DS5110 HW 3 - Due Apr. 3

Kylie Ariel Bemis

3/20/2017

Instructions

Create a directory with the following structure:

- `hw3-your-name/hw3-your-name.Rmd`
- `hw3-your-name/hw3-your-name.pdf`
- `hw3-your-name/mypackage.tar.gz`

where `hw3-your-name.Rmd` is an R Markdown file that compiles to create `hw3-your-name.pdf`, and `mypackage.tar.gz` is your solution for Problem 10.

Do not include data in the directory. Compress the directory as `.zip`.

Your solution should include all of the code necessary to answer the problems. All of your code should run (assuming the data is available). All plots should be generated using `ggplot2`. Missing values and overplotting should be handled appropriately. Axes should be labeled clearly and accurately.

To submit your solution, create a new private post of type "Note" on Piazza, select "Individual Student(s) / Instructor(s)" and type "Instructors", select the folder "hw3", go to Insert->Insert file in the Rich Text Editor, upload your `.zip` homework solution. Title your note "[hw3 solutions] - your name" and post the private note to Piazza. **Be sure to post it only to instructors**

Part A

Problems 1–3 use the `BostonHousing` data from the `mlbench` package. Install the `mlbench` package and use `data(BostonHousing)` to load the dataset.

Problem 1

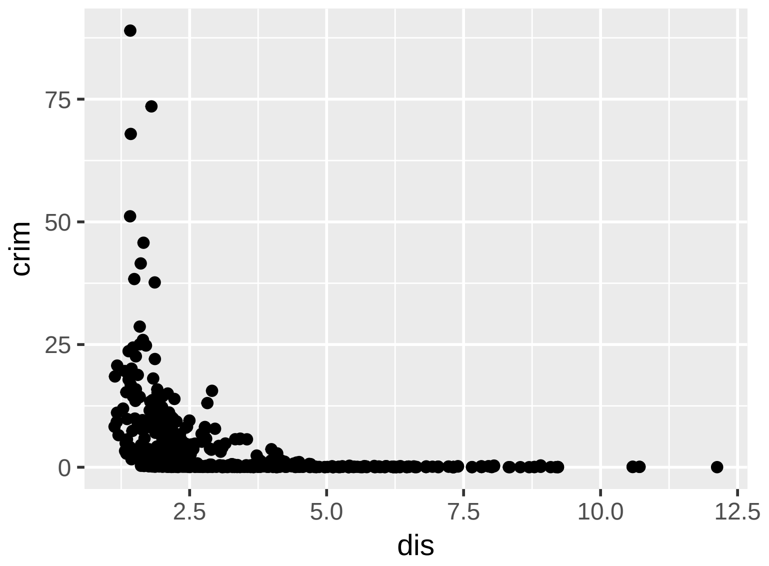
Fit a model that predicts per capita crime rate by town (‘crim‘) using only *one* predictor variable. Use plots to justify your choice of predictor variable and the appropriateness of any transformations you use. Print the values of the fitted model parameters.

```r
# Answers will vary
library(tidyverse)
library(modelr)
library(mlbench)
```

```r
data(BostonHousing)
```

First we plot `crim` versus `dis`.

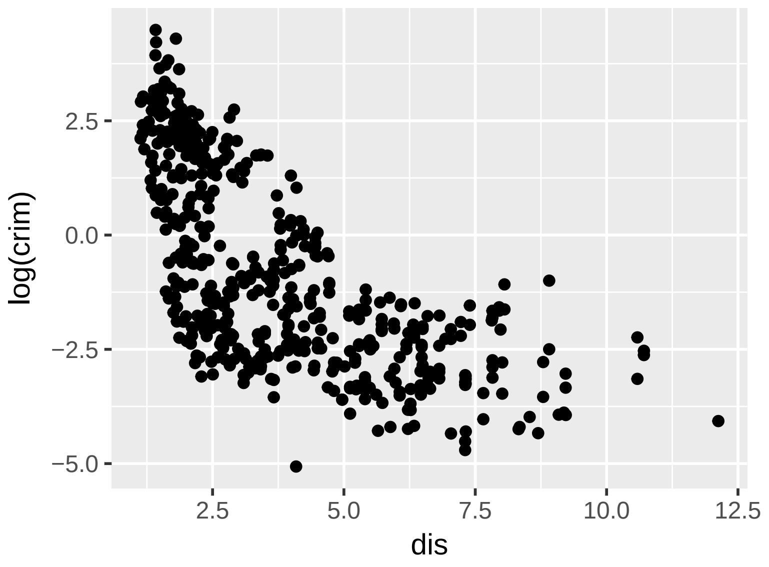
```r
ggplot(BostonHousing, aes(x=dis, y=crim)) + geom_point()
```



From the scatterplot, there appears to be a negative association between `dis` and `crim`, but it's not linear.

Next, we try plotting `log(crim)` versus `dis`.

