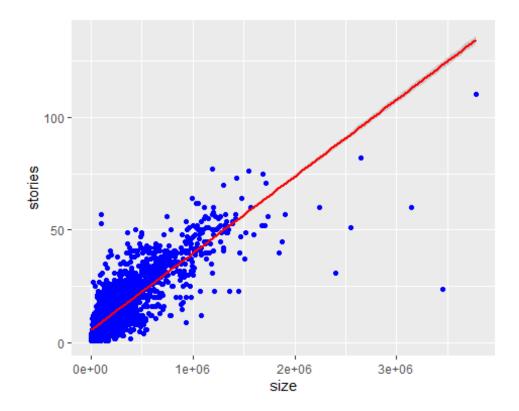
Final Exam STA380

MacAlistair Husted(MWH2359) 8/17/2020

Question 1

Before conducting any analysis, I first cleaned the data. The procedure was quite similar to the stat guru's besides ensuring that there were no missing data in any of the columns by omitting NAs. Since the stat guru did not analyze green buildings vs. non-green buildings based on the number of stories of a building, I first looked into whether size and buildings are highly correlated. And thus, size could be considered a proxy for stories. Based on the results of the correlation, the size of the building and the number of stories are highly correlated. This is visualized the scatter plot created.

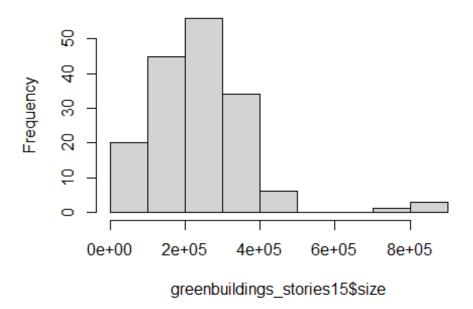
```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)</pre>
greenbuildings$amenities <- as.factor(greenbuildings$amenities)</pre>
greenbuildings$renovated <- as.factor(greenbuildings$renovated)</pre>
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)</pre>
greenbuildings$LEED <- as.factor(greenbuildings$LEED)</pre>
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)</pre>
#correlation of stories and size of building
cor(greenbuildings$size,greenbuildings$stories)
## [1] 0.8261416
ggplot(greenbuildings,aes(size, stories))+ geom_point(colour = "blue")+
 geom smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")
```



Now to determine if the square feet of 250000 that the stat guru used is correct. Based on the histogram of only 15 story buildings. The size of 15 story buildings is anywhere between 250000 and 35000. This is why we use only buildings that are between 250000 and 350000 to analyze further.

```
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
greenbuildings_stories15 <- subset(greenbuildings, greenbuildings$stories ==
15)
hist(greenbuildings_stories15$size)</pre>
```

Histogram of greenbuildings_stories15\$size

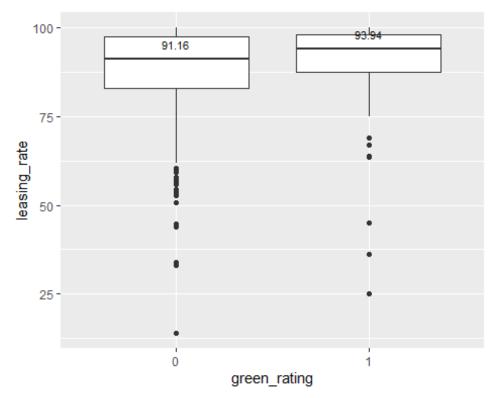


To determine whether her assumption that the green buildings would have 90% leasing rate was grounded in the data. I looked at leasing rate between green and non-green buildings via a boxplot. Based on the boxplot, it does seem that her assumption is grounded in the data as the median lease is very close to 90%, even slightly larger.

```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
greenbuildings$green rating <- as.factor(greenbuildings$green rating)</pre>
greenbuildings$amenities <- as.factor(greenbuildings$amenities)</pre>
greenbuildings$renovated <- as.factor(greenbuildings$renovated)</pre>
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)</pre>
greenbuildings$LEED <- as.factor(greenbuildings$LEED)</pre>
greenbuildings <- subset(greenbuildings,greenbuildings,fleasing rate > 0.1)
greenbuildings <- na.omit(greenbuildings)</pre>
subset greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)</pre>
```

```
green_subset_medians <- ddply(subset_greenbuildings, .(green_rating),
summarise, median = round(median(leasing_rate),4))

ggplot(data=subset_greenbuildings,aes(x = green_rating, y = leasing_rate)) +
    geom_boxplot()+
    geom_text(data = green_subset_medians,aes(x = green_rating, y = median,
label = median), size = 3, vjust = -1.1)</pre>
```

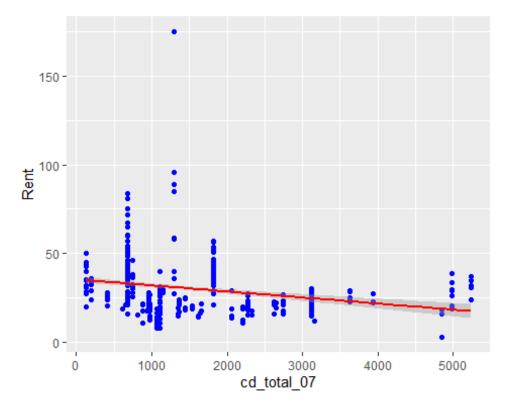


Now, I analyzed the

data further to find possible confounding variables that the stat guru missed in her analysis. The first variable is cd_total_07 which is the number of cooling degree days in the region. This is an especially important variable as Austin is on average quite hot. This variable was negatively correlated with Rent.

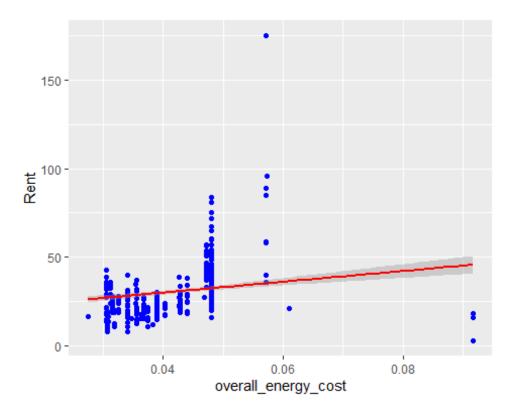
```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)</pre>
```

```
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings, greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& green_subset_medians <- ddply(subset_greenbuildings, .(green_rating),
summarise, median = round(median(leasing_rate),4))
ggplot(subset_greenbuildings,aes(cd_total_07, Rent))+ geom_point(colour =
"blue")+
geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")</pre>
```



```
subset_greenbuildings$overall_energy_cost <- subset_greenbuildings$Gas_Costs
+ subset_greenbuildings$Electricity_Costs
cor(subset_greenbuildings$Rent,subset_greenbuildings$cd_total_07)
## [1] -0.2579879

ggplot(subset_greenbuildings,aes(overall_energy_cost, Rent))+
geom_point(colour = "blue")+
    geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")</pre>
```



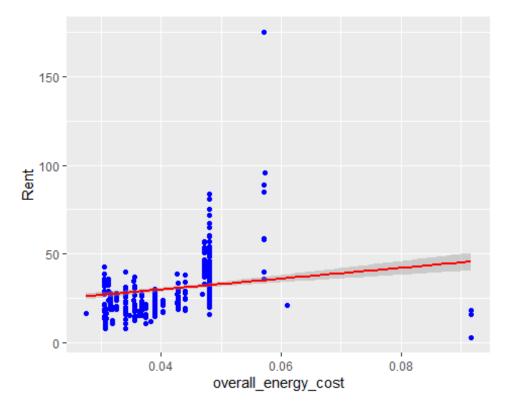
```
cor(subset_greenbuildings$Rent, subset_greenbuildings$overall_energy_cost)
## [1] 0.2291294
```

This is also illustrated by the scatter plot. The second confounding variable was Gas and Electricity Cost. Although, I combined these two variables into one in my analysis. This variable had a positive correlation with Rent of buildings. This is also illustrated by the scatter plot.

```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green rating <- as.factor(greenbuildings$green rating)</pre>
greenbuildings$amenities <- as.factor(greenbuildings$amenities)</pre>
greenbuildings$renovated <- as.factor(greenbuildings$renovated)</pre>
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)</pre>
greenbuildings$LEED <- as.factor(greenbuildings$LEED)</pre>
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
```

```
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)
subset_greenbuildings$overall_energy_cost <- subset_greenbuildings$Gas_Costs
+ subset_greenbuildings$Electricity_Costs
cor(subset_greenbuildings$Rent,subset_greenbuildings$cd_total_07)
## [1] -0.2579879

ggplot(subset_greenbuildings,aes(overall_energy_cost, Rent))+
geom_point(colour = "blue")+
geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")</pre>
```



```
cor(subset_greenbuildings$Rent, subset_greenbuildings$overall_energy_cost)
## [1] 0.2291294
```

I used these two confounding variables in my analysis of Rent between non-green and green buildings. To calculate the Rent by square feet for green and non-green while using the two confounding variables, I conducted a linear regression with Rent as the response variable and cd_total_07 and total energy cost as the predictors. Prior to conducting the linear regression model, I created a subset based on whether the buildings were green or not green. The indicator used was green_rating. The two data sets would be used in two linear regression looking at green buildings and another looking at non-green buildings for comparison. After conducting this regression, I input the mean rent for green and non-green into their corresponding linear regression equations. To get a comparison of Rent per square.

```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green rating <- as.factor(greenbuildings$green rating)</pre>
greenbuildings$amenities <- as.factor(greenbuildings$amenities)</pre>
greenbuildings$renovated <- as.factor(greenbuildings$renovated)</pre>
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)</pre>
greenbuildings$LEED <- as.factor(greenbuildings$LEED)</pre>
greenbuildings <- subset(greenbuildings,greenbuildings$leasing rate > 0.1)
greenbuildings <- na.omit(greenbuildings)</pre>
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)</pre>
subset_greenbuildings$overall_energy_cost <- subset_greenbuildings$Gas_Costs</pre>
+ subset greenbuildings$Electricity Costs
subset greenbuildings green <- subset(subset greenbuildings, green rating ==</pre>
1)
regression <- lm(log(subset greenbuildings green$Rent) ~ overall energy cost
+cd_total_07 , data = subset_greenbuildings_green)
summary(regression)
##
## Call:
## lm(formula = log(subset_greenbuildings_green$Rent) ~ overall_energy_cost +
##
       cd_total_07, data = subset_greenbuildings_green)
##
## Residuals:
                       Median
        Min
                  10
                                     30
                                             Max
## -0.97827 -0.22816 0.00924 0.23320 0.88719
##
## Coefficients:
##
                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)
                        2.939e+00 1.995e-01 14.732 < 2e-16 ***
## overall_energy_cost 1.483e+01 4.351e+00 3.410 0.000949 ***
## cd total 07
                       -8.772e-05 2.987e-05 -2.936 0.004148 **
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3721 on 97 degrees of freedom
## Multiple R-squared: 0.1951, Adjusted R-squared: 0.1785
## F-statistic: 11.76 on 2 and 97 DF, p-value: 2.681e-05
```

```
mean_green_reg <- mean(log(subset_greenbuildings_green$Rent))</pre>
result = mean green reg *regression$coefficients[2] +
mean_green_reg*regression$coefficients[3] + regression$coefficients[1]
print(paste("Overall Rent:", as.numeric(result)))
## [1] "Overall Rent: 53.9518358222461"
subset_greenbuildings_nongreen <- subset(subset_greenbuildings, green rating</pre>
== 0)
regression <- lm(log(subset greenbuildings nongreen$Rent) ~
overall energy cost +cd total 07+leasing rate , data =
subset_greenbuildings_nongreen)
summary(regression)
##
## Call:
## lm(formula = log(subset greenbuildings nongreen$Rent) ~
overall energy cost +
##
       cd_total_07 + leasing_rate, data = subset_greenbuildings_nongreen)
##
## Residuals:
       Min
                 1Q
                      Median
                                   30
                                           Max
## -2.00638 -0.22812 0.03007 0.34330 1.70124
##
## Coefficients:
##
                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)
                       2.932e+00 1.333e-01 22.004 < 2e-16 ***
## overall_energy_cost 9.335e+00 1.924e+00 4.853 1.58e-06 ***
## cd total 07 -1.937e-04 1.991e-05 -9.728 < 2e-16 ***
## leasing_rate
                       2.507e-03 1.412e-03 1.775
                                                      0.0764 .
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.05 '.' 0.1 ' ' 1
## Residual standard error: 0.4959 on 562 degrees of freedom
## Multiple R-squared: 0.1558, Adjusted R-squared: 0.1513
## F-statistic: 34.57 on 3 and 562 DF, p-value: < 2.2e-16
mean_nongreen_reg <- mean(log(subset_greenbuildings_nongreen$Rent))</pre>
result = mean nongreen reg *regression$coefficients[2] +
mean nongreen reg*regression$coefficients[3] + regression$coefficients[1]
print(paste("Overall Rent:", as.numeric(result)))
## [1] "Overall Rent: 33.5314134003712"
```

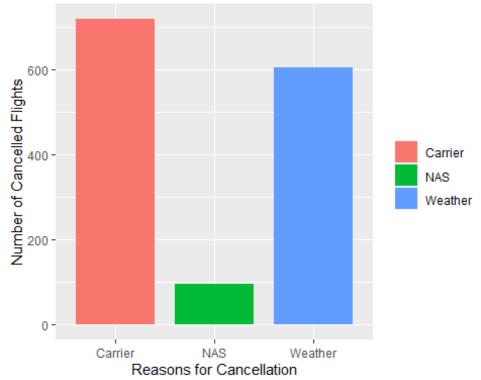
As you can see, the rent per square feet for green buildings is much higher than non-green buildings. Thus, the stat's guru's conclusion is correct that green buildings will be more profitable for the real estate developer over time.

Question 2

```
library(ggplot2)

ABIA <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ABIA.csv")

library(dplyr)
cancelled <- aggregate(x=ABIA$Cancelled,
by=list(TypeCancelled=ABIA$CancellationCode), FUN=sum)
cancelled$TypeCancelled <- ifelse(cancelled$TypeCancelled == 'A',"Carrier",ifelse(cancelled$TypeCancelled == 'B', 'Weather', 'NAS' ))
cancelled <-cancelled[-1,]
ggplot(cancelled, aes(x = TypeCancelled, y = x, fill = TypeCancelled))+
    geom_bar(stat = "identity", width = 0.8) +
    xlab("Reasons for Cancellation") + ylab("Number of Cancelled Flights") +
    theme(legend.title = element_blank())</pre>
```



Distribution of

Cancellation Code. It seems that flights get cancelled to and from Austin due to Weather or due to the airline.

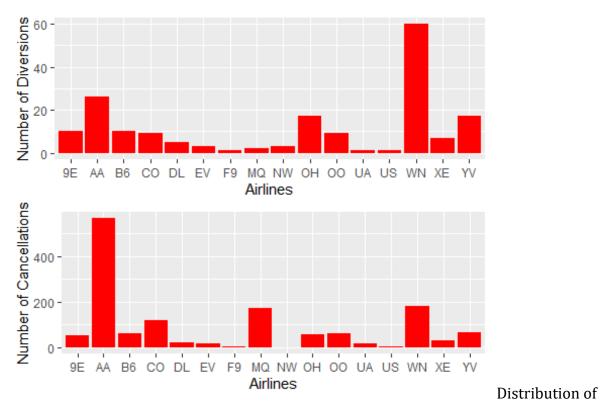
```
library(ggplot2)

df_cancelled <- aggregate(x=ABIA$Cancelled,
by=list(Carriers=ABIA$UniqueCarrier), FUN=sum)</pre>
```

```
df_diverted <- aggregate(x=ABIA$Diverted,
by=list(Carriers=ABIA$UniqueCarrier), FUN=sum)
colnames(df_cancelled) = c("Carrier", "Cancellations")

colnames(df_diverted) = c("Carrier", "Diversions")

divert= ggplot(df_diverted,aes(x= Carrier,y=Diversions))+ geom_bar(stat = 'identity',fill = 'red') + labs(y = 'Number of Diversions', x = 'Airlines',color =df_diverted$Carrier)
cancel = ggplot(df_cancelled,aes(x= Carrier,y=Cancellations))+ geom_bar(stat = 'identity',fill = 'red') + labs(y = 'Number of Cancellations', x = 'Airlines', color = df_cancelled$Carrier)
gridExtra::grid.arrange(divert,cancel)</pre>
```



cancellations and diversions by Airline. WN (Southwest) has highest number of cancellations going to and from Austin Airport. While, AA(American Airlines) has the highest number of diversions.

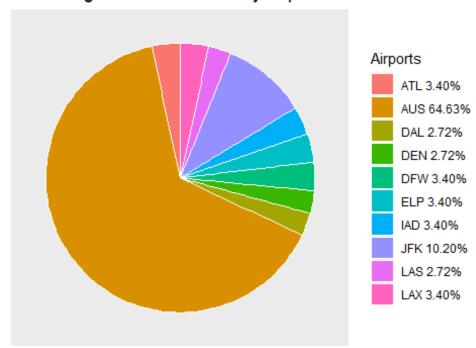
```
library(dplyr)
library(ggplot2)

airportcanceled<- aggregate(x=ABIA$Cancelled, by=list(Airport=ABIA$Origin),
FUN=sum)

colnames(airportcanceled) = c("Airport", "Cancellations")
airportcanceled <-airportcanceled[airportcanceled$Cancellations != 0,]</pre>
```

```
top10 = head(sort(airportcanceled$Cancellations,decreasing=TRUE), n = 10)
top10airports <- airportcanceled[which(airportcanceled$Cancellations %in%
top10),
top10airports_per = mutate(top10airports,
                cancel pct = top10airports$Cancellations /
sum(Cancellations))
top10airports per$cancel pct <- round(top10airports per$cancel pct,2)
percent <- function(x, digits = 2, format = "f", ...) {</pre>
                                                          # Create user-
defined function
  paste0(formatC(x * 100, format = format, digits = digits, ...), "%")
top10airports_per$labels <- percent(top10airports_per$cancel_pct)
top10airports per$str pct <- as.character(top10airports per$labels)</pre>
top10airports_per$Airports <- paste(top10airports_per$Airport,</pre>
top10airports per$str pct)
airportdiverted<- aggregate(x=ABIA$Diverted, by=list(Airport=ABIA$Origin),
FUN=sum)
colnames(airportdiverted) = c("Airport", "Diversions")
top10divert = head(sort(airportdiverted$Diversions, decreasing=TRUE), n = 10)
top10airports divert <- airportdiverted[which(airportdiverted$Diversions %in%
top10divert),
top10airports_divert_per = mutate(top10airports_divert,
                                  divert pct =
top10airports divert$Diversions / sum(Diversions))
top10airports_divert_per$labels <-</pre>
percent(top10airports_divert_per$divert_pct)
top10airports divert per$str pct <-
as.character(top10airports divert per$labels)
top10airports divert per$Airports <- paste(top10airports divert per$Airport,
top10airports divert per$str pct)
require(ggrepel)
## Loading required package: ggrepel
ggplot(top10airports_divert_per, aes(x = '', y=Diversions, fill=Airports)) +
  geom_bar(stat="identity", width=1, color="white") + coord_polar(theta ="y",
start=0) +
  theme(axis.title.x = element blank(), axis.title.y =
element blank(),axis.text = element blank(),
        axis.ticks = element_blank(),
        panel.grid = element blank()) + labs(title = "Percentage of all
Diversions by Airport")
```

Percentage of all Diversions by Airport



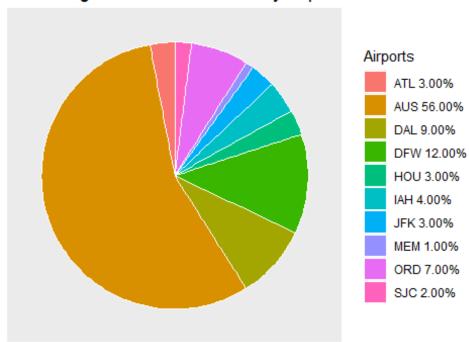
The percentage of

the top 10 Airports with the most diversions for flights going into Austin.

```
library(dplyr)
airportdiverted<- aggregate(x=ABIA$Diverted, by=list(Airport=ABIA$Origin),
FUN=sum)
colnames(airportdiverted) = c("Airport", "Diversions")
top10divert = head(sort(airportdiverted$Diversions, decreasing=TRUE), n = 10)
top10airports divert <- airportdiverted[which(airportdiverted$Diversions %in%
top10divert),]
top10airports divert per = mutate(top10airports divert,
                                  divert pct =
top10airports_divert$Diversions / sum(Diversions))
top10airports_divert_per$labels <-
percent(top10airports divert per$divert pct)
top10airports divert per$str pct <-
as.character(top10airports_divert_per$labels)
top10airports_divert_per$Airports <- paste(top10airports_divert_per$Airport,
top10airports divert per$str pct)
require(ggrepel)
ggplot(top10airports_per, aes(x ='', y=cancel_pct, fill=Airports)) +
  geom bar(stat="identity", width=1, color="white") + coord polar(theta ="y",
start=0) +
  theme(axis.title.x = element blank(), axis.title.y =
element_blank(), axis.text = element_blank(),
```

```
axis.ticks = element_blank(),
    panel.grid = element_blank()) +labs(title = "Percentage of all
Cancellations by Airport")
```

Percentage of all Cancellations by Airport



The percentage of

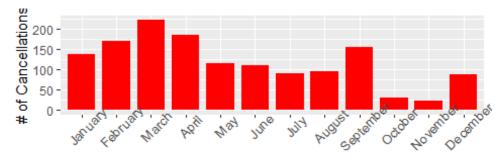
the top 10 Airports with the most cancellations for flights going into Austin

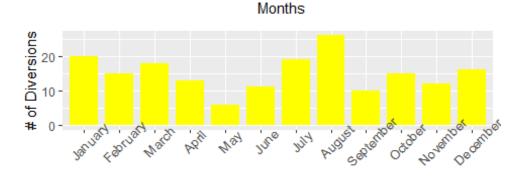
```
library(dplyr)
library(ggplot2)
airlinescanceled2<- aggregate(x=ABIA$Cancelled,</pre>
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinescanceled2) = c("Airlines", "Cancellations")
airlinesdiverted2 <- aggregate(x=ABIA$Diverted,</pre>
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinesdiverted2) = c("Airlines", "Diversions")
airlines <- merge(airlinesdiverted2, airlinescanceled2,by=c("Airlines"))</pre>
sortedairlines= airlines[with(airlines, order(-Cancellations, Diversions)), ]
top10airlines <- head(sortedairlines, n=10)</pre>
cancelled month <- aggregate(x=ABIA$Cancelled, by=list(ABIA$Month), FUN=sum)</pre>
divert_month <- aggregate(x=ABIA$Diverted, by =list(ABIA$Month), FUN = sum)</pre>
colnames(cancelled_month) = c("Months", "Cancellations")
colnames(divert_month) = c("Months", "Diversions")
cancelled month<- cancelled month %>% mutate(MonthName = month.name[Months])
```

```
divert_month<- divert_month %>% mutate(MonthName = month.name[Months])

cancelled_month1<-ggplot(cancelled_month, aes(x = factor(MonthName,levels=month.name), y = Cancellations))+
    geom_bar(stat = "identity", width = 0.8, fill = 'red') +
    xlab("Months") + ylab("# of Cancellations") + theme(axis.text.x = element_text(size=10, angle=45))

divert_month1<- ggplot(divert_month, aes(x = factor(MonthName,levels=month.name), y = Diversions))+
    geom_bar(stat = "identity", width = 0.8,fill = 'yellow') +
    xlab("Months") + ylab("# of Diversions") + theme(axis.text.x = element_text(size=10, angle=45))
gridExtra::grid.arrange(cancelled_month1,divert_month1)</pre>
```





Months

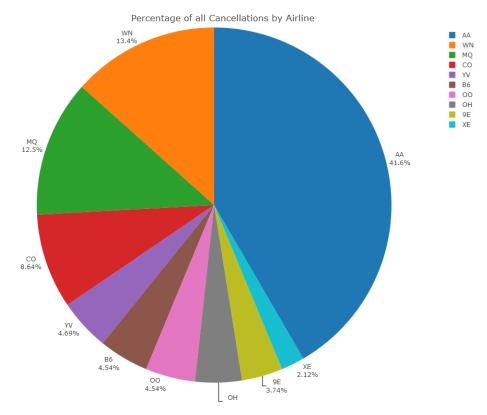
Number of

cancellations and diversions by month in 2008. February had the highest number of cancellations while July had the highest number diversions.

```
airlinescanceled3<- aggregate(x=ABIA$Cancelled,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
airlinesdiverted3<- aggregate(x=ABIA$Diverted,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinescanceled3) = c("Airlines", "Cancellations")
colnames(airlinesdiverted3) = c("Airlines", "Diversions")

top10canceled = head(sort(airlinescanceled3$Cancellations,decreasing=TRUE), n</pre>
```

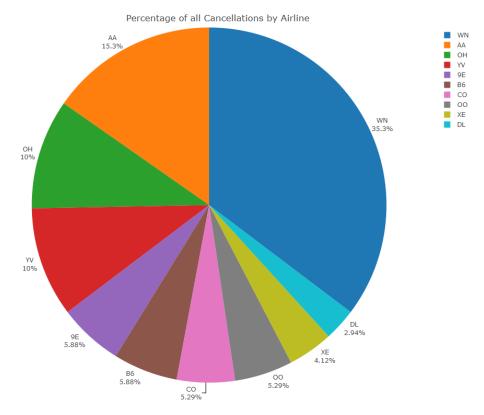
```
= 10)
top10airline canceled <-
airlinescanceled3[which(airlinescanceled3$Cancellations %in% top10canceled),]
top10diverted = head(sort(airlinesdiverted3$Diversions,decreasing=TRUE), n =
10)
top10airline_diverted <- airlinesdiverted3[which(airlinesdiverted3$Diversions</pre>
%in% top10diverted),]
top10airlines_per_cancelled = mutate(top10airline_canceled,
                           cancel pct = top10airline canceled$Cancellations /
sum(Cancellations))
top10airlines per cancelled$cancel pct <-
percent(top10airlines_per_cancelled$cancel_pct)
top10airlines per cancelled = mutate(top10airline canceled,
                                    cancel pct =
top10airline canceled$Cancellations /
sum(top10airline_canceled$Cancellations))
library(plotly)
library(webshot)
p1<- plot_ly(top10airlines_per_cancelled, labels = ~Airlines, values =
~cancel_pct, type = 'pie',textposition = 'outside',textinfo =
'label+percent') %>%
  layout(title = 'Percentage of all Cancellations by Airline',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))
top10airlines_per_diverted = mutate(top10airline_diverted,
                                     diverted pct =
top10airline_diverted$Diversions / sum(Diversions))
tmpFile <- tempfile(fileext = ".png")</pre>
export(p1, file = tmpFile)
```



The percentage of the top 10 Airports with the most cancellations for flights going into Austin.

```
library(dplyr)
library(ggplot2)
airlinescanceled3<- aggregate(x=ABIA$Cancelled,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
airlinesdiverted3<- aggregate(x=ABIA$Diverted,</pre>
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinescanceled3) = c("Airlines", "Cancellations")
colnames(airlinesdiverted3) = c("Airlines", "Diversions")
top10canceled = head(sort(airlinescanceled3$Cancellations, decreasing=TRUE), n
= 10)
top10airline canceled <-
airlinescanceled3[which(airlinescanceled3$Cancellations %in% top10canceled),]
top10diverted = head(sort(airlinesdiverted3$Diversions, decreasing=TRUE), n =
top10airline diverted <- airlinesdiverted3[which(airlinesdiverted3$Diversions
%in% top10diverted),]
top10airlines_per_cancelled = mutate(top10airline_canceled,
                           cancel pct = top10airline canceled$Cancellations /
sum(Cancellations))
```

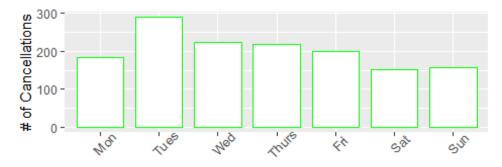
```
top10airlines_per_cancelled$cancel_pct <-</pre>
percent(top10airlines per cancelled$cancel pct)
top10airlines per cancelled = mutate(top10airline canceled,
                                     cancel pct =
top10airline canceled$Cancellations /
sum(top10airline canceled$Cancellations))
top10airlines_per_diverted = mutate(top10airline_diverted,
                                      diverted_pct =
top10airline diverted$Diversions / sum(Diversions))
library(plotly)
library(webshot)
p2 <- plot_ly(top10airlines_per_diverted, labels = ~Airlines, values =</pre>
~diverted_pct, type = 'pie',textposition = 'outside',textinfo =
'label+percent') %>%
  layout(title = 'Percentage of all Cancellations by Airline',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))
tmpFile <- tempfile(fileext = ".png")</pre>
export(p2, file = tmpFile)
```



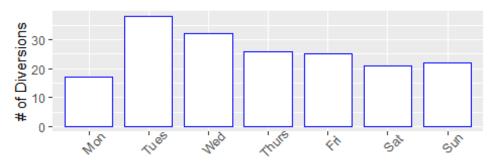
The percentage of the top 10 Airports with the most diversions for flights going into Austin.

```
library(dplyr)
library(ggplot2)
ABIA$DayOfWeeknames <- ifelse(ABIA$DayOfWeek == 1,
"Mon",ifelse(ABIA$DayOfWeek == 2, "Tues",ifelse(ABIA$DayOfWeek == 3,
"Wed", ifelse(ABIA$DayOfWeek == 4, "Thurs", ifelse(ABIA$DayOfWeek == 5,
"Fri",ifelse(ABIA$DayOfWeek == 6, "Sat", ifelse(ABIA$DayOfWeek == 7,
"Sun",0))))))
weekdiverted<- aggregate(x=ABIA$Diverted,</pre>
by=list(DayofWeek=ABIA$DayOfWeeknames), FUN=sum)
weekcanceled<- aggregate(x=ABIA$Cancelled,</pre>
by=list(DayofWeek=ABIA$DayOfWeeknames), FUN=sum)
weekcanceled1<-ggplot(weekcanceled, aes(x = factor(DayofWeek, levels =</pre>
c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sat", "Sun")), y = x))+
geom_bar(stat = "identity", width = 0.8, color = 'green', fill = 'white') +
  xlab("Days of the Week") + ylab("# of Cancellations") + theme(axis.text.x =
element text(size=10, angle=45))
weekdiverted1<- ggplot(weekdiverted, aes(x = factor(DayofWeek, levels =</pre>
c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sat", "Sun")), y = x))+
 geom bar(stat = "identity", width = 0.8, color = 'blue',fill ='white') +
```

```
xlab("Days of the Week") + ylab("# of Diversions") + theme(axis.text.x =
element_text(size=10, angle=45))
gridExtra::grid.arrange(weekcanceled1,weekdiverted1)
```



Days of the Week



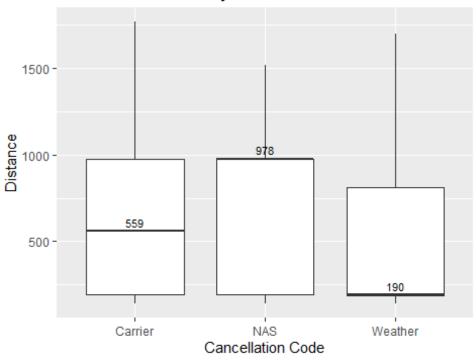
Days of the Week

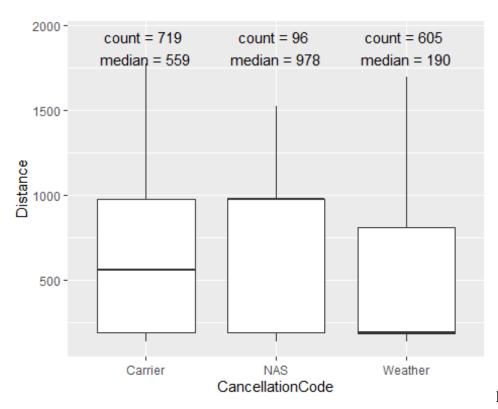
Number of

cancellations and diversions by day of the week in 2008. Tuesday had the largest number of cancellations and diversions.

```
library(dplyr)
library(ggplot2)
ABIA$CancellationCode <- ifelse(ABIA$CancellationCode ==
'A', "Carrier", ifelse (ABIA Cancellation Code == 'B',
'Weather',ifelse(ABIA$CancellationCode == 'C','NAS', 'None')))
ABIA sub <- subset(ABIA, ABIA$CancellationCode != 'None')
options(scipen = 999)
dataMedian <- summarise(group_by(ABIA_sub, CancellationCode), MD =</pre>
median(Distance))
ggplot(data=ABIA sub, aes(x=as.factor(ABIA sub$CancellationCode),
y=ABIA sub$Distance)) +
  geom_boxplot() + geom_text(data = dataMedian, aes(x=CancellationCode, y =
MD, label= MD),
                             position = position dodge(width = 0.8), size =
3, vjust = -0.5) +
  ggtitle("Cancellation Code by Distance")+ xlab("Cancellation Code") +
ylab("Distance")
```

Cancellation Code by Distance



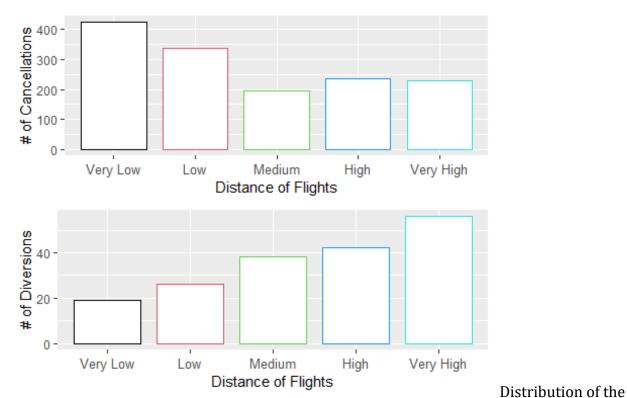


Boxplot displaying

the number of different types of cancellations by distance travelled.

```
library(gtools)
library(dplyr)
library(ggplot2)
ABIA$DistanceFactor <- quantcut(ABIA$Distance, q=5, na.rm = TRUE,c('Very
Low', 'Low', 'Medium', 'High', 'Very High'))
Distancecanceled<- aggregate(x=ABIA$Cancelled,
by=list(DistanceQuantile=ABIA$DistanceFactor), FUN=sum)
Distancediverted<- aggregate(x=ABIA$Diverted,</pre>
by=list(DistanceQuantile=ABIA$DistanceFactor), FUN=sum)
colnames(Distancecanceled) = c("Distance Quantiles", "Cancellations")
colnames(Distancediverted) = c("Distance Quantiles", "Diversions")
Distancecanceled1<-ggplot(Distancecanceled, aes(x =
Distancecanceled$`Distance Quantiles`, y = Cancellations))+
  geom_bar(stat = "identity", width = 0.8, color = Distancecanceled$`Distance
Quantiles`, fill = 'white' ) +
  xlab("Distance of Flights") + ylab("# of Cancellations") +
theme(axis.text.x = element text(size=10))
Distancediverted1<- ggplot(Distancediverted, aes(x =</pre>
Distancediverted$`Distance Quantiles`, y = Diversions))+
  geom_bar(stat = "identity", width = 0.8, color = Distancecanceled$`Distance
Quantiles`, fill = 'white') +
```

```
xlab("Distance of Flights") + ylab("# of Diversions") + theme(axis.text.x =
element_text(size=10))
gridExtra::grid.arrange(Distancecanceled1,Distancediverted1)
```



number of cancellations and diversion by distance. It seems flights at very low distances have more cancellations while flights with very high distances have more diversions.

Question 3

In my first portfolio, the underlying goal was to pick ETFs that were from multiple different industries and markets. This is what I would like to consider my diverse ETF portfolio. This portfolio had the following five ETFs: VIS(Vanguard Industrial ETF), XLV (Health Care Select Sector SPDR Fund), XNTK(SPDR Nyse Technology ETF), and VWO(Vanguard Emerging Markets Stock Index). From the plots of the close to close changes there is a lot of volatility after February 2020 which makes sense because this was when there was more volatility in the market due to COVID-19. When looking at the correlation between ETFs, it does look like there is a strong correlation between ETFs. All the correlations seem to also be positive. While, the correlation seems to be much more positive for XNTK and all other ETFs. This is quite interesting and unexpected as I would've thought the correlation would be only slight positive as these ETFs are focused on diverse markets and industries. In terms of the entire portfolios correlation between today's returns and tomorrow's returns, there was no correlation. There also only seem to be a slight autocorrelation between lags. Possibly, a slight correlation between lags. After going through a simulation of 25 days, it seems that I could make a profit of \$1200.67. The 5% VAR for this portfolio after running

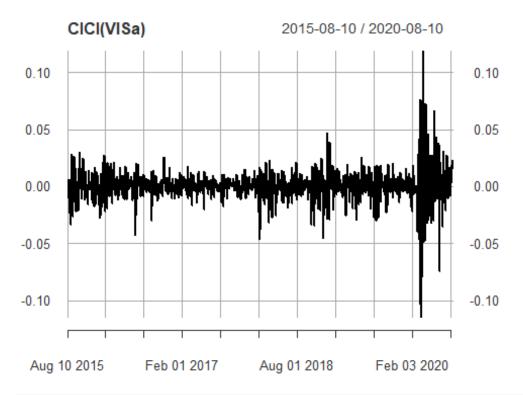
my simulation was 9083.38. Thus, my worst 5% of outcomes have me losing \$9083.38 during 4 week period. In my second portfolio, the underlying goal was to pick ETFs that are considered safe. My criteria is that the volatility of the ETF has to be considered minimal or low. This is to minimize my loses. The ETFs that were chosen are the following: SHY (iShares 1-3 Year Treasury Bond), SPLV(Invesco S&P 500 Low Volatility ETF), USMV(iShares Edge MSCI Min Vol USA ETF), IEF(iShares 7-10 Year Treasury Bond), and EFAV(iShares MSCI EAFE Min Vol Factor). Overall, the ETFs do have relative low volatility based on their plots of close to close changes. Although, there is still the largely volatile period starting at February 2020 which is expected. Unlike the diverse portfolio, the correlations between the ETFs for the safe portfolio are not very correlated overall. Although, there is a positive correlation between SPLV and USMV. Overall, this expected since many of these ETFs are not based on a particular industry. When looking at the correlation between all the returns for the portfolio, there is no correlation. In terms of autocorrelation, there only seems to be a slight correlation between lags. After going through a simulation of 25 days, it seems that I could make a profit of \$528.33. The 5% VAR for this portfolio is \$3766.34. This makes sense given that I was trying to be safe so my worst 5% of the outcomes would have me losing less than my diverse portfolio.

In my third portfolio, the underlying goal was to pick ETFs that are a bit more volatile with the goal of making more money, but with more risk overall. The ETF that were chosen are the following: SOXL(Direxion Daily Semiconductor Bull 3X Shares), TQQQ(ROSHARES TR/ULTRAPRO QQQ), ROM(ProShares Ultra Technology), TECL(Direxion Daily Technology Bull 3X Shares), and VGT(Vanguard Information Technology). Overall, the ETFs do have quite volatilities based on their close to close changes. There is still a relative increase in volatility after February 2020 for reason described previously. The correlations between ETFs is positive, but not as positive as the safe portfolio. There is no correlation in the portfolio's returns when looking at all the ETFs in this portfolio together. There is also no autocorrelation either. After going through a simulation of 25 days, it seems that I could make a profit of \$6200.00. Although, my 5% VAR is 20078 which is much higher than my other portfolio which is expected.

