

Self-marketing Approaches

An Analysis of Online Dating

Li Liu, Abhishek Pandit, Adam Shelton

12/11/2019

Contents

Contributions	1
Introduction	2
Literature Review	2
Empirical Strategy	3
Analysis & Results	5
Conclusion	31
Appendices	32
References	65

Contributions

Liu	Pandit	Shelton
KMeans	Item 1	EDA
Structural Topic Modeling	Item 2	AGNES
Volunteer for the App Demo	Item 3	DBSCAN

Introduction

“So tell me about yourself!” This seemingly straightforward question in day-to-day interactions is usually met with silence and hesitation. That can no longer be the case for the 1.67 trillion online dating industry, which has grown exponentially in popularity over the last decade. The dating apps, such as OkCupid and Coffee Meets Bagel, are designed to help the singles ‘get to know’ other people for short or long-term romantic relationships. In order to be popular and memorable, users usually have to write a short introduction to advertise themselves. Such activity could be regarded as self-marketing. As the users of dating apps come from diverse backgrounds, we are interested in how users from distinct backgrounds take different approaches to make themselves more memorable. Moreover, we design the framework of scoring users’ self-introduction and algorithm for providing writing tips (such as words for being memorable). Although our project is still preliminary, it has gained a lot of interest among our friends who struggle to find a date online. Also, our methods and analysis have the potential to be adopted by the dating website to improve the users’ experience and better achieve their mission as matchmakers.

Literature Review

Self-concept and self-representation have long served as grounds of debate in cognitive and positive psychology (Bruning, Schraw, and Ronning 1999) as well as social anthropology (Goffman 1975). The recent spread of social networking and its specific affordances have allowed individuals to build different online ‘selves’ (Papacharissi 2010). One such critical scenario may be that of mate selection, which several economists and sociologists have likened this to ‘marriage marketplace’ (Hitsch, Hortaçsu, and Ariely 2010). Several online dating service providers in developed countries may facilitate the expansion of potential mates beyond the limits of even extended offline social networks Cacioppo et al. (2013) assert that as many as one in three marriages in the United States is facilitated through these portals. Heino, Ellison, and Gibbs (2010) argue that these avenues further entrench the economic dimension through an acute, implicit awareness of ‘relationshopping’. Herein, potential partners are reduced to entries in a catalog to be scrolled through. In this sense, they suggest an emerging conscientiousness of ‘marketing’, with the product being themselves, and the potential mate assuming the role of a buyer (*ibid*). This perception thus links the private worlds of romantic intimacy with those of mass consumption and broader perceived appeal to the opposite sex.

Potentially, we will also use some marketing theories to understand our findings. Selling themselves and finding a mate on OkCupid is not very different from selling a product on eBay. Economists have been

interested in the matching problem of demand and supply, such as Hitsch, Hortaçsu, and Ariely (2010). Since we do not have data on users' interactions, we will focus primarily on understanding how people brand themselves to stand out in a crowd. For example, brand awareness is a key metric in marketing to quantify the degree to which people recall or recognize a brand. A high level of brand awareness helps a product stand out and get chosen when consumers face many alternatives.

This could be applied to understand online dating. Let us imagine your future mate uses the filter to narrow down the consideration sets. He/She might still face many similar choices with high matching scores to choose from. If you want to stand out from the pool, you must make yourself memorable by highlighting the uniqueness. Thus, one possible idea in this project is to explore and understand how users could increase their brand awareness and differentiate themselves in their segments

Empirical Strategy

This project will leverage a publicly open and anonymized user profile data from OkCupid. In particular, we will use several key demographic variables and the self-introduction text. Also, we currently focus on only male users as generally male users face more competition among peers.

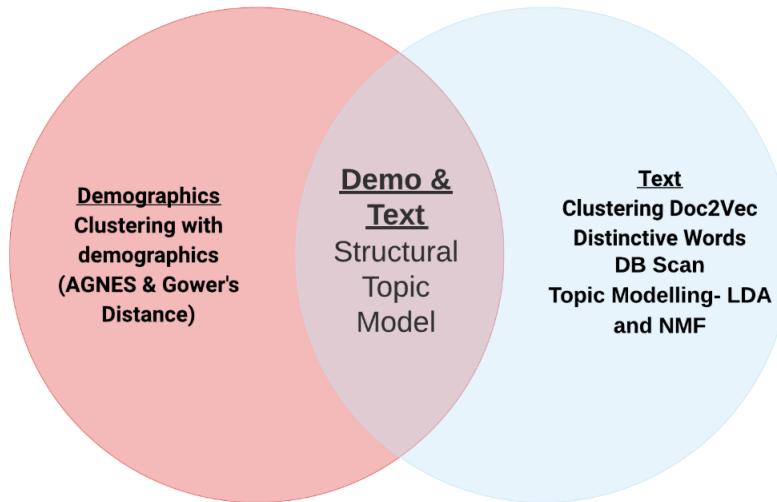


Figure 1: Venn Diagram of Methods

Firstly, we use different cluster analysis to group the users. We would expect users with similar backgrounds to introduce themselves in a similar manner. Since the demographic variables are mostly categorical with many levels and we use the Gowler distance, including them all would be computationally impossible. Instead, we rely on the literature on selecting the four most interesting variables and also apply other

Table 1: Continuous Variables

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
age	0	1	32.02	9.09	18.0	26.00	30.00	36.00	69
height	0	1	70.51	3.03	3.0	69.00	70.00	72.00	95
long_words	0	1	11.33	13.28	0.0	3.00	8.00	15.00	446
flesch	0	1	7.30	4.75	-3.6	4.86	6.73	8.96	268
profile_length	0	1	117.84	122.21	1.0	43.00	85.00	153.00	2973
prop_longwords	0	1	0.10	0.08	0.0	0.06	0.09	0.12	3

Table 2: Other Variables

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
clean_text	0	1	1	16952	0	18790	0
essay9	0	1	1	10849	0	18125	0
edu	0	1	7	21	0	3	0
fit	0	1	3	7	0	3	0
race_ethnicity	0	1	5	8	0	6	0
height_group	0	1	5	9	0	2	0

techniques to generate new features from the self-introductions.

Secondly, we use topic modeling (LDA) to visualize the latent topics behind the text. We also compare the differences in topic proportions for different groups and clusters of users.

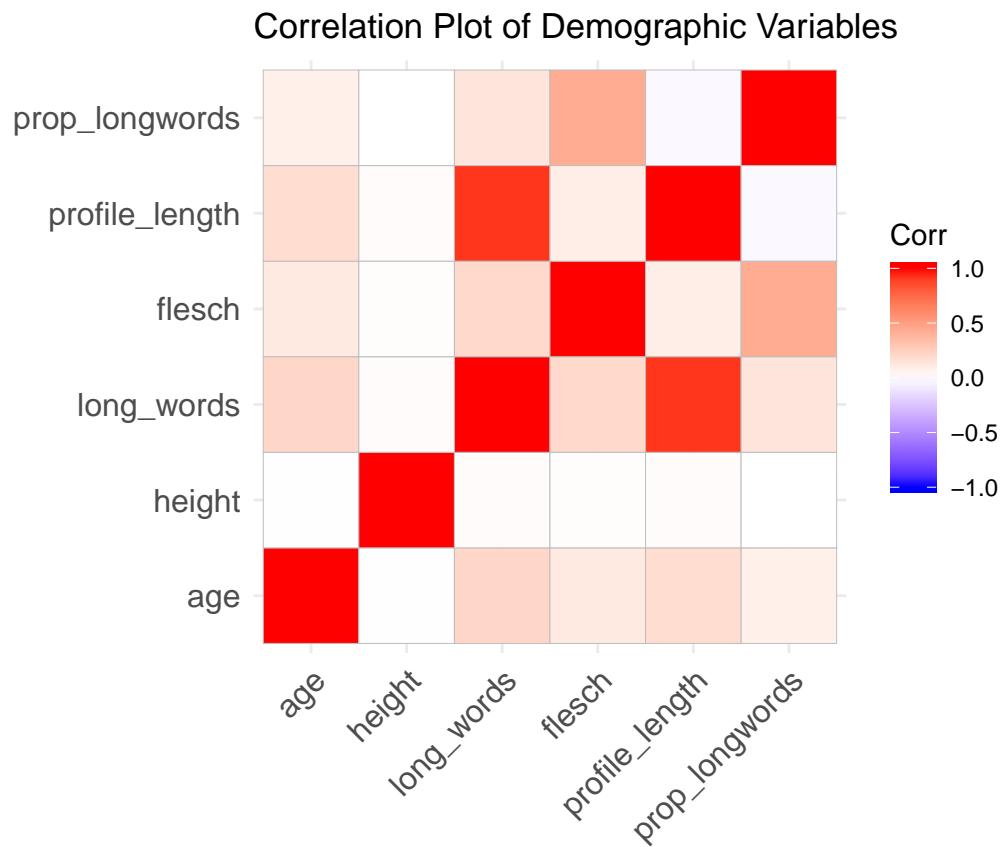
Thirdly, we use the structural topic modeling to estimate the impact of the demographic variables and humor measure (DBScan on Doc2Vec vectors) on the topic proportions.

Lastly, we discuss the potential use of our analysis and also acknowledge the challenges and limitations we face at this stage.

Analysis & Results

Exploratory Data Analysis

Descriptive Statistics



The data we used was approximately 60,000 anonymous OkCupid profiles from 2012 that were gathered with consent from users in the San Francisco area (Kim and Escobedo-Land 2015). This data was downloaded from the GitHub page for Kim and Escobedo-Land (2015), https://github.com/rudeboybert/JSE_OkCupid. The data contains demographic attributes of users that were submitted to their profile, including variables like age, height, race, and education, in addition to a selection of ten short essays that users have written in response to different prompts to display on their profiles. We subsetted this data to 18817 profiles of men, and generated additional features for the numbers and proportions of long words and Flesch–Kincaid readability scores of the main profile essay. The majority of male users in our sample are white, fit, not-short, and have more than a high school education. The mean reported age is 32 and the mean reported height is 70.5 inches (approximately 5 foot 9 inches).

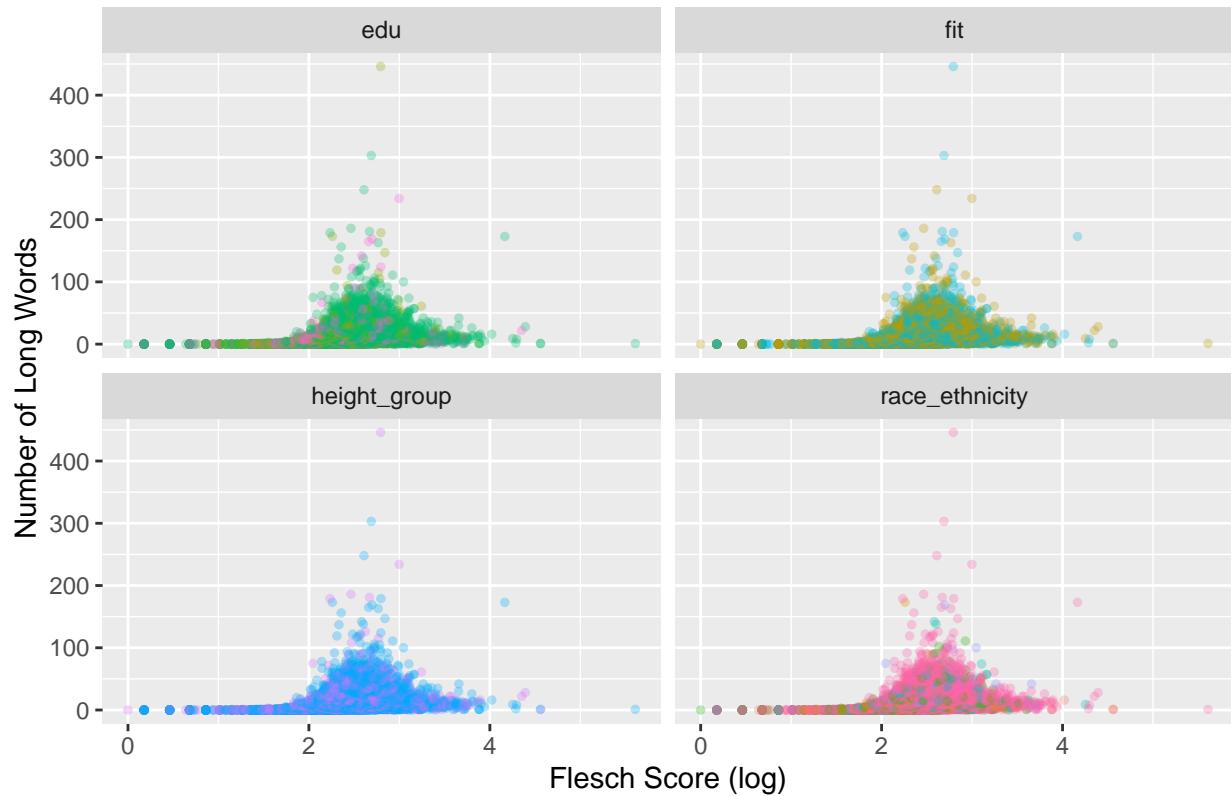
The majority of the variables included in the demographic data are independent, but some weaker correlations

do exist. As expected there are positive correlations between all the features generated from the essay text. Age is also positively correlated with our text-generated features, perhaps suggesting that older people are more educated and write with more complexity. While Flesch scores and the amount of long words are correlated, there do not appear to be any demographic interactions with that relationship.