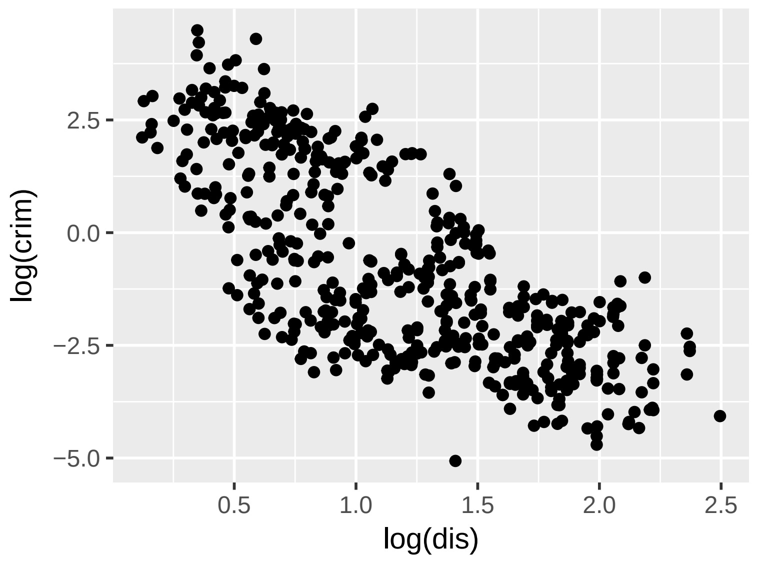
```
ggplot(BostonHousing, aes(x=dis, y=log(crim))) + geom_point()
```



Log transforming `crim` improves the relationship, but the relationship is still not quite linear.

We try plotting `log(crim)` versus `log(dis)` now.

```
ggplot(BostonHousing, aes(x=log(dis), y=log(crim))) + geom_point()
```



Log transforming `dis` as well improves the relationship, making it much more linear. We will include `dis` as the predictor variable in our model.

```
fit1 <- lm(log(crim) ~ log(dis), data=BostonHousing)
summary(fit1)
```

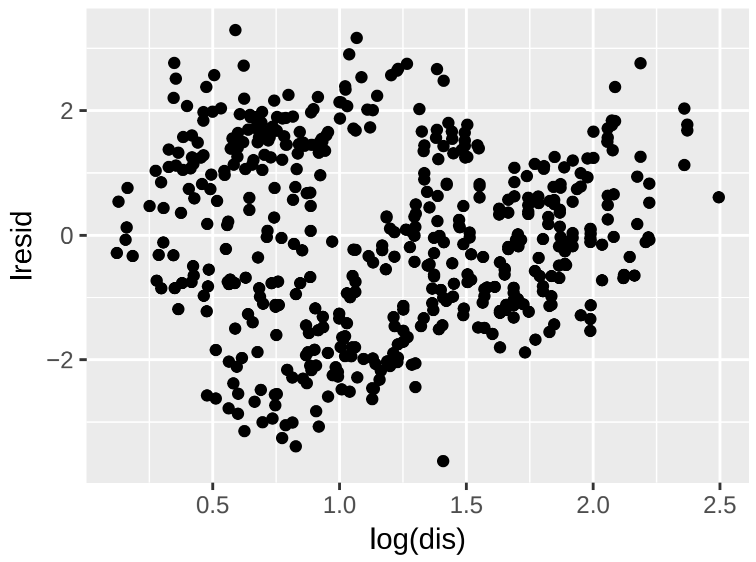
```
##
## Call:
## lm(formula = log(crim) ~ log(dis), data = BostonHousing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6262 -1.1145 -0.0187  1.2476  3.2931
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.7611     0.1556   17.74   <2e-16 ***
## log(dis)     -2.9810     0.1193  -24.99   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.446 on 504 degrees of freedom
## Multiple R-squared:  0.5534, Adjusted R-squared:  0.5525
## F-statistic: 624.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

Problem 2

Plot the residuals of the fitted model from Problem 1 against the predictor variable already in the model *and* against other potential predictor variables in the dataset. Comment on what you observe in each residual plot. (You do not have to include plots for predictor variables not in the model if you observe no systematic patterns in them.)

First we investigate the log residuals versus `log(dis)`, which is the predictor variable in the model.

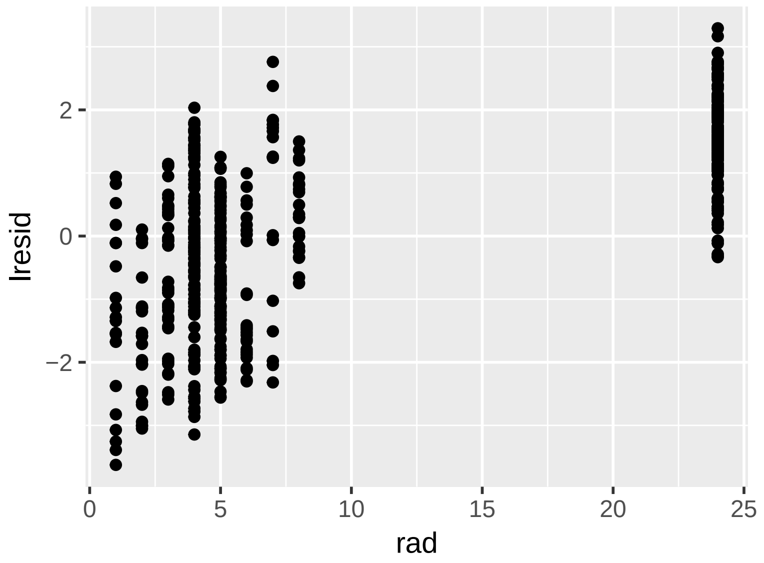
```
BostonHousing %>%
  add_residuals(fit1, "lresid") %>%
  ggplot(aes(x=log(dis), y=lresid)) +
  geom_point()
```



In the residual plot for `log(dis)`, we mostly see simple random scatter and no systematic patterns, indicating no violation of model assumptions.

Now we investigate the log residuals versus `rad`, which is a variable not currently in the model.