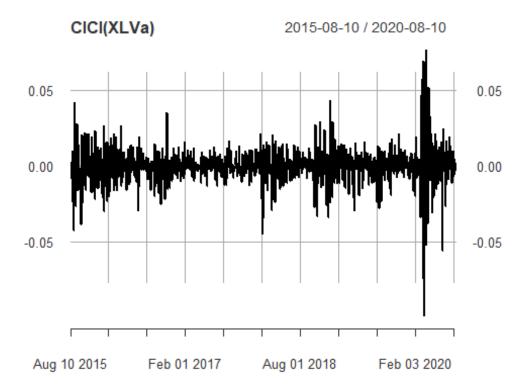
```
library(mosaic)
library(quantmod)
library(foreach)
#Diverse-----
# Import a few stocks
mystocks = c("VIS", "XLV", "IXC","XNTK","VWO")
getSymbols(mystocks)
## [1] "VIS" "XLV" "IXC" "XNTK" "VWO"

VIS <- VIS["2015-08-10::2020-08-10"]
XLV <- XLV["2015-08-10::2020-08-10"]
IXC <- IXC["2015-08-10::2020-08-10"]
XNTK <- XNTK["2015-08-10::2020-08-10"]
VWO <- VWO["2015-08-10::2020-08-10"]</pre>
# Adjust for splits and dividends
```

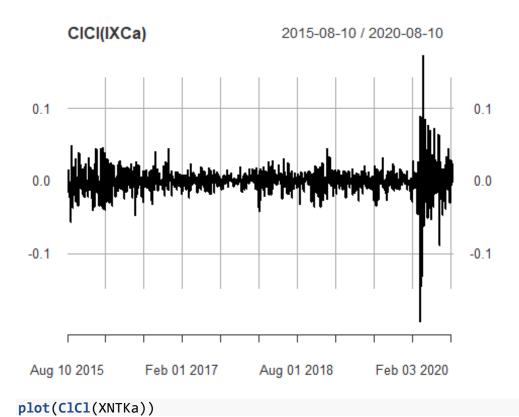
```
VISa = adjustOHLC(VIS)
XLVa = adjustOHLC(XLV)
IXCa = adjustOHLC(IXC)
XNTKa = adjustOHLC(XNTK)
VWOa = adjustOHLC(VWO)
# Look at close-to-close changes
plot(ClCl(VISa))
```

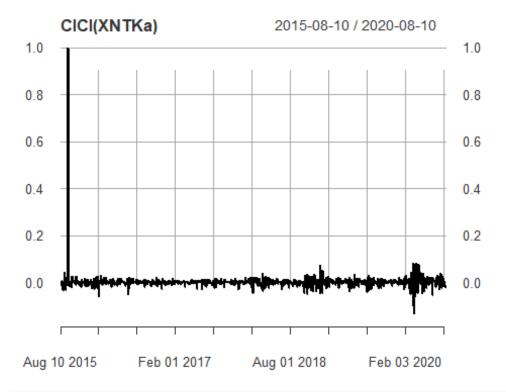


plot(ClCl(XLVa))

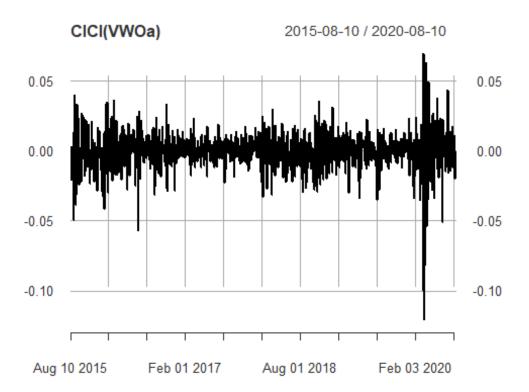


plot(ClCl(IXCa))

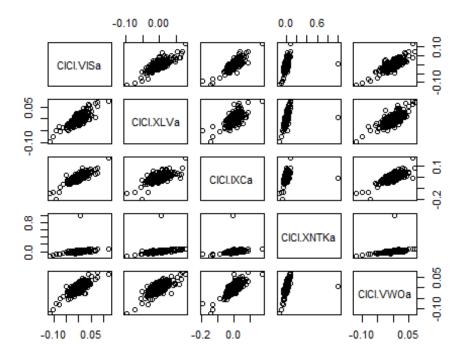


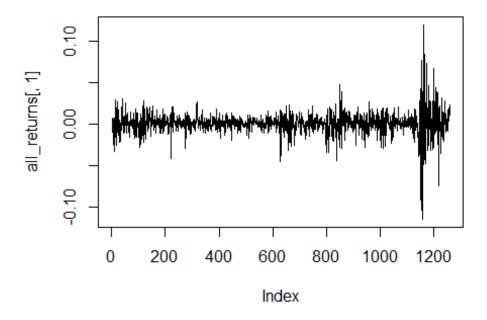


plot(C1C1(VWOa))

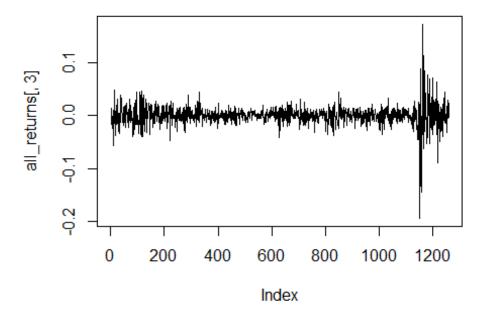


```
# Combine close to close changes in a single matrix
all_returns = cbind(ClCl(VISa),ClCl(XLVa),ClCl(IXCa),ClCl(XNTKa),ClCl(VWOa))
head(all_returns)
##
                ClCl.VISa
                             ClCl.XLVa
                                         ClCl.IXCa
                                                     ClCl.XNTKa
ClCl.VWOa
## 2015-08-10
                       NA
                                    NA
                                                NA
                                                             NA
NA
## 2015-08-11 -0.0106181453 -0.0084288028 -0.003054459 -0.0107836386 -
0.021477187
## 2015-08-12 -0.0006707551 0.0006640324 0.014705883 0.0011276571 -
0.015524678
0.001903208
## 2015-08-14 0.0065202799 0.0027936545 -0.004912496 0.0045984047
0.002996486
## 2015-08-17 0.0052395541 0.0100822769 -0.001234218 0.0037364968 -
0.010592070
# first row is NA because we didn't have a "before" in our data
all returns = as.matrix(na.omit(all returns))
N = nrow(all_returns)
# These returns can be viewed as draws from the joint distribution
# strong correlation, but certainly not Gaussian!
pairs(all_returns)
```

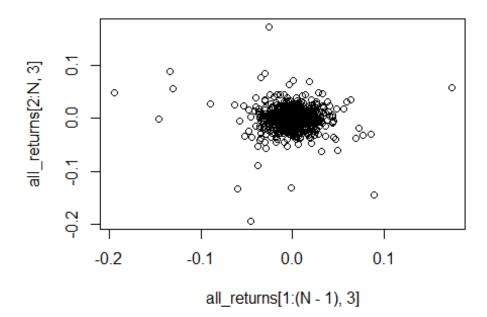




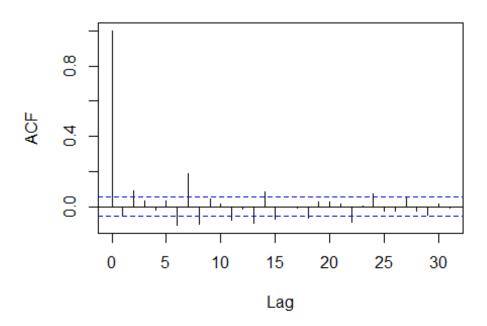
Look at the market returns over time
plot(all_returns[,3], type='1')



```
# are today's returns correlated with tomorrow's?
# not really!
plot(all_returns[1:(N-1),3], all_returns[2:N,3])
```

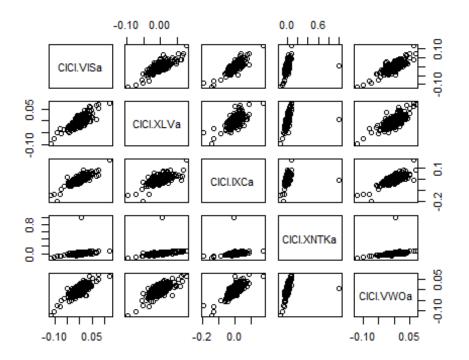


Series all_returns[, 3]



```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)
#### Now use a bootstrap approach
#### With more stocks
#mystocks = c("VIS", "PBE", "IXC", "XNTK", "VWO")
# myprices = getSymbols(mystocks, from = "2015-01-01")
# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in mystocks) {
  expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
  eval(parse(text=expr))
}
head(XNTKa)
##
              XNTK.Open XNTK.High XNTK.Low XNTK.Close XNTK.Volume
XNTK.Adjusted
## 2015-08-10 19.73367 19.84604 19.72998
                                             19.81657
                                                            10000
```

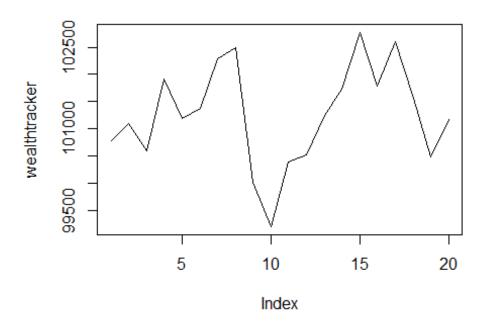
```
39.63312
              19.73919 19.80367 19.51076
## 2015-08-11
                                             19.60287
                                                            23000
39.20573
## 2015-08-12 19.47207
                       19.62498 19.32101
                                             19.62498
                                                             9000
39.24995
## 2015-08-13
              19.63787
                         19.74656 19.59182
                                             19.63050
                                                             9600
39,26100
## 2015-08-14
              19.59734 19.72077 19.57708
                                             19.72077
                                                            17600
39.44154
## 2015-08-17 19.59919 19.82946 19.59919
                                             19.79446
                                                            14400
39.58890
# Combine all the returns in a matrix
# Compute the returns from the closing prices
pairs(all returns)
```



```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# Update the value of your holdings
# Assumes an equal allocation to each asset
total_wealth = 100000
my_weights = c(0.2,0.2,0.2, 0.2, 0.2)
holdings = total_wealth*my_weights
```

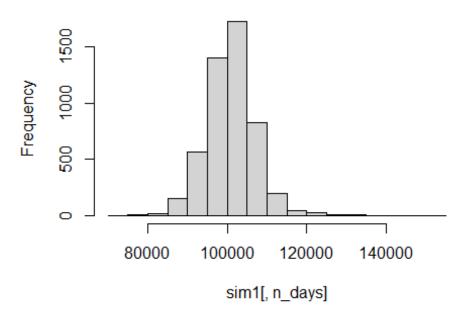
```
holdings = holdings*(1 + return.today)
sum(holdings)
## [1] 100687.5
# Now loop over two trading weeks
# let's run the following block of code 5 or 6 times
# to eyeball the variability in performance trajectories
## begin block
n_days = 20
wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
for(today in 1:n_days) {
  return.today = resample(all_returns, 1, orig.ids=FALSE)
  holdings = holdings + holdings*return.today
  total wealth = sum(holdings)
  wealthtracker[today] = total_wealth
}
total_wealth
## [1] 101166
plot(wealthtracker, type='l')
```



```
# Now simulate many different possible futures
# just repeating the above block thousands of times
initial wealth = 100000
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
 total_wealth = initial_wealth
 weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
 holdings = weights * total wealth
 n_{days} = 20
 wealthtracker = rep(0, n_days)
 for(today in 1:n days) {
    return.today = resample(all_returns, 1, orig.ids=FALSE)
    holdings = holdings + holdings*return.today
    total wealth = sum(holdings)
    wealthtracker[today] = total wealth
 }
 wealthtracker
}
# each row is a simulated trajectory
# each column is a data
head(sim1)
##
                           [,2]
                                     [,3]
                                               [,4]
                                                                   [,6]
                 [,1]
                                                         [,5]
[,7]
## result.1 99507.33 99536.61 98691.01 98245.15 100398.21 99122.24
99060.38
## result.2 98783.72 98269.19 98333.28 99269.39 98918.60
98727.93
## result.3 100467.93 102147.34 102687.63 103305.83 104736.55 103719.02
103222.54
## result.4 99756.90 101421.42 102221.83 100034.76 101073.71 101781.55
101572.75
## result.5 96739.88 98061.26 98449.51 97766.99 98723.05 98917.01
98795.43
                      99468.64 99474.59 100050.04 99955.20 100929.14
## result.6 98936.90
99705.91
##
                 [,8]
                           [,9]
                                    [,10]
                                              [,11]
                                                       [,12]
                                                                 [,13]
\lceil,14\rceil
## result.1 99912.86
                      98332.94 98884.68 99147.24 100022.4 100436.36
98640.40
## result.2 99352.52 98911.83 99256.21 100106.31 100687.5 99356.20
99490.70
## result.3 103505.35 104404.36 104151.69 103767.12 102370.4 102459.67
102259.75
## result.4 101950.13 100720.99 100621.22 100868.72 102519.7 102598.60
102258.86
## result.5 98655.58 98542.57 98722.72 98825.67 100288.9 99342.82
99302.75
## result.6 100505.69 101174.03 101465.12 101561.49 100854.4 100437.22
101662.50
```

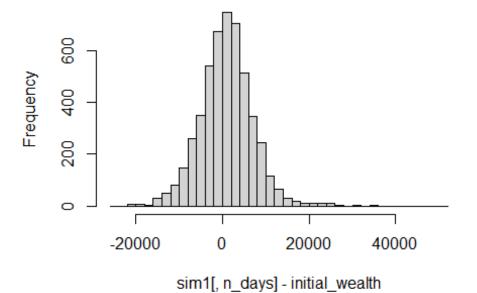
```
##
               [,15]
                         [,16]
                                  [,17]
                                            [,18]
                                                      [,19]
## result.1 99467.12
                      99888.93
                               98322.29 99006.57 100161.09 100172.49
## result.2 99468.74
                      98016.06 97315.75 100574.24 101373.40 102846.43
## result.3 99773.15
                      99268.43 99050.00
                                         99366.87
                                                 99766.55
                                                            99821.82
## result.4 102443.44 102852.94 102590.44 102356.98 103017.11 103474.14
## result.5 100633.56 102778.61 103642.56 104374.37 104215.95 104265.72
## result.6 101506.93 101273.42 102679.21 103334.79 103984.78 104168.84
hist(sim1[,n_days], 25)
```

Histogram of sim1[, n_days]



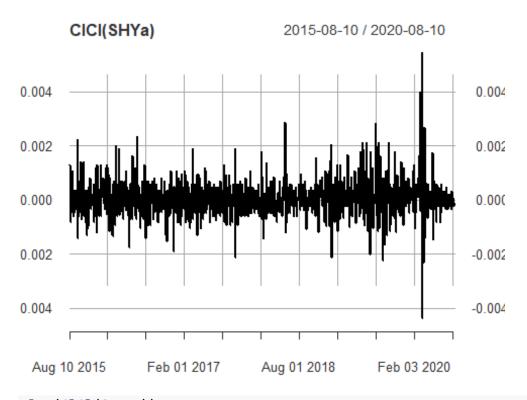
```
# Profit/Loss
mean(sim1[,n_days])
## [1] 100993.7
mean(sim1[,n_days] - initial_wealth)
## [1] 993.6644
hist(sim1[,n_days] - initial_wealth, breaks=30)
```

Histogram of sim1[, n_days] - initial_wealth

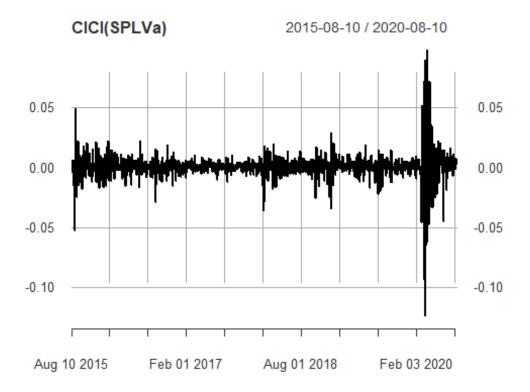


```
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)
##
## -9137.798
# note: this is a negative number (a loss, e.g. -500), but we conventionally
# express VaR as a positive number (e.g. 500)
#Safe/Defensive Approach
# Import a few stocks
mystocks = c("SHY", "SPLV", "USMV", "IEF", "EFAV")
getSymbols(mystocks)
## [1] "SHY" "SPLV" "USMV" "IEF" "EFAV"
SHY <- SHY["2015-08-10::2020-08-10"]
SPLV <- SPLV["2015-08-10::2020-08-10"]
USMV <- USMV["2015-08-10::2020-08-10"]
IEF <- IEF["2015-08-10::2020-08-10"]</pre>
EFAV <- EFAV["2015-08-10::2020-08-10"]
# Adjust for splits and dividends
SHYa = adjustOHLC(SHY)
SPLVa = adjustOHLC(SPLV)
USMVa = adjustOHLC(USMV)
IEFa = adjustOHLC(IEF)
```

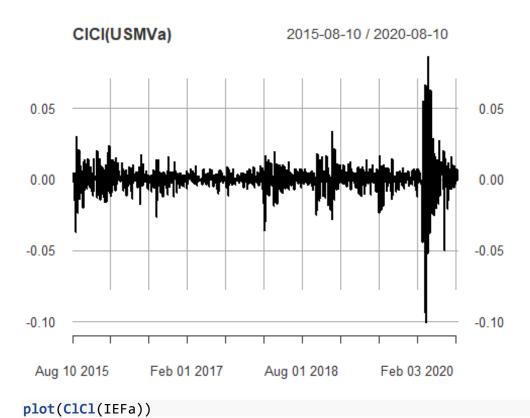
Look at close-to-close changes plot(ClCl(SHYa))

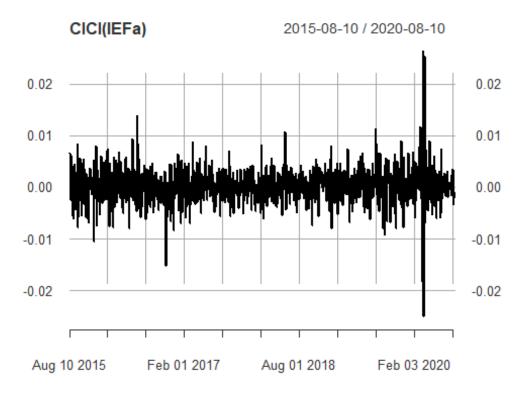


plot(ClCl(SPLVa))

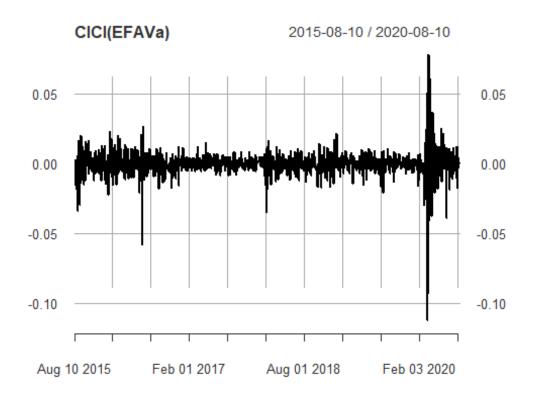


plot(C1C1(USMVa))

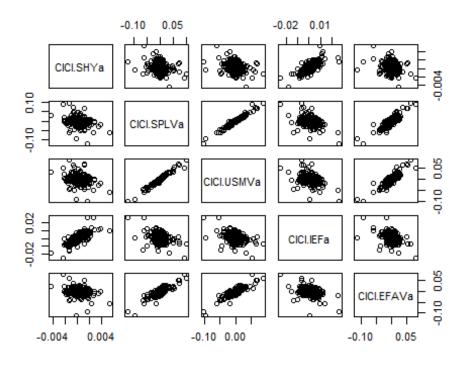


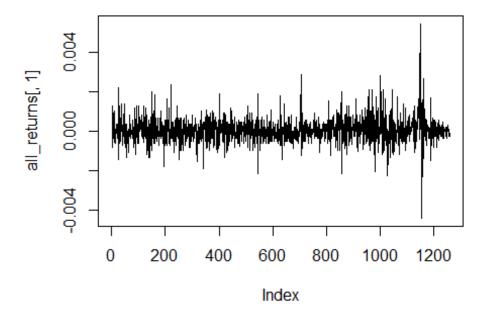


plot(C1C1(EFAVa))

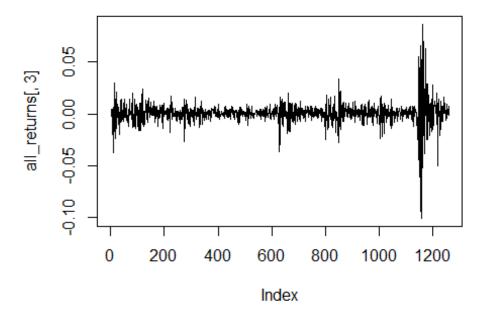


```
# Combine close to close changes in a single matrix
all returns =
cbind(ClCl(SHYa),ClCl(SPLVa),ClCl(USMVa),ClCl(IEFa),ClCl(EFAVa))
head(all returns)
##
                 C1C1.SHYa
                             ClCl.SPLVa
                                           C1C1.USMVa
                                                        ClCl.IEFa
ClCl.EFAVa
## 2015-08-10
                       NA
                                     NA
                                                  NA
                                                               NA
NA
## 2015-08-11 0.0012985598 -0.0036374121 -0.0021321962 0.006604435 -
0.0157964305
## 2015-08-12 0.0000000000 -0.0007822165 0.0028490741 -0.000374918 -
0.0017833705
## 2015-08-13 -0.0008252771 0.0018266962 0.0002366951 -0.002625401 -
0.0002976924
## 2015-08-14 -0.0004719882
                           0.0031273119
## 2015-08-17 0.0005902845 0.0033730409 0.0042402826 0.001788020
0.0001484857
# first row is NA because we didn't have a "before" in our data
all_returns = as.matrix(na.omit(all_returns))
N = nrow(all_returns)
# These returns can be viewed as draws from the joint distribution
# strong correlation, but certainly not Gaussian!
pairs(all returns)
```

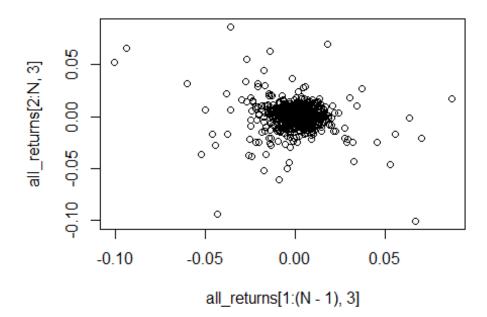




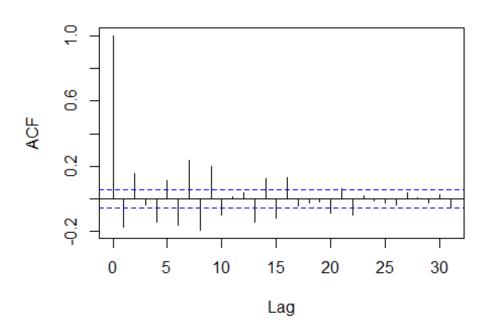
Look at the market returns over time
plot(all_returns[,3], type='1')



```
# are today's returns correlated with tomorrow's?
# not really!
plot(all_returns[1:(N-1),3], all_returns[2:N,3])
```

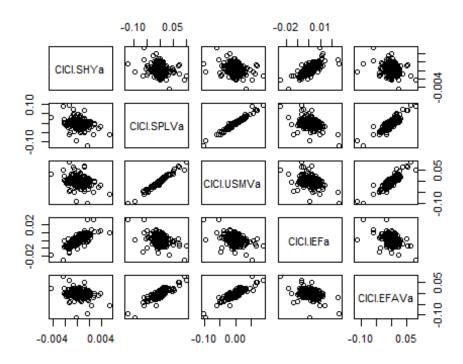


Series all_returns[, 3]



```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)
#### Now use a bootstrap approach
#### With more stocks
#mystocks = c("VIS", "PBE", "IXC", "XNTK", "VWO")
#myprices = getSymbols(mystocks, from = "2015-01-01")
# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in mystocks) {
  expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
  eval(parse(text=expr))
}
head(USMVa)
##
              USMV.Open USMV.High USMV.Low USMV.Close USMV.Volume
USMV.Adjusted
## 2015-08-10 37.94760 38.08284 37.94760
                                             38.05579
                                                           414000
```

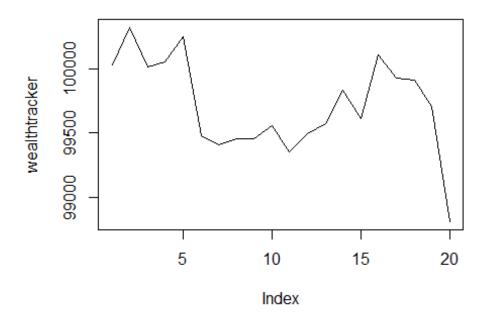
```
38.05579
## 2015-08-11 37.95662 38.03776 37.84843
                                            37.97465
                                                          2229400
37.97464
## 2015-08-12 37.84843
                       38.10087 37.61402
                                             38.08284
                                                         1055200
38.08285
## 2015-08-13
              38.10087
                        38.23611 37.96564
                                             38.09186
                                                           741700
38.09185
## 2015-08-14 38.04678 38.28119 38.03776
                                             38.27217
                                                          949000
38.27217
## 2015-08-17 38.25414 38.45249 38.06481
                                             38.43446
                                                          915300
38.43446
# Combine all the returns in a matrix
# Compute the returns from the closing prices
pairs(all returns)
```



```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# Update the value of your holdings
# Assumes an equal allocation to each asset
total_wealth = 100000
my_weights = c(0.2,0.2,0.2, 0.2, 0.2)
holdings = total_wealth*my_weights
```

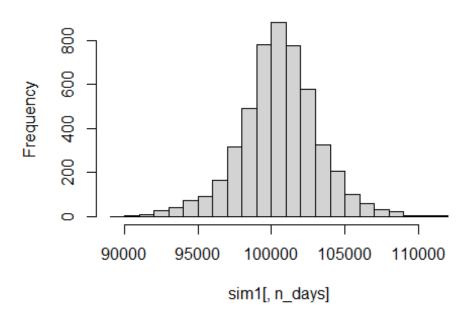
```
holdings = holdings*(1 + return.today)
holdings
              ClCl.SHYa ClCl.SPLVa ClCl.USMVa ClCl.IEFa ClCl.EFAVa
##
## 2017-04-18 20016.54
                          20013.78
                                     20004.18 20114.26
                                                          19969.95
# Now loop over two trading weeks
# let's run the following block of code 5 or 6 times
# to eyeball the variability in performance trajectories
## begin block
n_days = 20
wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
for(today in 1:n days) {
  return.today = resample(all_returns, 1, orig.ids=FALSE)
  holdings = holdings + holdings*return.today
  total_wealth = sum(holdings)
  wealthtracker[today] = total_wealth
}
total_wealth
## [1] 98808.36
plot(wealthtracker, type='l')
```



```
## end block
# Now simulate many different possible futures
# just repeating the above block thousands of times
initial wealth = 100000
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
  total wealth = initial wealth
  weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
  holdings = weights * total wealth
  n days = 20
  wealthtracker = rep(0, n_days)
  for(today in 1:n days) {
    return.today = resample(all_returns, 1, orig.ids=FALSE)
    holdings = holdings + holdings*return.today
    total wealth = sum(holdings)
    wealthtracker[today] = total_wealth
  }
  wealthtracker
# each row is a simulated trajectory
# each column is a data
head(sim1)
##
                                              [,4]
                 \lceil , 1 \rceil
                           [,2]
                                     [,3]
                                                       [55]
                                                                  [,6]
[,7]
## result.1 100026.19 100845.87 101235.98 101214.4 101759.9 102072.64
102388.33
## result.2 100124.60 100437.82 100511.48 100646.7 100921.2 100970.07
100904.41
## result.3 100289.18 100244.73 100270.27 100377.7 100586.8 100912.78
101135.11
## result.4 100197.45 100374.03 100435.98 100644.4 100665.7 100623.51
101811.33
## result.5 100099.63 100264.52 100344.61 100380.3 100548.6 100856.94
101018.88
## result.6 99917.44 99208.28 98770.66 98951.9
                                                    98393.4 98538.38
98323.86
                          [,9]
                                  [,10]
                                           [,11]
                 [,8]
                                                     [,12]
                                                              [,13]
## result.1 102215.70 102200.2 101781.9 101998.8 101658.6 101822.7 101991.1
## result.2 101839.74 101328.3 101484.9 101745.4 101817.1 101599.5 102034.0
## result.3 101847.59 101639.6 101126.3 100604.9 100165.5 100167.0 99951.8
## result.4 102312.15 102645.6 102502.9 102411.0 102460.3 101799.9 101676.7
## result.5 101295.43 101091.2 101026.0 101797.5 101737.6 101744.0 103290.8
## result.6 97879.01 101535.3 101218.4 100878.4 100724.6 100929.5 100846.5
##
               [,15]
                        [,16]
                                 [,17]
                                          [,18]
                                                    [,19]
                                                              [,20]
## result.1 101831.6 101477.1 101793.8 100614.4 100013.7 99806.57
## result.2 101784.2 101685.9 101810.1 102035.5 102086.8 102111.01
## result.3 100765.4 100806.9 100486.2 100154.6 104048.9 104131.21
## result.4 101615.3 102505.4 102533.1 102078.4 101993.9 102075.27
```

```
## result.5 103514.6 103771.5 103981.0 104308.0 104331.4 104809.16
## result.6 100723.5 100482.0 100557.5 100670.1 101000.7 100915.28
hist(sim1[,n_days], 25)
```

Histogram of sim1[, n_days]



```
# Profit/Loss
mean(sim1[,n_days])

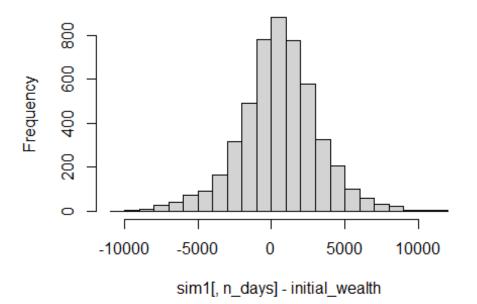
## [1] 100543.3

mean(sim1[,n_days] - initial_wealth)

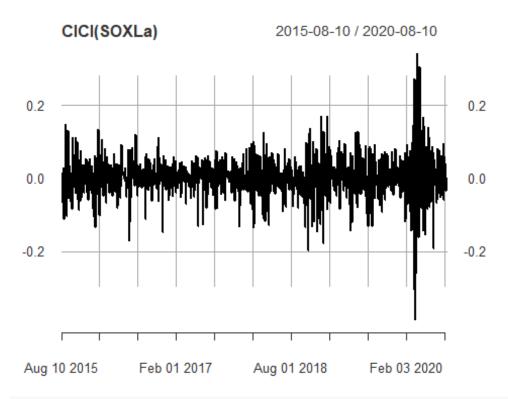
## [1] 543.2602

hist(sim1[,n_days] - initial_wealth, breaks=30)
```

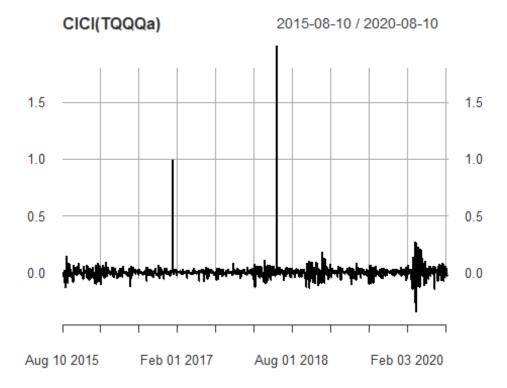
Histogram of sim1[, n_days] - initial_wealth



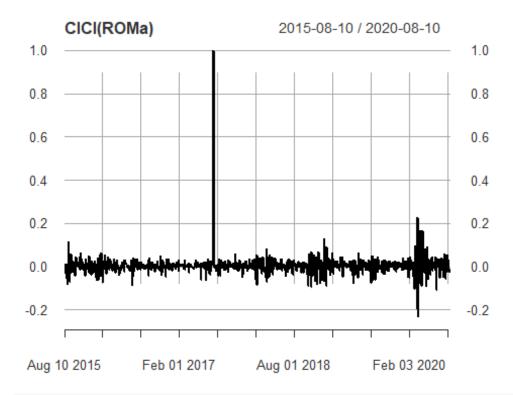
```
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)
##
## -3937.856
# note: this is a negative number (a loss, e.g. -500), but we conventionally
# express VaR as a positive number (e.g. 500)
mystocks = c("SOXL", "TQQQ", "ROM", "TECL", "VGT")
getSymbols(mystocks)
## [1] "SOXL" "TQQQ" "ROM" "TECL" "VGT"
SOXL <- SOXL["2015-08-10::2020-08-10"]
TQQQ <- TQQQ["2015-08-10::2020-08-10"]
ROM <- ROM["2015-08-10::2020-08-10"]
TECL <- TECL["2015-08-10::2020-08-10"]
VGT <- VGT["2015-08-10::2020-08-10"]
# Adjust for splits and dividends
SOXLa = adjustOHLC(SOXL)
TQQQa = adjustOHLC(TQQQ)
ROMa = adjustOHLC(ROM)
TECLa = adjustOHLC(TECL)
VGTa = adjustOHLC(VGT)
```



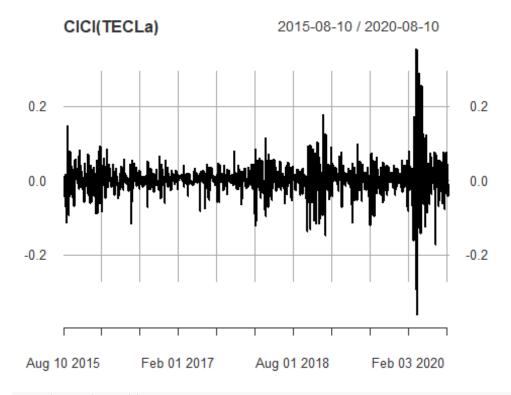
plot(ClCl(TQQQa))



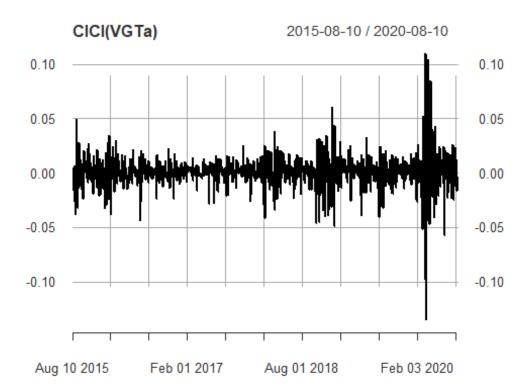
plot(ClCl(ROMa))



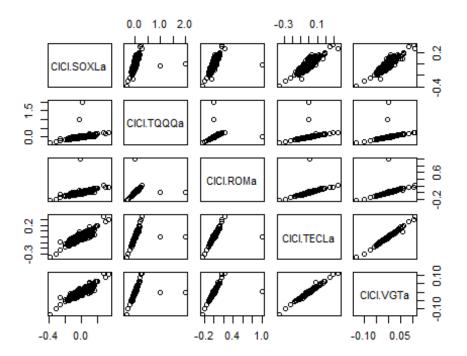
plot(ClCl(TECLa))



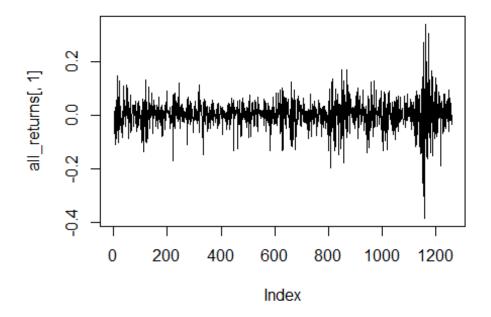
plot(C1C1(VGTa))



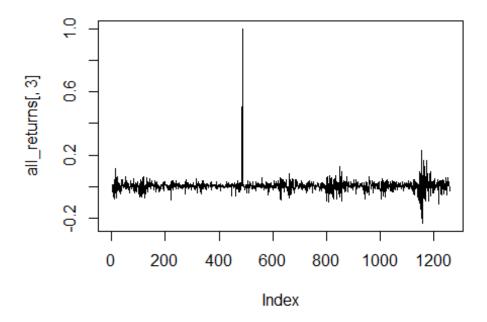
```
# Combine close to close changes in a single matrix
all returns =
cbind(ClCl(SOXLa),ClCl(TQQQa),ClCl(ROMa),ClCl(TECLa),ClCl(VGTa))
head(all returns)
                                          C1C1.ROMa
##
               ClCl.SOXLa
                            ClCl.TQQQa
                                                      ClCl.TECLa
                                                                    ClCl.VGTa
## 2015-08-10
                       NA
                                    NA
                                                 NA
                                                              NA
                                                                           NA
## 2015-08-11 -0.06798249 -0.038128800 -0.032714362 -0.044438582 -0.016334762
                                                     0.011974380
## 2015-08-12 0.01686275
                           0.008818633
                                        0.010838365
                                                                  0.004851171
## 2015-08-13 -0.03046668 -0.004500571 -0.002971244 -0.006053963 -0.002506703
## 2015-08-14 -0.01750191 0.005390350
                                        0.006867064
                                                    0.012458500
                                                                  0.005305287
## 2015-08-17 0.02995951 0.024558959
                                        0.012482306
                                                     0.017500656
                                                                  0.006480854
# first row is NA because we didn't have a "before" in our data
all returns = as.matrix(na.omit(all returns))
N = nrow(all returns)
# These returns can be viewed as draws from the joint distribution
# strong correlation, but certainly not Gaussian!
pairs(all_returns)
```



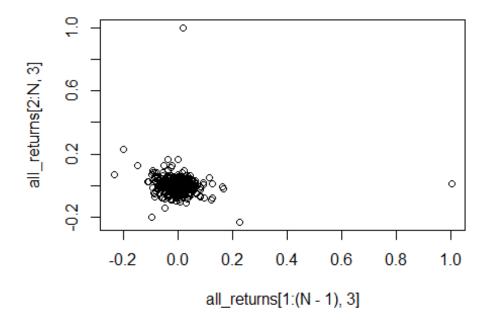
plot(all_returns[,1], type='l')



Look at the market returns over time
plot(all_returns[,3], type='l')

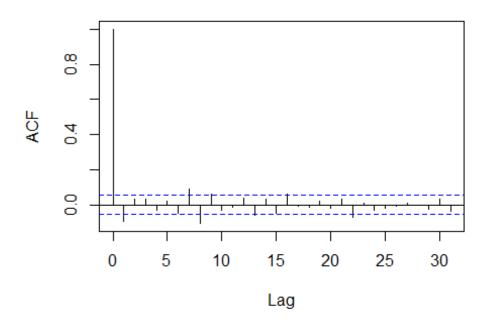


```
# are today's returns correlated with tomorrow's?
# not really!
plot(all_returns[1:(N-1),3], all_returns[2:N,3])
```



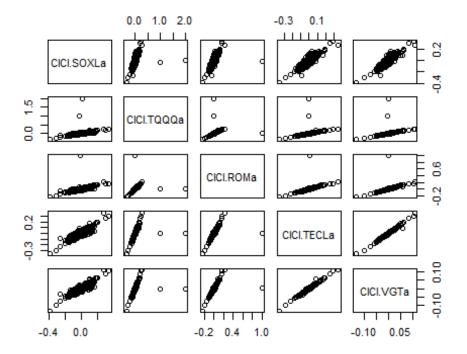
An autocorrelation plot: nothing there
acf(all_returns[,3])

