STAT0030 ICA 3

```
## Load in the proper package
# install.packages('topicmodels')
library(topicmodels)
library(tidytext)
library(reshape2)
library(dplyr)
library(ggplot2)

## Prepare the data?
class(Misinfo.DTM)
dim(Misinfo.DTM)

###################### FITTING THE LDA MODEL ########################
## Number of topics to fit (hyperparameter)
chosen_no_topics <- 6
## Fit the LDA: insert DTM as x
fitted_lda <- topicmodels::LDA(x = Misinfo.DTM, ## DocTermMatrix
                               k = chosen_no_topics, ## Number of topics
                               control = list(seed = 30))


## Extract the fitted parameters : alpha, beta, gamma
print(fitted_lda)
fitted_lda@alpha
dim(fitted_lda@beta)
fitted_lda@gamma

## Find top 10 terms for each of the 6 topics
top10terms <- topicmodels::get_terms(fitted_lda,
                       k = 10)

## Get the probabilities
beta <- fitted_lda@beta
dim(beta) ## 6 rows (topics) and 137564 columns (vocabulary)

topicterms_data <- tidytext::tidy(fitted_lda, matrix = 'beta')

## Visualise the top 10 terms

## Get top 10 terms and probabilities for each topic
top_10_dataset <- topicterms_data %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

## Plot out the barplots
ggplot(top_10_dataset, aes(
  y = tidytext::reorder_within(term, beta, topic), ## reorder within so that every topic is ordered
  x = beta,
  fill = factor(topic))
  ) +
  geom_col(show.legend = FALSE,
```

```
          position = 'identity') + ##barplot
  facet_wrap(~ topic, scales = "free") +
  labs(x = "Log Probability", y = "Terms",
       title = 'Top ten words for each topic',
       subtitle = 'k = 6 topics') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) ##rotate the tick labels
```