```
BostonHousing %>%
  add_residuals(fit1, "lresid") %>%
  ggplot(aes(x=rad, y=lresid)) +
  geom_point()
```



In the residual plot for `rad`, we see a positive linear relationship between the log residuals and `rad`, indicating that there is a relationship between `rad` and `log(crim)`, so we should add `rad` as a predictor in the model.

Problem 3

Fit a new model for predicting highway mileage, adding or removing variables based on the residual plots from Problem 2.

Based on the residual plot for `rad`, we will add `rad` to the model.

```r
fit2 <- lm(log(crim) ~ log(dis) + rad, data=BostonHousing)
summary(fit2)
```

```
##
## Call:
## lm(formula = log(crim) ~ log(dis) + rad, data = BostonHousing)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.59042 -0.65587 -0.04422  0.57162  2.34690
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.445314   0.146923  -3.031  0.00256 **
## log(dis)    -1.552176   0.088618 -17.515  < 2e-16 ***
## rad          0.158011   0.005491  28.775  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.89 on 503 degrees of freedom
## Multiple R-squared:  0.8312, Adjusted R-squared:  0.8306
## F-statistic:  1239 on 2 and 503 DF,  p-value: < 2.2e-16
```

Part B

Problem 4

Write a function that performs cross-validation for a linear model (fit using ‘lm‘) and returns the average root-mean-square-error across all folds. The function should take as arguments (1) a formula used to fit the model, (2) a dataset, and (3) the number of folds to use for cross-validation. The function should partition the dataset, fit a model on each training partition, make predictions on each test partition, and return the average root-mean-square-error.

```r
cvlm <- function(formula, data, nfold) {
  cvdata <- crossv_kfold(data, nfold)
  cvdata <- cvdata %>%
    mutate(fit = map(train, ~ lm(formula, data = .))) %>%
    mutate(rmse = map2_dbl(fit, test, rmse))
  c(cv_rmse=mean(cvdata$rmse))
}
```

Problem 5

Use your function from Problem 4 to compare the models you used from Part A with 5-fold cross-validation. Report the cross-validated root-mean-square-error for the models from Problems 1 and 3. Which model was more predictive?

