STA 380 Homework 1

Nicole Erich, Anying Li, Daniel Peng & Rachel Wang

August 5, 2016

Probability practice

Part A.

Based on law of total probability: P(Y) = P(Y|RC) * P(RC) + P(Y|TC) * P(TC)

Here: P(Y) = 0.65 P(Y|RC) = 0.5 P(RC) = 0.3 P(TC) = 0.7

Therefore: P(Y|TC) = 0.7142857

So about 71.43% of people who are truthful clickers answered yes.

Part B.

Based on Bayes' Rule: P(A|B) = [P(A) * P(B|A)]/ P(B)

Here:

Event A is someone has the disease; event B is someone's test result is positive. We want to know P(A|B).

P(A) = 0.000025 P(B|A) = 0.993 We can calculate P(B) based on law of total probability: P(B) = P(B|A) * P(A) + P(B|not A) * P(not A) = 0.993 * 0.000025 + (1 - 0.9999) * (1 - 0.000025) = 0.0001248225

So P(A|B) = (0.000025 * 0.993) / 0.0001248225 = 0.1988824

Q1 Exploratory analysis: green buildings

```
all.buildings = read.csv('https://raw.githubusercontent.com/jgscott/STA380/master/data/greenbuildings.c
green.buildings = subset(all.buildings,all.buildings$Energystar == 1 | all.buildings$LEED == 1)
not.green = all.buildings[!(all.buildings$CS_PropertyID %in% green.buildings$CS_PropertyID),]
```

To estimate the economic impact of a green certificate, we had to calculate the expected extra profit brought in with it. To do so, We needed to find out the additional cost and revenue associated with a green building. The extra cost for this property is the $5 Million premium ($100M x 5%). The extra revenue per year would be additional rent/sqft-year x size of the building (250,000 sqft). In different clusters, we might value a green certificate differently. Therefore, we could not simply find the median rent for regular buildings and green buildings and subtract one from the other. Naturally, we were going to find the difference between two types of buildings by clusters. In this case, we treated all the regular buidlings in a certain cluster as control group so we could see the effect of a certificate.

```
#calculate the average rent within each cluster
notgreen.mean.rent = aggregate(not.green$Rent, list(not.green$cluster), mean)
green.mean.rent = aggregate(green.buildings$Rent, list(green.buildings$cluster), mean)

#remove the clusters without any green buildings
```

```r
notgreen.mean.rent = notgreen.mean.rent[which(notgreen.mean.rent[,1] %in% green.buildings$cluster),]

#remove the clusters withou any regualr buildings
green.mean.rent = green.mean.rent[which(green.mean.rent[,1] %in% notgreen.mean.rent[,1]),]

rent.diff = green.mean.rent - notgreen.mean.rent
boxplot(rent.diff$x, ylab = '$', xlab = 'green building rent premium')
```

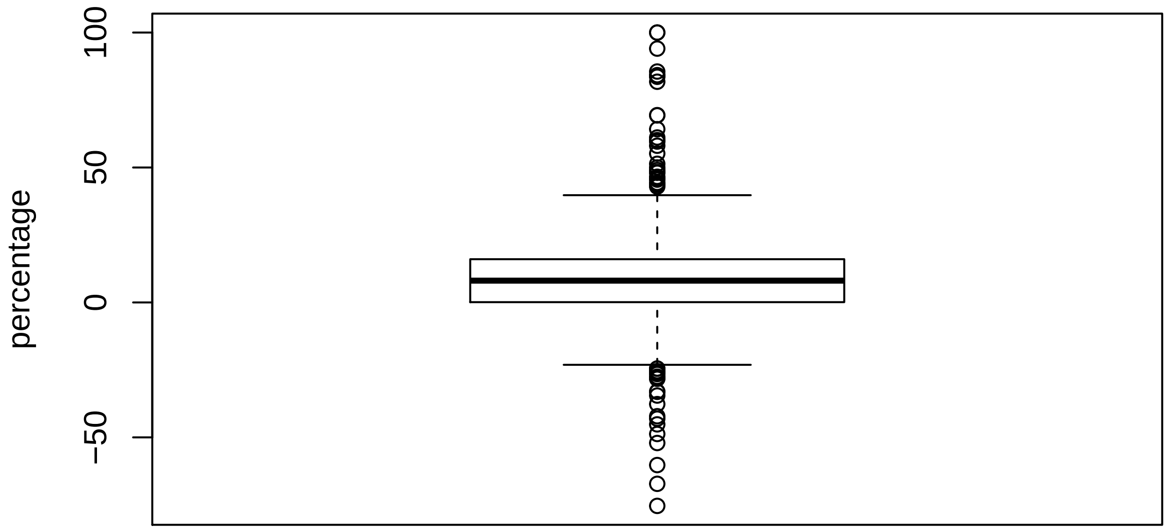

```r
rent = median(rent.diff$x)
c('expected additional rent:', rent)
```

```
## [1] "expected additional rent:" "2.66"
```

From the boxplot above, we can see the majority of the rent premium is concentrated around $2, with many outliers. So It's better to use median of $2.66 for estimating the new building.

We used the same approach to calculate the exptected occupancy rate change for the new property.

```r
#calculate the average leasing rate within each cluster
notgreen.mean.lr = aggregate(not.green$leasing_rate, list(not.green$cluster), mean)
green.mean.lr = aggregate(green.buildings$leasing_rate, list(green.buildings$cluster), mean)

#remove the clusters without any green buildings
notgreen.mean.lr = notgreen.mean.lr[which(notgreen.mean.lr[,1] %in% green.buildings$cluster),]

#remove the clusters without any green buildings
green.mean.lr = green.mean.lr[which(green.mean.lr[,1] %in% notgreen.mean.lr[,1]),]

lr.diff = green.mean.lr - notgreen.mean.lr
boxplot(lr.diff$x, ylab = 'percentage', xlab = 'leasing rate difference between green vs non-green buil
```