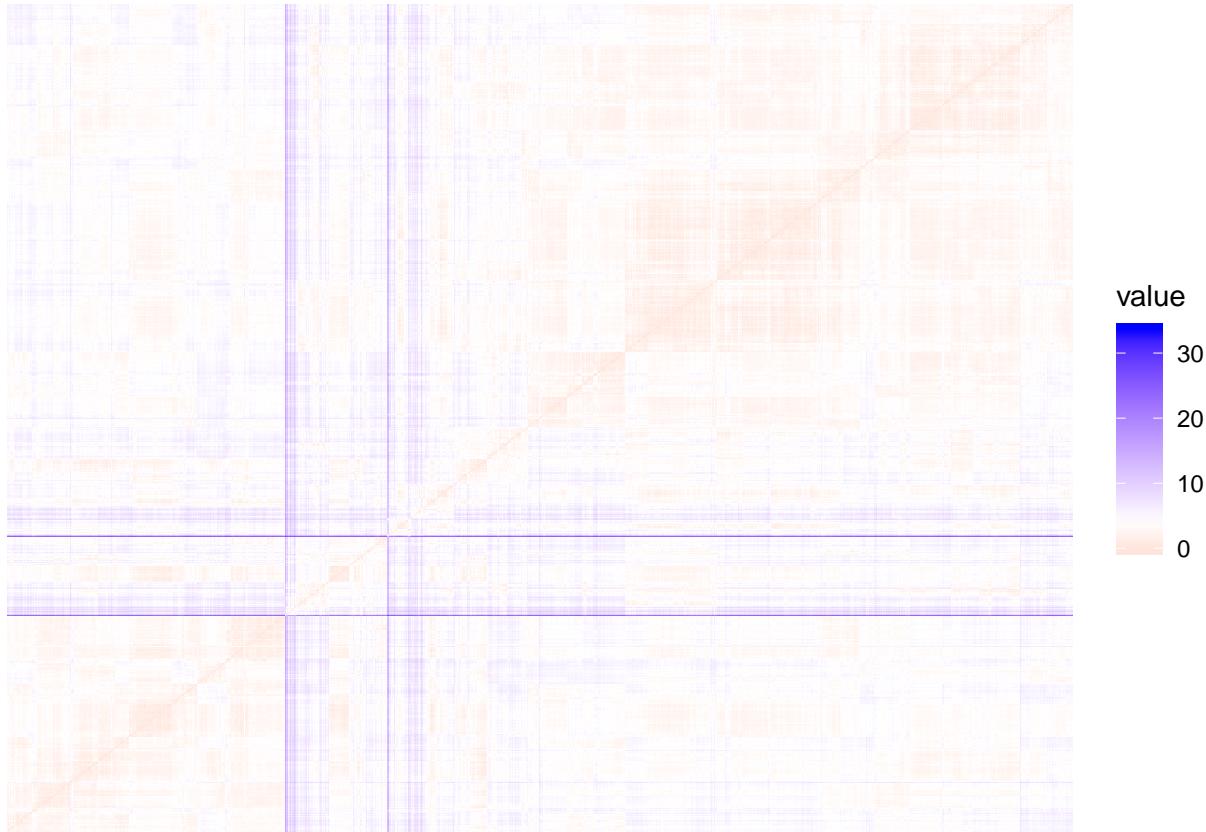
Categorical Variable Distributions



Flesch score vs. Long words by Variable



Clusterability



The demographic data is highly clusterable with a Hopkins Statistic of 0.929 on a random subset of 2,000 observations. The ordered dissimilarity image not only provides visual evidence of this clusterability, but also gives some indication of how many clusters may be in this data. Unfortunately, clusterability could not be determined for the whole dataset due to performance limitations. However, 2,000 observations should be sufficient for determining clusterability.

Clustering of Demographic Data

K-means

Kmeans is a type of partitioning clustering algorithm to maximize intra-cluster homogeneity and maximize inter-cluster heterogeneity. We use four demographic variables: ‘fit’, ‘education’, ‘height_group’, ‘race_ethnicity’ as previous research shows evidence these four variables are associated with users’ different behavior in online self-introductions. We use Gower’s distance as these variables are categorical. We start with 3 clusters and plot with the `fviz_cluster` function.

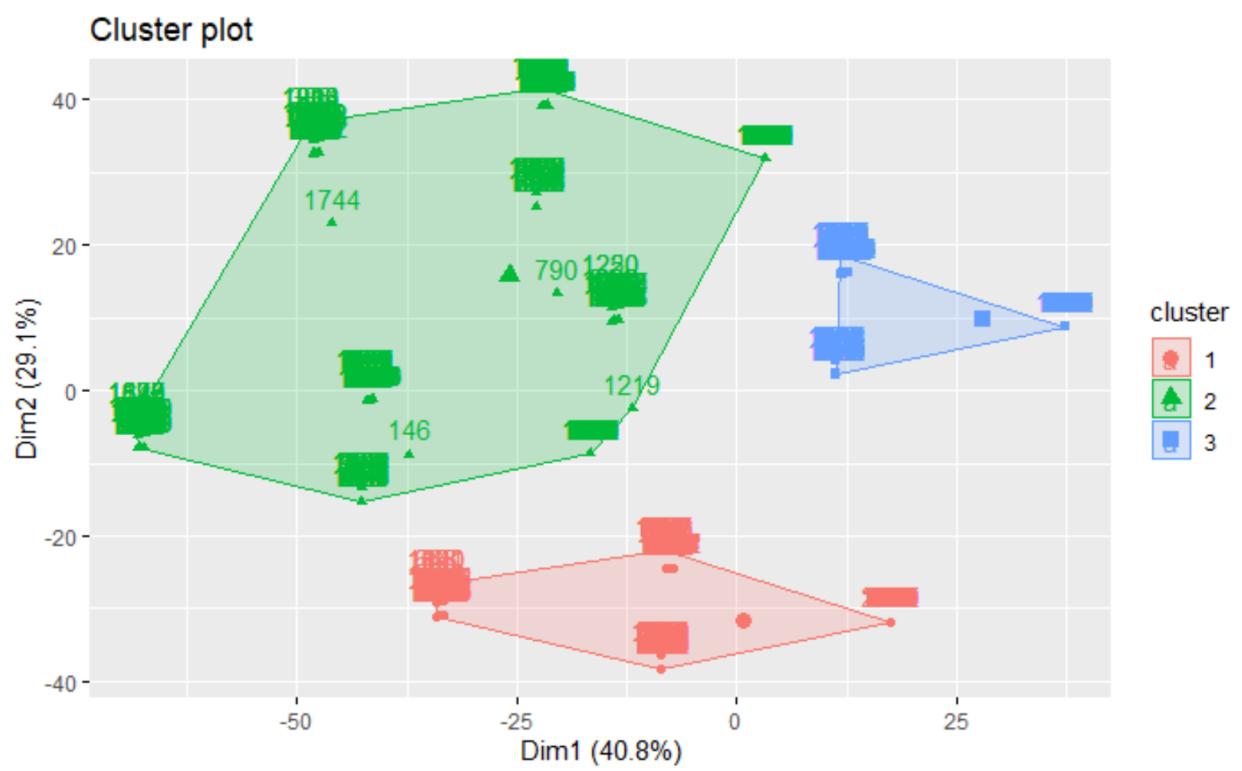


Figure 2: Kmeans with 3 clusters

Then we find the optimal number of clusters by comparing the metric of average silhouette width. However, the metric does not converge and keeps growing with more clusters. As a result, kmeans is not the best for finding the optimal demographic clusters.

AGNES

Agglomerative Nesting was used to cluster the demographic data with a bottom-up approach to contrast the K-means clustering. As an AGNES model would not complete on the full data-set, a random subset of 5,000 observations was used instead. This AGNES model used a Gower's dissimilarity matrix of the data to aid in the clustering of categorical variables, since most of the demographic variables are categorical. Using the `NbClust` package in R, we determined that the optimal number of clusters was two, when using the `ward.D2` method to match our AGNES model. The model gives us two well-defined clusters along the first two principal components. The defining factor between these two clusters, majorly appears to be height, which, per the Wilcox Test, has a statistically significant difference between the means of the two clusters by about five inches. These AGNES clusters can be used to reduce demensionality among a user's demographics so it can be more effectively used in further analyses. The dendrogram and cluster plots were generated using the `factoextra` package in R.

Text Analysis

Word2Vec

Topic Modeling

A topic is defined as a mixture of words where each word has a probability of generating from a topic. The example would be words such as 'books', 'college', 'MOOC' could come from the topic of study. A document is a mixture of topics, where a single document can have covered multiple topics. The example would be that a user talks about topics related to career, study, hobbies, and religion in the self-introduction.

We start the topic modeling by using the nonnegative matrix factorization (NMF). It is sometimes preferred to Latent Dirichlet allocation (LDA) as it is a deterministic algorithm and assumes that topic probabilities have to be fixed per self-introduction. Also, it is easy to implement with the `gensim` package in Python as we have prior experience with it. We train the NMF topic model with 25 latent topics and then further visualize the proportions of users who use each of the 25 topics most. We also compare the proportions across different levels in the 4 chosen demographic variables.