```r
set.seed(1)
cvlm(log(crim) ~ log(dis), BostonHousing, 5)
```

```
##  cv_rmse
## 1.441691
```

```r
cvlm(log(crim) ~ log(dis) + rad, BostonHousing, 5)
```

```
##   cv_rmse
## 0.8882928
```

The second model with `rad` in addition to `log(dis)` was more predictive. (It has a smaller root-mean-squared-error.)

Part C

Problems 6–8 use the text of 56 major speeches by Donald Trump from June 2015 through November 2016. Download the data from https://github.com/PedramNavid/trump_speeches. (Use either the "full_speech.txt" file or the "speech_##.txt" files but not both, as the former contains all of the text of the latter. If you use the "speech_##.txt" files, you should skip the first line of each file.) Use the `read_lines()` function from the `readr` package to import the data into R.

Problem 6

Import the text from all 56 Donald Trump speeches into R and tokenize the data into a tidy text data frame, using *words* as tokens. After removing stop words and the word "applause", plot the top 15 most common words used in Trump's speeches.

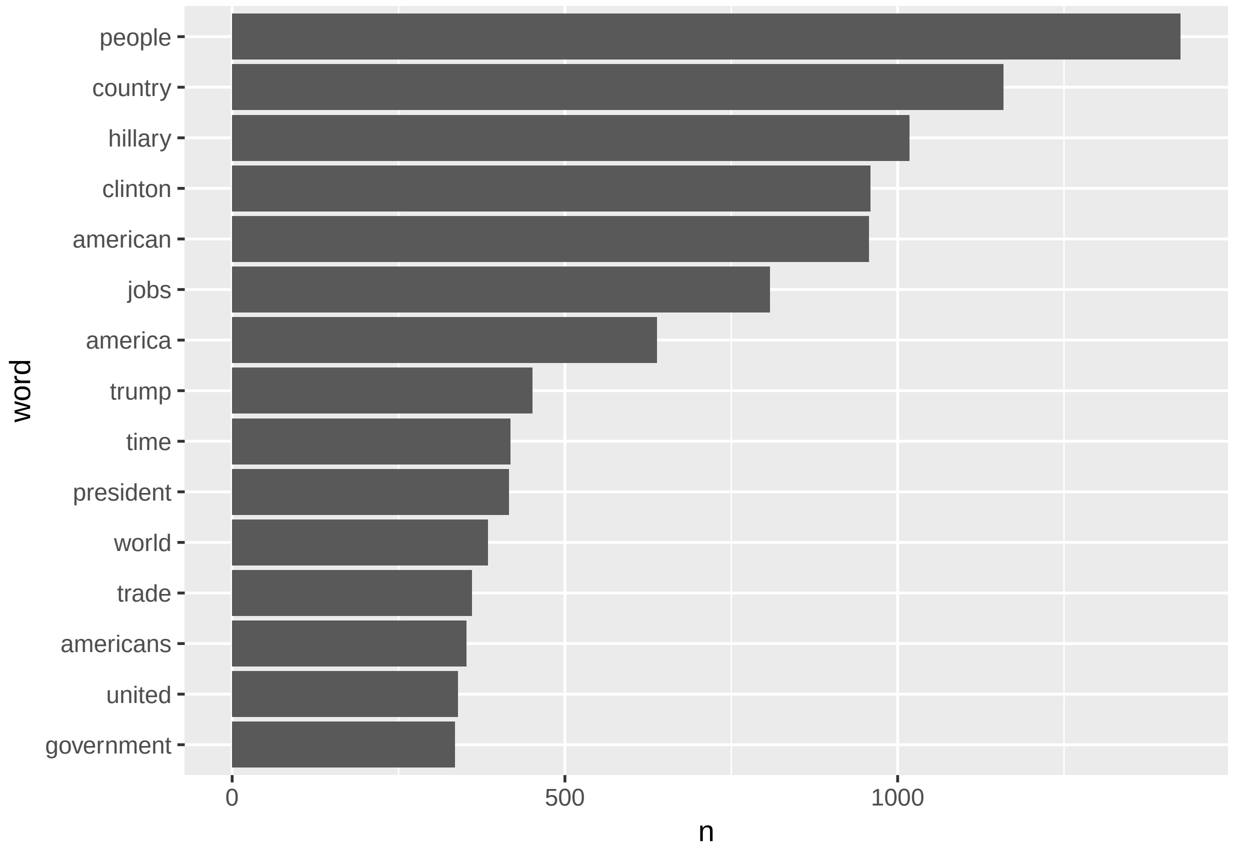
```r
require(tidyverse)
require(tidytext)
require(stringr)
```

```r
read_corpus <- function(dir = ".", ...) {
  files <- normalizePath(str_c(dir, "/", list.files(dir)))
  corpus <- map(files, function(file) {
    text <- read_lines(file, ...)
    tibble(doc=basename(file), text=text)
  })
  bind_rows(corpus)
}

trump <- read_corpus("../../../data/trump_speeches-master/data", skip=1)
```

```r
trump_words <- trump %>%
  unnest_tokens(word, text)

