TidytextPlot_vignette

Introduction

This vignette describes the content and use of the ***TidytextPlot*** package to transform a dataframe into tidy data structure and plot a graph of change in proportion of given word(s) over time, where time is indicated by turn. A cleaned version of dialogues for the three main presidential debates in the 2012 US presidential debate is provided for demonstration. The R package consist of two functions; `createTidyData` and `plotDebateWords`. The `createTidyData` fuction, transforms input dataframe into a tidy text data structure, while function `plotDebateWords`, calls the `createTidyData` function, and generates a ggplot2 graphics. The vignette also illustrate the use of `bind_tf_idf` function from the `tidytext` package to calculate the ***tf-idf indexes***, which is the product of the ***term frequency (tf)*** and the ***inverse document frequency (idf)***. The vignette also presents and describes a R code that test for ***Zipf's law*** in the debate data.

Installation and Usage

Step 1: Install package
install.packages("path/to/TidytextPlot_1.0.0.11.tar.gz", repos = NULL)

Step 2: Load package
library(TidytextPlot)

Step 3: Load required dependencies: ggplot2, dplyr, tidytext

Step 4: Create plot using included debateData
plotDebateWords(c("china","people","military","iran","energy"),debateData)

or

plotDebateWords(c("china","people","military","iran","energy"))
(no need to state second argument if you are using the debateData provided with the package)

Optional Step: To only transform input data to tidy text data structure, use `createTidyData` function;e.g.
tidyoutput <- createTidyData(debateData)

The dataset

The provided data "debateData" is a dataframe containing a cleaned version of the three main predidential debates for the 2012 US election. The variables are as follows:
* person: the speaker. Obama and Romney were the candidates, Crowley, Lehrer,Schieffer were the moderators and "question" indicates questions raised by the public.
* dialogue: the words spoken by each person.
* turn: progressive number indicating the turn of talk.

R Function Showing Change of Word Frequency Over Time

Function `createTidyData` reads in the data and transform it to a tidytext (Wickham,H 2014) data structure.

```
# createTidyData function takes as argument, an input dataframe
# with three columns, namely: 'person', 'dialogue' and 'turn'.
createTidyData <- function(input_DF) {
#
# Test that the input data is in the right format, and in particular,
# the columns names are; 'person', 'dialogue' and 'turn'.
#
  if(!is.data.frame(input_DF) || any(names(input_DF) != c('person', 'dialogue', 'turn')))
    {
    stop("input_DF must be a data.frame with columns 'person','dialogue' and
         'turn' to continue")
  }
#
# The dataframe can now be transformed into a tidy data. This is done by tokenizing
# the texts under the dialogue column using the unnest_tokens function in the tidytext
# package. This process creates the tokenized text under the new column "word".
  unnest_tokens(input_DF, word, dialogue) %>%
# Remove "stop words" with the anti_join function in dplyr.
  anti_join(stop_words) %>%
#
# Calculate the proportion of word for each turn using tools from dplyr package
#
  count(turn,word) %>%
  group_by(turn) %>%
  mutate(p=n/sum(n))
#
# End of the function.It returns the turn, the tokenized word, word count and the
# proportion of words within each turn.
}
```

Function test 1: Let's test our function on the debateData included in the package

```
test1 <- createTidyData(debateData)
```

View the first few lines of the result

```
test1
```

```
## # A tibble: 12,201 x 4
## # Groups:   turn [2,657]
##      turn word              n     p
##     <dbl> <chr>         <int> <dbl>
##  1      1 care              1 0.2
##  2      1 health            1 0.2
##  3      1 moment            1 0.2
##  4      1 specifically      1 0.2
##  5      1 talk              1 0.2
##  6      2 governor          1 0.25
##  7      2 support           1 0.25
##  8      2 system            1 0.25
##  9      2 voucher           1 0.25
## 10      3 change            1 0.167
## # ... with 12,191 more rows
```

Function test 2: We also want to test our function on a fictitious dataset. So, let's generate a fictitious dataframe having the required column names.

```r
fict_df<-data.frame(person=c("John","Todd","Texas","Rama"),
dialogue=c("We gather here", "meet basic needs","people of this nation",
"focus on implementation"),
turn=c(1,2,3,4),
stringsAsFactors = FALSE)
```

Run the function test

```r
test2 <- createTidyData(fict_df)
```

View the first few lines of the output

```r
test2
```

```
## # A tibble: 7 x 4
## # Groups:   turn [4]
##    turn word               n     p
##   <dbl> <chr>          <int> <dbl>
## 1     1 gather             1   1
## 2     2 basic              1   0.5
## 3     2 meet               1   0.5
## 4     3 nation             1   0.5
## 5     3 people             1   0.5
## 6     4 focus              1   0.5
## 7     4 implementation     1   0.5
```