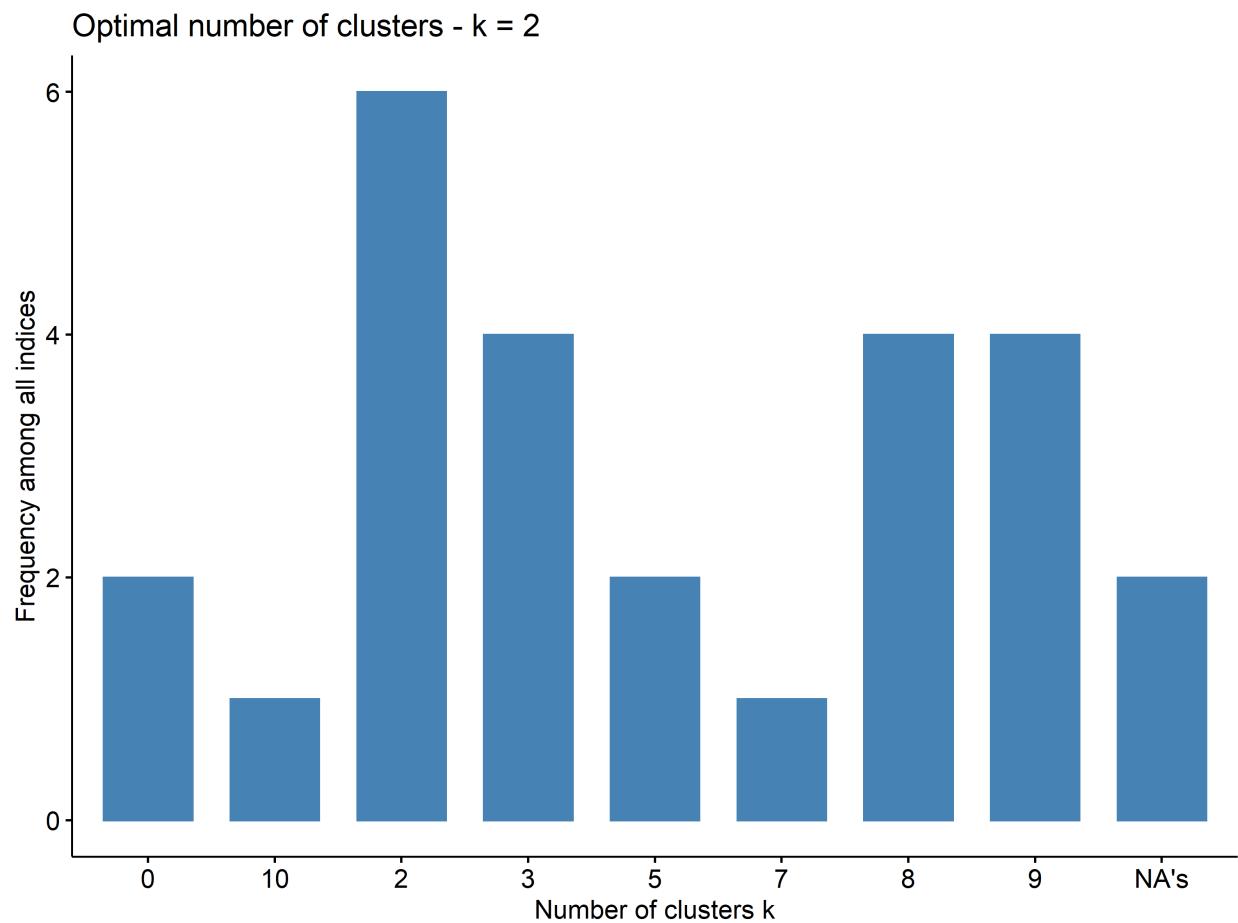


Figure 3: NbClust Results

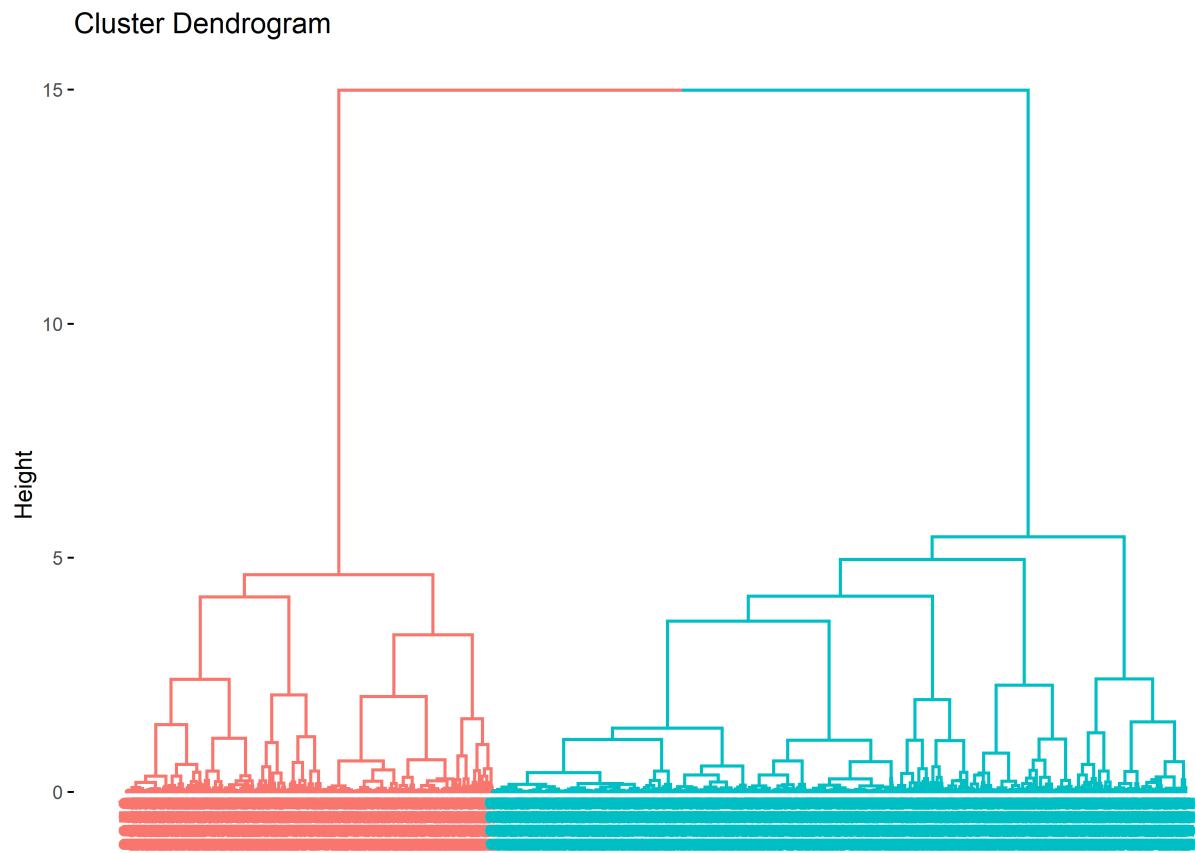


Figure 4: AGNES Dendrogram

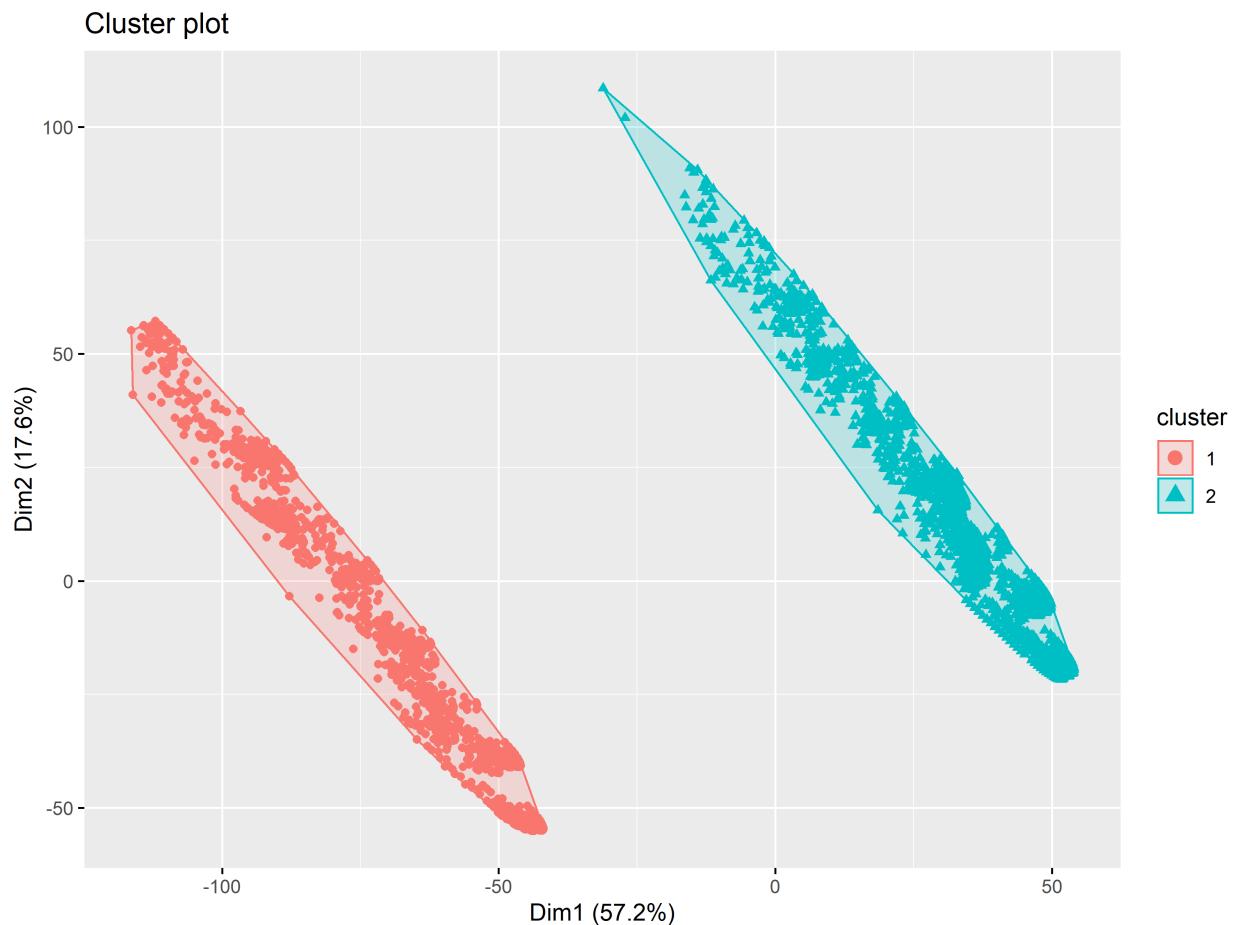
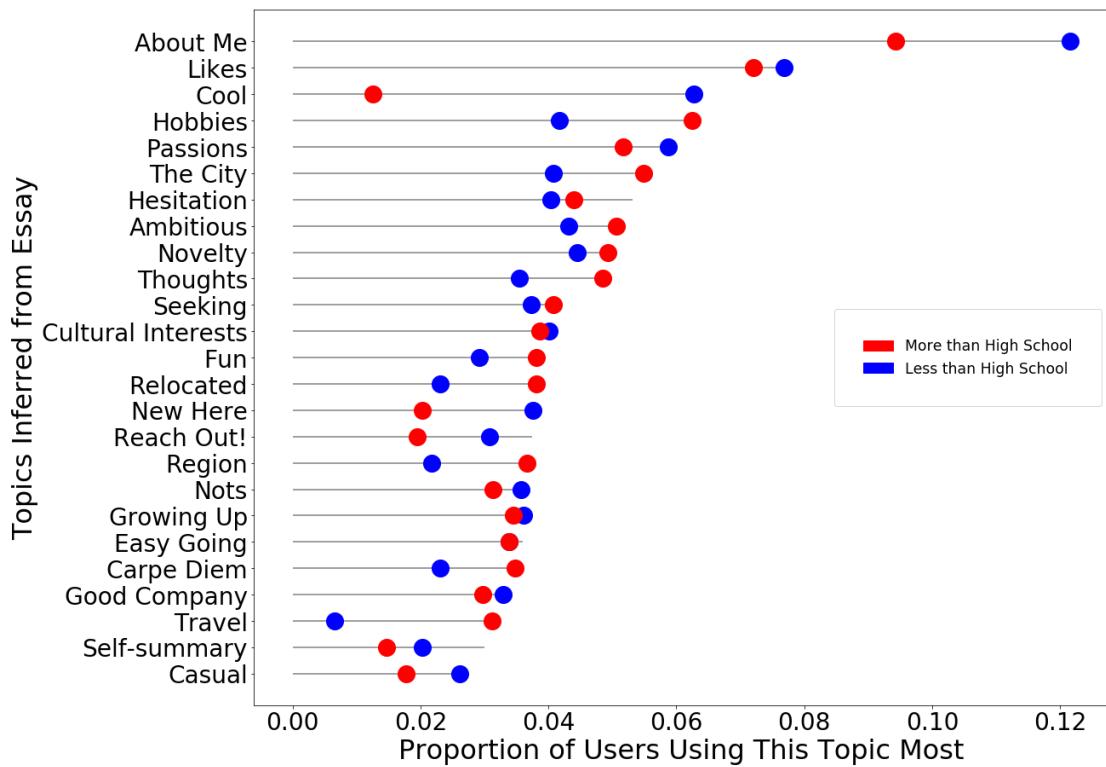
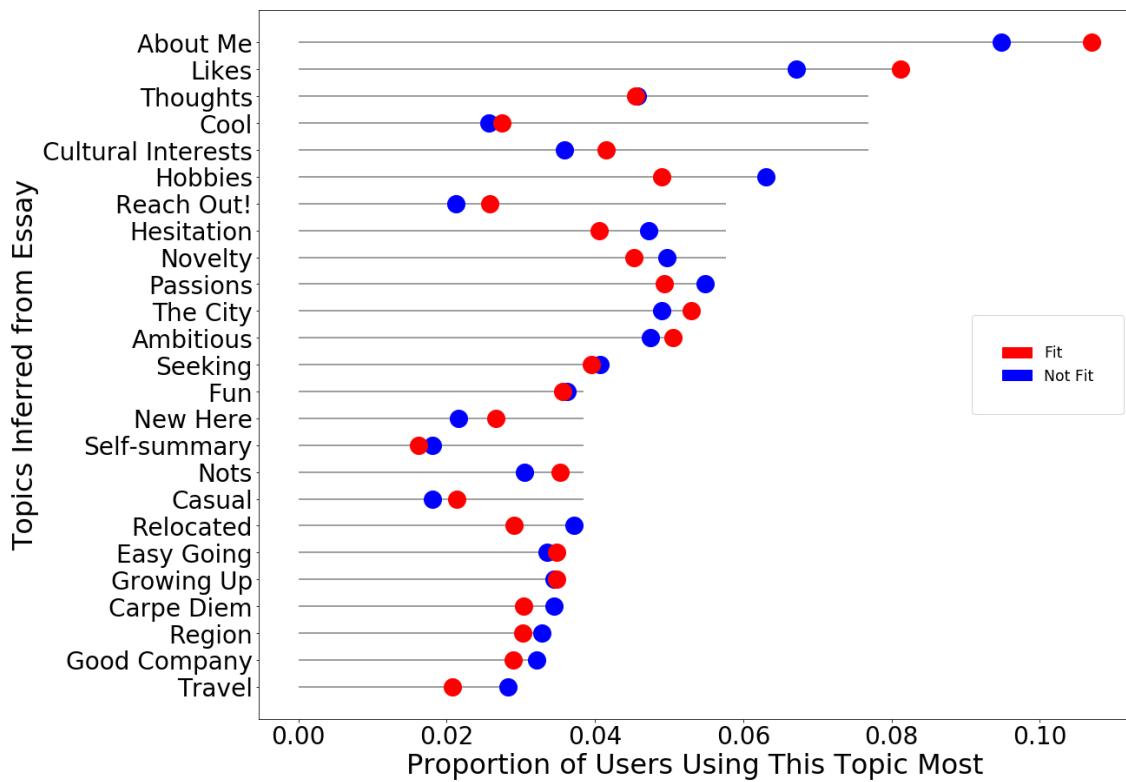


Figure 5: AGNES Cluster Results

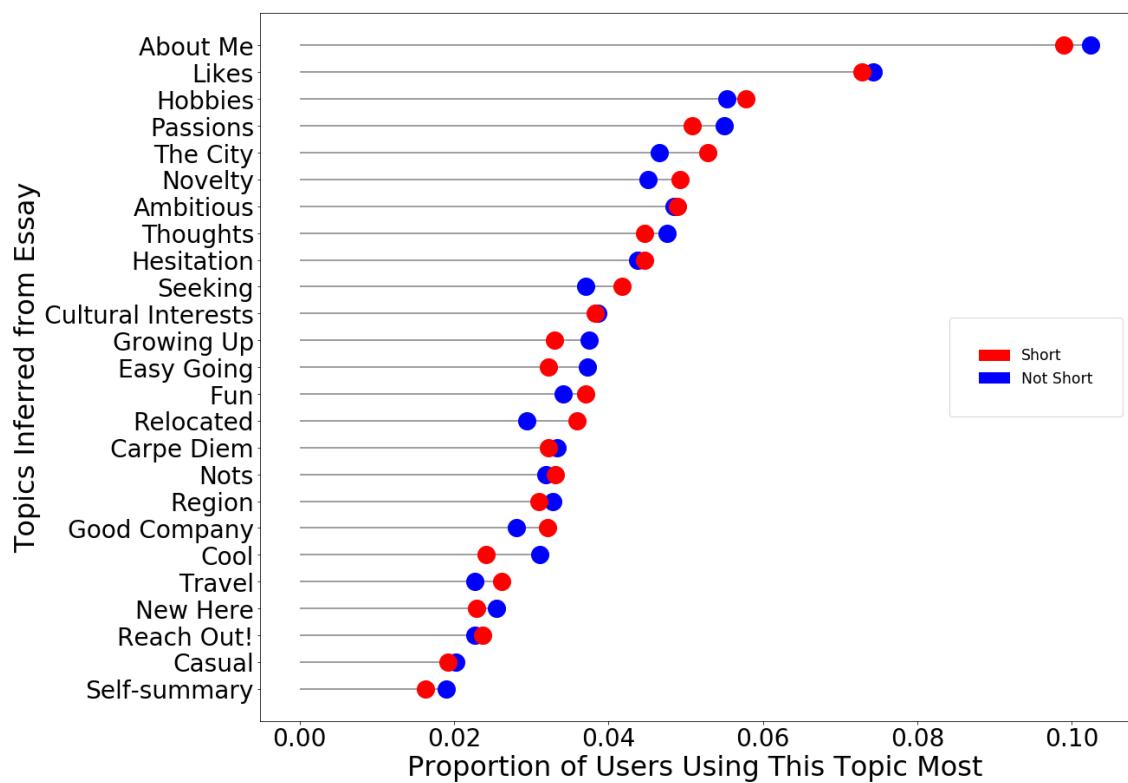
Topics in OkCupid Male Self-Introductions Across Education Levels



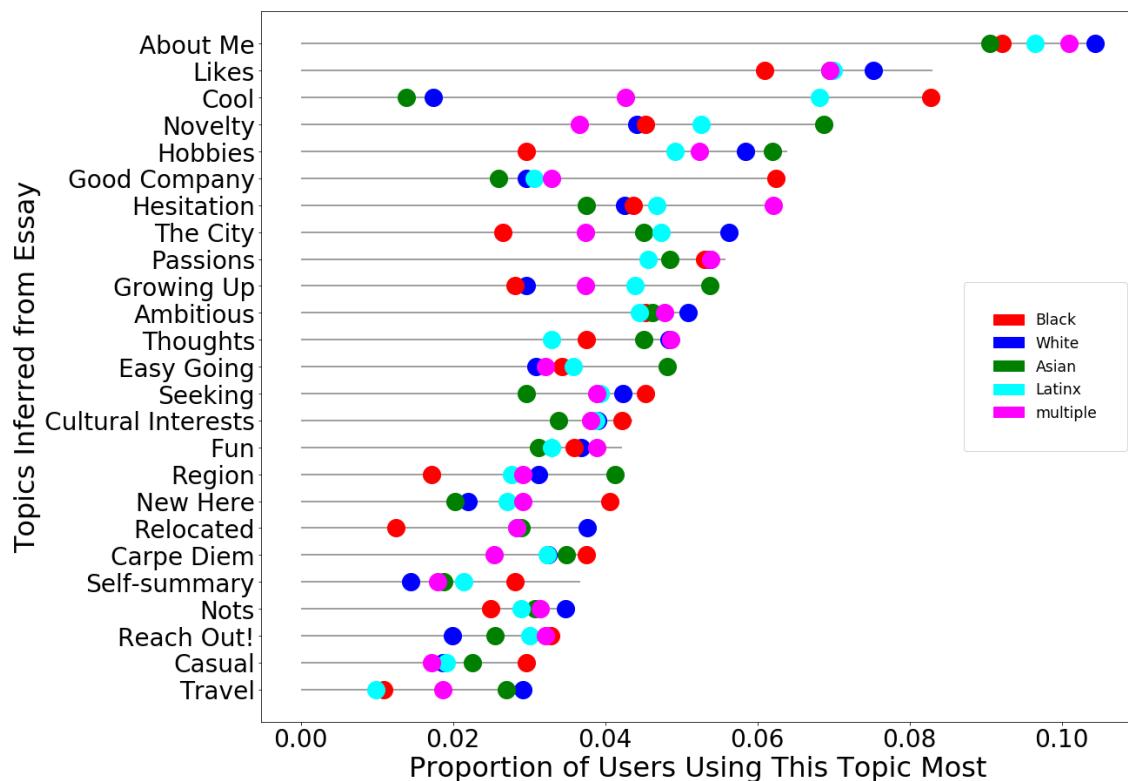
Topics in OkCupid Male Self-Introductions Across Fitness Levels