leasing rate difference between green vs non−green building

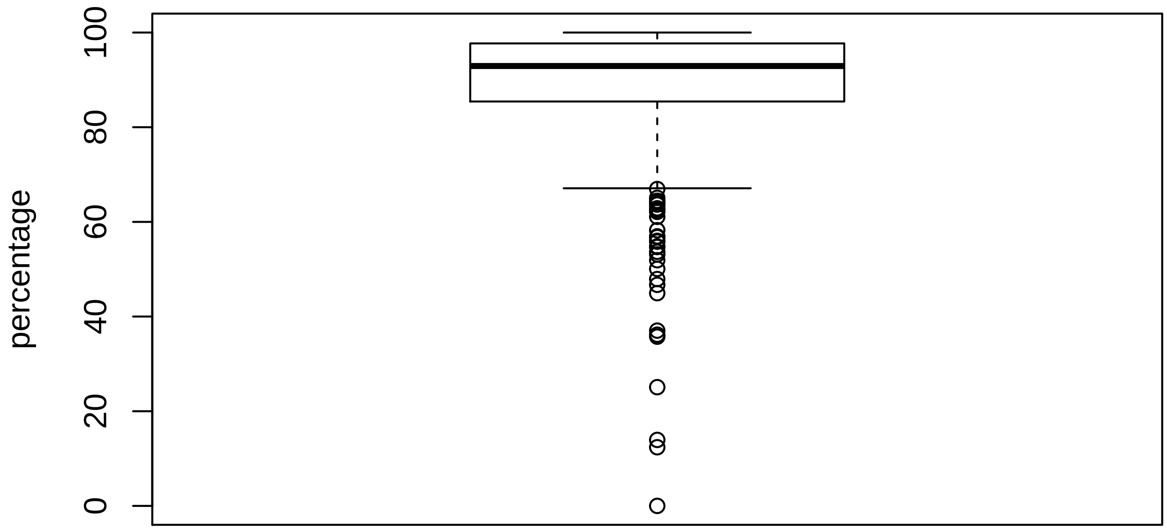
```
lr = median(lr.diff$x)
c('expected additional occupancy:', lr)
```

```
## [1] "expected additional occupancy:" "8.08"
```

Interestingly, the distribution of the occupancy difference is very concentrated around 10. It seems people prefer green buildings.

In order to find a good estimate for the occupancy rate for the new building, we plotted the distribution of all existing green buildings. The median of 92.93 looked like a good choice based on the boxplot.

```
boxplot(green.mean.lr[,2], ylab = 'percentage', xlab = 'green building leasing rate')
```



green building leasing rate

In conclusion, our finding is very similar to the original analysis even though with different approach: rent premium = $2.66, occupancy rante = 92.93%.

```
c('Number of Years to Break Even: ',5000000/(2.66 * 250000 * .9293))
```

```
## [1] "Number of Years to Break Even: " "8.0908178117736"
```

```
c('Additional Annual Revenue for Green Certification:', (2.66 * 250000 * .9293))
```

```
## [1] "Additional Annual Revenue for Green Certification:"
## [2] "617984.5"
```

Q2 Bootstrapping

et up and create the function for calculating percent returns.

```
rm(list=ls())
library(mosaic)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: lattice

## Loading required package: ggplot2

## Loading required package: mosaicData

## Loading required package: Matrix

##
## The 'mosaic' package masks several functions from core packages in order to add additional features.
## The original behavior of these functions should not be affected by this.

##
## Attaching package: 'mosaic'

## The following object is masked from 'package:Matrix':
##
##     mean

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cov, D, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```r
library(fImport)
```

```
## Loading required package: timeDate

## Loading required package: timeSeries
```

```r
library(foreach)
my_favorite_seed = 1234567
set.seed(my_favorite_seed)
YahooPricesToReturns = function(series) {
    mycols = grep('Adj.Close', colnames(series))
    closingprice = series[,mycols]
    N = nrow(closingprice)
    percentreturn = as.data.frame(closingprice[2:N,]) / as.data.frame(closingprice[1:(N-1),]) - 1
```

```
    mynames = strsplit(colnames(percentreturn), '.', fixed=TRUE)
    mynames = lapply(mynames, function(x) return(paste0(x[1], ".PctReturn")))
    colnames(percentreturn) = mynames
    as.matrix(na.omit(percentreturn))
 }
```

Import the 5 asset classes and calculate their repsective standard deviations.

```
mystocks = c("SPY",'TLT','LQD','EEM','VNQ')
myprices = yahooSeries(mystocks, from='2011-01-01', to='2016-08-03')
myreturns = YahooPricesToReturns(myprices)
sigma_SPY = sd(myreturns[,1])
sigma_SPY
```

```
## [1] 0.009650712
```

```
sigma_TLT = sd(myreturns[,2])
sigma_TLT
```

```
## [1] 0.009385556
```

```
sigma_LQD = sd(myreturns[,3])
sigma_LQD
```

```
## [1] 0.003428545
```

```
sigma_EEM = sd(myreturns[,4])
sigma_EEM
```

```
## [1] 0.01411309
```

```
sigma_VNQ = sd(myreturns[,5])
sigma_VNQ
```

```
## [1] 0.01136797
```

The standard deviations show that Emerging-market equities (EEM) and Real estate (VNQ) are the most volatile asset classes. We construct an aggressive portfolio with a 50/50 split between EEM and VNQ, a safe portfolio with the other three asset classes, and an even split by distributing 20% of money in each of the five ETFs.

```
weights_safe = c(0.3, 0.3, 0.4, 0.0, 0.0)
weights_even = c(0.2, 0.2, 0.2, 0.2, 0.2)
weights_aggressive = c(0.0, 0.0, 0.0, 0.5, 0.5)
```

Use bootstrap resampling to estimate 4-week VaR of each of the three portfolios at the 5% level.

```r
n_days = 20
wealth_tracker_safe = rep(0, 5000)
wealth_tracker_even = rep(0, 5000)
wealth_tracker_aggressive = rep(0, 5000)
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
    totalwealth = 100000
    holdings_safe = weights_safe * totalwealth
    holdings_even = weights_even * totalwealth
    holdings_aggressive = weights_aggressive * totalwealth
    for(today in 1:n_days) {
        return.today = resample(myreturns, 1, orig.ids=FALSE)
        holdings_safe = holdings_safe + holdings_safe*return.today
        holdings_even = holdings_even + holdings_even*return.today
    holdings_aggressive = holdings_aggressive + holdings_aggressive*return.today
        totalwealth_safe = sum(holdings_safe)
        totalwealth_even = sum(holdings_even)
        totalwealth_aggressive = sum(holdings_aggressive)
        holdings_safe = weights_safe * totalwealth_safe
        holdings_even = weights_even * totalwealth
    holdings_aggressive = weights_aggressive * totalwealth_aggressive
    }
    wealth_tracker_safe[i]=totalwealth_safe
    wealth_tracker_even[i]=totalwealth_even
    wealth_tracker_aggressive[i]=totalwealth_aggressive
}
```

```r
var_safe = quantile(wealth_tracker_safe, 0.05) - 100000
var_safe
```

```
##        5%
## -2045.256
```

```r
var_even = quantile(wealth_tracker_even, 0.05) - 100000
var_even
```

```
##        5%
## -931.2584
```

```r
var_aggressive = quantile(wealth_tracker_aggressive, 0.05) - 100000
var_aggressive
```

```
##        5%
## -7828.275
```

The VaR analysis tells us that the theoretically safer portfolio is in fact more volatile than the even split. One reasonable explanation is that the risk of a portfolio is decided by its overall diversity rather than the standalone volatility of its components. Therefore, event split (best diversity) outperforms the "safer" portfolio comprised by the "safer" asset classes.