Series all_returns[, 3]



```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)
#### Now use a bootstrap approach
#### With more stocks
#mystocks = c("VIS", "PBE", "IXC", "XNTK", "VWO")
#myprices = getSymbols(mystocks, from = "2015-01-01")
# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in mystocks) {
  expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
  eval(parse(text=expr))
}
head(USMVa)
              USMV.Open USMV.High USMV.Low USMV.Close USMV.Volume
USMV.Adjusted
## 2015-08-10 37.94760 38.08284 37.94760
                                             38.05579
                                                           414000
38.05579
## 2015-08-11 37.95662 38.03776 37.84843
                                             37.97465
                                                          2229400
```

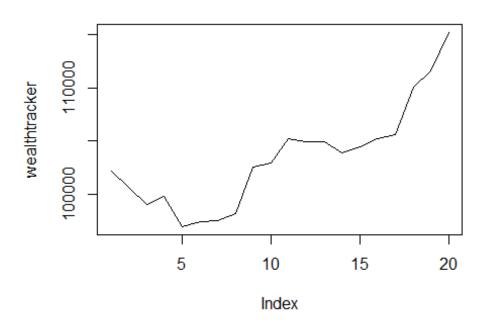
```
37.97464
## 2015-08-12 37.84843 38.10087 37.61402
                                            38.08284
                                                          1055200
38.08285
## 2015-08-13 38.10087
                        38.23611 37.96564
                                            38.09186
                                                          741700
38.09185
## 2015-08-14 38.04678 38.28119 38.03776
                                             38.27217
                                                          949000
38,27217
## 2015-08-17 38.25414 38.45249 38.06481
                                                          915300
                                            38.43446
38.43446
# Combine all the returns in a matrix
# Compute the returns from the closing prices
pairs(all returns)
```



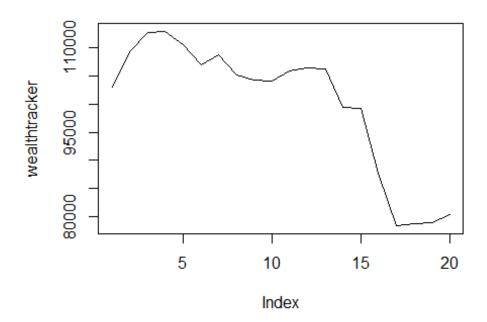
```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# Update the value of your holdings
# Assumes an equal allocation to each asset
total_wealth = 100000
my_weights = c(0.2,0.2,0.2, 0.2, 0.2)
holdings = total_wealth*my_weights
holdings = holdings*(1 + return.today)
```

```
holdings
##
              ClCl.SOXLa ClCl.TQQQa ClCl.ROMa ClCl.TECLa ClCl.VGTa
## 2020-08-04
                 20859.2
                           20260.03 20044.85
                                                20179.95 20065.53
# Now loop over two trading weeks
# let's run the following block of code 5 or 6 times
# to eyeball the variability in performance trajectories
## begin block
n_days = 20
wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
for(today in 1:n_days) {
  return.today = resample(all_returns, 1, orig.ids=FALSE)
  holdings = holdings + holdings*return.today
  total wealth = sum(holdings)
  wealthtracker[today] = total_wealth
}
total_wealth
## [1] 115212.3
plot(wealthtracker, type='l')
```



```
# Now simulate many different possible futures
# just repeating the above block thousands of times
initial_wealth = 100000
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
  total_wealth = initial_wealth
  weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
  holdings = weights * total wealth
  n days = 20
  wealthtracker = rep(0, n_days)
  for(today in 1:n_days) {
    return.today = resample(all_returns, 1, orig.ids=FALSE)
    holdings = holdings + holdings*return.today
    total_wealth = sum(holdings)
    wealthtracker[today] = total_wealth
  wealthtracker
}
plot(wealthtracker, type='l')
```

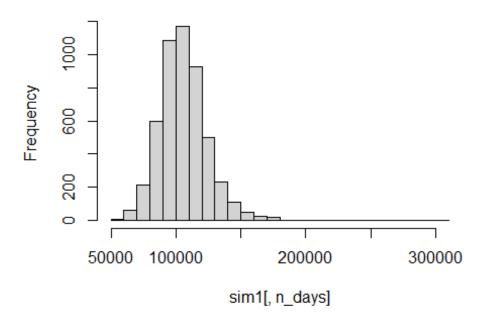


```
# each row is a simulated trajectory
# each column is a data
head(sim1)

## [,1] [,2] [,3] [,4] [,5] [,6]
[,7]
## result.1 101327.18 101789.37 102554.4 92477.9 95087.88 100619.57
101913.5
```

```
## result.2 103228.81 102194.19 101850.2 102743.6 105290.34 108204.78
107597.0
## result.3 99318.68 104084.23 102581.5 104338.5 101056.91 100612.02
102163.4
## result.4 100355.25 101420.57 100465.9 100891.8 103378.98 103593.88
103401.0
## result.5 100125.29 99800.53 100257.4 101644.0 100473.52 97946.73
97784.7
## result.6 105874.65 106113.05 106626.8 109886.2 111073.59 117445.29
119286.1
##
                [8,]
                          [,9]
                                   [,10]
                                             [,11]
                                                       [,12]
                                                                [,13]
[,14]
## result.1 98554.36 91816.08 91828.47 94456.16 90941.33 91725.34
93067.62
## result.2 108522.04 112088.08 112653.85 114884.93 118865.65 120001.36
## result.3 100574.62 111761.99 112421.69 107911.51 112707.25 125378.24
128781.98
## result.4 97822.16 98213.58 96557.53 100984.74 100944.33 98726.67
98846.70
## result.5 96036.30 96320.29 88170.57 72595.53 72971.27 73733.73
76594.78
## result.6 122542.09 123396.69 118814.98 119602.37 120782.31 120500.61
123275.44
               [,15]
##
                         [,16]
                                   [,17]
                                             [,18]
                                                       [,19]
## result.1 93512.05 95712.57 132118.94 134009.69 129163.83 130898.62
## result.2 118016.21 119089.72 112694.54 112504.04 110823.62 112862.37
## result.3 135380.75 135240.54 131071.02 131135.31 135628.53 135854.92
## result.4 102113.74 102532.65 106592.03 100475.12 99821.33 93521.49
## result.5 77404.99 76767.18 75664.82 69199.46 69509.62 70657.60
## result.6 124189.91 126359.46 126846.16 129497.81 128860.63 134598.47
hist(sim1[,n_days], 25)
```

Histogram of sim1[, n_days]

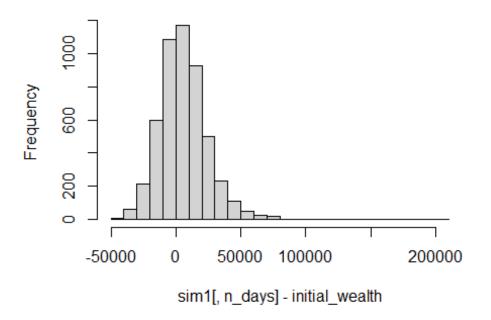


```
# Profit/Loss
mean(sim1[,n_days])
## [1] 105786.6

mean(sim1[,n_days] - initial_wealth)
## [1] 5786.605

hist(sim1[,n_days]- initial_wealth, breaks=30)
```

Histogram of sim1[, n_days] - initial_wealth



```
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)
## 5%
## -20791.28
```

Question 4

First, the data was preprocessed to ensure that the results were as meaningful of as possible. This meant removing all social media users who were indicated as writing spam, anything categorized as adult, and uncategorized. These users were not the main audience for the product which is why they were removed from the analysis. After conducting all the data preprocessing, a PCA was conducted on the cleaned data. The first result of the PCA had all loadings included, but after reviewing the result it was determined that 16 loadings would be enough as it accounted for 80% of the variance. This did not mean that all the marketing segments would be used for this analysis because after closely reviewing the loads only about 12 of the loads were of interest as some of them did not really provide a lot of insight.

Market segment 1: High... * Religion * Food * Parenting * School * Family * Beauty * Crafts

Market Segment 2: High... * Religion * Food * Parenting * Sports fandom * Food * School * Family Low... * Cooking * Photo sharing * Fashion

Market Segment 3: High... * Politics * Travel * Computers * News * Automotive Low... * Health and nutrient * Personal fitness

Market Segment 4: High... * Health and Nutrient * Personal fitness * Outdoors Low... * College and University * Online gaming

Market Segment 5: High... * Photo sharing Low... * College and university * Online gaming

Market Segment 6: High... * Beauty * Cooking * Fashion * Chatter * Shopping

Market Segment 7: High... * Automotive * Online Gaming * Sports playing Low... * Arts * TV Film

Market Segment 8: High... * Automotive * News * Tv film * Computers * Travel

Market Segment 9: High... * Dating * Home and gardening * School Low... * Music

Market Segment 10: High... * Music * Small business Low... * Arts * Crafts

Market Segment 11: High... * Home and gardening * Current events * Eco Low... * Business * Craft

Market Segment 12: High... * Music * Eco Low... * Small Business * Business

```
library(tidyverse)
library(randomForest)
library(splines)
socialmarketing <-</pre>
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/social marketi
ng.csv")
#data cleaning
socialmarketing <- subset(socialmarketing, socialmarketing$spam != 1</pre>
socialmarketing$adult != 1)
socialmarketing <- socialmarketing[,c(-36,-37,-6)]</pre>
socialmarketing_PCA = prcomp(socialmarketing[,-1], rank=34, scale=TRUE)
head(socialmarketing_PCA$rotation)
##
                                   PC2
                                               PC3
                                                           PC4
                                                                      PC5
                        PC1
## chatter
                 0.12434039 -0.20982909 0.06524036 -0.11722123
                                                               0.18510271
## current events 0.09699472 -0.07067962 0.04916933 -0.03246847
                                                               0.05611240
## travel
                 0.11730148 -0.05389072 0.42369620 0.14286147
                                                               0.01119277
## photo_sharing 0.17781387 -0.31768609 -0.02387418 -0.15742650
                                                               0.21634222
## tv film
                 0.09362971 -0.07034052 0.08803030 -0.08908232 -0.20446274
## sports_fandom 0.29469739 0.31163967 -0.04609136 -0.05311160 0.03602160
##
                          PC6
                                     PC7
                                                 PC8
                                                            PC9
PC10
## chatter
                 0.01404138
## current events -0.138583473 -0.03977839 0.05924758 -0.08925141
0.09406931
                  0.162193765 -0.10163928 -0.30191648 -0.10922170 -
## travel
```

```
0.04346621
## photo sharing
               0.11712987
               -0.073695612 -0.52145628 0.23925268 -0.07782077
## tv film
0.06701168
## sports_fandom
               0.04410351
##
                     PC11
                                  PC12
                                             PC13
                                                       PC14
PC15
## chatter
               -0.06861798 -0.0256648479 0.031322782 0.08911691
0.111154628
## current events 0.52102984 0.7931842845 -0.132203431 -0.12535225
0.009037982
## travel
                0.05303062 -0.0195845537 0.029047436 0.04980604
0.015196973
## photo_sharing -0.01045177 -0.0931551471 0.049490001 -0.01435000
0.054631909
               -0.05937961 -0.0007197019 0.036967552 0.10786090 -
## tv film
0.046027340
## sports fandom
               -0.02542150 -0.0131149773 0.005355396 -0.02308185 -
0.018847501
##
                     PC16
                                  PC17
                                             PC18
                                                        PC19
PC20
## chatter
               -0.13617038 -0.0072142691 -0.007999102 0.129032479
0.04408413
## current_events    0.03263011 -0.0001772328 -0.006482393 -0.018767323 -
0.01739717
## travel
               0.04492248
## photo sharing
               -0.07076666 -0.0423932573 0.003985895 -0.053313244 -
0.06331564
## tv film
               0.21113410
## sports fandom
               -0.01769400 -0.1280913874 0.018335674 0.097364976 -
0.14613414
                     PC21
                                PC22
                                           PC23
##
                                                       PC24
PC25
## chatter
                0.11389699 -0.09291684 -0.135977662 -0.212028466 -
0.0001442715
## current events 0.01353753 0.01723101 -0.009880252 0.001587184 -
0.0001560642
## travel
               -0.07638891   0.03136240   0.207438671   -0.224706600   -
0.3397394845
               0.16065752 -0.26896243 -0.221546022 -0.389312849
## photo_sharing
0.1152693914
## tv film
               -0.21396447 -0.55890816 0.087153576 0.104145524 -
0.0454526295
## sports fandom
                0.07011800 -0.05424472 0.122561944 0.079348635
0.5827530313
##
                     PC26
                                 PC27
                                            PC28
                                                       PC29
```

```
PC30
                 -0.31130188 -0.135684225   0.600203219   0.12745889 -
## chatter
0.0045253373
## current events 0.02731926 -0.008395394 -0.003171978 -0.02466891
0.0005984978
                 -0.05967286   0.359759745   0.111862737   -0.24280762   -
## travel
0.4479775621
## photo sharing
                 0.29673070 0.230420480 -0.468741794 0.01806054 -
0.0313608728
## tv film
                 -0.03435094 -0.122190254 -0.022116674 -0.03257722
0.0582150174
## sports fandom
                 0.0614328419
##
                        PC31
                                    PC32
                                                 PC33
                 -0.09076177 0.010922486 0.009432833
## chatter
## current_events -0.01161154 0.005481428 -0.005872545
## travel
                 ## photo sharing
                0.07931095 -0.021265310 -0.034530550
## tv film
                 -0.01689476   0.166292214   -0.051227993
## sports_fandom -0.00597990 -0.008890838 -0.014946669
summary(socialmarketing_PCA)
## Importance of components:
##
                           PC1
                                  PC2
                                          PC3
                                                  PC4
                                                         PC5
                                                                PC6
PC7
## Standard deviation
                         2.110 1.68608 1.59286 1.53355 1.4792 1.36842
1,27377
## Proportion of Variance 0.135 0.08615 0.07689 0.07127 0.0663 0.05674
0.04917
## Cumulative Proportion 0.135 0.22112 0.29800 0.36927 0.4356 0.49232
0.54148
##
                             PC8
                                    PC9
                                          PC10
                                                 PC11
                                                         PC12
                                                                PC13
PC14
## Standard deviation
                         1.19216 1.06095 0.9933 0.9681 0.96178 0.93977
0.92323
## Proportion of Variance 0.04307 0.03411 0.0299 0.0284 0.02803 0.02676
0.02583
## Cumulative Proportion 0.58455 0.61866 0.6486 0.6770 0.70499 0.73176
0.75758
##
                            PC15
                                   PC16
                                           PC17
                                                   PC18
                                                           PC19
                                                                  PC20
PC21
## Standard deviation
                         0.91593 0.85450 0.80886 0.75371 0.69818 0.68810
0.65473
## Proportion of Variance 0.02542 0.02213 0.01983 0.01721 0.01477 0.01435
## Cumulative Proportion 0.78301 0.80513 0.82496 0.84217 0.85694 0.87129
0.88428
##
                            PC22
                                   PC23
                                           PC24
                                                   PC25
                                                           PC26
                                                                  PC27
PC28
```

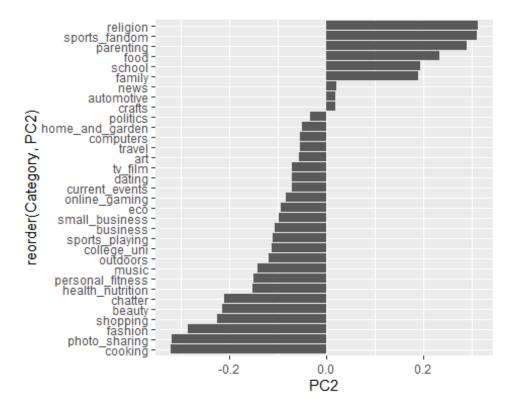
```
## Standard deviation
                          0.65059 0.63989 0.63728 0.61730 0.60211 0.59482
0.58787
## Proportion of Variance 0.01283 0.01241 0.01231 0.01155 0.01099 0.01072
0.01047
## Cumulative Proportion 0.89711 0.90952 0.92182 0.93337 0.94436 0.95508
0.96555
##
                             PC29
                                     PC30
                                             PC31
                                                     PC32
                                                            PC33
## Standard deviation
                          0.55071 0.48575 0.47615 0.43880 0.4223
## Proportion of Variance 0.00919 0.00715 0.00687 0.00583 0.0054
## Cumulative Proportion 0.97474 0.98189 0.98876 0.99460 1.0000
socialmarketing PCA2 = prcomp(socialmarketing[,-1], rank=16, scale=TRUE)
summary(socialmarketing_PCA2)
## Importance of first k=16 (out of 33) components:
##
                                    PC2
                                                    PC4
                                                           PC5
                                                                    PC6
                            PC1
PC7
                          2.110 1.68608 1.59286 1.53355 1.4792 1.36842
## Standard deviation
1,27377
## Proportion of Variance 0.135 0.08615 0.07689 0.07127 0.0663 0.05674
0.04917
## Cumulative Proportion 0.135 0.22112 0.29800 0.36927 0.4356 0.49232
0.54148
##
                              PC8
                                      PC9
                                            PC10
                                                   PC11
                                                           PC12
                                                                   PC13
PC14
## Standard deviation
                          1.19216 1.06095 0.9933 0.9681 0.96178 0.93977
0.92323
## Proportion of Variance 0.04307 0.03411 0.0299 0.0284 0.02803 0.02676
0.02583
## Cumulative Proportion 0.58455 0.61866 0.6486 0.6770 0.70499 0.73176
0.75758
##
                             PC15
                                     PC16
## Standard deviation
                          0.91593 0.85450
## Proportion of Variance 0.02542 0.02213
## Cumulative Proportion 0.78301 0.80513
loadings = socialmarketing PCA2$rotation %>%
  as.data.frame %>% rownames_to_column('Category')
loadings %>%
  select(Category, PC2) %>%
  arrange(desc(PC2))
##
              Category
                               PC2
## 1
              religion 0.31318929
## 2
         sports fandom 0.31163967
## 3
             parenting 0.29012449
## 4
                  food 0.23377946
## 5
                school 0.19415706
## 6
                family 0.18932264
## 7
                  news 0.02229173
```

```
## 8
            automotive
                         0.02031965
## 9
                 crafts
                         0.02008688
## 10
              politics -0.03307619
## 11
       home_and_garden -0.04839369
## 12
             computers -0.05309321
## 13
                travel -0.05389072
## 14
                    art -0.05491716
## 15
               tv film -0.07034052
## 16
                dating -0.07061193
## 17
        current events -0.07067962
## 18
         online_gaming -0.08343524
## 19
                    eco -0.09374260
## 20
        small business -0.09714565
## 21
               business -0.10636760
## 22
        sports_playing -0.10903269
## 23
           college uni -0.11264856
## 24
              outdoors -0.11839813
## 25
                  music -0.14191969
## 26 personal fitness -0.14935045
## 27 health nutrition -0.15063628
## 28
               chatter -0.20982909
## 29
                 beauty -0.21299485
## 30
              shopping -0.22347106
## 31
               fashion -0.28506359
## 32
         photo sharing -0.31768609
## 33
               cooking -0.31977474
loadings %>%
  select(Category, PC3) %>%
  arrange(desc(PC3))
##
              Category
                                 PC3
## 1
                         0.489199287
              politics
## 2
                 travel
                         0.423696202
## 3
                         0.365368765
             computers
## 4
                         0.336846723
                   news
## 5
            automotive
                         0.189918066
## 6
              business
                         0.101837280
## 7
        small business
                         0.098296766
## 8
               tv_film
                         0.088030302
## 9
           college_uni
                         0.083677564
                         0.065240356
## 10
                chatter
## 11
         online_gaming
                         0.053276980
## 12
                         0.050728140
                    art
## 13
        current events
                         0.049169326
## 14
        sports_playing
                         0.040492264
## 15
               shopping
                         0.036967301
## 16
                 dating
                         0.030807397
## 17
       home_and_garden
                         0.019968845
## 18
                 crafts
                         0.003101066
```

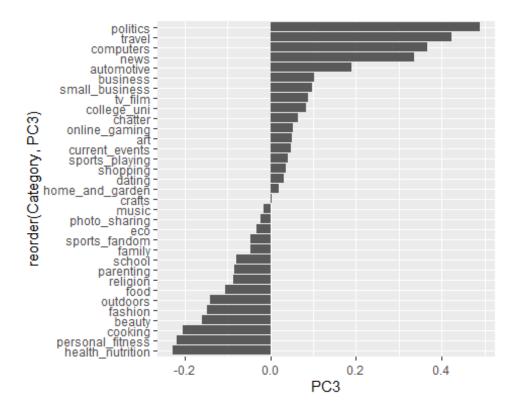
```
## 19
                 music -0.015176251
## 20
         photo sharing -0.023874179
## 21
                    eco -0.032698589
## 22
         sports_fandom -0.046091360
## 23
                family -0.047131188
## 24
                school -0.078432899
             parenting -0.084044430
## 25
## 26
              religion -0.086883035
## 27
                   food -0.105701916
## 28
              outdoors -0.141075535
## 29
               fashion -0.148632764
## 30
                beauty -0.158714742
## 31
               cooking -0.204235400
## 32 personal_fitness -0.219407444
## 33 health_nutrition -0.227408956
loadings %>%
  select(Category, PC4) %>%
  arrange(desc(PC4))
##
              Category
                                 PC4
## 1
      health nutrition
                         0.463218592
## 2
      personal_fitness
                         0.443782383
## 3
              outdoors
                         0.413458799
## 4
              politics
                         0.194231190
## 5
                         0.176097051
                   news
## 6
                travel
                         0.142861467
                         0.135653450
## 7
             computers
## 8
                         0.120197612
                    eco
## 9
                   food
                         0.076474116
## 10
            automotive
                         0.038028567
## 11
                dating
                         0.027507581
## 12
       home and garden
                         0.009065924
## 13
              business -0.014359814
## 14
               cooking -0.015092706
## 15
                 crafts -0.023027670
## 16
        current_events -0.032468473
## 17
             parenting -0.043698338
## 18
         sports_fandom -0.053111605
## 19
                    art -0.061674210
## 20
               religion -0.061955703
## 21
                family -0.070429227
## 22
        small_business -0.081253975
## 23
                 music -0.083573201
## 24
                school -0.083715710
## 25
               tv_film -0.089082322
## 26
              shopping -0.107490730
## 27
               chatter -0.117221234
## 28
               fashion -0.142549602
## 29
                beauty -0.150386888
```

```
## 30     photo_sharing -0.157426501
## 31     sports_playing -0.177019154
## 32     online_gaming -0.222104495
## 33     college_uni -0.256605911

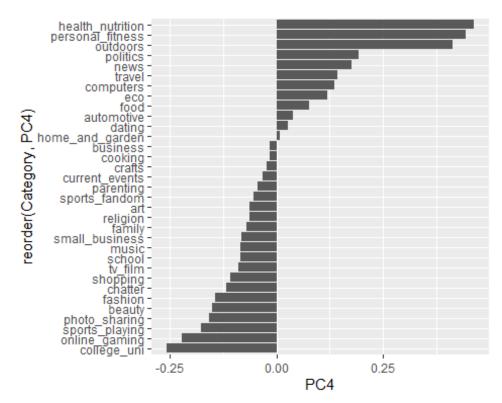
ggplot(loadings) +
    geom_col(aes(x=reorder(Category, PC2), y=PC2)) +
    coord_flip()
```



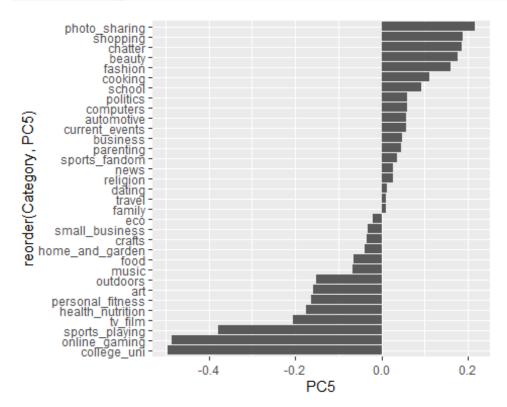
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC3), y=PC3)) +
  coord_flip()
```



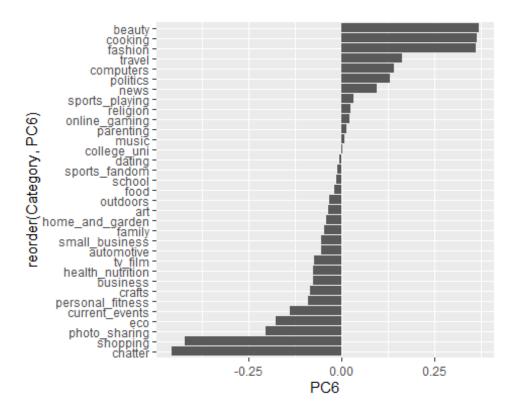
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC4), y=PC4)) +
  coord_flip()
```



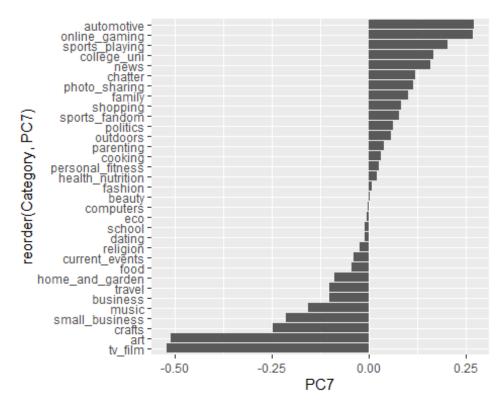
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC5), y=PC5)) +
  coord_flip()
```



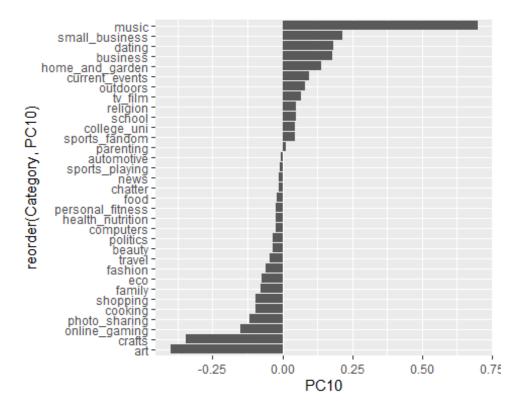
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC6), y=PC6)) +
  coord_flip()
```



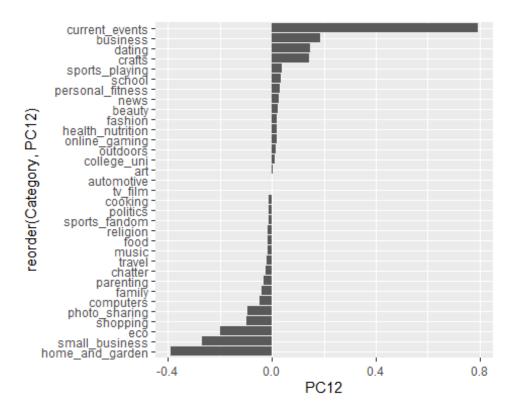
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC7), y=PC7)) +
  coord_flip()
```



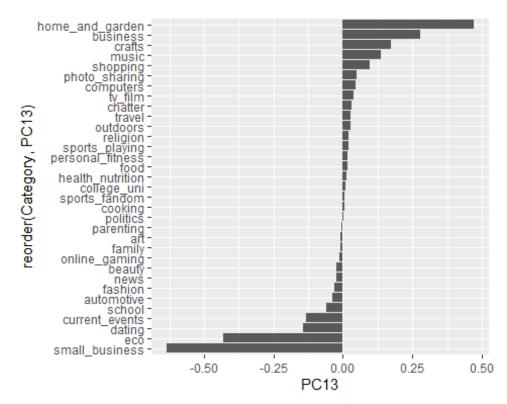
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC10), y=PC10)) +
  coord_flip()
```



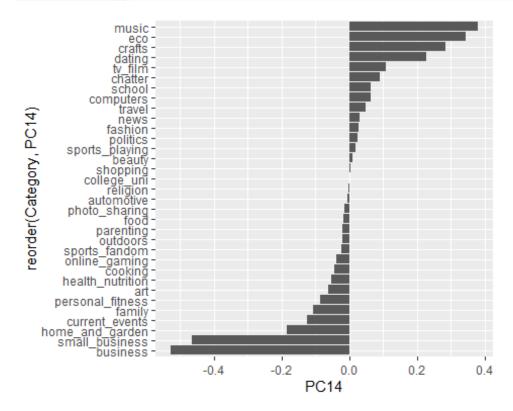
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC12), y=PC12)) +
  coord_flip()
```



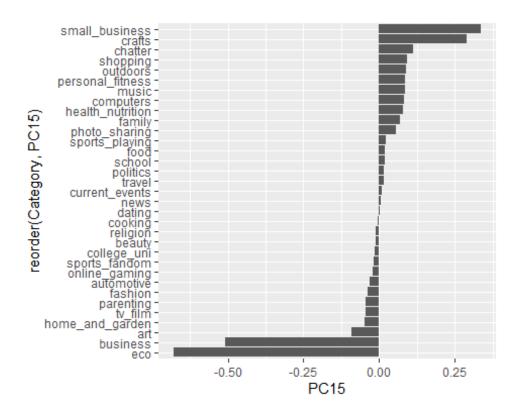
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC13), y=PC13)) +
  coord_flip()
```



```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC14), y=PC14)) +
  coord_flip()
```



```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC15), y=PC15)) +
  coord_flip()
```



Question 5

The dataset used was the training and test datasets from C50 Reuters. Prior to converting the documents into a corpus, I had to read it in using a readerPlain function. Once this task was completed, I converted all the documents from all the authors into corpus then cleaned the corpus to remove extra white space, punctuation, and numbers. Along with that I also removed any English stop words that were used in the document. I then converted my corpus for each of the authors into a document term matrix. Once I had all 50 document matrices, I removed any words that didn't appear in 95% of all 50 documents which improved the overall sparsity for each of my 50 document term matrices. I then calculated the TF IDF for each of the author's document term matrices. I did this for both the training dataset and the test dataset training. The only slight difference between the procedure for the test dataset was that I removed the words from the test dataset that were not present on the training set before training my model.

Once most of my preprocessing was completed, I continued with training my model. To train my model, I used the TF IDF of every author's document term matrix from the training dataset. I used a decision tree model using the rpart algorithm to train my model. Once I had the model, I then predicted my results using model and the TF IDF from the test dataset. Lastly, I calculated the accuracy. I did not continue to optimize my model since my

