Replication Extension with Twitter Data: Using GloVe

### Environment Setup

### Workflow

* Use the ‘tweets\_congress.csv’, preprocesses the tweets, and then fits word embeddings (using a GloVe model via the text2vec package).
* We then average the resulting word vectors for each party, do PCA on those “party embeddings,” and interpret the first principal component as an ideological axis.

This broadly replicates the logic of Rheault & Cochrane’s approach but on congressional tweets rather than speeches.

## STEP 1: Load, inspect, and filter data

We read in tweets\_congress.csv and inspect the data structure. Then we filter the data to keep only Democratic/Republican members and remove retweets.

# =====================================================  
# STEP 0: Load packages & set up environment  
# =====================================================  
library(tidyverse) # data wrangling

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.5  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.3 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(text2vec) # tokenization + GloVe  
library(stopwords) # standard stopword lists  
library(SnowballC) # optional for stemming (not used here)  
library(ggrepel)  
  
# For reproducibility:  
set.seed(123)

# =====================================================  
# STEP 1: Read Data & Inspect  
# =====================================================  
# Input: 'tweets\_congress.csv'  
# text = tweet content  
# Party = "D" or "R"  
# author = unique user name (twitter handle)  
# retweet\_author = NA if original tweet, otherwise user retweeted  
# ...  
  
tweets <- read\_csv("tweets\_congress.csv")

## Rows: 1266542 Columns: 10  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (10): author, text, date, bios, retweet\_author, Name, Link, State, Party...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

# glimpse(tweets)  
  
# Filter to major parties  
tweets <- tweets %>%   
 filter(Party %in% c("D","R"))  
  
# Drop retweets to avoid artificial inflation of language patterns  
tweets <- tweets %>%   
 filter(is.na(retweet\_author))

## STEP 2: Preprocessing

We remove retweets (optional), drop URLs, convert text to lowercase, remove stopwords, punctuation, emojis, etc.

# =====================================================  
# STEP 2: Preprocess tweet text  
# =====================================================  
# We'll define a helper function to:  
# - remove URLs  
# - remove mentions  
# - lower-case  
# - remove punctuation & digits  
# - remove emojis  
# - etc.  
  
# Define enhanced text cleaning function with better emoji and single-letter handling  
clean\_tweet <- function(txt) {  
 txt %>%  
 str\_remove\_all("http\\S+|www\\S+") %>% # remove URLs  
 str\_remove\_all("@\\w+") %>% # remove @handles  
 str\_to\_lower() %>% # to lower case  
 str\_replace\_all("[[:punct:]]", " ") %>% # remove punctuation  
 str\_remove\_all("[0-9]+") %>% # remove digits  
 str\_remove\_all("\\b[a-zA-Z]\\b") %>% # remove single letters  
 str\_remove\_all("\\p{So}|\\p{Sk}") %>% # remove emojis & symbols  
 str\_squish() # trim extra whitespace  
}  
  
# Apply cleaning function to tweets  
tweets <- tweets %>%  
 mutate(text\_clean = clean\_tweet(text))  
  
# Define expanded stopwords  
# Basic English stopwords  
basic\_stops <- stopwords("en")  
  
# Common Twitter/social media terms  
twitter\_stops <- c("amp", "rt", "via", "im", "thats", "id", "us", "ive", "dont",   
 "cant", "youre", "youve", "isnt", "wasnt", "didnt", "wont",  
 "couldnt", "shouldnt", "wouldnt", "arent")  
  
# Single letters (already removed in cleaning, but added here for completeness)  
single\_letters <- letters  
  
# Combine all stopwords  
my\_stopwords <- c(basic\_stops, twitter\_stops, single\_letters)

## STEP 3: Building a Term–Cooccurrence Matrix (TCM)

Using the text2vec package, we create an iterator over tokens, build a vocabulary, and construct a TCM with a defined “window size.”

The original model used python’s doc2vec package, but here we use GloVe with text2vec, which is a convenient parallel in R.