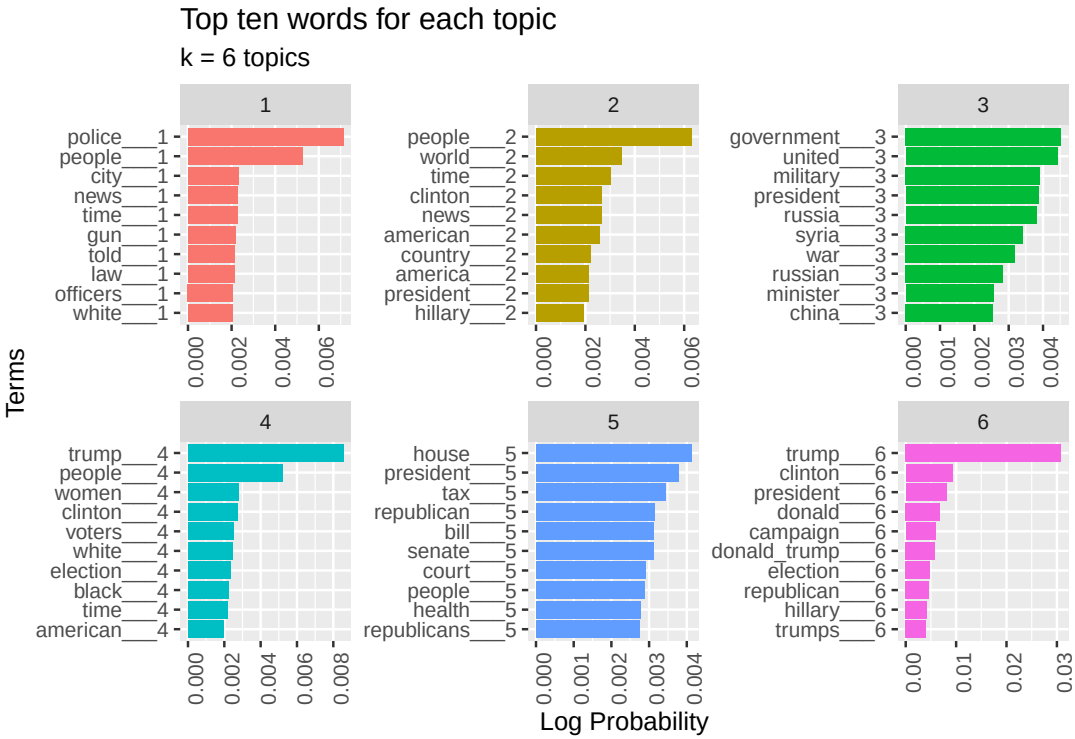

Figure 1: Topic-Word Distribution

0.0.0.1 1a) Fitting LDA model to the Document Term Matrix The α parameter is a hyperparameter of the dirichlet distribution for the prior **per-document topic distribution**, β similarly for the prior **per-topic word distribution**. γ represents the posterior **per-document topic distribution**.

```
## Split the dataset by sampling indices of the dataset
train_index <- sample(1:nrow(Misinfo.DTM), ## indices of sample
       size = round(0.3*nrow(Misinfo.DTM))) ## take the rounding off of 30% of sample size
train_set <- Misinfo.DTM[train_index,]
test_set <- Misinfo.DTM[-train_index,]

## Check
nrow(train_set); nrow(test_set)

## Convert type to simple_triplet_matrix (for perplexity later )
library(slam)
train_set <- as.simple_triplet_matrix(train_set)
test_set <- as.simple_triplet_matrix(test_set)

## Check
class(train_set); class(test_set)

## k values that we need to fit on
k_list <- seq(2,10,2)
lda_models_list <- list() ## placeholder for the fitted models

## For loop to run the LDA fitting for all k values
```

```r
for (k in k_list){
  lda_models_list[[as.character(k)]] <- LDA(train_set,
                                        k= k,
                                        control = list(seed = 30))
}

## For each fitted model, calculate the perplexity
# Placeholder vectors to store the perplexity data for train and test
train_perplexities <- rep(0, length(k_list))
test_perplexities <- rep(0, length(k_list))

## For loop to calculate perplexities for train and test data
## Store into placeholder vector
for (i in (1:length(lda_models_list))) {
  train_perplexities[i] <- perplexity(lda_models_list[[i]], train_set)
  test_perplexities[i] <- perplexity(lda_models_list[[i]], test_set)
}

## Check and compare perplexities
#train_perplexities ; test_perplexities
## Result:
# - the test perplexity is consistently higher than the train
# - this makes sense, since the models were fitted on the training data and not the test data
# - the lower perplexity the better the model fit

## Recall that we want a lower perplexity - which means that the MODEL GENERALISES BETTER
## Lower perplexity -- Better prediction on a sample

## Check
perplexity(lda_models_list[[1]], train_set)
perplexity(lda_models_list[[1]], test_set)

## Turn into dataframe for easier plottinng
perplex_df <- data.frame(
  k = k_list,
  train_perplexities = train_perplexities,
  test_perplexities = test_perplexities
)

## Plot out both perplexity vectors
train_plot <- ggplot(perplex_df, aes(x = k,
                       y = train_perplexities)) +
  geom_point() +
  geom_line(color = '#1f77b4') +
  labs(y = 'Perplexity',
         #x = 'Number of topics k',
         title = 'Perplexity for training data')

test_plot <- ggplot(perplex_df, aes(x = k, y = test_perplexities)) +
  geom_point() +
  geom_line(color = "#ff7f0e") +
  labs(y = 'Perplexity',
         #x = 'Number of topics k',
         title = 'Perplexity for testing data')

## Combine the plots
library(cowplot)
combined_plots <- plot_grid(train_plot,
                            test_plot,
                            ncol = 1,
                            align = 'v', ## vertical align
                            axis = 'tb') ## for top bottom aligning
```

```
print(combined_plots)
```

Figure 2: Perplexity Plots (train and test data)

0.0.0.2 1b) Perplexity of model In Fig 2, training perplexity goes down incrementally which makes sense since the model is being fitted to the training data and more topics means more flexibility due to higher number of parameters, hence better fit. Therefore we look at test data to determine optimal k. The perplexity drops to the lowest at k = 6 before increasing, hence k = 6 is optimal.