out of sample accuracy was 99% which is extremely close to what I would've gotten for the in-sample accuracy.

```
library(tm)
library(tidyverse)
library(slam)
library(proxy)
library(tidytext)
library(dplyr)
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths
list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,</pre>
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]</pre>
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
p <- function(..., sep='') {</pre>
  paste(..., sep=sep, collapse=sep)
for (i in list_folders){
  folder = i
  assign(i,i)
file_list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))
```

```
simon = lapply(file_list, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(simon) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(simon))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM simon = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
```

```
DTM simon = removeSparseTerms(DTM simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf simon = weightTfIdf(DTM simon)
simon_DF <- as.matrix(tfidf_simon)</pre>
DTM_simon2 <- as.data.frame(as.matrix(DTM_simon), stringsAsFactors=False)</pre>
#Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file_listA = Sys.glob(filename)
aaron = lapply(file_listA, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file listA %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(aaron) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                     # make everything
Lowercase
                                                   # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
```

```
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM aaron = DocumentTermMatrix(my documents)
## You can inspect its entries...
DTM aaron = removeSparseTerms(DTM aaron, 0.95)
tfidf aaron = weightTfIdf(DTM aaron)
aaron_DF <- as.matrix(tfidf_aaron)</pre>
DTM aaron2 <- as.data.frame(as.matrix(DTM aaron), stringsAsFactors=False)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlexanderSmith,'/*.txt'))
alex = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(alex) = mynames
## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alex))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM alex = DocumentTermMatrix(my documents)
DTM alex = removeSparseTerms(DTM alex, 0.95)
tfidf alex = weightTfIdf(DTM alex)
alex DF <- as.matrix(tfidf alex)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', AlanCrosby, '/*.txt'))
alan = lapply(file listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(alan) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alan))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM alan = DocumentTermMatrix(my documents)
DTM_alan = removeSparseTerms(DTM_alan, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf alan = weightTfIdf(DTM alan)
```

```
alan DF <- as.matrix(tfidf alan)</pre>
DTM alan2 <- as.data.frame(as.matrix(DTM alan), stringsAsFactors=False)</pre>
# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function
####
# Dimensionality reduction
####
# Now PCA on term frequencies
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
tm map(content transformer(removePunctuation)) %>% # remove punctuation
```

```
tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM ben = DocumentTermMatrix(my documents)
DTM ben = removeSparseTerms(DTM ben, 0.95)
tfidf_ben = weightTfIdf(DTM_ben)
ben DF <- as.matrix(tfidf ben)</pre>
DTM ben2 <- as.data.frame(as.matrix(DTM ben), stringsAsFactors=False)</pre>
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM ben = DocumentTermMatrix(my documents)
DTM ben = removeSparseTerms(DTM ben, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf ben = weightTfIdf(DTM ben)
ben DF <- as.matrix(tfidf ben)</pre>
DTM ben2 <- as.data.frame(as.matrix(DTM ben), stringsAsFactors=False)</pre>
##Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))
```

```
brad = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(brad) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(brad))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM brad = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
```

```
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad = removeSparseTerms(DTM_brad, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf brad = weightTfIdf(DTM brad)
brad_DF <- as.matrix(tfidf_brad)</pre>
DTM brad2 <- as.data.frame(as.matrix(DTM brad), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))
darren = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(darren) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
  tm map(content transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
```

```
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM darren = DocumentTermMatrix(my documents)
DTM darren = removeSparseTerms(DTM darren, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren = weightTfIdf(DTM_darren)
darren DF <- as.matrix(tfidf darren)</pre>
DTM darren2 <- as.data.frame(as.matrix(DTM darren), stringsAsFactors=False)</pre>
##Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))
david = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                         # make everything
Lowercase
```

```
tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM david = DocumentTermMatrix(my_documents)
DTM david = removeSparseTerms(DTM david, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf david = weightTfIdf(DTM david)
david DF <- as.matrix(tfidf david)</pre>
DTM_david2 <- as.data.frame(as.matrix(DTM_david), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))
edna = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(edna ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(edna ))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                     # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_edna = DocumentTermMatrix(my_documents)
DTM_edna = removeSparseTerms(DTM_edna , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf edna = weightTfIdf(DTM edna )
edna DF <- as.matrix(tfidf edna )</pre>
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)</pre>
###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))
eric = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(eric ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM eric = DocumentTermMatrix(my documents)
DTM eric = removeSparseTerms(DTM eric , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf eric = weightTfIdf(DTM eric )
eric DF <- as.matrix(tfidf eric )</pre>
DTM_eric2 <- as.data.frame(as.matrix(DTM_eric ), stringsAsFactors=False)</pre>
###Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))
```

```
fumiko = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(fumiko ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(fumiko ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM fumiko = DocumentTermMatrix(my documents)
DTM fumiko = removeSparseTerms(DTM fumiko , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko = weightTfIdf(DTM_fumiko )
fumiko DF <- as.matrix(tfidf fumiko )</pre>
```

```
DTM_fumiko2 <- as.data.frame(as.matrix(DTM_fumiko ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))
graham = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(graham ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
```

```
t"))
## create a doc-term-matrix from the corpus
DTM graham = DocumentTermMatrix(my documents)
DTM graham = removeSparseTerms(DTM graham , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham = weightTfIdf(DTM_graham )
graham DF <- as.matrix(tfidf graham )</pre>
DTM_graham2 <- as.data.frame(as.matrix(DTM_graham ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))
heather = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(heather ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
```

```
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_heather = DocumentTermMatrix(my_documents)
DTM heather = removeSparseTerms(DTM heather , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf heather = weightTfIdf(DTM heather )
heather DF <- as.matrix(tfidf heather )</pre>
DTM_heather2 <- as.data.frame(as.matrix(DTM_heather ),</pre>
stringsAsFactors=False)
###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))
jan = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jan ) = mynames
## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jan ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                          # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jan = DocumentTermMatrix(my documents)
DTM jan = removeSparseTerms(DTM jan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jan = weightTfIdf(DTM jan )
jan_DF <- as.matrix(tfidf_jan )</pre>
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))
jane = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jane ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jane = DocumentTermMatrix(my documents)
DTM_jane = removeSparseTerms(DTM_jane , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jane = weightTfIdf(DTM jane )
jane_DF <- as.matrix(tfidf_jane )</pre>
DTM_jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)</pre>
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))
```

```
jim = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jim ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_jim = DocumentTermMatrix(my_documents)
DTM_jim = removeSparseTerms(DTM_jim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jim = weightTfIdf(DTM jim )
jim_DF <- as.matrix(tfidf_jim )</pre>
DTM_jim2 <- as.data.frame(as.matrix(DTM_jim ), stringsAsFactors=False)</pre>
```

```
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))
jo = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jo ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jo ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM jo = DocumentTermMatrix(my documents)
DTM jo = removeSparseTerms(DTM jo , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo = weightTfIdf(DTM_jo )
jo DF <- as.matrix(tfidf jo )</pre>
DTM jo2 <- as.data.frame(as.matrix(DTM jo ), stringsAsFactors=False)</pre>
```

```
## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))
joe = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(joe ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                  # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM joe = DocumentTermMatrix(my documents)
DTM joe = removeSparseTerms(DTM joe , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf joe = weightTfIdf(DTM joe )
joe_DF <- as.matrix(tfidf_joe )</pre>
DTM joe2 <- as.data.frame(as.matrix(DTM joe ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))
john = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., tail, n=2) } %>%
    { lapply(., paste0, collapse = '') } %>%
    unlist
names(john ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
    tm_map(content_transformer(tolower)) %>%
                                                                                                                                # make everything
Lowercase
    tm map(content transformer(removeNumbers)) %>%
                                                                                                                               # remove numbers
    tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
    tm map(content transformer(stripWhitespace))
                                                                                                                  # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c (\verb"cusersmachuonedrivedocuments githubstadatareuters cctrains imoncowell newsmltx") and the standard contract of the 
t"))
```

```
## create a doc-term-matrix from the corpus
DTM john = DocumentTermMatrix(my documents)
DTM john = removeSparseTerms(DTM john , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_john = weightTfIdf(DTM_john )
john_DF <- as.matrix(tfidf_john )</pre>
DTM john2 <- as.data.frame(as.matrix(DTM john ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))
jonathan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jonathan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jonathan = DocumentTermMatrix(my documents)
DTM jonathan = removeSparseTerms(DTM jonathan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jonathan = weightTfIdf(DTM_jonathan )
jonathan DF <- as.matrix(tfidf jonathan )</pre>
DTM jonathan2 <- as.data.frame(as.matrix(DTM jonathan ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))
karl = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(karl ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(karl))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
```

```
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                        # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM karl = DocumentTermMatrix(my documents)
DTM_karl = removeSparseTerms(DTM karl , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf karl = weightTfIdf(DTM karl )
karl_DF <- as.matrix(tfidf_karl )</pre>
DTM karl2 <- as.data.frame(as.matrix(DTM karl ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))
keith = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(keith ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c (\verb"cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx") \\
t"))
## create a doc-term-matrix from the corpus
DTM_keith = DocumentTermMatrix(my_documents)
DTM_keith = removeSparseTerms(DTM_keith , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith = weightTfIdf(DTM_keith )
keith_DF <- as.matrix(tfidf_keith )</pre>
DTM keith2 <- as.data.frame(as.matrix(DTM keith ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))
kevind = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
{ lapply(., tail, n=2) } %>%
```

```
{ lapply(., paste0, collapse = '') } %>%
 unlist
names(kevind ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kevind))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kevind = DocumentTermMatrix(my documents)
## ...find words with greater than a min count...
## ...or fiDTM_kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM_kevind = removeSparseTerms(DTM_kevind , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kevind = weightTfIdf(DTM kevind )
kevind DF <- as.matrix(tfidf kevind )</pre>
DTM kevind2 <- as.data.frame(as.matrix(DTM kevind ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
```

```
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))
kevinm = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kevinm ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kevinm))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kevinm = DocumentTermMatrix(my documents)
DTM kevinm = removeSparseTerms(DTM kevinm , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kevinm = weightTfIdf(DTM kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )</pre>
DTM kevinm2 <- as.data.frame(as.matrix(DTM kevinm ), stringsAsFactors=False)</pre>
```

```
####
# Compare /cluster documents
####
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))
kirstin = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kirstin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kirstin = DocumentTermMatrix(my documents)
```

```
DTM_kirstin = removeSparseTerms(DTM_kirstin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kirstin = weightTfIdf(DTM kirstin )
kirstin_DF <- as.matrix(tfidf_kirstin )</pre>
DTM kirstin2 <- as.data.frame(as.matrix(DTM kirstin ),</pre>
stringsAsFactors=False)
library(tm)
library(tidyverse)
library(slam)
library(proxy)
library(tidytext)
library(dplyr)
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths
list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,</pre>
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]</pre>
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {</pre>
  paste(..., sep=sep, collapse=sep)
```

```
for (i in list folders){
  folder = i
  assign(i,i)
}
file list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))
simon = lapply(file list, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(simon) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM simon = DocumentTermMatrix(my_documents)
```

```
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM simon = removeSparseTerms(DTM simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf simon = weightTfIdf(DTM simon)
simon DF <- as.matrix(tfidf simon)</pre>
DTM simon2 <- as.data.frame(as.matrix(DTM simon), stringsAsFactors=False)</pre>
#Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file listA = Sys.glob(filename)
aaron = lapply(file listA, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file listA %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(aaron) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(aaron))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
tm map(content transformer(tolower)) %>%
                                                      # make everything
```

```
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_aaron = DocumentTermMatrix(my_documents)
## You can inspect its entries...
DTM aaron = removeSparseTerms(DTM aaron, 0.95)
tfidf aaron = weightTfIdf(DTM aaron)
aaron DF <- as.matrix(tfidf aaron)</pre>
DTM_aaron2 <- as.data.frame(as.matrix(DTM_aaron), stringsAsFactors=False)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlexanderSmith,'/*.txt'))
alex = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
{ strsplit(., '/', fixed=TRUE) } %>%
```

```
{ lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(alex) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alex))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                    # remove numbers
 tm_map(content_transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_alex = DocumentTermMatrix(my_documents)
DTM alex = removeSparseTerms(DTM alex, 0.95)
tfidf_alex = weightTfIdf(DTM_alex)
alex DF <- as.matrix(tfidf alex)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
```

```
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlanCrosby,'/*.txt'))
alan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(alan) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM alan = DocumentTermMatrix(my documents)
```

```
DTM alan = removeSparseTerms(DTM alan, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf alan = weightTfIdf(DTM alan)
alan DF <- as.matrix(tfidf alan)</pre>
DTM_alan2 <- as.data.frame(as.matrix(DTM_alan), stringsAsFactors=False)</pre>
# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function
# Dimensionality reduction
####
# Now PCA on term frequencies
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
    tm map(content transformer(tolower)) %>%
                                                                                                                             # make everything
Lowercase
    tm_map(content_transformer(removeNumbers)) %>% # remove numbers
    tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
    tm_map(content_transformer(stripWhitespace))
                                                                                                                           # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords.</pre>
c("cusers machuoned rivedo cuments githubstada tareuters cctrains imoncowell news mltx") and the second contract of the second contract
t"))
## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)
DTM_ben = removeSparseTerms(DTM_ben, 0.95)
tfidf ben = weightTfIdf(DTM ben)
ben DF <- as.matrix(tfidf ben)</pre>
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)</pre>
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file_listAl, readerPlain)
# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
  tm map(content transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)
DTM_ben = removeSparseTerms(DTM_ben, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf ben = weightTfIdf(DTM ben)
ben_DF <- as.matrix(tfidf_ben)</pre>
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)</pre>
##Next Author
```

```
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))
brad = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(brad) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsaithubstadatareuterscctrainsimoncowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM brad = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad = removeSparseTerms(DTM_brad, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf brad = weightTfIdf(DTM brad)
brad DF <- as.matrix(tfidf brad)</pre>
DTM_brad2 <- as.data.frame(as.matrix(DTM_brad), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))
darren = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(darren) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm map(content_transformer(removePunctuation)) %>% # remove punctuation
tm map(content transformer(stripWhitespace)) # remove excess
```

```
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsaithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM darren = DocumentTermMatrix(my documents)
DTM darren = removeSparseTerms(DTM darren, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf darren = weightTfIdf(DTM darren)
darren DF <- as.matrix(tfidf darren)</pre>
DTM darren2 <- as.data.frame(as.matrix(DTM darren), stringsAsFactors=False)</pre>
##Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))
david = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents raw = Corpus(VectorSource(david))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                       # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM david = DocumentTermMatrix(my documents)
DTM david = removeSparseTerms(DTM david, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david = weightTfIdf(DTM_david)
david DF <- as.matrix(tfidf david)</pre>
DTM david2 <- as.data.frame(as.matrix(DTM david), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))
edna = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
{ lapply(., paste0, collapse = '') } %>%
```

```
unlist
names(edna ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(edna ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
tm_map(content_transformer(stripWhitespace))  # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_edna = DocumentTermMatrix(my_documents)
DTM_edna = removeSparseTerms(DTM_edna , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_edna = weightTfIdf(DTM_edna )
edna DF <- as.matrix(tfidf edna )</pre>
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)</pre>
###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))
```

```
eric = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(eric ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(eric ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM eric = DocumentTermMatrix(my documents)
DTM eric = removeSparseTerms(DTM eric , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric = weightTfIdf(DTM_eric )
eric DF <- as.matrix(tfidf eric )</pre>
```

```
DTM eric2 <- as.data.frame(as.matrix(DTM eric ), stringsAsFactors=False)</pre>
###Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))
fumiko = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(fumiko ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                    # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM fumiko = DocumentTermMatrix(my documents)
```

```
DTM_fumiko = removeSparseTerms(DTM_fumiko , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf fumiko = weightTfIdf(DTM fumiko )
fumiko_DF <- as.matrix(tfidf_fumiko )</pre>
DTM fumiko2 <- as.data.frame(as.matrix(DTM fumiko ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))
graham = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(graham ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(graham ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM graham = DocumentTermMatrix(my documents)
DTM graham = removeSparseTerms(DTM graham , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham = weightTfIdf(DTM_graham )
graham DF <- as.matrix(tfidf graham )</pre>
DTM graham2 <- as.data.frame(as.matrix(DTM graham ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))
heather = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(heather ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
```

```
tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_heather = DocumentTermMatrix(my documents)
DTM heather = removeSparseTerms(DTM heather , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf heather = weightTfIdf(DTM heather )
heather DF <- as.matrix(tfidf heather )</pre>
DTM heather2 <- as.data.frame(as.matrix(DTM heather ),</pre>
stringsAsFactors=False)
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))
jan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
{ strsplit(., '/', fixed=TRUE) } %>%
```

```
{ lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jan ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_jan = DocumentTermMatrix(my_documents)
DTM_jan = removeSparseTerms(DTM_jan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan = weightTfIdf(DTM_jan )
jan_DF <- as.matrix(tfidf_jan )</pre>
DTM jan2 <- as.data.frame(as.matrix(DTM jan ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', JaneMacartney, '/*.txt'))
```

```
jane = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jane ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jane ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords.</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jane = DocumentTermMatrix(my documents)
DTM jane = removeSparseTerms(DTM jane , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jane = weightTfIdf(DTM jane )
jane DF <- as.matrix(tfidf jane )</pre>
DTM_jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)</pre>
###Next Author
#file list + list folders[i] =
```

```
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))
jim = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jim ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jim ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                          # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove numbers
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jim = DocumentTermMatrix(my documents)
DTM jim = removeSparseTerms(DTM jim , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jim = weightTfIdf(DTM jim )
jim DF <- as.matrix(tfidf jim )</pre>
DTM jim2 <- as.data.frame(as.matrix(DTM jim ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', JoWinterbottom, '/*.txt'))
jo = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jo ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM jo = DocumentTermMatrix(my documents)
```

```
DTM jo = removeSparseTerms(DTM jo , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo = weightTfIdf(DTM_jo )
jo DF <- as.matrix(tfidf jo )</pre>
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))
joe = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(joe ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(joe ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
```

```
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM joe = DocumentTermMatrix(my_documents)
DTM_joe = removeSparseTerms(DTM_joe , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf joe = weightTfIdf(DTM joe )
joe_DF <- as.matrix(tfidf_joe )</pre>
DTM joe2 <- as.data.frame(as.matrix(DTM joe ), stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))
john = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(john ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
```

```
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM john = DocumentTermMatrix(my documents)
DTM john = removeSparseTerms(DTM john , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf john = weightTfIdf(DTM john )
john_DF <- as.matrix(tfidf_john )</pre>
DTM john2 <- as.data.frame(as.matrix(DTM john ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))
jonathan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jonathan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
```

```
tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                        # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_jonathan = DocumentTermMatrix(my_documents)
DTM jonathan = removeSparseTerms(DTM jonathan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jonathan = weightTfIdf(DTM jonathan )
jonathan_DF <- as.matrix(tfidf_jonathan )</pre>
DTM jonathan2 <- as.data.frame(as.matrix(DTM jonathan ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))
karl = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(karl ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(karl))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_karl = DocumentTermMatrix(my_documents)
DTM_karl = removeSparseTerms(DTM_karl , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl = weightTfIdf(DTM_karl )
karl_DF <- as.matrix(tfidf_karl )</pre>
DTM karl2 <- as.data.frame(as.matrix(DTM karl ), stringsAsFactors=False)</pre>
## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))
keith = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(keith ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM keith = DocumentTermMatrix(my documents)
DTM keith = removeSparseTerms(DTM keith , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf keith = weightTfIdf(DTM keith )
keith_DF <- as.matrix(tfidf_keith )</pre>
DTM_keith2 <- as.data.frame(as.matrix(DTM_keith ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))
kevind = lapply(file_listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(kevind ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kevind))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsaithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kevind = DocumentTermMatrix(my documents)
## ...find words with greater than a min count...
## ...or fiDTM kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM kevind = removeSparseTerms(DTM kevind , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind = weightTfIdf(DTM_kevind )
kevind DF <- as.matrix(tfidf kevind )</pre>
DTM kevind2 <- as.data.frame(as.matrix(DTM kevind ), stringsAsFactors=False)</pre>
```

```
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))
kevinm = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kevinm ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_kevinm = DocumentTermMatrix(my_documents)
```

```
DTM kevinm = removeSparseTerms(DTM kevinm , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kevinm = weightTfIdf(DTM kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )</pre>
DTM_kevinm2 <- as.data.frame(as.matrix(DTM_kevinm ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))
kirstin = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kirstin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kirstin = DocumentTermMatrix(my documents)
DTM kirstin = removeSparseTerms(DTM kirstin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin = weightTfIdf(DTM_kirstin )
kirstin_DF <- as.matrix(tfidf_kirstin )</pre>
DTM_kirstin2 <- as.data.frame(as.matrix(DTM_kirstin ),</pre>
stringsAsFactors=False)
library(tm)
library(tidyverse)
library(slam)
library(proxy)
library(tidytext)
library(dplyr)
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths
list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,</pre>
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]</pre>
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
```

```
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {</pre>
  paste(..., sep=sep, collapse=sep)
for (i in list folders){
 folder = i
  assign(i,i)
file list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))
simon = lapply(file list, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(simon) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                    # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
```

```
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM simon = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM simon = removeSparseTerms(DTM simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf simon = weightTfIdf(DTM simon)
simon_DF <- as.matrix(tfidf_simon)</pre>
DTM simon2 <- as.data.frame(as.matrix(DTM simon), stringsAsFactors=False)</pre>
#Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file listA = Sys.glob(filename)
aaron = lapply(file listA, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file listA %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(aaron) = mynames
## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(aaron))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                           # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM aaron = DocumentTermMatrix(my documents)
## You can inspect its entries...
DTM aaron = removeSparseTerms(DTM aaron, 0.95)
tfidf_aaron = weightTfIdf(DTM_aaron)
aaron DF <- as.matrix(tfidf aaron)</pre>
DTM aaron2 <- as.data.frame(as.matrix(DTM aaron), stringsAsFactors=False)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', AlexanderSmith, '/*.txt'))
alex = lapply(file_listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(alex) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alex))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM alex = DocumentTermMatrix(my documents)
DTM_alex = removeSparseTerms(DTM_alex, 0.95)
tfidf alex = weightTfIdf(DTM alex)
alex DF <- as.matrix(tfidf alex)</pre>
```

```
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', AlanCrosby, '/*.txt'))
alan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(alan) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                    # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM alan = DocumentTermMatrix(my documents)
DTM alan = removeSparseTerms(DTM alan, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf alan = weightTfIdf(DTM alan)
alan DF <- as.matrix(tfidf alan)</pre>
DTM_alan2 <- as.data.frame(as.matrix(DTM_alan), stringsAsFactors=False)</pre>
# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function
####
# Dimensionality reduction
# Now PCA on term frequencies
###Next Author
list_folders <-</pre>
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                       # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)
DTM ben = removeSparseTerms(DTM ben, 0.95)
tfidf ben = weightTfIdf(DTM ben)
ben DF <- as.matrix(tfidf ben)</pre>
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)</pre>
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
```

```
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)
DTM_ben = removeSparseTerms(DTM_ben, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
```

```
tfidf ben = weightTfIdf(DTM ben)
ben_DF <- as.matrix(tfidf_ben)</pre>
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)</pre>
##Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))
brad = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(brad) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_brad = DocumentTermMatrix(my_documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM brad = removeSparseTerms(DTM brad, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf brad = weightTfIdf(DTM brad)
brad DF <- as.matrix(tfidf brad)</pre>
DTM brad2 <- as.data.frame(as.matrix(DTM brad), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))
darren = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(darren) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))
## Some pre-processing/tokenization steps.
```

```
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM darren = DocumentTermMatrix(my documents)
DTM darren = removeSparseTerms(DTM darren, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf darren = weightTfIdf(DTM darren)
darren_DF <- as.matrix(tfidf_darren)</pre>
DTM_darren2 <- as.data.frame(as.matrix(DTM_darren), stringsAsFactors=False)</pre>
##Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))
david = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
{ strsplit(., '/', fixed=TRUE) } %>%
```

```
{ lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(david) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_david = DocumentTermMatrix(my_documents)
DTM david = removeSparseTerms(DTM david, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf david = weightTfIdf(DTM david)
david_DF <- as.matrix(tfidf_david)</pre>
DTM david2 <- as.data.frame(as.matrix(DTM david), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))
edna = lapply(file listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(edna ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(edna ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM edna = DocumentTermMatrix(my documents)
DTM edna = removeSparseTerms(DTM edna , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf edna = weightTfIdf(DTM edna )
edna DF <- as.matrix(tfidf edna )</pre>
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)</pre>
###Next Author
```

```
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', EricAuchard, '/*.txt'))
eric = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(eric ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(eric ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM eric = DocumentTermMatrix(my documents)
```

```
DTM_eric = removeSparseTerms(DTM_eric , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf eric = weightTfIdf(DTM eric )
eric_DF <- as.matrix(tfidf_eric )</pre>
DTM eric2 <- as.data.frame(as.matrix(DTM eric ), stringsAsFactors=False)</pre>
###Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))
fumiko = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(fumiko ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                     # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
```

```
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusers machuoned rivedo cuments githubstada tareuters cctrains imoncowell newsmltx") and the second contract of the second contract 
t"))
## create a doc-term-matrix from the corpus
DTM fumiko = DocumentTermMatrix(my documents)
DTM fumiko = removeSparseTerms(DTM fumiko , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf fumiko = weightTfIdf(DTM fumiko )
fumiko DF <- as.matrix(tfidf fumiko )</pre>
DTM_fumiko2 <- as.data.frame(as.matrix(DTM_fumiko ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))
graham = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., tail, n=2) } %>%
    { lapply(., paste0, collapse = '') } %>%
    unlist
names(graham ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(graham ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
    tm_map(content_transformer(tolower)) %>%
                                                                                                                                  # make everything
Lowercase
```

```
tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_graham = DocumentTermMatrix(my_documents)
DTM graham = removeSparseTerms(DTM graham , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf graham = weightTfIdf(DTM graham )
graham_DF <- as.matrix(tfidf_graham )</pre>
DTM graham2 <- as.data.frame(as.matrix(DTM graham ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))
heather = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(heather ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM heather = DocumentTermMatrix(my documents)
DTM_heather = removeSparseTerms(DTM_heather , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf heather = weightTfIdf(DTM heather )
heather DF <- as.matrix(tfidf heather )</pre>
DTM heather2 <- as.data.frame(as.matrix(DTM heather ),</pre>
stringsAsFactors=False)
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))
jan = lapply(file listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_jan = DocumentTermMatrix(my_documents)
DTM_jan = removeSparseTerms(DTM_jan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jan = weightTfIdf(DTM jan )
jan DF <- as.matrix(tfidf jan )</pre>
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
```

```
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))
jane = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jane ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jane ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                      # make everything
Lowercase
 tm map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jane = DocumentTermMatrix(my documents)
DTM_jane = removeSparseTerms(DTM_jane , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf jane = weightTfIdf(DTM jane )
jane_DF <- as.matrix(tfidf_jane )</pre>
DTM jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)</pre>
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))
jim = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jim ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jim ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
```

```
t"))
## create a doc-term-matrix from the corpus
DTM jim = DocumentTermMatrix(my_documents)
DTM_jim = removeSparseTerms(DTM_jim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jim = weightTfIdf(DTM jim )
jim_DF <- as.matrix(tfidf_jim )</pre>
DTM jim2 <- as.data.frame(as.matrix(DTM jim ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))
jo = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jo ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
```

```
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM jo = DocumentTermMatrix(my documents)
DTM jo = removeSparseTerms(DTM jo , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo = weightTfIdf(DTM_jo )
jo_DF <- as.matrix(tfidf_jo )</pre>
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))
joe = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
names(joe ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
```

```
tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM joe = DocumentTermMatrix(my documents)
DTM_joe = removeSparseTerms(DTM_joe , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf joe = weightTfIdf(DTM_joe )
joe_DF <- as.matrix(tfidf_joe )</pre>
DTM_joe2 <- as.data.frame(as.matrix(DTM_joe ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', JohnMastrini, '/*.txt'))
john = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(john ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
tm map(content transformer(tolower)) %>%
                                                       # make everything
```

```
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                   # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_john = DocumentTermMatrix(my_documents)
DTM john = removeSparseTerms(DTM john , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf john = weightTfIdf(DTM john )
john DF <- as.matrix(tfidf john )</pre>
DTM_john2 <- as.data.frame(as.matrix(DTM_john ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))
jonathan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jonathan ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jonathan = DocumentTermMatrix(my documents)
DTM jonathan = removeSparseTerms(DTM jonathan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jonathan = weightTfIdf(DTM jonathan )
jonathan_DF <- as.matrix(tfidf_jonathan )</pre>
DTM_jonathan2 <- as.data.frame(as.matrix(DTM_jonathan ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))
karl = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(karl ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(karl))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
                                                  # remove excess
  tm map(content transformer(stripWhitespace))
white-space
# Let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM karl = DocumentTermMatrix(my documents)
DTM karl = removeSparseTerms(DTM karl , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf karl = weightTfIdf(DTM karl )
karl DF <- as.matrix(tfidf karl )</pre>
DTM_karl2 <- as.data.frame(as.matrix(DTM_karl ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))
```

```
keith = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(keith ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(keith))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords.</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM keith = DocumentTermMatrix(my documents)
DTM keith = removeSparseTerms(DTM keith , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf keith = weightTfIdf(DTM keith )
keith DF <- as.matrix(tfidf keith )</pre>
DTM_keith2 <- as.data.frame(as.matrix(DTM_keith ), stringsAsFactors=False)</pre>
### Next Author
```

```
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))
kevind = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kevind ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kevind))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                       # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kevind = DocumentTermMatrix(my documents)
## ...find words with greater than a min count...
## ...or fiDTM kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM kevind = removeSparseTerms(DTM kevind , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kevind = weightTfIdf(DTM kevind )
kevind DF <- as.matrix(tfidf kevind )</pre>
DTM kevind2 <- as.data.frame(as.matrix(DTM kevind ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))
kevinm = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(kevinm ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kevinm))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                     # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
```

```
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kevinm = DocumentTermMatrix(my documents)
DTM kevinm = removeSparseTerms(DTM kevinm , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm = weightTfIdf(DTM_kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )</pre>
DTM_kevinm2 <- as.data.frame(as.matrix(DTM_kevinm ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
### Next Author
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))
kirstin = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kirstin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%
                                                     # remove numbers
tm map(content_transformer(removePunctuation)) %>% # remove punctuation
```

```
tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kirstin = DocumentTermMatrix(my documents)
DTM kirstin = removeSparseTerms(DTM kirstin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kirstin = weightTfIdf(DTM kirstin )
kirstin DF <- as.matrix(tfidf kirstin )</pre>
DTM_kirstin2 <- as.data.frame(as.matrix(DTM_kirstin ),</pre>
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', KouroshKarimkhany, '/*.txt'))
kourosh = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(kourosh ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kourosh))
## Some pre-processing/tokenization steps.
```

```
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kourosh = DocumentTermMatrix(my documents)
DTM kourosh = removeSparseTerms(DTM kourosh , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kourosh = weightTfIdf(DTM_kourosh )
kourosh DF <- as.matrix(tfidf kourosh )</pre>
DTM_kourosh2 <- as.data.frame(as.matrix(DTM_kourosh),</pre>
stringsAsFactors=False)
####
# Compare /cluster documents
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LydiaZajc,'/*.txt'))
lydia = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
{ lapply(., tail, n=2) } %>%
```

```
{ lapply(., paste0, collapse = '') } %>%
 unlist
names(lydia ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(lydia))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lydia = DocumentTermMatrix(my documents)
DTM lydia = removeSparseTerms(DTM lydia , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf lydia = weightTfIdf(DTM lydia )
lydia DF <- as.matrix(tfidf lydia )</pre>
DTM_lydia2 <- as.data.frame(as.matrix(DTM_lydia ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
```

```
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',`LynneO'Donnell`,'/*.txt'))
lynne = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(lynne ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynne))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lynne = DocumentTermMatrix(my documents)
DTM lynne = removeSparseTerms(DTM lynne , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf lynne = weightTfIdf(DTM lynne )
lynne_DF <- as.matrix(tfidf_lynne )</pre>
DTM_lynne2 <- as.data.frame(as.matrix(DTM_lynne ), stringsAsFactors=False)</pre>
####
```

```
# Compare /cluster documents
####
#Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LynnleyBrowning,'/*.txt'))
lynnley = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(lynnley ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynnley))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                      # remove numbers
 tm_map(content_transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_lynnley = DocumentTermMatrix(my_documents)
DTM_lynnley = removeSparseTerms(DTM_lynnley , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf lynnley = weightTfIdf(DTM lynnley )
lynnley DF <- as.matrix(tfidf lynnley )</pre>
DTM lynnley2 <- as.data.frame(as.matrix(DTM lynnley ),</pre>
stringsAsFactors=False)
####
# Compare /cluster documents
####
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', MarcelMichelson, '/*.txt'))
marcel = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(marcel ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(marcel))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM marcel = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_marcel = removeSparseTerms(DTM_marcel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_marcel = weightTfIdf(DTM_marcel)
marcel DF <- as.matrix(tfidf marcel )</pre>
DTM_marcel2 <- as.data.frame(as.matrix(DTM_marcel ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', MarkBendeich, '/*.txt'))
mark = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(mark ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mark))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_mark = DocumentTermMatrix(my_documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM mark = removeSparseTerms(DTM mark , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mark = weightTfIdf(DTM_mark )
mark_DF <- as.matrix(tfidf mark )</pre>
DTM mark2 <- as.data.frame(as.matrix(DTM mark ), stringsAsFactors=False)</pre>
### Next Question
#file list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', MartinWolk, '/*.txt'))
martin = lapply(file listAl, readerPlain)
# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(martin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(martin))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM martin = DocumentTermMatrix(my documents)
DTM martin = removeSparseTerms(DTM martin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_martin = weightTfIdf(DTM_martin )
martin DF <- as.matrix(tfidf martin )</pre>
DTM martin2 <- as.data.frame(as.matrix(DTM martin ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
```

```
50train/',MichaelConnor,'/*.txt'))
michael = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(michael ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(michael))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_michael = DocumentTermMatrix(my_documents)
DTM michael = removeSparseTerms(DTM michael , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_michael = weightTfIdf(DTM_michael )
michael DF <- as.matrix(tfidf michael )</pre>
DTM michael2 <- as.data.frame(as.matrix(DTM michael ),</pre>
stringsAsFactors=False)
### Next Author
```

```
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MureDickie,'/*.txt'))
mure = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(mure ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mure))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                     # remove numbers
 tm_map(content_transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_mure = DocumentTermMatrix(my_documents)
DTM_mure = removeSparseTerms(DTM_mure , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mure = weightTfIdf(DTM_mure )
mure DF <- as.matrix(tfidf mure )</pre>
DTM_mure2 <- as.data.frame(as.matrix(DTM_mure ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',NickLouth,'/*.txt'))
nick = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(nick ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(nick))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                     # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM nick = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_nick = removeSparseTerms(DTM_nick , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf nick = weightTfIdf(DTM nick )
nick DF <- as.matrix(tfidf nick )</pre>
DTM_nick2 <- as.data.frame(as.matrix(DTM_nick ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PatriciaCommins,'/*.txt'))
patricia = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(patricia ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(patricia))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
tm_map(content_transformer(removeNumbers)) %>% # remove numbers
```

```
tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM patricia = DocumentTermMatrix(my documents)
DTM_patricia = removeSparseTerms(DTM_patricia , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf patricia = weightTfIdf(DTM patricia )
patricia DF <- as.matrix(tfidf patricia )</pre>
DTM_patricia2 <- as.data.frame(as.matrix(DTM_patricia ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PeterHumphrey,'/*.txt'))
peter = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(peter ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(peter))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_peter = DocumentTermMatrix(my_documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM peter = removeSparseTerms(DTM peter , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf peter = weightTfIdf(DTM peter )
peter DF <- as.matrix(tfidf peter )</pre>
DTM_peter2 <- as.data.frame(as.matrix(DTM_peter ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PierreTran,'/*.txt'))
pierre = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(pierre ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(pierre))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_pierre = DocumentTermMatrix(my_documents)
DTM_pierre = removeSparseTerms(DTM_pierre , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf pierre = weightTfIdf(DTM pierre )
pierre_DF <- as.matrix(tfidf_pierre )</pre>
DTM_pierre2 <- as.data.frame(as.matrix(DTM_pierre ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RobinSidel,'/*.txt'))
```

```
robin = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(robin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(robin))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM robin = DocumentTermMatrix(my documents)
DTM robin = removeSparseTerms(DTM robin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_robin = weightTfIdf(DTM_robin )
robin DF <- as.matrix(tfidf robin )</pre>
DTM robin2 <- as.data.frame(as.matrix(DTM robin ), stringsAsFactors=False)</pre>
### Next Author
```

```
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RogerFillion,'/*.txt'))
roger = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(roger ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(roger))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 # make everything
Lowercase
                                                     # remove numbers
 tm_map(content_transformer(removeNumbers)) %>%
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm map(content_transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_roger = DocumentTermMatrix(my_documents)
DTM roger = removeSparseTerms(DTM roger , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf roger = weightTfIdf(DTM roger )
roger DF <- as.matrix(tfidf roger )</pre>
DTM_roger2 <- as.data.frame(as.matrix(DTM_roger ), stringsAsFactors=False)</pre>
```

```
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SarahDavison,'/*.txt'))
sarah = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(sarah ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(sarah))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM sarah = DocumentTermMatrix(my documents)
```

```
DTM_sarah = removeSparseTerms(DTM_sarah , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf sarah = weightTfIdf(DTM sarah )
sarah_DF <- as.matrix(tfidf_sarah )</pre>
DTM sarah2 <- as.data.frame(as.matrix(DTM sarah ), stringsAsFactors=False)</pre>
### Next Question
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ScottHillis,'/*.txt'))
scott = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(scott ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(scott))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
```

```
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
DTM scott = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_scott = removeSparseTerms(DTM_scott , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf scott = weightTfIdf(DTM scott )
scott_DF <- as.matrix(tfidf_scott )</pre>
DTM_scott2 <- as.data.frame(as.matrix(DTM_scott ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TheresePoletti,'/*.txt'))
therese = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(therese ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(therese))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
```

```
tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM therese = DocumentTermMatrix(my documents)
DTM therese = removeSparseTerms(DTM therese , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf therese = weightTfIdf(DTM therese )
therese DF <- as.matrix(tfidf therese )</pre>
DTM therese2 <- as.data.frame(as.matrix(DTM therese ),
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TimFarrand,'/*.txt'))
tim = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(tim ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents raw = Corpus(VectorSource(tim))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM tim = DocumentTermMatrix(my documents)
DTM tim = removeSparseTerms(DTM tim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf tim = weightTfIdf(DTM tim )
tim DF <- as.matrix(tfidf tim )</pre>
DTM tim2 <- as.data.frame(as.matrix(DTM tim ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',WilliamKazer,'/*.txt'))
will = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
{ lapply(., tail, n=2) } %>%
```

```
{ lapply(., paste0, collapse = '') } %>%
  unlist
names(will ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(will))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM will = DocumentTermMatrix(my documents)
DTM_will = removeSparseTerms(DTM_will , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf will = weightTfIdf(DTM will )
will DF <- as.matrix(tfidf will )</pre>
DTM will2 <- as.data.frame(as.matrix(DTM will ), stringsAsFactors=False)</pre>
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BernardHickey,'/*.txt'))
bernard = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(bernard ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(bernard))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
                                                  # remove excess
  tm map(content transformer(stripWhitespace))
white-space
# Let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM bernard = DocumentTermMatrix(my documents)
## ...find words with greater than a min count...
DTM_bernard = removeSparseTerms(DTM_bernard , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_bernard = weightTfIdf(DTM_bernard )
bernard_DF <- as.matrix(tfidf_bernard )</pre>
DTM bernard2 <- as.data.frame(as.matrix(DTM bernard ),</pre>
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MatthewBunce,'/*.txt'))
```

```
matthew = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(matthew ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(matthew))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM matthew = DocumentTermMatrix(my documents)
DTM matthew = removeSparseTerms(DTM matthew , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf matthew = weightTfIdf(DTM matthew )
bernard_DF <- as.matrix(tfidf_matthew )</pre>
DTM matthew2 <- as.data.frame(as.matrix(DTM matthew ),</pre>
stringsAsFactors=False)
```

```
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SamuelPerry,'/*.txt'))
samuel = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., tail, n=2) } %>%
    { lapply(., paste0, collapse = '') } %>%
    unlist
names(samuel ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(samuel))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
    tm_map(content_transformer(tolower)) %>%
                                                                                                                        # make everything
Lowercase
                                                                                                                     # remove numbers
    tm map(content transformer(removeNumbers)) %>%
    tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
    tm map(content transformer(stripWhitespace))
                                                                                                               # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusers machuoned rivedo cuments githubstada tareuters cctrains imoncowell news mltx") and the second contract of the second contract
t"))
## create a doc-term-matrix from the corpus
DTM samuel = DocumentTermMatrix(my documents)
DTM samuel = removeSparseTerms(DTM samuel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_samuel = weightTfIdf(DTM_samuel )
bernard_DF <- as.matrix(tfidf_samuel )</pre>
DTM samuel2 <- as.data.frame(as.matrix(DTM samuel ), stringsAsFactors=False)</pre>
```

```
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', TanEeLyn, '/*.txt'))
tan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(tan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(tan))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                     # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                   # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM tan = DocumentTermMatrix(my documents)
DTM_tan = removeSparseTerms(DTM_tan , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tan = weightTfIdf(DTM_tan )
bernard DF <- as.matrix(tfidf tan )</pre>
DTM tan2 <- as.data.frame(as.matrix(DTM tan ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ToddNissen,'/*.txt'))
todd = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(todd ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(todd))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
```

```
DTM todd = DocumentTermMatrix(my documents)
DTM todd = removeSparseTerms(DTM todd , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_todd = weightTfIdf(DTM_todd )
bernard DF <- as.matrix(tfidf todd )</pre>
DTM_todd2 <- as.data.frame(as.matrix(DTM_todd ), stringsAsFactors=False)</pre>
library(e1071)
### Creating Tests
#Author1
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0test/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list folders[i]+'/*.txt'))
p <- function(..., sep='') {</pre>
  paste(..., sep=sep, collapse=sep)
for (i in list_folders){
  folder = i
  assign(i,i)
}
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SimonCowell,'/*.txt'))
simon test = lapply(file list test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
#mynames
names(simon test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(simon test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
#?stopwords
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM simon test = DocumentTermMatrix(my documents)
DTM simon test = removeSparseTerms(DTM simon test, 0.95)
#DTM_simon_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf simon test = weightTfIdf(DTM simon test)
simon_DF <- as.data.frame(as.matrix(tfidf_simon))</pre>
simon DF test <- as.data.frame(as.matrix(tfidf simon test))</pre>
simon_words <- names(simon_DF)</pre>
simon_words_test <- names(simon_DF_test)</pre>
remove simon <- simon words test[!(simon words test %in% simon words)]
simon_DF_test <- simon_DF_test[, !colnames(simon_DF_test) %in% remove simon]</pre>
```

```
### Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0test/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list folders[i]+'/*.txt'))
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', AaronPressman, '/*.txt'))
aaron_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mvnames
names(aaron_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(aaron test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
 tm map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
my documents = tm map(my documents, content transformer(removeWords),
```

```
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainaaroncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM aaron test = DocumentTermMatrix(my documents)
DTM_aaron_test = removeSparseTerms(DTM_aaron_test, 0.95)
#DTM aaron test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_aaron_test = weightTfIdf(DTM_aaron_test)
aaron_DF <- as.data.frame(as.matrix(tfidf aaron))</pre>
aaron DF test <- as.data.frame(as.matrix(tfidf aaron test))</pre>
aaron words <- names(aaron DF)</pre>
aaron words test <- names(aaron DF test)</pre>
remove aaron <- aaron words test[!(aaron words test %in% aaron words)]
aaron DF test <- aaron DF test[, !colnames(aaron DF test) %in% remove aaron]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', AlanCrosby, '/*.txt'))
alan_test = lapply(file_list_test, readerPlain)
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(alan_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alan test))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                     # remove numbers
  tm map(content transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalancowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM_alan_test = DocumentTermMatrix(my_documents)
DTM_alan_test = removeSparseTerms(DTM_alan_test, 0.95)
tfidf alan test = weightTfIdf(DTM alan test)
alan_DF <- as.data.frame(as.matrix(tfidf_alan))</pre>
alan DF test <- as.data.frame(as.matrix(tfidf alan test))</pre>
alan words <- names(alan DF)</pre>
alan_words_test <- names(alan_DF_test)</pre>
remove alan <- alan words test[!(alan words test %in% alan words)]
alan DF test <- alan DF test[, !colnames(alan DF test) %in% remove alan]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', AlexanderSmith, '/*.txt'))
alex_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(alex_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alex test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalexcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM_alex_test = DocumentTermMatrix(my documents)
DTM alex test = removeSparseTerms(DTM alex test, 0.95)
#DTM alex test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf alex test = weightTfIdf(DTM alex test)
alex DF <- as.data.frame(as.matrix(tfidf alex))</pre>
alex_DF_test <- as.data.frame(as.matrix(tfidf_alex_test))</pre>
alex words <- names(alex DF)
alex_words_test <- names(alex_DF_test)</pre>
remove alex <- alex words test[!(alex words test %in% alex words)]
alex_DF_test <- alex_DF_test[, !colnames(alex_DF_test) %in% remove_alex]</pre>
### Next Author
```

```
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', BenjaminKangLim, '/*.txt'))
ben test = lapply(file list test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(ben_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbencowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM ben test = DocumentTermMatrix(my documents)
DTM ben test = removeSparseTerms(DTM ben test, 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben_test = weightTfIdf(DTM_ben_test)
ben DF <- as.data.frame(as.matrix(tfidf ben))</pre>
ben_DF_test <- as.data.frame(as.matrix(tfidf_ben_test))</pre>
ben_words <- names(ben_DF)</pre>
ben words test <- names(ben DF test)</pre>
remove_ben <- ben_words_test[!(ben_words_test %in% ben_words)]</pre>
ben_DF_test <- ben_DF_test[, !colnames(ben_DF_test) %in% remove_ben]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BernardHickey,'/*.txt'))
bernard_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(bernard_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(bernard_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbernardcowellnewsml
txt"))
```

```
## create a doc-term-matrix from the corpus
DTM_bernard_test = DocumentTermMatrix(my_documents)
DTM_bernard_test = removeSparseTerms(DTM_bernard_test, 0.95)
#DTM bernard test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf bernard test = weightTfIdf(DTM bernard test)
bernard_DF <- as.data.frame(as.matrix(tfidf_bernard))</pre>
bernard_DF_test <- as.data.frame(as.matrix(tfidf_bernard_test))</pre>
bernard words <- names(bernard DF)</pre>
bernard words test <- names(bernard DF test)
remove_bernard <- bernard_words_test[!(bernard_words_test %in%</pre>
bernard words)]
bernard DF test <- bernard DF test[, !colnames(bernard DF test) %in%
remove bernard]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BradDorfman,'/*.txt'))
brad test = lapply(file list test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(brad test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(brad test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```
tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                        # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbradcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM brad test = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM brad test = removeSparseTerms(DTM brad test, 0.95)
#DTM_brad_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad_test = weightTfIdf(DTM_brad_test)
brad DF <- as.data.frame(as.matrix(tfidf brad))</pre>
brad_DF_test <- as.data.frame(as.matrix(tfidf_brad_test))</pre>
brad words <- names(brad DF)</pre>
brad words test <- names(brad DF test)</pre>
remove brad <- brad words test[!(brad words test %in% brad words)]
brad DF test <- brad DF test[, !colnames(brad DF test) %in% remove brad]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DarrenSchuettler,'/*.txt'))
darren test = lapply(file list test, readerPlain)
```

```
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(darren_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(darren test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindarrencowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM_darren_test = DocumentTermMatrix(my_documents)
DTM darren test = removeSparseTerms(DTM darren test, 0.95)
#DTM_darren_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren_test = weightTfIdf(DTM_darren_test)
darren DF <- as.data.frame(as.matrix(tfidf darren))</pre>
```

```
darren DF test <- as.data.frame(as.matrix(tfidf darren test))</pre>
darren words <- names(darren DF)</pre>
darren_words_test <- names(darren_DF_test)</pre>
remove_darren <- darren_words_test[!(darren_words_test %in% darren_words)]</pre>
darren DF test <- darren DF test[, !colnames(darren DF test) %in%</pre>
remove darren]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DavidLawder,'/*.txt'))
david_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindavidcowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM david test = DocumentTermMatrix(my documents)
DTM_david_test = removeSparseTerms(DTM_david_test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david_test = weightTfIdf(DTM_david_test)
david DF <- as.data.frame(as.matrix(tfidf david))</pre>
david DF test <- as.data.frame(as.matrix(tfidf david test))</pre>
david words <- names(david DF)</pre>
david words test <- names(david DF test)</pre>
remove david <- david words test[!(david words test %in% david words)]
david DF_test <- david DF_test[, !colnames(david DF_test) %in% remove_david]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', EdnaFernandes, '/*.txt'))
edna_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(edna_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(edna test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
```

```
tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                        # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainednacowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM edna test = DocumentTermMatrix(my documents)
DTM edna test = removeSparseTerms(DTM edna test, 0.95)
#DTM edna test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf edna test = weightTfIdf(DTM edna test)
edna DF <- as.data.frame(as.matrix(tfidf edna))</pre>
edna DF test <- as.data.frame(as.matrix(tfidf edna test))
edna_words <- names(edna_DF)</pre>
edna words test <- names(edna DF test)
remove edna <- edna words test[!(edna words test %in% edna words)]
edna_DF_test <- edna_DF_test[, !colnames(edna_DF_test) %in% remove_edna]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',EricAuchard,'/*.txt'))
eric_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mvnames
names(eric_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(eric test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainericcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM eric test = DocumentTermMatrix(my documents)
#DTM eric test # some basic summary statistics
## You can inspect its entries...
#inspect(DTM eric test[1:10,1:20])
## ...find words with greater than a min count...
#findFreqTerms(DTM_eric_test, 50)
## ...or find words whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
#findAssocs(DTM_eric_test, "genetic", .5)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM eric test = removeSparseTerms(DTM eric test, 0.95)
```

```
#DTM eric test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf eric test = weightTfIdf(DTM eric test)
eric_DF <- as.data.frame(as.matrix(tfidf_eric))</pre>
eric DF test <- as.data.frame(as.matrix(tfidf eric test))</pre>
eric words <- names(eric DF)</pre>
eric_words_test <- names(eric_DF_test)</pre>
remove eric <- eric words test[!(eric words test %in% eric words)]
eric_DF_test <- eric_DF_test[, !colnames(eric_DF_test) %in% remove_eric]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',FumikoFujisaki,'/*.txt'))
fumiko_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(fumiko test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                        # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
```

```
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainfumikocowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM fumiko test = DocumentTermMatrix(my documents)
DTM_fumiko_test = removeSparseTerms(DTM_fumiko_test, 0.95)
#DTM fumiko test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko_test = weightTfIdf(DTM_fumiko_test)
fumiko_DF <- as.data.frame(as.matrix(tfidf_fumiko))</pre>
fumiko DF test <- as.data.frame(as.matrix(tfidf fumiko test))</pre>
fumiko_words <- names(fumiko_DF)</pre>
fumiko words test <- names(fumiko DF test)</pre>
remove fumiko <- fumiko words test[!(fumiko words test %in% fumiko words)]
fumiko DF test <- fumiko DF test[, !colnames(fumiko DF test) %in%</pre>
remove fumiko]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',GrahamEarnshaw,'/*.txt'))
graham_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(graham test) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(graham test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                         # remove excess
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraingrahamcowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM graham test = DocumentTermMatrix(my documents)
DTM graham test = removeSparseTerms(DTM graham test, 0.95)
#DTM graham test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf graham test = weightTfIdf(DTM graham test)
graham_DF <- as.data.frame(as.matrix(tfidf_graham))</pre>
graham_DF_test <- as.data.frame(as.matrix(tfidf_graham_test))</pre>
graham_words <- names(graham_DF)</pre>
graham words test <- names(graham DF test)</pre>
remove graham <- graham words test[!(graham words test %in% graham words)]
graham_DF_test <- graham_DF_test[, !colnames(graham_DF_test) %in%</pre>
remove graham]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',HeatherScoffield,'/*.txt'))
heather_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(heather_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainheathercowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM_heather_test = DocumentTermMatrix(my_documents)
DTM heather test = removeSparseTerms(DTM heather test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf heather test = weightTfIdf(DTM heather test)
heather_DF <- as.data.frame(as.matrix(tfidf_heather))</pre>
heather DF test <- as.data.frame(as.matrix(tfidf heather test))</pre>
heather words <- names(heather DF)</pre>
heather_words_test <- names(heather_DF_test)</pre>
```

```
remove heather <- heather words test[!(heather words test %in%
heather words)]
heather_DF_test <- heather_DF_test[, !colnames(heather_DF_test) %in%
remove heather]
## Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JanLopatka,'/*.txt'))
jan_test = lapply(file_list_test, readerPlain)
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(jan_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jan test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjancowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM jan test = DocumentTermMatrix(my documents)
DTM jan test = removeSparseTerms(DTM jan test, 0.95)
```

```
#DTM jan test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jan test = weightTfIdf(DTM jan test)
jan_DF <- as.data.frame(as.matrix(tfidf_jan))</pre>
jan_DF_test <- as.data.frame(as.matrix(tfidf jan test))</pre>
jan words <- names(jan DF)</pre>
jan_words_test <- names(jan_DF_test)</pre>
remove jan <- jan words test[!(jan words test %in% jan words)]
jan_DF_test <- jan_DF_test[, !colnames(jan_DF_test) %in% remove_jan]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JaneMacartney,'/*.txt'))
jane test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(jane_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jane test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
tm map(content transformer(removePunctuation)) %>% # remove punctuation
```

```
tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# Let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjanecowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM jane test = DocumentTermMatrix(my documents)
DTM jane test = removeSparseTerms(DTM jane test, 0.95)
#DTM jane test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane_test = weightTfIdf(DTM_jane_test)
jane DF <- as.data.frame(as.matrix(tfidf jane))</pre>
jane_DF_test <- as.data.frame(as.matrix(tfidf_jane_test))</pre>
jane words <- names(jane DF)</pre>
jane words test <- names(jane DF test)</pre>
remove_jane <- jane_words_test[!(jane_words_test %in% jane_words)]</pre>
jane_DF_test <- jane_DF_test[, !colnames(jane_DF_test) %in% remove_jane]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JimGilchrist,'/*.txt'))
jim_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
```

```
#mynames
names(jim test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jim test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
                                                        # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjimcowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM_jim_test = DocumentTermMatrix(my_documents)
DTM_jim_test = removeSparseTerms(DTM_jim_test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim_test = weightTfIdf(DTM_jim_test)
jim_DF <- as.data.frame(as.matrix(tfidf_jim))</pre>
jim DF test <- as.data.frame(as.matrix(tfidf jim test))</pre>
jim_words <- names(jim_DF)</pre>
jim_words_test <- names(jim_DF_test)</pre>
remove_jim <- jim_words_test[!(jim_words_test %in% jim_words)]</pre>
jim_DF_test <- jim_DF_test[, !colnames(jim_DF_test) %in% remove_jim]</pre>
library(tm)
library(tidyverse)
library(slam)
library(proxy)
```

```
library(tidytext)
library(dplyr)
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths
list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,</pre>
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]</pre>
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
p <- function(..., sep='') {</pre>
  paste(..., sep=sep, collapse=sep)
for (i in list folders){
  folder = i
  assign(i,i)
file list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))
simon = lapply(file_list, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(simon) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
tm_map(content_transformer(removePunctuation)) %>% # remove numbers
# remove numbers
  tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_simon = DocumentTermMatrix(my_documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM simon = removeSparseTerms(DTM simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf simon = weightTfIdf(DTM simon)
simon_DF <- as.matrix(tfidf_simon)</pre>
DTM simon2 <- as.data.frame(as.matrix(DTM simon), stringsAsFactors=False)</pre>
```

```
#Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file listA = Sys.glob(filename)
aaron = lapply(file listA, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file_listA %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(aaron) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(aaron))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                     # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                     # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
```

```
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM aaron = DocumentTermMatrix(my documents)
## You can inspect its entries...
DTM aaron = removeSparseTerms(DTM aaron, 0.95)
tfidf_aaron = weightTfIdf(DTM_aaron)
aaron_DF <- as.matrix(tfidf_aaron)</pre>
DTM_aaron2 <- as.data.frame(as.matrix(DTM_aaron), stringsAsFactors=False)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', AlexanderSmith, '/*.txt'))
alex = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(alex) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
tm map(content transformer(tolower)) %>%
                                                        # make everything
```

```
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c (\verb"cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx") \\
t"))
## create a doc-term-matrix from the corpus
DTM alex = DocumentTermMatrix(my documents)
DTM alex = removeSparseTerms(DTM alex, 0.95)
tfidf alex = weightTfIdf(DTM alex)
alex_DF <- as.matrix(tfidf_alex)</pre>
###next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlanCrosby,'/*.txt'))
alan = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
{ strsplit(., '/', fixed=TRUE) } %>%
```

```
{ lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(alan) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(alan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_alan = DocumentTermMatrix(my_documents)