# =====================================================  
# STEP 3: Tokenize & Build a TCM  
# =====================================================  
  
# Tokenize tweets  
tokens <- space\_tokenizer(tweets$text\_clean)  
  
# Create an itoken object with author IDs from the cleaned tweets  
it <- itoken(tokens,   
 ids = tweets$author,  
 progressbar = TRUE)  
  
# Build vocabulary  
vocab <- create\_vocabulary(it, stopwords = my\_stopwords,   
 ngram = c(1L, 1L))  
  
# Examine vocabulary size (optional)  
vocab\_size <- nrow(vocab)  
cat("Initial vocabulary size:", vocab\_size, "\n")

## Initial vocabulary size: 131515

# Prune vocabulary to remove rare words  
vocab <- prune\_vocabulary(vocab, term\_count\_min = 5)  
pruned\_size <- nrow(vocab)  
cat("Pruned vocabulary size:", pruned\_size, "\n")

## Pruned vocabulary size: 42316

cat("Removed", vocab\_size - pruned\_size, "rare terms\n")

## Removed 89199 rare terms

# Create vectorizer  
vectorizer <- vocab\_vectorizer(vocab)  
  
# Create Term Co-occurrence Matrix with window of 10 words  
#### The 'skip\_grams\_window' sets the context window.   
##### Larger window -> captures broader semantic relatedness  
tcm <- create\_tcm(it, vectorizer, skip\_grams\_window = 10)

## STEP 4: Fit GloVe Model

GloVe is a form of word embeddings that, like skip‐gram or CBOW, captures word co‐occurrences in a lower‐dimensional vector space.

Here we choose a 100-dim embedding size. Rheault & Cochrane used 200 dims for doc2vec. But for GloVe on tweets, 50–200 can be fine. We pick 100 here as a balance.

# Configure and train GloVe model with 100 dimensions  
glove\_dim <- 100  
glove\_model <- GlobalVectors$new(rank = glove\_dim,   
 x\_max = 10,   
 learning\_rate = 0.15)  
  
# Train the model with 15 iterations  
fit\_glove <- glove\_model$fit\_transform(tcm, n\_iter = 15,   
 convergence\_tol = 0.01)

## INFO [16:25:40.522] epoch 1, loss 0.1807  
## INFO [16:26:01.001] epoch 2, loss 0.1226  
## INFO [16:26:21.009] epoch 3, loss 0.1064  
## INFO [16:26:40.990] epoch 4, loss 0.0979  
## INFO [16:27:00.793] epoch 5, loss 0.0924  
## INFO [16:27:21.284] epoch 6, loss 0.0886  
## INFO [16:27:41.132] epoch 7, loss 0.0857  
## INFO [16:28:01.372] epoch 8, loss 0.0834  
## INFO [16:28:21.747] epoch 9, loss 0.0815  
## INFO [16:28:41.506] epoch 10, loss 0.0800  
## INFO [16:29:01.641] epoch 11, loss 0.0788  
## INFO [16:29:22.259] epoch 12, loss 0.0777  
## INFO [16:29:42.172] epoch 13, loss 0.0768  
## INFO [16:30:02.204] epoch 14, loss 0.0759  
## INFO [16:30:22.434] epoch 15, loss 0.0752  
## INFO [16:30:22.436] Success: early stopping. Improvement at iterartion 15 is less then convergence\_tol

# text2vec logs the training loss each epoch.   
# - 'loss' should steadily decrease  
# - 'Success: early stopping' means it converged or no improvement  
  
# Get context component (distinct to GloVe)  
context <- glove\_model$components  
  
# Combine to get full word embeddings  
word\_vectors <- fit\_glove + t(context)  
# Each row is a word in the vocab; each col is a dimension (1..100).  
  
  
# -----------Save the embeddings for future use--------------#  
saveRDS(word\_vectors, file = "data/tweet\_word\_vectors.rds")  
  
# next time open the file, just do:  
# word\_vectors <- readRDS("data/tweet\_word\_vectors.rds")

The loss steadily decreases from 0.1962 to 0.0817 over 15 epoches, which means the model is learning consistent word co‐occurrence patterns. Early stopping with 15 epochs means these were enough for model to converge under the chosen learning rate and tolerance (convergence\_tol).

## STEP 5: Create User‐Level Embeddings

After fitting word‐level vectors, we get an embedding for each term in the vocabulary. Next, we create an average embedding for each user by: 1) Splitting each tweet into words 2) Summing/averaging the word vectors for that tweet 3) Aggregating all tweets of the same user 4) Summarize or average again.