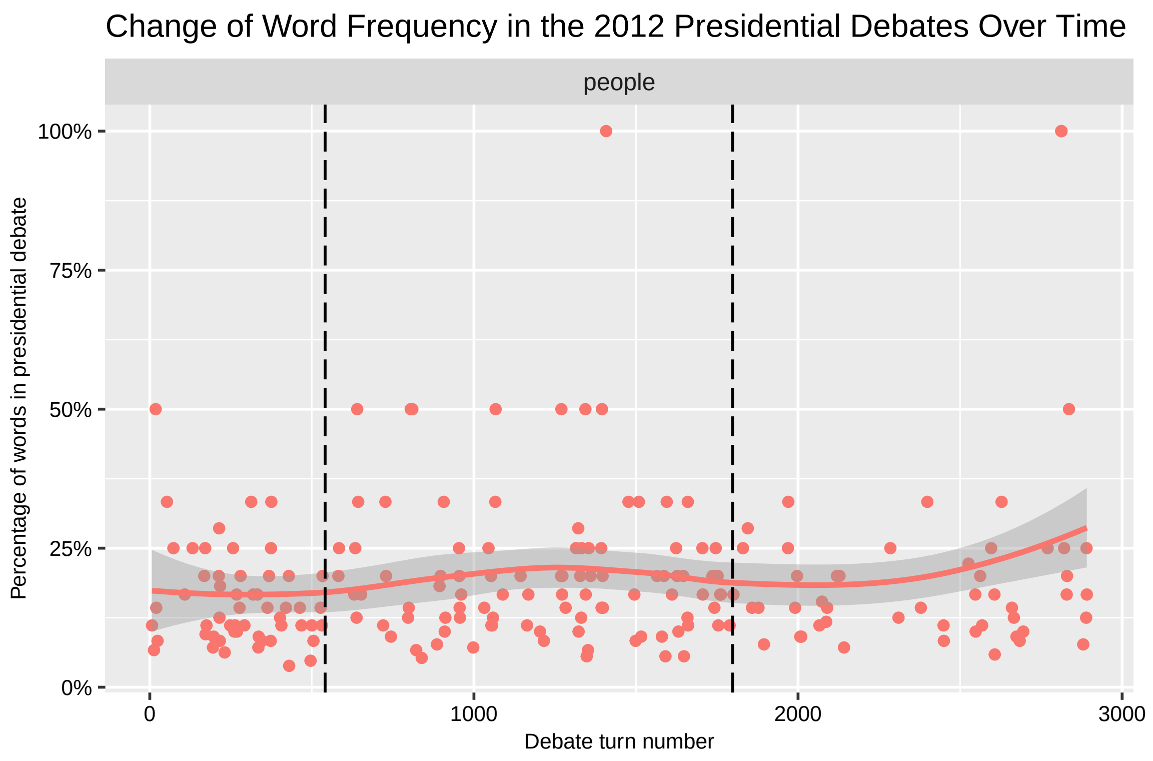
Good, the `createTidyData` function work well on the provided data and a fictitious data.

Now to define the `plotDebateWords` function. This function calls the `createTidyData` function, and produce a ggplot2 graphics.

```r
#
# The plotDebateWords function takes two inputs as argument. The first
# argument "word" is set to NULL as default - so  that during the function call, the
# user can specify the word or words required to plot. The second argument is set to
# the provided debateData as default. The second argument is however "optional". If the
# debate data included in this package is used, the second argument is not required.
# However if user is using own data, the second argument is required.
#
plotDebateWords <- function(word=NULL,data=debateData) {
#
# Call "createTidyData" function to transform the input data (second argument of
# our function) into tidytext data structure.
#
tidyData <-createTidyData(data)
#
# The next two lines check if the first argument is set to null, and if not null, it
# retrieve from the data, a subset given by the non-null argument. With %in%, we can use
# as many words as argument as required.
#
  if(!is.null(word)) {
    tidyData<-tidyData[tidyData$word %in% word,]
  }
#
# Here we plot the data using ggplot
#
plot<-tidyData %>%
    ggplot(aes(turn,p,colour=word)) +
    geom_point() +
    geom_smooth() +
    geom_vline(xintercept = c(541,1798), # Line specifying end of first and
               # second debate.
               linetype="longdash",
               colour="black",
               size=0.5) +
    facet_wrap(~word, scales="free_y") +
    scale_y_continuous(labels=scales::percent_format()) +
    xlab("Debate turn number") +
    ylab("Percentage of words in presidential debate") +
    ggtitle("Change of Word Frequency in the 2012 Presidential Debates Over Time") +
    theme(legend.position = "none") +
  theme(axis.text = element_text(size = 8, color = "black"),
        axis.title = element_text(size = 8, color = "black"),
        title = element_text(size = 10))
#
# End of function. It generate a ggplot graphic showing change in proportion of
# given words over time. The words are provided as the first argument.
return(plot)
}
#
```