Q3 Market segmentation

Step 1. In terms of data processing, first we get the frequency of each catogory by dividing each data by corresponding row sum, so we can get the relative weight of each category in each twitter. Then we try to limit the noise from categories such as "chatter","spam","adult" and "uncategorized" because these are not really user twitter with helpful information. We decide to get rid of the records if the sum of the frequency of the four noise categories is bigger than 0.5. In the end, we scale and centralize the data to get the data prepared for clustering.

```
library(cluster)
library(fpc)
library(flexclust)
```

```
## Loading required package: grid

## Loading required package: modeltools

## Loading required package: stats4
```

```
library(foreach)
library(ggplot2)

social_marketing <- read.csv("https://raw.githubusercontent.com/jgscott/STA380/master/data/social_marke

# Center/scale the data
X_freq = social_marketing/rowSums(social_marketing)
dim(X_freq)
```

```
## [1] 7882   36
```

```
any(is.na(X_freq))
```

```
## [1] FALSE
```

```
#take the record out if frequency of uninformative category is over 0.5
X_freq['sum'] = X_freq$chatter +X_freq$spam +X_freq$adult +X_freq$uncategorized
X_freq = subset(X_freq, X_freq$sum <= 0.5)
dim(X_freq)
```

```
## [1] 7849   37
```

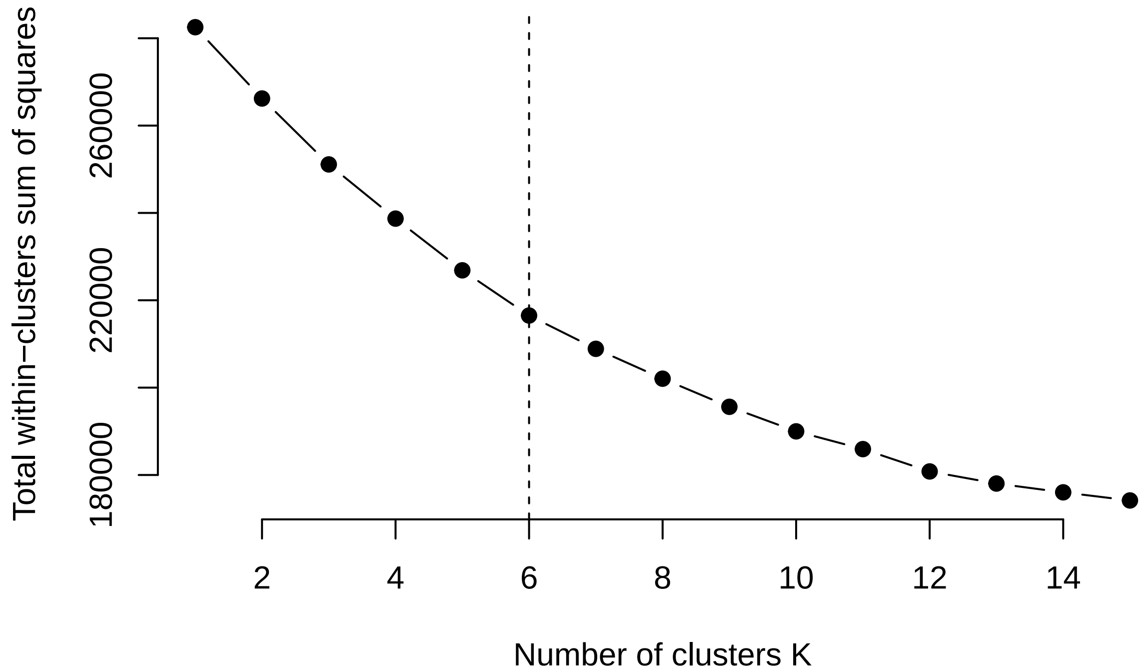
```
X_freq <- X_freq[,-37]

social_marketing_scaled  = scale(X_freq, center=TRUE, scale=TRUE)
```

Step 2. We decide to begin with a k-mean clustering to see whether we can find something interesting. To use k-mean clustering, finding the appropriate k is our first task. To get the optimal k, we tried two approaches. First approach is to calculate the within group sum of square(wss). It's obvious that when k increases, wss will keep decreasing. But after plotting the relationship between k and wss, we can see that with 6 clusters the wss has already decreased significantly. As a second approach, we also calculated CH index that we discussed at class to decide the optimal k. For the seed we set, we get a optimal k equal to 12. But we should also notice that after 6 clusters, CH index does not improve significantly as k keeps increasing. Meanwhile, too many clusters can make it hard to intepret the meaning behind the clusering. Therefore, we decide to use k = 6 for the clustering.

```
#decide optimal k based on within group sum of square
k.max <- 15  # Maximal number of clusters
data <- social_marketing_scaled

# Compute and plot wss for k = 2 to k = 15
set.seed(1234567)
wss <- sapply(1:k.max,
        function(k){kmeans(data, k, nstart=50 )$tot.withinss})
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
plot(1:k.max, wss,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
abline(v = 6, lty =2)
```

