VIP Preliminary Content Analysis and Topic Modeling

Katie Murenbeeld

2023-01-31

VIP Fall 2022/Spring 2023

1. Content Analysis and Topic Modeling of Wildlife Conservation Articles

Content analysis is a systemic approach to infer the meaning or focus of a text, or collection of texts, based on data derived from the text(s) (Stemler, 2001). Commonly during content analysis one looks at the word frequency within texts to determine the focus of the text or collection of texts. Additionally, one can categorize or qualitatively code the text based on the words, groups of words, or the over all sentiment within the text. Content analysis is a type of data reduction because a larger text is reduced to content a few content categories.

Topic modeling is a technique in which an algorithm analyzes words in a collection of texts or documents and then determines the themes, or topics, within the collection of the texts, which topic best represents each text, and how those topic can change through time (Blei, 2012). In general, topic modeling is an unsupervised, machine learning approach requiring no prior coding or categorization of the articles needs to occur (Blei, 2012).

In the Fall 2022 VIP, students collected and coded 50 news articles related to bears or wolves. From the initial reading and analysis, 15 codes were created ranging in theme from wildlife conflicts with grazing to wildlife issues with people, wildlife, management, and so on. Each article was classified as one or several of the 15 codes and a valence (or positive or negative sentiment) was decided for each code in the article. Here I used R and R packages to complete a preliminary (and very simple) content analysis and topic modeling of these approximately 50 newspaper articles. I referenced this book for the majority of the preliminary work presented below https://www.tidytextmining.com/.

2. Creating the Corpus (and some preprocessing)

A corpus is any collection of texts, where texts can be the written word, spoken word, or music. A corpus could be all of the novels written by Jane Austen, a collection of geology text books, or the collection of new articles about the British Royal Family. In this case the corpus is the collection of roughly 50 articles that were reviewed in the VIP class in the fall of 2022.

The act of actually compiling the corpus can be complicated. Here I downloaded pdf files of the articles collected from the infoweb Newsbank. When downloading the pdfs, the file name is generated based on the article title, the newspaper it was published in, the date published, and the page. There are other options for collecting or compiling news articles from online sources such as "scrapping" the text from the html source code for online articles. (I found this website to be very helpful while building the corpus for this preliminary work: https://data.library.virginia.edu/reading-pdf-files-into-r-for-text-mining/.)

Through a bit of trial and error I found that the code below created a corpus with some extra lines of text. These extra lines contained information about the article title, the newspaper, date, and page, all of which is also in the file name. Because of this I decided to removed the first two lines of text and the line of text with the copyright information from the files before processing into a corpus.

```r
# Load in the files as a list of file names.
# In this case all of the files are pdfs and they all are in the working directory.
files <- list.files(pattern = "pdf$")

corp <- Corpus(URISource(files), # First argument is what we want to use to create
               # the corpus, the vector of pdf files.
               # URI (uniform resource identifier) source means that the vector of
               # file names identifies our resources
               readerControl = list(reader = readPDF)) # Second argument requires a
               # list of control parameters, one is reader, to read in PDF files.
```

In code block below, the files are loaded the same as before, but we will remove the first two lines of text and the copyright information before building the corpus.

```r
# Load in the files as a list of file names.  In this case
# all of the files are pdfs and they all are in the working
# directory.
files <- list.files(pattern = "pdf$")

# Use lapply to to extract the text from all of the pdf
# files in the list
vip <- lapply(files, pdf_text)

# Remove the first two lines of text which contains the
# title, newspaper, and page number Remove the line of text
# which has the copyright info
for (x in 1:length(vip)) {
    # print(x)
    vip[[x]] <- str_replace(vip[[x]], regex("[a-zA-Z0-9_].*[\n]"),
        "")
    vip[[x]] <- str_replace(vip[[x]], regex("[a-zA-Z0-9_].*[\n]"),
        "")
    vip[[x]] <- str_replace(vip[[x]], regex("Copyright.*[\n]"),
        "")
}

# Next, build the corpus using the Corpus() function from
# the tm package
vip_corp <- Corpus(VectorSource(vip))
```

We can now create different data sets from the corpus of articles.

3. Creating a Document Term Matrix and a Tidy Data Frame

There are several useful data formats for content analysis and topic modeling. These include, but are not limited to, a document term matrix (DTM) and a tidy data frame.

In a DTM each row represents one document, each column represents one term, and each value in the cell contains the number of times that term occurs in the document. This leads to a "sparse" data frame where many of the values are zero. Later on we will use this format to complete some more complex types of analysis on the corpus.

A tidy data frame, on the other hand, is not a sparse matrix. In a tidy data frame each variable is a column and each observation is a row. In the context of text mining this results in a one-token-per-row data frame (and in working with a corpus one-token-per-document-per-row). A *token* is a meaningful unit of text. Here a token refers to to a single word (unigram), but in other analyses a token could be pairs of words (bigrams), sentences or paragraphs.

To build the DTM we will remove all punctuation and make all words lower case. We will also remove "stop words", words that are not helpful in the analysis or do not contribute to the overall meaning of a text, such as "to", "the", and "and". In this example we will not "stem" the words. For example we will not take words such as "killing", "killed", "kills" and reduce them down to their word stem "kill". We will also keep all numbers, and we will keep all words that occur in at least one article.

In order to create the DTM, I again referenced this website https://data.library.virginia.edu/reading-pdf-files-into-r-for-text-mining/. To create the tidy data frame and complete the subsequent analysis I referenced https://www.tidytextmining.com/.

```r
# From this corpus, create a document term matrix.
# The DTM will be useful for topic modeling but
# can be converted into a tidy data frame
# for descriptive and exploratory analysis

vip.dtm <- DocumentTermMatrix(vip_corp,
                              control =
                                list(# remove all punctuation
                                     removePunctuation = TRUE,
                                     # remove common "stop words"
                                     stopwords = TRUE,
                                     # make all the characters lower case
                                     tolower = TRUE,
                                     # do not "stem" the words
                                     stemming = FALSE,
                                      # keep numbers
                                     removeNumbers = FALSE,
                                     # include words that appear in at least 1 document
                                     bounds = list(global = c(1, Inf))))


# Have a look at the dtm
inspect(vip.dtm)
```

```
## <<DocumentTermMatrix (documents: 47, terms: 5783)>>
## Non-/sparse entries: 11480/260321
## Sparsity           : 96%
## Maximal term length: 99
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs bear bears can grizzly national park said wildlife will wolves
##   11    0     3   0       4        2    0   19        0    2      0
##   12    9     9   0      17        6    7    5        7    4      0
##   15   14    16   1      15        0    0    9        2    2      0
##   25    0     0   2       0        6    9    3        3    4     10
##   32    1     0   7       0        0    0    1        5    9      0
##   35    0     1   4       0        0    0    9        4    3      2
##   36    0     0   9       0        0    0    0        0    2      0
##   39    0     2   1       1        0    4   11        8    1      2
##   42    0     0   3       0        0    0    0        8    1      8
##   8     0     0   7       0        1    0    4        2    0      4
```

```r
# Now we can convert the dtm to a tidy dataframe
vip.tidy <- tidy(vip.dtm)
head(vip.tidy, 5)
```

```
## # A tibble: 5 x 3
##   document term     count
##   <chr>    <chr>    <dbl>
## 1 1        100          1
## 2 1        2022         6
## 3 1        accessed     1
## 4 1        across       3
## 5 1        activity     1
```

One of the problems with building the corpus this way instead of directly from the original *files* list is that we lose the information in the document column. Here the document column has a number for each article, but we want the actual file name. However, we want to keep the document number as an id which will help to keep the figures a little cleaner.

3.1 (More) Text preprocessing

```r
# Make a new column called 'id' with the document number
vip.tidy$id <- vip.tidy$document
# Then make the data in the document a number
vip.tidy$document <- as.numeric(vip.tidy$document)

# Replace the document number with the document name from
# the list of files
for (x in 1:length(files)) {
    vip.tidy$document[vip.tidy$document == x] <- files[x]
}

# Make the characters in the document column all lowercase
vip.tidy$document <- tolower(vip.tidy$document)
```

3.2 Parsing out information from the document file name and adding an animal label

One thing that may be helpful for the content analysis is to take the information found in the document column and parse it out into other data columns. The file name contains the name of the article, the name of the newspaper, the date published, and the page number. Here we will take the name of the article, the newspaper and the date and separate that information into three new columns. We will also create a column called "animal" in which the article is classified as being about bears or wolves.

Luckily for us, the information in the file name is split by some character patterns. The article title is the first string of characters followed by two underscores (__), the newspaper is between two underscores and three underscores, and the date can be parsed out because of the pattern of letters and numbers.

