
```

```
## [1] "Before: The show was viewed by an estimated 3.17 million Americans with a 1.1 / 3 18-49 rating /
## [1] "After : The show was viewed by an estimated three point one seven million Americans with a one p
```

"TIME"

The "TIME" formatting and normalisation are comparatively simple:

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "TIME") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class before  after
##         <int> <fct> <chr>   <chr>
## 1      666327 TIME  4 a.m.  four a m
## 2      515686 TIME  2 pm    two p m
## 3      555071 TIME  05:00   five o'clock
## 4      508322 TIME  12:00am twelve a m
## 5      324937 TIME  21:50   twenty one fifty
```

```
before_vs_after(471678)
```

```
## [1] "Before: The show shifted its timeslot from 10:30pm into 6:00pm in order to compete TV Patrol ."
## [1] "After : The show shifted its timeslot from ten thirty p m into six p m in order to compete t v p
```

"FRACTION"

Here it become tricky again:

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "FRACTION") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class    before  after
##         <int> <fct>    <chr>   <chr>
## 1      645167 FRACTION 641/640 six hundred forty one six hundred fortieths
## 2      714098 FRACTION 1/5     one fifth
## 3      639299 FRACTION 9/15    nine fifteenths
## 4      404748 FRACTION 1/2     one half
## 5      577010 FRACTION 58/292  fifty eight two hundred ninety seconds
```

Many of those are unlikely to be fractions and are therefore normalised in a rather clumsy way:

```
before_vs_after(470330)
```

```
## [1] "Before: \"\" WWE Superstars Results ( 6/6 ) : Ascension vs"
## [1] "After : \"\" w w e Superstars Results ( six sixths ) : Ascension versus"
```

"ADDRESS"

The "ADDRESS" *class* appears to assigned to alpha-numeric combinations:

```
set.seed(1234)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "ADDRESS") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class   before   after
##         <int> <fct>   <chr>    <chr>
## 1      387277 ADDRESS "A5"     a five
## 2      151891 ADDRESS "US 30"  u s thirty
## 3      561652 ADDRESS "M66"    m sixty six
## 4      151119 ADDRESS "C1 "    c one
## 5      152777 ADDRESS "SR 931" s r nine thirty one
```

```
before_vs_after(88342)
```

```
## [1] "Before: \"\" Seat Map Singapore Airlines Airbus A380 \"\" ."
## [1] "After : \"\" Seat Map Singapore Airlines Airbus a three eighty \"\" ."
```

Acronyms and Initials: "LETTERS" *class*

The tokens in the *class* "LETTERS" appear to be normalised to a form which spells them out one by one:

```
set.seed(4321)
train %>%
  select(-token_id, -transformed) %>%
  filter(class == "LETTERS") %>%
  sample_n(5)
```

```
## # A tibble: 5 x 4
##   sentence_id class   before after
##         <int> <fct>   <chr>  <chr>
## 1      296887 LETTERS SS     s s
## 2      655334 LETTERS ODM    o d m
## 3      244399 LETTERS LOM    l o m
## 4      602982 LETTERS L.     l
## 5      485217 LETTERS A.     a
```

```r
before_vs_after(571356)
```

```
## [1] "Before: Kose , M. A. ; Prasad , E. S. ; Terrones , M. E. ( 2006 ) ."
## [1] "After : Kose , m a ; Prasad , e s ; Terrones , m e ( two thousand six ) ."
```

Since those text elements are typically completely in upper case it should be relatively simple to define a normalization.

Websites: "ELECTRONIC" *class*

This *class* includes websites that are normalised using single characters and a "dot":

```r
set.seed(4321)
train %>%
  filter(class == "ELECTRONIC") %>%
  select(-token_id, -class, -sentence_id, -transformed) %>%
  sample_n(5)
```

```
## # A tibble: 5 x 2
##   before                                 after
##   <chr>                                  <chr>
## 1 thepeerage.com                         t h e p e e r a g e dot c o m
## 2 Inquirer.net                           i n q u i r e r dot n e t
## 3 Austin.com                             a u s t i n dot c o m
## 4 http://decisions.fct-cf.gc.ca/fc-cf/d~ h t t p colon slash slash d e c i s i ~
## 5 #Foreign                               hash tag foreign
```

```r
before_vs_after(99083)
```

```
## [1] "Before: Climate Summary for Madison , Florida \"\" Weatherbase.com \"\" ."
## [1] "After : Climate Summary for Madison , Florida \"\" w e a t h e r b a s e dot c o m \"\" ."
```

This is another format that should be relatively easy to learn and implement.

In summary: The different *classes* appear to follow different translation rules for the text normalisation. Even though the *class* feature is not present in the *test* data set it might be useful to train a model to identify the specific class for a token and then apply its normalisation rules.

Context matters: a next-neighbour analysis

In this section we turn to analysing whether the context of our normalised token can provide any indications towards its *class*. We're doing this by preparing a data frame in which every token is listed together with the token (and its *class*) that came immediately previous or next in the text data. (Note, that for the first/last tokens of a sentence the *previous/next* tokens will belong to the the previous/next sentence.)

We build this new data frame using the `dplyr` tool `lead` which shifts the contents of a vector by a certain number of indices

```
t3 <- train %>%
  select(class, before) %>%
  mutate(before2 = lead(train$before, 1),
         before3 = lead(train$before,2),
         class2 = lead(train$class, 1),
         class3 = lead(train$class, 2),
         transformed = c(train$transformed[-1], NA),
         after = c(train$after[-1], 0)) %>%
  filter(!is.na(before3)) %>%
  rename(class_prev = class, class_next = class3, class = class2,
         before_prev = before, before_next = before3, before = before2) %>%
  select(before_prev, before, before_next, class_prev,
         class, class_next, transformed, after)
```

Treemaps- Token *class* overview - *previous* vs *next*

The treemaps summaries at a glance which neighbor combinations exist and are most frequent:

```
t3 %>%
  group_by(class, class_prev) %>%
  count() %>%
  ungroup() %>%
  mutate(n = log10(n+1)) %>%
  ggplot(aes(area = n, fill = class_prev, label = class_prev, subgroup = class)) +
  geom_treemap() +
  geom_treemap_subgroup_border() +
  geom_treemap_subgroup_text(place = "centre", grow = T, alpha = 0.5, colour =
                             "black", fontface = "italic", min.size = 0) +
  geom_treemap_text(colour = "white", place = "topleft", reflow = T) +
  theme(legend.position = "null") +
  ggtitle("Previous classes grouped by token class; log scale frequencies")
```

Previous classes grouped by token class; log scale frequencies

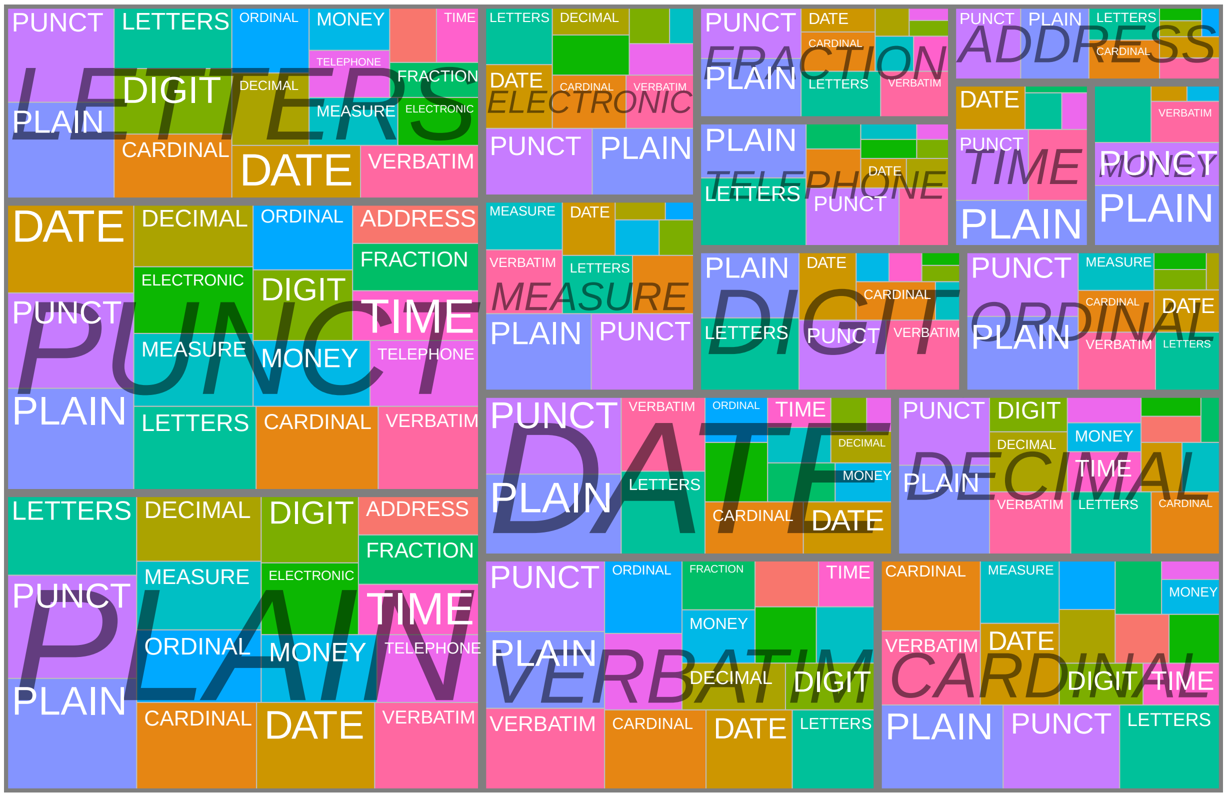

Figure 6: Fig. 38