0.0.0.3 2a) Get estimated topic mixture for all documents, run kmeans on this matrix of topic mixture Here we want to run a clustering method on the document-topic mixture matrix, to see if we can identify patterns among documents through their topic distribution.

```
## Get the matrix that represents documents and topics
## Should be # documents by # topics or the transversed
dim(fitted_lda@gamma) ## 4865 documents by 6 topics
fitted_gamma_matrix <- fitted_lda@gamma

## Method 2 to get gamma: posterior
# posterior <- posterior(fitted_lda, Misinfo.DTM)
# dim(posterior$topics)
# posterior_gamma <- posterior$topics

## Kmeans clustering on the matrix -- cluster the documents based on their similarity in terms of topics
## Fit k = 2 clusters
fitted_kmeans_2 <- kmeans(x = (fitted_gamma_matrix),
                          centers = 2,
                          nstart = 10)

## Method 2
# fitted_kmeans_2.2 <- kmeans(x = posterior_gamma,
#                             centers = 2,
#                             nstart = 10)


## Compare kmeans results to GROUNDTRUTH labels
contin_table <- table(MisinformationLabel$label, fitted_kmeans_2$cluster)
contin_table

## Compare via adjusted Rand Index from mclust library
# install.packages('mclust')
```

```r
library(mclust)
library(tidyverse)

## Match true/false in groundtruth label to 1 and 2
MisinformationLabel$label_1_2 <- ifelse(MisinformationLabel$label == 'True',
                                        1, 2)

## Rand Index: 1 means similar, 0 means random, less than 0 means each cluster of one clustering is evenly distribu
adj_rand_index <- adjustedRandIndex(MisinformationLabel$label_1_2,
                                    fitted_kmeans_2$cluster)
## Make dataframe for easier plotting
clusters_df <- data.frame(kmeans_cluster = as.factor(fitted_kmeans_2$cluster),
                          true_cluster = MisinformationLabel$label)

## Combine gamma matrix and k-means cluster assignments
clustered_gamma <- data.frame(fitted_gamma_matrix, cluster = as.factor(fitted_kmeans_2$cluster))

## Calculate means of gamma values for each cluster
cluster_means <- clustered_gamma %>%
  group_by(cluster) %>%
  summarize(across(everything(), mean))


## Add cluster labels to the gamma matrix
gamma_clusters <- cbind(fitted_gamma_matrix, Cluster = as.factor(fitted_kmeans_2$cluster))

## Merge with Misinformation data to get true labels
merged_data <- merge(MisinformationLabel$label,
                     gamma_clusters,
                     by = "row.names",
                     all.x = TRUE)
colnames(merged_data)[1] <- "DocID"
colnames(merged_data)[3:8] <- paste0('topic_', 1:6)

## Get the proportions of each topic for each cluster
cluster_props <- merged_data %>%
  group_by(Cluster) %>%
  summarize(Topic1 = mean(topic_1),
            Topic2 = mean(topic_2),
            Topic3 = mean(topic_3),
            Topic4 = mean(topic_4),
            Topic5 = mean(topic_5),
            Topic6 = mean(topic_6))

## Reshape the data for plotting
cluster_props_longer <- pivot_longer(cluster_props, cols = starts_with("Topic"),
                                     names_to = "Topic", values_to = "Proportion")
# Plot the results
ggplot(cluster_props_longer, aes(x = as.factor(Cluster),
                                 y = Proportion,
                                 fill = Topic)) + ## Fill by topic to see topic mixture
  geom_bar(stat = "identity",
           position = "fill") +
  scale_fill_brewer(palette = "Paired") +
  labs(x = "Cluster", y = "Proportion", fill = "Topic") +
  theme_classic() +
  labs(title = "Proportions of topics for each cluster")
```

The agreement between the clustering and the ground-truth labels can be represented by the Adjusted Rand Index score of 0, which is very close to 0 and suggests no relationship/correlation between the 2 categories.