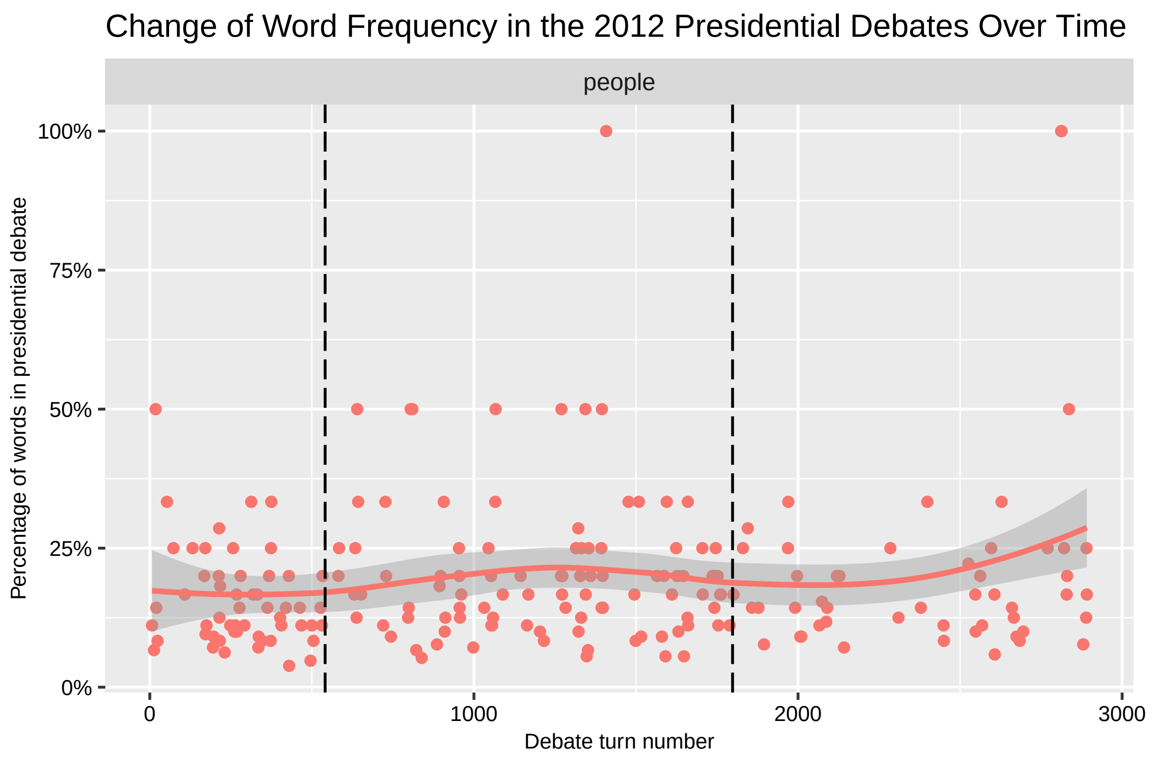
Function test 3: We can now test the second function using the provided debateData

```
plotDebateWords("people",debateData)
```



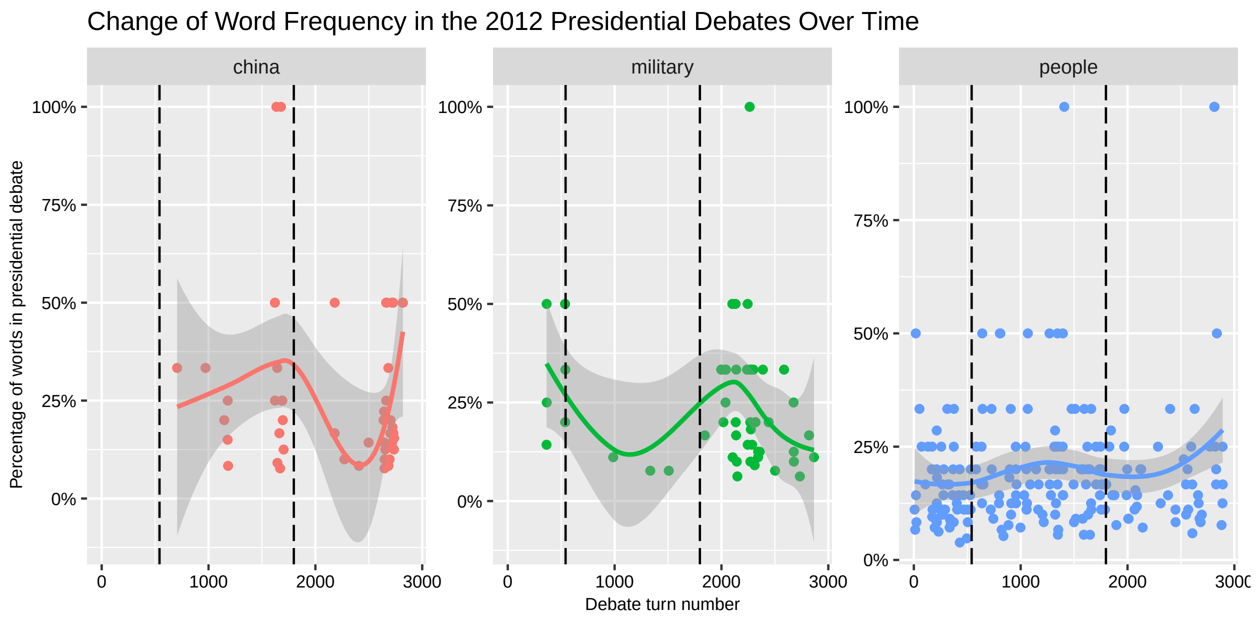
Function test 4:As stated however, since we are using the provided data, we do not need to use the second argument. So we can do,

```
plotDebateWords("people")
```