We also look at the files name, and if "bear" is in the name then the animal column will be set to "bear", and if "bear" is not in the document name then the animal column will be set to "wolf". The result is a very rough approximation of the content of the text. In the future, we will need to come up with a better way to label the articles (if that is something we want to do in this study).

```r
vip.tidy <- vip.tidy %>%
    mutate(title = str_extract(document, regex("(.*?)(?=__)"))) %>%
    mutate(newspaper = str_extract(document, regex("(?<=__)[a-zA-Z]+-?_?.*(?=___)"))) %>%
    mutate(date = str_extract(document, "[a-zA-Z]+_[0-9]+_[0-9]+(?=__)")) %>%
    mutate(animal = ifelse(str_detect(vip.tidy$document, "bear") ==
        TRUE, "bear", "wolf"))
head(vip.tidy, 5)
```

```
## # A tibble: 5 x 8
##   document              term  count id    title   newspaper       date   animal
##   <chr>                 <chr> <dbl> <chr> <chr>   <chr>           <chr>  <chr>
## 1 2022_bear_update__dr~ 100       1 1     2022_b~ drayton_valley~ novem~ bear
## 2 2022_bear_update__dr~ 2022      6 1     2022_b~ drayton_valley~ novem~ bear
## 3 2022_bear_update__dr~ acces~    1 1     2022_b~ drayton_valley~ novem~ bear
## 4 2022_bear_update__dr~ across    3 1     2022_b~ drayton_valley~ novem~ bear
## 5 2022_bear_update__dr~ activ~    1 1     2022_b~ drayton_valley~ novem~ bear
```

3.2.1 Considerations Is there any other information we would want to include in the data frame? Is there a more efficient way to build a corpus? I ended up downloading each article as a pdf from newsbank. I had to do this because the format of the html from newsbank was beyond my beginner skills at web scrapping and text mining. While downloading the pdfs did not take long for approximately 50 articles, I imagine it may be more challenging with 100s of articles.

How can I incorporate the Article Coding table from the VIP course? Will I need to add in this information before building the corpus?

4. A Simple Content Analysis

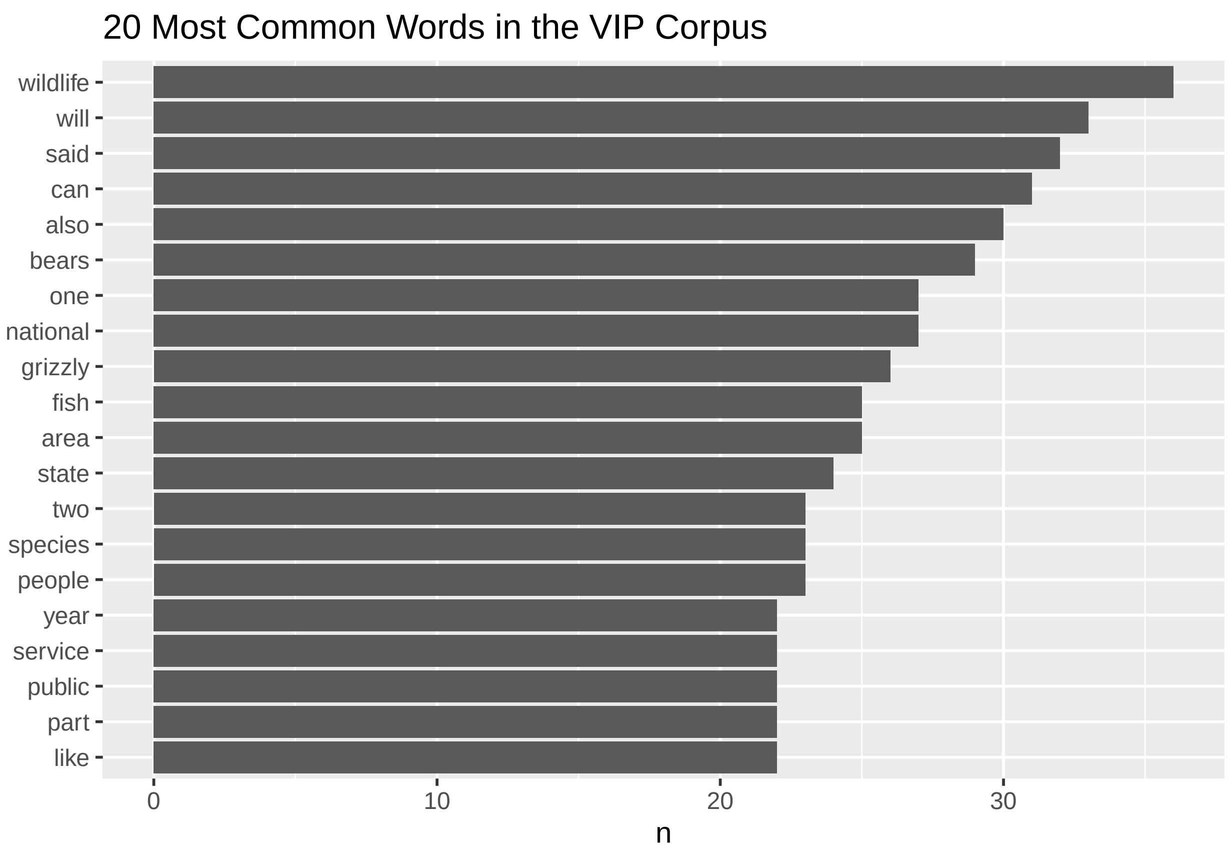
In the following section we will determine the most common words in the corpus, the overall sentiment of the articles, and look at the term frequency for the articles in the corpus.

4.1 Let's explore the corpus

We will start out simply by looking at the most common words in the corpus.

What are the 20 most common words in the corpus?

```
vip.tidy %>%
    count(term, sort = TRUE) %>%
    slice_max(n, n = 20) %>%
    mutate(term = reorder(term, n)) %>%
    ggplot(aes(n, term)) + geom_col() + labs(title = "20 Most Common Words in the VIP Corpus",
    y = NULL)
```



The most common word in the corpus is "wildlife"!

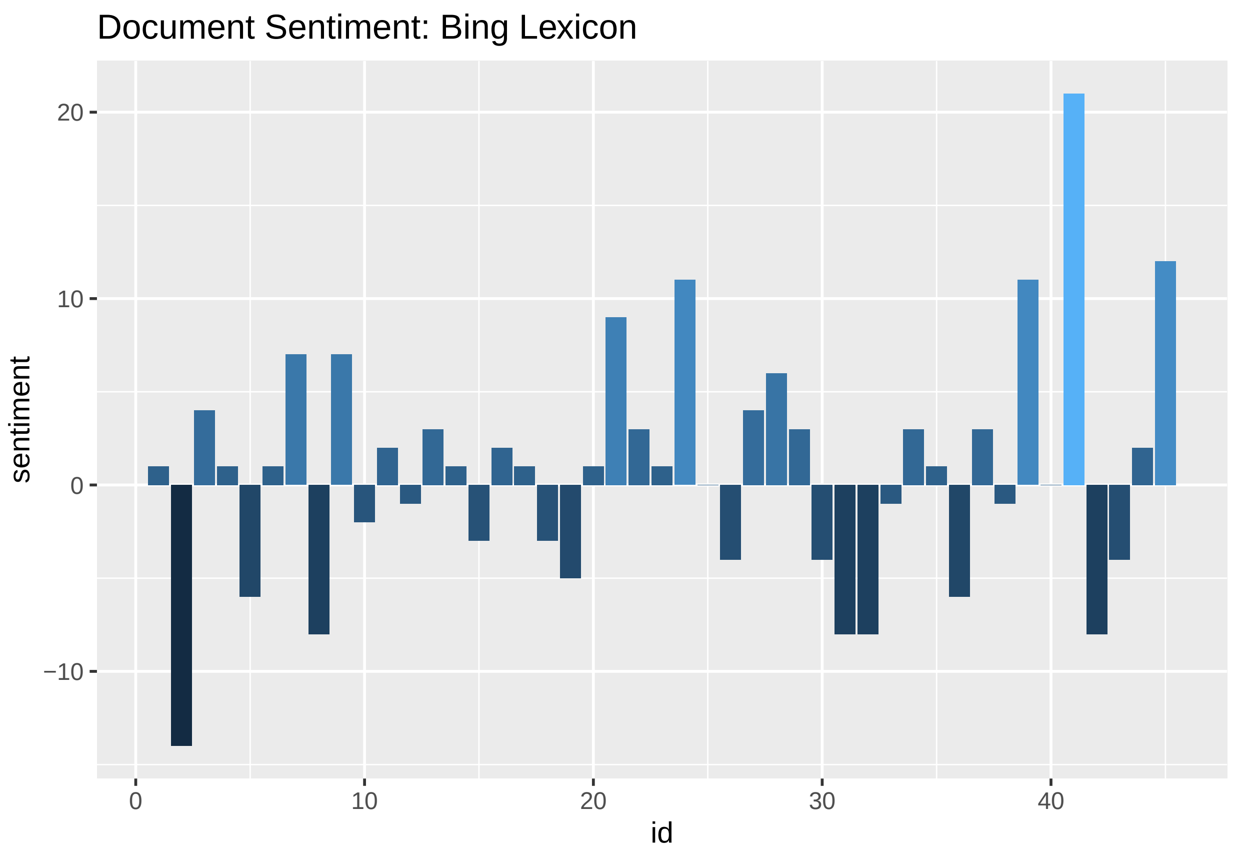
4.1.1 Considerations

Should we make our own list of stop words? For example, how helpful are the words *"will"*, *"said"*, *"can"*, *"also"*, and *"like"* to our content analysis and topic modeling?