Topics in OkCupid Male Self-Introductions Across Height Groups

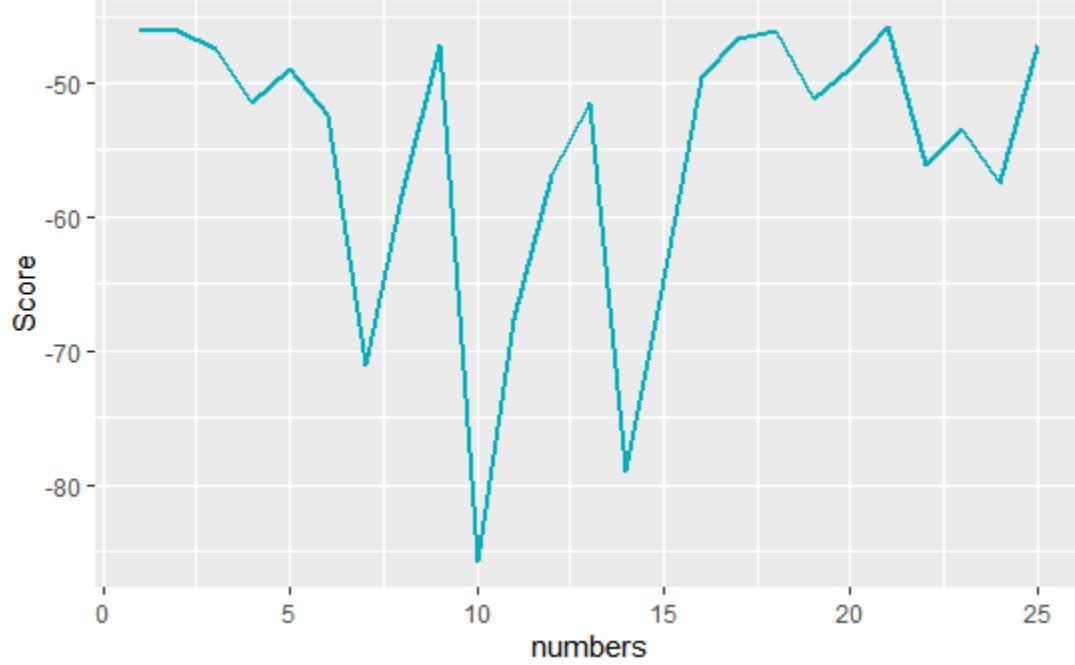


Topics in OkCupid Male Self-Introductions Across Racial Groups



However, although the plots suggest some differences, we don't know if they are statistically significant and also cannot quantify its association with the topic proportions.

As a result, we return to the LDA approach. We choose the topics based on semantic coherence scores.

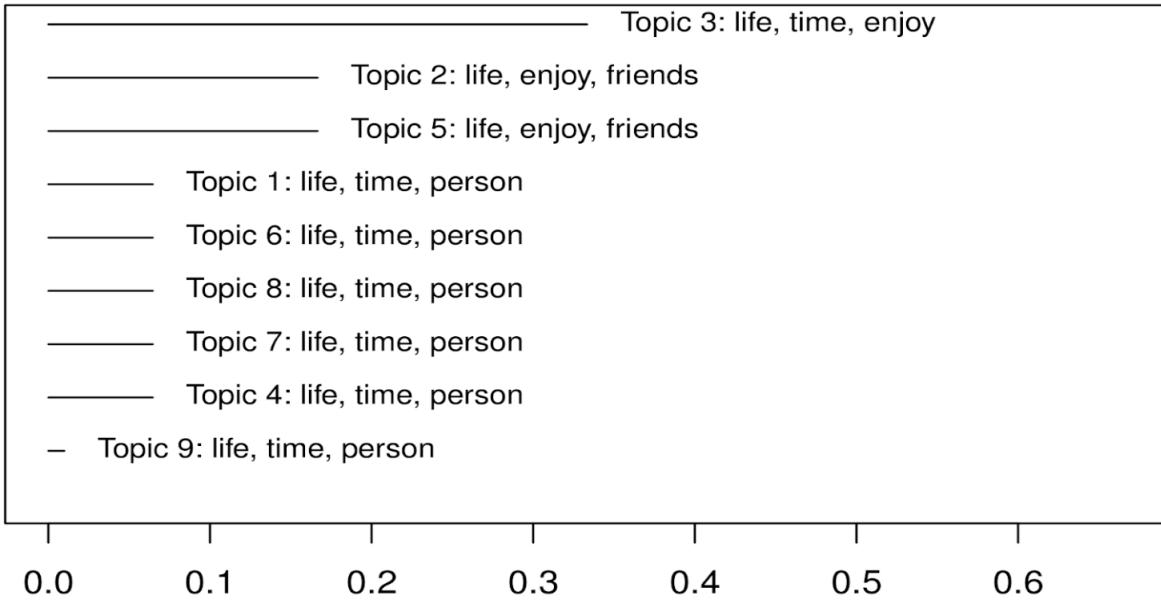


The plot

above suggests 25 topics yields the highest score, but 9 topics yields a good score as well. So we further calculate perplexity , which is regarded as the model's "surprise" at the data. The perplexities are 3261and 3265 for 9 and 25 topics, respectively. Thus, we fix 9 topics for the following analysis.

Next, we fit a LDA topic model with 9 topics where each document is labeled with cluster number (1, 2, 3).

Top Topics



This plot shows that for Cluster 1, almost all topics are associated with the common words such as “life, time, friends”. This result is somewhat discouraging as we cannot draw any insights from it. The topic proportion plots are almost identical for Cluster 2 and Cluster 3.

DBSCAN

As we wanted to determine factors behind why a profile might stick out, we decided to use a DBSCAN model. A DBSCAN model was used to detect outliers within a data-set of 50 vectors calculated by Doc2Vec. As Doc2Vec vectors capture different characteristics about the text, any outliers in these vectors should be profiles that deviate from the norm to some degree. As shown below, these Doc2Vec vectors are highly independent with very few correlations, but also highly clusterable, with a Hopkins Statistic of 0.808.

Using a K-nearest-neighbors distribution plot, we determined that the optimal value for the epsilon neighborhood size of the DBSCAN model was 9. This was determined using 5 nearest neighbors, to match the default minimum number of points in the epsilon region, which we used for the model.

The DBSCAN model identified one cluster, and 108 outliers, accounting for 0.57% of the observations. Using a Wilcox Test, we can determine that the difference in the means between the “typical” profiles and outlier profiles are insignificant for the continuous variables for height and number of long words, but are highly significant for Flesch score and age. Mean age is higher among the outliers by about 4.5 years, while the mean Flesch score of outliers is about 8 points higher. This suggests that the outliers, being older and writing at

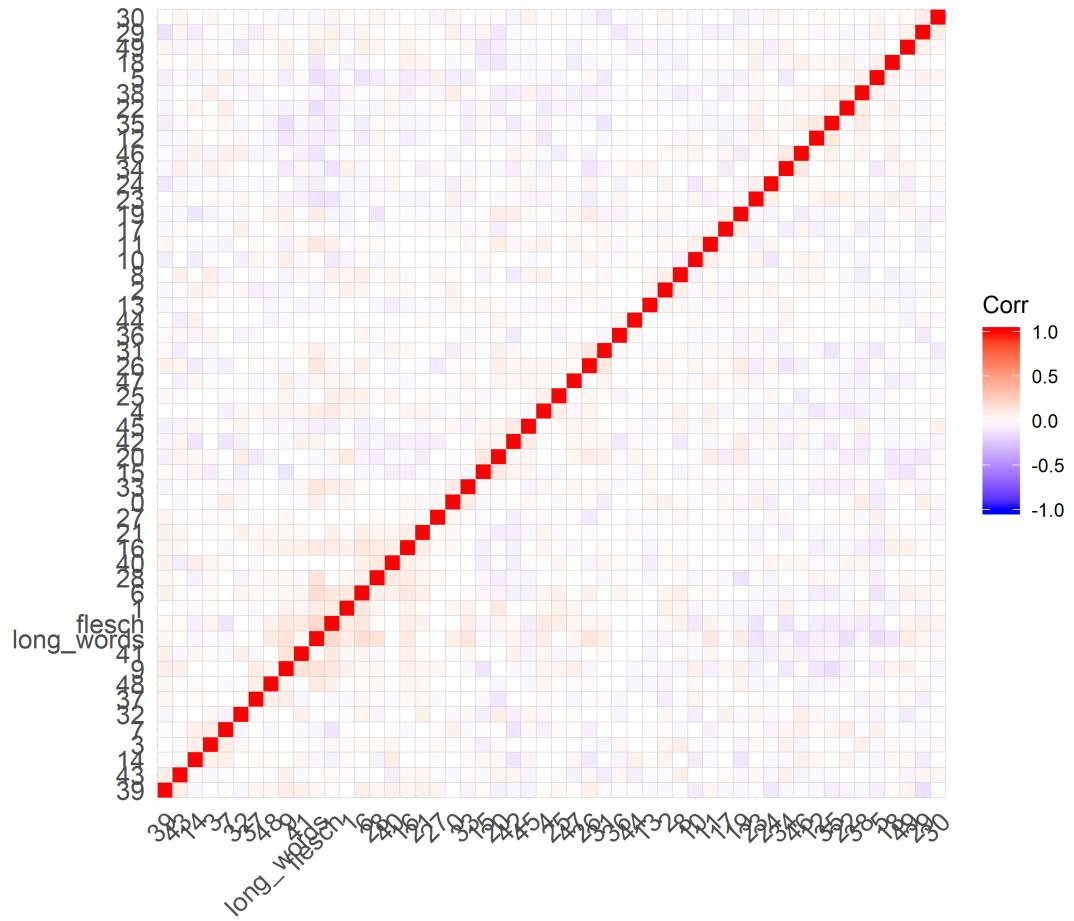


Figure 6: Doc2Vec Correlation Plot

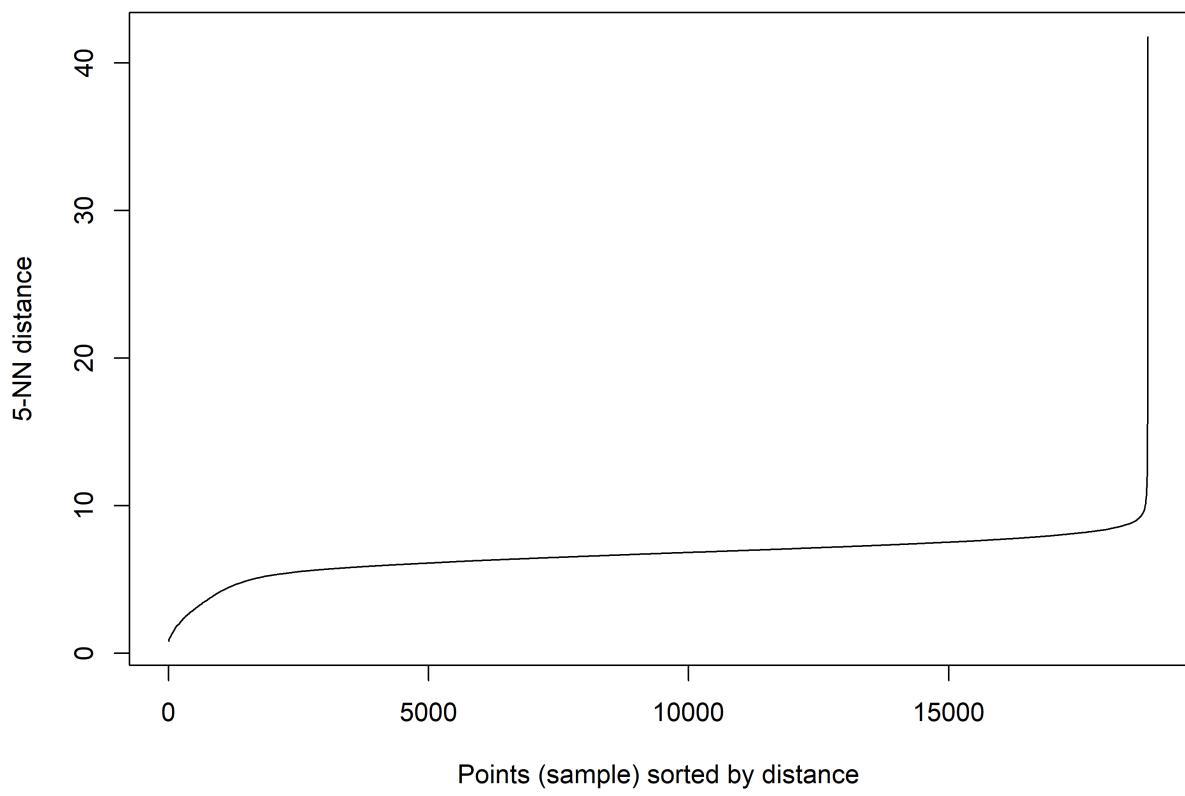


Figure 7: DBSCAN KNN Distribution Plot

a more advanced level, might be more educated (or at least trying to appear so) than their “typical” peers.