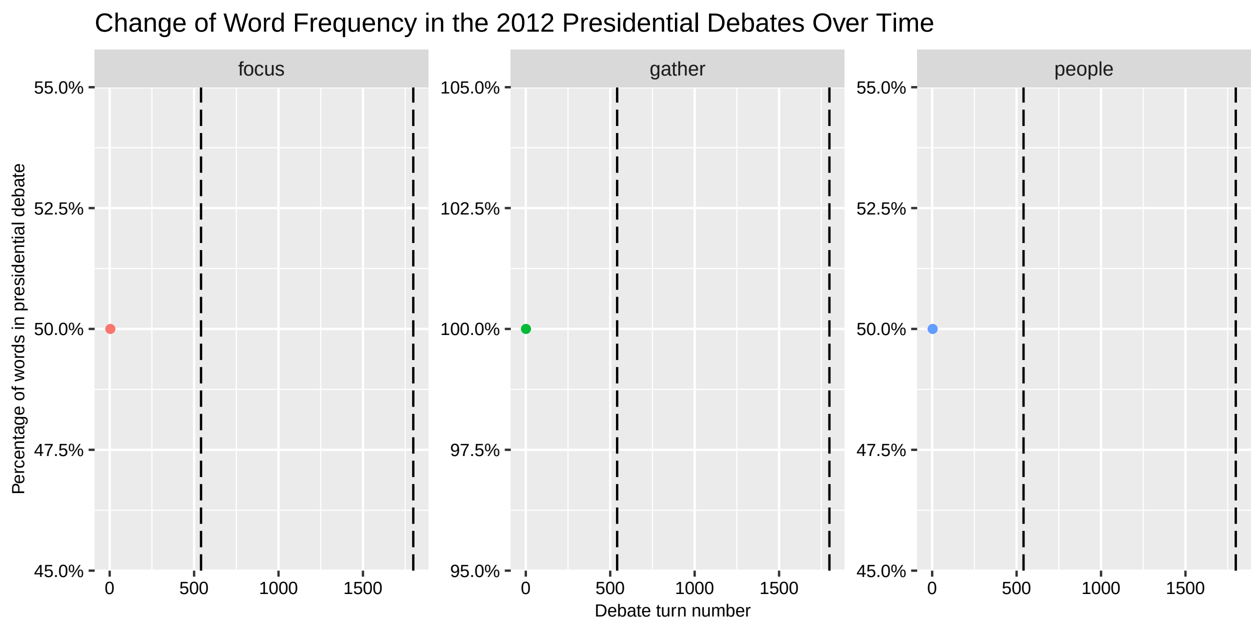
Function test 5:Test with more than one word with provided data.

```
plotDebateWords(c("people","china","military"))
```



Function test 6: We now test the second function on the fictitious data created in page 3. Note that we have to state second argument because we are using a different data.

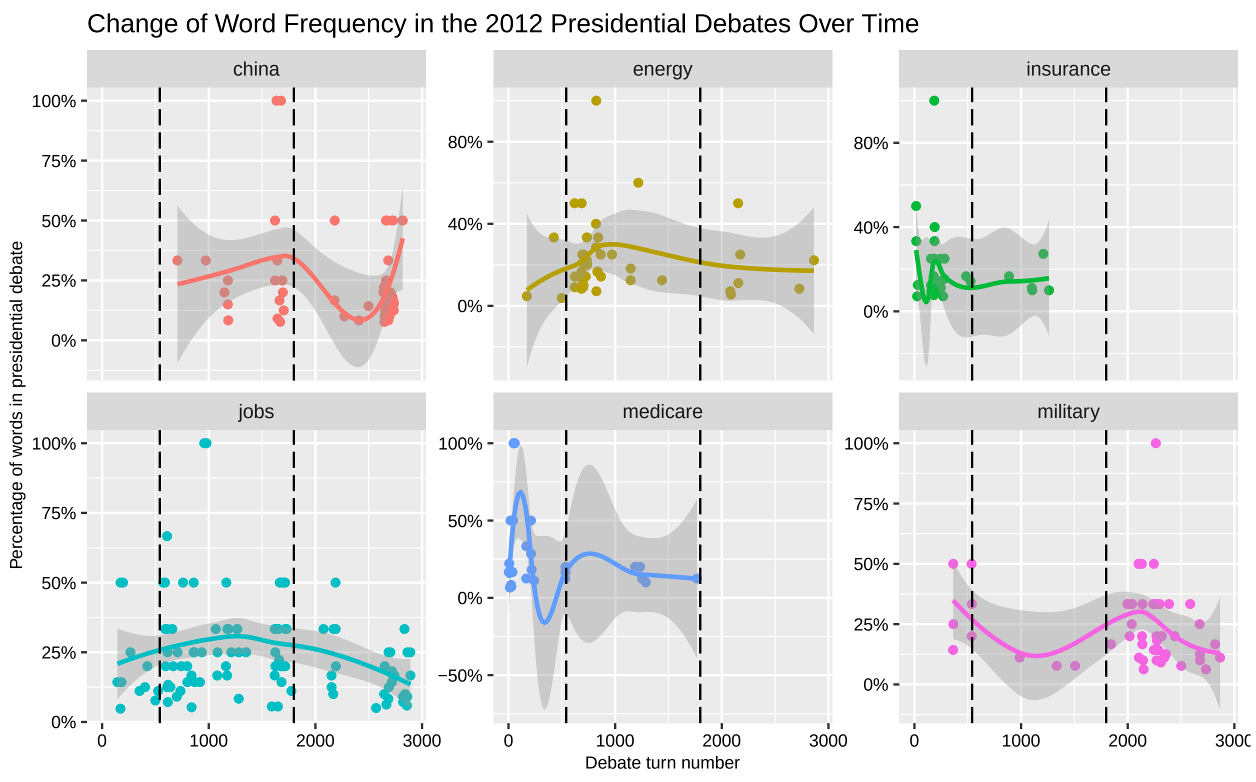
```r
plotDebateWords(c("gather","focus","people"),fict_df)
```



Discussion on 2012 US presidential debate

Let's look at the change in frequency of the following words ("jobs", "military", "medicare", "energy", "insurance", "china") over the three main debates.

```r
plotDebateWords(c("jobs", "military", "medicare", "energy", "insurance", "china"))
```



Just a brief explanation of above graph.