From Fig 3, it is very clear that documents in cluster 1, on average, are mostly on topic 3, while those in cluster 2 have a more even spread of topics except a small proportion of topic 3.

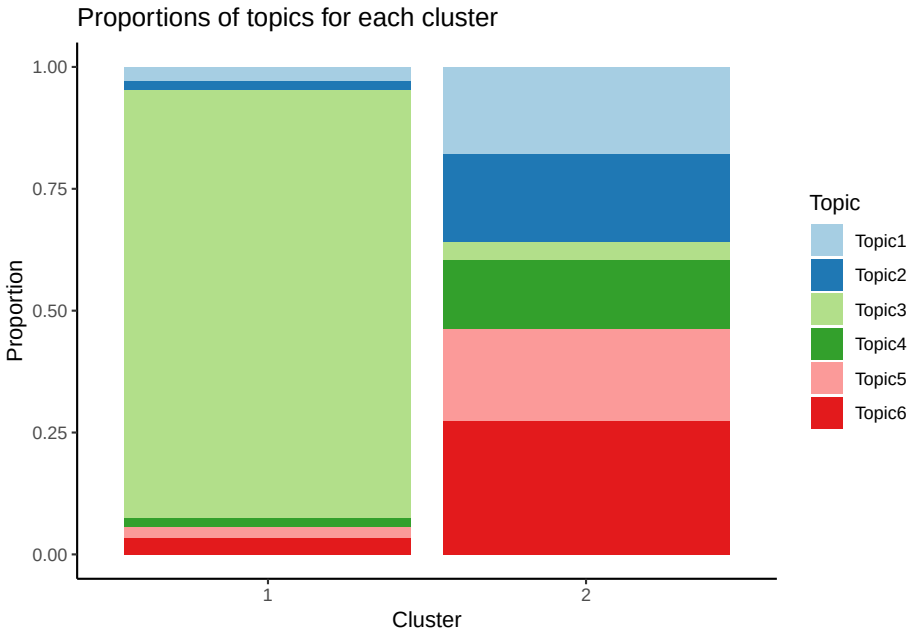

Figure 3: Clusters in terms of Topics

One possible way is to increase the number of topics during the LDA fitting process. The current dimensionality of the gamma now (6) may not be granular enough to identify the documents with misinformation, i.e. the 'true' number of latent topics may be higher, and thus we need to fit more topics to get more information before we can cluster. Alternatively, we possibly need to identify more clusters as we might identify subclusters of both misinformed and non-misinformed groups.

```
## options for number of clusters
number_clusters_list <- (2:10)

## Iteratively fit kmeans, save the total within sum of sqaures
wss <- sapply(number_clusters_list,
              function(k) {kmeans(x = fitted_gamma_matrix,
                                  centers = k,
                                  nstart = 10)}$tot.withinss)


## Plot the wss against number of clusters

## Save as df for easier plotting
wss_df <- data.frame(number_clusters = number_clusters_list,
                     wss = wss)

## Save the Elbow plot as an object for later
wss_elbow_plot <- ggplot(wss_df, aes(x = number_clusters, y = wss)) +
  geom_point() +
  geom_line(color = "#ff7f0e") +
  scale_x_continuous(breaks = number_clusters_list) +
  labs(x = "Number of clusters",
       y = "WCSS")

## Placeholder vector for silhouette coeffs for
avg_silh_coeffs <- numeric(length(number_clusters_list))

## Loop through each value of number of clusters to find the silhouette score
library(cluster)
for (k in number_clusters_list){
  fitted_kmeans <- kmeans(x = fitted_gamma_matrix,
                          centers = k,
```

```
                          nstart = 10)
  avg_silh_coeffs[k-1] <- mean(silhouette(fitted_kmeans$cluster, dist(fitted_gamma_matrix))[,3]## Take the third c
                               )
}

## Check
avg_silh_coeffs

## Save as df for easier plotting
avg_silh_df <- data.frame(number_clusters = number_clusters_list,
                      average_silhouette_coeff = avg_silh_coeffs)

## Save the Silhouette plot as an object for later
avg_silh_plot <- ggplot(avg_silh_df, aes(x = number_clusters, y = average_silhouette_coeff)) +
  geom_point() +
  geom_line(color = '#1f77b4') +
  scale_x_continuous(breaks = number_clusters_list) +
  labs(x = "Number of clusters",
       y = "Average Silhouette")

## Combined plot to save space
combined_plots <- plot_grid(

  wss_elbow_plot + theme(axis.text.x = element_blank(),                                    axis.ticks.x = element_bl
    labs(title = "Elbow Method"),

  avg_silh_plot + labs(title = "Silhouette Method"),
  ncol = 1,
  align = 'v', ## vertical align
  axis = 'tb') ## for top bottom aligning

## Print combined plots
print(combined_plots)
```