4.2 Sentiment analysis of the corpus

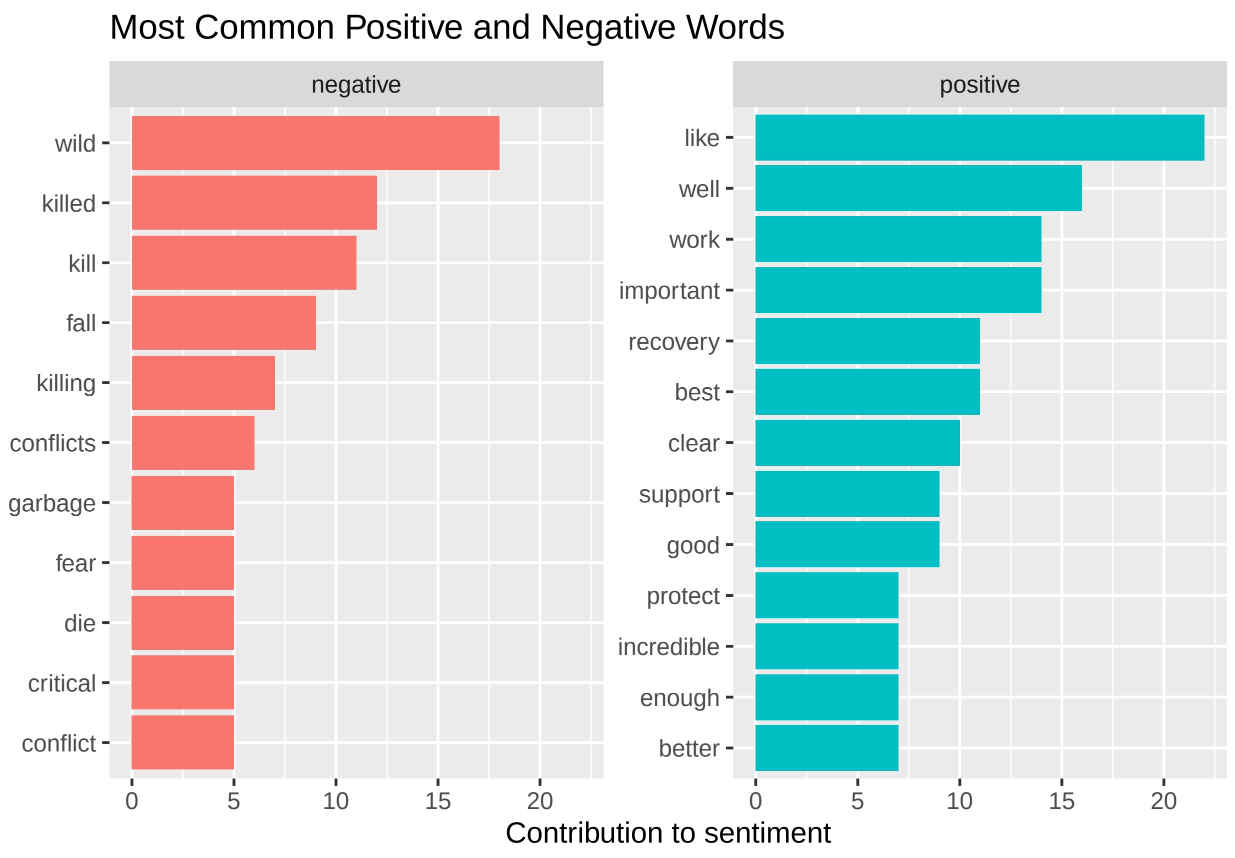
Often one wants to understand the overall sentiment or feeling of a text. Within the VIP class we referred to this as the "valence". One common approach to determining the sentiment of the text is to the combine or sum the sentiment content (or connotation) of each word within the text. There are a variety of sentiment lexicons that assign different sentiments to words. Here we will use the Bing lexicon which classifies a word as either positive or negative. There are other common sentiment lexicons available in R such as the AFINN lexicon which rates the valence of words on a scale of -5 (most negative) to +5 (most positive).

```r
# First, join with the Bing sentiment lexicon.
vip.sent <- vip.tidy %>%
    inner_join(get_sentiments("bing"), by = c(term = "word")) %>%
    count(document, sentiment) %>%
    pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
    mutate(sentiment = positive - negative)

## Try to relabel the x axis.
vip.sent$id <- row.names(vip.sent)
vip.sent$id <- as.numeric(vip.sent$id)
ggplot(vip.sent, aes(id, sentiment, fill = sentiment)) + geom_col(show.legend = FALSE) +
    labs(title = "Document Sentiment: Bing Lexicon")
```



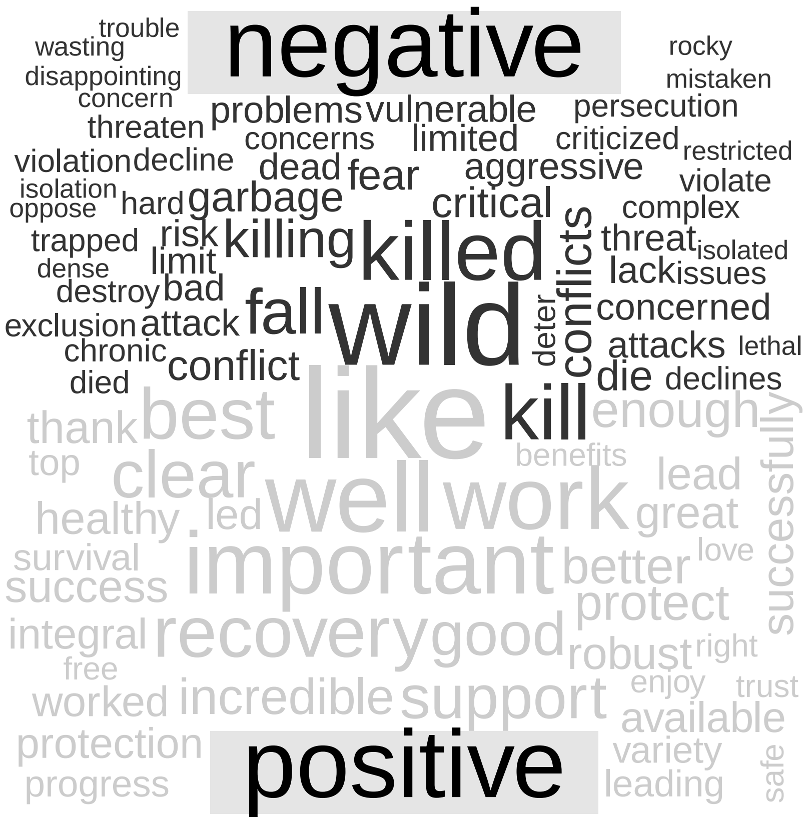
What are the most common positive and negative words?

```
vip.bing.counts <- vip.tidy %>%
    inner_join(get_sentiments("bing"), by = c(term = "word")) %>%
    count(term, sentiment, sort = TRUE) %>%
    ungroup()

vip.bing.counts %>%
    group_by(sentiment) %>%
    slice_max(n, n = 10) %>%
    ungroup() %>%
    mutate(term = reorder(term, n)) %>%
    ggplot(aes(n, term, fill = sentiment)) + geom_col(show.legend = FALSE) +
    facet_wrap(~sentiment, scales = "free_y") + labs(x = "Contribution to sentiment",
    y = NULL, title = "Most Common Positive and Negative Words")
```



Another fun visualization of content analysis is a **word cloud**.

```
vip.tidy %>%
    inner_join(get_sentiments("bing"), by = c(term = "word")) %>%
    count(term, sentiment, sort = TRUE) %>%
    acast(term ~ sentiment, value.var = "n", fill = 0) %>%
    comparison.cloud(colors = c("gray20", "gray80"), max.words = 100)
```



4.2.1 Considerations From this list of words we may want to consider "stemming" as described above. The Bing lexicon is very useful in this case since we only labeled an article code as either positive or negative. In the future we may want to consider using a different lexicon, for example is "wild" really a negative word in this project?

How well does the sentiment analysis in R match the valence we coded in Fall 2022?

4.3 Analyzing word and document frequency: tf-idf

Term frequency can be a useful metric to help one determine the meaning or focus of a text. However, often times common stop words like "the" and "to" will be the most frequent words within a text. A term frequency weighted by how often the term occurs in the text is another way to use term frequency instead of just removing stop words from the text or corpus. A type of weighted term frequency is term frequency - inverse document frequency, or tf-idf.

Term frequency refers to how frequently a word occurs in a document relative to the total number of words in the document.