DTM alan = removeSparseTerms(DTM alan, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alan = weightTfIdf(DTM_alan)
alan DF <- as.matrix(tfidf alan)</pre>
DTM alan2 <- as.data.frame(as.matrix(DTM alan), stringsAsFactors=False)</pre>
# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function
####
```

```
# Dimensionality reduction
####
# Now PCA on term frequencies
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                    # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)
DTM ben = removeSparseTerms(DTM ben, 0.95)
tfidf ben = weightTfIdf(DTM ben)
ben DF <- as.matrix(tfidf ben)</pre>
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)</pre>
###Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))
ben = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(ben) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(ben))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM ben = DocumentTermMatrix(my documents)
DTM ben = removeSparseTerms(DTM ben, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf ben = weightTfIdf(DTM ben)
ben DF <- as.matrix(tfidf ben)</pre>
DTM ben2 <- as.data.frame(as.matrix(DTM ben), stringsAsFactors=False)</pre>
##Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))
brad = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
names(brad) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(brad))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM brad = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM brad = removeSparseTerms(DTM brad, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf brad = weightTfIdf(DTM brad)
```

```
brad DF <- as.matrix(tfidf brad)</pre>
DTM brad2 <- as.data.frame(as.matrix(DTM brad), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))
darren = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(darren) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(darren))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                           # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                     # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM darren = DocumentTermMatrix(my documents)
DTM darren = removeSparseTerms(DTM darren, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf darren = weightTfIdf(DTM darren)
darren DF <- as.matrix(tfidf darren)</pre>
DTM_darren2 <- as.data.frame(as.matrix(DTM_darren), stringsAsFactors=False)</pre>
##Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))
david = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(david) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
```

```
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM david = DocumentTermMatrix(my documents)
DTM david = removeSparseTerms(DTM david, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf david = weightTfIdf(DTM david)
david_DF <- as.matrix(tfidf_david)</pre>
DTM david2 <- as.data.frame(as.matrix(DTM david), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))
edna = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(edna ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
tm map(content transformer(removeNumbers)) %>%  # remove numbers
```

```
tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                  # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM edna = DocumentTermMatrix(my documents)
DTM edna = removeSparseTerms(DTM edna , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf edna = weightTfIdf(DTM edna )
edna_DF <- as.matrix(tfidf_edna )</pre>
DTM edna2 <- as.data.frame(as.matrix(DTM edna ), stringsAsFactors=False)</pre>
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))
eric = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(eric ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM eric = DocumentTermMatrix(my documents)
DTM eric = removeSparseTerms(DTM eric , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf eric = weightTfIdf(DTM eric )
eric_DF <- as.matrix(tfidf eric )</pre>
DTM_eric2 <- as.data.frame(as.matrix(DTM_eric ), stringsAsFactors=False)</pre>
###Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))
fumiko = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(fumiko ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(fumiko ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                  # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM fumiko = DocumentTermMatrix(my documents)
DTM fumiko = removeSparseTerms(DTM fumiko , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko = weightTfIdf(DTM_fumiko )
fumiko DF <- as.matrix(tfidf fumiko )</pre>
DTM fumiko2 <- as.data.frame(as.matrix(DTM fumiko ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
```

```
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))
graham = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(graham ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
                                                   # remove excess
  tm map(content transformer(stripWhitespace))
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM graham = DocumentTermMatrix(my documents)
DTM graham = removeSparseTerms(DTM graham , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham = weightTfIdf(DTM_graham )
graham DF <- as.matrix(tfidf graham )</pre>
DTM_graham2 <- as.data.frame(as.matrix(DTM_graham ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))
heather = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(heather ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                  # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
```

```
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM heather = DocumentTermMatrix(my documents)
DTM heather = removeSparseTerms(DTM heather , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf heather = weightTfIdf(DTM heather )
heather_DF <- as.matrix(tfidf_heather )</pre>
DTM heather2 <- as.data.frame(as.matrix(DTM heather ),</pre>
stringsAsFactors=False)
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', JanLopatka, '/*.txt'))
jan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
tm_map(content_transformer(tolower)) %>%
                                                       # make everything
```

```
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                    # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jan = DocumentTermMatrix(my documents)
DTM jan = removeSparseTerms(DTM jan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jan = weightTfIdf(DTM jan )
jan DF <- as.matrix(tfidf jan )</pre>
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))
jane = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(jane ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
    tm_map(content_transformer(tolower)) %>%
                                                                                                                          # make everything
Lowercase
    tm_map(content_transformer(removeNumbers)) %>% # remove numbers
    tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
    tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusers machuoned rivedo cuments githubstada tareuters cctrains imoncowell news mltx") and the second contract of the second contract
t"))
## create a doc-term-matrix from the corpus
DTM_jane = DocumentTermMatrix(my_documents)
DTM_jane = removeSparseTerms(DTM_jane , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane = weightTfIdf(DTM_jane )
jane_DF <- as.matrix(tfidf_jane )</pre>
DTM jane2 <- as.data.frame(as.matrix(DTM jane ), stringsAsFactors=False)</pre>
###Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))
jim = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jim ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                    # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM jim = DocumentTermMatrix(my documents)
DTM_jim = removeSparseTerms(DTM_jim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jim = weightTfIdf(DTM jim )
jim DF <- as.matrix(tfidf jim )</pre>
DTM_jim2 <- as.data.frame(as.matrix(DTM_jim ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))
jo = lapply(file listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jo ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jo ))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM jo = DocumentTermMatrix(my documents)
DTM jo = removeSparseTerms(DTM jo , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo = weightTfIdf(DTM_jo )
jo_DF <- as.matrix(tfidf jo )</pre>
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)</pre>
## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
```

```
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))
joe = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
names(joe ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                           # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
tm_map(content_transformer(removePunctuation)) %>% # remove numbers
# remove numbers
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM_joe = DocumentTermMatrix(my_documents)
DTM joe = removeSparseTerms(DTM joe , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_joe = weightTfIdf(DTM_joe )
joe_DF <- as.matrix(tfidf_joe )</pre>
DTM_joe2 <- as.data.frame(as.matrix(DTM_joe ), stringsAsFactors=False)</pre>
```

```
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))
john = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(john ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(john ))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM john = DocumentTermMatrix(my documents)
DTM_john = removeSparseTerms(DTM_john , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf john = weightTfIdf(DTM john )
john_DF <- as.matrix(tfidf_john )</pre>
DTM john2 <- as.data.frame(as.matrix(DTM john ), stringsAsFactors=False)</pre>
## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))
jonathan = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(jonathan ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM_jonathan = DocumentTermMatrix(my_documents)
DTM_jonathan = removeSparseTerms(DTM_jonathan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jonathan = weightTfIdf(DTM jonathan )
jonathan DF <- as.matrix(tfidf jonathan )</pre>
DTM_jonathan2 <- as.data.frame(as.matrix(DTM_jonathan ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))
karl = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(karl ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(karl))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_karl = DocumentTermMatrix(my_documents)
DTM karl = removeSparseTerms(DTM karl , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf karl = weightTfIdf(DTM karl )
karl_DF <- as.matrix(tfidf_karl )</pre>
DTM_karl2 <- as.data.frame(as.matrix(DTM_karl ), stringsAsFactors=False)</pre>
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))
keith = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(keith ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
```

```
tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords.</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM keith = DocumentTermMatrix(my documents)
DTM keith = removeSparseTerms(DTM keith , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf keith = weightTfIdf(DTM keith )
keith DF <- as.matrix(tfidf keith )</pre>
DTM keith2 <- as.data.frame(as.matrix(DTM keith ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))
kevind = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(kevind ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents raw = Corpus(VectorSource(kevind))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                       # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kevind = DocumentTermMatrix(my documents)
## ...find words with greater than a min count...
## ...or fiDTM kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM_kevind = removeSparseTerms(DTM_kevind , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind = weightTfIdf(DTM_kevind )
kevind_DF <- as.matrix(tfidf_kevind )</pre>
DTM kevind2 <- as.data.frame(as.matrix(DTM kevind ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))
kevinm = lapply(file_listAl, readerPlain)
# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kevinm ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kevinm))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                      # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_kevinm = DocumentTermMatrix(my_documents)
DTM kevinm = removeSparseTerms(DTM kevinm , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kevinm = weightTfIdf(DTM kevinm )
kevinm DF <- as.matrix(tfidf kevinm )</pre>
DTM kevinm2 <- as.data.frame(as.matrix(DTM kevinm ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
### Next Author
```

```
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))
kirstin = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(kirstin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kirstin = DocumentTermMatrix(my documents)
DTM kirstin = removeSparseTerms(DTM kirstin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin = weightTfIdf(DTM_kirstin )
kirstin DF <- as.matrix(tfidf kirstin )</pre>
```

```
DTM kirstin2 <- as.data.frame(as.matrix(DTM kirstin ),</pre>
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KouroshKarimkhany,'/*.txt'))
kourosh = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(kourosh ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kourosh))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM kourosh = DocumentTermMatrix(my_documents)
DTM_kourosh = removeSparseTerms(DTM_kourosh , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kourosh = weightTfIdf(DTM_kourosh )
kourosh DF <- as.matrix(tfidf kourosh )</pre>
DTM_kourosh2 <- as.data.frame(as.matrix(DTM_kourosh),</pre>
stringsAsFactors=False)
####
# Compare /cluster documents
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LydiaZajc,'/*.txt'))
lydia = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(lydia ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lydia))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# Let's just use the "basic English" stop words
```

```
my documents = tm map(my documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lydia = DocumentTermMatrix(my documents)
DTM lydia = removeSparseTerms(DTM lydia , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lydia = weightTfIdf(DTM_lydia )
lydia DF <- as.matrix(tfidf lydia )</pre>
DTM lydia2 <- as.data.frame(as.matrix(DTM lydia ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
## Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',`LynneO'Donnell`,'/*.txt'))
lynne = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(lynne ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynne))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lynne = DocumentTermMatrix(my documents)
DTM lynne = removeSparseTerms(DTM lynne , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf lynne = weightTfIdf(DTM lynne )
lynne_DF <- as.matrix(tfidf_lynne )</pre>
DTM lynne2 <- as.data.frame(as.matrix(DTM lynne ), stringsAsFactors=False)</pre>
####
# Compare /cluster documents
####
#Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LynnleyBrowning,'/*.txt'))
lynnley = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(lynnley ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(lynnley))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lynnley = DocumentTermMatrix(my documents)
DTM lynnley = removeSparseTerms(DTM lynnley , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf lynnley = weightTfIdf(DTM lynnley )
lynnley_DF <- as.matrix(tfidf_lynnley )</pre>
DTM lynnley2 <- as.data.frame(as.matrix(DTM lynnley ),</pre>
stringsAsFactors=False)
####
# Compare /cluster documents
####
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', MarcelMichelson, '/*.txt'))
```

```
marcel = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(marcel ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(marcel))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM marcel = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_marcel = removeSparseTerms(DTM_marcel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf marcel = weightTfIdf(DTM marcel )
marcel DF <- as.matrix(tfidf marcel )</pre>
DTM marcel2 <- as.data.frame(as.matrix(DTM marcel ), stringsAsFactors=False)</pre>
```

```
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', MarkBendeich, '/*.txt'))
mark = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(mark ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(mark))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
```

```
DTM mark = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_mark = removeSparseTerms(DTM_mark , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf mark = weightTfIdf(DTM mark )
mark DF <- as.matrix(tfidf mark )</pre>
DTM mark2 <- as.data.frame(as.matrix(DTM mark ), stringsAsFactors=False)</pre>
### Next Ouestion
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MartinWolk,'/*.txt'))
martin = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(martin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(martin))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
```

```
tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM martin = DocumentTermMatrix(my documents)
DTM martin = removeSparseTerms(DTM martin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_martin = weightTfIdf(DTM_martin )
martin DF <- as.matrix(tfidf martin )</pre>
DTM_martin2 <- as.data.frame(as.matrix(DTM_martin ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MichaelConnor,'/*.txt'))
michael = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(michael ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(michael))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
```

```
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                        # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM michael = DocumentTermMatrix(my documents)
DTM_michael = removeSparseTerms(DTM_michael , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf michael = weightTfIdf(DTM michael )
michael_DF <- as.matrix(tfidf_michael )</pre>
DTM michael2 <- as.data.frame(as.matrix(DTM michael ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', MureDickie, '/*.txt'))
mure = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(mure ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(mure))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM mure = DocumentTermMatrix(my documents)
DTM mure = removeSparseTerms(DTM mure , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mure = weightTfIdf(DTM_mure )
mure DF <- as.matrix(tfidf mure )</pre>
DTM_mure2 <- as.data.frame(as.matrix(DTM_mure ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', NickLouth, '/*.txt'))
nick = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
```

```
{ strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(nick ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(nick))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                          # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_nick = DocumentTermMatrix(my_documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_nick = removeSparseTerms(DTM_nick , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf nick = weightTfIdf(DTM nick )
nick DF <- as.matrix(tfidf nick )</pre>
DTM_nick2 <- as.data.frame(as.matrix(DTM_nick ), stringsAsFactors=False)</pre>
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
```

```
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PatriciaCommins,'/*.txt'))
patricia = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(patricia ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(patricia))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM patricia = DocumentTermMatrix(my documents)
DTM patricia = removeSparseTerms(DTM patricia , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_patricia = weightTfIdf(DTM_patricia )
patricia DF <- as.matrix(tfidf patricia )</pre>
```

```
DTM patricia2 <- as.data.frame(as.matrix(DTM patricia ),</pre>
stringsAsFactors=False)
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PeterHumphrey,'/*.txt'))
peter = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(peter ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(peter))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
```

```
DTM peter = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to Learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_peter = removeSparseTerms(DTM_peter , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf peter = weightTfIdf(DTM peter )
peter DF <- as.matrix(tfidf peter )</pre>
DTM peter2 <- as.data.frame(as.matrix(DTM peter ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PierreTran,'/*.txt'))
pierre = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(pierre ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(pierre))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_pierre = DocumentTermMatrix(my_documents)
DTM pierre = removeSparseTerms(DTM pierre , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf pierre = weightTfIdf(DTM pierre )
pierre DF <- as.matrix(tfidf pierre )</pre>
DTM_pierre2 <- as.data.frame(as.matrix(DTM_pierre ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RobinSidel,'/*.txt'))
robin = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(robin ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(robin))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
```

```
tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM robin = DocumentTermMatrix(my documents)
DTM robin = removeSparseTerms(DTM robin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf robin = weightTfIdf(DTM robin )
robin DF <- as.matrix(tfidf robin )</pre>
DTM_robin2 <- as.data.frame(as.matrix(DTM_robin ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RogerFillion,'/*.txt'))
roger = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(roger ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(roger))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                    # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM roger = DocumentTermMatrix(my documents)
DTM roger = removeSparseTerms(DTM roger , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_roger = weightTfIdf(DTM_roger )
roger DF <- as.matrix(tfidf roger )</pre>
DTM_roger2 <- as.data.frame(as.matrix(DTM_roger ), stringsAsFactors=False)</pre>
### Next Author
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SarahDavison,'/*.txt'))
sarah = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
```

```
names(sarah ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(sarah))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm_map(content_transformer(removePunctuation)) %>%  # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM sarah = DocumentTermMatrix(my documents)
DTM sarah = removeSparseTerms(DTM sarah , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf sarah = weightTfIdf(DTM sarah )
sarah DF <- as.matrix(tfidf sarah )</pre>
DTM sarah2 <- as.data.frame(as.matrix(DTM sarah ), stringsAsFactors=False)</pre>
### Next Ouestion
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ScottHillis,'/*.txt'))
scott = lapply(file listAl, readerPlain)
# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(scott ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(scott))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
  tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
DTM_scott = DocumentTermMatrix(my_documents)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM scott = removeSparseTerms(DTM scott , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_scott = weightTfIdf(DTM_scott )
scott DF <- as.matrix(tfidf scott )</pre>
DTM scott2 <- as.data.frame(as.matrix(DTM scott ), stringsAsFactors=False)</pre>
### Next Author
```

```
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TheresePoletti,'/*.txt'))
therese = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(therese ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(therese))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM therese = DocumentTermMatrix(my documents)
DTM therese = removeSparseTerms(DTM therese , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf therese = weightTfIdf(DTM therese )
therese_DF <- as.matrix(tfidf_therese )</pre>
DTM therese2 <- as.data.frame(as.matrix(DTM therese ),
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TimFarrand,'/*.txt'))
tim = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(tim ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(tim))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
## create a doc-term-matrix from the corpus
DTM tim = DocumentTermMatrix(my documents)
```

```
DTM_tim = removeSparseTerms(DTM_tim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf tim = weightTfIdf(DTM tim )
tim DF <- as.matrix(tfidf tim )</pre>
DTM_tim2 <- as.data.frame(as.matrix(DTM_tim ), stringsAsFactors=False)</pre>
# Compare /cluster documents
####
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',WilliamKazer,'/*.txt'))
will = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(will ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(will))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM will = DocumentTermMatrix(my documents)
DTM_will = removeSparseTerms(DTM_will , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_will = weightTfIdf(DTM_will )
will_DF <- as.matrix(tfidf_will )</pre>
DTM will2 <- as.data.frame(as.matrix(DTM will ), stringsAsFactors=False)</pre>
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BernardHickey,'/*.txt'))
bernard = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(bernard ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(bernard))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content_transformer(removeNumbers)) %>% # remove numbers
  tm map(content transformer(removePunctuation)) %>%  # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM bernard = DocumentTermMatrix(my documents)
## ...find words with greater than a min count...
DTM bernard = removeSparseTerms(DTM bernard , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf bernard = weightTfIdf(DTM bernard )
bernard_DF <- as.matrix(tfidf_bernard )</pre>
DTM bernard2 <- as.data.frame(as.matrix(DTM bernard ),</pre>
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MatthewBunce,'/*.txt'))
matthew = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(matthew ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(matthew))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
tm map(content transformer(tolower)) %>%
                                                        # make everything
```

```
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                     # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM matthew = DocumentTermMatrix(my documents)
DTM matthew = removeSparseTerms(DTM matthew , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf matthew = weightTfIdf(DTM matthew )
bernard_DF <- as.matrix(tfidf_matthew )</pre>
DTM matthew2 <- as.data.frame(as.matrix(DTM matthew ),</pre>
stringsAsFactors=False)
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/', SamuelPerry, '/*.txt'))
samuel = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(samuel ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(samuel))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 # make everything
Lowercase
                                                   # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                               # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM samuel = DocumentTermMatrix(my documents)
DTM samuel = removeSparseTerms(DTM samuel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf samuel = weightTfIdf(DTM samuel )
bernard_DF <- as.matrix(tfidf_samuel )</pre>
DTM samuel2 <- as.data.frame(as.matrix(DTM samuel ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TanEeLyn,'/*.txt'))
tan = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(tan ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tan))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my documents <- tm map(my documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsqithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM tan = DocumentTermMatrix(my documents)
DTM tan = removeSparseTerms(DTM tan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf tan = weightTfIdf(DTM tan )
bernard DF <- as.matrix(tfidf tan )</pre>
DTM_tan2 <- as.data.frame(as.matrix(DTM_tan ), stringsAsFactors=False)</pre>
### Next Author
file listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ToddNissen,'/*.txt'))
todd = lapply(file listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file listAl %>%
{ strsplit(., '/', fixed=TRUE) } %>%
```

```
{ lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(todd ) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(todd))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM todd = DocumentTermMatrix(my documents)
DTM todd = removeSparseTerms(DTM todd , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_todd = weightTfIdf(DTM_todd )
bernard_DF <- as.matrix(tfidf_todd )</pre>
DTM_todd2 <- as.data.frame(as.matrix(DTM_todd ), stringsAsFactors=False)</pre>
library(e1071)
### Creating Tests
#Author1
readerPlain = function(fname){
 readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
```

```
0test/')
#file list + list folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {</pre>
  paste(..., sep=sep, collapse=sep)
for (i in list folders){
 folder = i
  assign(i,i)
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SimonCowell,'/*.txt'))
simon test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(simon_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
```

```
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
#?stopwords
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM simon test = DocumentTermMatrix(my documents)
DTM simon test = removeSparseTerms(DTM simon test, 0.95)
#DTM_simon_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf simon test = weightTfIdf(DTM simon test)
simon DF <- as.data.frame(as.matrix(tfidf simon))</pre>
simon DF test <- as.data.frame(as.matrix(tfidf simon test))</pre>
simon_words <- names(simon_DF)</pre>
simon words test <- names(simon DF test)</pre>
remove simon <- simon words test[!(simon words test %in% simon words)]
simon DF test <- simon DF test[, !colnames(simon DF test) %in% remove simon]</pre>
### Next Author
list folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0test/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list_folders[i]+'/*.txt'))
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', AaronPressman, '/*.txt'))
aaron_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
```

```
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(aaron test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                    # remove numbers
  tm map(content transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainaaroncowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM aaron test = DocumentTermMatrix(my documents)
DTM_aaron_test = removeSparseTerms(DTM_aaron_test, 0.95)
#DTM aaron test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf aaron test = weightTfIdf(DTM aaron test)
aaron_DF <- as.data.frame(as.matrix(tfidf_aaron))</pre>
aaron DF test <- as.data.frame(as.matrix(tfidf aaron test))</pre>
aaron words <- names(aaron DF)</pre>
aaron_words_test <- names(aaron_DF_test)</pre>
```

```
remove aaron <- aaron words test[!(aaron words test %in% aaron words)]
aaron DF test <- aaron DF test[, !colnames(aaron DF test) %in% remove aaron]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', AlanCrosby, '/*.txt'))
alan test = lapply(file list test, readerPlain)
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(alan test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                        # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
\mathbf{c} ("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalancowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM alan test = DocumentTermMatrix(my documents)
```

```
DTM_alan_test = removeSparseTerms(DTM_alan_test, 0.95)
tfidf_alan_test = weightTfIdf(DTM_alan_test)
alan DF <- as.data.frame(as.matrix(tfidf alan))</pre>
alan DF test <- as.data.frame(as.matrix(tfidf alan test))</pre>
alan_words <- names(alan_DF)</pre>
alan_words_test <- names(alan DF test)</pre>
remove alan <- alan words test[!(alan words test %in% alan words)]
alan DF test <- alan DF test[, !colnames(alan DF test) %in% remove alan]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AlexanderSmith,'/*.txt'))
alex_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(alex_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
```

```
tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
\mathbf{c} ("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalexcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM alex test = DocumentTermMatrix(my documents)
DTM alex test = removeSparseTerms(DTM alex test, 0.95)
#DTM_alex_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf alex test = weightTfIdf(DTM alex test)
alex DF <- as.data.frame(as.matrix(tfidf alex))</pre>
alex DF test <- as.data.frame(as.matrix(tfidf alex test))</pre>
alex words <- names(alex DF)</pre>
alex words test <- names(alex DF test)</pre>
remove alex <- alex words test[!(alex words test %in% alex words)]</pre>
alex DF test <- alex DF test[, !colnames(alex DF test) %in% remove alex]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', BenjaminKangLim, '/*.txt'))
ben_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(ben_test) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbencowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM ben test = DocumentTermMatrix(my documents)
DTM ben test = removeSparseTerms(DTM_ben_test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben_test = weightTfIdf(DTM_ben_test)
ben DF <- as.data.frame(as.matrix(tfidf ben))</pre>
ben DF test <- as.data.frame(as.matrix(tfidf ben test))</pre>
ben_words <- names(ben_DF)</pre>
ben words test <- names(ben DF test)</pre>
remove ben <- ben words test[!(ben words test %in% ben words)]
ben_DF_test <- ben_DF_test[, !colnames(ben_DF_test) %in% remove_ben]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', BernardHickey, '/*.txt'))
bernard test = lapply(file list test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(bernard test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(bernard test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
  tm map(content transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbernardcowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM bernard test = DocumentTermMatrix(my documents)
DTM bernard test = removeSparseTerms(DTM bernard test, 0.95)
#DTM bernard test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf bernard test = weightTfIdf(DTM bernard test)
bernard DF <- as.data.frame(as.matrix(tfidf bernard))</pre>
bernard DF test <- as.data.frame(as.matrix(tfidf bernard test))</pre>
bernard_words <- names(bernard_DF)</pre>
bernard words test <- names(bernard DF test)</pre>
remove bernard <- bernard words test[!(bernard words test %in%
bernard words)]
bernard_DF_test <- bernard_DF_test[, !colnames(bernard_DF_test) %in%</pre>
remove bernard]
```

```
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BradDorfman,'/*.txt'))
brad_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
names(brad_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(brad test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>%
                                                    # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbradcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM brad test = DocumentTermMatrix(my documents)
## Finally, let's drop those terms that only occur in one or two documents
```

```
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad_test = removeSparseTerms(DTM_brad_test, 0.95)
#DTM brad test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf brad test = weightTfIdf(DTM brad test)
brad DF <- as.data.frame(as.matrix(tfidf brad))</pre>
brad_DF_test <- as.data.frame(as.matrix(tfidf_brad_test))</pre>
brad words <- names(brad DF)</pre>
brad_words_test <- names(brad_DF_test)</pre>
remove brad <- brad words test[!(brad words test %in% brad words)]
brad_DF_test <- brad_DF_test[, !colnames(brad DF test) %in% remove brad]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DarrenSchuettler,'/*.txt'))
darren_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(darren test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren_test))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindarrencowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM darren test = DocumentTermMatrix(my documents)
DTM_darren_test = removeSparseTerms(DTM_darren_test, 0.95)
#DTM darren test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf darren test = weightTfIdf(DTM darren test)
darren DF <- as.data.frame(as.matrix(tfidf darren))</pre>
darren_DF_test <- as.data.frame(as.matrix(tfidf_darren_test))</pre>
darren_words <- names(darren_DF)</pre>
darren_words_test <- names(darren_DF_test)</pre>
remove_darren <- darren_words_test[!(darren_words_test %in% darren_words)]
darren DF test <- darren DF test[, !colnames(darren DF test) %in%</pre>
remove darren]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DavidLawder,'/*.txt'))
david_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
```

```
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(david test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindavidcowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM david test = DocumentTermMatrix(my documents)
DTM david test = removeSparseTerms(DTM david test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf david test = weightTfIdf(DTM david test)
david DF <- as.data.frame(as.matrix(tfidf david))</pre>
david_DF_test <- as.data.frame(as.matrix(tfidf_david_test))</pre>
david_words <- names(david_DF)</pre>
david_words_test <- names(david_DF_test)</pre>
remove david <- david words test[!(david words test %in% david words)]
david_DF_test <- david_DF_test[, !colnames(david_DF_test) %in% remove_david]</pre>
### Next Author
```

```
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', EdnaFernandes, '/*.txt'))
edna test = lapply(file list test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(edna_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(edna test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm_map(content_transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainednacowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM_edna_test = DocumentTermMatrix(my_documents)
DTM edna test = removeSparseTerms(DTM edna test, 0.95)
#DTM_edna_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf edna test = weightTfIdf(DTM edna test)
edna_DF <- as.data.frame(as.matrix(tfidf_edna))</pre>
edna DF test <- as.data.frame(as.matrix(tfidf edna test))</pre>
edna_words <- names(edna_DF)
edna_words_test <- names(edna_DF_test)</pre>
remove edna <- edna words test[!(edna words test %in% edna words)]
edna_DF_test <- edna_DF_test[, !colnames(edna_DF_test) %in% remove_edna]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',EricAuchard,'/*.txt'))
eric_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(eric test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
```

```
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainericcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM eric test = DocumentTermMatrix(my documents)
#DTM eric test # some basic summary statistics
## You can inspect its entries...
#inspect(DTM eric test[1:10,1:20])
## ...find words with greater than a min count...
#findFreqTerms(DTM eric test, 50)
## ...or find words whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
#findAssocs(DTM_eric_test, "genetic", .5)
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_eric_test = removeSparseTerms(DTM_eric_test, 0.95)
#DTM eric test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf eric test = weightTfIdf(DTM eric test)
eric_DF <- as.data.frame(as.matrix(tfidf_eric))</pre>
eric_DF_test <- as.data.frame(as.matrix(tfidf_eric_test))</pre>
eric words <- names(eric DF)</pre>
eric_words_test <- names(eric_DF_test)</pre>
remove eric <- eric words test[!(eric words test %in% eric words)]</pre>
eric DF test <- eric DF test[, !colnames(eric DF test) %in% remove eric]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',FumikoFujisaki,'/*.txt'))
```

```
fumiko test = lapply(file list test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(fumiko test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%  # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainfumikocowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM fumiko test = DocumentTermMatrix(my documents)
DTM fumiko test = removeSparseTerms(DTM fumiko test, 0.95)
#DTM_fumiko_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf fumiko test = weightTfIdf(DTM fumiko test)
fumiko DF <- as.data.frame(as.matrix(tfidf fumiko))</pre>
fumiko_DF_test <- as.data.frame(as.matrix(tfidf_fumiko_test))</pre>
fumiko words <- names(fumiko DF)</pre>
fumiko words test <- names(fumiko DF test)</pre>
remove_fumiko <- fumiko_words_test[!(fumiko_words_test %in% fumiko_words)]</pre>
```

```
fumiko DF test <- fumiko DF test[, !colnames(fumiko DF test) %in%</pre>
remove fumiko]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',GrahamEarnshaw,'/*.txt'))
graham test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(graham test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
                                                      # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                       # remove excess
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraingrahamcowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
```

```
DTM graham test = DocumentTermMatrix(my documents)
DTM graham test = removeSparseTerms(DTM graham test, 0.95)
#DTM_graham_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham_test = weightTfIdf(DTM_graham_test)
graham_DF <- as.data.frame(as.matrix(tfidf graham))</pre>
graham_DF_test <- as.data.frame(as.matrix(tfidf_graham_test))</pre>
graham words <- names(graham DF)</pre>
graham words test <- names(graham DF test)</pre>
remove graham <- graham words test[!(graham words test %in% graham words)]
graham_DF_test <- graham_DF_test[, !colnames(graham_DF_test) %in%</pre>
remove_graham]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',HeatherScoffield,'/*.txt'))
heather test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(heather test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
```

```
tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                        # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainheathercowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM heather test = DocumentTermMatrix(my documents)
DTM heather test = removeSparseTerms(DTM heather test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf heather test = weightTfIdf(DTM heather test)
heather_DF <- as.data.frame(as.matrix(tfidf_heather))</pre>
heather DF test <- as.data.frame(as.matrix(tfidf heather test))</pre>
heather words <- names(heather DF)
heather_words_test <- names(heather_DF_test)</pre>
remove heather <- heather words test[!(heather words test %in%
heather words)]
heather_DF_test <- heather_DF_test[, !colnames(heather_DF_test) %in%
remove heather]
## Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', JanLopatka, '/*.txt'))
jan_test = lapply(file_list_test, readerPlain)
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
#mynames
names(jan_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjancowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM_jan_test = DocumentTermMatrix(my_documents)
DTM jan test = removeSparseTerms(DTM jan test, 0.95)
#DTM jan test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jan test = weightTfIdf(DTM jan test)
jan DF <- as.data.frame(as.matrix(tfidf jan))</pre>
jan_DF_test <- as.data.frame(as.matrix(tfidf_jan_test))</pre>
jan words <- names(jan DF)</pre>
jan_words_test <- names(jan_DF_test)</pre>
remove_jan <- jan_words_test[!(jan_words_test %in% jan_words)]</pre>
jan DF test <- jan DF test[, !colnames(jan DF test) %in% remove jan]</pre>
### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', JaneMacartney, '/*.txt'))
jane test = lapply(file list test, readerPlain)
```

```
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(jane_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjanecowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM jane test = DocumentTermMatrix(my documents)
DTM jane test = removeSparseTerms(DTM jane test, 0.95)
#DTM jane test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jane test = weightTfIdf(DTM jane test)
jane_DF <- as.data.frame(as.matrix(tfidf_jane))</pre>
```

```
jane DF test <- as.data.frame(as.matrix(tfidf jane test))</pre>
jane words <- names(jane DF)</pre>
jane_words_test <- names(jane_DF_test)</pre>
remove_jane <- jane_words_test[!(jane_words_test %in% jane words)]</pre>
jane DF_test <- jane DF test[, !colnames(jane DF test) %in% remove jane]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JimGilchrist,'/*.txt'))
jim test = lapply(file list test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(jim_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                  # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
```

```
#stopwords("SMART")
# Let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjimcowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM jim test = DocumentTermMatrix(my documents)
DTM jim test = removeSparseTerms(DTM jim test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jim test = weightTfIdf(DTM jim test)
jim_DF <- as.data.frame(as.matrix(tfidf_jim))</pre>
jim_DF_test <- as.data.frame(as.matrix(tfidf_jim_test))</pre>
jim_words <- names(jim_DF)</pre>
jim_words_test <- names(jim_DF_test)</pre>
remove_jim <- jim_words_test[!(jim_words_test %in% jim_words)]</pre>
jim_DF_test <- jim_DF_test[, !colnames(jim_DF_test) %in% remove_jim]</pre>
### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', JoWinterbottom, '/*.txt'))
jo_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(jo test) = mynames
## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(jo test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                           # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjocowellnewsmltxt")
## create a doc-term-matrix from the corpus
DTM jo test = DocumentTermMatrix(my documents)
DTM jo test = removeSparseTerms(DTM jo test, 0.95)
#DTM_jo_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jo test = weightTfIdf(DTM jo test)
jo DF <- as.data.frame(as.matrix(tfidf jo))</pre>
jo_DF_test <- as.data.frame(as.matrix(tfidf_jo_test))</pre>
jo_words <- names(jo_DF)</pre>
jo words test <- names(jo DF test)</pre>
remove_jo <- jo_words_test[!(jo_words_test %in% jo_words)]</pre>
jo_DF_test <- jo_DF_test[, !colnames(jo_DF_test) %in% remove_jo]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JoeOrtiz,'/*.txt'))
```

```
joe test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(joe_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjoecowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM_joe_test = DocumentTermMatrix(my_documents)
DTM joe test = removeSparseTerms(DTM joe test, 0.95)
#DTM_joe_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf joe test = weightTfIdf(DTM joe test)
joe_DF <- as.data.frame(as.matrix(tfidf_joe))</pre>
joe DF test <- as.data.frame(as.matrix(tfidf joe test))</pre>
joe_words <- names(joe_DF)</pre>
joe_words_test <- names(joe_DF_test)</pre>
remove joe <- joe words test[!(joe words test %in% joe words)]
joe_DF_test <- joe_DF_test[, !colnames(joe_DF_test) %in% remove_joe]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JohnMastrini,'/*.txt'))
john_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(john_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                  # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjohncowellnewsmltxt
```

```
"))
## create a doc-term-matrix from the corpus
DTM john test = DocumentTermMatrix(my documents)
DTM_john_test = removeSparseTerms(DTM_john_test, 0.95)
#DTM john test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf john test = weightTfIdf(DTM john test)
john DF <- as.data.frame(as.matrix(tfidf john))</pre>
john DF test <- as.data.frame(as.matrix(tfidf john test))</pre>
john_words <- names(john_DF)</pre>
john_words_test <- names(john_DF_test)</pre>
remove john <- john words test[!(john words test %in% john words)]
john_DF_test <- john_DF_test[, !colnames(john_DF_test) %in% remove_john]</pre>
### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', JonathanBirt, '/*.txt'))
jonathan test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(jonathan test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan_test))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                     # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjonathancowellnewsm
ltxt"))
## create a doc-term-matrix from the corpus
DTM_jonathan_test = DocumentTermMatrix(my_documents)
DTM_jonathan_test = removeSparseTerms(DTM_jonathan_test, 0.95)
#DTM jonathan test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf jonathan test = weightTfIdf(DTM jonathan test)
jonathan DF <- as.data.frame(as.matrix(tfidf jonathan))</pre>
jonathan_DF_test <- as.data.frame(as.matrix(tfidf_jonathan_test))</pre>
jonathan_words <- names(jonathan_DF)</pre>
jonathan_words_test <- names(jonathan_DF_test)</pre>
remove_jonathan <- jonathan_words_test[!(jonathan_words_test %in%</pre>
jonathan words)]
jonathan DF test <- jonathan DF test[, !colnames(jonathan DF test) %in%</pre>
remove jonathan]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', KarlPenhaul, '/*.txt'))
karl_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(karl test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(karl test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkarlcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM karl test = DocumentTermMatrix(my documents)
DTM karl test = removeSparseTerms(DTM karl test, 0.95)
#DTM_karl_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl_test = weightTfIdf(DTM_karl_test)
karl DF <- as.data.frame(as.matrix(tfidf karl))</pre>
karl_DF_test <- as.data.frame(as.matrix(tfidf_karl_test))</pre>
karl_words <- names(karl_DF)</pre>
karl words test <- names(karl DF test)</pre>
remove karl <- karl words test[!(karl words test %in% karl words)]</pre>
```

```
karl DF test <- karl DF test[, !colnames(karl DF test) %in% remove karl]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KeithWeir,'/*.txt'))
keith test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(keith_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                           # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
tm_map(content_transformer(removePunctuation)) %>% # remove numbers
# remove numbers
  tm map(content transformer(stripWhitespace))
                                                    # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
```

```
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkeithcowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM keith test = DocumentTermMatrix(my documents)
DTM_keith_test = removeSparseTerms(DTM_keith_test, 0.95)
#DTM_keith_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith_test = weightTfIdf(DTM_keith_test)
keith_DF <- as.data.frame(as.matrix(tfidf_keith))</pre>
keith_DF_test <- as.data.frame(as.matrix(tfidf_keith_test))</pre>
keith words <- names(keith DF)</pre>
keith_words_test <- names(keith_DF_test)</pre>
remove_keith <- keith_words_test[!(keith_words_test %in% keith_words)]</pre>
keith DF test <- keith DF test[, !colnames(keith DF test) %in% remove keith]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KevinDrawbaugh,'/*.txt'))
kevind_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(kevind test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents raw = Corpus(VectorSource(kevind test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                       # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkevindcowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM kevind test = DocumentTermMatrix(my documents)
DTM kevind test = removeSparseTerms(DTM kevind test, 0.95)
#DTM kevind test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kevind test = weightTfIdf(DTM kevind test)
kevind DF <- as.data.frame(as.matrix(tfidf kevind))</pre>
kevind_DF_test <- as.data.frame(as.matrix(tfidf_kevind_test))</pre>
kevind_words <- names(kevind_DF)</pre>
kevind_words_test <- names(kevind_DF_test)</pre>
remove kevind <- kevind words_test[!(kevind_words_test %in% kevind_words)]</pre>
kevind DF test <- kevind DF test[, !colnames(kevind DF test) %in%
remove kevind]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KevinMorrison,'/*.txt'))
kevinm test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(kevinm_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkevinmcowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM_kevinm_test = DocumentTermMatrix(my_documents)
DTM kevinm test = removeSparseTerms(DTM kevinm test, 0.95)
#DTM kevinm test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm_test = weightTfIdf(DTM_kevinm_test)
kevinm DF <- as.data.frame(as.matrix(tfidf kevinm))</pre>
kevinm DF test <- as.data.frame(as.matrix(tfidf kevinm test))</pre>
kevinm_words <- names(kevinm_DF)</pre>
```

```
kevinm words test <- names(kevinm DF test)</pre>
remove kevinm <- kevinm words test[!(kevinm words test %in% kevinm words)]
kevinm_DF_test <- kevinm_DF_test[, !colnames(kevinm_DF_test) %in%</pre>
remove kevinm]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KirstinRidley,'/*.txt'))
kirstin_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(kirstin_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kirstin test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                     # remove excess
white-space
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
```

```
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkirstincowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM kirstin test = DocumentTermMatrix(my documents)
DTM kirstin test = removeSparseTerms(DTM kirstin test, 0.95)
#DTM_kirstin_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin_test = weightTfIdf(DTM_kirstin_test)
kirstin DF <- as.data.frame(as.matrix(tfidf kirstin))</pre>
kirstin_DF_test <- as.data.frame(as.matrix(tfidf_kirstin_test))</pre>
kirstin_words <- names(kirstin_DF)</pre>
kirstin words test <- names(kirstin DF test)</pre>
remove kirstin <- kirstin words test[!(kirstin words test %in%
kirstin words)]
kirstin DF test <- kirstin DF test[, !colnames(kirstin DF test) %in%
remove_kirstin]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KouroshKarimkhany,'/*.txt'))
kourosh_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(kourosh test) = mynames
## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(kourosh test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                           # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm map(content transformer(stripWhitespace))
                                                     # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkouroshcowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM kourosh test = DocumentTermMatrix(my documents)
## Probably a bit stringent here... but only 50 docs!
DTM kourosh test = removeSparseTerms(DTM kourosh test, 0.95)
#DTM_kourosh_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf kourosh test = weightTfIdf(DTM kourosh test)
kourosh DF <- as.data.frame(as.matrix(tfidf kourosh))</pre>
kourosh_DF_test <- as.data.frame(as.matrix(tfidf kourosh test))</pre>
kourosh words <- names(kourosh DF)</pre>
kourosh words test <- names(kourosh DF test)</pre>
remove kourosh <- kourosh words test[!(kourosh words test %in%
kourosh words)]
kourosh_DF_test <- kourosh_DF_test[, !colnames(kourosh_DF_test) %in%</pre>
remove_kourosh]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',LydiaZajc,'/*.txt'))
lydia_test = lapply(file_list_test, readerPlain)
```

```
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(lydia_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lydia_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%  # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainlydiacowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lydia test = DocumentTermMatrix(my documents)
DTM lydia test = removeSparseTerms(DTM lydia test, 0.95)
#DTM Lydia test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
```

```
tfidf lydia test = weightTfIdf(DTM lydia test)
lydia DF <- as.data.frame(as.matrix(tfidf lydia))</pre>
lydia_DF_test <- as.data.frame(as.matrix(tfidf_lydia_test))</pre>
lydia words <- names(lydia DF)</pre>
lydia_words_test <- names(lydia_DF_test)</pre>
remove_lydia <- lydia_words_test[!(lydia_words_test %in% lydia_words)]</pre>
lydiah_DF_test <- lydia_DF_test[, !colnames(lydia_DF_test) %in% remove_lydia]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',`LynneO'Donnell`,'/*.txt'))
lynne test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(lynne_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynne_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
 # make everything
Lowercase
                                                     # remove numbers
 tm map(content transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace))
                                                      # remove excess
white-space
# let's just use the "basic English" stop words
```

```
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainlynnecowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM lynne test = DocumentTermMatrix(my documents)
DTM lynne test = removeSparseTerms(DTM lynne test, 0.95)
#DTM Lynne test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynne_test = weightTfIdf(DTM_lynne_test)
lynne DF <- as.data.frame(as.matrix(tfidf lynne))</pre>
lynne_DF_test <- as.data.frame(as.matrix(tfidf_lynne_test))</pre>
lynne words <- names(lynne DF)</pre>
lynne words test <- names(lynne DF test)</pre>
remove lynne <- lynne words test[!(lynne words test %in% lynne words)]
lynne_DF_test <- lynne_DF_test[, !colnames(lynne_DF_test) %in% remove_lynne]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',LynnleyBrowning,'/*.txt'))
lynnley_test = lapply(file_list_test, readerPlain)
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mvnames
names(lynnley_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(lynnley test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
```

```
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                        # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                         # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainlynnleycowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM lynnley test = DocumentTermMatrix(my documents)
DTM lynnley test = removeSparseTerms(DTM lynnley test, 0.95)
#DTM_Lynnley_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynnley_test = weightTfIdf(DTM_lynnley_test)
lynnley DF <- as.data.frame(as.matrix(tfidf lynnley))</pre>
lynnley_DF_test <- as.data.frame(as.matrix(tfidf_lynnley_test))</pre>
lynnley_words <- names(lynnley_DF)</pre>
lynnley words test <- names(lynnley DF test)</pre>
remove lynnley <- lynnley words test[!(lynnley words test %in%
lynnley words)]
lynnley DF test <- lynnley DF test[, !colnames(lynnley DF test) %in%
remove lynnley]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MarcelMichelson,'/*.txt'))
marcel test = lapply(file list test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(marcel test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(marcel_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                  # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmarcelcowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM marcel test = DocumentTermMatrix(my documents)
DTM marcel test = removeSparseTerms(DTM marcel test, 0.95)
#DTM marcel test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf marcel test = weightTfIdf(DTM marcel test)
marcel_DF <- as.data.frame(as.matrix(tfidf_marcel))</pre>
marcel DF test <- as.data.frame(as.matrix(tfidf marcel test))</pre>
marcel_words <- names(marcel_DF)</pre>
marcel_words_test <- names(marcel_DF_test)</pre>
remove_marcel <- marcel_words_test[!(marcel_words_test %in% marcel words)]</pre>
marcel DF test <- marcel DF test[, !colnames(marcel DF test) %in%</pre>
```

```
remove marcel]
### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MarkBendeich,'/*.txt'))
mark test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(mark_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mark_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmarkcowellnewsmltxt
"))
```

```
## create a doc-term-matrix from the corpus
DTM mark test = DocumentTermMatrix(my documents)
DTM mark test = removeSparseTerms(DTM mark test, 0.95)
#DTM mark test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mark_test = weightTfIdf(DTM_mark_test)
mark DF <- as.data.frame(as.matrix(tfidf mark))</pre>
mark DF test <- as.data.frame(as.matrix(tfidf mark test))</pre>
mark words <- names(mark DF)</pre>
mark words test <- names(mark DF test)</pre>
remove_mark <- mark_words_test[!(mark_words_test %in% mark_words)]</pre>
mark_DF_test <- mark_DF_test[, !colnames(mark_DF_test) %in% remove_mark]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', MartinWolk, '/*.txt'))
martin_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mvnames
names(martin_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(martin test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
```

```
my documents = documents raw %>%
                                                         # make everything
  tm map(content transformer(tolower)) %>%
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmartincowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM martin test = DocumentTermMatrix(my documents)
DTM martin test = removeSparseTerms(DTM martin test, 0.95)
#DTM_martin_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_martin_test = weightTfIdf(DTM_martin_test)
martin DF <- as.data.frame(as.matrix(tfidf martin))</pre>
martin DF test <- as.data.frame(as.matrix(tfidf martin test))</pre>
martin_words <- names(martin_DF)</pre>
martin words test <- names(martin DF test)</pre>
remove martin <- martin words test[!(martin words test %in% martin words)]
martin DF test <- martin DF test[, !colnames(martin DF test) %in%</pre>
remove martin]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', MatthewBunce, '/*.txt'))
matthew_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
```

```
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(matthew test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(matthew_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
 tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
 tm map(content transformer(removeNumbers)) %>%
                                                    # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace))
                                                   # remove excess
white-space
## Remove stopwords. Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmatthewcowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM_matthew_test = DocumentTermMatrix(my_documents)
DTM_matthew_test = removeSparseTerms(DTM_matthew_test, 0.95)
#DTM matthew test # now ~ 1000 terms (versus ~3000 before)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_matthew_test = weightTfIdf(DTM_matthew_test)
matthew DF <- as.data.frame(as.matrix(tfidf matthew))</pre>
matthew_DF_test <- as.data.frame(as.matrix(tfidf_matthew_test))</pre>
matthew_words <- names(matthew_DF)</pre>
matthew words test <- names(matthew DF test)</pre>
remove_matthew <- matthew_words_test[!(matthew_words_test %in%</pre>
matthew words)]
matthew DF test <- matthew DF test[, !colnames(matthew DF test) %in%</pre>
remove_matthew]
### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MichaelConnor,'/*.txt'))
michael test = lapply(file list test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(michael test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(michael test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm map(content transformer(stripWhitespace)) # remove excess
```

```
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmichaelcowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM michael test = DocumentTermMatrix(my documents)
DTM michael test = removeSparseTerms(DTM michael test, 0.95)
#DTM michael test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_michael_test = weightTfIdf(DTM_michael_test)
michael DF <- as.data.frame(as.matrix(tfidf michael))</pre>
michael_DF_test <- as.data.frame(as.matrix(tfidf_michael_test))</pre>
michael words <- names(michael DF)</pre>
michael words test <- names(michael DF test)</pre>
remove michael <- michael words test[!(michael words test %in%
michael words)]
michael DF test <- michael DF test[, !colnames(michael DF test) %in%
remove michael]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', MureDickie, '/*.txt'))
mure test = lapply(file list test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
#mynames
names(mure test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(mure test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmurecowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM mure test = DocumentTermMatrix(my documents)
DTM mure test = removeSparseTerms(DTM mure test, 0.95)
#DTM_mure_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf mure test = weightTfIdf(DTM mure test)
mure DF <- as.data.frame(as.matrix(tfidf mure))</pre>
mure_DF_test <- as.data.frame(as.matrix(tfidf_mure_test))</pre>
mure words <- names(mure DF)</pre>
mure_words_test <- names(mure_DF_test)</pre>
remove_mure <- mure_words_test[!(mure_words_test %in% mure_words)]</pre>
mure_DF_test <- mure_DF_test[, !colnames(mure_DF_test) %in% remove_mure]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', NickLouth, '/*.txt'))
```

```
nick_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(nick test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(nick test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
\textbf{c} (\texttt{"cusersmachuonedrive} documents \texttt{githubstadatareuters} cctrainnick cowellnews \texttt{mltxt})
"))
## create a doc-term-matrix from the corpus
DTM_nick_test = DocumentTermMatrix(my_documents)
DTM nick test = removeSparseTerms(DTM nick test, 0.95)
#DTM_nick_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf nick test = weightTfIdf(DTM nick test)
nick_DF <- as.data.frame(as.matrix(tfidf_nick))</pre>
nick DF test <- as.data.frame(as.matrix(tfidf nick test))</pre>
nick_words <- names(nick_DF)</pre>
nick_words_test <- names(nick_DF_test)</pre>
remove nick <- nick words test[!(nick words test %in% nick words)]
nick_DF_test <- nick_DF_test[, !colnames(nick_DF_test) %in% remove_nick]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',PatriciaCommins,'/*.txt'))
patricia test = lapply(file list test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(patricia test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(patricia test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainpatriciacowellnewsm
ltxt"))
## create a doc-term-matrix from the corpus
DTM patricia test = DocumentTermMatrix(my documents)
DTM patricia test = removeSparseTerms(DTM patricia test, 0.95)
#DTM_patricia_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf patricia test = weightTfIdf(DTM patricia test)
patricia_DF <- as.data.frame(as.matrix(tfidf_patricia))</pre>
patricia_DF_test <- as.data.frame(as.matrix(tfidf_patricia_test))</pre>
patricia words <- names(patricia DF)</pre>
patricia_words_test <- names(patricia_DF_test)</pre>
remove patricia <- patricia words test[!(patricia words test %in%
patricia words)]
patricia_DF_test <- patricia_DF_test[, !colnames(patricia_DF_test) %in%</pre>
remove patricia]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/', PeterHumphrey, '/*.txt'))
peter_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
```

```
names(peter test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(peter test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainpetercowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM peter test = DocumentTermMatrix(my documents)
DTM peter test = removeSparseTerms(DTM peter test, 0.95)
#DTM peter test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_peter_test = weightTfIdf(DTM_peter_test)
peter DF <- as.data.frame(as.matrix(tfidf peter))</pre>
peter_DF_test <- as.data.frame(as.matrix(tfidf_peter_test))</pre>
peter_words <- names(peter_DF)</pre>
peter words test <- names(peter DF test)</pre>
remove_peter <- peter_words_test[!(peter_words_test %in% peter_words)]</pre>
peter_DF_test <- peter_DF_test[, !colnames(peter_DF_test) %in% remove_peter]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',PierreTran,'/*.txt'))
pierre_test = lapply(file_list_test, readerPlain)
```

```
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(pierre_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(pierre test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                  # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainpierrecowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM_pierre_test = DocumentTermMatrix(my_documents)
DTM pierre test = removeSparseTerms(DTM pierre test, 0.95)
#DTM pierre test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf pierre test = weightTfIdf(DTM pierre test)
pierre DF <- as.data.frame(as.matrix(tfidf pierre))</pre>
pierre DF test <- as.data.frame(as.matrix(tfidf pierre test))</pre>
```

```
pierre words <- names(pierre DF)</pre>
pierre words test <- names(pierre DF test)</pre>
remove_pierre <- pierre_words_test[!(pierre_words_test %in% pierre_words)]</pre>
pierre_DF_test <- pierre_DF_test[, !colnames(pierre_DF_test) %in%</pre>
remove_pierre]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',RobinSidel,'/*.txt'))
robin_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mvnames
names(robin_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(robin test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainrobincowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM robin test = DocumentTermMatrix(my documents)
DTM_robin_test = removeSparseTerms(DTM_robin_test, 0.95)
#DTM_robin_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_robin_test = weightTfIdf(DTM_robin_test)
robin_DF <- as.data.frame(as.matrix(tfidf_robin))</pre>
robin_DF_test <- as.data.frame(as.matrix(tfidf_robin_test))</pre>
robin words <- names(robin DF)</pre>
robin_words_test <- names(robin_DF_test)</pre>
remove_robin <- robin_words_test[!(robin_words_test %in% robin_words)]</pre>
robin DF test <- robin DF test[, !colnames(robin DF test) %in% remove robin]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',RogerFillion,'/*.txt'))
roger_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>% { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(roger_test) = mynames
## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(roger test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                           # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%  # remove numbers
tm_map(content_transformer(removePunctuation)) %>%  # remove numbers
# remove punctuation
  tm map(content transformer(stripWhitespace))
                                                     # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainrogercowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM roger test = DocumentTermMatrix(my documents)
DTM roger test = removeSparseTerms(DTM roger test, 0.95)
#DTM_roger_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf roger test = weightTfIdf(DTM roger test)
roger DF <- as.data.frame(as.matrix(tfidf roger))</pre>
roger DF_test <- as.data.frame(as.matrix(tfidf_roger_test))</pre>
roger words <- names(roger DF)</pre>
roger words test <- names(roger DF test)</pre>
remove_roger <- roger_words_test[!(roger_words_test %in% roger_words)]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SamuelPerry,'/*.txt'))
samuel_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(samuel_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(samuel_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                      # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsamuelcowellnewsmlt
xt"))
## create a doc-term-matrix from the corpus
DTM samuel test = DocumentTermMatrix(my documents)
DTM samuel test = removeSparseTerms(DTM samuel test, 0.95)
#DTM samuel test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_samuel_test = weightTfIdf(DTM_samuel_test)
samuel DF <- as.data.frame(as.matrix(tfidf samuel))</pre>
samuel_DF_test <- as.data.frame(as.matrix(tfidf_samuel_test))</pre>
samuel words <- names(samuel DF)</pre>
```

```
samuel words test <- names(samuel DF test)</pre>
remove samuel <- samuel words test[!(samuel words test %in% samuel words)]
samuel_DF_test <- samuel_DF_test[, !colnames(samuel_DF_test) %in%</pre>
remove samuel]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SarahDavison,'/*.txt'))
sarah_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(sarah test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(sarah_test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsarahcowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM_sarah_test = DocumentTermMatrix(my_documents)
DTM_sarah_test = removeSparseTerms(DTM_sarah_test, 0.95)
#DTM sarah test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf sarah test = weightTfIdf(DTM sarah test)
sarah_DF <- as.data.frame(as.matrix(tfidf_sarah))</pre>
sarah_DF_test <- as.data.frame(as.matrix(tfidf_sarah_test))</pre>
sarah words <- names(sarah DF)</pre>
sarah_words_test <- names(sarah_DF_test)</pre>
remove sarah <- sarah words test[!(sarah words test %in% sarah words)]
sarah_DF_test <- sarah_DF_test[, !colnames(sarah_DF_test) %in% remove_sarah]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',ScottHillis,'/*.txt'))
scott test = lapply(file list test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(scott_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(scott test))
```

```
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
                                                      # remove numbers
  tm map(content transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                 # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainscottcowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM scott test = DocumentTermMatrix(my documents)
DTM scott test = removeSparseTerms(DTM scott test, 0.95)
#DTM scott test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf scott test = weightTfIdf(DTM scott test)
scott_DF <- as.data.frame(as.matrix(tfidf_scott))</pre>
scott DF test <- as.data.frame(as.matrix(tfidf scott test))</pre>
scott words <- names(scott DF)</pre>
scott_words_test <- names(scott_DF_test)</pre>
remove scott <- scott words test[!(scott words test %in% scott words)]</pre>
scott_DF_test <- scott_DF_test[, !colnames(scott_DF_test) %in% remove_scott]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',TanEeLyn,'/*.txt'))
tan_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
{ strsplit(., '/', fixed=TRUE) } %>%
```

```
{ lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(tan_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(tan test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm_map(content_transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>% # remove numbers
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm_map(content_transformer(stripWhitespace))
                                                       # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraintancowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM tan_test = DocumentTermMatrix(my_documents)
DTM_tan_test = removeSparseTerms(DTM_tan_test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf tan test = weightTfIdf(DTM tan test)
tan DF <- as.data.frame(as.matrix(tfidf tan))</pre>
tan_DF_test <- as.data.frame(as.matrix(tfidf_tan_test))</pre>
tan_words <- names(tan_DF)</pre>
tan words test <- names(tan DF test)</pre>
remove tan <- tan words test[!(tan words test %in% tan words)]</pre>
tan DF test <- tan DF test[, !colnames(tan DF test) %in% remove tan]
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',TheresePoletti,'/*.txt'))
```

```
therese_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
 { strsplit(., '/', fixed=TRUE) } %>%
 { lapply(., tail, n=2) } %>%
 { lapply(., paste0, collapse = '') } %>%
 unlist
# Rename the articles
#mynames
names(therese test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(therese test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
 tm map(content transformer(tolower)) %>%
                                                       # make everything
Lowercase
 tm_map(content_transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,</pre>
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraintheresecowellnewsml
txt"))
## create a doc-term-matrix from the corpus
DTM therese test = DocumentTermMatrix(my documents)
DTM therese test = removeSparseTerms(DTM therese test, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf therese test = weightTfIdf(DTM therese test)
therese_DF <- as.data.frame(as.matrix(tfidf_therese))</pre>
therese DF test <- as.data.frame(as.matrix(tfidf therese test))</pre>
therese_words <- names(therese_DF)</pre>
therese_words_test <- names(therese_DF_test)</pre>
remove therese <- therese words test[!(therese words test %in%
therese words)]
therese_DF_test <- therese_DF_test[, !colnames(therese_DF_test) %in%</pre>
remove theresel
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',TimFarrand,'/*.txt'))
tim_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file list test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(tim_test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tim_test))
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my documents = documents raw %>%
  tm map(content transformer(tolower)) %>%
                                                        # make everything
Lowercase
  tm map(content transformer(removeNumbers)) %>% # remove numbers
 tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
 tm_map(content_transformer(stripWhitespace)) # remove excess
```

```
white-space
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraintimcowellnewsmltxt"
))
## create a doc-term-matrix from the corpus
DTM tim test = DocumentTermMatrix(my documents)
DTM tim test = removeSparseTerms(DTM tim test, 0.95)
#DTM_tim_test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tim_test = weightTfIdf(DTM_tim_test)
tim DF <- as.data.frame(as.matrix(tfidf tim))</pre>
tim_DF_test <- as.data.frame(as.matrix(tfidf_tim_test))</pre>
tim words <- names(tim DF)</pre>
tim words test <- names(tim DF test)</pre>
remove tim <- tim words test[!(tim words test %in% tim words)]</pre>
tim DF test <- tim DF test[, !colnames(tim DF test) %in% remove tim]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',ToddNissen,'/*.txt'))
todd_test = lapply(file_list_test, readerPlain)
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file list test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
```

```
#mynames
names(todd test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(todd test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
                                                        # remove numbers
  tm_map(content_transformer(removeNumbers)) %>%
  tm_map(content_transformer(removePunctuation)) %>% # remove punctuation
  tm map(content_transformer(stripWhitespace)) # remove excess
white-space
# let's just use the "basic English" stop words
my documents = tm map(my documents, content transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
\mathbf{c} ("cusersmachuonedrivedocumentsgithubstadatareuterscctraintoddcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM todd test = DocumentTermMatrix(my documents)
## Probably a bit stringent here... but only 50 docs!
DTM todd test = removeSparseTerms(DTM todd test, 0.95)
#DTM todd test # now ~ 1000 terms (versus ~3000 before)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf todd test = weightTfIdf(DTM_todd_test)
todd DF <- as.data.frame(as.matrix(tfidf todd))</pre>
todd DF test <- as.data.frame(as.matrix(tfidf todd test))</pre>
todd_words <- names(todd_DF)</pre>
todd_words_test <- names(todd_DF_test)</pre>
remove todd <- todd words test[!(todd words test %in% todd words)]
todd_DF_test <- todd_DF_test[, !colnames(todd_DF_test) %in% remove_todd]</pre>
### Next Author
file list test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',WilliamKazer,'/*.txt'))
will_test = lapply(file_list_test, readerPlain)
```

```
# The file names are ugly...
#file_list_test
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
# Rename the articles
#mynames
names(will test) = mynames
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents raw = Corpus(VectorSource(will test))
## Some pre-processing/tokenization steps.
## tm map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm map(content transformer(tolower)) %>%
                                                         # make everything
Lowercase
  tm_map(content_transformer(removeNumbers)) %>%
                                                       # remove numbers
  tm map(content transformer(removePunctuation)) %>% # remove punctuation
  tm map(content transformer(stripWhitespace))
                                                        # remove excess
white-space
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my documents <- tm map(my documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainwillcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM_will_test = DocumentTermMatrix(my_documents)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf will test = weightTfIdf(DTM will test)
will_DF <- as.data.frame(as.matrix(tfidf_will))</pre>
will_DF_test <- as.data.frame(as.matrix(tfidf_will_test))</pre>
will_words <- names(will_DF)</pre>
will words test <- names(will DF test)</pre>
remove will <- will words test[!(will words test %in% will words)]</pre>
will DF test <- will DF test[, !colnames(will DF test) %in% remove will]</pre>
```