trump_words %>%
  anti_join(stop_words, by="word") %>%
  filter(word != "applause") %>%
  count(word, sort=TRUE) %>%
  mutate(word = reorder(word, n)) %>%
  top_n(15) %>%
  ggplot(aes(x=word, y=n)) +
  geom_col() +
  coord_flip()
```

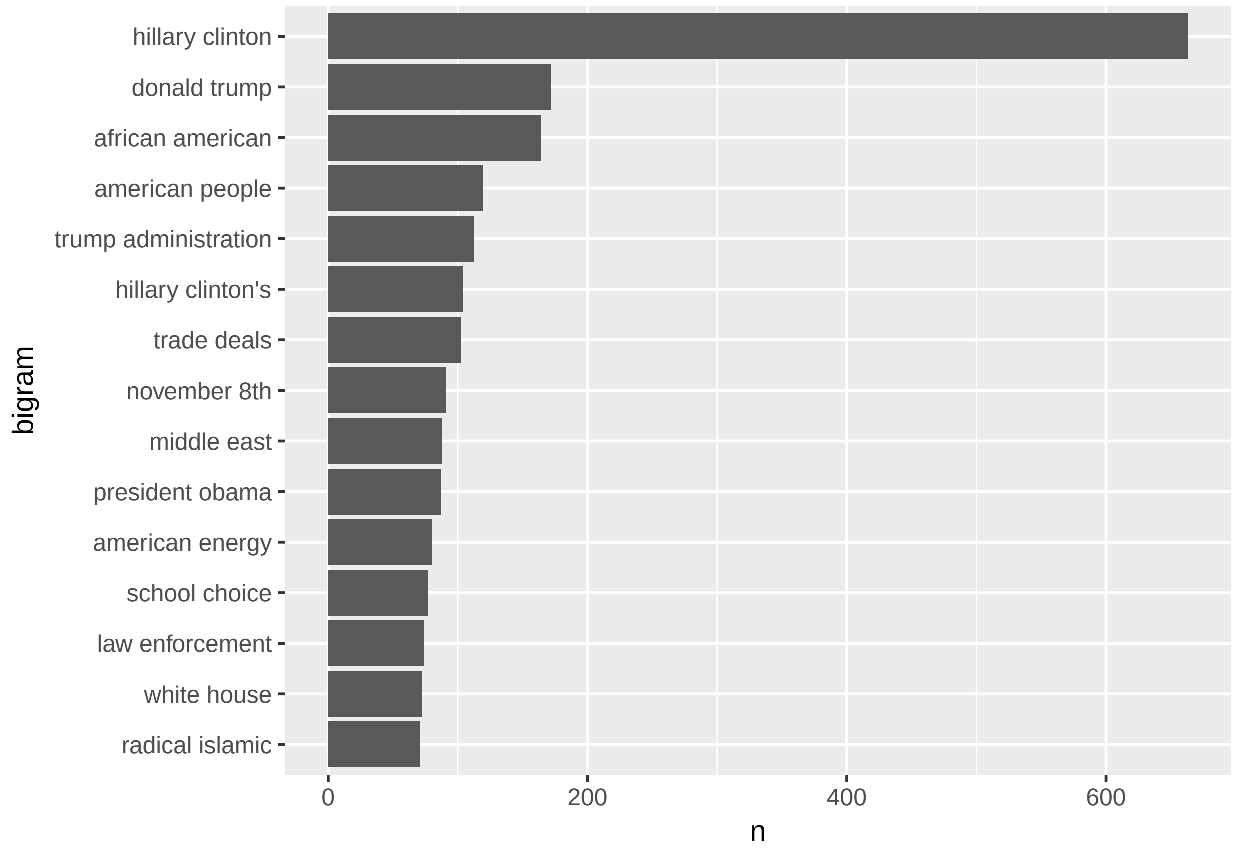


Problem 7

Re-tokenize the text of all 56 Donald Trump Speeches into a new tidy text data frame, using *bigrams* as tokens. Remove each bigram where either word is a stop word or the word "applause". Then plot the top 15 most common bigrams in Trump's speeches.

```
trump_bigrams <- trump %>%
  unnest_tokens(bigram, text, token="ngrams", n=2) %>%
  separate(bigram, c("word1", "word2"), sep = " ")