First Debate Based on the above graph, and considering only the six words (jobs, military, medicare, energy, insurance, china) we have selected, we can see that the first debate focused more on health related dialogues. Insurance (health insurance) and medicare (aka Obamacare) occupied the first segment (before the first dash line) of our graph. Dialogues relating to China on the other hand, seems not to be discussed at all. Towards the end of the first debate however, we see brief debate on military issues.

Second Debate Debate on China related issues occurred in the second debate with very high interest towards the end of the second debate (second dash line). It can also be seen that energy issues were debated at the start of the second debate.

Third Debate Military affairs dominated the third debate while health issues were completely avoided. China was also discussed later in the third debate. All though the three debates however, jobs related dialogues were debated.

The tf-idf index

The ***tf-idf*** index computes the frequency of a term adjusted for how rarely it is used. It is calculated as the product of the term frequency (**tf**) and the inverse document frequency (**idf**):

$$tf - idf = tf \times idf.$$

- The term frequency (**tf**) identifies how frequently a word occurs in a document.

- The inverse document frequency (**idf**) identify the important words by decreasing the weight for commonly used words and increasing the weight for words that are not used very much in a collection of documents. It is defined as:

$$idf = \log(\frac{N}{n_t})$$

where N is the total number of documents being assessed and n_t is the number of documents where the term t appears .

To calculate the **tf-idf** for the 2012 US presidential debate, we are simply attempting to find the important words (after adjustment for frequency/rarity of use) mentioned by each of the speakers in the debates dialogues. Put in another way, we are looking for words frequently used by a speaker, but not all the speakers. We use the `bind_tf_idf` function to compute the tf-idf.

Before we do that, we need to have our data in a tidy format. Again, we use the `unnest_tokens` function in the tidytext package, which takes in a dataframe of the debate data and tokenize the dialogue, splitting each sentence in separate words. The two arguments for the `unnest_tokens` function are column names; the output column (word in this case) and the input column (dialogue in this case). We then use dplyr to remove stop words with an anti_join(). stop words are common english words such as "the", "of", "to", etc, which are not useful for our analysis.

Create tidy text data

```
library(tidytext)
library(dplyr)

words <- debateData %>%
  unnest_tokens(word,dialogue) %>%
  anti_join(stop_words) %>%
  count(person,word)
```

This is followed by the `bind_tf_idf` function from the `tidytext` package, which takes in the tidy data containing 3 columns; First column (person) contains the debate speakers, the second column (word) contains the tokens and the last column (counts) is the no of times each speaker used a particular word, that is "n" in this case.

```
words <- words %>%
  bind_tf_idf(word,person,n)
```

Lets look at the idf and tf-idf statistics:

```
words
```

```
## # A tibble: 4,220 x 6
##    person  word           n      tf   idf   tf_idf
##    <chr>   <chr>      <int>   <dbl> <dbl>    <dbl>
##  1 CROWLEY absolutely     2 0.00413 0.693 0.00286
##  2 CROWLEY act            2 0.00413 0.405 0.00168
##  3 CROWLEY add            2 0.00413 0.405 0.00168
##  4 CROWLEY address        2 0.00413 1.79  0.00740
##  5 CROWLEY agreed         1 0.00207 0.693 0.00143
##  6 CROWLEY agreement      1 0.00207 0.693 0.00143
##  7 CROWLEY ahead          2 0.00413 1.10  0.00454
##  8 CROWLEY ak             1 0.00207 0.693 0.00143
##  9 CROWLEY american       1 0.00207 0.182 0.000377
## 10 CROWLEY answer         1 0.00207 0.405 0.000838
## # ... with 4,210 more rows
```

We expect that the inverse document frequency (**idf**) value, and thus the **tf-idf** value, will be very low for common words; that is words commonly mentioned by all the speakers. In contrast, the **tf-idf** value will be higher for words that are unique to a speaker and not commonly used by all the speakers.