$tf = \frac{n_{word_i}}{totalwordcount}$

Inverse document frequency is the natural log of the number of documents relative to the number of documents containing a term.

$idf = ln(\frac{n_{documents}}{n_{documentscontainingtheterm}})$

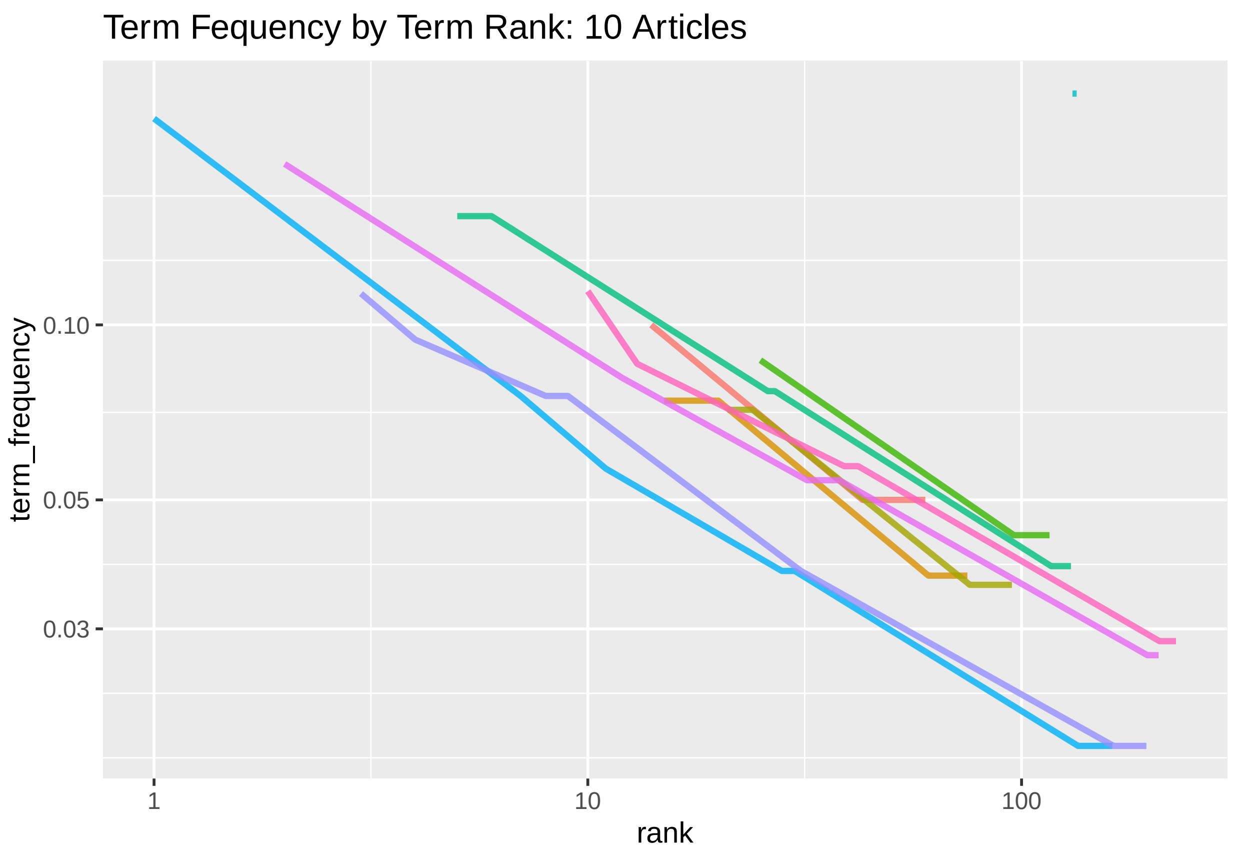
The idf will decrease the weight for commonly used words and increase the weight for words that are not used very much in a collection of documents. Term frequency - inverse document frequency (tf-idf) is term frequency and idf multiplied together. The idf part of the equation will decrease the weight of a term for common terms and increase the weight of a word for rarely used words. Basically, "the frequency of a term weighted for how rarely it occurs. Tf-idf answers the question,"How important is this word in a document?".

For part of this analysis we will look at a subset of the tidy data, just 10 articles and also by "animal".

Below we show the term frequency for the 10 article subset and the corpus grouped by animal type. Here we determine the count n for each term in a document and the total word count *total* for each document.

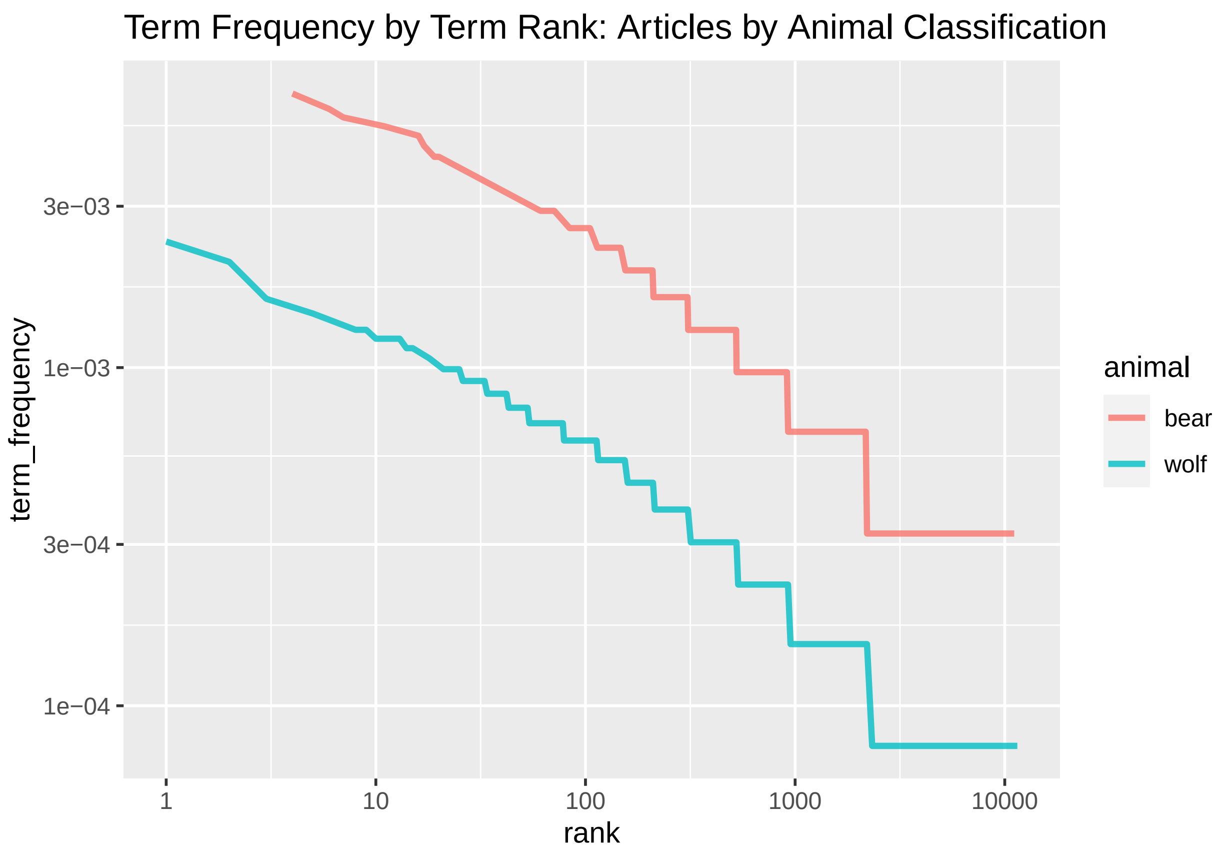
In the figure below the rank is determined by the word count, so the most common word is ranked 1, the second most common word is 2, and so on. Because we are looking at data on the scale of 0 to 1 (term-frequency) and 1 to over 100 the plot is has a log-log scale.

```r
# Take a subset of the documents for this part

vip.tidy.sub <- subset(vip.tidy, id == c("1", "2", "3", "4",
    "5", "6", "7", "8", "9", "10"))

# We need to order the table by count first
vip.words <- vip.tidy.sub %>%
    arrange(desc(count))

total.words <- vip.words %>%
    group_by(id) %>%
    summarise(total = sum(count))

vip.words <- left_join(vip.words, total.words)

vip.freq.rank <- vip.words %>%
    mutate(rank = row_number(), term_frequency = count/total) %>%
    ungroup()

vip.freq.rank %>%
    ggplot(aes(rank, term_frequency, color = document)) + geom_line(size = 1.1,
    alpha = 0.8, show.legend = FALSE) + scale_x_log10() + scale_y_log10() +
    labs(title = "Term Fequency by Term Rank: 10 Articles")
```



We could also group the corpus by the animal topic of the article.

```
# We need to order the table by count first
vip.words <- vip.tidy %>%
    arrange(desc(count))

total.words <- vip.words %>%
    group_by(animal) %>%
    summarise(total = sum(count))

vip.words <- left_join(vip.words, total.words)

vip.freq.rank <- vip.words %>%
    mutate(rank = row_number(), term_frequency = count/total) %>%
    ungroup()

vip.freq.rank %>%
    ggplot(aes(rank, term_frequency, color = animal)) + geom_line(size = 1.1,
    alpha = 0.8, show.legend = TRUE) + scale_x_log10() + scale_y_log10() +
    labs(title = "Term Frequency by Term Rank: Articles by Animal Classification")
```



4.3.1 Bind_tf_idf

There is a special function in R called bind_tf_idf() which will calculate the tf, idf and tf-idf from a tidy data frame for each document in a corpus.