trump_bigrams %>%
  filter(!word1 %in% c(stop_words$word, "applause")) %>%
  filter(!word2 %in% c(stop_words$word, "applause")) %>%
  unite(bigram, word1, word2, sep = " ") %>%
  count(bigram, sort=TRUE) %>%
  mutate(bigram = reorder(bigram, n)) %>%
  top_n(15) %>%
  ggplot(aes(x=bigram, y=n)) +
  geom_col() +
  coord_flip()
```

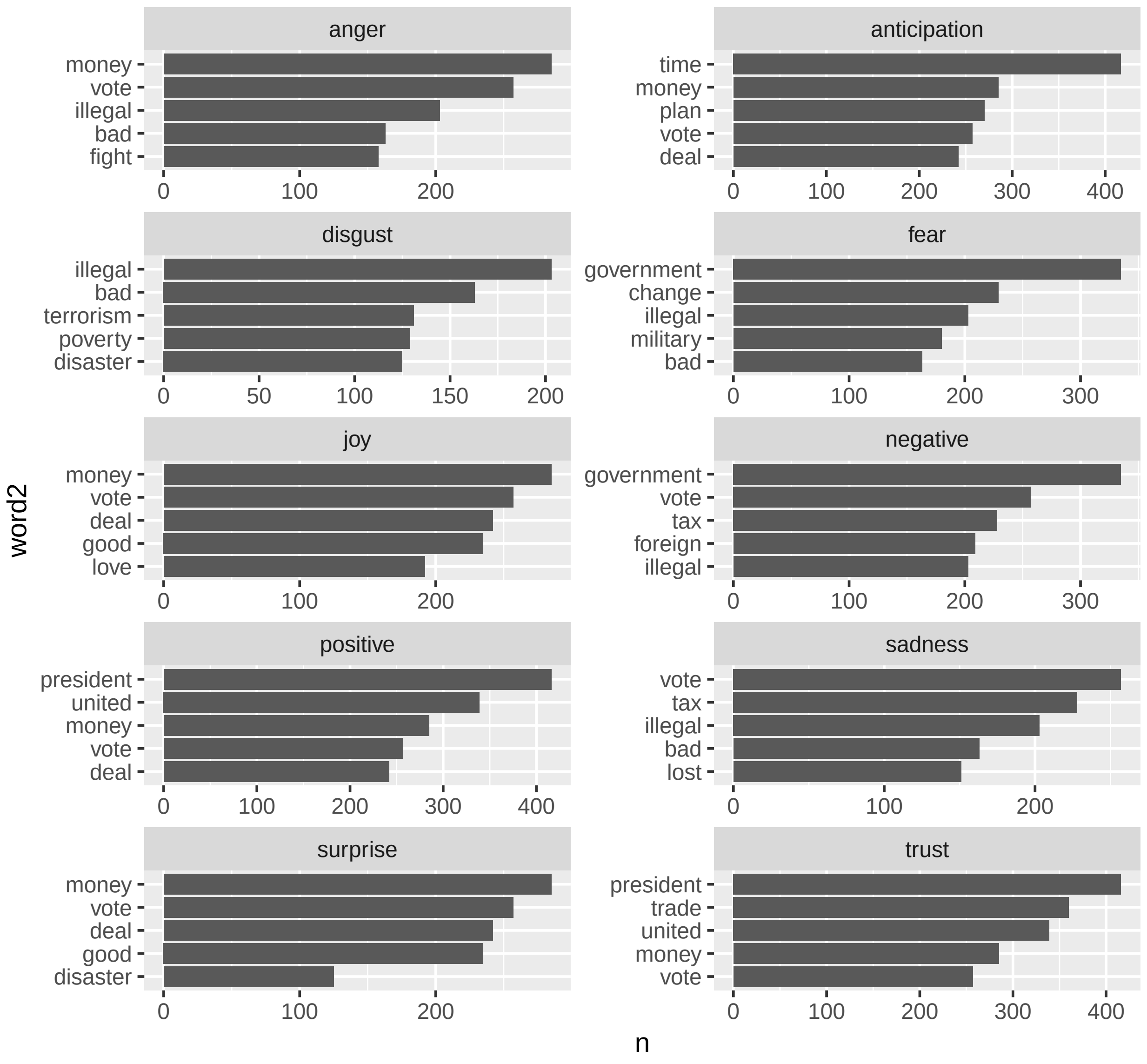


Problem 8

We would like to do a sentiment analysis of Donald Trump's speeches. In order to make sure sentiments are assigned to appropriate contexts, first tokenize the speeches into bigrams, and filter out all bigrams where the first word is any of the words "not", "no", or "never".

Now consider only the second word of each bigram. After filtering out the words "applause" and "trump", create a plot of the most common words in Trump's speeches that are associated with each of the 10 sentiments in the "nrc" lexicon.