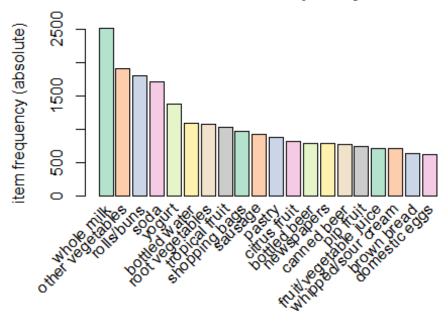
Question 6

First, I took a look at the most frequent items to ground myself in the data. It also helped in downstream analysis when analyzing the association rules of basket items. As you can see, whole milk is predominately purchased the most in the dataset of basket items.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)
baskets <- cbind(Baskets = rownames(baskets), baskets)</pre>
rownames(baskets) <- 1:nrow(baskets)</pre>
library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")</pre>
# baskets1= split(x=baskets 1[,-1], f=baskets$Baskets)
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
summary(baskettrans)
## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609146
##
## most frequent items:
         whole milk other vegetables rolls/buns
                                                                     soda
```

```
##
                2513
                                  1903
                                                   1809
                                                                      1715
##
                               (Other)
             yogurt
##
                1372
                                 34055
##
## element (itemset/transaction) length distribution:
## sizes
           2
                           5
                                           8
##
                 3
                      4
                                6
                                      7
                                                9
                                                    10
                                                          11
                                                               12
                                                                    13
                                                                               15
      1
                                                                          14
16
                         855
## 2159 1643 1299 1005
                              645
                                    545
                                         438
                                              350
                                                         182
                                                              117
                                                                    78
                                                                          77
                                                                               55
                                                   246
46
                     20
                                          24
                                               26
                                                     27
                                                               29
##
     17
          18
                19
                          21
                               22
                                     23
                                                          28
                                                                     32
                          11
##
     29
          14
               14
                      9
                                4
                                      6
                                           1
                                                1
                                                      1
                                                           1
                                                                3
                                                                     1
##
##
      Min. 1st Qu. Median
                               Mean 3rd Qu.
                                                Max.
##
     1.000
             2.000
                      3.000
                              4.409
                                       6.000
                                              32.000
##
## includes extended item information - examples:
##
                labels
## 1 abrasive cleaner
## 2 artif. sweetener
       baby cosmetics
## 3
##
## includes extended transaction information - examples:
     transactionID
## 1
                  1
                 10
## 2
## 3
                100
require("RColorBrewer")
itemFrequencyPlot(baskettrans,topN=20,type="absolute",col=brewer.pal(8,'Paste
12'), main="Absolute Item Frequency Plot")
```

Absolute Item Frequency Plot



Prior to running the apriori algorithm, I wanted some way to determine which thresholds I should as parameters in the algorithm. I figured out away to plot the number of rules by confidence level at different support levels. Based on the plot, I determine that the thresholds should be .005 and .2 as this would provide me with enough rules to analyze. In addition, I also found that that the lift is negatively effected as you increase the support and confidence level.

```
library(dplyr)
library(arules)
library(reshape)
library(didyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)
library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)</pre>
```

```
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
summary(baskettrans)
## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609146
##
## most frequent items:
         whole milk other vegetables
##
                                             rolls/buns
                                                                      soda
##
                2513
                                  1903
                                                    1809
                                                                      1715
##
             yogurt
                               (Other)
##
               1372
                                 34055
##
## element (itemset/transaction) length distribution:
## sizes
##
           2
                 3
                      4
                           5
                                      7
                                           8
                                                     10
      1
                                6
                                                 9
                                                          11
                                                                12
                                                                     13
                                                                          14
                                                                                15
16
## 2159 1643 1299 1005
                         855
                              645
                                    545
                                         438
                                              350
                                                    246
                                                         182
                                                              117
                                                                     78
                                                                          77
                                                                               55
46
##
     17
          18
                19
                     20
                          21
                               22
                                     23
                                          24
                                                26
                                                     27
                                                          28
                                                                29
                                                                     32
##
     29
          14
               14
                      9
                          11
                                4
                                      6
                                           1
                                                 1
                                                      1
                                                           1
                                                                 3
                                                                      1
##
##
      Min. 1st Ou. Median
                               Mean 3rd Ou.
                                                 Max.
##
     1.000
             2.000
                      3.000
                              4.409
                                       6.000
                                              32.000
##
## includes extended item information - examples:
##
                labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3
       baby cosmetics
##
## includes extended transaction information - examples:
##
     transactionID
## 1
                  1
## 2
                 10
## 3
                100
```

```
supportLevels <- c(0.1, 0.05, 0.01, 0.005)
confidenceLevels \leftarrow c(0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1)
# Empty integers
rules_sup10 <- integer(length=9)</pre>
rules sup5 <- integer(length=9)</pre>
rules sup1 <- integer(length=9)</pre>
rules_sup0.5 <- integer(length=9)</pre>
# Apriori algorithm with a support level of 10%
for (i in 1:length(confidenceLevels)) {
  rules sup10[i] <- length(apriori(baskettrans,</pre>
parameter=list(sup=supportLevels[1],
conf=confidenceLevels[i], target="rules")))
}
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
##
           0.9
                  0.1
                                                  TRUE
                                                              5
                                                                    0.1
##
  maxlen target ext
        10 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                          TRUE
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
                                                  TRUE
##
           0.8
                  0.1
                                                                    0.1
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
```

```
0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 983
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
           0.7
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
## maxlen target ext
        10 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
## Absolute minimum support count: 983
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
                                                 TRUE
                                                            5
##
           0.6
                  0.1
                                                                  0.1
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
```

```
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
           0.5
                 0.1
                        1 none FALSE
                                                TRUE
                                                           5
                                                                 0.1
  maxlen target ext
##
        10
           rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 983
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                        1 none FALSE
                                                TRUE
                                                           5
##
           0.4
                  0.1
                                                                  0.1
## maxlen target ext
##
        10 rules TRUE
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
          0.3 0.1 1 none FALSE TRUE
```

```
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 983
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.2
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                  0.1
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 983
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [1 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                  0.1
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                         TRUE
## Absolute minimum support count: 983
```

```
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [8 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
# Apriori algorithm with a support level of 5%
for (i in 1:length(confidenceLevels)){
  rules sup5[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[2],
conf=confidenceLevels[i], target="rules")))
}
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
                         1 none FALSE
                                                 TRUE
                                                                 0.05
           0.9
                  0.1
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.8
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                 0.05
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
```

```
0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 491
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
           0.7
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                 0.05
## maxlen target ext
        10 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
## Absolute minimum support count: 491
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
                                                 TRUE
                                                            5
##
           0.6
                  0.1
                                                                 0.05
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
```

```
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
           0.5
                  0.1
                        1 none FALSE
                                                TRUE
                                                           5
                                                                0.05
## maxlen target ext
##
        10
           rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 491
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                        1 none FALSE
                                                TRUE
                                                           5
##
           0.4
                  0.1
                                                                0.05
## maxlen target ext
##
        10 rules TRUE
## Algorithmic control:
## filter tree heap memopt load sort verbose
      0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [1 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
          0.3 0.1 1 none FALSE TRUE
```

```
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 491
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [3 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.2
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                 0.05
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 491
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [7 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                 0.05
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                         TRUE
## Absolute minimum support count: 491
```

```
##
## set item appearances \dots[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [14 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
for (i in 1:length(confidenceLevels)){
  rules_sup0.5[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[4],
conf=confidenceLevels[i], target="rules")))
}
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                                                                0.005
##
           0.9
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
##
   maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.8
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                0.005
##
  maxlen target ext
##
        10 rules TRUE
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE 2
```

```
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.7
                  0.1
                         1 none FALSE
                                                 TRUE
                                                             5
                                                                0.005
##
  maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [1 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
                  0.1
                         1 none FALSE
                                                 TRUE
                                                                 0.005
           0.6
                                                                            1
##
  maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
```

```
## writing ... [22 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
  confidence minval smax arem aval originalSupport maxtime support minlen
##
                  0.1
                         1 none FALSE
                                                 TRUE
                                                                0.005
                                                            5
  maxlen target ext
##
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [120 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
           0.4
                  0.1
                         1 none FALSE
                                                                0.005
##
                                                 TRUE
                                                            5
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                         TRUE
##
## Absolute minimum support count: 49
##
## set item appearances ... [0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [270 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.3
                  0.1
                         1 none FALSE
                                                 TRUE
                                                                0.005
                                                                            1
## maxlen target ext
```

```
10 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [482 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
  confidence minval smax arem aval original Support maxtime support minlen
##
           0.2
                  0.1
                         1 none FALSE
                                                 TRUE
                                                             5
                                                                 0.005
## maxlen target ext
        10 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [873 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.1
                  0.1
                         1 none FALSE
                                                 TRUE
                                                                 0.005
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 49
```

```
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [1582 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
# Apriori algorithm with a support level of 1%
for (i in 1:length(confidenceLevels)){
  rules sup1[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[3],
conf=confidenceLevels[i], target="rules")))
}
## Apriori
## Parameter specification:
   confidence minval smax arem aval original Support maxtime support minlen
##
           0.9
                  0.1
                         1 none FALSE
                                                 TRUE
                                                             5
                                                                  0.01
##
  maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                         TRUE
##
## Absolute minimum support count: 98
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.8
                  0.1
                         1 none FALSE
                                                 TRUE
                                                                  0.01
##
    maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
```

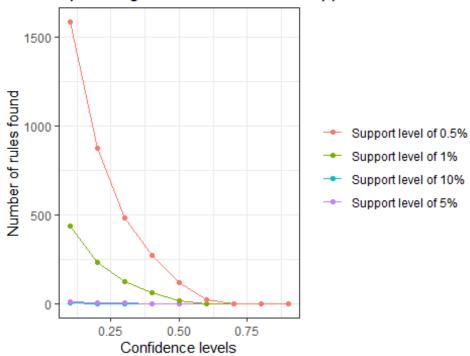
```
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.6
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                 0.01
##
  maxlen target ext
        10 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
```