```
t3 %>%
  group_by(class, class_next) %>%
  count() %>%
  ungroup() %>%
  mutate(n = log10(n+1)) %>%
  ggplot(aes(area = n, fill = class_next, label = class_next, subgroup = class)) +
  geom_treemap() +
  geom_treemap_subgroup_border() +
  geom_treemap_subgroup_text(place = "centre", grow = T, alpha = 0.5, colour =
                             "black", fontface = "italic", min.size = 0) +
  geom_treemap_text(colour = "white", place = "topleft", reflow = T) +
  theme(legend.position = "null") +
  ggtitle("Next classes grouped by token class; log scale frequencies")
```

Next classes grouped by token class; log scale frequencies

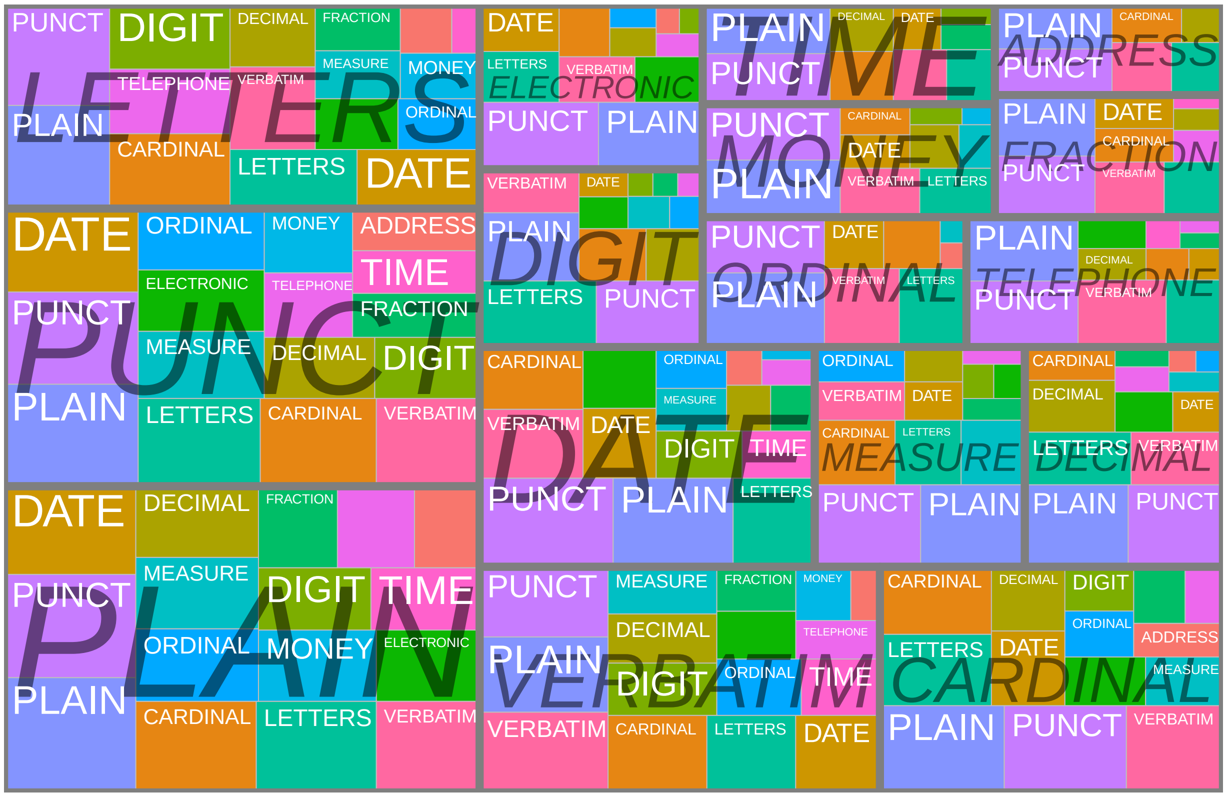

Figure 7: Fig. 39

The first plot shows the frequency of *previous* token *classes* and the second treemap the frequency of *next* token *classes* (all labeled in white) for each target token *class* (labeled in a large black font and separated by group with grey borders). We use a again a logarithmic frequency scaling to improve the visibility of the rare combinations. Group sizes decrease from the bottom left to the top right of the plot and each subgroup box. The colours of the white-labeled neighbour boxes are identical troughout the plot (e.g. "PUNCT" is always purple). Note, that the subgroup boxes in the two plots have different sizes for identical *classes* because of the `log(n+1)` transformation.

Relative percentages

All tokens

Based on these raw numbers we can study the relative contributions of a certain *class* to the *previous* or *next* tokens of another class. To visualise these dependencies, we again determine the `log10(n+1)` frequency distributions for each *class* among the *previous*/*next* tokens depending on the *class* of the reference token. These are the numbers in the bar plots above. We then normalise the range of these numbers (i.e. the height of the bars) to the interval `[0,1]` for each class. This data wrangling is done in the following code block:

```
prev_stat <- t3 %>%
  count(class, class_prev) %>%
  mutate(n = log10(n+1)) %>%
  group_by(class) %>%
  summarise(mean_n = mean(n),
            max_n = max(n),
            min_n = min(n))

next_stat <- t3 %>%
  count(class, class_next) %>%
  mutate(n = log10(n+1)) %>%
  group_by(class) %>%
  summarise(mean_n = mean(n),
            max_n = max(n),
            min_n = min(n))

t3_norm_prev <- t3 %>%
  count(class, class_prev) %>%
  mutate(n = log10(n+1)) %>%
  left_join(prev_stat, by = "class") %>%
  mutate(frac_norm = (n-min_n)/(max_n - min_n),
         test1 = max_n - min_n,
         test2 = n - min_n)

t3_norm_next <- t3 %>%
  count(class, class_next) %>%
  mutate(n = log10(n+1)) %>%
  left_join(next_stat, by = "class") %>%
  mutate(frac_norm = (n-min_n)/(max_n - min_n),
         test1 = max_n - min_n,
         test2 = n - min_n)
```

The result is displayed in two *tile plots* for the *class* vs *previous class* and *class* vs *next class*, respectively. Here, each tile shows the relative frequency (by *class*) of a specific neighbour pairing. The colour coding assigns bluer colours to lower frequencies and redder colours to higher frequencies:

```
t3_norm_prev %>%
  ggplot(aes(class, class_prev, fill = frac_norm)) +
  geom_tile() +
  theme(axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  scale_fill_distiller(palette = "Spectral") +
  labs(x = "Token class", y = "Previous token class", fill = "Rel freq") +
  ggtitle("Class vs previous class; relative log scale frequencies")
```

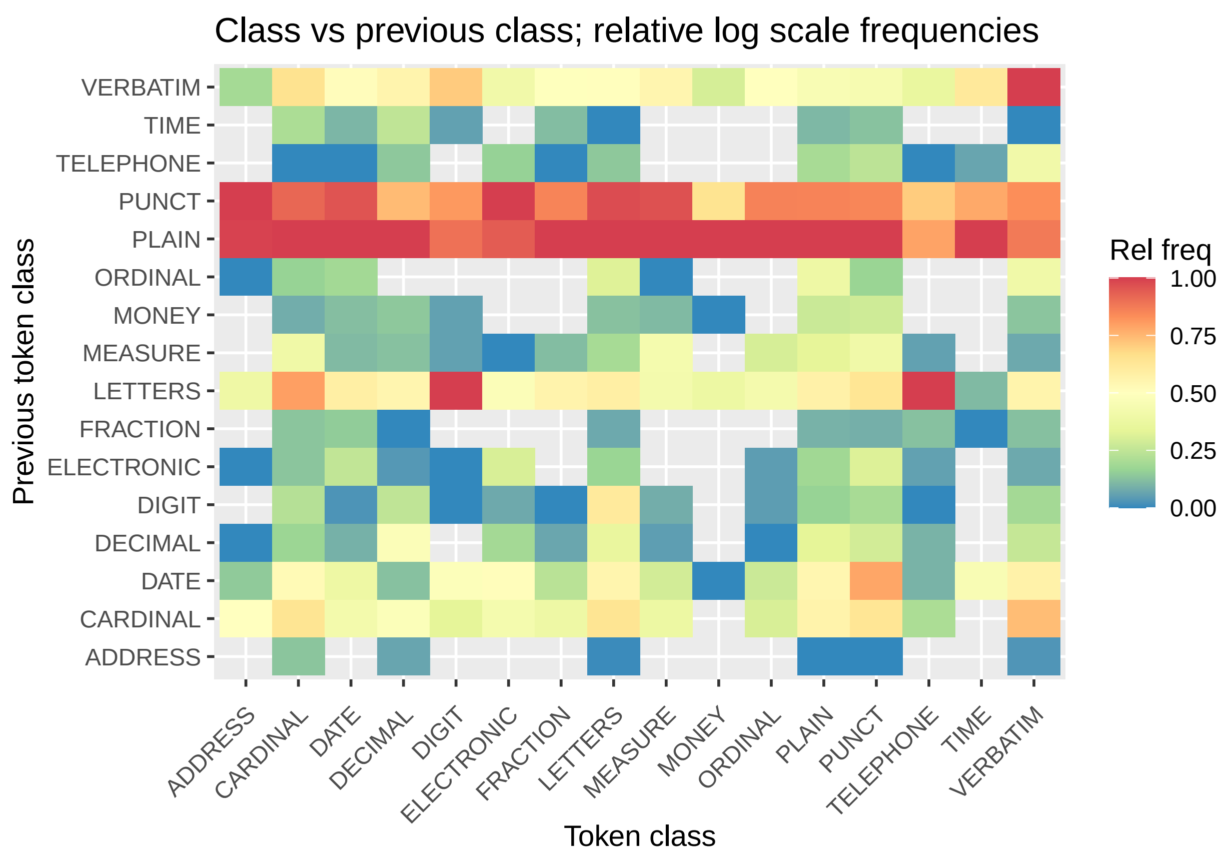

Figure 8: Fig. 40

```
t3_norm_next %>%
  ggplot(aes(class, class_next, fill = frac_norm)) +
  geom_tile() +
  theme(axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  scale_fill_distiller(palette = "Spectral") +
  labs(x = "Token class", y = "Next token class", fill = "Rel freq") +
  ggtitle("Class vs next class; relative log scale frequencies")
```

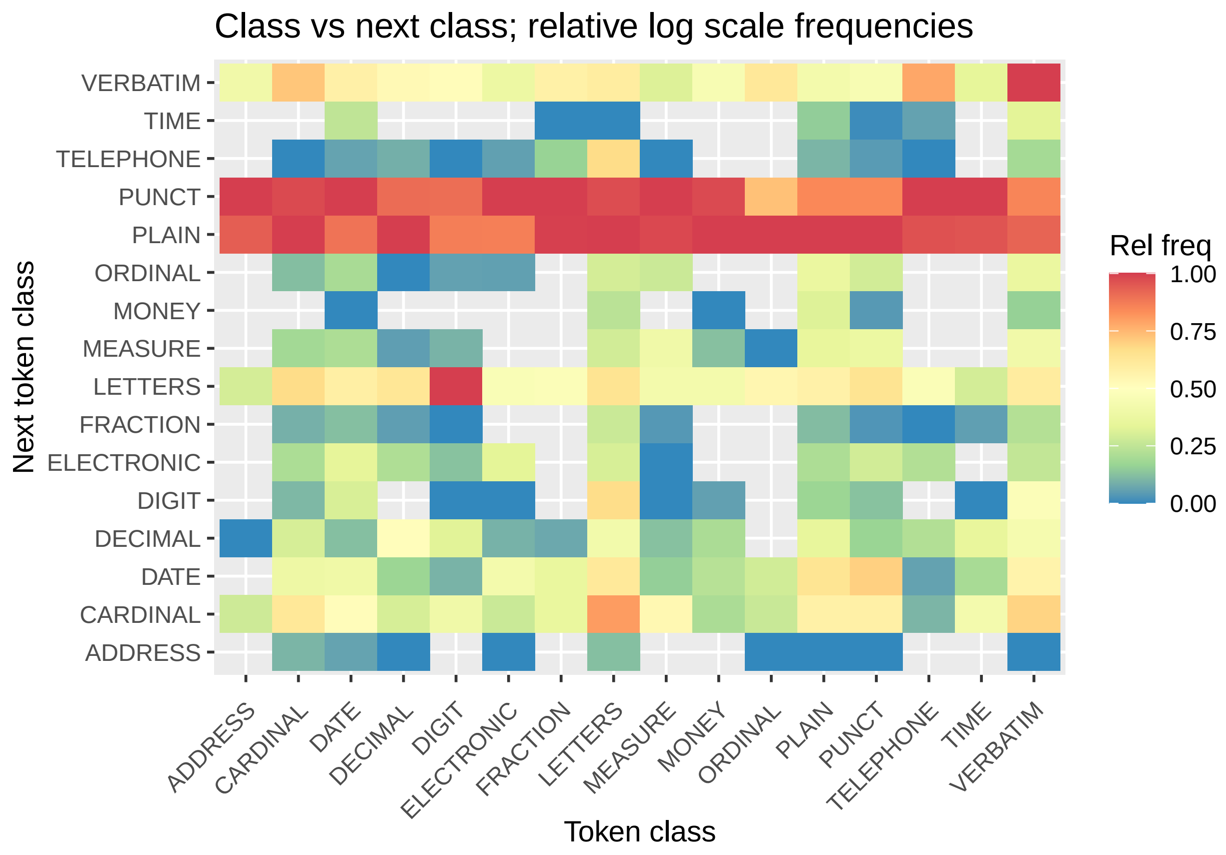

Figure 9: Fig. 41

We find:

- Not all possible neighbouring combinations exist: only about 77% of the tile plots are filled. Certain potential pairs such as "ORDINAL" and "FRACTION" or "TIME" and "MONEY" are never found next to each other.
- "PUNCT" and "PLAIN", the overall most frequent *classes*, also dominate the relative frequencies of *previous*/*next* classes for every token *class*. "PUNCT" is relatively weakly related to "MONEY" (*previous*) and "ORDINAL" (*next*).
- "DIGIT" tokens are often preceded or followed by "LETTERS" tokens.
- "TELEPHONE" is very likely to be preceded by "LETTERS", as well.
- "VERBATIM" tokens are very likely to be followed and preceded by other "VERBATIM" tokens. No other token *class* has such a strong correlation with itself. Some, such as "DIGIT", "MONEY", or "TELEPHONE" are rather unlikely to be preceded by the same *class*
- "VERBATIM" is also very unlikely to be preceded by "ADDRESS" and "TIME" or followed by "ADDRESS".

Transformed tokens only

Now we restrict this neighbour analysis to the *transformed* tokens only. Naturally, these will still have transformed or untransformed tokens in the *previous* and *next* positions. Having set up our "context" data frame with this option in mind, we only need to modify our code slightly to prepare the corresponding tile plots:

```r
prev_stat <- t3 %>%
  filter(transformed == TRUE) %>%
  count(class, class_prev) %>%
  mutate(n = log10(n+1)) %>%
  group_by(class) %>%
  summarise(mean_n = mean(n),
            max_n = max(n),
            min_n = min(n))

next_stat <- t3 %>%
  filter(transformed == TRUE) %>%
  count(class, class_next) %>%
  mutate(n = log10(n+1)) %>%
  group_by(class) %>%
  summarise(mean_n = mean(n),
            max_n = max(n),
            min_n = min(n))

t3_norm_prev <- t3 %>%
  filter(transformed == TRUE) %>%
  count(class, class_prev) %>%
  mutate(n = log10(n+1)) %>%
  left_join(prev_stat, by = "class") %>%
  mutate(frac_norm = (n-min_n)/(max_n - min_n),
         test1 = max_n - min_n,
         test2 = n - min_n)

t3_norm_next <- t3 %>%
  filter(transformed == TRUE) %>%
  count(class, class_next) %>%
  mutate(n = log10(n+1)) %>%
  left_join(next_stat, by = "class") %>%
  mutate(frac_norm = (n-min_n)/(max_n - min_n),
         test1 = max_n - min_n,
         test2 = n - min_n)
```

```r
t3_norm_prev %>%
  ggplot(aes(class, class_prev, fill = frac_norm)) +
  geom_tile() +
  theme(axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  scale_fill_distiller(palette = "Spectral") +
  labs(x = "Token class", y = "Previous token class", fill = "Rel freq") +
  ggtitle("Class vs previous class; relative log scale; transformed")
```

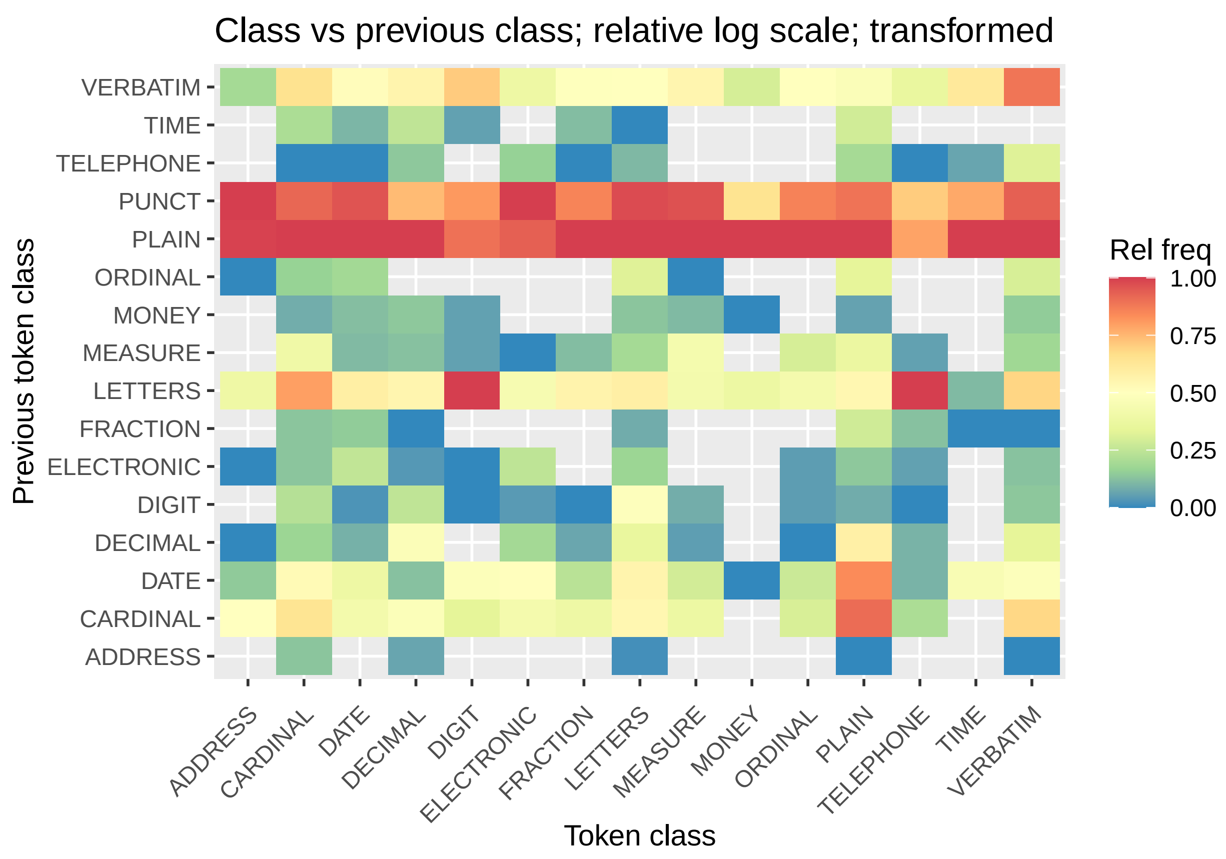

Figure 10: Fig. 42

```
t3_norm_next %>%
  ggplot(aes(class, class_next, fill = frac_norm)) +
  geom_tile() +
  theme(axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  scale_fill_distiller(palette = "Spectral") +
  labs(x = "Token class", y = "Next token class", fill = "Rel freq") +
  ggtitle("Class vs next class; relative log scale; transformed")
```

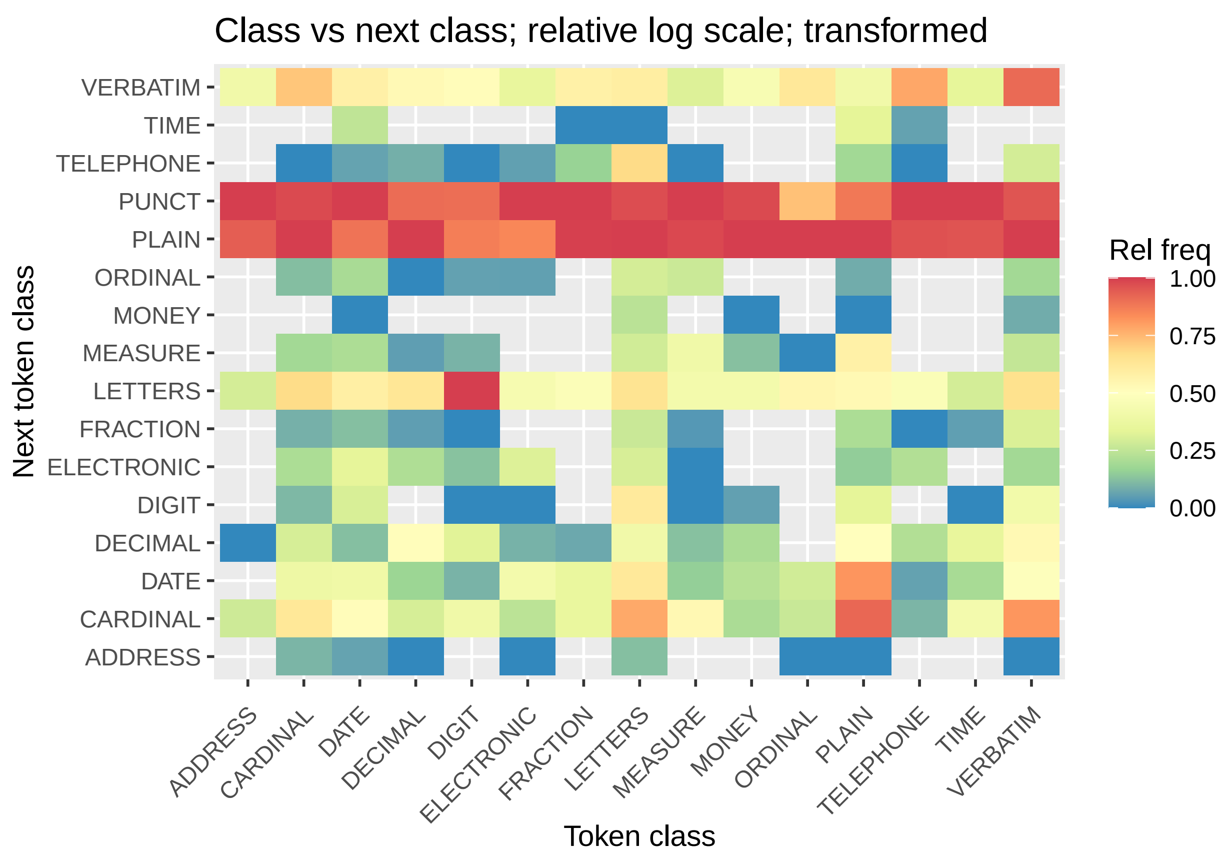

Figure 11: Fig. 43

We find:

- The *class* "PUNCT" is now gone from the "Token class" axis, since those symbols are never transformed.
- Two combinations disappear: "VERBATIM" is not transformed if preceeded or followed by "TIME" (relatively infrequent neighbours to start with).
- The pair of "PLAIN" preceded by "CARDINAL" significantly increases in frequency (within the rarely transformed "PLAIN" *class*).
- In general, the *classes* "PLAIN" and "VERBATM" are experiences the most visible changes with respect to the total set of neighbouring tokens since these are the *classes* with the highest percentage of untransformed tokens (after "PUNCT", of course).

Summary

Certain combinations of classes are more likely to be found next to one another than other combinations. Ultimately, this reflects the grammar structure of the language.

By making use of these next-neighbour statistics we can estimate the probability that a token was classified correctly by iteratively cross-checking the other tokens in the same sentence. **This adds a certain degree of context to a classification/normalisation attempt that is only considering the token itself.**

Transformation statistics

This section will look at specific sentence parameters and how they affect the transformation statistics or behaviour.

Sentence length and classes

We begin by studying how the *length* (token lengths) of the sentences affects the statistics of *classes* and transformed tokens. For this we estimate the mean transformed fraction for each group of sentences with the same length together with corresponding uncertainties. In addition, we examine the *class* frequencies for the shortest sentences and the sentence length distributions for each *class*:

```
foo <- train %>%
  group_by(sentence_id, transformed) %>%
  count() %>%
  spread(transformed, n, fill = 0) %>%
  mutate(frac = `TRUE`/(`TRUE` + `FALSE`),
         sen_len = `TRUE` + `FALSE`)

bar <- foo %>%
  group_by(sen_len) %>%
  summarise(mean_frac = mean(frac),
            sd_frac = sd(frac),
            ct = n())

foobar <- foo %>%
  left_join(train, by = "sentence_id")
```

```
p1 <- bar %>%
  ggplot(aes(sen_len, mean_frac, size = ct)) +
  geom_errorbar(aes(ymin = mean_frac-sd_frac, ymax = mean_frac+sd_frac),
                width = 0., size = 0.7, color = "gray30") +
  geom_point(col = "red") +
  scale_x_log10() +
  labs(x = "Sentence length", y = "Average transformation fraction")

p2 <- foobar %>%
  filter(sen_len < 6) %>%
  group_by(class, sen_len) %>%
  count() %>%
  ungroup() %>%
  filter(n > 100) %>%
  ggplot(aes(sen_len, n, fill = class)) +
  geom_col(position = "fill") +
  scale_fill_brewer(palette = "Set1") +
  labs(x = "Sentence length", y = "Proportion per class")

p3 <- foobar %>%
  ggplot(aes(sen_len)) +
  geom_density(bw = .1, size = 1.5, fill = "red") +
  scale_x_log10() +
  labs(x = "Sentence length") +
  facet_wrap(~ class, nrow = 2)

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
p1 <- 1; p2 <- 1; p3 <- 1
```

We find:

- For sentences with more than 10 tokens the proportion of transformed tokens first decreases somewhat up to about 20 and then increases slightly afterwards. However, none of the changes appear to be significant within their standard deviations. Here, the size of the red data points is proportional to the number of cases per group.
- Interestingly, for very short sentences of only 2 or 3 tokens the mean fraction of transformed tokens is significantly higher: about 33% for 3 tokens and ** practically always** 50% for 2 tokens.
- Upon closer inspection in the bar plot we find that **almost all 2-token sentences consist of a "DATE" *class* and a "PUNCT" *class* token.** Note that our plot omits rare *classes* with less than 100 cases for better visibility. Below is the complete table for the 2-token sentences.
- The 3-token sentences basically just add a "PLAIN" token to the mix, reducing the proportions of "DATE" and "PUNCT" tokens to 1/3 each. For longer sentences, the mix starts to become more heterogenenous. Interestingly, for longer sentences the "PUNCT" fraction does not continue to decline geomtrically - indicating the presence of tokens other than the final full stop.
- Among the individual *classes* we can see considerable differences in the shape of their sentence length distributions. For instance, "VERBATIM" has a far broader distribution that most; reaching notable number above 100 tokens. "PLAIN", "MEASURE", and "DECIMAL" have the sharpest peaks, while "DATE" has an interesting step structure in addition to the dominance in short sentences that we discussed above (here is the promised table for 2-token sentences:)

```
foobar %>%
  filter(sen_len <= 2) %>%
  group_by(sen_len, class, transformed) %>%
  count()
```

```
## # A tibble: 5 x 4
## # Groups:   sen_len, class, transformed [5]
##   sen_len class     transformed     n
##     <dbl> <fct>     <lgl>       <int>
## 1       2 DATE      TRUE        10127
## 2       2 PLAIN     FALSE           3
## 3       2 PLAIN     TRUE            1
## 4       2 PUNCT     FALSE       10133
## 5       2 TELEPHONE TRUE           10
```

Normalised tokens and where to find them

Let's determine whether the position of a token in a sentence reveals something about the probability of token being transformed.

```r
foo <- train %>%
  group_by(sentence_id) %>%
  summarise(sentence_len = max(token_id))

p1 <- train %>%
  ggplot(aes(token_id, fill = transformed)) +
  geom_density(bw = .1, alpha = 0.5) +
  scale_x_log10() +
  theme(legend.position = "none")

p2 <- train %>%
  left_join(foo, by = "sentence_id") %>%
  mutate(token_rel = token_id/sentence_len) %>%
  ggplot(aes(token_rel, fill = transformed)) +
  geom_density(bw = .05, alpha = 0.5) +
  labs(x = "Relative token position")

p3 <- train %>%
  left_join(foo, by = "sentence_id") %>%
  mutate(token_rel = token_id/sentence_len) %>%
  ggplot(aes(token_rel, fill = transformed)) +
  geom_density(bw = .05, alpha = 0.5) +
  labs(x = "Relative token position") +
  facet_wrap(~class, nrow = 2) +
  theme(legend.position = "none")

layout <- matrix(c(1,2,1,2,3,3,3,3,3,3),5,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```