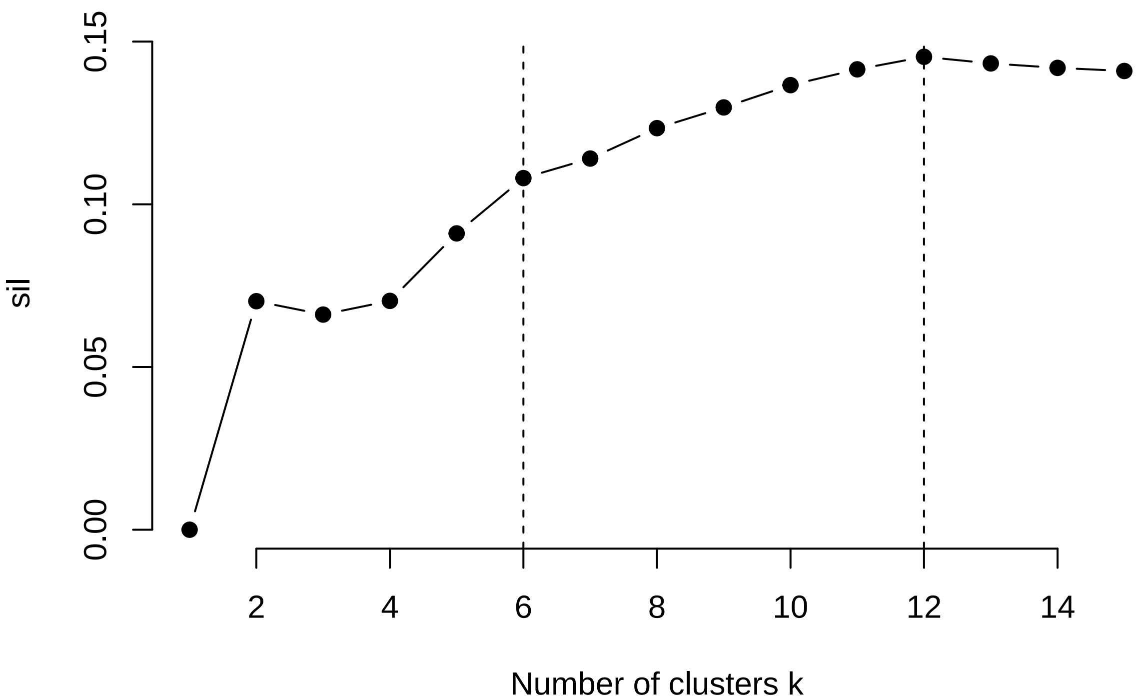



```
#6 clusters are suggested

#decide optimal k based on within CH index/Average silhouette method
sil <- rep(0, k.max)

# Compute the average silhouette width for k = 2 to k = 15
for(i in 2:k.max){
  set.seed(1234567)
  km.res <- kmeans(data, centers = i, nstart = 50)
  ss <- silhouette(km.res$cluster, dist(data))
  sil[i] <- mean(ss[, 3])
}
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
# Plot the  average silhouette width
plot(1:k.max, sil, type = "b", pch = 19,
     frame = FALSE, xlab = "Number of clusters k")
abline(v = which.max(sil), lty = 2)
abline(v = 6, lty =2)
```




```
# 12 clusters are suggested
```

Step3: After deciding the optimal k, we tried to build the clustering with both kmean and kmean++ for initialization, and it turns out that the error from these two methods are very close. It might be related with the seed we choose, and as we set nstart = 50 for kmean, it also helps imporve the accuracy of kmean clustering.

After looking through the centers of the 6 clusters, we find:

second cluster has high positive weight on sports_fandom,food,family,religion,parenting and school,so this group may include married people who pay more attention to their family and parenting related topic.

third cluster has high positive weight on cooking,beauty and fashion,so this group should consist of younger woman and younger housewives who cook a lot and pay a lot of attention to beauty and fashion.

fourth cluster has high positive weight on politics,news,travel,computer and automotive,so this group might be younger man who are interested in politics, read lots of news online, loves computer and automotive and travels a lot.

fifth cluster has high positive weight on health_nutrition, outdoors and personal_fitness, so this group of people really care about nutrition and fitness, and they have lots of outdoor activity to keep fit.

sixth cluster has high positive weight on online_gaming,college_uni and sports_playing,this group is very likely to be college student whose main entertainment is online_gaming and sports.

first cluster's highest positive weight is on chatter, and nothing else really stands out. So we think this cluster may be just everything that is left out and is hard to get into any of the other market segment.

```
# fit cluster model with 6 centers
set.seed(1234567)
cluster_6 <- kmeans(data, centers=6, nstart=50)
names(cluster_6)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
cluster_6$centers
```

```
##      chatter current_events      travel photo_sharing uncategorized
## 1 -0.3837312     -0.2108030 -0.18108941    -0.3361177   -0.05627069
## 2 -0.4339944     -0.2137770 -0.27376862    -0.4081966   -0.11315049
## 3 -0.4362083     -0.1669467 -0.28527970    -0.4193508   -0.21586985
## 4 -0.4570848     -0.1835754 -0.24617030     0.4358365    0.02503411
## 5 -0.3667735     -0.1015367  0.96771010    -0.4778611   -0.14669241
## 6  0.8083460      0.3361859 -0.04272988     0.5508299    0.22049860
##       tv_film sports_fandom   politics       food      family
## 1  0.19171228    -0.2934869 -0.3490226 -0.2142310 -0.09722020
## 2 -0.24938308    -0.3491980 -0.3302150  0.1256574 -0.25737526
## 3 -0.19392881     1.3885068 -0.3803047  1.2437855  0.81916145
## 4 -0.26599694    -0.4041387 -0.3604385 -0.4221893 -0.23678827
## 5 -0.08841854     0.1186687  1.7171596 -0.2293812 -0.09206439
## 6  0.27848491    -0.2577476 -0.2242795 -0.3011719 -0.07515292
##   home_and_garden      music       news online_gaming   shopping
## 1     -0.10447006  0.1318158 -0.3117212     2.3187349 -0.3735883
## 2     -0.09614825 -0.1361084 -0.1920087    -0.2121673 -0.2872899
## 3     -0.07896679 -0.1159392 -0.2653374    -0.2217606 -0.3069829
## 4     -0.12161566  0.1260787 -0.2868907    -0.2006455 -0.1782972
## 5     -0.05648961 -0.1676258  1.5227455    -0.2493419 -0.3689054
## 6      0.17212577  0.1184911 -0.2900471    -0.2418014  0.5919211
##   health_nutrition college_uni sports_playing    cooking         eco
## 1       -0.3685201   2.4361909     1.19273070 -0.3423883 -0.19162232
## 2        1.7709001  -0.3348953    -0.15758760  0.2118675  0.11849306
## 3       -0.3518720  -0.2973465    -0.15077221 -0.3471649 -0.04517358
## 4       -0.3035556  -0.2314739    -0.05938603  2.1879569 -0.20040270
## 5       -0.4049666  -0.2690893    -0.15882567 -0.3973473 -0.16046545
## 6       -0.3858046  -0.1603625    -0.08417465 -0.3663041  0.14143868
##     computers    business    outdoors     crafts  automotive          art
## 1 -0.20964361 -0.13506862 -0.24498869 -0.1739022 -0.18221011  0.009683922
## 2 -0.18708480 -0.15157659  1.08848837 -0.1213240 -0.28218207 -0.125017628
## 3 -0.09822232 -0.09561711 -0.25615098  0.2371491 -0.09281254 -0.072296995
## 4 -0.16071024 -0.03464582 -0.18582091 -0.1735611 -0.21043650 -0.090258403
## 5  0.76748918 -0.01639691 -0.01674134 -0.1374715  0.88205766 -0.189787115
## 6 -0.09624176  0.17122869 -0.31786781  0.1217041 -0.09250004  0.202846063
##     religion      beauty  parenting      dating      school
```

```
## 1 -0.2604777 -0.29058793 -0.3046890 -0.13145477 -0.33414491
## 2 -0.2926531 -0.28638240 -0.2667258  0.01814441 -0.29427076
## 3  1.6900537  0.06817419  1.5442861 -0.16015488  1.01732114
## 4 -0.3031804  1.76497590 -0.2809682 -0.13030015 -0.09246893
## 5 -0.2271477 -0.28854443 -0.1105381 -0.02522172 -0.20215764
## 6 -0.3011191 -0.21932366 -0.2993223  0.14411013 -0.07279344
##   personal_fitness    fashion small_business        spam       adult
## 1       -0.3646911 -0.2549782   -0.001005938  0.02812103 -0.06435910
## 2        1.6434108 -0.2610103   -0.183707292 -0.03294390 -0.07653893
## 3       -0.3149193 -0.1759703   -0.075352035 -0.04088964 -0.04968130
## 4       -0.3145744  2.0056300   -0.063076878 -0.04318552 -0.08976994
## 5       -0.3886946 -0.3366833   -0.055735098 -0.02648512 -0.08902164
## 6       -0.3411691 -0.1913383    0.169559661  0.05146539  0.14371580
```

```
head(sort(cluster_6$centers[1,], decreasing=TRUE), 10)
```

```
##    college_uni  online_gaming sports_playing        tv_film          music
##    2.436190862    2.318734947    1.192730698    0.191712282    0.131815757
##           spam            art small_business  uncategorized          adult
##    0.028121027    0.009683922   -0.001005938   -0.056270690   -0.064359098
```