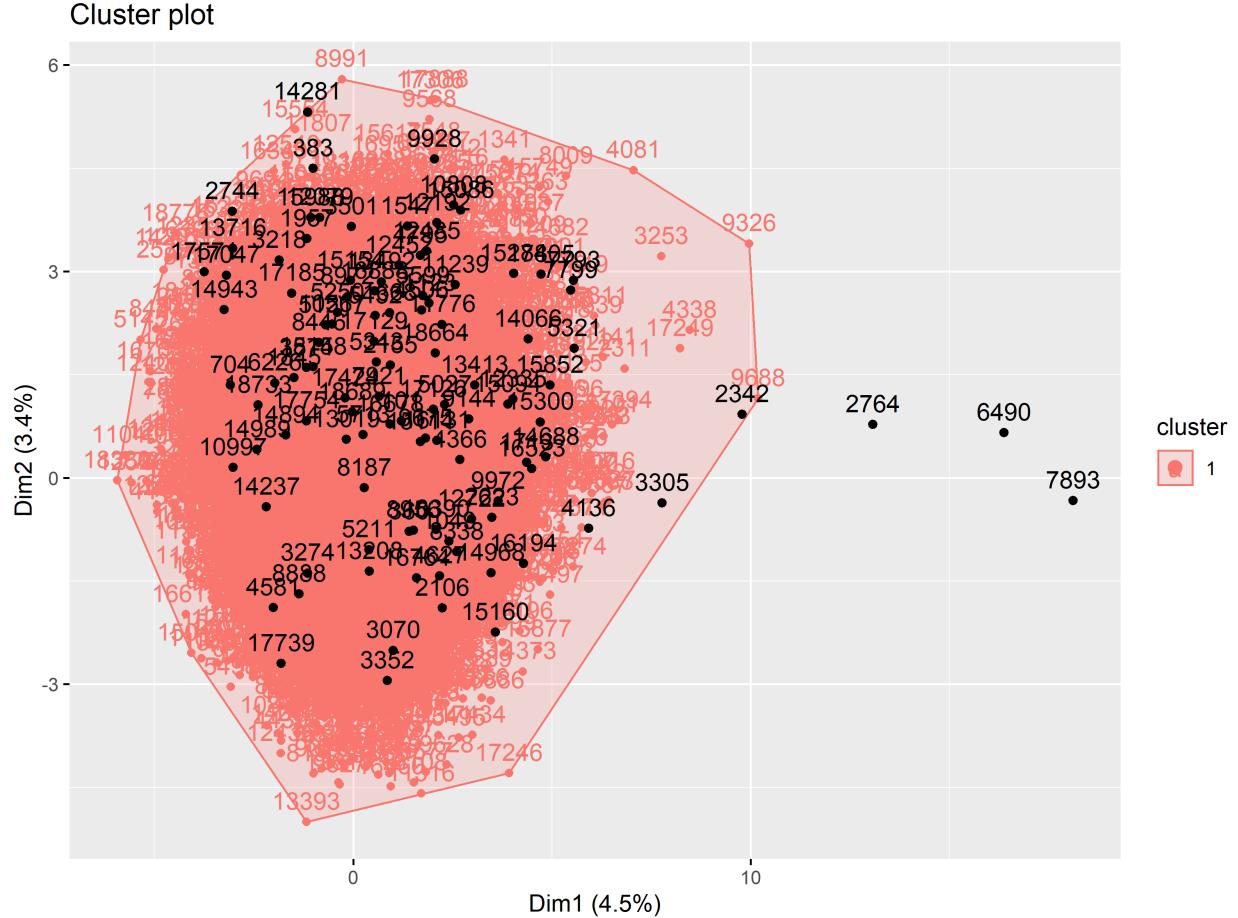


Figure 8: DBSCAN Cluster Plot

Structural Topic Modeling

Motivation

So far, we have explored the patterns within the demographic variables using clustering and self-introduction text using LDA. However, in order to quantify whether people of different backgrounds write different topics, we need to model the topic distributions as a function of the metadata (demographic variables and DBScan score). The structural topic modeling (STM) solves the problem as it allows us to estimate the relationships between topic proportions and document metadata. Similar to the Latent Dirichlet Allocation model, STM also assumes there are some latent document-topic and topic-word distributions generating documents. However, STM differs from LDA as it handles the document-associated metadata. As a result, STM allows

us to predict how topical prevalence (proportion of a topic across multiple documents) or topic content (the topic composition of terms) would shift when the metadata changes.

Estimation

We use the `stm` R package to estimate the model, summarize results, and visualize with the word cloud and topic network. In our project, we are interested in the topical prevalence. So the response (dependent variable) is the proportion of a topic across multiple self-introductions. The metadata (independent variables) is ‘fit’, ‘education’, ‘height_group’, ‘race_ethnicity’, and ‘dbscan_cluster’.

In the preparation step, we use `textProcessor` function to stem the words and remove stopwords. Then we use `prepDocuments` function to structure and index the data. As the low-frequency words are probably the more memorable ones, we set the ‘lower.thresh’ option to 0 to keep all words.

Then we estimate the model for the topic prevalence using 9 topic models. During the estimation, the proportions of 9 topics are regressed on the metadata. There are many ways to visualize the results. The following plot shows the expected topic proportions

The proportions across 9 topics do not have a large gap, as the most common topic is 0.14 and the least common one is 0.10. We observe that the top three topics seem to consist of common nouns and verbs that are not very memorable. So we further explore the most frequent words in the model for both the most common topics (#5, #4, and #7) and the least common topics (#1, #2, and #9).

Top Topics

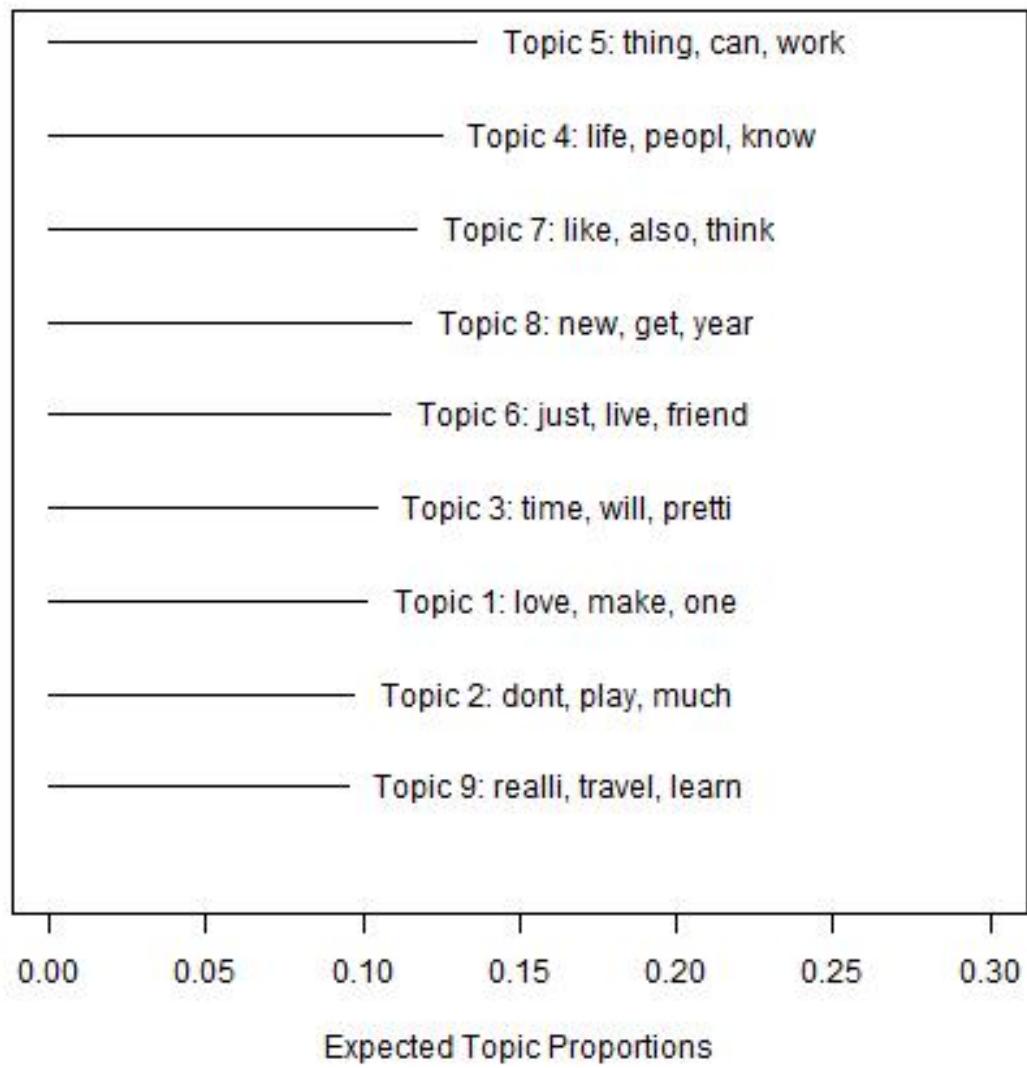


Figure 9: Top Topic

Topic 5:

g, can, work, enjoy, good, music, move, take, day, well, area, feel, now,
laugh, adventur, movi, lot, art, better, nice

Topic 4:

life, peopl, know, want, alway, meet, citi, place, littl, francisco, still,
famili, come, spend, sometim, bit, fun, mean, happi, hope

Topic 7:

, also, think, someon, ive, world, san, see, fun, even, hike, cook, best,
passion, use, self, date, need, often, night

Topic 1:

e, make, one, person, interest, right, mani, sens, share, recent, heart,
ook, general, california, especi, point, seem, spent, moment, studi

Topic 2:

nt, play, much, way, great, someth, game, sport, ill, care, job, believ,
differ, travel, funni, drink, current, high, end, intellig

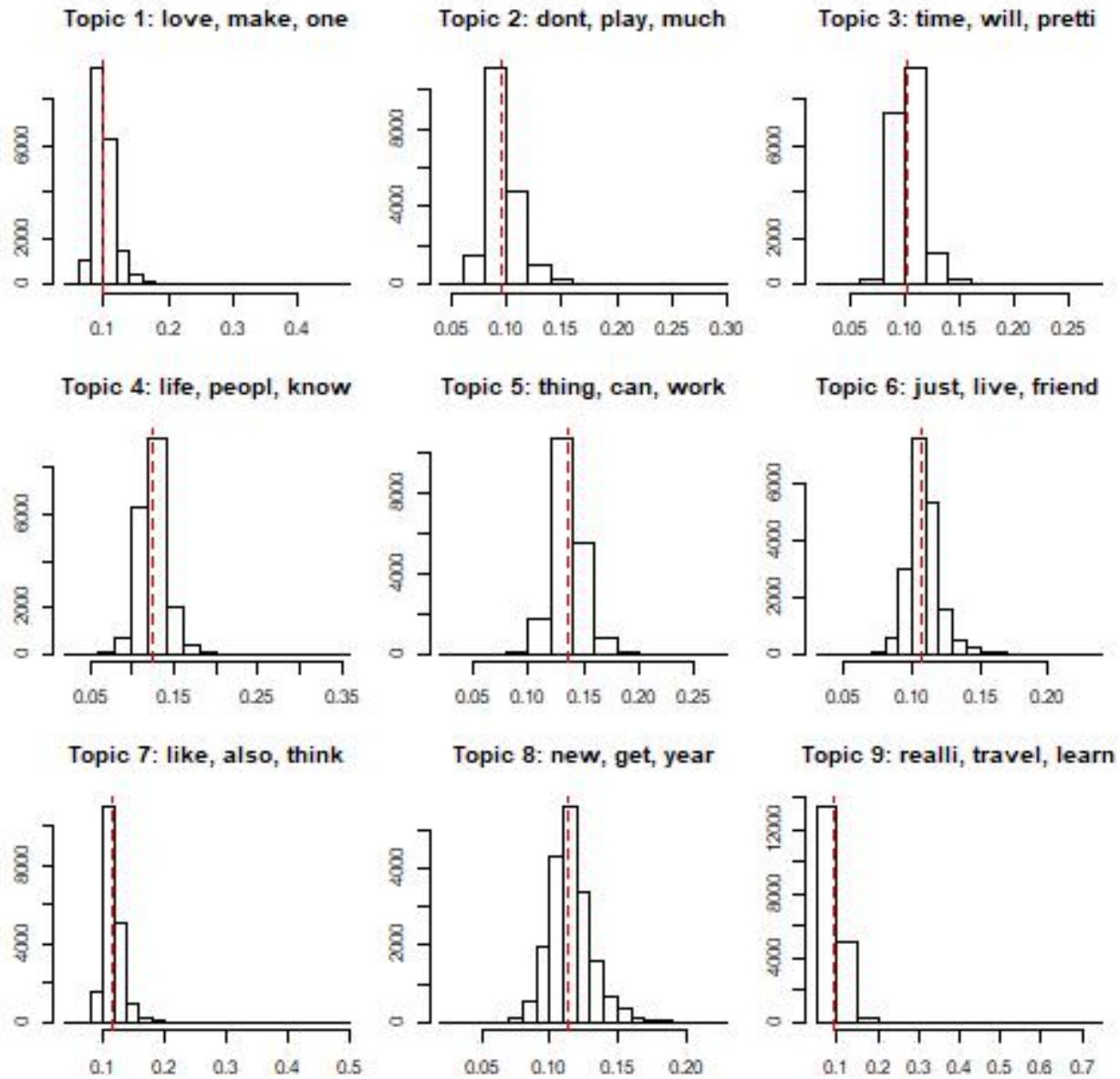
Topic 9:

balli, travel, learn, kind, never, serious, home, everi, danc, that, easi,
now, hang, ask, woman, word, free, favorit, enough, ride

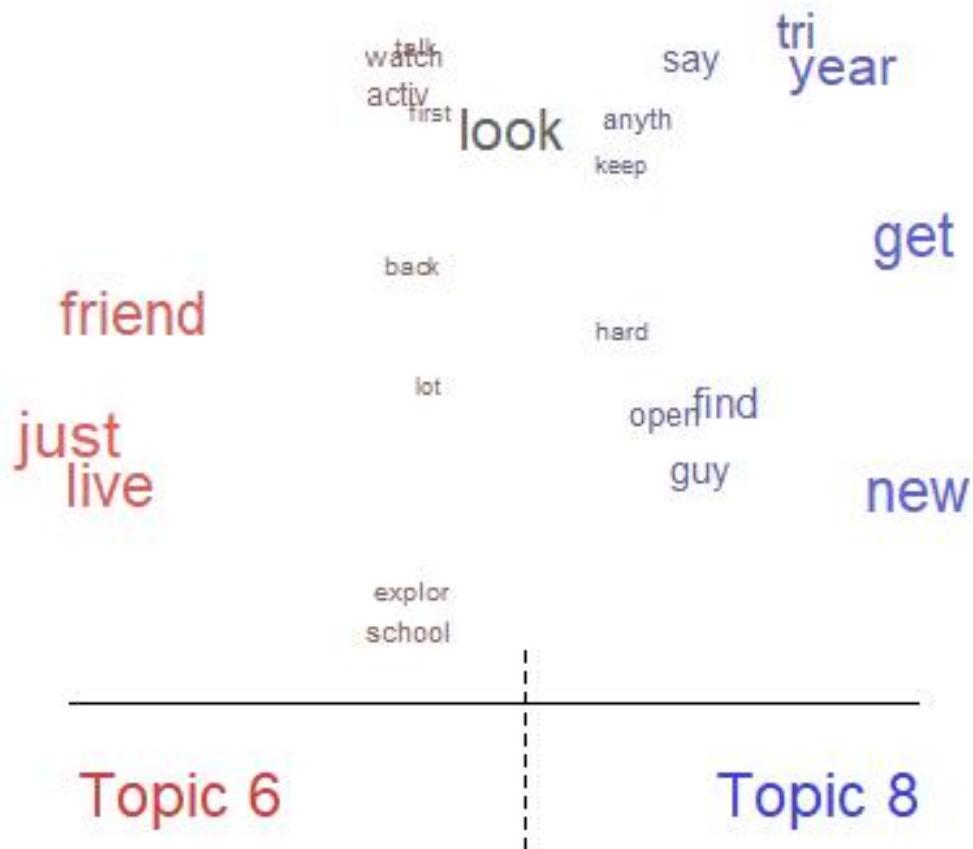
Although it's hard to tell the quantified difference, we find the words in the least common topics are a little more memorable, such as 'intelligent' and 'dance'.