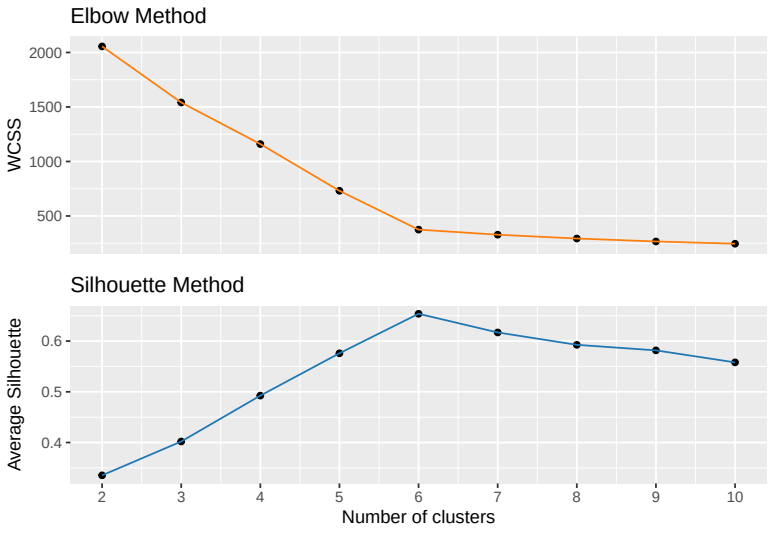

Figure 4: Selecting Optimal k

0.0.0.4 2b) Using Elbow and BIC method to determine optimal number of clusters (hyperparameter tuning) From the elbow method in Fig 4, we choose 6 as it is where the rate of decrease of WCSS (within-cluster sum of squares) starts to slow down. From the Silhouette method we choose 6 as it is the highest point of the average Silhouette coefficient (across all data points), since this indicates good clustering (points within a cluster have the highest relative similarity compared to points without, across different number of clusters)

```
## refit the model using the optimal number of clusters
optimal_fitted_kmeans <- kmeans(x = fitted_gamma_matrix, centers = 6)
```

```r
## Compare kmeans results to GROUNDTRUTH labels
contin_table2 <- table(MisinformationLabel$label, optimal_fitted_kmeans$cluster)
contin_table2

## Compare via adjusted Rand Index from mclust library
## Rand Index: 1 means similar, 0 means random, less than 0 means each cluster of one clustering is evenly distrib
adj_rand_index2 <- adjustedRandIndex(MisinformationLabel$label,
                                     optimal_fitted_kmeans$cluster)

## Check
adj_rand_index2
```