```
trump_bigrams %>%
  filter(!word1 %in% c("not", "no", "never")) %>%
  filter(word2 != "applause", word2 != "trump") %>%
  inner_join(get_sentiments("nrc"), by=c("word2"="word")) %>%
  count(word2, sentiment, sort=TRUE) %>%
  mutate(word2 = reorder(word2, n)) %>%
  group_by(sentiment) %>%
  top_n(5) %>%
  ggplot(aes(x=word2, y=n)) +
  geom_col(show.legend=FALSE) +
  facet_wrap(~sentiment, ncol=2, scales="free") +
coord_flip()
```



Part C

Problem 9

Create an S3 class called `tidy_corpus` which inherits from a `tibble`. A `tidy_corpus` is a tidy text data frame that always has at least one column called `token`, which gives the tokens, and an attribute called `token_type` which gives the type of token. A `tidy_corpus` object may also have an additional attribute called `n` when `token_type` is "ngrams".

Create constructor function that creates `tidy_corpus` objects called `tidy_corpus(src, token, ...)` which takes three arguments:

- `src` is either a character vector or a directory of text files

- token gives the kind of tokenization to be performed
- ... are additional arguments passed to unnest_tokens()

Write a plot() method for the tidy_corpus class that plots the top n most common tokens in the corpus where n is a user parameter defaulting to 10. Include an optional parameter for removing stop words. The removal of stop words should support at least words and bigrams.

Test your class and methods on the data in Part C.

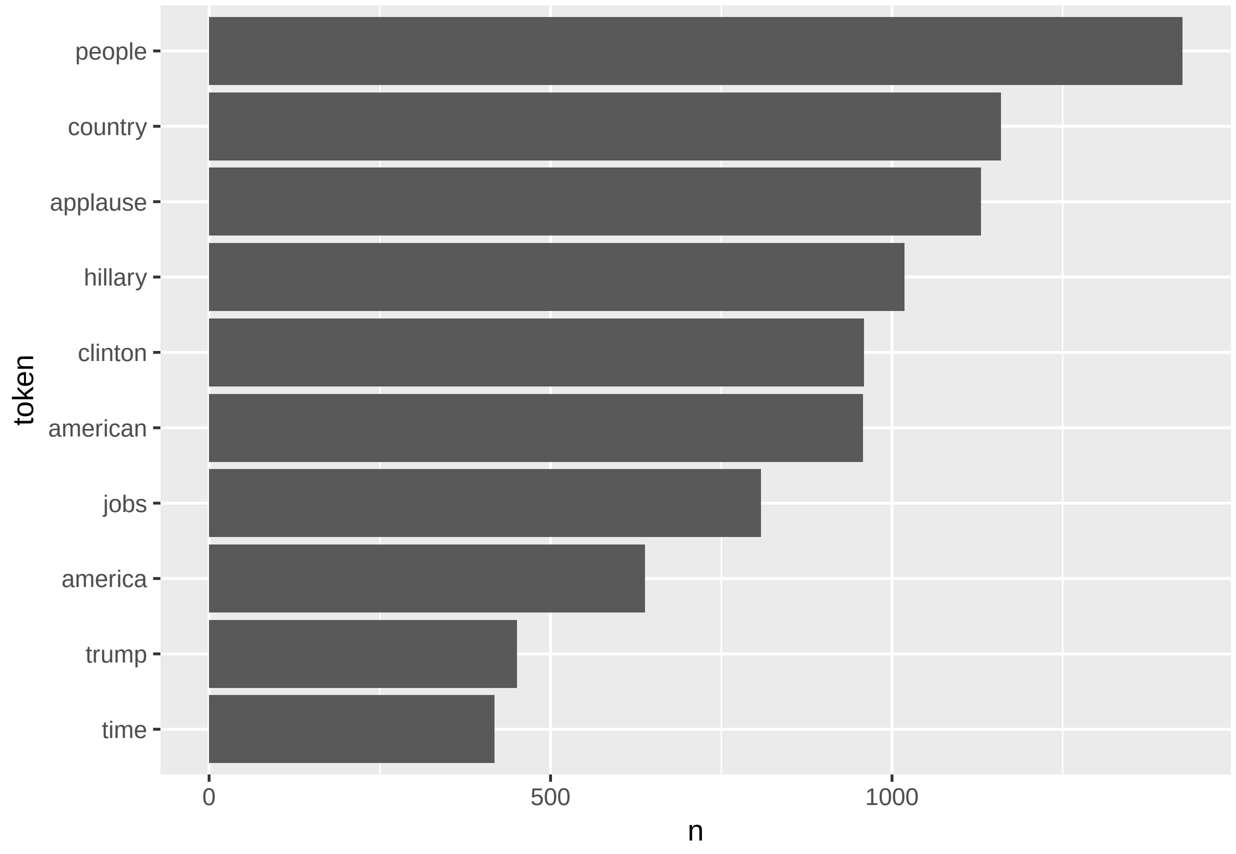
```r
tidy_corpus <- function(src, token = "words", ...) {
  files <- list.files(normalizePath(src, mustWork=FALSE))
  if ( length(files) > 0 ) {
    files <- normalizePath(str_c(src, "/", files))
    corpus <- map(files, function(file) {
      text <- read_lines(file, ...)
      tibble(doc=basename(file), text=text)
    })
    corpus <- bind_rows(corpus)
  } else {
    corpus <- tibble(text=src)
  }
  corpus <- unnest_tokens(corpus, "token", "text",
                          token=token, ...)
  class(corpus) <- c("tidy_corpus", class(corpus))
  attr(corpus, "token_type") <- token
  if ( token == "ngrams" ) {
    dots <- list(...)
    if ( "n" %in% names(dots) ) {
      attr(corpus, "n") <- list(...)$n
    } else {
      attr(corpus, "n") <- formals(tokenizers::tokenize_ngrams)$n
    }
  }
  corpus
}
```