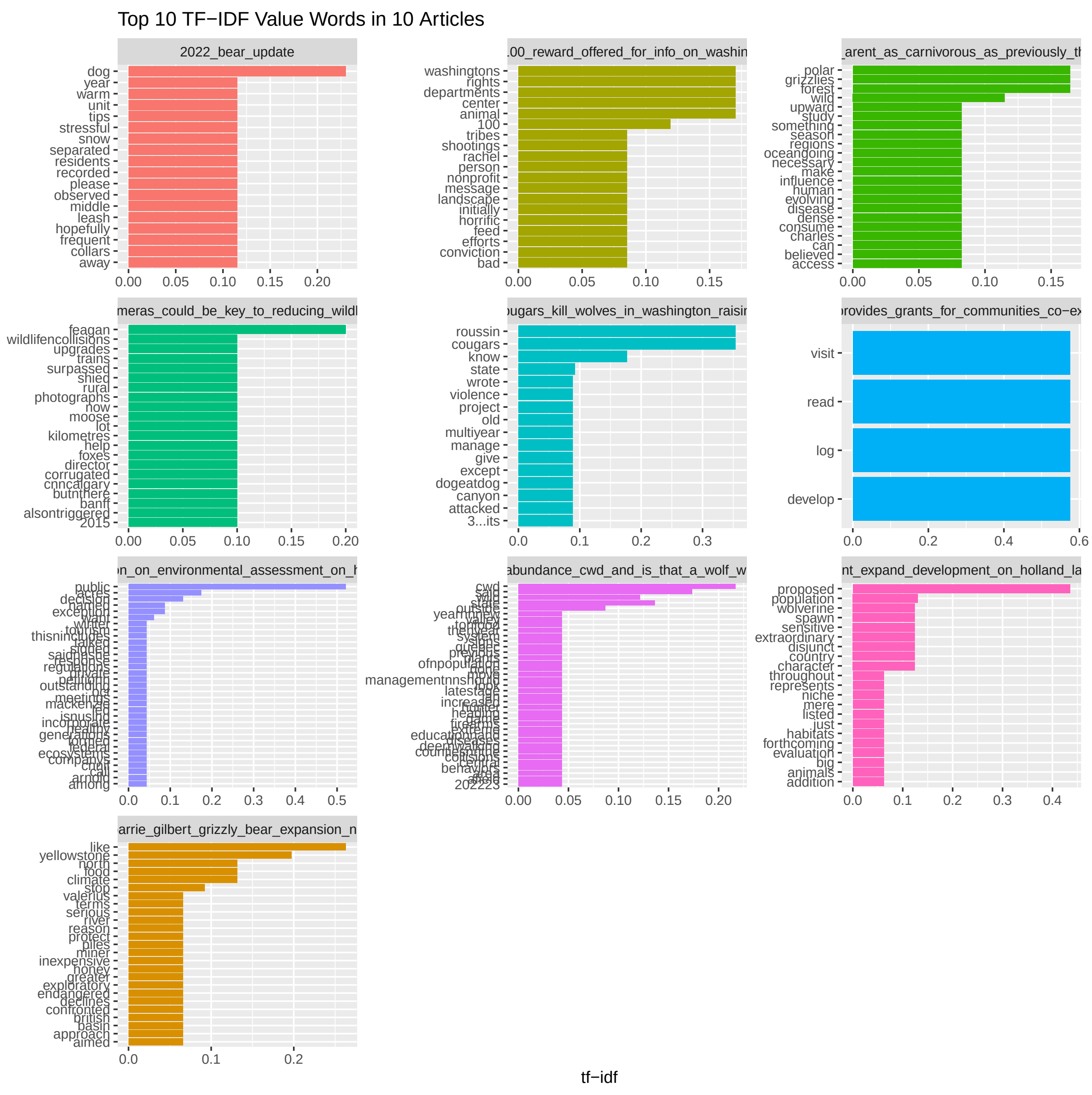
Idf will be lower, close to 0, for words that in occur in many of the documents in a collection and will be higher for words that occur in only a few documents of the collection.

The highest tf-idf values for a document will show the most important words for a document. For example, in the "Deer abundance_CWD_and_is_it_that_a_wolf" article, the term "cwd" has the highest tf-idf for the document.

```
# We need to order the table by count first
vip.words <- vip.tidy.sub %>%
    arrange(desc(count))

vip.tf.idf <- vip.words %>%
    bind_tf_idf(term, title, count)

vip.tf.idf %>%
    # select(-total) %>%
arrange(desc(tf_idf))
```

```
## # A tibble: 227 x 11
##    document   term   count id    title newspaper date  animal    tf   idf tf_idf
##    <chr>      <chr>  <dbl> <chr> <chr> <chr>     <chr> <chr>  <dbl> <dbl>  <dbl>
##  1 csu_provi~ devel~     1 6     csu_~ ag_journ~ nove~ wolf    0.25  2.30  0.576
##  2 csu_provi~ log        1 6     csu_~ ag_journ~ nove~ wolf    0.25  2.30  0.576
##  3 csu_provi~ read       1 6     csu_~ ag_journ~ nove~ wolf    0.25  2.30  0.576
##   4 csu_provi~ visit      1 6     csu_~ ag_journ~ nove~ wolf   0.25   2.30  0.576
##   5 decision_~ public    12 7     deci~ daily_in~ nove~ wolf   0.226  2.30  0.521
##   6 dont_expa~ propo~     7 9     dont~ daily_in~ octo~ wolf   0.189  2.30  0.436
##   7 cougars_k~ couga~     4 5     coug~ spokesma~ nove~ wolf   0.154  2.30  0.354
##   8 cougars_k~ rouss~     4 5     coug~ spokesma~ nove~ wolf   0.154  2.30  0.354
##   9 dr._barri~ like       4 10    dr._~ missouli~ nove~ bear   0.114  2.30  0.263
##  10 2022_bear~ dog        2 1     2022~ drayton_~ nove~ bear   0.1    2.30  0.230
## # ... with 217 more rows
```

```
vip.tf.idf %>%
    group_by(id) %>%
    slice_max(tf_idf, n = 10) %>%
    ungroup() %>%
    mutate(term = reorder(term, tf_idf)) %>%
    ggplot(aes(tf_idf, term, fill = id)) + geom_col(show.legend = FALSE) +
    labs(x = "tf-idf", y = NULL, title = "Top 10 TF-IDF Value Words in 10 Articles") +
    facet_wrap(~title, ncol = 3, scales = "free")
```



5. Topic Modeling

Topic modeling uses algorithms to examine a collection of texts and discover the topics of the corpus and the topic of each text. Topic models can organize a collection of documents based on these discovered themes. One very common approach is the latent dirichlet allocation which a probabilistic topic model.