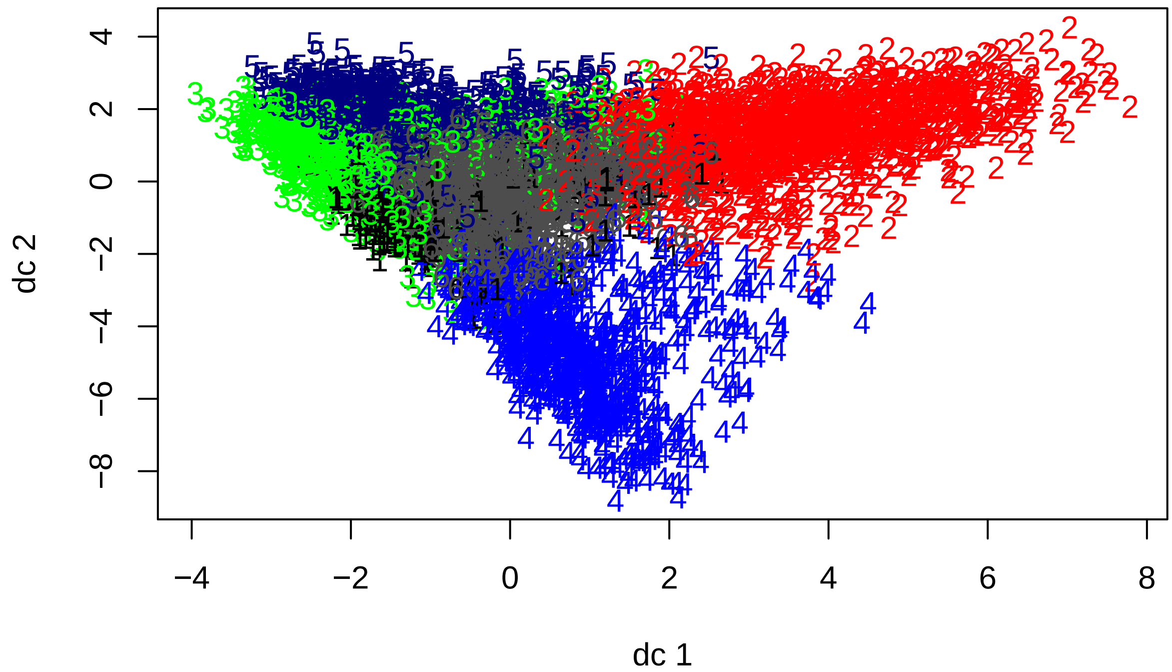
```
head(sort(cluster_6$centers[2,], decreasing=TRUE), 10)
```

```
## health_nutrition personal_fitness         outdoors          cooking
##       1.77090014       1.64341084       1.08848837       0.21186751
##             food              eco           dating             spam
##       0.12565742       0.11849306       0.01814441      -0.03294390
##            adult  home_and_garden
##      -0.07653893      -0.09614825
```

```
head(sort(cluster_6$centers[3,], decreasing=TRUE), 10)
```

```
##      religion     parenting sports_fandom          food        school
##    1.69005373    1.54428612    1.38850676    1.24378546    1.01732114
##        family        crafts        beauty          spam           eco
##    0.81916145    0.23714909    0.06817419   -0.04088964   -0.04517358
```

```
plotcluster(data, cluster_6$cluster)
```




```
summary(cluster_6)
```

```
##              Length Class  Mode
## cluster      7849   -none- numeric
## centers       216   -none- numeric
## totss           1   -none- numeric
## withinss        6   -none- numeric
## tot.withinss    1   -none- numeric
## betweenss       1   -none- numeric
## size            6   -none- numeric
## iter            1   -none- numeric
## ifault          1   -none- numeric
```

```
cluster_6$tot.withinss
```

```
## [1] 216511
```

```
# use kmean++ for clustering initialization
set.seed(123)
cluster_kmeansPP = cclust(data, k=6, control=list(initcent="kmeanspp"))
```

```
## Found more than one class "kcca" in cache; using the first, from namespace 'kernlab'
## Found more than one class "kcca" in cache; using the first, from namespace 'kernlab'
```

```
parameters(cluster_kmeansPP)
```

```
## Found more than one class "kcca" in cache; using the first, from namespace 'kernlab'
```

```
##         chatter current_events      travel photo_sharing uncategorized
## [1,] -0.3501608    -0.01499684 -0.05722007    -0.3558094    0.16234557
## [2,] -0.4409824    -0.14597170 -0.28299482    -0.4244248   -0.19939911
## [3,]  1.0741199     0.37351013 -0.08421317     0.8918532    0.13148004
## [4,] -0.3817482    -0.08836128  0.96096527    -0.4879916   -0.14753243
## [5,]  0.3838886    -0.10991406 -0.17527064    -0.3454694    0.26606373
## [6,] -0.4729166    -0.21541104 -0.25799758    -0.1132325   -0.07257773
##          tv_film sports_fandom   politics        food       family
## [1,]  0.95225957    -0.2634359 -0.3126114 -0.13556094 -0.174417047
## [2,] -0.18891548     1.3944322 -0.3803793  1.24708073  0.812576704
## [3,] -0.10148613    -0.2478624 -0.1868494 -0.36733918 -0.001533922
## [4,] -0.07911577     0.1188571  1.6848121 -0.24537197 -0.085534781
## [5,] -0.19515933    -0.3637056 -0.2798609 -0.34060771 -0.251642013
## [6,] -0.26162064    -0.3813797 -0.3598248 -0.04455022 -0.256628146
##      home_and_garden       music       news online_gaming   shopping
## [1,]      0.04173941  0.37446597 -0.2736503     1.3217010 -0.3234939
## [2,]     -0.08066413 -0.10908600 -0.2653342    -0.2271706 -0.3141034
## [3,]      0.12156932  0.03263285 -0.3136961    -0.2503472  0.8939454
## [4,]     -0.04238565 -0.17026960  1.5244505    -0.2417953 -0.3764808
## [5,]      0.11140532 -0.21388475 -0.2310183    -0.1578970 -0.3291675
## [6,]     -0.09492986 -0.05242260 -0.2331256    -0.2143617 -0.2627523
##      health_nutrition college_uni sports_playing    cooking         eco
## [1,]       -0.4078003   1.6272346     0.79360485 -0.3171857 -0.15801379
## [2,]       -0.3491096  -0.3056514    -0.15349342 -0.3385289 -0.04634232
## [3,]       -0.3791366  -0.2441802    -0.13523739 -0.3294268  0.21217071
## [4,]       -0.3711231  -0.2777735    -0.17280931 -0.3874233 -0.16891296
## [5,]       -0.2941602  -0.2247665     0.05124328 -0.3007900 -0.11737999
## [6,]        1.0729447  -0.3203662    -0.14243433  0.9758396  0.02158752
##        computers    business     outdoors      crafts   automotive
## [1,] -0.23835731  0.02309149 -0.260998947  0.02463569 -0.255819524
## [2,] -0.09436667 -0.09098105 -0.260533678  0.24027580 -0.089834426
## [3,] -0.05716320  0.15492124 -0.351864036  0.05032262 -0.004196339
## [4,]  0.76788984 -0.02360072  0.007706184 -0.14291931  0.894689517
## [5,] -0.08931371  0.15480875 -0.135262301  0.03377071 -0.271780691
## [6,] -0.18712317 -0.13012067  0.664835142 -0.11979289 -0.276542952
##              art   religion      beauty  parenting     dating     school
## [1,]  0.69370732 -0.2115014 -0.20603227 -0.3476384 -0.1898712 -0.2849086
## [2,] -0.07689029  1.6801603  0.07499968  1.5377590 -0.2149842  1.0184294
## [3,] -0.14106991 -0.3456032 -0.22536283 -0.2848370 -0.2350152 -0.1566664
## [4,] -0.17181440 -0.2298729 -0.29244394 -0.1170041 -0.0967113 -0.2163188
## [5,] -0.09123341 -0.1806937  0.15591244 -0.1380454  3.8320173  0.7329884
## [6,] -0.09865064 -0.2980649  0.44765725 -0.2749978 -0.1144050 -0.2382919
##      personal_fitness    fashion small_business        spam       adult
## [1,]       -0.3813658 -0.1912280     0.22164207  0.03061494 -0.04254122
## [2,]       -0.3099154 -0.1813723    -0.07825804 -0.03197029 -0.01629351
## [3,]       -0.3262126 -0.2239218     0.08251367  0.05149493  0.08281286
## [4,]       -0.3592700 -0.3488453    -0.07836819 -0.02069886 -0.03580816
## [5,]       -0.2983826  0.5948847     0.08300706 -0.04883438  0.02420529
## [6,]        0.9767734  0.5398863    -0.13462788 -0.03134730 -0.03272741
```

```
cluster_kmeansPP@clusinfo
```

```
##   size  av_dist max_dist separation
## 1 1161 5.581387 33.70702   3.506096
```

```
## 2 1120 4.828279 13.39611   3.498190
## 3 2029 5.165092 33.29725   3.096357
## 4 1181 5.151554 14.40995   3.496701
## 5  335 4.772608 10.41878   3.802397
## 6 2023 4.839438 17.06977   3.186441
```

```
print(apply(parameters(cluster_kmeansPP),1,function(x) colnames(data)[order(x, decreasing=TRUE)[1:10]]))
```

```
##       [,1]              [,2]             [,3]              [,4]
##  [1,] "college_uni"     "religion"       "chatter"         "politics"
##  [2,] "online_gaming"   "parenting"      "shopping"        "news"
##  [3,] "tv_film"         "sports_fandom" "photo_sharing"   "travel"
##  [4,] "sports_playing"  "food"           "current_events"  "automotive"
##  [5,] "art"             "school"         "eco"             "computers"
##  [6,] "music"           "family"         "business"        "sports_fandom"
##  [7,] "small_business"  "crafts"         "uncategorized"   "outdoors"
##  [8,] "uncategorized"   "beauty"         "home_and_garden" "spam"
##  [9,] "home_and_garden" "adult"          "adult"           "business"
## [10,] "spam"            "spam"           "small_business"  "adult"
##       [,5]              [,6]
##  [1,] "dating"          "health_nutrition"
##  [2,] "school"          "personal_fitness"
##  [3,] "fashion"         "cooking"
##  [4,] "chatter"         "outdoors"
##  [5,] "uncategorized"   "fashion"
##  [6,] "beauty"          "beauty"
##  [7,] "business"        "eco"
##  [8,] "home_and_garden" "spam"
##  [9,] "small_business"  "adult"
## [10,] "sports_playing"  "food"
```

```
# Roll our own function
centers = parameters(cluster_kmeansPP)
kpp_residualss = foreach(i=1:nrow(data), .combine='c') %do% {
    x = data[i,]
    a = cluster_kmeansPP@cluster[i]
    m = centers[a,]
    sum((x-m)^2)
}
sum(kpp_residualss)
```

```
## [1] 222943.2
```