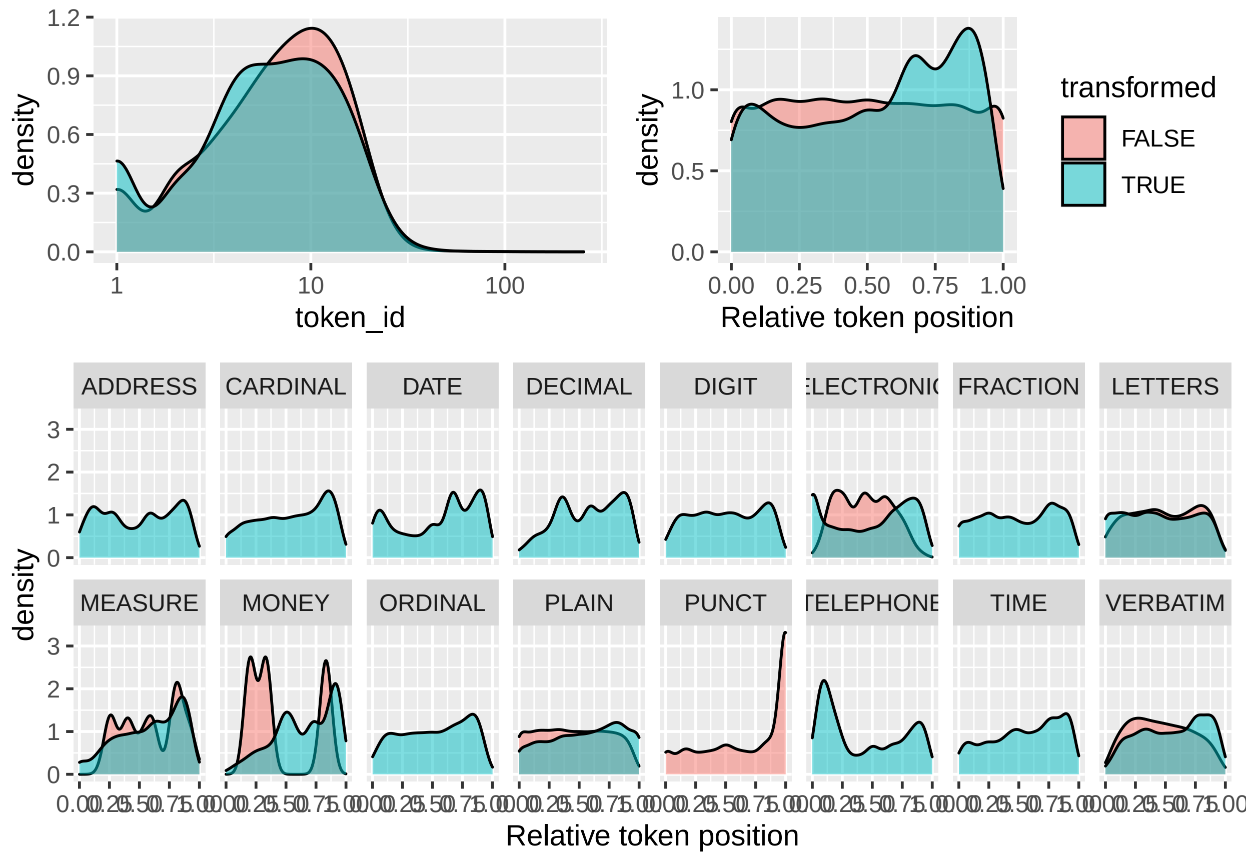

Figure 12: Fig. 47

```
p1 <- 1; p2 <- 1; p3 <- 1
```

We find:

- Tokens around absolute positions of 10 have notably fewer transformations that others. We also the effect of the 50% transformations for 2-token sentences. There is a small effect for the highest *token_ids* but it doesn't look significant.
- In terms of relative position we find that transformed tokens are more likely to be found in the second half of a sentence compared to the first half. An exception is the very end of the sentence where we most likely see the influence of the untransformed punctuation. It is interesting how much more evenly distributed the non-transformed tokens are.
- By facetting the second plot by *class* recover some of the information we had seen in the boxplots above, but we can also clearly see the impact of the normalisation on the individual *classes*. The most interesting distributions are clearly "ELECTRONIC", more likely to transformed early and late in a sentence, and "MONEY", practically the opposite. But also "VERBATIM" and "MEASURE" have interesting features.

Feature engineering

We have already derived *sentence length* and *transformed* features. Now we want to engineer a few more variables that might be useful for our understanding of the data and our ultimate prediction goal. The following subsections will examine these features and their impact on the token normalisation.

```r
train <- train %>%
  rowid_to_column(var = "row_id")

# class-specific distances
#---
classes <- train %>%
  select(class) %>%
  unique() %>%
  unlist(use.names = FALSE) %>%
  as.character()

cl_dist <- NULL
for (i in classes){

  cl_dist <- train %>%
    filter(class == !!quo(i)) %>%
    mutate(cl_dist = pmax(0,lead(token_id,1)-token_id)*pmax(0,sentence_id-lead(sentence_id,1)+1)) %>%
    select(row_id, cl_dist) %>%
    bind_rows(cl_dist)
}
cl_dist <- cl_dist %>%
  mutate(cl_dist = as.integer(cl_dist)) %>%
  replace_na(list(cl_dist = 0)) %>%
  arrange(row_id)
#---

sen_len <- train %>%
  group_by(sentence_id) %>%
  summarise(sentence_len = max(token_id))

train <- train %>%
  mutate(str_len = str_length(before),
         cl_dist = cl_dist$cl_dist,
         num = !is.na(as.numeric(before)),
         is_name = before %in% babynames$name
         ) %>%
  left_join(sen_len, by = "sentence_id")
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```r
t3 <- t3 %>%
  mutate(is_name = train$is_name[2:(nrow(train)-1)])
```

String length

We start with a pretty basic feature: the length of the *before* token string in numbers of characters. These are the overall distribution, the impact on the “transformed” status, and the string lengths by class:

```r
p1 <- train %>%
  ggplot(aes(str_len)) +
  geom_histogram(fill = "red", bins = 50) +
  scale_x_log10() +
  scale_y_log10()

p2 <- train %>%
  ggplot(aes(str_len, fill = transformed)) +
  geom_density(alpha = 0.5, bw = 0.1) +
  scale_x_log10()

p3 <- train %>%
  ggplot(aes(class, str_len, col = class)) +
  geom_boxplot() +
  scale_y_log10() +
  theme(legend.position = "none", axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9))

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```

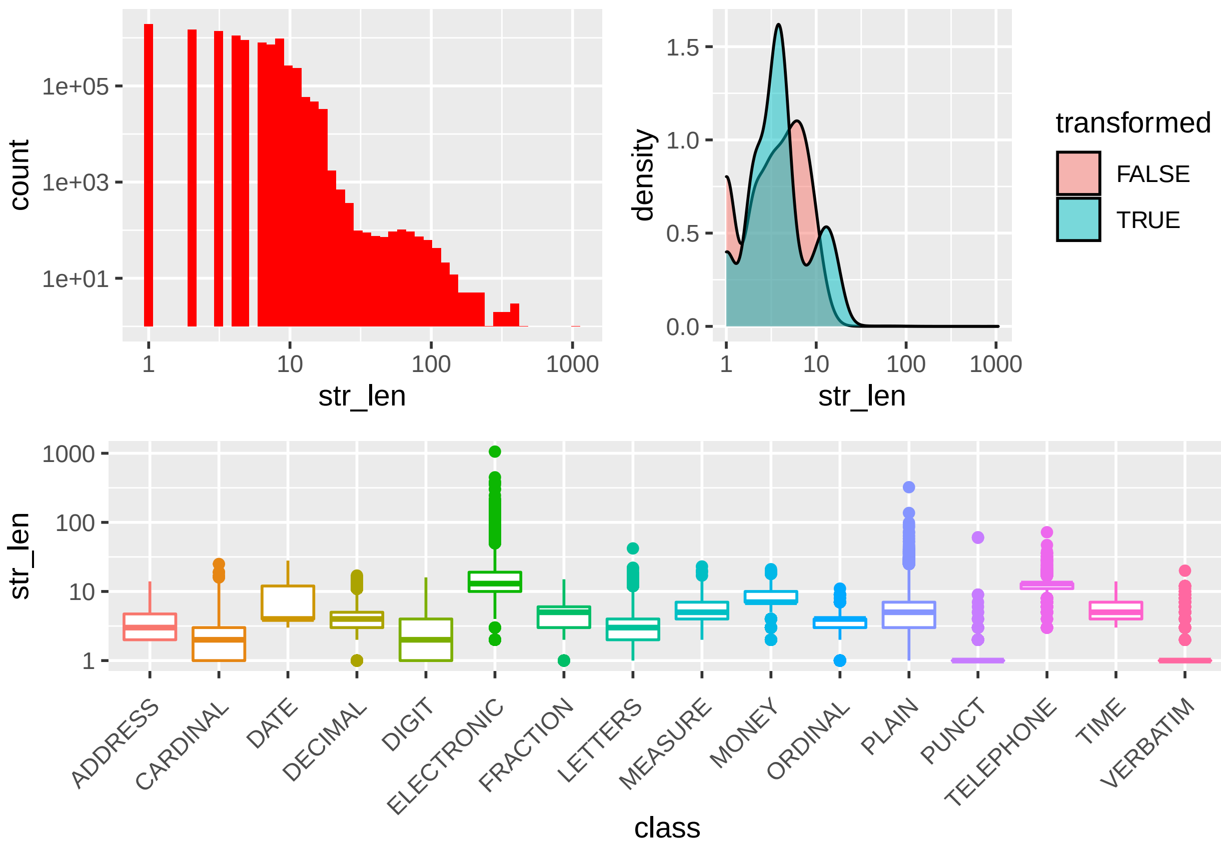

Figure 13: Fig. 47