```
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
                                                 TRUE
                                                            5
##
           0.5
                  0.1
                                                                 0.01
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
                                                 TRUE
                                                                 0.01
##
           0.4
                  0.1
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                         TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [62 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
           0.3
                  0.1
                                                 TRUE
                                                            5
                                                                 0.01
## maxlen target ext
## 10 rules TRUE
```

```
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
## Absolute minimum support count: 98
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.2
                  0.1
                         1 none FALSE
                                                 TRUE
                                                            5
                                                                 0.01
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [232 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                                                 TRUE
##
           0.1
                  0.1
                         1 none FALSE
                                                                 0.01
## maxlen target ext
##
        10 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                    2
                                         TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
```

```
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [435 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
# Data frame
num rules <- data.frame(rules sup10, rules sup5, rules sup1, rules sup0.5,
confidenceLevels)
library(ggplot2)
# Number of rules found with a support level of 10%, 5%, 1% and 0.5%
ggplot(data=num_rules, aes(x=confidenceLevels)) +
  # Plot line and points (support level of 10%)
  geom line(aes(y=rules sup10, colour="Support level of 10%")) +
  geom_point(aes(y=rules_sup10, colour="Support level of 10%")) +
  # Plot line and points (support level of 5%)
  geom_line(aes(y=rules_sup5, colour="Support level of 5%")) +
  geom_point(aes(y=rules_sup5, colour="Support level of 5%")) +
  # Plot line and points (support level of 1%)
  geom line(aes(y=rules sup1, colour="Support level of 1%")) +
  geom point(aes(y=rules sup1, colour="Support level of 1%")) +
  # Plot line and points (support level of 0.5%)
  geom_line(aes(y=rules_sup0.5, colour="Support level of 0.5%")) +
  geom point(aes(y=rules sup0.5, colour="Support level of 0.5%")) +
  # Labs and theme
  labs(x="Confidence levels", y="Number of rules found",
       title="Apriori algorithm with different support levels") +
  theme bw() +
  theme(legend.title=element_blank())
```

Apriori algorithm with different support levels



Using the thresholds specified, I got 873 rules. This quite high, but this is the number of rules I wanted in order to find the associate rules with higher lifts.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)
baskets <- cbind(Baskets = rownames(baskets), baskets)</pre>
rownames(baskets) <- 1:nrow(baskets)</pre>
library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")</pre>
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
```

```
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                  0.1
                         1 none FALSE
                                                 TRUE
                                                                0.005
##
## maxlen target ext
         5 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                         TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [873 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
inspect(basketrules)
##
         lhs
                                       rhs
                                                                    support
confidence
                            lift count
              coverage
## [1]
                                    => {whole milk}
                                                               0.255516014
0.2555160 1.000000000 1.0000000
                                 2513
## [2] {cake bar}
                                    => {whole milk}
                                                               0.005592272
0.4230769 0.013218099 1.6557746
## [3]
       {dishes}
                                    => {other vegetables}
                                                               0.005998983
0.3410405 0.017590239 1.7625502
                                   59
## [4] {dishes}
                                    => {whole milk}
                                                               0.005287239
0.3005780 0.017590239 1.1763569
                                   52
                                    => {whole milk}
                                                               0.005185562
## [5]
        {mustard}
0.4322034 0.011997966 1.6914924
                                   51
## [6]
       {pot plants}
                                    => {whole milk}
                                                               0.006914082
0.4000000 0.017285206 1.5654596
```

## [7] {chewing gum} 0.2560386 0.021047280 1.4683033	=> 53	{soda}	0.005388917
## [8] {chewing gum}	=>	{whole milk}	0.005083884
0.2415459 0.021047280 0.9453259 ## [9] {canned fish}	50 =>	<pre>{other vegetables}</pre>	0.005083884
0.3378378 0.015048297 1.7459985	50	(other vegetubies)	0.003003004
## [10] {pasta}		<pre>{whole milk}</pre>	0.006100661
0.4054054 0.015048297 1.5866145 ## [11] {herbs}	60 =>	<pre>{root vegetables}</pre>	0.007015760
0.4312500 0.016268429 3.9564774	69	(0.007.0207.00
## [12] {herbs} 0.4750000 0.016268429 2.4548739	=> 76	{other vegetables}	0.007727504
## [13] {herbs}		{whole milk}	0.007727504
0.4750000 0.016268429 1.8589833	76		
## [14] {processed cheese} 0.3190184 0.016573462 1.8294729	=> 52	{soda}	0.005287239
## [15] {processed cheese}		{other vegetables}	0.005490595
0.3312883 0.016573462 1.7121497	54		
## [16] {processed cheese} 0.4233129 0.016573462 1.6566981	=> 69	{whole milk}	0.007015760
<pre>## [17] {semi-finished bread}</pre>		<pre>{other vegetables}</pre>	0.005185562
0.2931034 0.017691917 1.5148042	51	(uhala milk)	0 007117420
## [18] {semi-finished bread} 0.4022989 0.017691917 1.5744565	=> 70	{whole milk}	0.007117438
## [19] {beverages}		{yogurt}	0.005490595
0.2109375 0.026029487 1.5120775 ## [20] {beverages}	54	{rolls/buns}	0.005388917
0.2070312 0.026029487 1.1255679	53	(1 O113/ Dull3)	0.005388517
## [21] {beverages}		<pre>{whole milk}</pre>	0.006812405
0.2617188 0.026029487 1.0242753 ## [22] {ice cream}	67 =>	{soda}	0.006100661
0.2439024 0.025012710 1.3987058	60	(55.4)	
## [23] {ice cream}		<pre>{other vegetables}</pre>	0.005083884
0.2032520 0.025012710 1.0504381 ## [24] {ice cream}	50 =>	{whole milk}	0.005897306
0.2357724 0.025012710 0.9227303	58		
## [25] {detergent} 0.3333333 0.019217082 1.7227185	=> 63	<pre>{other vegetables}</pre>	0.006405694
## [26] {detergent}		{whole milk}	0.008947636
0.4656085 0.019217082 1.8222281	88		0.006405604
## [27] {pickled vegetables} 0.3579545 0.017895272 1.8499648	=> 63	{other vegetables}	0.006405694
<pre>## [28] {pickled vegetables}</pre>		{whole milk}	0.007117438
0.3977273 0.017895272 1.5565650 ## [29] {baking powder}	70	(other vegetables)	0.007320793
0.4137931 0.017691917 2.1385471	72	{other vegetables}	0.00/320/93
## [30] {baking powder}		<pre>{whole milk}</pre>	0.009252669
0.5229885 0.017691917 2.0467935 ## [31] {flour}	91	<pre>{other vegetables}</pre>	0.006304016
0.3625731 0.017386884 1.8738342	62	(Selici Vegetables)	2.000304010

## [32] {flour}		<pre>{whole milk}</pre>	0.008439248
0.4853801 0.017386884 1.8996074 ## [33] {soft cheese}	83 =>	{yogurt}	0.005998983
0.3511905 0.017081851 2.5174623	59	() - 5 5	
## [34] {soft cheese}		<pre>{rolls/buns}</pre>	0.005388917
0.3154762 0.017081851 1.7151511	53	(athon vagatables)	0 007117420
## [35] {soft cheese} 0.4166667 0.017081851 2.1533981	= <i>></i> 70	{other vegetables}	0.007117438
## [36] {soft cheese}		{whole milk}	0.007524148
0.4404762 0.017081851 1.7238692	74		
## [37] {specialty bar}		{soda}	0.007219115
0.2639405 0.027351296 1.5136181 ## [38] {specialty bar}	71 ->	{rolls/buns}	0.005592272
0.2044610 0.027351296 1.1115940	55	(10113/bull3)	0.003332272
## [39] {specialty bar}		<pre>{other vegetables}</pre>	0.005592272
0.2044610 0.027351296 1.0566861	55	-	
## [40] {specialty bar}		<pre>{whole milk}</pre>	0.006507372
<pre>0.2379182 0.027351296 0.9311284 ## [41] {misc. beverages}</pre>	64 ->	{soda}	0.007320793
0.2580645 0.028368073 1.4799210	72	(Soud)	0.00/320/93
## [42] {misc. beverages}		{whole milk}	0.007015760
0.2473118 0.028368073 0.9678917	69		
## [43] {grapes}		{tropical fruit}	0.006100661
0.2727273 0.022369090 2.5991015	60	(other vegetables)	0.009049314
## [44] {grapes} 0.4045455 0.022369090 2.0907538	= <i>></i> 89	{other vegetables}	0.009049314
## [45] {grapes}		{whole milk}	0.007320793
0.3272727 0.022369090 1.2808306	72	,	
## [46] {cat food}		{yogurt}	0.006202339
0.2663755 0.023284189 1.9094778	61	(-th	0.006507373
## [47] {cat food} 0.2794760 0.023284189 1.4443753	=> 64	{other vegetables}	0.006507372
## [48] {cat food}		{whole milk}	0.008845958
0.3799127 0.023284189 1.4868448	87	(01000012220
<pre>## [49] {specialty chocolate}</pre>	=>	{soda}	0.006304016
0.2073579 0.030401627 1.1891338	62		
## [50] {specialty chocolate}		<pre>{other vegetables}</pre>	0.006100661
0.2006689 0.030401627 1.0370881 ## [51] {specialty chocolate}	60 ->	{whole milk}	0.008032537
0.2642140 0.030401627 1.0340410	79	(WHOLE WILK)	0.000032337
## [52] {meat}		{sausage}	0.005287239
0.2047244 0.025826131 2.1790742	52		
## [53] {meat}		{soda}	0.005490595
0.2125984 0.025826131 1.2191869 ## [54] {meat}	54 ->	{yogurt}	0.005287239
0.2047244 0.025826131 1.4675398	52	(yogure)	0.003287239
## [55] {meat}		<pre>{rolls/buns}</pre>	0.006914082
0.2677165 0.025826131 1.4554959	68		
## [56] {meat}		<pre>{other vegetables}</pre>	0.009964413
0.3858268 0.025826131 1.9940128	98		

## [57] {meat} 0.3858268 0.025826131 1.5099906		<pre>{whole milk}</pre>	0.009964413
## [58] {frozen meals}	98 =>	{soda}	0.006202339
0.2186380 0.028368073 1.2538220	61	(55.5)	
## [59] {frozen meals}	=>	{yogurt}	0.006202339
0.2186380 0.028368073 1.5672774	61		
## [60] {frozen meals}		{other vegetables}	0.007524148
0.2652330 0.028368073 1.3707653 ## [61] {frozen meals}	74	(whole milk)	0.009862735
0.3476703 0.028368073 1.3606593	97	{whole milk}	0.009602733
## [62] {hard cheese}		{sausage}	0.005185562
0.2116183 0.024504321 2.2524519	51	0 3	
## [63] {hard cheese}	=>	<pre>{root vegetables}</pre>	0.005592272
0.2282158 0.024504321 2.0937519	55		
## [64] {hard cheese}		{yogurt}	0.006405694
0.2614108 0.024504321 1.8738886 ## [65] {hard cheese}	63 ->	{rolls/buns}	0.005897306
0.2406639 0.024504321 1.3084187	58	(1 0113/ bulls)	0.003897300
## [66] {hard cheese}		<pre>{other vegetables}</pre>	0.009456024
0.3858921 0.024504321 1.9943505	93	,	
## [67] {hard cheese}		<pre>{whole milk}</pre>	0.010066090
0.4107884 0.024504321 1.6076815	99		
## [68] {butter milk}		{yogurt}	0.008540925
0.3054545 0.027961362 2.1896104 ## [69] {butter milk}	84	{rolls/buns}	0.007625826
0.2727273 0.027961362 1.4827378	75	(1 0113/ bull3)	0.007023020
## [70] {butter milk}		<pre>{other vegetables}</pre>	0.010371124
0.3709091 0.027961362 1.9169159	102	5 ,	
## [71] {butter milk}		<pre>{whole milk}</pre>	0.011591256
0.4145455 0.027961362 1.6223854	114		
## [72] {candy}		{soda}	0.008642603
0.2891156 0.029893238 1.6579897 ## [73] {candy}	85 ->	{rolls/buns}	0.007117438
0.2380952 0.029893238 1.2944537	70	(1 0113/ bull3)	0.00/11/430
## [74] {candy}		<pre>{other vegetables}</pre>	0.006914082
0.2312925 0.029893238 1.1953557	68	5	
## [75] {candy}	=>	<pre>{whole milk}</pre>	0.008235892
0.2755102 0.029893238 1.0782502	81		
## [76] {ham}		{tropical fruit}	0.005388917
0.2070312 0.026029487 1.9730158 ## [77] {ham}	53 ->	{yogurt}	0.006710727
0.2578125 0.026029487 1.8480947	66	(yogur c)	0.000/10/2/
## [78] {ham}		{rolls/buns}	0.006914082
0.2656250 0.026029487 1.4441249	68		
## [79] {ham}	=>	<pre>{other vegetables}</pre>	0.009150991
0.3515625 0.026029487 1.8169297	90	(1 111)	0.044400==0
## [80] {ham}		{whole milk}	0.011489578
0.4414062 0.026029487 1.7275091 ## [81] {sliced cheese}	113	{sausage}	0.007015760
0.2863071 0.024504321 3.0474349	69	(sausage J	0.007010700
0.10000,1 0.01,001011 0.01,4040	3,5		

## [82] {sliced cheese} 0.2157676 0.024504321 2.0562		{tropical fruit}	0.005287239
## [83] {sliced cheese}		<pre>{root vegetables}</pre>	0.005592272
0.2282158 0.024504321 2.0937	519 55		
## [84] {sliced cheese}		{soda}	0.005083884
0.2074689 0.024504321 1.1897			
<pre>## [85] {sliced cheese}</pre>		<pre>{yogurt}</pre>	0.008032537
0.3278008 0.024504321 2.34979		(0 007635036
## [86] {sliced cheese} 0.3112033 0.024504321 1.69192		{rolls/buns}	0.007625826
## [87] {sliced cheese}		{other vegetables}	0.009049314
0.3692946 0.024504321 1.90857		(other vegetables)	0.000014
## [88] {sliced cheese}		{whole milk}	0.010777834
0.4398340 0.024504321 1.7213		(0.020///03
## [89] {UHT-milk}		{bottled water}	0.007320793
0.2188450 0.033451957 1.98007		,	
## [90] {UHT-milk}	=>	{soda}	0.007625826
0.2279635 0.033451957 1.30736	75 75		
## [91] {UHT-milk}	=>	<pre>{yogurt}</pre>	0.007422471
0.2218845 0.033451957 1.59054			
## [92] {UHT-milk}		<pre>{other vegetables}</pre>	0.008134215
0.2431611 0.033451957 1.25669			
## [93] {oil}		<pre>{root vegetables}</pre>	0.007015760
0.2500000 0.028063040 2.29363		6 44	0.000064443
## [94] {oil}		<pre>{other vegetables}</pre>	0.009964413
0.3550725 0.028063040 1.83506		(uhala milk)	0 011106112
## [95] {oil} 0.4021739 0.028063040 1.57396		{whole milk}	0.011286223
## [96] {onions}		<pre>{root vegetables}</pre>	0.009456024
0.3049180 0.031011693 2.7974		(1000 Vegetables)	0.005450024
## [97] {onions}		{yogurt}	0.007219115
0.2327869 0.031011693 1.66870		() -8)	0,00,11,11
## [98] {onions}		<pre>{rolls/buns}</pre>	0.006812405
0.2196721 0.031011693 1.19429			
## [99] {onions}	=>	<pre>{other vegetables}</pre>	0.014234875
0.4590164 0.031011693 2.37226	581 140		
## [100] {onions}	=>	{whole milk}	0.012099644
0.3901639 0.031011693 1.52696			
## [101] {berries}		{whipped/sour cream}	0.009049314
0.2721713 0.033248602 3.79688			
## [102] {berries}		{tropical fruit}	0.006710727
0.2018349 0.033248602 1.92349		(4-)	0.007330703
## [103] {berries} 0.2201835 0.033248602 1.26268		{soda}	0.007320793
## [104] {berries}		(vogunt)	0.010574479
0.3180428 0.033248602 2.27984		{yogurt}	0.0103/44/9
## [105] {berries}		{other vegetables}	0.010269446
0.3088685 0.033248602 1.59628		(Jene, Vegetables)	0.010207770
## [106] {berries}		{whole milk}	0.011794611
0.3547401 0.033248602 1.38832		,	<u></u>

<pre>## [107] {hamburger meat} 0.2599388 0.033248602 1.413210</pre>		<pre>{rolls/buns}</pre>	0.008642603
## [108] {hamburger meat}		{other vegetables}	0.013828165
0.4159021 0.033248602 2.14944	70 136	-	
## [109] {hamburger meat}		<pre>{whole milk}</pre>	0.014743264
0.4434251 0.033248602 1.735410		{tropical fruit}	0.006710727
0.2037037 0.032943569 1.94130		(cropical fruit)	0.000/10/2/
<pre>## [111] {hygiene articles}</pre>		{soda}	0.007015760
0.2129630 0.032943569 1.22127			
<pre>## [112] {hygiene articles} 0.2222222 0.032943569 1.592976</pre>		{yogurt}	0.007320793
## [113] {hygiene articles}		{other vegetables}	0.009557702
0.2901235 0.032943569 1.49940		(other vegetables)	0.005557702
<pre>## [114] {hygiene articles}</pre>		<pre>{whole milk}</pre>	0.012811388
0.3888889 0.032943569 1.52197			
## [115] {salty snack}		{soda}	0.009354347
0.2473118 0.037824098 1.41825 ## [116] {salty snack}		{other vegetables}	0.010777834
0.2849462 0.037824098 1.47264		(Other Vegetables)	0.010///054
## [117] {salty snack}		<pre>{whole milk}</pre>	0.011184545
0.2956989 0.037824098 1.15726			
## [118] {sugar}		{soda}	0.007320793
0.2162162 0.033858668 1.23993 ## [119] {sugar}		{yogurt}	0.006914082
0.2042042 0.033858668 1.46381		(yogur e)	0.000914002
## [120] {sugar}		<pre>{rolls/buns}</pre>	0.007015760
0.2072072 0.033858668 1.12652			
## [121] {sugar}		<pre>{other vegetables}</pre>	0.010777834
0.3183183 0.033858668 1.64511 ## [122] {sugar}		{whole milk}	0.015048297
0.4444444 0.033858668 1.73939		(WHOLE WILLK)	0.013010237
## [123] {waffles}	=>	{soda}	0.009557702
0.2486772 0.038434164 1.42608			
## [124] {waffles} 0.2380952 0.038434164 1.29445		{rolls/buns}	0.009150991
## [125] {waffles}		{other vegetables}	0.010066090
0.2619048 0.038434164 1.35356		(other vegetables)	0.01000000
## [126] {waffles}		<pre>{whole milk}</pre>	0.012709710
0.3306878 0.038434164 1.29419			0 007407004
## [127] {long life bakery pro 0.2038043 0.037417387 1.16875	•	{soda}	0.007625826
## [128] {long life bakery pr		{vogurt}	0.008744281
0.2336957 0.037417387 1.67521	-	() -81	
## [129] {long life bakery pr	-	<pre>{rolls/buns}</pre>	0.007930859
0.2119565 0.037417387 1.15234		(athan wast-172	0.010676157
## [130] {long life bakery pro 0.2853261 0.037417387 1.47460	-	{other vegetables}	0.010676157
## [131] {long life bakery pro		{whole milk}	0.013523132
0.3614130 0.037417387 1.41444	-	,	

## [132] {dessert}		{soda}	0.009862735
0.2657534 0.037112354 1.5240145 ## [133] {dessert}	97 =>	{yogurt}	0.009862735
0.2657534 0.037112354 1.9050182	97	,	
## [134] {dessert}	=>	<pre>{other vegetables}</pre>	0.011591256
0.3123288 0.037112354 1.6141636	114		
## [135] {dessert}	=>	{whole milk}	0.013726487
0.3698630 0.037112354 1.4475140	135		
## [136] {cream cheese}		{yogurt}	0.012404677
0.3128205 0.039654296 2.2424123	122	6 77 ()	
## [137] {cream cheese}		<pre>{rolls/buns}</pre>	0.009964413
0.2512821 0.039654296 1.3661465	98	(athoratable)	0.012726407
## [138] {cream cheese}		<pre>{other vegetables}</pre>	0.013726487
0.3461538 0.039654296 1.7889769	135	(whole milk)	0.016471784
## [139] {cream cheese} 0.4153846 0.039654296 1.6256696	162	{whole milk}	0.0104/1/64
## [140] {chicken}		<pre>{root vegetables}</pre>	0.010879512
0.2535545 0.042907982 2.3262206	107	(1000 vegetables)	0.0100/5512
## [141] {chicken}		{rolls/buns}	0.009659380
0.2251185 0.042907982 1.2239029	95	(. 0113, 04.13)	0.003033300
## [142] {chicken}		<pre>{other vegetables}</pre>	0.017895272
0.4170616 0.042907982 2.1554393	176	,	
## [143] {chicken}	=>	<pre>{whole milk}</pre>	0.017590239
0.4099526 0.042907982 1.6044106	173	-	
## [144] {white bread}	=>	{tropical fruit}	0.008744281
0.2077295 0.042094560 1.9796699	86		
## [145] {white bread}		{soda}	0.010269446
0.2439614 0.042094560 1.3990437	101		
## [146] {white bread}		{yogurt}	0.009049314
0.2149758 0.042094560 1.5410258	89	(ath a a constable land)	0.042726407
## [147] {white bread}		<pre>{other vegetables}</pre>	0.013726487
0.3260870 0.042094560 1.6852681 ## [148] {white bread}	135	(whole milk)	0.017081851
0.4057971 0.042094560 1.5881474	168	{whole milk}	0.01/001031
## [149] {chocolate}		{soda}	0.013523132
0.2725410 0.049618709 1.5629391	133	(Soud)	0.013323132
## [150] {chocolate}		{rolls/buns}	0.011794611
0.2377049 0.049618709 1.2923316	116	(. 0115, 005)	***************************************
## [151] {chocolate}		<pre>{other vegetables}</pre>	0.012709710
0.2561475 0.049618709 1.3238103	125		
## [152] {chocolate}	=>	<pre>{whole milk}</pre>	0.016675140
0.3360656 0.049618709 1.3152427	164		
## [153] {coffee}	=>	<pre>{other vegetables}</pre>	0.013421454
0.2311734 0.058057956 1.1947400	132		
## [154] {coffee}	=>	{whole milk}	0.018708693
0.3222417 0.058057956 1.2611408	184		
## [155] {frozen vegetables}		<pre>{root vegetables}</pre>	0.011591256
0.2410148 0.048093543 2.2111759	114	(0.012404677
## [156] {frozen vegetables}		{yogurt}	0.012404677
0.2579281 0.048093543 1.8489235	122		

## [157] {frozen vegetables} 0.2114165 0.048093543 1.1494092	=> 100	<pre>{rolls/buns}</pre>	0.010167768
<pre>## [158] {frozen vegetables}</pre>	=>	{other vegetables}	0.017793594
0.3699789 0.048093543 1.9121083	175	()	
## [159] {frozen vegetables} 0.4249471 0.048093543 1.6630940	=> 201	{whole milk}	0.020437214
## [160] {beef}		<pre>{root vegetables}</pre>	0.017386884
0.3313953 0.052465684 3.0403668	171	(. oot vegetuoies)	0.02730000.
## [161] {beef}	=>	{yogurt}	0.011692933
0.2228682 0.052465684 1.5976012	115		
## [162] {beef}		<pre>{rolls/buns}</pre>	0.013624809
0.2596899 0.052465684 1.4118576	134	(athanaatablaa)	0.010725470
## [163] {beef} 0.3759690 0.052465684 1.9430662	=> 194	<pre>{other vegetables}</pre>	0.019725470
## [164] {beef}		{whole milk}	0.021250635
0.4050388 0.052465684 1.5851795	209	(WHOLE WILK)	0.021230033
## [165] {curd}		<pre>{root vegetables}</pre>	0.010879512
0.2041985 0.053279105 1.8734067	107	,	
## [166] {curd}	=>	{yogurt}	0.017285206
0.3244275 0.053279105 2.3256154	170		
## [167] {curd}		<pre>{other vegetables}</pre>	0.017183528
0.3225191 0.053279105 1.6668288	169	()	0.00640464
## [168] {curd}		<pre>{whole milk}</pre>	0.026131164
0.4904580 0.053279105 1.9194805 ## [169] {napkins}	257	{soda}	0.011997966
0.2291262 0.052364006 1.3139687	118	(Soua)	0.011997900
## [170] {napkins}		{yogurt}	0.012302999
0.2349515 0.052364006 1.6842183	121	() - 8 1	
## [171] {napkins}	=>	<pre>{rolls/buns}</pre>	0.011692933
0.2233010 0.052364006 1.2140216	115		
## [172] {napkins}		<pre>{other vegetables}</pre>	0.014438231
0.2757282 0.052364006 1.4250060	142	(.d1	0.040725470
## [173] {napkins} 0.3766990 0.052364006 1.4742678		{whole milk}	0.019725470
## [174] {pork}	194 =>	<pre>{root vegetables}</pre>	0 013624809
0.2363316 0.057651246 2.1682099	134	(Toot vegetables)	0.013024003
## [175] {pork}		{soda}	0.011896289
0.2063492 0.057651246 1.1833495	117		
## [176] {pork}	=>	<pre>{other vegetables}</pre>	0.021657346
0.3756614 0.057651246 1.9414764	213		
## [177] {pork}		{whole milk}	0.022165735
0.3844797 0.057651246 1.5047187	218	(0.040247002
## [178] {frankfurter} 0.3258621 0.058973055 1.7716161	=> 189	{rolls/buns}	0.019217082
## [179] {frankfurter}		{other vegetables}	0.016471784
0.2793103 0.058973055 1.4435193	162	(other vegetables)	0.010471704
## [180] {frankfurter}	-	{whole milk}	0.020538892
0.3482759 0.058973055 1.3630295	202	,	
## [181] {bottled beer}		{soda}	0.016980173
0.2108586 0.080528724 1.2092094	167		

## [182] {bottled beer}		<pre>{other vegetables}</pre>	0.016166751
0.2007576 0.080528724 1.0375464 ## [183] {bottled beer}	159 =>	{whole milk}	0.020437214
0.2537879 0.080528724 0.9932367	201	(
## [184] {brown bread}		<pre>{yogurt}</pre>	0.014539908
0.2241379	143	(ath an ath la a)	0.010700603
## [185] {brown bread} 0.2884013 0.064870361 1.4905025	=> 184	{other vegetables}	0.018708693
## [186] {brown bread}		{whole milk}	0.025216065
0.3887147 0.064870361 1.5212930	248	,	
## [187] {margarine}		{yogurt}	0.014234875
0.2430556 0.058566345 1.7423115	140	(nolle/hune)	0 014742264
## [188] {margarine} 0.2517361 0.058566345 1.3686151	=> 145	{rolls/buns}	0.014743264
## [189] {margarine}		<pre>{other vegetables}</pre>	0.019725470
0.3368056 0.058566345 1.7406635	194	,	
## [190] {margarine}		<pre>{whole milk}</pre>	0.024199288
0.4131944 0.058566345 1.6170980	238	(mant wasatablas)	0.012012066
## [191] {butter} 0.2330275 0.055414337 2.1378971	=> 127	<pre>{root vegetables}</pre>	0.012913066
## [192] {butter}		{yogurt}	0.014641586
0.2642202 0.055414337 1.8940273	144		
## [193] {butter}		<pre>{rolls/buns}</pre>	0.013421454
0.2422018 0.055414337 1.3167800	132	(ath an ath la a)	0.00000000
## [194] {butter} 0.3614679 0.055414337 1.8681223	=> 197	{other vegetables}	0.020030503
## [195] {butter}		{whole milk}	0.027554652
0.4972477 0.055414337 1.9460530	271	(
## [196] {newspapers}		<pre>{rolls/buns}</pre>	0.019725470
0.2471338 0.079816980 1.3435934	194	(ath atab 1)	0.010310760
## [197] {newspapers} 0.2420382 0.079816980 1.2508912	=> 190	{other vegetables}	0.019318760
## [198] {newspapers}		{whole milk}	0.027351296
0.3426752 0.079816980 1.3411103	269	(
## [199] {domestic eggs}	=>	<pre>{root vegetables}</pre>	0.014336553
0.2259615 0.063446873 2.0730706	141		0.044004550
## [200] {domestic eggs} 0.2259615 0.063446873 1.6197753	=> 141	{yogurt}	0.014336553
## [201] {domestic eggs}		{rolls/buns}	0.015658363
0.2467949 0.063446873 1.3417510	154	(. 00, 000)	0.00=000000
## [202] {domestic eggs}	=>	<pre>{other vegetables}</pre>	0.022267412
0.3509615 0.063446873 1.8138238	219		
## [203] {domestic eggs} 0.4727564 0.063446873 1.8502027	=> 295	{whole milk}	0.029994916
## [204] {fruit/vegetable juice}		{soda}	0.018403660
0.2545710 0.072292832 1.4598869	181	(2000)	3.023 103000
<pre>## [205] {fruit/vegetable juice}</pre>		<pre>{yogurt}</pre>	0.018708693
0.2587904 0.072292832 1.8551049	184	6 77 W	
## [206] {fruit/vegetable juice}		<pre>{rolls/buns}</pre>	0.014539908
0.2011252 0.072292832 1.0934583	143		

## [207] {fruit/vegetable juice} 0.2911392 0.072292832 1.5046529	=> 207	<pre>{other vegetables}</pre>	0.021047280
## [208] {fruit/vegetable juice}		{whole milk}	0.026639553
0.3684951 0.072292832 1.4421604	262		
## [209] {whipped/sour cream}		<pre>{root vegetables}</pre>	0.017081851
0.2382979 0.071682766 2.1862496	168		
## [210] {whipped/sour cream}		{yogurt}	0.020742247
0.2893617 0.071682766 2.0742510	204	(m. 11 - /h	0.014641506
## [211] {whipped/sour cream} 0.2042553 0.071682766 1.1104760	=> 144	<pre>{rolls/buns}</pre>	0.014641586
## [212] {whipped/sour cream}		{other vegetables}	0.028876462
0.4028369 0.071682766 2.0819237	284	(Other Vegetables)	0.020070402
## [213] {whipped/sour cream}	-	{whole milk}	0.032231825
0.4496454 0.071682766 1.7597542	317	(miere mark)	0.032232023
## [214] {pip fruit}		{tropical fruit}	0.020437214
0.2701613 0.075648195 2.5746476	201	,	
## [215] {pip fruit}	=>	<pre>{root vegetables}</pre>	0.015556685
0.2056452 0.075648195 1.8866793	153	-	
## [216] {pip fruit}	=>	<pre>{yogurt}</pre>	0.017996950
0.2379032 0.075648195 1.7053777	177		
## [217] {pip fruit}		<pre>{other vegetables}</pre>	0.026131164
0.3454301 0.075648195 1.7852365	257		
## [218] {pip fruit}		{whole milk}	0.030096594
0.3978495 0.075648195 1.5570432	296	()	0.004047000
## [219] {pastry}		{soda}	0.021047280
0.2365714 0.088967972 1.3566647	207	(nolls/buns)	0.020945602
## [220] {pastry} 0.2354286 0.088967972 1.2799558	= <i>></i> 206	{rolls/buns}	0.020945002
## [221] {pastry}		<pre>{other vegetables}</pre>	0.022572445
0.2537143 0.088967972 1.3112349	222	(Other Vegetables)	0.022372443
## [222] {pastry}		{whole milk}	0.033248602
0.3737143 0.088967972 1.4625865	327	,	
## [223] {citrus fruit}	=>	{tropical fruit}	0.019928826
0.2407862 0.082765633 2.2947022	196		
## [224] {citrus fruit}	=>	<pre>{root vegetables}</pre>	0.017691917
0.2137592 0.082765633 1.9611211	174		
## [225] {citrus fruit}	=>	<pre>{yogurt}</pre>	0.021657346
0.2616708 0.082765633 1.8757521	213		
## [226] {citrus fruit}		{rolls/buns}	0.016776817
0.2027027 0.082765633 1.1020349	165		0.00075450
## [227] {citrus fruit}		<pre>{other vegetables}</pre>	0.028876462
0.3488943	284	(hala målla)	0 020502205
## [228] {citrus fruit} 0.3685504 0.082765633 1.4423768	= <i>></i> 300	{whole milk}	0.030503305
## [229] {shopping bags}		{soda}	0.024605999
0.2497420 0.098525674 1.4321939	242	(Soua)	0.024003333
## [230] {shopping bags}		{other vegetables}	0.023182511
0.2352941 0.098525674 1.2160366	228	(
## [231] {shopping bags}	_	{whole milk}	0.024504321
0.2487100 0.098525674 0.9733637	241	-	

## [232] {sausage}		{soda}	0.024300966
0.2586580 0.093950178 1.4833245 ## [233] {sausage}	239 =>	{yogurt}	0.019623793
0.2088745 0.093950178 1.4972889	193		
## [234] {sausage}		<pre>{rolls/buns}</pre>	0.030604982
0.3257576 0.093950178 1.7710480	301	(ath an an art ab 1 a a)	0.026044506
## [235] {sausage} 0.2867965 0.093950178 1.4822091	=> 265	{other vegetables}	0.026944586
## [236] {sausage}		{whole milk}	0.029893238
0.3181818 0.093950178 1.2452520	294	(WHOLE WILK)	0.023033230
## [237] {bottled water}	=>	{soda}	0.028978139
0.2621895 0.110523640 1.5035766	285		
## [238] {bottled water}		<pre>{yogurt}</pre>	0.022979156
0.2079117 0.110523640 1.4903873	226		
## [239] {bottled water}		<pre>{rolls/buns}</pre>	0.024199288
0.2189512 0.110523640 1.1903734 ## [240] {bottled water}	238	(athan vagatables)	0 024900254
0.2244710 0.110523640 1.1601012	= <i>></i> 244	{other vegetables}	0.024809354
## [241] {bottled water}		{whole milk}	0.034367056
0.3109476 0.110523640 1.2169396	338	(WHOLE WILLY)	0.031307030
<pre>## [242] {tropical fruit}</pre>	=>	<pre>{root vegetables}</pre>	0.021047280
0.2005814 0.104931368 1.8402220	207		
<pre>## [243] {tropical fruit}</pre>	=>	{yogurt}	0.029283172
0.2790698 0.104931368 2.0004746	288		
## [244] {yogurt}		{tropical fruit}	0.029283172
0.2099125 0.139501779 2.0004746	288	(nolle/hune)	0 024605000
## [245] {tropical fruit} 0.2344961 0.104931368 1.2748863	= <i>></i> 242	{rolls/buns}	0.024605999
## [246] {tropical fruit}		{other vegetables}	0.035892222
0.3420543 0.104931368 1.7677896	353	(other vegetables)	0.033032222
<pre>## [247] {tropical fruit}</pre>	=>	{whole milk}	0.042297916
0.4031008 0.104931368 1.5775950	416		
<pre>## [248] {root vegetables}</pre>	=>	<pre>{yogurt}</pre>	0.025826131
0.2369403 0.108998475 1.6984751	254		
## [249] {root vegetables}		<pre>{rolls/buns}</pre>	0.024300966
0.2229478 0.108998475 1.2121013	239	(athan vagatables)	0 047201000
## [250] {root vegetables} 0.4347015 0.108998475 2.2466049	= <i>></i> 466	{other vegetables}	0.047381800
## [251] {other vegetables}		<pre>{root vegetables}</pre>	0.047381800
0.2448765 0.193492628 2.2466049	466	(. oot vegetables)	01017302000
<pre>## [252] {root vegetables}</pre>	=>	{whole milk}	0.048906965
0.4486940 0.108998475 1.7560310	481		
## [253] {soda}	=>	<pre>{rolls/buns}</pre>	0.038332486
0.2198251 0.174377224 1.1951242	377		
## [254] {rolls/buns}		{soda}	0.038332486
0.2084024 0.183934926 1.1951242	377	{whole milk}	0.040061007
## [255] {soda} 0.2297376 0.174377224 0.8991124	= <i>></i> 394	(MIIOTE IIITIK)	0.04000100/
## [256] {yogurt}		{rolls/buns}	0.034367056
0.2463557 0.139501779 1.3393633	338	-, ,	

## [257] {yogurt} 0.3112245 0.139501779 1.6084566	=> 427	{other vegetables}	0.043416370
<pre>## [258] {other vegetables}</pre>	=>	{yogurt}	0.043416370
0.2243826 0.193492628 1.6084566 ## [259] {yogurt}		{whole milk}	0.056024403
0.4016035 0.139501779 1.5717351 ## [260] {whole milk}	551 =>	{yogurt}	0.056024403
0.2192598 0.255516014 1.5717351	551		
## [261] {rolls/buns} 0.2316197 0.183934926 1.1970465	=> 419	{other vegetables}	0.042602949
<pre>## [262] {other vegetables}</pre>	=>	{rolls/buns}	0.042602949
0.2201787 0.193492628 1.1970465 ## [263] {rolls/buns}	419 =>	{whole milk}	0.056634469
0.3079049 0.183934926 1.2050318	557	(WHOIC MIIK)	0.030034403
## [264] {whole milk} 0.2216474 0.255516014 1.2050318	=> 557	<pre>{rolls/buns}</pre>	0.056634469
## [265] {other vegetables}		{whole milk}	0.074834774
0.3867578 0.193492628 1.5136341	736		0 074004774
## [266] {whole milk} 0.2928770 0.255516014 1.5136341	=> 736	{other vegetables}	0.074834774
## [267] {oil,			
## other vegetables} 0.5102041 0.009964413 1.9967597	=> 50	{whole milk}	0.005083884
## [268] {oil,			
## whole milk} 0.4504505 0.011286223 2.3279980	=> 50	<pre>{other vegetables}</pre>	0.005083884
## [269] {onions,	70		
## root vegetables} 0.6021505 0.009456024 3.1120076		<pre>{other vegetables}</pre>	0.005693950
## [270] {onions,	56		
## other vegetables}		<pre>{root vegetables}</pre>	0.005693950
0.4000000 0.014234875 3.6697761 ## [271] {onions,	56		
<pre>## other vegetables}</pre>		<pre>{whole milk}</pre>	0.006609049
0.4642857 0.014234875 1.8170513 ## [272] {onions,	65		
## whole milk}		<pre>{other vegetables}</pre>	0.006609049
0.5462185 0.012099644 2.8229421 ## [273] {hamburger meat,	65		
<pre>## other vegetables}</pre>	=>	{whole milk}	0.006304016
0.4558824 0.013828165 1.7841635 ## [274] {hamburger meat,	62		
## whole milk}	=>	{other vegetables}	0.006304016
0.4275862 0.014743264 2.2098320	62		
<pre>## [275] {hygiene articles, ## other vegetables}</pre>	=>	{whole milk}	0.005185562
0.5425532 0.009557702 2.1233628	51		
<pre>## [276] {hygiene articles, ## whole milk}</pre>	=>	{other vegetables}	0.005185562
0.4047619 0.012811388 2.0918725	51	,	