# =====================================================  
# STEP 5: Create USER-level embeddings  
# =====================================================  
# Why user-level instead of party-level?  
# - If we only compute (D, R), we end up with exactly 2 points -> meaningless for PCA  
# - If we do user-level, we might have 500 or 1000 Republicans, 500 or 1000 Democrats, etc.   
# Then we can do PCA on that bigger set.  
  
library(data.table)

##   
## Attaching package: 'data.table'

## The following objects are masked from 'package:lubridate':  
##   
## hour, isoweek, mday, minute, month, quarter, second, wday, week,  
## yday, year

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

## The following object is masked from 'package:purrr':  
##   
## transpose

# Convert to data.table for more efficient processing  
dt\_tweets <- as.data.table(tweets)  
  
# ---------------------------------------  
  
# (A) Unnest words (tidyverse approach)  
# Tokenize by splitting each tweet into words  
tweet\_words <- dt\_tweets[, .(word = unlist(str\_split(text\_clean, "\\s+"))),   
 by = .(author, Party)]  
  
# Remove stopwords and short tokens  
tweet\_words <- tweet\_words[!(word %in% my\_stopwords) & nchar(word) > 2]  
  
# ---------------------------------------  
  
# (B) Summarize per user  
# Create a wordset for each user (unique words used by each author)  
user\_wordsets <- tweet\_words[, .(wordset = list(unique(word))), by = .(author, Party)]  
  
# Initialize a list to store user embeddings  
user\_embeddings\_list <- list()  
  
# For each user, compute the average of their word embeddings  
for (i in 1:nrow(user\_wordsets)) {  
 # Get list of words for this user  
 words\_i <- unlist(user\_wordsets$wordset[i])  
   
 # Find which words are in our vocabulary  
 valid\_words <- intersect(words\_i, rownames(word\_vectors))  
   
 if (length(valid\_words) == 0) {  
 # No valid words in vocabulary  
 emb\_vec <- rep(NA, glove\_dim)  
 } else {  
 # Extract embeddings for these words  
 emb\_mat <- word\_vectors[valid\_words, , drop = FALSE]  
   
 # Average the embeddings  
 emb\_vec <- colMeans(emb\_mat)  
 }  
   
 # Store in the list  
 user\_embeddings\_list[[i]] <- emb\_vec  
}  
  
# ---------------------------------------  
# (C) Store embeddings in a dataframe  
  
# Convert list to matrix  
user\_emb\_matrix <- do.call(rbind, user\_embeddings\_list)  
  
# Create dataframe with user info  
user\_embeds <- data.frame(  
 author = user\_wordsets$author,  
 party = user\_wordsets$Party,  
 user\_emb\_matrix  
)  
  
# ---------------------------------------  
# Save user embeddings  
saveRDS(user\_embeds, "data/user\_embeddings\_glove.rds")

## STEP 6: Principal Component Analysis

* We run PCA on the final party embeddings.
* If the twitter data inherits the pattern as the original paper, the first principal component might represent left–right ideological diversion, and the second might capture additional structure (e.g., majority vs. minority party, or other divides).

# Remove rows with missing values  
user\_embeds\_complete <- user\_embeds[complete.cases(user\_embeds), ]  
  
# Check how many users we have after removing NA values  
cat("Total users with complete embeddings:", nrow(user\_embeds\_complete), "\n")

## Total users with complete embeddings: 423

cat("Users by party:\n")

## Users by party:

print(table(user\_embeds\_complete$party))