```
p1 <- 1; p2 <- 1; p3 <- 1
```

We find:

- Short strings are dominant in our data (note the log axes) but longer ones with up to a few 100 characters can occur.
- The transformed vs untransformed string lengths reveal an interesting multi-modal distribution where transformed strings are likely to be very long or just a few characters long. Very short strings are more likely to be untransformed, which might be due to the impact of the "PUNCT" class since punctuation usually is only a single character.
- Looking closer at the boxplots of *string length* per *class* there are significant differences. We confirm that "PUNCT" tokens are typically short, but the same appears to be true for "VERBATIM" tokens. "ELECTRONIC" tokens make up the majority of the longest tokens and, together with "TELEPHONE", are the only ones with a median above 10 characters.

From these plots, it's definitely worth it to examine the transformed vs untransformed statistics in more detail.

Analysis after splitting the corresponding density plots by *class*:

```
train %>%
  ggplot(aes(str_len, fill = transformed)) +
  geom_density(alpha = 0.5, bw = 0.1) +
  scale_x_log10() +
  facet_wrap(~ class)
```

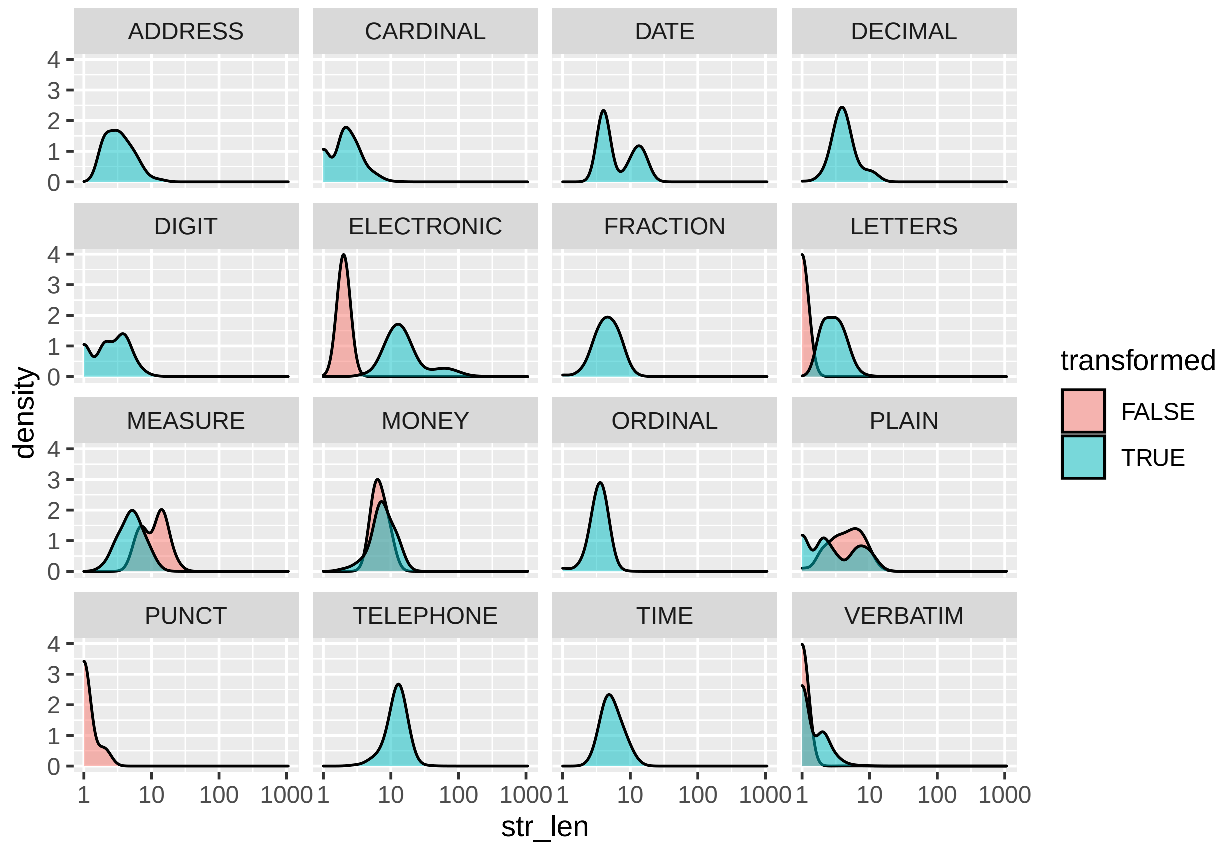

Figure 14: Fig. 48

We find:

- Most classes are always transformed and we see the difference in their overall string length distribution; e.g. “TELEPHONE” peaks just above 10 vs “TIME” peasking below 10, or “DIGIT”, “ADDRESS”, “CARDINAL”, and “FRACTION” tokens being predominantly shorter than 10 characters. “PUNCT” is short and never transformed.
- There’s a really interesting bimodality in the “ELECTRONIC” class, with short tokens never transformed and longer tokens always transformed. The overlap between the two populations appears to be marginal. The same observation is true for the “LETTERS” class, albeit with a larger overlap.
- A similar effect, although less pronounced, can be seen in the “MEASURE” class, where untransformed tokens are on average longer than transformed ones. For “VERBATIM” we see that longer tokens are always transformed.

Class-specific token distance

We define a measure called *class-specific token distance*. It can thought of simply as the number of tokens it takes to get from one specific token to the next token of the same *class*. For instance, if the sentence goes like “DATE PLAIN PLAIN DATE PUNCT” then the *distances* would be “3 1 0 0 0”: It takes 3 steps to get from the first “DATE” token to the next “DATE” token, and 1 step to get from the first “PLAIN” token to the one right next to it. The last three tokens don't have another token of the same *class* following them **in the same sentence**, so we assign a distance of zero.

The aim here is to get an idea of how the different *classes* cluster. Are "LETTER" tokens more likely to follow each other? Are "ORDINAL" tokens more often far apart? Knowing these patterns could help us in optimising the *classification* of a sentence and thereby its normalisation patterns.

We start out with a bar plot that compares how often a *class* occurs alone in a sentence (or as the last token of its kind), resulting in a zero distance, versus how often the *class* can be found multiple times in addition to that last token. Here, two tokens per *class* per sentence would give us a 50/50 split:

```r
train %>%
  ggplot(aes(cl_dist==0, fill = cl_dist==0)) +
  geom_bar() +
  theme(legend.position = "none") +
  scale_y_log10() +
  facet_wrap(~class) +
  labs(x = "Alone or the last token of its kind in a sentence")
```

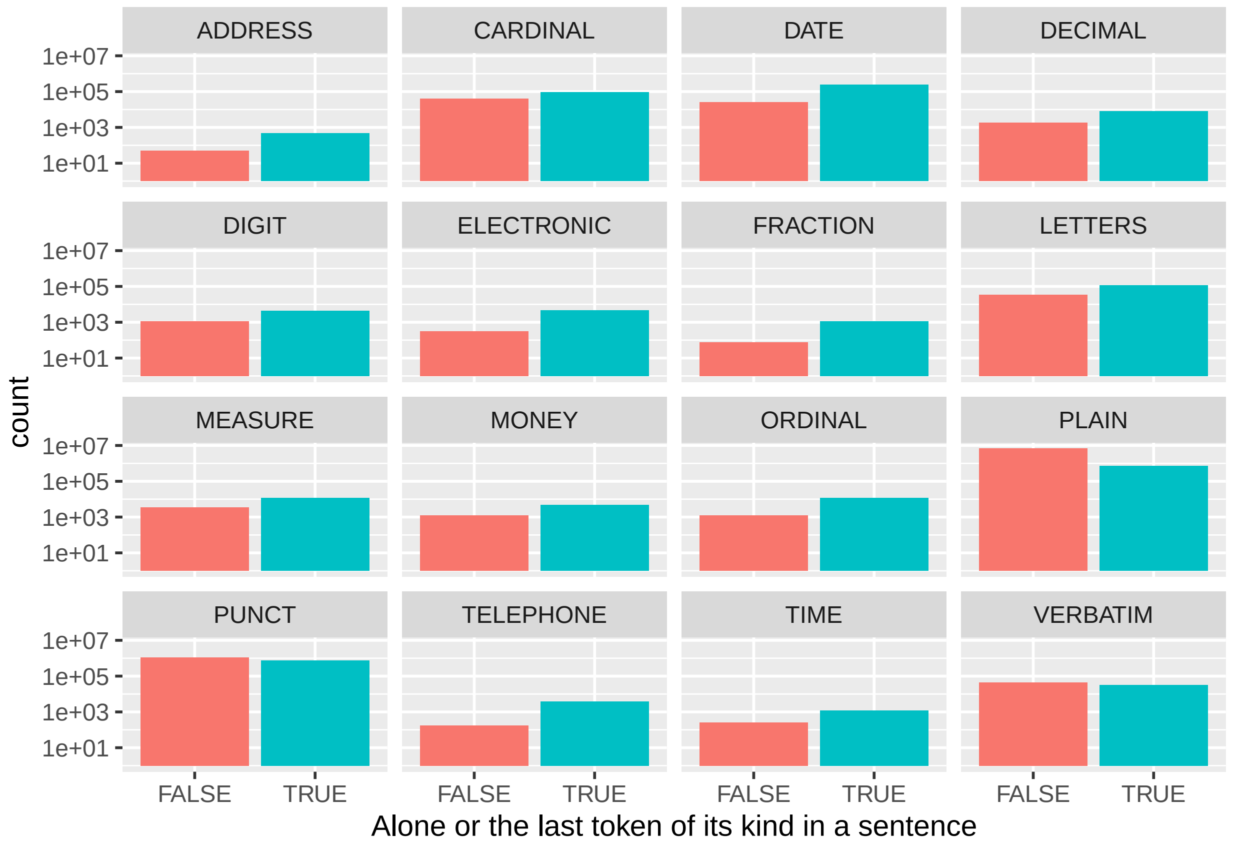

Figure 15: Fig. 49

We find:

- Most *classes* are predominantly alone in their sentences, except for "PLAIN", "PUNCT", and "VERBATIM". In particular the "PLAIN" tokens are likely to be clustered together. Note the logarithmic y-axes.

What's in a name?

The next feature will be all about names. In order to determine whether a token is a name, we will use Hadley Wickham's `babynames` package which includes comprehensive data, from the year 1880 to 2015, provided by the US Social Security Administration. Interestingly, this includes entries such as "Ramses" or "Socrates", which will be useful for historical documents.

```r
p1 <- t3 %>%
  group_by(class, is_name) %>%
  count() %>%
  spread(is_name, n, fill = 0) %>%
  mutate(frac_name = `TRUE`/(`TRUE`+`FALSE`)*100) %>%
  filter(frac_name > 0) %>%
  ggplot(aes(reorder(class, -frac_name, FUN = min), frac_name, fill = class)) +
  geom_col() +
  geom_text(aes(reorder(class, -frac_name, FUN = min), frac_name+0.3, label = sprintf("%2.2f", frac_nam
  labs(x = "Class", y = "Name percentage") +
  theme(legend.position = "none")

p2 <- t3 %>%
  group_by(class_next, is_name) %>%
  count() %>%
  spread(is_name, n, fill = 0) %>%
  mutate(frac_name = `TRUE`/(`TRUE`+`FALSE`)*100) %>%
  filter(frac_name > 0) %>%
  ggplot(aes(reorder(class_next, -frac_name, FUN = min), frac_name, fill = class_next)) +
  geom_col() +
  labs(x = "Next class", y = "Name percentage") +
  theme(legend.position = "none", axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9))

p3 <- t3 %>%
  group_by(class_prev, is_name) %>%
  count() %>%
  spread(is_name, n, fill = 0) %>%
  mutate(frac_name = `TRUE`/(`TRUE`+`FALSE`)*100) %>%
  filter(frac_name > 0) %>%
  ggplot(aes(reorder(class_prev, -frac_name, FUN = min), frac_name, fill = class_prev)) +
  geom_col() +
  labs(x = "Previous class", y = "Name percentage") +
  theme(legend.position = "none", axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9))

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```

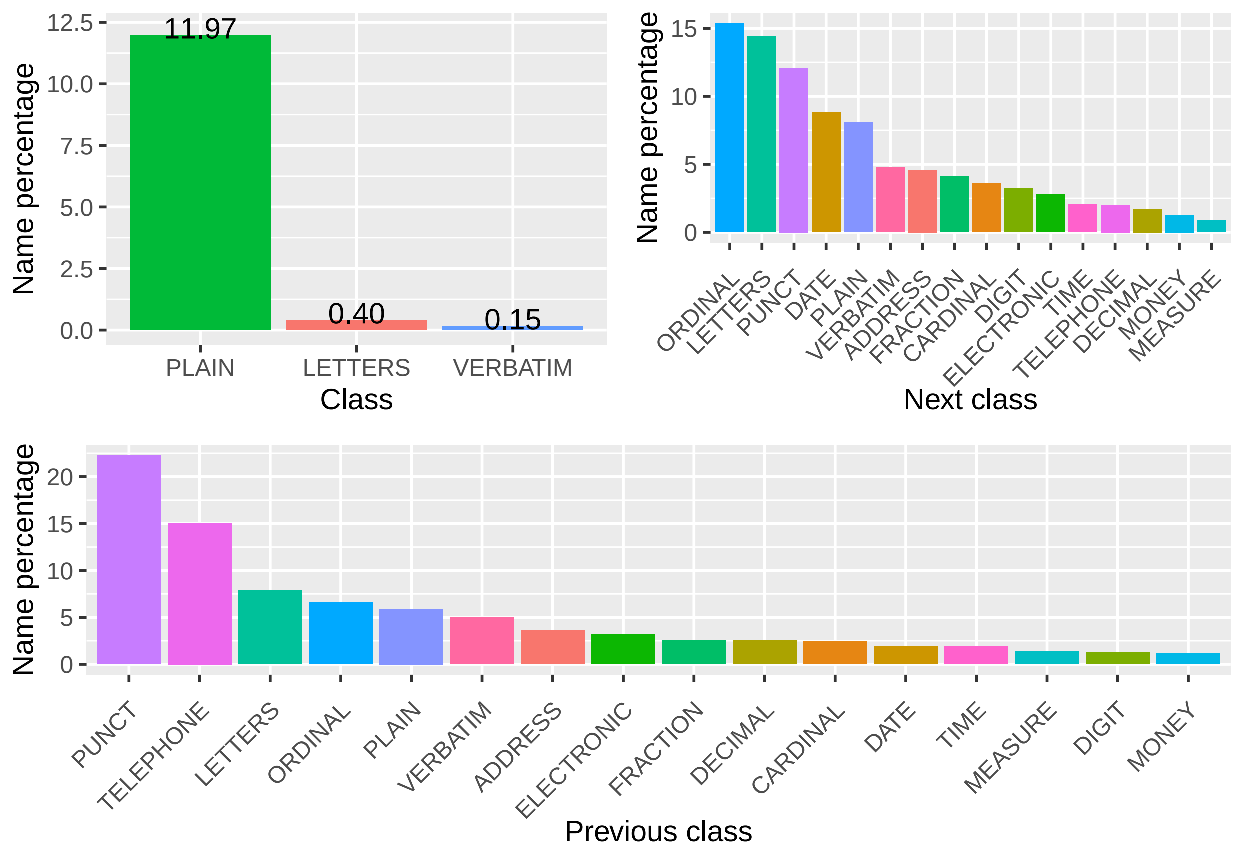

Figure 16: Fig. 52

```
p1 <- 1; p2 <- 1; p3 <- 1
```

We find:

- Almost 12% of "PLAIN" tokens are names. This is a notable percentage, especially since "PLAIN" tokens are the clear majority of tokens in our data set. This makes names an important sub-class and a pretty strong predictor for the "PLAIN" class. In addition, there are a few names in the "LETTERS" and "VERBATIM" *classes*.
- Comparing the relative fraction of *classes* immediately following or preceding a *name* token we see interesting features: "Ordinal" and "LETTERS" tokens are more likely to be found right after a name than other *classes*, whereas "PUNCT" tokens occur relatively frequently right before a name. The otherwise frequent "PLAIN" tokens are of less importance here.

Summary

After carrying out this extensive Exploratory Data Analysis on this Dataset, it was possible to successfully bring out the relationships between elements within a sentence and also the grammar system used. This understanding will help in transforming written expressions to appropriate spoken forms effectively and can be used to train Smart Voice Assistants and be more human like with their conversations.