The Adjusted Rand Index is essentially 0 ($-2.7706831 \times 10^{-4}$), indicating poor agreement. This could be due to the fact that the clustering acts on the topic distribution of the articles, whereas there is no correlation between topic choices and whether the article spreads misinformation or not.

```r
## Import randomForest library for randomforest algo
library(randomForest)

## First create the 10 cols of noise : use a matrix
size_noise_matrix <- nrow(fitted_gamma_matrix)*10 ## same number of rows (number of documents) x 10 columns

## Create matrix of noise
noise_matrix<- matrix(rnorm(size_noise_matrix), ncol = 10)

## Make new df with gamma, then append the noise matrix
gamma_df <- data.frame(fitted_gamma_matrix)
Misinfo_noisy <- cbind(gamma_df, noise_matrix)


## Append the misinformation label to dataset
Misinfo_noisy$Label <- MisinformationLabel$label
## Turn into 1 and 0
Misinfo_noisy$Label <- as.integer(Misinfo_noisy$Label == 'True')
## Convert to factor
Misinfo_noisy$Label <- as.factor(Misinfo_noisy$Label)

## Rename the noise columns
colnames(Misinfo_noisy)[7:16] <- paste0("noise", 1:10)

## Fit the random forest model on the new dataset
fitted_rforest <- randomForest(
  Misinfo_noisy$Label ~ .,
  data = Misinfo_noisy)

## Check
print(fitted_rforest)
OOB <- fitted_rforest$err.rate[,1]

## Train test split
train_size <- 2/3 * nrow(Misinfo_noisy)
train_points <- sample(1:nrow(Misinfo_noisy),
                       train_size, replace = F)

## Fit noisy dataset on kmeans with 2 clusters
kmeans_noisy <- kmeans(x = Misinfo_noisy,
       centers = 2,
       nstart = 10)
## Get contingency table
table(Misinfo_noisy$Label, kmeans_noisy$cluster)
```

```
## Compare using adj rand index again
adjustedRandIndex(Misinfo_noisy$Label, kmeans_noisy$cluster)

adjustedRandIndex(fitted_rforest$predicted, Misinfo_noisy$Label)
```

0.0.0.5 2c) Generate new dataset with noise, use randomForest function (Lab 7) to predict the groundtruth label from this new dataset (use out of bag error evaluation)

The OOB estimate of error rate is 47.46%, which is almost half and suggests that the model is worse than a flip-of-coin chance.

We added the ground truth label for random forest (ensemble method) in order to try to use the data (topic distribution) to predict the outcome (misinformation ground-truth), along with some noise. The fact that it is a supervised method (it 'knows' which datapoints were 'true' and 'false') and is an ensemble method means that it might be more robust to noise, compared to kmeans which is unsupervised and is non-ensemble, meaning that it might fit more to the noise by chance when trying to cluster datapoints based on geometric similarity on the predictors.