Let us extract and sort list of words with high **tf-idf**.

```r
words %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 4,220 x 6
##    person    word              n      tf   idf tf_idf
##    <chr>     <chr>         <int>   <dbl> <dbl>  <dbl>
##  1 QUESTION  department        3 0.0138  1.79  0.0248
##  2 LEHRER    minute            5 0.0213  1.10  0.0234
##  3 SCHIEFFER segment           9 0.0203  1.10  0.0223
##  4 LEHRER    federal           6 0.0255  0.693 0.0177
##  5 LEHRER    minutes          10 0.0426  0.405 0.0173
##  6 QUESTION  chu               2 0.00922 1.79  0.0165
##  7 QUESTION  misperception     2 0.00922 1.79  0.0165
##  8 QUESTION  stated            2 0.00922 1.79  0.0165
##  9 LEHRER    improve           2 0.00851 1.79  0.0152
## 10 LEHRER    views             2 0.00851 1.79  0.0152
## # ... with 4,210 more rows
```

and those with low **tf-idf**

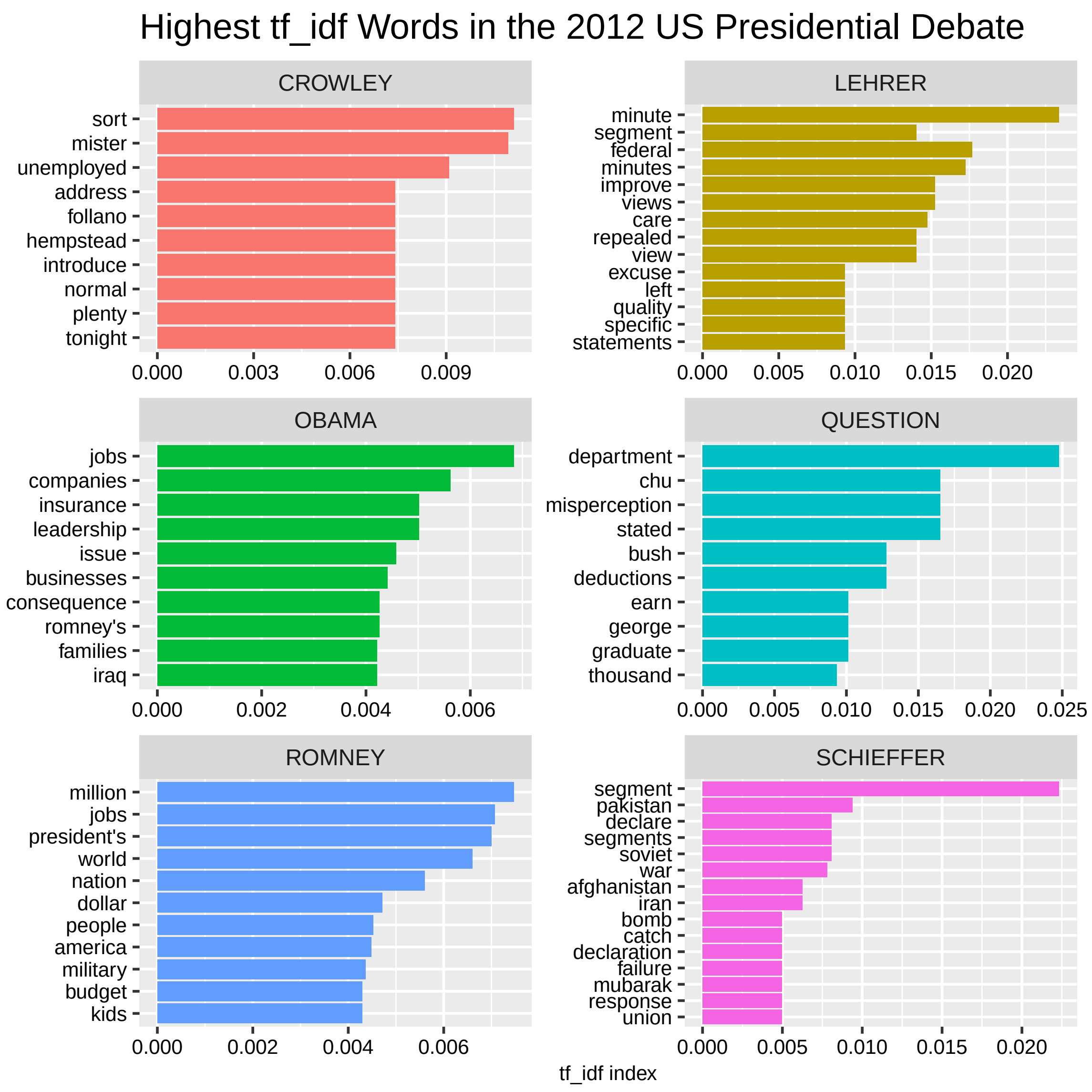
```r
words %>%
  arrange(tf_idf)
```

```
## # A tibble: 4,220 x 6
##    person   word          n      tf   idf tf_idf
##    <chr>    <chr>     <int>   <dbl> <dbl>  <dbl>
##  1 CROWLEY  governor     33 0.0682      0      0
##  2 CROWLEY  president    33 0.0682      0      0
##  3 CROWLEY  question     23 0.0475      0      0
##  4 LEHRER   governor     14 0.0596      0      0
##  5 LEHRER   president     9 0.0383      0      0
##  6 LEHRER   question      2 0.00851     0      0
##  7 OBAMA    governor    122 0.0223      0      0
##  8 OBAMA    president    21 0.00383     0      0
##  9 OBAMA    question      9 0.00164     0      0
## 10 QUESTION governor      4 0.0184      0      0
## # ... with 4,210 more rows
```

Here, the **tf-idf** values are zero for these very common words, which indicates that they are words that are commonly used in the debates by all the speakers. The list is quite realatable. Obama was the incumbent president during the 2012 presidential debate while Romney was the governor of Massachusetts from 2003 to 2007. It is thus expected that during the debate they will be commonly referred to with these titles (i.e. President for Obama and Governor for Romney) by all the speakers, including the candidates themselves. The words with high tf-idf values on the other hand are unique words to the particular speaker. For example, on the list are lot of words from the audience.