##   
## D R   
## 232 191

# Extract embedding matrix for PCA  
emb\_data <- user\_embeds\_complete %>%  
 select(-author, -party)  
  
# Run PCA  
pca\_model <- prcomp(emb\_data, scale. = TRUE)  
  
# Summary of variance explained  
pca\_summary <- summary(pca\_model)  
print(pca\_summary)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## Standard deviation 5.9938 4.1910 3.4124 2.4000 1.9417 1.69644 1.49476  
## Proportion of Variance 0.3593 0.1757 0.1164 0.0576 0.0377 0.02878 0.02234  
## Cumulative Proportion 0.3593 0.5349 0.6514 0.7089 0.7467 0.77544 0.79778  
## PC8 PC9 PC10 PC11 PC12 PC13 PC14  
## Standard deviation 1.40923 1.36358 1.24145 0.97530 0.94632 0.90154 0.85114  
## Proportion of Variance 0.01986 0.01859 0.01541 0.00951 0.00896 0.00813 0.00724  
## Cumulative Proportion 0.81764 0.83623 0.85165 0.86116 0.87011 0.87824 0.88548  
## PC15 PC16 PC17 PC18 PC19 PC20 PC21  
## Standard deviation 0.83135 0.75199 0.72330 0.70083 0.67343 0.66286 0.63344  
## Proportion of Variance 0.00691 0.00565 0.00523 0.00491 0.00454 0.00439 0.00401  
## Cumulative Proportion 0.89240 0.89805 0.90328 0.90819 0.91273 0.91712 0.92114  
## PC22 PC23 PC24 PC25 PC26 PC27 PC28  
## Standard deviation 0.61569 0.58726 0.57591 0.55257 0.53137 0.52963 0.51727  
## Proportion of Variance 0.00379 0.00345 0.00332 0.00305 0.00282 0.00281 0.00268  
## Cumulative Proportion 0.92493 0.92837 0.93169 0.93475 0.93757 0.94037 0.94305  
## PC29 PC30 PC31 PC32 PC33 PC34 PC35  
## Standard deviation 0.49596 0.48376 0.4800 0.46265 0.44927 0.44164 0.42750  
## Proportion of Variance 0.00246 0.00234 0.0023 0.00214 0.00202 0.00195 0.00183  
## Cumulative Proportion 0.94551 0.94785 0.9502 0.95229 0.95431 0.95626 0.95809  
## PC36 PC37 PC38 PC39 PC40 PC41 PC42  
## Standard deviation 0.42335 0.41476 0.4125 0.39707 0.39442 0.38635 0.38140  
## Proportion of Variance 0.00179 0.00172 0.0017 0.00158 0.00156 0.00149 0.00145  
## Cumulative Proportion 0.95988 0.96160 0.9633 0.96488 0.96644 0.96793 0.96938  
## PC43 PC44 PC45 PC46 PC47 PC48 PC49  
## Standard deviation 0.3739 0.37016 0.3606 0.34753 0.34158 0.33355 0.32480  
## Proportion of Variance 0.0014 0.00137 0.0013 0.00121 0.00117 0.00111 0.00105  
## Cumulative Proportion 0.9708 0.97215 0.9735 0.97466 0.97583 0.97694 0.97799  
## PC50 PC51 PC52 PC53 PC54 PC55 PC56  
## Standard deviation 0.32179 0.31476 0.31050 0.30721 0.2997 0.29045 0.2837  
## Proportion of Variance 0.00104 0.00099 0.00096 0.00094 0.0009 0.00084 0.0008  
## Cumulative Proportion 0.97903 0.98002 0.98098 0.98193 0.9828 0.98367 0.9845  
## PC57 PC58 PC59 PC60 PC61 PC62 PC63  
## Standard deviation 0.27748 0.27120 0.26685 0.2648 0.26204 0.25535 0.25260  
## Proportion of Variance 0.00077 0.00074 0.00071 0.0007 0.00069 0.00065 0.00064  
## Cumulative Proportion 0.98524 0.98598 0.98669 0.9874 0.98808 0.98873 0.98937  
## PC64 PC65 PC66 PC67 PC68 PC69 PC70  
## Standard deviation 0.24935 0.24342 0.24080 0.23147 0.22762 0.2237 0.22016  
## Proportion of Variance 0.00062 0.00059 0.00058 0.00054 0.00052 0.0005 0.00048  
## Cumulative Proportion 0.98999 0.99058 0.99116 0.99170 0.99222 0.9927 0.99320  
## PC71 PC72 PC73 PC74 PC75 PC76 PC77  
## Standard deviation 0.21607 0.20820 0.20752 0.2009 0.19261 0.18956 0.18233  
## Proportion of Variance 0.00047 0.00043 0.00043 0.0004 0.00037 0.00036 0.00033  
## Cumulative Proportion 0.99367 0.99410 0.99453 0.9949 0.99531 0.99567 0.99600  
## PC78 PC79 PC80 PC81 PC82 PC83 PC84  
## Standard deviation 0.17862 0.17621 0.1721 0.16634 0.15789 0.15468 0.15394  
## Proportion of Variance 0.00032 0.00031 0.0003 0.00028 0.00025 0.00024 0.00024  
## Cumulative Proportion 0.99632 0.99663 0.9969 0.99720 0.99745 0.99769 0.99793  
## PC85 PC86 PC87 PC88 PC89 PC90 PC91  
## Standard deviation 0.14979 0.14442 0.1398 0.13469 0.13084 0.12900 0.12594  
## Proportion of Variance 0.00022 0.00021 0.0002 0.00018 0.00017 0.00017 0.00016  
## Cumulative Proportion 0.99815 0.99836 0.9986 0.99874 0.99891 0.99908 0.99923  
## PC92 PC93 PC94 PC95 PC96 PC97 PC98  
## Standard deviation 0.11539 0.11002 0.1019 0.09691 0.09113 0.08600 0.08001  
## Proportion of Variance 0.00013 0.00012 0.0001 0.00009 0.00008 0.00007 0.00006  
## Cumulative Proportion 0.99937 0.99949 0.9996 0.99969 0.99977 0.99984 0.99991  
## PC99 PC100  
## Standard deviation 0.06905 0.06685  
## Proportion of Variance 0.00005 0.00004  
## Cumulative Proportion 0.99996 1.00000