However, as we have seen, both methods had poor agreement with the ground-truth.

```
## Import libraries
library(gtools)

number_documents <- nrow(Misinfo.DTM)

## Define the function that simulates document generation
simulate_corpus_function <- function(M = number_documents, N = 10, lda = fitted_lda){

  ## 1: Create placeholder matrix to fill up
  final_word_matrix_MbyN <- matrix(nrow = M,ncol = N)

  ## 1.1: Extract parameters from the fitted LDA object
  fitted_alpha <- lda@alpha ## for dirichlet distribution
  fitted_beta <- lda@beta ## for topic-word distribution


  ## 2: Loop through each document (each row of the final matrix)
  for (doci in (1:M)){

    ## For each document ##

    ## First: get topic distribution by sampling from dirichlet
    ## This will return a probability vector of length 6, each for one topic
    topic_mixture <- rdirichlet(1, alpha = rep(fitted_alpha, chosen_no_topics))

    ## 3: Loop through each word (for this document)
    for (wordj in (1:N)) {

    ## Second: Choose topic by sampling from generated categorical distribution (generated from the rdirichlet fun
    topic_selected <- sample((1:chosen_no_topics),
                         size = 1,
                         prob = topic_mixture ## Generated from step 1
                         )

    ## Third: Get the topic-word distribution (for particular topic)
    ## Use the beta distribution for this
    selected_topic_word_dist <- exp(beta[topic_selected, ]) ## Apply exp to un-log the log probabilites, and subse

    ## Finally: Choose word by sampling from generated categorical distribution
    word_selected_index <- sample((1:length(selected_topic_word_dist)),
                         size = 1,
                         prob = selected_topic_word_dist)

    ## Get actual word from vocab
```

```
    word_selected <- colnames(Misinfo.DTM)[word_selected_index]

    ## Append the word
    final_word_matrix_MbyN[doci, wordj] <- word_selected
    }
  }
return(final_word_matrix_MbyN)
}

## Test out function
simulated_corpus <- simulate_corpus_function()

## Check
as.data.frame(simulated_corpus)
dim(simulated_corpus)

## SIMULATED DATA : create dataframe
simulated_df <- as.data.frame(simulated_corpus)
colnames(simulated_df) <- paste0("word", 1:10)
simulated_df

## Add doc id and concat all the words to text
simulated_df$text <- apply(simulated_df, 1, paste, collapse = " ")
simulated_df$ID <- MisinformationData$ID

## Subset for only ID and text
simulated_df <- simulated_df[c('ID', 'text')]

## Save original vocab
vocab <- colnames(Misinfo.DTM)

## Convert simulated data to DTM?
library(textmineR)
simulated_DTM <-CreateDtm(simulated_df$text,
          doc_names = simulated_df$ID)

## Refit LDA to get topic distribution
simulated_lda_fit <- LDA(simulated_DTM,
                         k = 6,
                         control = list(seed = 30))



# Get topic distribution for each document in the original dataset
fitted_topics <- tidy(fitted_lda, matrix = "gamma")
# Get topic distribution for each document in the simulated dataset
simulated_topics <- tidy(simulated_lda_fit, matrix = "gamma")

## Plot!

## Loop over each topic for both lda extracted gammas
par(mfrow = c(2,3))
for (i in 1:chosen_no_topics) {
  ## Filter for topic
  fitted_topic_dist <- fitted_topics %>% filter(topic == i) %>% pull(gamma)
  simulated_topic_dist <- simulated_topics %>% filter(topic == i) %>% pull(gamma)

  ## Create QQ plot
  qqplot(fitted_topic_dist, simulated_topic_dist,
         xlab = "Original Topic Distribution",
         ylab = "Simulated Topic Distribution",
         main = paste0("Topic ", i, " Distribution QQ-Plot"))
  abline(a = 0, b = 1, col = 'red', lty = 2)
```

```
}
```