We now create a plot of top 10 words with the highest tf-idf index for each speaker.

```r
library(ggplot2)
library(dplyr)
words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(person) %>%
  top_n(10) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = person)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf_idf index") +
  ggtitle("Highest tf_idf Words in the 2012 US Presidential Debate") +
  facet_wrap(~person, ncol = 2, scales = "free") +
  coord_flip() +
  theme(axis.text = element_text(size = 8, color = "black"),
        axis.title = element_text(size = 8, color = "black"),
        title = element_text(size = 12))
```



Looking at the candidates, there does not seem to be a major difference between them. “jobs” related dialogues seem to be one of their most important campaign/debate strategy. It could be that Obama was emphasizing on how much jobs he created during his first term, while Romney also advertising his jobs creation records when he was a governor. The “million” with high tf-idx index under Romney refer to the millions of jobs he created as governor. So basically “jobs” was the center point of the candidate’s message. We can also see that Schieffer questions were focused on global politics and security concerns in Asia, with words like pakistan, afghanistan, war and Iran coming up top.

Zipf’s law

We now test our debate data for Zipf’s law. Again we need to have our debate data in a tidy data structure, but this time, we are not removing the stopwords as they are important to illustrate Zipf’s law. As usual, we use the unnest_tokens() to tokenize the dialogue and create a new column (word) containing the tokenized words, and count the number of times each word is used by the speakers.

```
words <- debateData %>%
  unnest_tokens(word, dialogue) %>%
  count(person,word, sort = TRUE) %>%
  ungroup()
```

This is followed by group_by() and summarize() to calculate the total words spoken by each speakers during the debate (i.e. person in our case)

```
total_words <- words %>%
  group_by(person) %>%
  summarize(total = sum(n))
```

Using left_join() in the dplyr package, we join out tidy text data (words) with the total_words data. With this, for every word, we have the 'n' number of times the particular word is used by a speaker and the 'total' words used by that speaker.

```
words <- left_join(words, total_words)
```

Then, the term frequency (or tf) can be easily calculated as $(\frac{n}{total})$. With our data already sorted in decreasing order of tf, the rank can be obtained as the row number.

```
freq_by_rank <- words %>%
  group_by(person) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total)
```

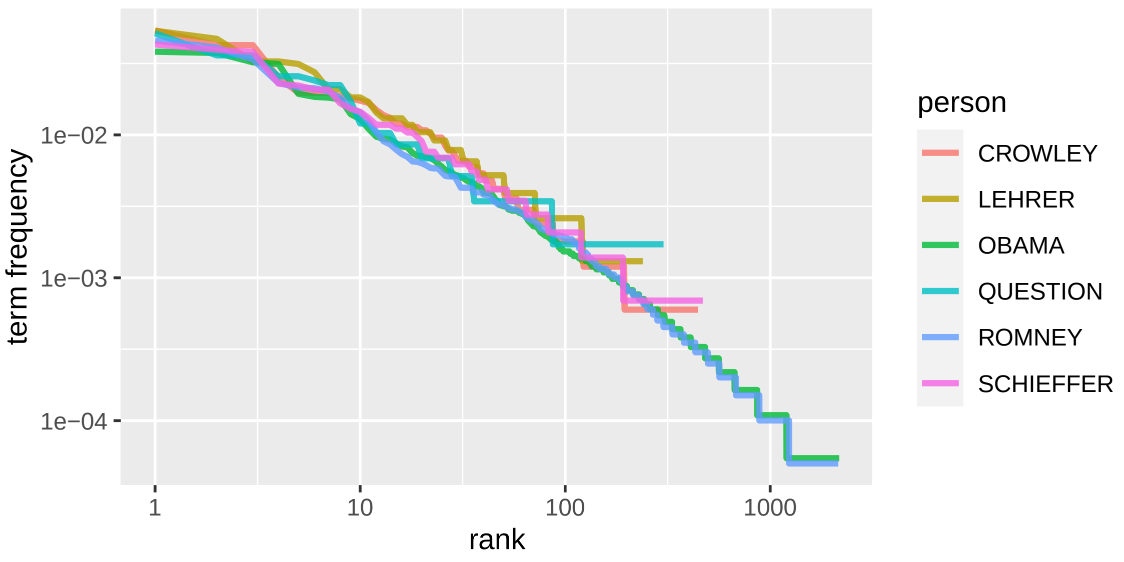
The rank column gives the rank of each word within the frequency table.

```
freq_by_rank
```

```
## # A tibble: 5,776 x 6
## # Groups:   person [6]
##    person word      n total  rank `term frequency`
##    <chr>  <chr> <int> <int> <int>            <dbl>
##  1 ROMNEY the     913 19924     1           0.0458
##  2 ROMNEY to      812 19924     2           0.0408
##  3 OBAMA  the     700 18319     1           0.0382
##  4 OBAMA  to      686 18319     2           0.0374
##  5 ROMNEY and     671 19924     3           0.0337
##  6 OBAMA  and     590 18319     3           0.0322
##  7 OBAMA  that    574 18319     4           0.0313
##  8 ROMNEY that    460 19924     4           0.0231
##  9 ROMNEY a       427 19924     5           0.0214
## 10 ROMNEY of      422 19924     6           0.0212
## # ... with 5,766 more rows
```

A quick glance of the above table shows that very frequently used english words, such as "the", "to", "that", etc have very low rank as predicted by Zipf's law. As stated earlier, Zipf's law is best observed by plotting the data on a log-log graph, with the axes being log (rank order) and log (frequency).