5.1 Latent Dirichlet Allocation (LDA)

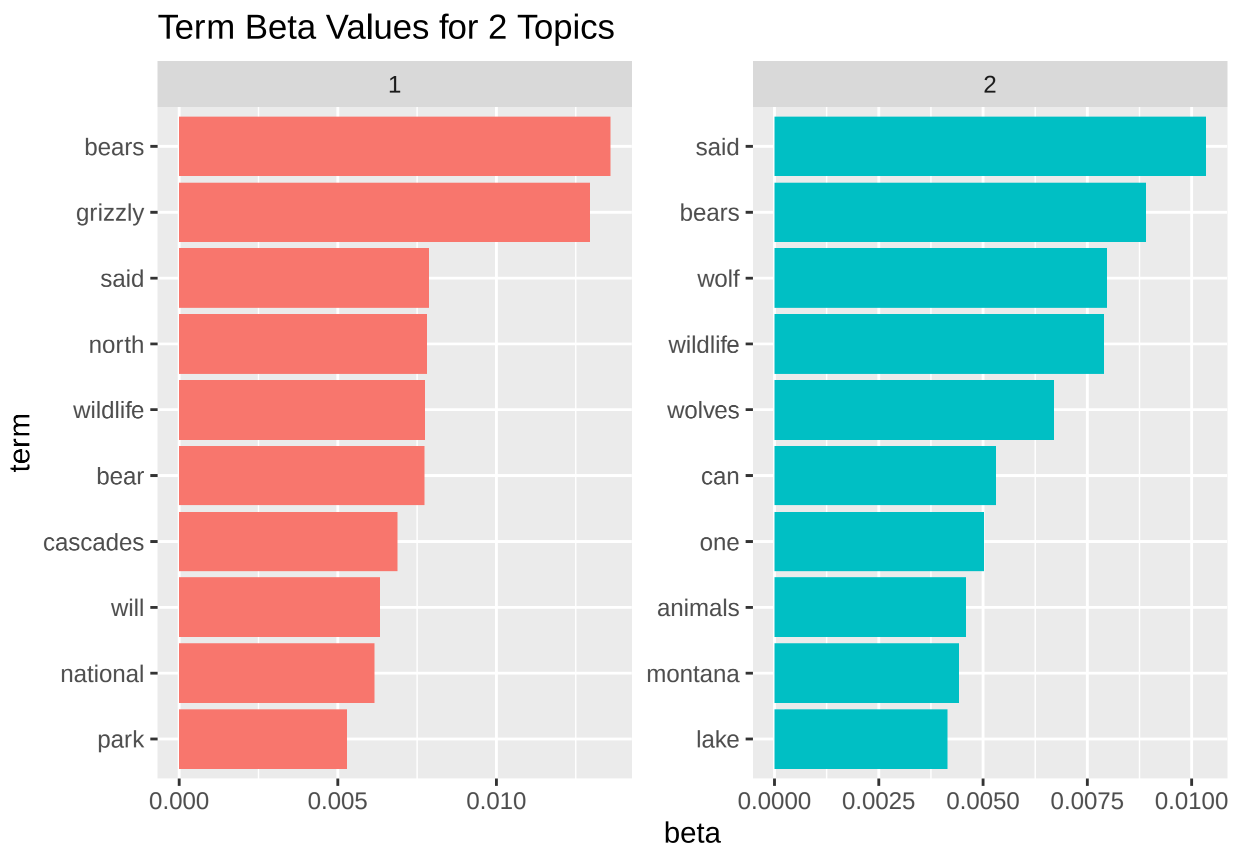
Latent dirichlet allocation (LDA) is a probabilistic topic model. The LDA assumes that documents can have multiple topics (i.e. a document is a mixture of topics) and that each topic is made up of a distribution of words. The words within one topic can also occur within another topic's distribution of words.

LDA is a generative model (i.e. it models how a document could be generated) which assumes that a set of topics are generated first. Then, the model randomly chooses a distribution over that collection of topics. Next, for each word in the text, the model will randomly choose a topic from the distribution and then randomly choose a word from the distribution of words associated with the chosen topic (Blei, 2003).

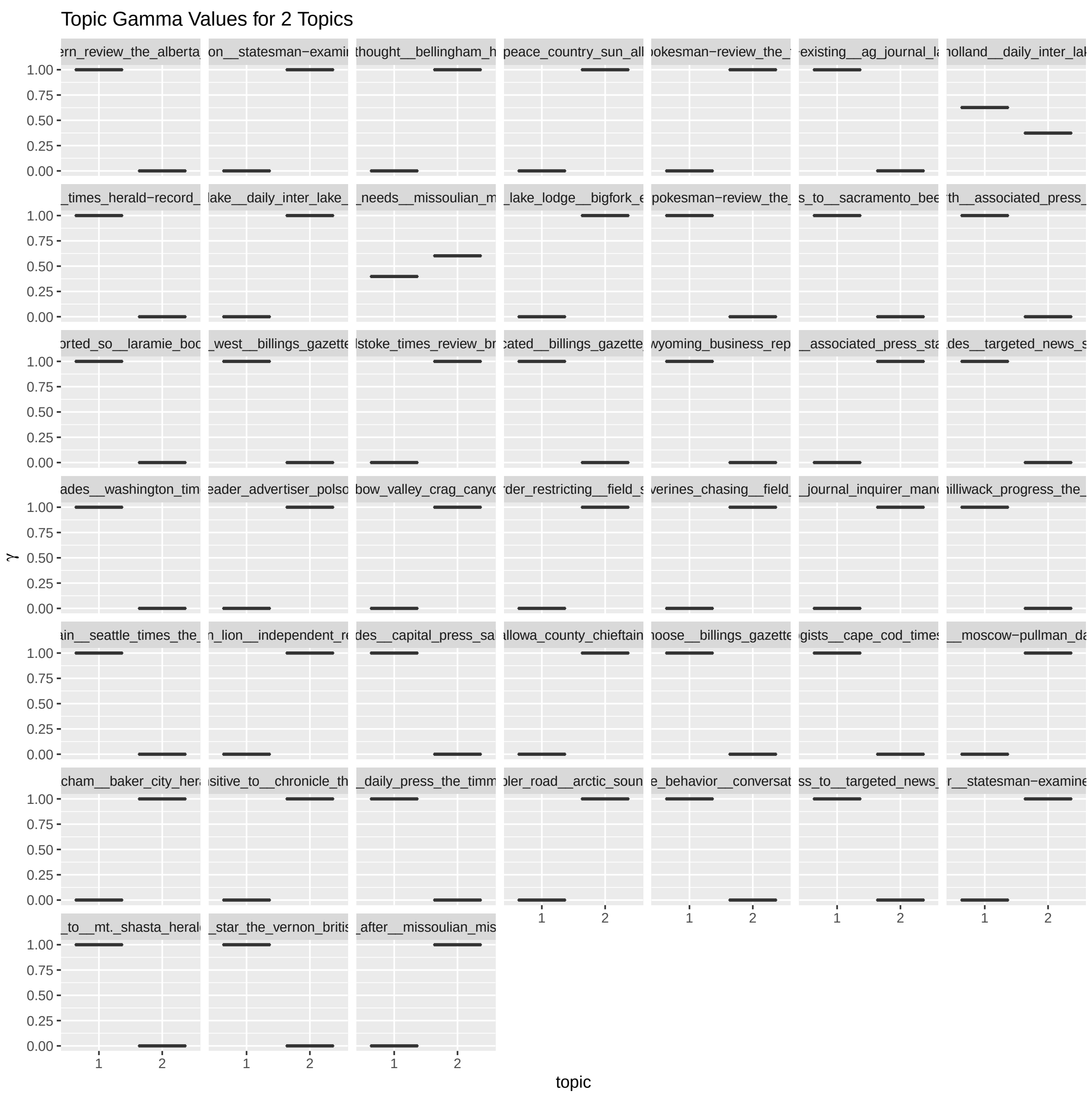
There are two (hyper)parameters of interest for a LDA. The first, *"beta"*, is the probability that a word belongs to the word distribution of a topic. The second, *"gamma"*, is the probability that a document belongs within a specific topic.

Let's start with 2 topics, to see if the model can identify articles as relating to bears or wolves.

```r
# recast the tidy data frame back to a DTM Not sure why,
# but otherwise I get a funny error
vip.dtm2 <- vip.tidy %>%
    cast_dtm(document, term, count)  # recast, not sure why, but otherwise I get a funny error

# Here, k is equal to 2 topics
vip.lda2 <- LDA(vip.dtm2, k = 2, control = list(seed = 1234))

# Get the beta values for each term
vip.topics2 <- tidy(vip.lda2, matrix = "beta")

vip.top.terms2 <- vip.topics2 %>%
    group_by(topic) %>%
    slice_max(beta, n = 10) %>%
    ungroup() %>%
    # arrange sorts the data
arrange(topic, -beta)

vip.top.terms2 %>%
    mutate(term = reorder_within(term, beta, topic)) %>%
    ggplot(aes(beta, term, fill = factor(topic))) + geom_col(show.legend = FALSE) +
    facet_wrap(~topic, scales = "free") + scale_y_reordered() +
    labs(title = "Term Beta Values for 2 Topics")
```



```
vip.docs2 <- tidy(vip.lda2, matrix = "gamma")

vip.docs2 %>%
    mutate(title = reorder(document, gamma * topic)) %>%
    ggplot(aes(factor(topic), gamma)) + geom_boxplot() + facet_wrap(~document) +
    labs(x = "topic", y = expression(gamma), title = "Topic Gamma Values for 2 Topics")
```



How well did the classifier work? We can compare the given animal topic to the modeled animal topic.

```r
vip.docs2.classification <- vip.docs2 %>%
    group_by(document) %>%
    slice_max(gamma) %>%
    ungroup()

vip.docs2.topics <- vip.tidy %>%
    group_by(document) %>%
    mutate(animal_orig = animal) %>%
    count(document, animal_orig) %>%
    group_by(document) %>%
    select(-n) %>%
    ungroup()
```

```r
vip.docs2.topics$topic <- ifelse(vip.docs2.topics$animal_orig ==
    "wolf", 1, 2)
vip.docs2.classification$animal_mod <- ifelse(vip.docs2.classification$topic ==
    1, "wolf", "bear")
vip.docs2.classification$document <- tolower(vip.docs2.classification$document)
vip.docs2.misclass <- vip.docs2.classification %>%
    inner_join(vip.docs2.topics, by = "document") %>%
    filter(animal_orig != animal_mod)

vip.docs2.misclass %>%
    count(animal_orig)
```