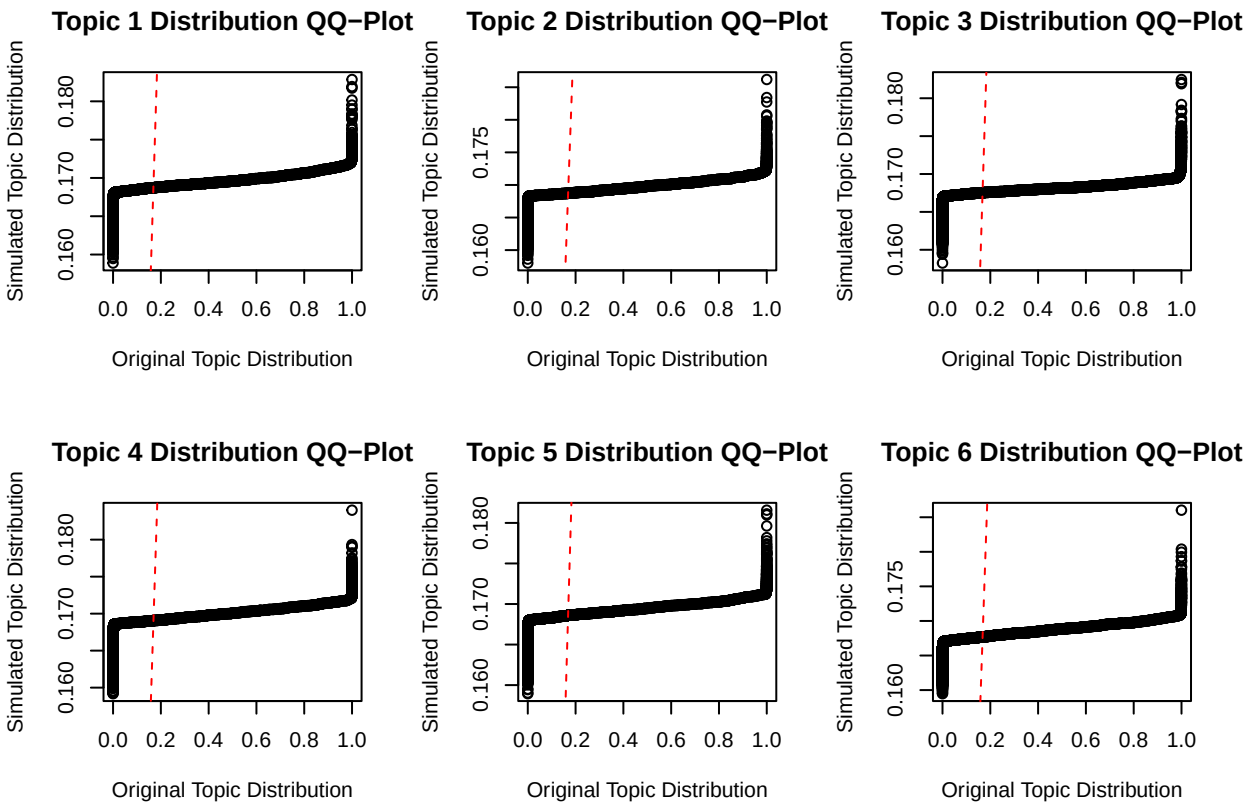

Figure 5: QQ-plots between real and simulated corpus

0.0.0.6 3a) Comparing simulated and actual corpus To compare the topic distributions between actual and simulated, I plotted the qqplot of the gamma values (topic distribution) for each of the 6 topics in Fig 5. The qqplots for all 6 distributions are similarly 'off' the line of equivalence (red dotted line), suggested that the distribution of topic-proportions are very different. The most likely reason was that the topic proportions were gathered via a refitting of the simulated corpus, and thus the underlying model (and the 6 topics) may be fundamentally different. In addition, we are only simulating 10 words per document (when the true length of each document is much higher and variable). Also, the fitness of the topic-word distribution could be poor.

0.0.0.7 3c) Model selection to decide on the choice of the number of topics We could use perplexity scores for cross validation to decide the best number of topic to fit the LDA model. Alternatively, we could use AIC or BIC methods (related to model likelihood, thus suitable for LDA models) that both penalise for the number of parameters used (and hence overfitting or using unnecessary predictors). We would select the model with the number of topics that give the lowest AIC or BIC values.

```
## Subset first 50
first_50 <- Misinfo.DTM[1:50,]

## Check
dim(first_50) ## 50 docs by vocab size

## Create placeholder to collect alpha
number_bootstrap_iter <- 100
alpha_bootstrapped <- numeric(number_bootstrap_iter)

## Run the bootstrap analysis
set.seed(42)
for (i in (1:number_bootstrap_iter)){
  ## Perform sampling
  ## Sample indices
```

```
  sample_indices <- sample(1:nrow(first_50),
                           replace = T,
                           size = )
  ## Use indices to subset
  sample_data <- first_50[sample_indices, ]

  ## Fit the model
  sample_LDA <- LDA(sample_data,
                    k = 6,
                    control = list(seed = 30))

  ## Extract alpha
  alpha_bootstrapped[i] <- sample_LDA@alpha
}

## check
alpha_bootstrapped

## Compute 95% confidence interval for alpha
ci_alpha <- quantile(alpha_bootstrapped, c(0.025, 0.975))

# Print the confidence interval
cat("95% CI for alpha:", ci_alpha[1], "-", ci_alpha[2], "\n")
```

0.0.0.8 3d) Bootstrapping the first 50 rows (first 50 documents) to get confidence interval for alpha parameter

The original estimate: 0.035 doesn't lie within the bootstrapped confidence interval (0.0150886, 0.0266761), where the confidence level is 95%. This is likely because we are only bootstrapping for the first 50 datasets, which is a % of the entire dataset, hence a poor representation of the entire corpus.