```r
rm_stop_words <- function(x) {
  if ( attr(x, "token_type") == "words" ) {
    x <- anti_join(x, tidytext::stop_words,
                   by=c("token"="word"))
  } else if ( attr(x, "token_type") == "ngrams" ) {
    if ( attr(x, "n") != 2 )
      stop("'token_type' = 'ngrams' with n != 2 not supported")
    x <- x %>%
      separate(token, into=c("word1", "word2"), sep=" ") %>%
      filter(!word1 %in% tidytext::stop_words$word) %>%
      filter(!word2 %in% tidytext::stop_words$word) %>%
      unite("token", "word1", "word2", sep=" ")
  } else {
    stop("unsupported 'token_type'")
  }
  x
}
```

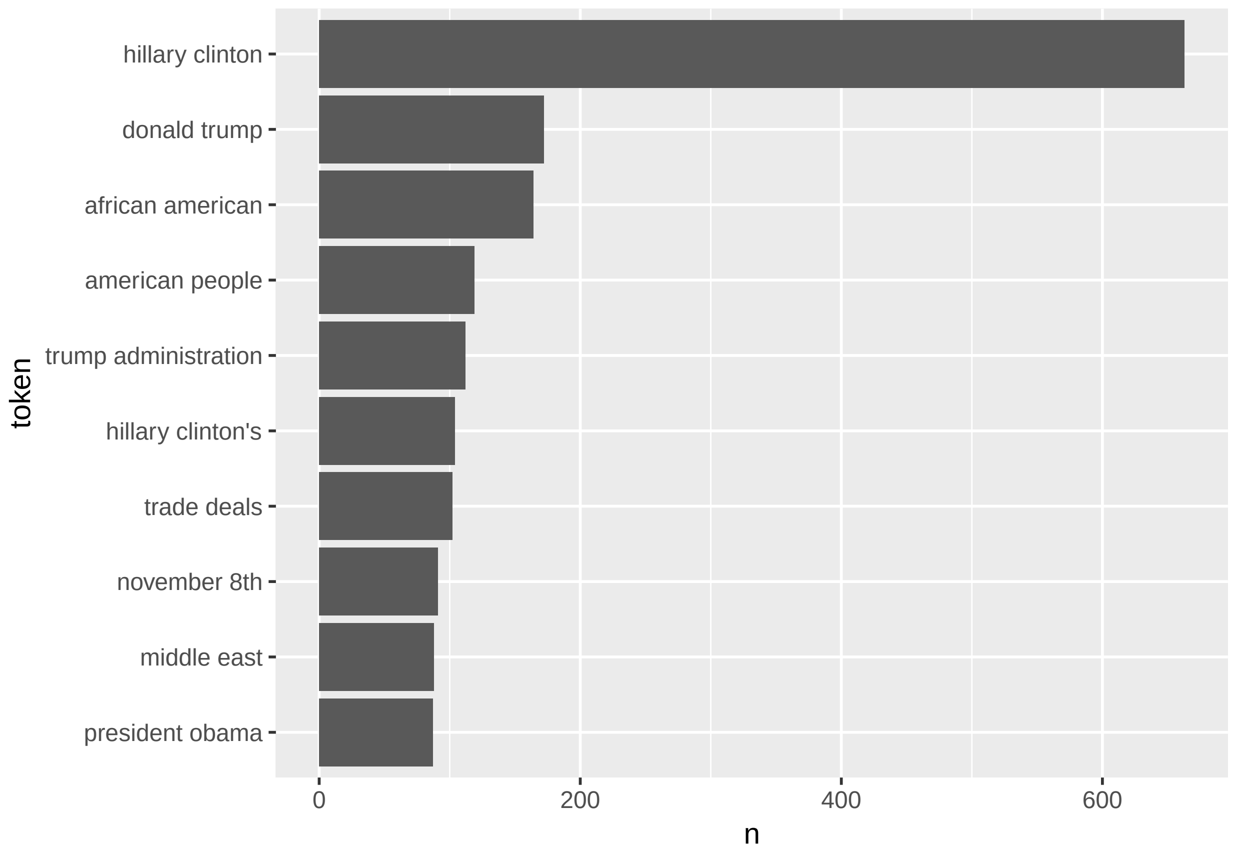
```r
print.tidy_corpus <- function(x, ...) {
  cat("# Tidy corpus: token =", attr(x, "token_type"), "\n")
  NextMethod(x, ...)
}
```

```r
plot.tidy_corpus <- function(x, n = 10, rm_stop = TRUE, ...) {
  if ( rm_stop )
    x <- rm_stop_words(x)
  x %>%
    count(token, sort=TRUE) %>%
    mutate(token = reorder(token, n)) %>%
    top_n(n) %>%
    ggplot(aes(x=token, y=n)) +
    geom_col() +
    coord_flip()
}
```

```r
trump_tdc1 <- tidy_corpus(trump$text)
plot(trump_tdc1)
```



```r
trump_tdc2 <- tidy_corpus(trump$text, token = "ngrams", n = 2)
plot(trump_tdc2)
```



Problem 10

Create an R package for the class and methods you created in Problem 9. For full credit, it should pass `R CMD check` without errors. (Warnings are okay. You do not need to include documentation.) Build the package and include the `.tar.gz` compressed file in your homework directory.

A simple package can be created by running the following code:

```
package.skeleton(name="tidycorpus",
                 list=c(
                   "tidy_corpus",
                   "rm_stop_words",
                   "print.tidy_corpus",
                   "plot.tidy_corpus"))
```

You should add a "Depends: tidyverse, tidytext" field in the DESCRIPTION file to ensure that they will be loaded and attached.

Attempting to install as-is will result in errors due to the empty `.Rd` documentation templates. However, the package can be made installable by simply deleting all of the `.Rd` documentation files.

This will result in many warnings and notes, but the package will install and pass `R CMD check` without errors.

```
library(devtools)
install("tidycorpus")
library(tidycorpus)
```