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = person)) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = TRUE) +
  scale_x_log10() +
  scale_y_log10()
```



We can see that the plot is approximately linear on the log-log graph, and the relationship between rank and the term frequency have a negative slope. Although, there are slight deviations at the high rank and low rank range. This is however not unusual and common in most representation of Zipf's law, e.g. (Adamic.2000).

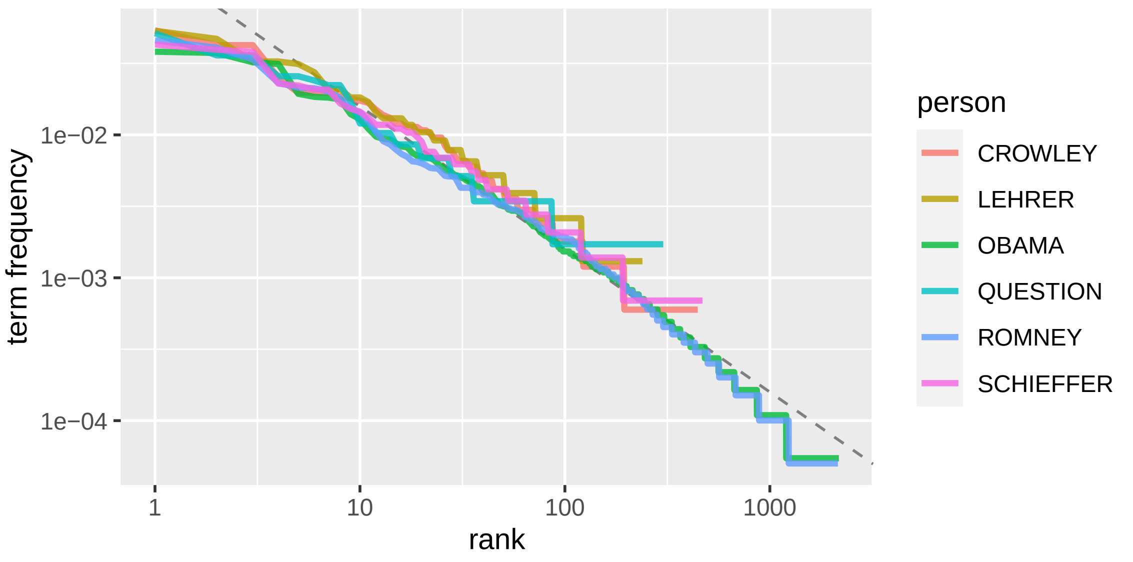
Lets now fit a linear regression model. We however need to adjust for the different lengths of dialogues in the debate data, otherwise, long dialogues would weighted higher than shorter ones. We are therefore going to subset the data to rank smaller than 500 and greater than 10. We use the function lm() to fit the linear models.

```
rank_subset <- freq_by_rank %>%
filter(rank < 500,rank > 10)
lm(log10(`term frequency`) ~ log10(rank), data = rank_subset)
```

```
##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##     -0.7650      -0.9793
```

As explained earlier, to test for Zipf's law, we hope to get a slope close to -1 on the log-log graph, which is what we have here. We can now use the function geom_abline() to add the regression line to our graph using values of intercept and slope from our linear regression model.

```
freq_by_rank %>%
ggplot(aes(rank, `term frequency`, color = person)) +
geom_abline(intercept = -0.8, slope = -1.0, color = "gray50", linetype = 2) +
geom_line(size = 1.1, alpha = 0.8, show.legend = TRUE) +
scale_x_log10() +
scale_y_log10()
```



It can be seen clearly that our data fit the classic version of Zipf's law - the log(freq) vs log(rank) graph is approximately fitted by a straight line of slope -1. One other analysis we can infer is the similarity of words used by Obama and Romney, especially within the middle section of the rank range, as we can see that the trend of Obama and Romney lines mimic each other, which is distinct from the other speakers (person)

References

1.Wickham, H., 2014. Tidy data. Journal of Statistical Software, 59(10), pp.1-23.
2.Clauset, A., Shalizi, C.R. and Newman, M.E., 2009. Power-law distributions in empirical data. SIAM review, 51(4), pp.661-703.
3.Li, W.; Miramontes, P.; Cocho, G. Fitting Ranked Linguistic Data with Two-Parameter Functions. Entropy 2010, 12, 1743-1764.
4.Goldstein, M.L., Morris, S.A. and Yen, G.G., 2004. Problems with fitting to the power-law distribution. The European Physical Journal B-Condensed Matter and Complex Systems, 41(2), pp.255-258.
5.White, E.P., Enquist, B.J. and Green, J.L., 2008. On estimating the exponent of power-law frequency distributions. Ecology, 89(4), pp.905-912.
6.Adamic, L.A., 2000. Zipf, power-laws, and pareto-a ranking tutorial. Xerox Palo Alto Research Center, Palo Alto, CA, http://ginger. hpl. hp. com/shl/papers/ranking/ranking.