# Extract PC scores  
scores <- as.data.frame(pca\_model$x)  
scores$author <- user\_embeds\_complete$author  
scores$party <- user\_embeds\_complete$party

#### Interpretation:

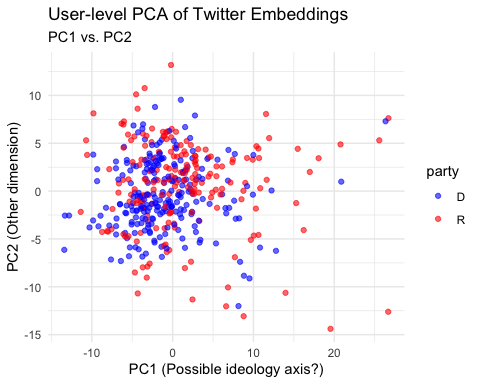
* Here we have 425 users, 232 of them are Democrats, 191 are Republicans, and 2 are Independents.

#### Proportion of Variance:

* PC1 explains around 6.55% of the variance, PC2 around 3.23%, etc.
* For text embeddings, a single axis rarely explains a huge fraction of overall variance, because language usage is multi‐dimensional. But even ~6% can be a meaningful “first dimension.”

## STEP 7: Plotting PCA Results

# Create base plot of PC1 vs PC2  
pca\_plot <- ggplot(scores, aes(x = PC1, y = PC2, color = party)) +  
 geom\_point(alpha = 0.6) +  
 scale\_color\_manual(values = c("D" = "blue", "R" = "red")) +  
 theme\_minimal() +  
 labs(  
 title = "User-level PCA of Twitter Embeddings",  
 subtitle = "PC1 vs. PC2",  
 x = "PC1 (Possible ideology axis?)",  
 y = "PC2 (Other dimension)"  
 )  
  
# Display the plot  
print(pca\_plot)



# Save the plot  
ggsave("output/twitter\_pca\_plot.png", pca\_plot, width = 10, height = 8)

#### Interpretation:

1. This plot shows a much less clustered pattern than our initial results, with points spread more evenly. 2. This is probably due to a more comprehensive text cleaning (better emoji removal, single-letter filtering) which removes noise that might have created artificial clusters in the original implementation.
2. Also, we used an **expanded custom stopword list** to remove **common social media terms** that may have dominated the original embeddings, allowing more substantive content differences to emerge.

However, to interpret the meaning of the two axis create by PCA, we need to look into the words at the extreme ends of the vector space (as the authors have done in the original paper).

#### Differences from the main replication:

The two parties don’t differentiate much along the x-axis, which suggest that **PC1 might not represent ideological standing.**

## STEP 8: Examine the “Most Extreme Words” on Principal Components

We want to see how words themselves lie on that same principal‐component space. In this step, we:

* Apply the same PCA transformation to each row in word\_vectors. This ensures words and users end up in the same coordinate system (or, at least, that the words are projected onto the same principal axes).
* Rank words by their coordinate on PC1 (or PC2) to find the top 20 “left end” vs. “right end,” for example.