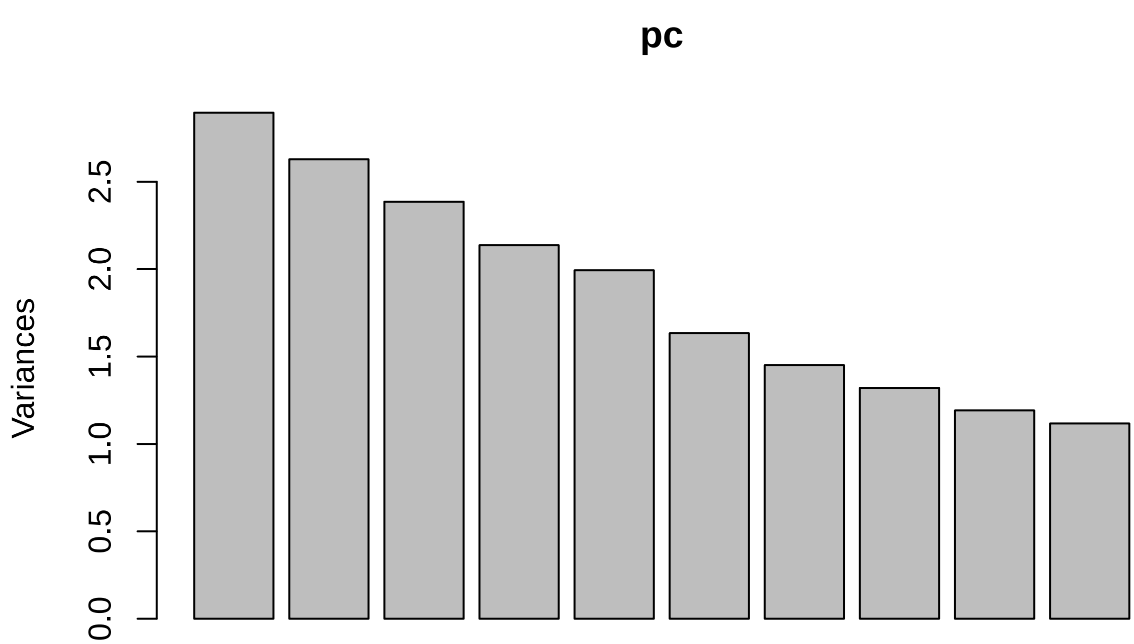
step4: We also tried Principal Component Analysis on the dataset. However, we find that the variance of the dataset cannot be simply explained by the top2 or top3 factors. In fact, top10 pc all have a pretty strong explanation power for the dataset variance. And after we print out the top words associated with first and second PC, it's harder to interpret the "market segment" compared with using just the clustering. Therefore we just use PCA as a supporting evidence to what we find in kmean clustering.

```
# PCA
pc = prcomp(X_freq, scale=TRUE)
```