When we plot the histograms of topics, we find that most of the topics consist of around 10% of the topics of the documents. Interestingly, the histograms of topics #6 and #8 seem to have a bell-curve shape, which suggests there are many documents have higher or lower proportions of 10%.

Distribution of MAP Estimates of Document-Topic Proportions



In particular, we plot these two plots on the same line. Topic #6 is about ‘friend’ and ‘live’ and topic #8 is about ‘year’ and ‘get’. So our interpretation is that topic #6 is about casually looking for friends while topic #8 is about setting a goal of getting something new in the year. As a result, although most users talk about them explicitly, some talk them more and some use other expressions.



Also, `topicCorr()` calculates correlations between topics, where positive correlations mean that both topics are likely to be discussed within a document.

For the OkCupid beginners, it will be helpful if they could learn what topics are they expected to write together. However, this would only make an ordinary self-introduction. To make it really memorable and stand out, they should consider writing a few topics that are unique and quirky.

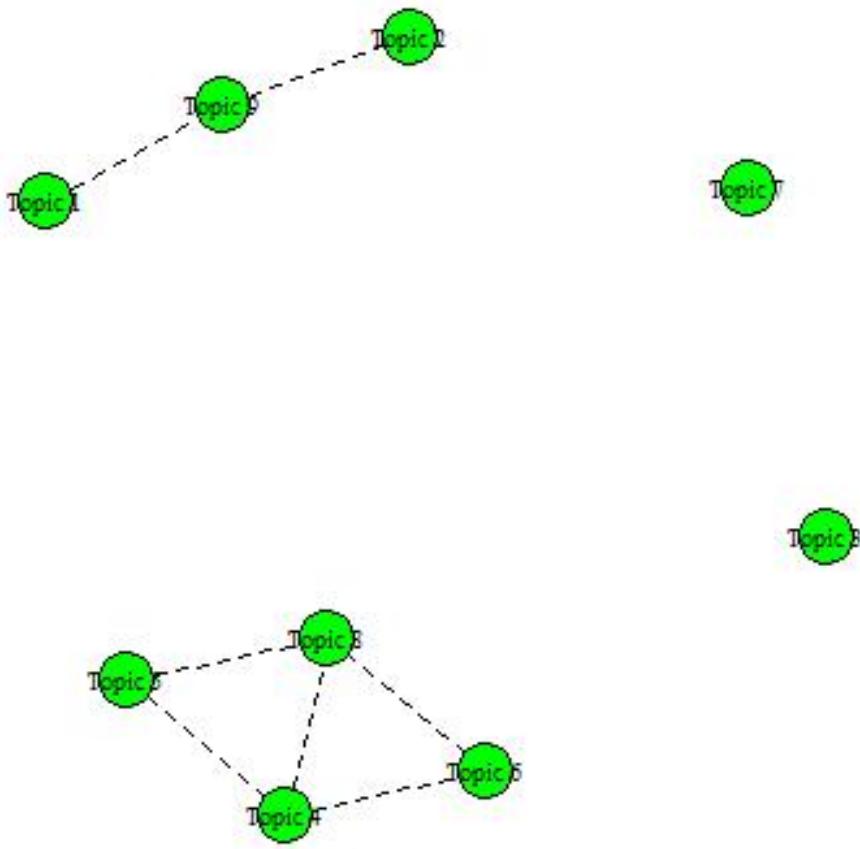


Figure 10: Topic Networks

Evaluation

To further understand whether the topics make sense, we use the `topicQuality()` function to plot the semantic coherence and exclusivity values associated with each topic.

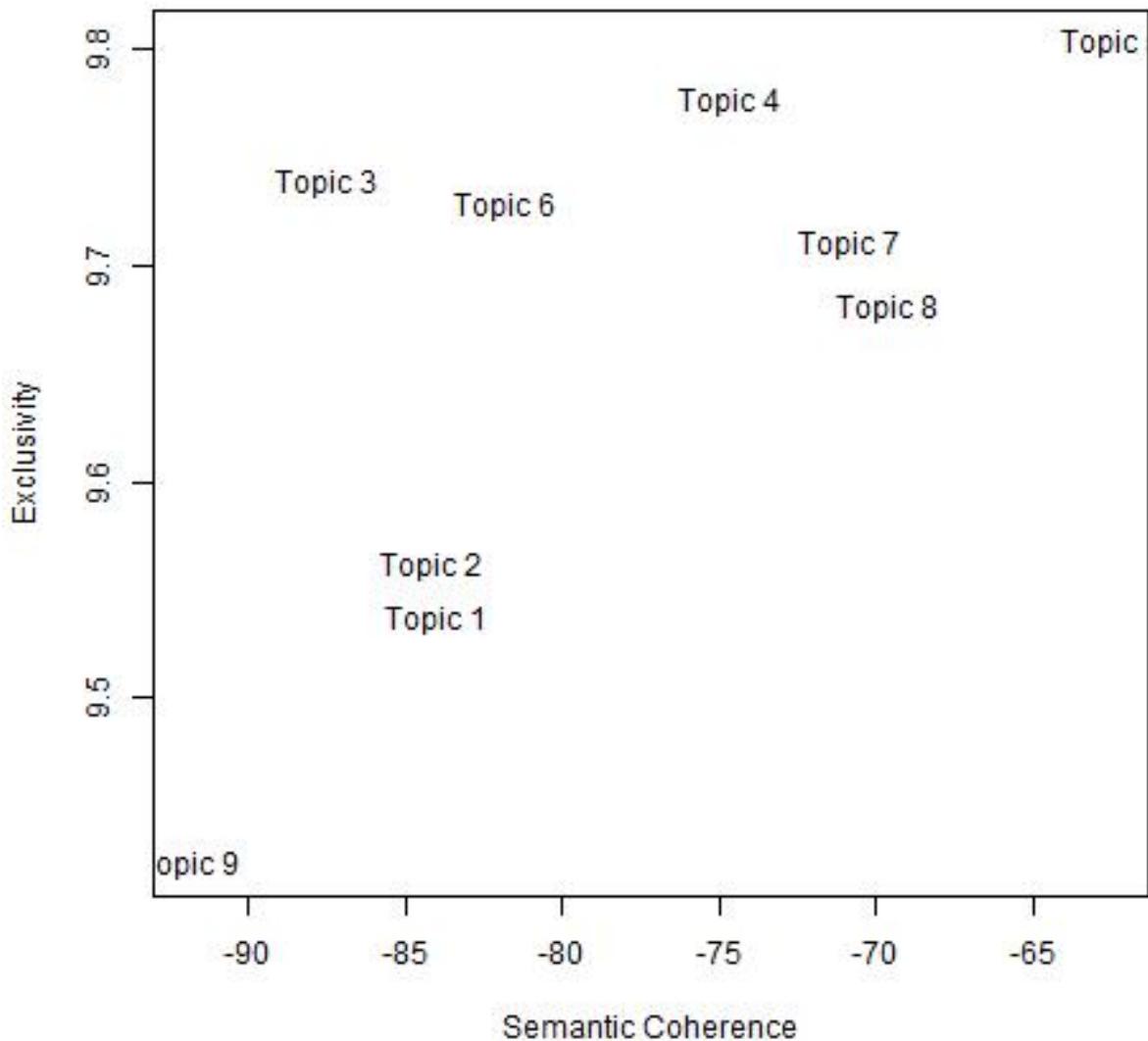


Figure 11: Topic Quality

The results seem to highly correlated with the topic proportions, as topics with higher proportions also have higher semantic coherence and exclusivity values.

Effect

Finally, we use `estimateEffect()` function to estimating relationships between metadata and topic prevalence. There are 9 regressions under the hood and 12 coefficients associated with each. To effectively analyze the results with 108 coefficients, we compile them into a table by indicating the coefficients which are statistically significant at the level of 5%. These significant coefficients suggest that we can reject the null hypothesis and accept the alternative hypothesis that a relationship exists between the topic prevalence and metadata.

Significant Covariates of STM (0: '****' 0.001: '***' 0.01: '**' 0.05: '*')

Topic / X	fit_not_fit	fit_unknown	edu_More than High School	edu_unknow n	height_groupshort	race_ethnicityBlac k	race_ethnicityLatinx	race_ethnicitymult iple	race_ethnicityother	race_ethnicityWhite	dbscan_c luster
1			*					***	**	**	***
2	***		***	**			**	*		**	
3	***	*	***	***				**			
4	.		***	*				*		***	**
5	.		*			**					
6			***								
7					*		**
8	***		**	***		***	***	***		***	*
9	***		***			***					**

Figure 12: Significant Coefficients

We observe from the table that the factor “edu_More than HIgh Schoo” is significant at all of the 9 regressions. This means that being more educated is associated with some variations in the topic proportions. Intuitively, this suggests more educated people might want to write certain topics that make them memorized as intelligent and knowledgable.

We notice ‘height_groupshort’ is only significant for regression #7. Topic # 7 has high probability words such as “like, also, think, someone”, which seems quite ordinary. It’s likely the use of ‘also’ indicates they are disconfident when introducing themselves, which might because they feel they are too short.

When looking at the coefficients at each regression, we are surprised that in the regression #6, only edu_More than HIgh Schoo" matters for Topic #6's proportions. In comparison, 8 out of the 11 coefficients are

significant in the regression #8. Our current understanding is that for topic#8, people with different backgrounds have various ideas of how much to write about it.

In summary, we get some interesting results during our first time using structural topic modeling. Although we are still learning it, we believe it has great potential to uncover the underlying relationships between metadata and topic prevalence. Such relationships would be useful for OkCupid to provide users with writing tips to make their self-introduction more memorable.

Output

The above three pictures show the protocol of the final product that we have in mind. The protocol uses the example of Li (one of the project members), although the information is hypothetical. The first picture displays his demographic information and a short self-introduction. The second picture shows that the new algorithm highlights the common and memorable words by comparing them with the frequencies among his peers. He also gets a score of 63 and some tips on improving the writing. The third picture is the revised version of previous writing, which now Li has a memorable self-introduction with an increased score of 87. He should be confident to reach out to other users on OkCupid now.

Discussion

The data of this project has several limitations at this stage. Firstly, without data on the outcome (such as the number of messages), we do not know have a relative measure for the self-introductions' memorabilities. As a result, we don't have the outcomes to train the scoring function with machine learning. Secondly, without follow-up interviews, we cannot measure whether the specific choice of words was aimed at authenticity or matches with an awareness of 'relationshopping' (experienced users use certain phrases fraudulently to hook others). Thirdly, without the users' other profile images, we cannot estimate the effect of a memorable self-introduction on outcomes as the quality of images is a potential moderating variable.

One important issue of applying the structural topic modeling is overfitting, especially when we have a large number of covariates. We tried to measure the model's accuracy by residual sum of squares on both training set and testing test. However, at this point, we haven't figured out how to predict the new topic prevalences for the testing set. Also there are many remaining questions for this to work. For example, should we measure the accuracy by the average of 9 RSSs? If we use Lasso regression to select a subset of covariates, how should we make the interpretation as there could be different covariates in 9 regressions?

Conclusion

In this project, we apply different unsupervised machine learning methods learned from class, especially clustering and text mining. Although we have got some interesting results, we realize that there are many other alternative methods that are potentially helpful, such as factor model and association rule mining. A factor analysis may be useful for future analysis as the variables driving choice of words- such as cognitive skills, cultural attitudes, personality traits are likely to be latent and unobservable factors. Association rule mining would further discover the interesting associations within people's demographics or find certain phrases in self-introductions that often occur together.

Lastly, we want to express our excitement for learning these useful tools from the UML class. Also we would like to thank you Dr. Waggoner and our classmates for providing valuable feedback during the two rounds of presentation. This experience greatly arises our interest to study more UML methods in the future.