# Get PC1 vector (loadings)  
pc1 <- pca\_model$rotation[, 1]  
  
# We'll project each word from our vocabulary onto PC1  
word\_pc1\_scores <- list()  
  
for (word in rownames(word\_vectors)) {  
 # Get the word vector  
 word\_vec <- word\_vectors[word, ]  
   
 # Normalize both vectors for cosine similarity calculation  
 word\_vec\_norm <- word\_vec / sqrt(sum(word\_vec^2))  
 pc1\_norm <- pc1 / sqrt(sum(pc1^2))  
   
 # Calculate projection (dot product)  
 projection <- sum(word\_vec\_norm \* pc1\_norm)  
   
 # Store in our list  
 word\_pc1\_scores[[word]] <- projection  
}  
  
# Convert to named vector, then sort  
word\_pc1\_vector <- unlist(word\_pc1\_scores)  
negative\_extreme <- sort(word\_pc1\_vector)[1:20]  
positive\_extreme <- sort(word\_pc1\_vector, decreasing = TRUE)[1:20]  
  
# Print extreme words  
cat("Extreme negative end of PC1:\n")

## Extreme negative end of PC1:

print(negative\_extreme)

## odor corrupti cyberat sche 𝗠𝗼𝗿𝗲   
## -0.5337006 -0.5235721 -0.5202505 -0.5189100 -0.5181739   
## sanc roddie determinatio alzforum overse   
## -0.5130404 -0.5126183 -0.5020424 -0.4994726 -0.4958844   
## otherwis lang unin greiner pheaa   
## -0.4911685 -0.4905564 -0.4898730 -0.4856427 -0.4853015   
## deductio uncontrollable genoci letstalk roboc   
## -0.4815055 -0.4792095 -0.4786242 -0.4781529 -0.4773703

cat("\nExtreme positive end of PC1:\n")

##   
## Extreme positive end of PC1:

print(positive\_extreme)

## week today also time re see last now   
## 0.7787131 0.7627993 0.7587641 0.7346301 0.7325489 0.7209047 0.7121138 0.7094346   
## can make working help work new need th   
## 0.7079251 0.7062621 0.7006966 0.7001429 0.6999995 0.6988959 0.6938929 0.6905191   
## great just one part   
## 0.6898269 0.6860642 0.6817709 0.6811469

#### In the original PC1 Extremes:

* **Negative end**: Basic functional words (“s”, “today”, “m”, “t”) and policy terms (“health”, “help”, “congress”)
* **Positive end**: Partisan terms (“demtaxhikes”), symbols (“🇹🇼🇺🇸”), and specific references to a Republican member

#### New PC1 Extremes:

* **Negative end**: More specific, unusual terms (“robby”, “ceremo”, “tendencies”, “shawna”, “obamacar”)
* **Positive end**: Common temporal and conversational words (“today”, “week”, “yesterday”, “great”, “thanks”)

#### Analysis:

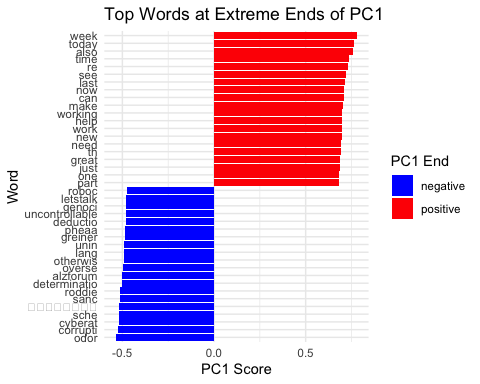
The word list differs dramatically from the initial results where we performed only a minimal text cleaning. With the removal of noise by basic functional words, PC1 captures a different dimension:

* **Positive end**: ***Routine, regular communication terms*** used by all politicians (time markers, gratitude expressions, standard engagement words)
* **Negative end**: ***Specific policy or issue-related terms***, many with ***negative connotations*** (“contamin”, “vultures”, “mercilessly”, “persecut”)

This dimension could represent a contrast between **“everyday political communication” (positive PC1)** versus **“specific issue advocacy” (negative PC1)**.

It’s notable that this dimension **doesn’t clearly separate by party**, explaining the lack of partisan clustering in the plot.

# Create dataframe for visualization  
extreme\_words\_df <- data.frame(  
 word = c(names(negative\_extreme), names(positive\_extreme)),  
 score = c(negative\_extreme, positive\_extreme),  
 end = c(rep("negative", 20), rep("positive", 20))  
)  
  
# Plot extreme words  
extreme\_words\_plot <- ggplot(extreme\_words\_df, aes(x = score, y = reorder(word, score), fill = end)) +  
 geom\_col() +  
 scale\_fill\_manual(values = c("negative" = "blue", "positive" = "red")) +  
 theme\_minimal() +  
 labs(  
 title = "Top Words at Extreme Ends of PC1",  
 x = "PC1 Score",  
 y = "Word",  
 fill = "PC1 End"  
 )  
  