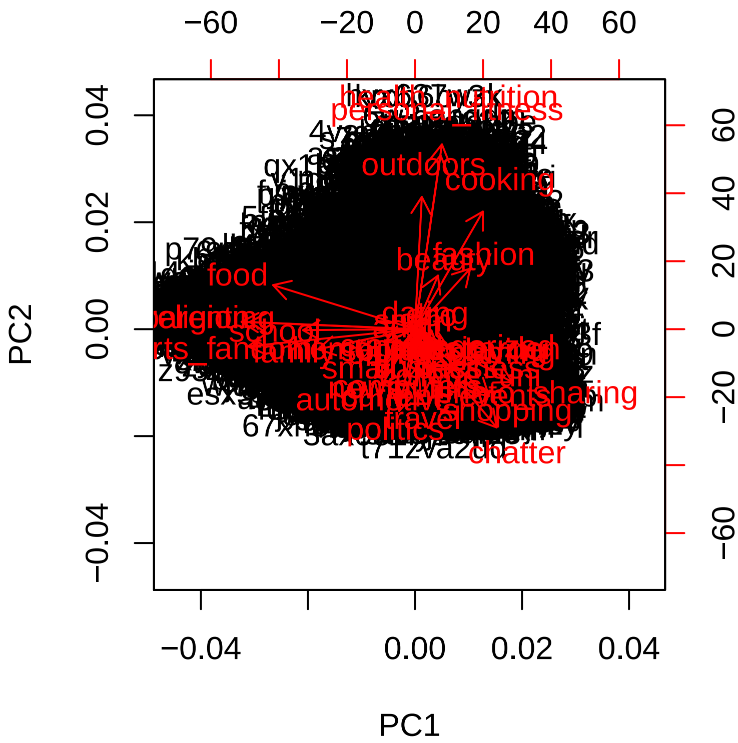
```r
set.seed(1234567)
loadings = pc$rotation
scores = pc$x

summary(pc)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6
## Standard deviation     1.70154 1.62129 1.54469 1.46176 1.41187 1.27794
## Proportion of Variance 0.08042 0.07302 0.06628 0.05935 0.05537 0.04537
## Cumulative Proportion  0.08042 0.15344 0.21972 0.27907 0.33444 0.37981
##                           PC7     PC8    PC9    PC10    PC11    PC12
## Standard deviation     1.20439 1.14929 1.0916 1.05686 1.02932 0.99744
## Proportion of Variance 0.04029 0.03669 0.0331 0.03103 0.02943 0.02764
## Cumulative Proportion  0.42010 0.45679 0.4899 0.52092 0.55035 0.57799
##                         PC13    PC14    PC15    PC16   PC17    PC18
## Standard deviation     0.986 0.98301 0.97282 0.95188 0.9467 0.92384
## Proportion of Variance 0.027 0.02684 0.02629 0.02517 0.0249 0.02371
## Cumulative Proportion  0.605 0.63183 0.65812 0.68329 0.7082 0.73189
##                          PC19    PC20    PC21    PC22   PC23    PC24
## Standard deviation     0.89358 0.86179 0.84943 0.82782 0.8204 0.80437
## Proportion of Variance 0.02218 0.02063 0.02004 0.01904 0.0187 0.01797
## Cumulative Proportion  0.75407 0.77470 0.79475 0.81378 0.8325 0.85045
##                          PC25    PC26    PC27    PC28    PC29    PC30
## Standard deviation     0.78796 0.77161 0.76796 0.75867 0.74529 0.73110
## Proportion of Variance 0.01725 0.01654 0.01638 0.01599 0.01543 0.01485
## Cumulative Proportion  0.86770 0.88424 0.90062 0.91661 0.93204 0.94688
##                          PC31    PC32    PC33    PC34    PC35      PC36
## Standard deviation     0.69765 0.64682 0.61895 0.56561 0.55144 4.714e-15
## Proportion of Variance 0.01352 0.01162 0.01064 0.00889 0.00845 0.000e+00
## Cumulative Proportion  0.96040 0.97203 0.98267 0.99155 1.00000 1.000e+00
```