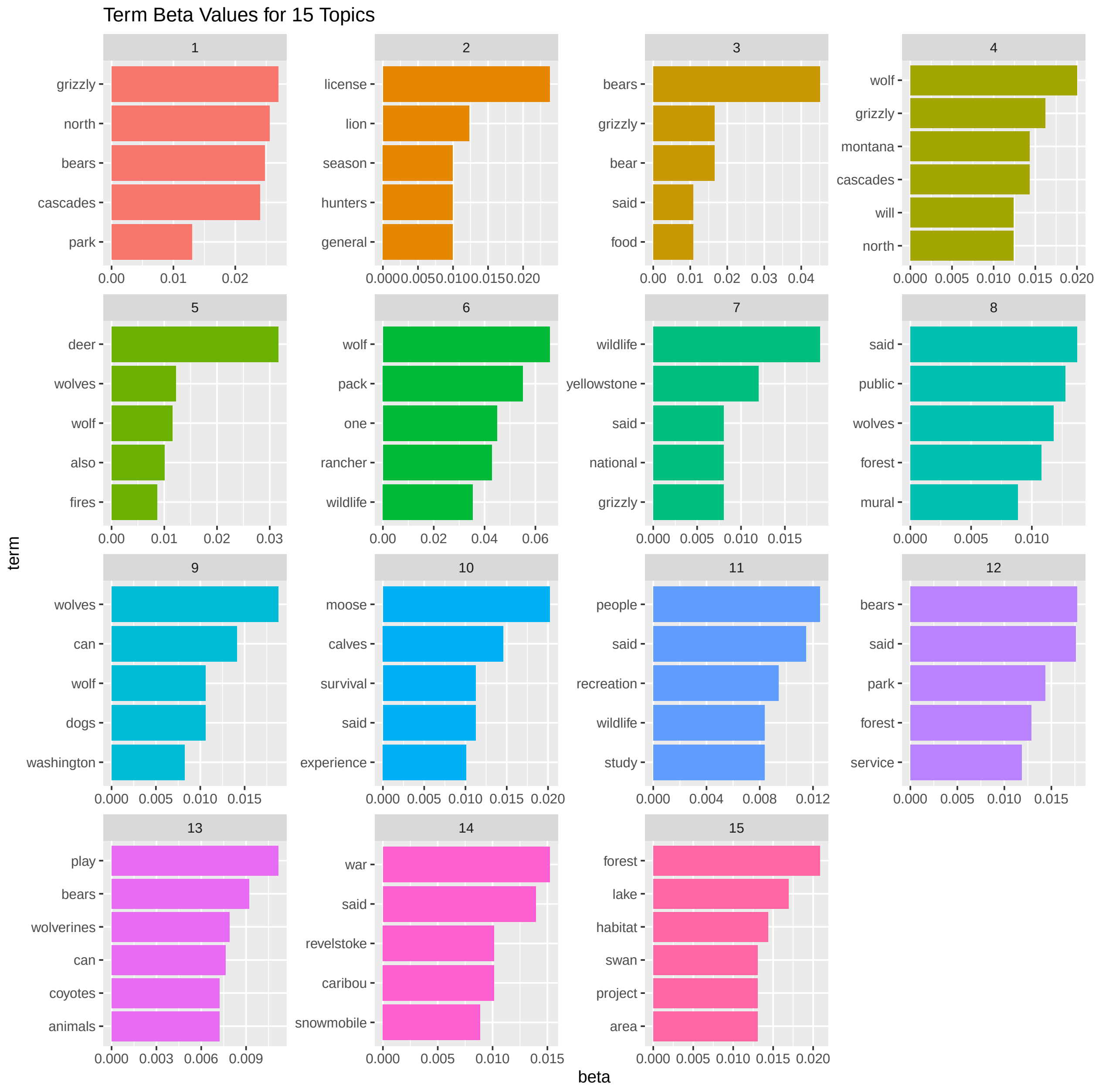
```
## # A tibble: 2 x 2
##   animal_orig     n
##   <chr>       <int>
## 1 bear            6
## 2 wolf           18
```

The output here shows the number of misclassified animal topic. There were 6 bear articles that the model misclassified as wolf articles, and there were 18 wolf articles that the model misclassified as bear articles.

The model didn't work that well. 24 of the 47 articles were misclassified. The vast majority were articles that were supposed to be about wolves. This may very well have to do with how I classified the articles (was bear in the article name, if not label the article as wolf).

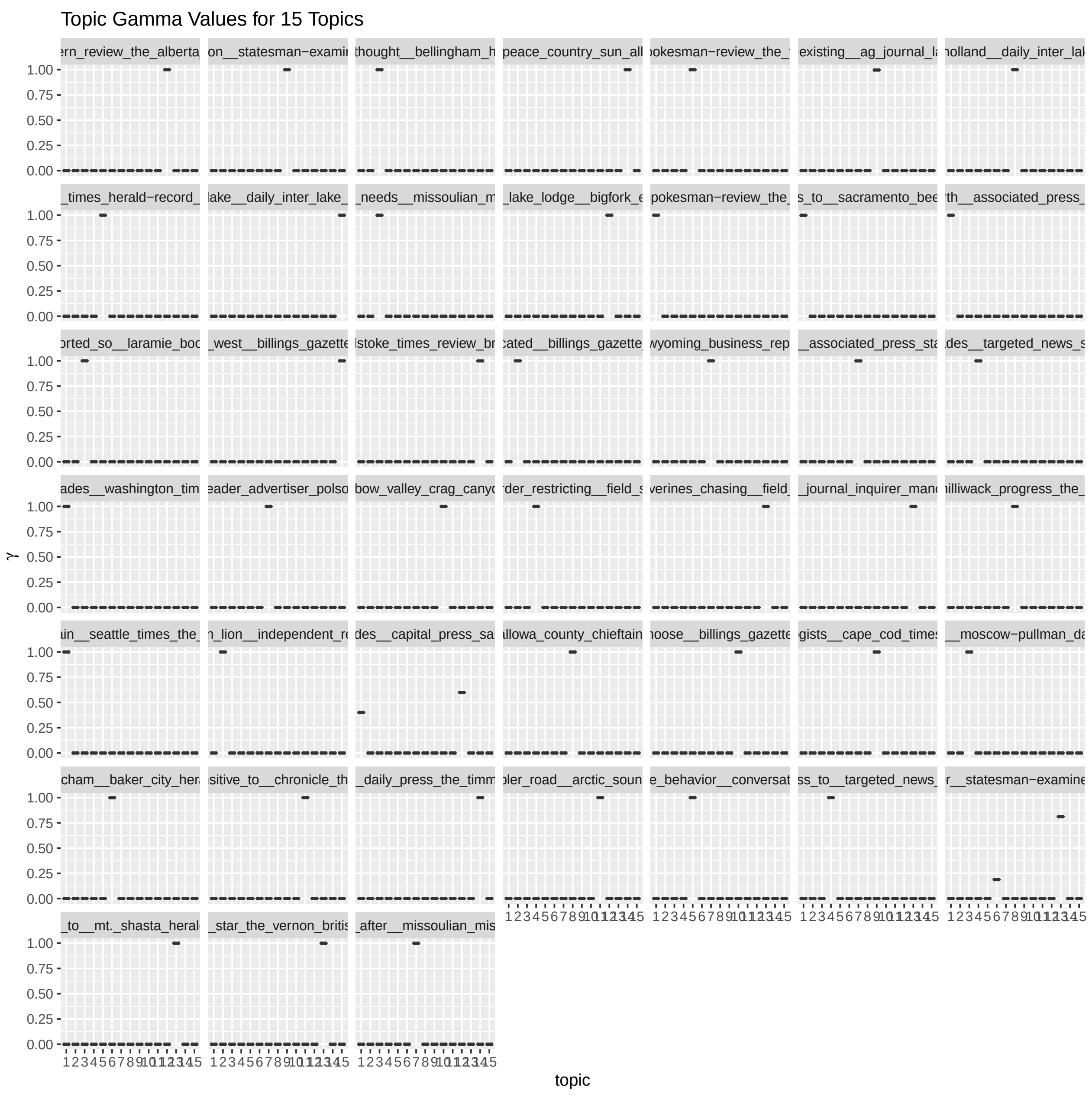
Let's do 15 topics (the number of codes in the class code book).

```r
vip.lda15 <- LDA(vip.dtm2, k = 15, control = list(seed = 1234))

vip.topics15 <- tidy(vip.lda15, matrix = "beta")

vip.top.terms15 <- vip.topics15 %>%
    group_by(topic) %>%
    slice_max(beta, n = 5) %>%
    ungroup() %>%
    arrange(topic, -beta)  # arrange sorts the data

vip.top.terms15 %>%
    mutate(term = reorder_within(term, beta, topic)) %>%
    ggplot(aes(beta, term, fill = factor(topic))) + geom_col(show.legend = FALSE) +
    facet_wrap(~topic, scales = "free") + scale_y_reordered() +
    labs(title = "Term Beta Values for 15 Topics")
```