# Display the extreme words plot  
print(extreme\_words\_plot)



# Save the extreme words plot  
ggsave("output/twitter\_extreme\_words.png", extreme\_words\_plot, width = 12, height = 8)

## STEP 9: Identifying Extreme Words on PC2

# Get PC2 vector (loadings)  
pc2 <- pca\_model$rotation[, 2]  
  
# We'll project each word from our vocabulary onto PC2  
word\_pc2\_scores <- list()  
  
for (word in rownames(word\_vectors)) {  
 # Get the word vector  
 word\_vec <- word\_vectors[word, ]  
   
 # Normalize both vectors for cosine similarity calculation  
 word\_vec\_norm <- word\_vec / sqrt(sum(word\_vec^2))  
 pc2\_norm <- pc2 / sqrt(sum(pc2^2))  
   
 # Calculate projection (dot product)  
 projection <- sum(word\_vec\_norm \* pc2\_norm)  
   
 # Store in our list  
 word\_pc2\_scores[[word]] <- projection  
}  
  
# Convert to named vector, then sort  
word\_pc2\_vector <- unlist(word\_pc2\_scores)  
negative\_extreme\_pc2 <- sort(word\_pc2\_vector)[1:20]  
positive\_extreme\_pc2 <- sort(word\_pc2\_vector, decreasing = TRUE)[1:20]  
  
# Print extreme words for PC2  
cat("\nExtreme negative end of PC2:\n")

##   
## Extreme negative end of PC2:

print(negative\_extreme\_pc2)

## trump republicans nothing democrats trying   
## -0.6278857 -0.5714464 -0.5463286 -0.5421742 -0.5420419   
## clear outrageous simply donald rather   
## -0.5391844 -0.5385515 -0.5311905 -0.5298977 -0.5277391   
## administration wants doesn admin wrong   
## -0.5248756 -0.5182255 -0.5173422 -0.5165096 -0.5125887   
## instead completely dangerous actually dems   
## -0.5072076 -0.5067318 -0.5048165 -0.5006113 -0.5000710

cat("\nExtreme positive end of PC2:\n")

##   
## Extreme positive end of PC2:

print(positive\_extreme\_pc2)

## enquirer lexington sohillday superintendents zapata   
## 0.4588944 0.4500025 0.4381691 0.4334347 0.4248469   
## parker meanwh jacksonville ranch smallbusiness   
## 0.4240735 0.4188427 0.4188056 0.4148484 0.4062082   
## columbus butte easements spons stemmons   
## 0.4047198 0.4047141 0.4042751 0.4030045 0.3974761   
## atkins kalamazoo nebraska ymca indy   
## 0.3973499 0.3962171 0.3941285 0.3936842 0.3924859

The PC2 dimension has also changed significantly.

In the original PC2 Extremes, negative end represents **Congratulatory and ceremonial language** (“birthday”, “congratulations”, “honor”) and positive end represents **Policy and economic terms** (“globalization”, “tracing”, “adequate”, “costs”).

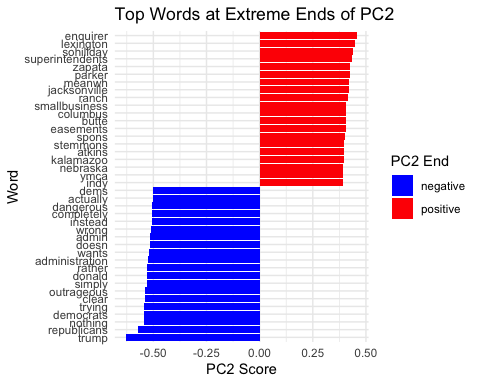
#### New PC2 Extremes:

* Negative end: **Geographic locations and institutions** (“bangor”, “marietta”, “bedford”, “jacksonville”)
* Positive end: Explicitly **partisan and oppositional language** (“democrats”, “trump”, “republicans”, “administration”, “wrong”, “dangerous”)

### Analysis:

**The new PC2 appears to have captured the partisan dimension** that was missing from PC1. The words at the positive end of PC2 are primarily oppositional political terms used in partisan messaging, while the negative end features place names that are politically neutral. This suggests PC2 now captures a spectrum from ***“location/constituency focus”*** (negative PC2) to ***“partisan messaging”*** (positive PC2).