Appendices

Below is all the source code used for this project, which is available on the GitHub repo for this project:
<https://github.com/tonofshell/unsupervised-dating>

Kmeans Code

```
## ----prep-----  
library(tidyverse)  
library(here)  
library(cluster)  
library(FactoMineR)  
library(factoextra)  
library(NbClust)  
library(dbSCAN)  
library(cowplot)  
library(skimr)  
library(ggcorrplot)  
library(missMDA)  
library(cluster)  
library(data.table)  
  
setwd("C:/Users/lliu9/Desktop/UML_Project/unsupervised-dating/Data")  
data <- read_csv("final_okcupid.csv")  
data_sample <- sample_n(data, 2000)  
  
demo <- data_sample[c("fit", "edu", "height_group",  
"race_ethnicity")]  
  
gower_demo <- demo %>% mutate_all(factor) %>% daisy(metric = "gower")  
  
## ----Kmeans-----
```

```

set.seed(123)

kmeans <- kmeans(gower_demo, centers = 3, nstart = 15)

## -----
text <- data_sample[c("clean_text")]
text$kmeanscluster = as.factor(kmeans$cluster)
write_csv(text, "essay0.csv")

## ----Examination-----
data_sample$kmeanscluster = as.factor(kmeans$cluster)

# Inspect the kmeans object
str(kmeans)

## ----Visualization-----
jpeg("kmeans3.jpeg")
fviz_cluster(kmeans, gower_demo)
dev.off()

## ----validation-----
library(fpc)

# Calinski Harabasz index
clustering.ch <- kmeansruns(gower_demo, krange = 1:5,
                             criterion = "ch")

# average silhouette width
clustering.asw <- kmeansruns(gower_demo, krange = 1:5,

```

```

criterion = "asw")

critframe <- data.frame(k = 1:10, ch = scale(clustering.ch$crit),
                        asw = scale(clustering.asw$crit))

critframe <- melt(critframe, id.vars = c("k"), variable.name = "measure",
                   value.name = "score")

ggplot(critframe, aes(x = k, y = score, color = measure)) +
  geom_point(aes(shape = measure)) + geom_line(aes(linetype = measure)) +
  scale_x_continuous(breaks = 1:10, labels = 1:10)
summary(clustering.ch)

```

AGNES Code

```

## ----setup,
## include=FALSE-----
library(tidyverse)
library(knitr)
library(here)
library(cluster)
library(FactoMineR)
library(factoextra)
library(NbClust)
library(dbSCAN)
library(cowplot)
library(skimr)
library(ggcorrplot)
library(missMDA)
library(cluster)
library(missForest)
library(tictoc)

```

```

library(doParallel)

knitr::opts_chunk$set(echo = TRUE, fig.height = 6,
fig.width = 8, dpi = 400)

set.seed(60615)

## ----data-----
original_data = read_csv(here("Data", "final_okcupid.csv")) %>%
  select(-c("new_index", "orig_index", "clean_text",
  "essay9", "long_words", "flesch", "dbscan_cluster",
  "profile_length", "prop_longwords"))

names(original_data)

## ----descr-stats, cache=TRUE, fig.height = 8,
## fig.width = 11-----
skim_list = original_data %>% skim() %>% partition()

skim_list$numeric %>% kable()
skim_list$character %>% kable()

original_data %>% mutate_if(is.character, factor) %>%
  mutate_all(as.numeric) %>% cor(use = "pairwise.complete.obs") %>%
  ggcorrplot()

clusterability = original_data %>% mutate_if(is.character,
factor) %>% mutate_all(as.numeric) %>% sample_n(5000) %>%
  get_clust_tendency(n = 50)
clusterability$hopkins_stat

```

```

clusterability$plot

## ----pca,
## cache=TRUE-----

original_data %>% mutate_if(is.character, factor) %>%
  mutate_all(as.numeric) %>% mutate_all(scale) %>%
  PCA(graph = FALSE) %>% fviz_pca_biplot(label = "var",
  col.var = "red", col.ind = "grey")

ggsave2(here("Clustering", "pca_v2.png"), height = 7,
  width = 11)

## ----agg-nest, error=TRUE,
## cache=TRUE-----

sampled_data = original_data %>% sample_n(5000)

agnes_data = sampled_data %>% mutate_if(is.character,
  factor) %>% mutate_all(as.numeric) %>% mutate_all(scale)

agnes_diss = agnes_data %>% as.matrix() %>% daisy(metric = "gower")

nb_results = NbClust(data = agnes_data, diss = agnes_diss,
  distance = NULL, min.nc = 2, max.nc = 10, method = "ward.D2")

fviz_nbclust(nb_results)

agnes_mod = agnes_diss %>% hcut(isdiss = TRUE, k = 2,
  hc_func = "agnes", hc_method = "ward.D2")

fviz_dend(agnes_mod)

sampled_data$cluster = agnes_mod$cluster

fviz_cluster(agnes_mod, data = agnes_diss, labelsize = 0)

saveRDS(sampled_data, here("Data", "Results", "agnes_results.rds"))
write_csv(sampled_data, here("Data", "Results", "agnes_results.csv"))

```

```

## ----clust-inter-----
modal = function(vect, percent = FALSE, only_one = FALSE) {
  library(tidyverse)
  modal_val = vect %>% unlist() %>% table() %>% .[. == max(.)] %>% names()
  if (only_one) {
    modal_val = modal_val[1]
  }
  if (percent) {
    return(vect %>% unlist() %>% .[. == modal_val] %>%
      (function(x) length(x)/length(vect)))
  }
  modal_val
}

cluster_significance = function(var, data, clus_var = "cluster") {
  wilcox.test(as.formula(paste(var, "~", clus_var)),
  data)$p.value
}

sampled_data %>% select_if(is.numeric) %>% group_by(cluster) %>%
  summarise_all(mean) %>% kable(caption = "Mean by Cluster")

sampled_data %>% select_if(is.numeric) %>% {
  sapply(names(select(., -cluster)), cluster_significance,
  data = .)
} %>% round(3) %>% enframe() %>% kable(caption = "Wilcox Test P-values")

sampled_data %>% select_if(is.numeric) %>% group_by(cluster) %>%
  group_by(cluster) %>% summarise_all(sd) %>% kable(caption = "Standard Deviation by Cluster")

```

```

sampled_data %>% select_if(is.numeric) %>% group_by(cluster) %>%
  group_by(cluster) %>% summarise_all(median) %>%
  kable(caption = "Median by Cluster")

sampled_data %>% mutate(cluster = factor(cluster)) %>%
  select_if(function(x) !is.numeric(x)) %>% group_by(cluster) %>%
  summarise_all(modal, only_one = TRUE) %>% kable(caption = "Mode by Cluster")
sampled_data %>% mutate(cluster = factor(cluster)) %>%
  select_if(function(x) !is.numeric(x)) %>% group_by(cluster) %>%
  summarise_all(modal, percent = TRUE, only_one = TRUE) %>%
  mutate_if(is.numeric, round, 3) %>% kable(caption = "Mode by Cluster")

```

DBSCAN Code

```

## ----setup,
## include=FALSE-----
library(tidyverse)
library(knitr)
library(here)
library(cluster)
library(FactoMineR)
library(factoextra)
library(NbClust)
library(dbscan)
library(cowplot)
library(skimr)
library(ggcorrplot)

knitr:::opts_chunk$set(echo = TRUE, fig.height = 6,
                      fig.width = 8, dpi = 400)

set.seed(60615)

```

```

## ----data-----
demo_data = read_csv(here("Data", "compressed_okcupid.csv"))
doc2vec_data = read_csv(here("Data", "doc2vec_results.csv")) %>%
  bind_cols(select(demo_data, long_words, flesch)) %>%
  scale() %>% as_tibble()

doc2vec_data %>% skim() %>% partition() %>% .$numeric %>%
  kable()

doc2vec_data %>% select(-X1) %>% {
  ggcorrplot(cor(.), p.mat = cor_pmat(.), hc.order = TRUE,
  insig = "blank")
}

## ----clusterability-----
clusterability = doc2vec_data %>% select(-X1) %>% get_clust_tendency(n = 15)
# clusterability$plot

## ----dbscan-----
doc2vec_data %>% select(-X1) %>% kNNdistplot(k = 5)
dbscan_mod = doc2vec_data %>% select(-X1) %>% dbSCAN(9,
  5)
doc2vec_data %>% select(-X1) %>% {
  fviz_cluster(dbscan_mod, data = .)
}

dbscan_results = doc2vec_data %>% bind_cols(enframe(dbscan_mod$cluster,
  name = NULL, value = "cluster"))

read_csv(here("Data", "compressed_with_results.csv")) %>%
  mutate(dbscan_cluster = dbscan_mod$cluster) %>%
  write_csv(here("Data", "compressed_with_results.csv"))

```

```

## -----merge-----
merged_demo_data = demo_data %>% select(-essay0, -essay9) %>%
  bind_cols(select(dbSCAN_results, cluster))

merged_doc2vec_data = demo_data %>% select(X1, essay0,
  essay9) %>% bind_cols(select(dbSCAN_results, -X1))

## -----demo-data-----
modal = function(vect, percent = FALSE, only_one = FALSE) {
  library(tidyverse)
  modal_val = vect %>% unlist() %>% table() %>% .[. == max(.)] %>% names()
  if (only_one) {
    modal_val = modal_val[1]
  }
  if (percent) {
    return(vect %>% unlist() %>% .[. == modal_val] %>%
      (function(x) length(x)/length(vect)))
  }
  modal_val
}

cluster_significance = function(var, data, clus_var = "cluster") {
  wilcox.test(as.formula(paste(var, "~", clus_var)),
  data)$p.value
}

merged_demo_data %>% select(-c(X1, education)) %>%
  select_if(is.numeric) %>% group_by(cluster) %>%
  summarise_all(mean) %>% kable(caption = "Mean by Cluster")

```

```

merged_demo_data %>% select(-X1) %>% select_if(is.numeric) %>%
{
  sapply(names(select(., -cluster)), cluster_significance,
  data = .)
} %>% round(3) %>% enframe() %>% kable(caption = "Wilcox Test P-values")

merged_demo_data %>% select(-c(X1, education)) %>%
  select_if(is.numeric) %>% group_by(cluster) %>%
  group_by(cluster) %>% summarise_all(sd) %>% kable(caption = "Standard Deviation by Cluster")
merged_demo_data %>% select(-c(X1, education)) %>%
  select_if(is.numeric) %>% group_by(cluster) %>%
  group_by(cluster) %>% summarise_all(median) %>%
  kable(caption = "Median by Cluster")

merged_demo_data %>% select(-c(X1, education)) %>%
  mutate(cluster = factor(cluster)) %>% select_if((function(x) !is.numeric(x))) %>%
  group_by(cluster) %>% summarise_all(modal, only_one = TRUE) %>%
  kable(caption = "Mode by Cluster")
merged_demo_data %>% select(-c(X1, education)) %>%
  mutate(cluster = factor(cluster)) %>% select_if((function(x) !is.numeric(x))) %>%
  group_by(cluster) %>% summarise_all(modal, percent = TRUE,
  only_one = TRUE) %>% mutate_if(is.numeric, round,
  3) %>% kable(caption = "Mode by Cluster")

## ----interpret-outliers-----
dbscan_results %>% select(-X1) %>% group_by(cluster) %>%
  summarise_all(mean) %>% kable(caption = "Mean by Cluster")
dbscan_results %>% select(-X1) %>% group_by(cluster) %>%
  summarise_all(sd) %>% kable(caption = "Standard Deviation by Cluster")
dbscan_results %>% select(-X1) %>% group_by(cluster) %>%
  summarise_all(median) %>% kable(caption = "Median by Cluster")

```

```
## ----profile-diff-----  
merged_doc2vec_data %>% select(cluster, essay0) %>%  
  group_by(cluster) %>% sample_n(10) %>% kable()
```

Doc2Vec Code

```
#!/usr/bin/env python  
# coding: utf-8  
  
# In[1]:  
  
  
  
from gensim.models.doc2vec import Doc2Vec, TaggedDocument  
from nltk.tokenize import word_tokenize  
import nltk  
nltk.download('punkt')  
import numpy as np  
import pandas as pd  
import warnings  
warnings.filterwarnings('ignore')  
import seaborn as sns  
  
  
  
  
# In[2]:  
  
  
  
import matplotlib.pyplot as plt  
  
  
  
# In[3]:
```

```

df = pd.read_csv('../Data/compressed_okcupid.csv').dropna(subset=['essay0'])

df.info()

# In[4]:


data = df['essay0']


# In[5]:


tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()),
                               tags=[str(i)]) for i, _d in enumerate(df['essay0'])]

# In[6]:


from gensim.models.doc2vec import Doc2Vec

max_epochs = 50
vec_size = 50
alpha = 0.025

model = Doc2Vec(size=vec_size,
                alpha=alpha,
                min_alpha=0.00025,
                min_count=1,
                dm =1)

```

```

model.build_vocab(tagged_data)

for epoch in range(max_epochs):
    print('iteration {}'.format(epoch))
    model.train(tagged_data,
                total_examples=model.corpus_count,
                epochs=model.iter)
    # decrease the learning rate
    model.alpha -= 0.0002
    # fix the learning rate, no decay
    model.min_alpha = model.alpha

model.save("dv_50.model")
print("Model Saved")

# In[7]:


#from gensim.models.doc2vec import Doc2Vec

model= Doc2Vec.load("dv_50.model")
#to find the vector of a document which is not in training data
#test_data = word_tokenize("I love chatbots".lower())
#v1 = model.infer_vector(test_data)
#print("V1_infer", v1)

# to find most similar doc using tags
similar_doc = model.docvecs.most_similar('O')
print(similar_doc)

```

```
# In[8]:
```

```
model.docvecs.vectors_docs.shape  
type(model.docvecs.vectors_docs)
```

```
# In[9]:
```

```
#Visualize TSNE with doc2vec  
  
from sklearn.manifold import TSNE  
  
def doc2vec_tsne_plot(doc_model, labels):  
  
    tokens = []  
    for i in range(len(doc_model.docvecs.vectors_docs)):  
        tokens.append(doc_model.docvecs.vectors_docs[i])  
  
    # Reduce 100 dimensional vectors down into 2-dimensional space so that we can see them  
    tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)  
    new_values = tsne_model.fit_transform(tokens)  
  
    X = [doc[0] for doc in new_values]  
    y = [doc[1] for doc in new_values]  
  
    # Combine data into DataFrame, so that we plot it easily using Seaborn  
    df = pd.DataFrame({'X':X, 'y':y, 'Cuisine':labels})  
    plt.figure(figsize=(16, 16))  
    sns.scatterplot(x="X", y="y", hue="Cuisine", data=df)  
    return  
  
doc2vec_tsne_plot(model, df['height'])
```

```
# In[10]:
```

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

X = model.docvecs.vectors_docs
Sum_of_squared_distances = []
K = range(2,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X)
    Sum_of_squared_distances.append(km.inertia_)
```

```
# In[11]:
```

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Distances from Cluster Centres')
plt.title('Elbow Method For Optimal Number of Clusters', fontsize=15)
plt.savefig('Elbow_Plot.png', bbox_inches='tight')
```

```
# In[12]:
```

```
num_clusters = 4
num_seeds = 4
```

```

max_iterations = 300

labels_color_map = {
    0: '#20b2aa', 1: '#ff7373', 2: '#ffe4e1', 3: '#005073', 4: '#4d0404'
}

pca_num_components = 2

#texts_list = df['essay0']

# calculate tf-idf of texts

#tf_idf_vectorizer = TfidfVectorizer(analyzer="word", use_idf=True,
                                      #smooth_idf=True, ngram_range=(2, 3))

#tf_idf_matrix = tf_idf_vectorizer.fit_transform(texts_list)

# create k-means model with custom config

clustering_model = KMeans(
    n_clusters=num_clusters,
    max_iter=max_iterations,
    precompute_distances="auto",
    n_jobs=-1
)

X = model.docvecs.vectors_docs

reduced_data = PCA(n_components=pca_num_components).fit_transform(X)

labels = clustering_model.fit_predict(reduced_data)