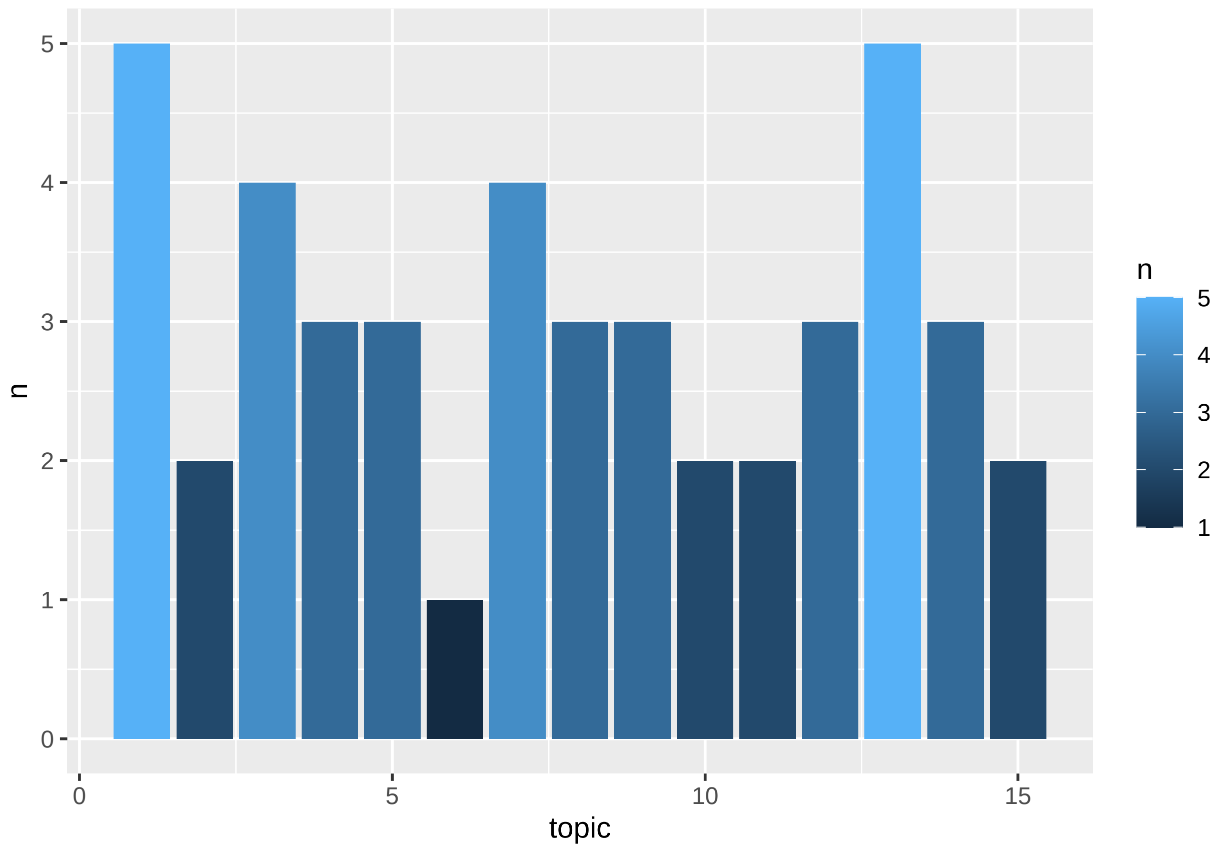
Considerations How would we describe the topics based on these words?

```r
vip.docs15 <- tidy(vip.lda15, matrix = "gamma")

vip.docs15 %>%
    mutate(title = reorder(document, gamma * topic)) %>%
    ggplot(aes(factor(topic), gamma)) + geom_boxplot() + facet_wrap(~document) +
    labs(x = "topic", y = expression(gamma), title = "Topic Gamma Values for 15 Topics")
```



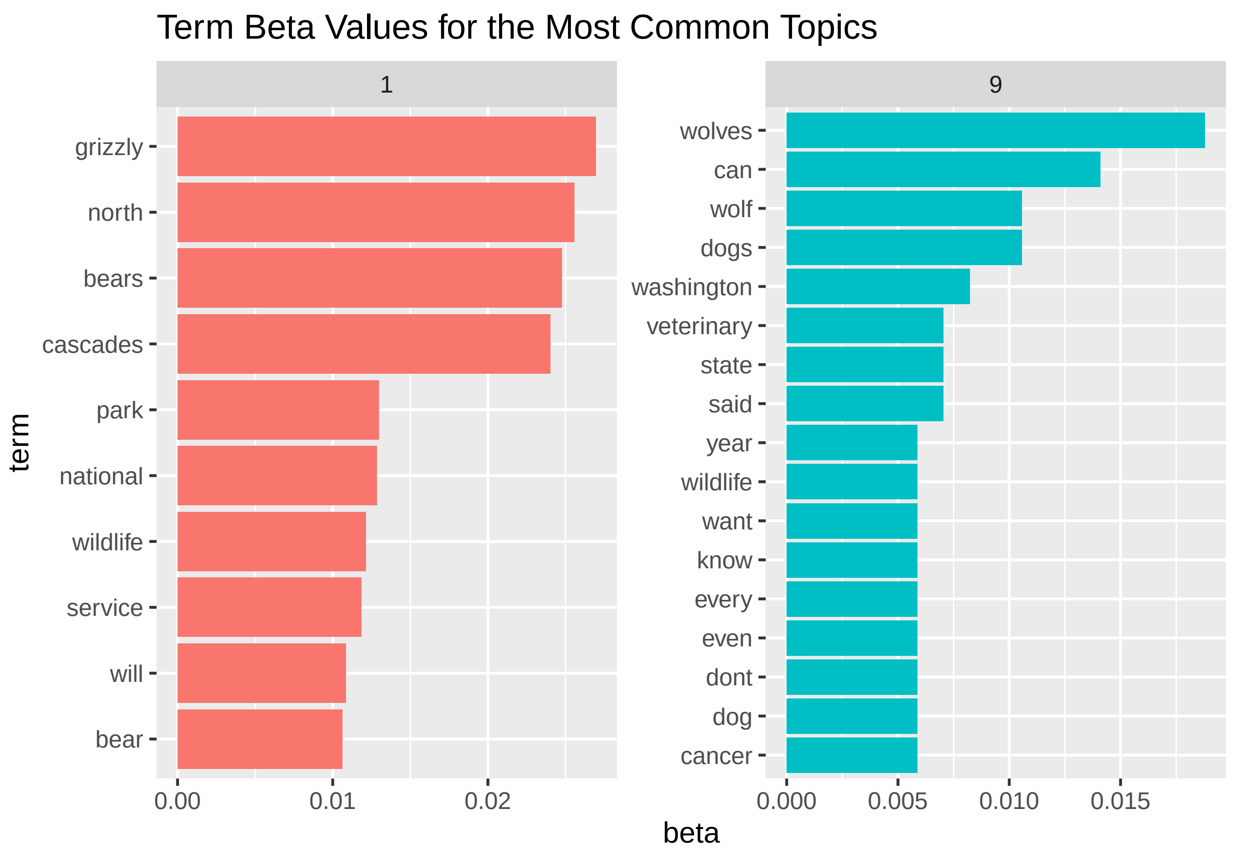
What are the most common topics in the corpus?

```
vip.docs15.classification <- vip.docs15 %>%
    group_by(document) %>%
    slice_max(gamma) %>%
    ungroup()

vip.docs15.classification %>%
    count(topic) %>%
    ggplot(aes(topic, n, fill = n)) + geom_bar(stat = "identity")
```



The most common topics in the corpus are topic 1 and topic 9.

```
vip.top.terms15.2 <- vip.topics15 %>%
    group_by(topic) %>%
    slice_max(beta, n = 10) %>%
    ungroup() %>%
    arrange(topic, -beta)


vip.top.terms15.2 %>%
    mutate(term = reorder_within(term, beta, topic)) %>%
    filter(topic == 1 | topic == 9) %>%
    ggplot(aes(beta, term, fill = factor(topic))) + geom_col(show.legend = FALSE) +
    facet_wrap(~topic, scales = "free") + scale_y_reordered() +
    labs(title = "Term Beta Values for the Most Common Topics")
```



References

Blei, David M. "Probabilistic topic models." Communications of the ACM 55.4 (2012): 77-84.

Ford, Clay. Reading PDF Files into R for Text Mining, University of Virginia Library, 14 May 2019, https://data.library.virginia.edu/reading-pdf-files-into-r-for-text-mining/.

Silge, Julia, and David Robinson. Text mining with R: A tidy approach. " O'Reilly Media, Inc.", 2017.

Stemler, Steve. "An overview of content analysis." Practical assessment, research, and evaluation 7.1 (2000): 17.