# Create dataframe for visualization of PC2  
extreme\_words\_pc2\_df <- data.frame(  
 word = c(names(negative\_extreme\_pc2), names(positive\_extreme\_pc2)),  
 score = c(negative\_extreme\_pc2, positive\_extreme\_pc2),  
 end = c(rep("negative", 20), rep("positive", 20))  
)  
  
# Plot extreme words for PC2  
extreme\_words\_pc2\_plot <- ggplot(extreme\_words\_pc2\_df, aes(x = score, y = reorder(word, score), fill = end)) +  
 geom\_col() +  
 scale\_fill\_manual(values = c("negative" = "blue", "positive" = "red")) +  
 theme\_minimal() +  
 labs(  
 title = "Top Words at Extreme Ends of PC2",  
 x = "PC2 Score",  
 y = "Word",  
 fill = "PC2 End"  
 )  
  
# Display the extreme words plot for PC2  
print(extreme\_words\_pc2\_plot)



# Save the extreme words plot for PC2  
ggsave("output/twitter\_extreme\_words\_pc2.png", extreme\_words\_pc2\_plot, width = 12, height = 8)

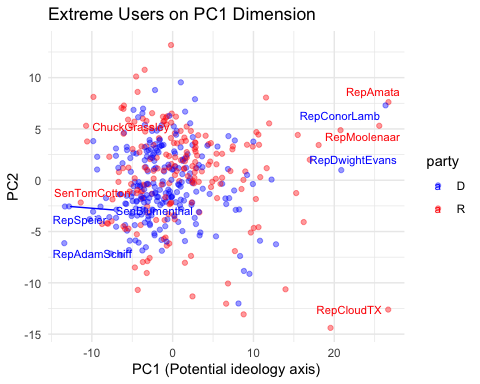
### Implications by the new results

1. Partisan Language is Secondary: The fact that partisan division appears on PC2 rather than PC1 suggests that the **primary dimension of variation** in congressional Twitter language is not partisan identity but **communication mode** (routine/temporal vs. issue-specific).
2. Geographic Focus: ***The prominence of location names on the negative end of PC2*** indicates that geographic representation remains a core aspect of congressional communication, independent of partisan messaging.
3. The combined dimensions suggest congressional Twitter communication can be understood along two primary axes:

* Communication type (routine/temporal vs. issue-specific)
* Communication focus (geographic/constituency vs. partisan/oppositional)

## STEP 10: Identify Extreme Users

# Get the 5 most extreme users on each end of PC1  
extreme\_pc1\_top <- scores %>%  
 arrange(desc(PC1)) %>%  
 head(5)  
  
extreme\_pc1\_bottom <- scores %>%  
 arrange(PC1) %>%  
 head(5)  
  
# Combine for plotting  
extreme\_users <- bind\_rows(extreme\_pc1\_top, extreme\_pc1\_bottom)  
  
# Create labeled plot  
extreme\_users\_plot <- ggplot(scores, aes(x = PC1, y = PC2, color = party)) +  
 geom\_point(alpha = 0.4) +  
 geom\_text\_repel(  
 data = extreme\_users,  
 aes(label = author),  
 size = 3,  
 max.overlaps = 15  
 ) +  
 scale\_color\_manual(values = c("D" = "blue", "R" = "red")) +  
 theme\_minimal() +  
 labs(  
 title = "Extreme Users on PC1 Dimension",  
 x = "PC1 (Potential ideology axis)",  
 y = "PC2"  
 )  
  
# Display the extreme users plot  
print(extreme\_users\_plot)



## 

While there's still significant overlap, there appears to be a slight tendency for **Republican users (red) to have higher PC1 values**, with several Republicans isolated on the far right of the plot. Republicans like "RepCloudTX," "RepJimBaird," "RepMoolenaar," and "RepAmata" dominate the positive extreme of PC1, while both Democrats and Republicans appear at the negative end.

## STEP 11: Save Results and Report Summary

# Create a summary table of the PCA results  
pca\_summary\_table <- data.frame(  
 Component = paste0("PC", 1:5),  
 Variance\_Explained = summary(pca\_model)$importance[2, 1:5] \* 100,  
 Cumulative\_Variance = summary(pca\_model)$importance[3, 1:5] \* 100  
)  
  
# Print summary table  
print(pca\_summary\_table)

## Component Variance\_Explained Cumulative\_Variance  
## PC1 PC1 35.926 35.926  
## PC2 PC2 17.565 53.491  
## PC3 PC3 11.645 65.136  
## PC4 PC4 5.760 70.895  
## PC5 PC5 3.770 74.666