```

In[13]:

```

pca_num_components = 2

X = model.docvecs.vectors_docs

reduced_data = PCA(n_components=pca_num_components).fit_transform(X)

```

```
# In[16]:
```

```
# there appears to be no column named 'clust_label' in df
#df[df['clust_label']==1]['essay0']
```

```
# In[17]:
```

```
from sklearn.manifold import TSNE
# Creating and fitting the tsne model to the document embeddings
tsne_model = TSNE(early_exaggeration=4,
                  n_components=2,
                  verbose=1,
                  random_state=2018,
                  n_iter=300)
tsne_d2v = tsne_model.fit_transform(model.docvecs.vectors_docs)
```

```
# In[18]:
```

```
plt.scatter(tsne_d2v[:, 0], tsne_d2v[:, 1])
plt.show()
plt.savefig('TSNE_blob.png', bbox_inches='tight')
```

```
# In[30]:
```

```
output = pd.DataFrame(model.docvecs.vectors_docs)
```

```
output.to_csv("../Data/doc2vec_results.csv")
output
```

Data Filtering Code

```
#!/usr/bin/env python
# coding: utf-8

# # Data Filter

# ### The purpose of this notebook is four-fold:
# 1) Filter data to only the relevant rows
#
# 2) Delete the unnecessary columns
#
# 3) Suitably edit the text to allow for topic modeling
#
# 4) Create new variables to assist with demographic comparisons of topics
#


# Note:
# URL: https://github.com/rudeboybert/JSE_OkCupid
#
# At 141 MB (unzipped) and 53 MB (zipped), this is too large to be uploaded onto Github.
# This notebook is only required to convert the original data to a manageable size.
# If you already have access to the filtered data, 'compressed_okcupid.csv', then you may proceed to the next step.

# In[1]:


get_ipython().system('pip install pyenchant==1.6.6')
get_ipython().system('pip install compound-word-splitter')
get_ipython().system('pip install pyspellchecker')
```

```
get_ipython().system('pip install wordninja')
```

```
# In[2]:
```

```
get_ipython().system(' pip install spellchecker')
```

```
# In[4]:
```

```
#General Purpose Imports
```

```
import numpy as np
import pandas as pd
import warnings
# import pyenchant
# import splitter
from spellchecker import SpellChecker
warnings.filterwarnings('ignore')
from tqdm import tqdm
tqdm.pandas()
import wordninja
```

```
# In[5]:
```

```
spell = SpellChecker(distance=2)
lolly = ['I', 'love', 'cantilever', 'pecocks']
spell.correction('greatbutt')
```

```
# In[18]:
```

```

import re

def split_word(compound_word):
    """
    Takes in compound word
    Splits it into individual words
    Returns string with spaced words
    """

    sep_words = wordninja.split(compound_word)
    #print('The separated words are {}'.format(sep_words))
    cleaned = ' '.join(sep_words)
    #print(cleaned)

    return cleaned


def decide_split(word):
    spellcheck = SpellChecker()
    if not spellcheck[word]:
        nearest = spellcheck.correction(word)
        #When there is no valid word, the nearest word
        #is the same as the original
        if word == nearest:
            #print('The compound word is {}'.format(word))
            return split_word(word)
        else:
            #print('The accepted word is {}'.format(word))
            #print(nearest)
            return nearest
    else:
        return word

```

```

def split_incorrect(text, punctuation=True):
    """
    Takes in a long string
    The punctuation parameter checks if punctuation marks are to
    be preserved
    Splits into component words, checks for incorrect spellings
    For incorrect spellings, checks if is possibly compound
    If not, then looks for the closest one word in the dictionary
    Returns the entire text with all words corrected
    """

    #if punctuation:

        cleaned_words = []
        words= re.split('\s+|\.|\\?|,', text)
        #print(words)

        for word in words:
            checked_word = decide_split(word)
            #print('The checked word is {}'.format(checked_word))
            cleaned_words.append(checked_word)

        final = " ".join(cleaned_words)
        #print(final)

        return final

    test_text = 'John had a big cock, it crowdedatsunrise every morning'
    real_text = 'i really like meeting new people. small-world networks fascinateme.beer is important. with
    real_text_2 = "i'm an adventurer first, i take calculatedrisks whenever it seems fun, and i always take
    #print(split_incorrect(test_text))
    print(split_incorrect(real_text_2))

# In[17]:

```

```
import wordninja

#wordninja.split('hammerandtongs')

def split_word(compound_word):
    """
    Takes in compound word
    Splits it into individual words
    Returns string with spaced words
    """

    sep_words = wordninja.split(compound_word)
    cleaned = ' '.join(sep_words)

    return cleaned

split_word('missit')
```

```
# In[14]:
```

```
#Imports for Text Analytics
from bs4 import BeautifulSoup
import spacy
import textacy
nlp = spacy.load('en_core_web_sm')
en = textacy.load_spacy_lang("en_core_web_sm")
from textacy import TextStats
```

```
# In[13]:
```

```

df = pd.read_csv('profiles.csv')

must_haves = ['body_type', 'height', 'education', 'ethnicity', 'sex', 'essay0', 'essay9']

df = df.dropna(subset= must_haves)

df = df[(df['sex']=="m")&(df['orientation']=="straight")]

df = df.drop(columns=['essay1', 'essay2', 'essay3', 'essay4', 'essay5', 'essay6', 'essay7',
                      'essay8', 'essay9', 'income', 'job', 'last_online', 'location', 'offspring',
                      'orientation', 'pets', 'religion', 'sex', 'sign', 'smokes', 'speaks', 'status',
                      'diet', 'drinks', 'drugs'])

# #### CREATING NEW COLUMNS

# Many of the sections here are taken directly from the following link, with specific modifications
# Taken directly from:
# https://github.com/UM-CSS/CSSLabs-NLP/blob/master/1_Data_munging.ipynb

# In[18]:


def recode(text, dictionary, default=np.nan):
    '''Function for recoding categories in a column based on exact matches'''

    out = default

    text = str(text)

    for x in dictionary.keys():

        for y in dictionary[x]:
            if y == text: #exact match
                out = x

        return out

    return out

```

```

def recode_fuzzy(text, dictionary, default=np.nan):
    '''Function for recoding categories in a column based on partial matches'''
    out = default
    text = str(text)

    for x in dictionary.keys():
        for y in dictionary[x]:
            if y in text: #partial match
                out = x
        return out

    return out

# In[19]:


ed_levels = {'High School or less': ['dropped out of high school', 'working on high school','graduated from high school'],
             'two-year college', 'dropped out of college/university',
             'high school'],
             'More than High School': ['graduated from college/university',
             'working on masters program', 'working on ph.d program',
             'college/university', 'working on law school',
             'dropped out of masters program',
             'dropped out of ph.d program', 'dropped out of law school',
             'dropped out of med school',
             'graduated from masters program',
             'graduated from ph.d program',
             'graduated from law school',
             'graduated from med school', 'masters program',
             'ph.d program', 'law school', 'med school']}

```

#body type

```
bodies = {'fit': ['fit', 'athletic', 'jacked'],
          'not_fit': ['average', 'thin', 'skinny', 'curvey', 'a little extra',
                      'full figured', 'overweight', 'rather not say', 'used up']}
      }
```

In[20]:

```
df['edu'] = df.education.apply(recode, dictionary=ed_levels,
                               default='unknown')

df['fit'] = df.body_type.apply(recode, dictionary=bodies,
                               default='unknown')
```

In[21]:

```
# race/ethnicity for exact matching

ethn = {'White': ['white', 'middle eastern', 'middle eastern, white'],
        'Asian': ['asian', 'indian', 'asian, pacific islander'],
        'Black': ['black']}
```

race/ethnicity for fuzzy matching

```
ethn2 = {'Latinx': ['latin'], 'multiple': ['.', ''], np.nan: ['nan']}
```

In[22]:

```
def census_2010_ethnicity(t):
    text = str(t)
```

```

e = recode(text, ethn, default='other')

if 'other' == e:
    e = recode_fuzzy(text, ethn2, default='other')

return e

df['race_ethnicity'] = df.ethnicity.apply(census_2010_ethnicity)

# In[23]:


def height_check(inches):
    h = 'not_short'
    if inches <= 69:
        h = 'short'
    return h

df['height'] = pd.to_numeric(df['height'])

df['height_group'] = df.height.apply(height_check)

# ## TEXT EDITING

# In[24]:


# Some of the essays have just a link in the text. BeautifulSoup sees that and gets
# the wrong idea. This line hides those warnings.

warnings.filterwarnings("ignore", category=UserWarning, module='bs4')

def clean(text):

    if pd.isnull(text):
        t = np.nan
    else:

```

```

t = BeautifulSoup(text, 'lxml').get_text()
t = t.lower()
t = t.strip().replace('\n', '').replace("\r", " ").replace('\t', '')
bad_words = ['http', 'www', '\nan']

for b in bad_words:
    t = t.replace(b, '')

if t == '':
    t = np.nan

return t

```

In[25]:

```

#Clearing out all HTML and unnecessary characters
df['essay0'] = df['essay0'].progress_apply(clean)
df['essay9'] = df['essay9'].progress_apply(clean)

```

In[33]:

#These functions help assess the level of complexity

```

def get_flesch(text):
    doc = textacy.make_spacy_doc(text, lang=en)
    ts = TextStats(doc)
    try:
        return ts.flesch_kincaid_grade_level
    except ZeroDivisionError:
        return (11.8 * ts.n_syllables) + (0.39 * ts.n_words) - 15.59

```

```
def get_npoly(text):  
    doc = textacy.make_spacy_doc(text, lang=en)  
    ts = TextStats(doc)  
    return ts.n_polysyllable_words
```

In[28]:

```
df['long_words'] = df['essay0'].progress_apply(get_npoly)
```

In[30]:

```
df['flesch'] = df['essay0'].progress_apply(get_flesch)
```

In[38]:

```
df['long_words'].describe()
```

In[39]:

```
df['flesch'].describe()
```

In[35]:

```
df.to_csv('compressed_okcupid.csv')
```

```
# In[ ]:
```

```
pip3 install pyenchant==1.6.6
```

Structural topic modeling Code

```
## ----setup,  
## include=FALSE-----  
knitr::opts_chunk$set(echo = FALSE)  
  
library(stm)  
  
library(quanteda)  
  
library(topicmodels)  
  
library(tidytext)  
  
library(ggplot2)  
  
library(dplyr)  
  
library(tidyr)  
  
library(scales)  
  
library(tm)  
  
library(grid)  
  
library(wordcloud)  
  
library(wordcloud2)  
  
library(tidyverse)  
  
library(igraph)  
  
library(stmCorrViz)  
  
  
## ----data,
```

```

## echo=FALSE-----
setwd("C:/Users/lliu9/Desktop/UML_Project/unsupervised-dating/Data")

# cleaned okcupid data
data <- read_csv("final_okcupid.csv")

# data <- sample_n(essay, 100)

data <- data[c("clean_text", "fit", "edu", "height_group",
  "race_ethnicity", "dbscan_cluster")]

## ----clean-----
processed <- textProcessor(data$clean_text, metadata = data)
out <- prepDocuments(processed$documents, processed$vocab,
  processed$meta, lower.thresh = 0)
docs <- out$documents
vocab <- out$vocab
meta <- out$meta

## ----stm-----
Fit <- stm(documents = out$documents, vocab = out$vocab,
  K = 9, prevalence = ~fit + edu + height_group +
  race_ethnicity + dbscan_cluster, max.em.its = 50,
  data = out$meta, init.type = "Spectral", verbose = FALSE)

## ----label-----
labelTopics(Fit, c(1:9))

## ----estimate-----
prep <- estimateEffect(~fit + edu + height_group +

```

```

race_ethnicity + dbscan_cluster, Fit, meta = out$meta,
uncertainty = "Global")

## ----print-----
for (i in seq(1:9)) {
  print(summary(prep, topic = i))
}

## ----summary-----
## jpeg('toptopics.jpeg')
plot(Fit, type = "summary", xlim = c(0, 0.3))
# dev.off()

## ----labels-----
jpeg("top3topics.jpeg")
plot(Fit, type = "labels", topics = c(5, 4, 7))
dev.off()

## ----labels2-----
jpeg("bottom3topics.jpeg")
plot(Fit, type = "labels", topics = c(1, 2, 9))
dev.off()

## ----hist-----
jpeg("hist.jpeg")
plot(Fit, type = "hist")
dev.off()

```

```

## ----perspectives-----
jpeg("comparison.jpeg")
plot(Fit, type = "perspectives", topics = c(6, 8))
dev.off()

## ----quality-----
jpeg("topicQuality.jpeg")
topicQuality(model = Fit, documents = docs)
dev.off()

## ----word-cloud-----
cloud(Fit, topic = 1)

## ----cloud-----
for (i in seq(1:9)) {
  name <- paste("topic", str(i), ".jpg", sep = "")
  jpeg(name)
  cloud(Fit, topic = i, scale = c(4, 0.2))
  dev.off
}

## ----network-----
## Positive correlations between topics indicate
## that both topics are likely to be discussed
## within a document.

mod.out.corr <- topicCorr(Fit)

```

```
# Graphical display of topic correlations.  
jpeg("network.jpeg")  
plot(mod.out.corr)  
dev.off()
```

References

- Bruning, Roger H, Gregory J Schraw, and Royce R Ronning. 1999. *Cognitive Psychology and Instruction*. ERIC.
- Cacioppo, John T, Stephanie Cacioppo, Gian C Gonzaga, Elizabeth L Ogburn, and Tyler J VanderWeele. 2013. “Marital Satisfaction and Break-Ups Differ Across on-Line and Off-Line Meeting Venues.” *Proceedings of the National Academy of Sciences* 110 (25): 10135–40.
- Goffman, E. 1975. *The Presentation of Self in Everyday Life*. Pelican Book. Penguin Books. <https://books.google.com/books?id=tYvNnQEACAAJ>.
- Heino, Rebecca D, Nicole B Ellison, and Jennifer L Gibbs. 2010. “Relationshopping: Investigating the Market Metaphor in Online Dating.” *Journal of Social and Personal Relationships* 27 (4): 427–47.
- Hitsch, Gunter J, Ali Hortaçsu, and Dan Ariely. 2010. “Matching and Sorting in Online Dating.” *American Economic Review* 100 (1): 130–63.
- Kim, Albert Y, and Adriana Escobedo-Land. 2015. “OkCupid Data for Introductory Statistics and Data Science Courses.” *Journal of Statistics Education* 23 (2).
- Papacharissi, Zizi. 2010. *A Networked Self: Identity, Community, and Culture on Social Network Sites*. Routledge.