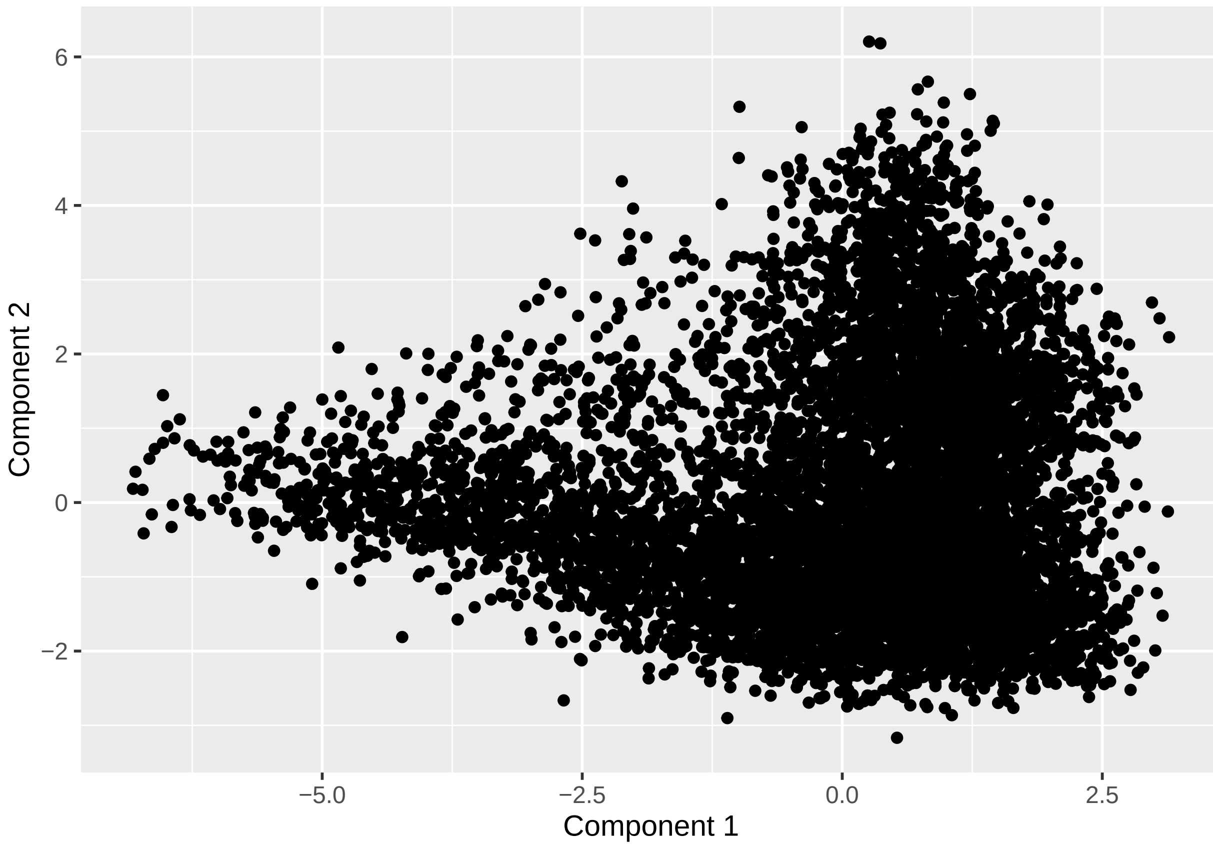
```r
plot(pc)
```



```r
biplot(pc)
```




```r
qplot(scores[,1], scores[,2], xlab='Component 1', ylab='Component 2')
```




```
# The top words associated with each component
o1 = order(loadings[,1])
colnames(X_freq)[head(o1,25)]
```

```
##  [1] "religion"       "sports_fandom"  "parenting"
##  [4] "food"           "school"         "family"
##  [7] "news"           "automotive"     "crafts"
## [10] "politics"       "adult"          "computers"
## [13] "spam"           "art"            "travel"
## [16] "outdoors"       "dating"         "home_and_garden"
## [19] "small_business" "eco"            "tv_film"
## [22] "business"       "music"          "beauty"
## [25] "sports_playing"
```

```
colnames(X_freq)[tail(o1,25)]
```

```
##  [1] "computers"       "spam"             "art"
##  [4] "travel"          "outdoors"         "dating"
##  [7] "home_and_garden" "small_business"   "eco"
## [10] "tv_film"         "business"         "music"
## [13] "beauty"          "sports_playing"   "current_events"
## [16] "personal_fitness" "health_nutrition" "online_gaming"
## [19] "uncategorized"   "college_uni"      "fashion"
## [22] "cooking"         "shopping"         "chatter"
## [25] "photo_sharing"
```

```r
o2 = order(loadings[,2])
colnames(X_freq)[head(o2,25)]
```

```
##  [1] "chatter"         "politics"        "travel"
##  [4] "shopping"        "automotive"      "current_events"
##  [7] "photo_sharing"   "news"            "computers"
## [10] "tv_film"         "college_uni"     "small_business"
## [13] "business"        "online_gaming"   "family"
## [16] "sports_playing"  "home_and_garden" "sports_fandom"
## [19] "art"             "uncategorized"   "music"
## [22] "crafts"          "school"          "spam"
## [25] "adult"
```

```r
colnames(X_freq)[tail(o2,25)]
```

```
##  [1] "small_business"   "business"         "online_gaming"
##  [4] "family"           "sports_playing"   "home_and_garden"
##  [7] "sports_fandom"    "art"              "uncategorized"
## [10] "music"            "crafts"           "school"
## [13] "spam"             "adult"            "parenting"
## [16] "religion"         "eco"              "dating"
## [19] "food"             "beauty"           "fashion"
## [22] "cooking"          "outdoors"         "personal_fitness"
## [25] "health_nutrition"
```

seems pure clustering is just better