## [277] {other vegetables,	_\	{whole milk}	0.006304016
## sugar} 0.5849057 0.010777834 2.2891155	= <i>></i>	{whose mirk}	0.000304010
## [278] {sugar,	02		
## whole milk}	=>	{other vegetables}	0.006304016
0.4189189 0.015048297 2.1650381	62		
## [279] {long life bakery product	,		
<pre>## other vegetables}</pre>		<pre>{whole milk}</pre>	0.005693950
0.5333333 0.010676157 2.0872795	56		
## [280] {long life bakery product			
## whole milk}		<pre>{other vegetables}</pre>	0.005693950
0.4210526 0.013523132 2.1760655	56		
## [281] {cream cheese,			
## yogurt}		<pre>{other vegetables}</pre>	0.005287239
0.4262295 0.012404677 2.2028204	52		
## [282] {cream cheese,		(vecumb)	0 005307330
## other vegetables} 0.3851852 0.013726487 2.7611489		{yogurt}	0.005287239
## [283] {cream cheese,	52		
## [283] {Cream cheese,	->	{whole milk}	0.006609049
0.5327869 0.012404677 2.0851409	65	(whole milk)	0.000000040
## [284] {cream cheese,	0,5		
## whole milk}	=>	{yogurt}	0.006609049
0.4012346 0.016471784 2.8761968	65	() -8)	
## [285] {cream cheese,			
<pre>## other vegetables}</pre>	=>	<pre>{whole milk}</pre>	0.006710727
0.4888889 0.013726487 1.9133395	66		
## [286] {cream cheese,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.006710727
0.4074074 0.016471784 2.1055449	66		
## [287] {chicken,			
		<pre>{other vegetables}</pre>	0.005693950
0.5233645 0.010879512 2.7048291	56		
## [288] {chicken,			0 005400050
<pre>## other vegetables} 0.2121212 0.2122222 2.2121421</pre>		<pre>{root vegetables}</pre>	0.005693950
0.3181818 0.017895272 2.9191401	56		
<pre>## [289] {chicken, ## root vegetables}</pre>	_\	(whole milk)	0.005998983
0.5514019 0.010879512 2.1579934	= <i>></i> 59	{whole milk}	0.005996965
## [290] {chicken,	23		
## whole milk}	=>	<pre>{root vegetables}</pre>	0.005998983
0.3410405 0.017590239 3.1288554	59	(100c vegetables)	0.005550505
## [291] {chicken,			
## rolls/buns}	=>	{whole milk}	0.005287239
0.5473684 0.009659380 2.1422079	52	,	
## [292] {chicken,			
## whole milk}	=>	<pre>{rolls/buns}</pre>	0.005287239
0.3005780 0.017590239 1.6341542	52		
## [293] {chicken,			
## other vegetables}	=>	{whole milk}	0.008439248

0.4715909 0.017895272 1.8456413 ## [294] {chicken,	83	
## whole milk}	<pre>=> {other vegetables}</pre>	0.008439248
0.4797688 0.017590239 2.4795197	83	
## [295] {other vegetables,		
## white bread}	<pre>=> {whole milk}</pre>	0.005897306
0.4296296 0.013726487 1.6814196	58	
## [296] {white bread,		
## whole milk}	=> {other vegetables}	0.005897306
0.3452381 0.017081851 1.7842442	58	
## [297] {chocolate, ## soda}	=> {whole milk}	0.005083884
0.3759398 0.013523132 1.4712966	50	0.005005004
## [298] {chocolate,	30	
## whole milk}	=> {soda}	0.005083884
0.3048780 0.016675140 1.7483823	50	
## [299] {chocolate,		
<pre>## other vegetables}</pre>	<pre>=> {whole milk}</pre>	0.005490595
0.4320000 0.012709710 1.6906964	54	
## [300] {chocolate,	. (-41	0.005400505
## whole milk} 0.3292683 0.016675140 1.7017098	<pre>=> {other vegetables}</pre>	0.005490595
## [301] {coffee,	54	
## yogurt}	=> {whole milk}	0.005083884
0.5208333 0.009761057 2.0383589	50	0.003003001
## [302] {coffee,		
## whole milk}	=> {yogurt}	0.005083884
0.2717391 0.018708693 1.9479259	50	
## [303] {coffee,		
<pre>## other vegetables}</pre>		0.006405694
0.4772727 0.013421454 1.8678779	63	
<pre>## [304] {coffee, ## whole milk}</pre>	<pre>=> {other vegetables}</pre>	0 006105601
0.3423913 0.018708693 1.7695315	63	0.000403034
## [305] {frozen vegetables,	03	
## root vegetables}	<pre>=> {other vegetables}</pre>	0.006100661
0.5263158 0.011591256 2.7200819	60	
## [306] {frozen vegetables,		
<pre>## other vegetables}</pre>	<pre>=> {root vegetables}</pre>	0.006100661
0.3428571 0.017793594 3.1455224	60	
## [307] {frozen vegetables,		
## root vegetables}	=> {whole milk}	0.006202339
0.5350877 0.011591256 2.0941455	61	
<pre>## [308] {frozen vegetables, ## whole milk}</pre>	<pre>=> {root vegetables}</pre>	0.006202339
0.3034826 0.020437214 2.7842829	61	0.000202333
## [309] {frozen vegetables,	5 -	
## yogurt}	<pre>=> {other vegetables}</pre>	0.005287239
0.4262295 0.012404677 2.2028204	52	
## [310] {frozen vegetables,		

## other vegetables} 0.2971429 0.017793594 2.1300292	=> 52	{yogurt}	0.005287239
<pre>## [311] {frozen vegetables, ## yogurt}</pre>	=>	{whole milk}	0.006100661
0.4918033 0.012404677 1.9247454 ## [312] {frozen vegetables,	60		
## whole milk} 0.2985075 0.020437214 2.1398111	=> 60	{yogurt}	0.006100661
## [313] {frozen vegetables,		(0.005003004
## rolls/buns} 0.5000000 0.010167768 1.9568245	=> 50	{whole milk}	0.005083884
<pre>## [314] {frozen vegetables, ## whole milk}</pre>	=>	{rolls/buns}	0.005083884
0.2487562 0.020437214 1.3524143 ## [315] {frozen vegetables,	50	,	
## other vegetables} 0.5428571 0.017793594 2.1245523	=> 95	{whole milk}	0.009659380
## [316] {frozen vegetables,	_		
## whole milk} 0.4726368 0.020437214 2.4426606	=> 95	{other vegetables}	0.009659380
<pre>## [317] {beef, ## root vegetables}</pre>	=>	<pre>{other vegetables}</pre>	0.007930859
0.4561404 0.017386884 2.3574043 ## [318] {beef,	78	,	
## other vegetables}		<pre>{root vegetables}</pre>	0.007930859
0.4020619 0.019725470 3.6886925 ## [319] {beef,	78		
## root vegetables} 0.4619883 0.017386884 1.8080601	=> 79	{whole milk}	0.008032537
<pre>## [320] {beef, ## whole milk}</pre>	=>	<pre>{root vegetables}</pre>	0.008032537
0.3779904 0.021250635 3.4678506 ## [321] {beef,	79	(
## yogurt}		<pre>{other vegetables}</pre>	0.005185562
0.4434783 0.011692933 2.2919646 ## [322] {beef,	51		
## other vegetables} 0.2628866 0.019725470 1.8844677	=> 51	{yogurt}	0.005185562
## [323] {beef, ## yogurt}	=>	{whole milk}	0.006100661
0.5217391 0.011692933 2.0419038	60	(0.00010001
## [324] {beef, ## whole milk}		{yogurt}	0.006100661
0.2870813 0.021250635 2.0579045 ## [325] {beef,	60		
## rolls/buns} 0.4253731 0.013624809 2.1983945	=> 57	<pre>{other vegetables}</pre>	0.005795628
<pre>## [326] {beef, ## other vegetables}</pre>	= \	{rolls/buns}	0.005795628
0.2938144 0.019725470 1.5973825		[1 0113/ 04113]	3.005/35020

	buns}		{whole milk}	0.006812405
## [328] {beef,	24809 1.9568245			
## whole 0.3205742 0.0212	milk} 50635 1.7428673	=> 67	{rolls/buns}	0.006812405
		=>	{whole milk}	0.009252669
## [330] {beef,	25470 1.8357838	91		
## whole 0.4354067 0.0212		=> 91	<pre>{other vegetables}</pre>	0.009252669
## [331] {curd, ## whippe	d/sour cream}		{whole milk}	0.005897306
## [332] {curd,	72801 2.2038024	58		
## whole 0.2256809 0.0261	_	=> 58	{whipped/sour cream}	0.005897306
•	al fruit}		{yogurt}	0.005287239
## [334] {curd,	69446 3.6906446	52		
## yogurt 0.3058824 0.0172		=> 52	{tropical fruit}	0.00528/239
	al fruit} 69446 2.6608326	=> 52	{other vegetables}	0.005287239
## [336] {curd,			{tropical fruit}	0.005287239
0.3076923 0.0171 ## [337] {curd,		52	(cropical fruit)	0.003287239
## tropic	al fruit} 69446 2.4799360		{whole milk}	0.006507372
## [338] {curd, ## whole			{tropical fruit}	0.006507372
0.2490272 0.0261 ## [339] {curd,		64	(c. opical ale)	0.000307372
## root v 0.5046729 0.0108		=> 54	{other vegetables}	0.005490595
## [340] {curd, ## other		=>	<pre>{root vegetables}</pre>	0.005490595
0.3195266 0.0171 ## [341] {curd,	83528 2.9314780	54		
## root v 0.5700935 0.0108	egetables} 79512 2.2311457	=> 61	{whole milk}	0.006202339
## [342] {curd, ## whole	milk}	=>	<pre>{root vegetables}</pre>	0.006202339
0.2373541 0.0261 ## [343] {curd,	31164 2.1775909	61		
## yogurt	}	=>	<pre>{other vegetables}</pre>	0.006100661

0.3529412 0.017285206 1.8240549 ## [344] {curd,	60		
## other vegetables}	=>	{yogurt}	0.006100661
0.3550296 0.017183528 2.5449825	60		
## [345] {curd,			
## yogurt}	=>	<pre>{whole milk}</pre>	0.010066090
0.5823529 0.017285206 2.2791250	99		
## [346] {curd,			
## whole milk}	=>	<pre>{yogurt}</pre>	0.010066090
0.3852140 0.026131164 2.7613555	99		
## [347] {curd,			
## rolls/buns}		{whole milk}	0.005897306
0.5858586 0.010066090 2.2928449	58		
## [348] {curd,		(1 1 - / l)	0.005007306
## whole milk}		{rolls/buns}	0.005897306
0.2256809 0.026131164 1.2269607	58		
## [349] {curd,	_\	(whole milk)	0.009862735
## other vegetables} 0.5739645 0.017183528 2.2462956	97	{whole milk}	0.009002733
## [350] {curd,	31		
## whole milk}	=>	{other vegetables}	0.009862735
0.3774319 0.026131164 1.9506268	97	(other vegetables)	0.003002733
## [351] {napkins,	,		
## yogurt}	=>	{whole milk}	0.006100661
0.4958678 0.012302999 1.9406524	60	,	
## [352] {napkins,			
## whole milk}	=>	{yogurt}	0.006100661
0.3092784 0.019725470 2.2170208	60		
## [353] {napkins,			
## rolls/buns}		{whole milk}	0.005287239
0.4521739 0.011692933 1.7696500	52		
## [354] {napkins,			
<pre>## whole milk}</pre>		{rolls/buns}	0.005287239
0.2680412 0.019725470 1.4572612	52		
## [355] {napkins,		(0.000012405
## other vegetables}		{whole wilk}	0.006812405
0.4718310 0.014438231 1.8465809	67		
<pre>## [356] {napkins, ## whole milk}</pre>	_\	(othon vogotables)	0.006812405
0.3453608 0.019725470 1.7848785	= > 67	{other vegetables}	0.000012405
## [357] {pork,	07		
## root vegetables}	=>	{other vegetables}	0.007015760
0.5149254 0.013624809 2.6612144	69	(other vegetables)	0.007013700
## [358] {other vegetables,	0,5		
## pork}	=>	<pre>{root vegetables}</pre>	0.007015760
0.3239437 0.021657346 2.9720018	69	(1.00.025.00
## [359] {pork,			
## root vegetables}	=>	{whole milk}	0.006812405
0.5000000 0.013624809 1.9568245	67	-	
## [360] {pork,			

## whole milk} 0.3073394 0.022165735 2.8196674 ## [361] {pork,	<pre>=> {root vegetables} 67</pre>	0.006812405
## rolls/buns} 0.4954955 0.011286223 2.5607978	<pre>=> {other vegetables} 55</pre>	0.005592272
## [362] {other vegetables, ## pork} 0.2582160 0.021657346 1.4038441	<pre>=> {rolls/buns} 55</pre>	0.005592272
<pre>## [363] {pork, ## rolls/buns} 0.5495495 0.011286223 2.1507441</pre>	<pre>=> {whole milk} 61</pre>	0.006202339
<pre>## [364] {pork, ## whole milk} 0.2798165 0.022165735 1.5212799</pre>	=> {rolls/buns} 61	0.006202339
<pre>## [365] {other vegetables, ## pork} 0.4694836 0.021657346 1.8373939</pre>	=> {whole milk}	0.010167768
## [366] {pork, ## whole milk} 0.4587156 0.022165735 2.3707136	<pre>=> {other vegetables} 100</pre>	0.010167768
<pre>## [367] {frankfurter, ## tropical fruit}</pre>	=> {whole milk}	0.005185562
0.5483871 0.009456024 2.1461946 ## [368] {frankfurter, ## whole milk}	<pre>51 => {tropical fruit}</pre>	0.005185562
0.2524752 0.020538892 2.4060989 ## [369] {frankfurter, ## root vegetables}	<pre>51 => {whole milk}</pre>	0.005083884
0.5000000 0.010167768 1.9568245 ## [370] {frankfurter,	50	
## whole milk} 0.2475248 0.020538892 2.2709011 ## [371] {frankfurter,	<pre>=> {root vegetables} 50</pre>	0.005083884
## yogurt} 0.5545455 0.011184545 2.1702963 ## [372] {frankfurter,	<pre>=> {whole milk} 61</pre>	0.006202339
## whole milk} 0.3019802 0.020538892 2.1647050	=> {yogurt} 61	0.006202339
<pre>## [373] {frankfurter, ## rolls/buns} 0.2910053 0.019217082 1.5039606</pre>	<pre>=> {other vegetables} 55</pre>	0.005592272
<pre>## [374] {frankfurter, ## other vegetables} 0.3395062 0.016471784 1.8457950</pre>	=> {rolls/buns}	0.005592272
<pre>## [375] {frankfurter, ## rolls/buns} 0.3121693 0.019217082 1.2217211</pre>	=> {whole milk}	0.005998983
<pre>## [376] {frankfurter, ## whole milk}</pre>	59 => {rolls/buns}	0.005998983
0.2920792 0.020538892 1.5879486	59	

## [377] {frankfurter, ## other vegetables} 0.4629630 0.016471784 1.8118745	=> {whole milk}	0.007625826
<pre>## [378] {frankfurter, ## whole milk}</pre>	<pre>-> {other vegetables}</pre>	0.007625826
0.3712871 0.020538892 1.9188696 ## [379] {bottled beer, ## bottled water}	75 => {soda}	0.005083884
0.3225806 0.015760041 1.8499013 ## [380] {bottled beer,	50	0.005085884
## soda} 0.2994012 0.016980173 2.7089336	<pre>=> {bottled water} 50</pre>	0.005083884
## [381] {bottled beer, ## bottled water} 0.3870968 0.015760041 1.5149609	=> {whole milk}	0.006100661
<pre>## [382] {bottled beer, ## whole milk}</pre>	<pre>=> {bottled water}</pre>	0.006100661
0.2985075 0.020437214 2.7008472 ## [383] {bottled beer,	60	0.005105563
## yogurt} 0.5604396 0.009252669 2.1933637 ## [384] {bottled beer,	<pre>=> {whole milk} 51</pre>	0.005185562
## whole milk} 0.2537313 0.020437214 1.8188395	=> {yogurt} 51	0.005185562
<pre>## [385] {bottled beer, ## rolls/buns} 0.3955224 0.013624809 1.5479358</pre>	=> {whole milk}	0.005388917
## [386] {bottled beer, ## whole milk}	=> {rolls/buns}	0.005388917
0.2636816 0.020437214 1.4335591 ## [387] {bottled beer,	53	
## other vegetables} 0.4716981 0.016166751 1.8460609 ## [388] {bottled beer,	=> {whole milk} 75	0.007625826
## whole milk} 0.3731343 0.020437214 1.9284162	<pre>=> {other vegetables} 75</pre>	0.007625826
## [389] {brown bread, ## tropical fruit}	=> {whole milk}	0.005693950
0.5333333 0.010676157 2.0872795 ## [390] {brown bread, ## whole milk}	<pre>56 => {tropical fruit}</pre>	0.005693950
0.2258065 0.025216065 2.1519442 ## [391] {brown bread,	56	
## root vegetables} 0.5600000 0.010167768 2.1916435 ## [392] {brown bread,	=> {whole milk} 56	0.005693950
## whole milk} 0.2258065 0.025216065 2.0716478	<pre>=> {root vegetables} 56</pre>	0.005693950
## [393] {brown bread, ## soda}	=> {whole milk}	0.005083884

0.4032258 0.012608033 1.5780843 ## [394] {brown bread,	50		
## whole milk}	=>	{soda}	0.005083884
0.2016129 0.025216065 1.1561883	50	(Soud)	0.003003001
## [395] {brown bread,	30		
## yogurt}	=>	<pre>{other vegetables}</pre>	0.005185562
0.3566434 0.014539908 1.8431883	51	(other vegetables)	0.003103302
## [396] {brown bread,	7-		
## other vegetables}	=>	{yogurt}	0.005185562
0.2771739 0.018708693 1.9868844	51	() 564. 2)	0.003203302
## [397] {brown bread,			
## yogurt}	=>	{whole milk}	0.007117438
0.4895105 0.014539908 1.9157723	70		
## [398] {brown bread,			
## whole milk}	=>	{yogurt}	0.007117438
0.2822581 0.025216065 2.0233295	70	() -8)	
## [399] {brown bread,			
## rolls/buns}	=>	{whole milk}	0.005287239
0.4193548 0.012608033 1.6412077	52	(
## [400] {brown bread,			
## whole milk}	=>	{rolls/buns}	0.005287239
0.2096774 0.025216065 1.1399544	52	(
## [401] {brown bread,			
<pre>## other vegetables}</pre>	=>	{whole milk}	0.009354347
0.5000000 0.018708693 1.9568245	92	,	
## [402] {brown bread,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.009354347
0.3709677 0.025216065 1.9172190	92		
## [403] {domestic eggs,			
## margarine}	=>	<pre>{whole milk}</pre>	0.005185562
0.6219512 0.008337570 2.4340988	51		
## [404] {margarine,			
## whole milk}	=>	<pre>{domestic eggs}</pre>	0.005185562
0.2142857 0.024199288 3.3774038	51		
## [405] {margarine,			
<pre>## root vegetables}</pre>	=>	<pre>{other vegetables}</pre>	0.005897306
0.5321101 0.011082867 2.7500277	58		
## [406] {margarine,			
<pre>## other vegetables}</pre>	=>	<pre>{root vegetables}</pre>	0.005897306
0.2989691 0.019725470 2.7428739	58		
## [407] {margarine,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.005693950
0.4000000 0.014234875 2.0672622	56		
## [408] {margarine,			
<pre>## other vegetables}</pre>	=>	<pre>{yogurt}</pre>	0.005693950
0.2886598 0.019725470 2.0692194	56		
## [409] {margarine,			
## yogurt}	=>	<pre>{whole milk}</pre>	0.007015760
0.4928571 0.014234875 1.9288699	69		
## [410] {margarine,			

<pre>## whole milk} 0.2899160 0.024199288 2.0782241 ## [411] {margarine,</pre>	=> 69	{yogurt}	0.007015760
## rolls/buns} 0.3517241 0.014743264 1.8177651	=> 51	<pre>{other vegetables}</pre>	0.005185562
## [412] {margarine, ## other vegetables} 0.2628866 0.019725470 1.4292370	=> 51	{rolls/buns}	0.005185562
## [413] {margarine, ## rolls/buns} 0.5379310 0.014743264 2.1052733	=> 78	{whole milk}	0.007930859
## [414] {margarine, ## whole milk} 0.3277311 0.024199288 1.7817774	=> 78	{rolls/buns}	0.007930859
## [415] {margarine, ## other vegetables} 0.4690722 0.019725470 1.8357838	=> 91	{whole milk}	0.009252669
<pre>## [416] {margarine, ## whole milk} 0.3823529 0.024199288 1.9760595</pre>	=> 91	{other vegetables}	0.009252669
## [417] {butter, ## domestic eggs} 0.6210526 0.009659380 2.4305820	=> 59	{whole milk}	0.005998983
<pre>## [418] {butter, ## whole milk} 0.2177122 0.027554652 3.4314091</pre>	=> 59	{domestic eggs}	0.005998983
## [419] {domestic eggs, ## whole milk} 0.2000000 0.029994916 3.6091743	=> 59	{butter}	0.005998983
## [420] {butter, ## whipped/sour cream} 0.5700000 0.010167768 2.9458487		{other vegetables}	0.005795628
## [421] {butter, ## other vegetables} 0.2893401 0.020030503 4.0363970		{whipped/sour cream}	0.005795628
## [422] {other vegetables, ## whipped/sour cream} 0.2007042 0.028876462 3.6218827	=>	{butter}	0.005795628
<pre>## [423] {butter, ## whipped/sour cream}</pre>		{whole milk}	0.006710727
0.6600000 0.010167768 2.5830084 ## [424] {butter, ## whole milk}		{whipped/sour cream}	0.006710727
<pre>0.2435424 0.027554652 3.3975033 ## [425] {whipped/sour cream, ## whole milk}</pre>	66 =>	{butter}	0.006710727
0.2082019 0.032231825 3.7571846 ## [426] {butter, ## citrus fruit}	66 =>	{whole milk}	0.005083884
0.5555556 0.009150991 2.1742495	50	(2.003003004

<pre>## [427] {bottled water, ## butter}</pre>	=> {whole	milkl	0.005388917
0.6022727 0.008947636 2.3570841	53	milk)	0.005588517
## [428] {butter,	. Cathan		0.005400505
## tropical fruit} 0.5510204 0.009964413 2.8477592	=> {otner 54	vegetables}	0.005490595
## [429] {butter,	54		
<pre>## other vegetables}</pre>	• •	cal fruit}	0.005490595
0.2741117 0.020030503 2.6122949	54		
<pre>## [430] {butter, ## tropical fruit}</pre>	=> {whole	milk}	0.006202339
0.6224490 0.009964413 2.4360468	61	,	0.000=0=0=
## [431] {butter,			
## whole milk} 0.2250923 0.027554652 2.1451379	=> {tropi	cal fruit}	0.006202339
## [432] {butter,	01		
<pre>## root vegetables}</pre>	=> {other	vegetables}	0.006609049
0.5118110 0.012913066 2.6451190	65		
<pre>## [433] {butter, ## other vegetables}</pre>	-> Spoot v	vegetablesl	0 006600010
0.3299492 0.020030503 3.0270996	65	regerables	0.000009049
## [434] {butter,			
<pre>## root vegetables}</pre>		milk}	0.008235892
0.6377953 0.012913066 2.4961069 ## [435] {butter,	81		
## whole milk}	=> {root '	vegetables}	0.008235892
0.2988930 0.027554652 2.7421759	81		
## [436] {butter,			0.006405604
## yogurt} 0.4375000 0.014641586 2.2610681	=> {otner 63	vegetables}	0.006405694
## [437] {butter,	05		
<pre>## other vegetables}</pre>		t}	0.006405694
0.3197970 0.020030503 2.2924220	63		
## [438] {butter, ## yogurt}	=> {whole	milk}	0.009354347
0.6388889 0.014641586 2.5003869	92		0.00333.13.17
## [439] {butter,	_		
## whole milk} 0.3394834 0.027554652 2.4335417	=> {yogur ⁻ 92	t}	0.009354347
## [440] {butter,	92		
## rolls/buns}	=> {other	vegetables}	0.005693950
0.4242424 0.013421454 2.1925508	56		
<pre>## [441] {butter, ## other vegetables}</pre>	=> {rolls,	/huncl	0.005693950
0.2842640 0.020030503 1.5454594	56	, build j	0.000000000
## [442] {butter,			
## rolls/buns}	=> {whole	milk}	0.006609049
0.4924242 0.013421454 1.9271757 ## [443] {butter,	65		
## whole milk}	=> {rolls,	/buns}	0.006609049

0.2398524 0.027554652 1.3040068 ## [444] {butter,	65		
		(uhala milk)	0 011400570
## other vegetables} 0.5736041 0.020030503 2.2448850		{whore mirk}	0.011489578
	113		
## [445] {butter,		(-th	0.044400570
## whole milk}		<pre>{other vegetables}</pre>	0.011489578
0.4169742 0.027554652 2.1549874	113		
## [446] {newspapers,		6 1 2 1212	
<pre>## tropical fruit}</pre>		<pre>{whole milk}</pre>	0.005083884
0.4310345 0.011794611 1.6869177	50		
## [447] {newspapers,			
<pre>## root vegetables}</pre>	=>	<pre>{other vegetables}</pre>	0.005998983
0.5221239 0.011489578 2.6984175	59		
## [448] {newspapers,			
<pre>## other vegetables}</pre>	=>	<pre>{root vegetables}</pre>	0.005998983
0.3105263 0.019318760 2.8489051	59		
## [449] {newspapers,			
<pre>## root vegetables}</pre>	=>	{whole milk}	0.005795628
0.5044248 0.011489578 1.9741415	57	-	
## [450] {newspapers,			
## whole milk}	=>	<pre>{root vegetables}</pre>	0.005795628
0.2118959 0.027351296 1.9440264	57	,	
## [451] {newspapers,			
## yogurt}	=>	{rolls/buns}	0.005083884
0.3311258 0.015353330 1.8002336	50	(,)	
## [452] {newspapers,			
## rolls/buns}	=>	{yogurt}	0.005083884
0.2577320 0.019725470 1.8475174	50	(Jogui e)	
## [453] {newspapers,	50		
## yogurt}	=>	<pre>{other vegetables}</pre>	0.005592272
0.3642384 0.015353330 1.8824408	55	(other vegetables)	0.003332272
## [454] {newspapers,	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		
## other vegetables}	=>	{yogurt}	0.005592272
0.2894737 0.019318760 2.0750537	55	(yogur c)	0.005552272
## [455] {newspapers,))		
		{whole milk}	0.006609049
## yogurt} 0.4304636 0.015353330 1.6846834	65	(MIDIE MITK)	0.000003043
	05		
## [456] {newspapers,	_\$	(vogunt)	0.006600040
## whole milk}		{yogurt}	0.006609049
0.2416357 0.027351296 1.7321334	65		
## [457] {newspapers,		(athan	0.005400505
## rolls/buns}		<pre>{other vegetables}</pre>	0.005490595
0.2783505 0.019725470 1.4385588	54		
## [458] {newspapers,		(11 ())	0.005400505
<pre>## other vegetables}</pre>		<pre>{rolls/buns}</pre>	0.005490595
0.2842105 0.019318760 1.5451689	54		
## [459] {newspapers,			
## rolls/buns}		{whole milk}	0.007625826
0.3865979 0.019725470 1.5130086	75		
## [460] {newspapers,			

## whole milk} 0.2788104 0.027351296 1.5158100 ## [461] {newspapers,	=> 75	{rolls/buns}	0.007625826
## other vegetables} 0.4315789 0.019318760 1.6890485	=> 82	{whole milk}	0.008337570
## [462] {newspapers, ## whole milk} 0.3048327 0.027351296 1.5754229	=> 82	{other vegetables}	0.008337570
## [463] {domestic eggs, ## whipped/sour cream} 0.5102041 0.009964413 2.6368141	=> 50	{other vegetables}	0.005083884
## [464] {domestic eggs, ## other vegetables} 0.2283105 0.022267412 3.1850125	=> 50	{whipped/sour cream}	0.005083884
## [465] {domestic eggs, ## whipped/sour cream} 0.5714286 0.009964413 2.2363709	=> 56	{whole milk}	0.005693950
## [466] {domestic eggs, ## pip fruit} 0.6235294 0.008642603 2.4402753	=> 53	{whole milk}	0.005388917
## [467] {citrus fruit, ## domestic eggs} 0.5490196 0.010371124 2.1486701	=> 56	{whole milk}	0.005693950
<pre>## [468] {domestic eggs, ## tropical fruit} 0.6071429 0.011387900 2.3761441</pre>	=> 68	{whole milk}	0.006914082
## [469] {domestic eggs, ## whole milk} 0.2305085 0.029994916 2.1967547		{tropical fruit}	0.006914082
## [470] {domestic eggs, ## root vegetables} 0.5106383 0.014336553 2.6390582		{other vegetables}	0.007320793
<pre>## [471] {domestic eggs, ## other vegetables}</pre>	=>	<pre>{root vegetables}</pre>	0.007320793
<pre>0.3287671 0.022267412 3.0162543 ## [472] {domestic eggs, ## root vegetables}</pre>		{whole milk}	0.008540925
0.5957447 0.014336553 2.3315356 ## [473] {domestic eggs, ## whole milk}	84 =>	<pre>{root vegetables}</pre>	0.008540925
<pre>0.2847458 0.029994916 2.6123830 ## [474] {domestic eggs, ## soda}</pre>	84 =>	<pre>{other vegetables}</pre>	0.005083884
0.4098361 0.012404677 2.1180965 ## [475] {domestic eggs,	50		
## other vegetables} 0.2283105 0.022267412 1.3092908 ## [476] {domestic eggs,	50	{soda}	0.005083884
## soda} 0.4180328 0.012404677 1.6360336	=> 51	{whole milk}	0.005185562

<pre>## [477] {domestic eggs, ## yogurt}</pre>	=>	<pre>{other vegetables}</pre>	0.005795628
0.4042553 0.014336553 2.0892544 ## [478] {domestic eggs,	57		
## other vegetables}	=>	{yogurt}	0.005795628
0.2602740 0.022267412 1.8657394	57		
<pre>## [479] {domestic eggs, ## yogurt}</pre>	=>	{whole milk}	0.007727504
0.5390071 0.014336553 2.1094846	76	(WHOLE IIIIK)	0.007727304
## [480] {domestic eggs,			
## whole milk} 0.2576271 0.029994916 1.8467658		{yogurt}	0.007727504
## [481] {domestic eggs,	76		
## rolls/buns}	=>	<pre>{other vegetables}</pre>	0.005897306
0.3766234 0.015658363 1.9464482	58		
## [482] {domestic eggs,		(nolls/huns)	0 005007306
## other vegetables} 0.2648402 0.022267412 1.4398580	= <i>></i> 58	{rolls/buns}	0.005897306
## [483] {domestic eggs,	50		
## rolls/buns}	=>	<pre>{whole milk}</pre>	0.006609049
0.4220779 0.015658363 1.6518648	65		
<pre>## [484] {domestic eggs, ## whole milk}</pre>		{rolls/buns}	0.006609049
0.2203390 0.029994916 1.1979181	65	(10115/buils)	0.000009049
## [485] {domestic eggs,			
<pre>## other vegetables}</pre>		<pre>{whole milk}</pre>	0.012302999
0.5525114 0.022267412 2.1623358	121		
<pre>## [486] {domestic eggs, ## whole milk}</pre>	=>	{other vegetables}	0 012302999
0.4101695 0.029994916 2.1198197	121	(Jene, Vegetables)	0.012302333
## [487] {bottled water,			
<pre>## fruit/vegetable juice}</pre>		{soda}	0.005185562
0.3642857 0.014234875 2.0890671 ## [488] {fruit/vegetable juice,	51		
	=>	{bottled water}	0.005185562
0.2817680 0.018403660 2.5493908	51	,	
## [489] {bottled water,			
## fruit/vegetable juice} 0.4071429 0.014234875 1.5934142	=> 57	<pre>{whole milk}</pre>	0.005795628
## [490] {fruit/vegetable juice,	57		
## whole milk}	=>	{bottled water}	0.005795628
0.2175573 0.026639553 1.9684228	57		
<pre>## [491] {fruit/vegetable juice,</pre>		(athanatablaa)	0.00000000
## tropical fruit} 0.4814815 0.013726487 2.4883712	=> 65	{other vegetables}	0.006609049
## [492] {fruit/vegetable juice,	05		
<pre>## other vegetables}</pre>	=>	{tropical fruit}	0.006609049
0.3140097 0.021047280 2.9925242	65		
<pre>## [493] {fruit/vegetable juice, ## tropical fruit}</pre>	->	{whole milk}	0.005998983
"" Cropical Truits	-/	(MIIOTC IIITIK)	0.0055505

0.4370370 0.013726487 1.7104096 ## [494] {fruit/vegetable juice,	59		
<pre>## whole milk} 0.2251908 0.026639553 2.1460774 ## [495] {fruit/vegetable juice,</pre>	=> 59	{tropical fruit}	0.005998983
<pre>## root vegetables} 0.5508475 0.011997966 2.8468653 ## [496] {fruit/vegetable juice,</pre>	=> 65	<pre>{other vegetables}</pre>	0.006609049
<pre>## other vegetables} 0.3140097 0.021047280 2.8808629 ## [497] {fruit/vegetable juice,</pre>	=> 65	<pre>{root vegetables}</pre>	0.006609049
<pre>## root vegetables} 0.5423729 0.011997966 2.1226571 ## [498] {fruit/vegetable juice,</pre>	64	{whole milk}	0.006507372
<pre>## whole milk} 0.2442748 0.026639553 2.2410847 ## [499] {fruit/vegetable juice,</pre>	=> 64	<pre>{root vegetables}</pre>	0.006507372
<pre>## soda} 0.2762431 0.018403660 1.9802120 ## [500] {fruit/vegetable juice,</pre>	50	{yogurt}	0.005083884
<pre>## yogurt} 0.2717391 0.018708693 1.5583407 ## [501] {fruit/vegetable juice,</pre>	=> 50	{soda}	0.005083884
<pre>## soda} 0.3314917 0.018403660 1.2973422 ## [502] {fruit/vegetable juice,</pre>	=> 60	<pre>{whole milk}</pre>	0.006100661
<pre>## whole milk} 0.2290076 0.026639553 1.3132887 ## [503] {fruit/vegetable juice,</pre>	=> 60	{soda}	0.006100661
<pre>## yogurt} 0.4402174 0.018708693 2.2751120 ## [504] {fruit/vegetable juice,</pre>	=> 81	<pre>{other vegetables}</pre>	0.008235892
## other vegetables} 0.3913043 0.021047280 2.8050133 ## [505] {fruit/vegetable juice,	=> 81	{yogurt}	0.008235892
## yogurt} 0.5054348 0.018708693 1.9780943 ## [506] {fruit/vegetable juice,	=> 93	{whole milk}	0.009456024
## whole milk} 0.3549618 0.026639553 2.5444968 ## [507] {fruit/vegetable juice,	=> 93	{yogurt}	0.009456024
## rolls/buns} 0.3846154 0.014539908 1.5052496 ## [508] {fruit/vegetable juice,	=> 55	{whole milk}	0.005592272
## whole milk} 0.2099237 0.026639553 1.1412931 ## [509] {fruit/vegetable juice,	=> 55	{rolls/buns}	0.005592272
<pre>## other vegetables} 0.4975845 0.021047280 1.9473713 ## [510] {fruit/vegetable juice,</pre>	=> 103	{whole milk}	0.010472801

<pre>## whole milk} 0.3931298 0.026639553 2.0317558 ## [511] {pip fruit,</pre>	<pre>=> {other vegetables} 103</pre>	0.010472801
## whipped/sour cream} 0.6043956 0.009252669 3.1236105 ## [512] {other vegetables,	<pre>=> {other vegetables} 55</pre>	0.005592272
## pip fruit} 0.2140078 0.026131164 2.9854844 ## [513] {pip fruit,	<pre>=> {whipped/sour cream} 55</pre>	0.005592272
## whipped/sour cream} 0.6483516 0.009252669 2.5374208 ## [514] {citrus fruit,	<pre>=> {whole milk} 59</pre>	0.005998983
## whipped/sour cream} 0.5233645 0.010879512 2.7048291 ## [515] {citrus fruit,	<pre>=> {other vegetables} 56</pre>	0.005693950
## whipped/sour cream} 0.5794393 0.010879512 2.2677219	<pre>=> {whole milk} 62</pre>	0.006304016
## [516] {citrus fruit, ## whole milk} 0.2066667 0.030503305 2.8830733	<pre>=> {whipped/sour cream} 62</pre>	0.006304016
## [517] {sausage, ## whipped/sour cream} 0.5617978 0.009049314 2.1986792	<pre>=> {whole milk} 50</pre>	0.005083884
## [518] {tropical fruit, ## whipped/sour cream} 0.4485294 0.013828165 3.2152236	=> {yogurt} 61	0.006202339
## [519] {whipped/sour cream, ## yogurt} 0.2990196 0.020742247 2.8496685	<pre>=> {tropical fruit} 61</pre>	0.006202339
<pre>## [520] {tropical fruit, ## yogurt} 0.2118056 0.029283172 2.9547626</pre>	<pre>=> {whipped/sour cream} 61</pre>	0.006202339
<pre>## [521] {tropical fruit, ## whipped/sour cream} 0.5661765 0.013828165 2.9260881</pre>	<pre>=> {other vegetables} 77</pre>	0.007829181
## [522] {other vegetables, ## whipped/sour cream} 0.2711268 0.028876462 2.5838485	<pre>=> {tropical fruit} 77</pre>	0.007829181
<pre>## [523] {other vegetables, ## tropical fruit} 0.2181303 0.035892222 3.0429952</pre>	=> {whipped/sour cream} 77	0.007829181
<pre>## [524] {tropical fruit, ## whipped/sour cream} 0.5735294 0.013828165 2.2445928</pre>	=> {whole milk} 78	0.007930859
<pre>## [525] {whipped/sour cream, ## whole milk} 0.2460568 0.032231825 2.3449307</pre>	<pre>=> {tropical fruit} 78</pre>	0.007930859
<pre>## [526] {root vegetables, ## whipped/sour cream} 0.3750000 0.017081851 2.6881378</pre>	=> {yogurt} 63	0.006405694

<pre>## [527] {whipped/sour cream, ## yogurt}</pre>	=>	<pre>{root vegetables}</pre>	0.006405694
0.3088235 0.020742247 2.8332830 ## [528] {root vegetables,	63		
## yogurt} 0.2480315 0.025826131 3.4601273	=> 63	{whipped/sour cream}	0.006405694
<pre>## [529] {root vegetables, ## whipped/sour cream}</pre>		{other vegetables}	0.008540925
0.5000000 0.017081851 2.5840778 ## [530] {other vegetables,	84		
## whipped/sour cream} 0.2957746 0.028876462 2.7135668	=> 84	<pre>{root vegetables}</pre>	0.008540925
## [531] {root vegetables, ## whipped/sour cream}		{whole milk}	0.009456024
<pre>0.5535714 0.017081851 2.1664843 ## [532] {whipped/sour cream, ## whole milk}</pre>	93	(most vegetables)	0.000456034
## WHOTE WITK; 0.2933754 0.032231825 2.6915550 ## [533] {soda,	93	<pre>{root vegetables}</pre>	0.009456024
## whipped/sour cream} 0.4736842 0.011591256 1.8538337	=> 54	{whole milk}	0.005490595
## [534] {whipped/sour cream, yogurt}		<pre>{other vegetables}</pre>	0.010167768
0.4901961 0.020742247 2.5334096 ## [535] {other vegetables,	100	(other regerances)	0.02020,700
## whipped/sour cream} 0.3521127 0.028876462 2.5240730	=> 100	{yogurt}	0.010167768
<pre>## [536] {other vegetables, ## yogurt}</pre>	=>	{whipped/sour cream}	0.010167768
0.2341920 0.043416370 3.2670620 ## [537] {whipped/sour cream,	100		
## yogurt} 0.5245098 0.020742247 2.0527473	=> 107	{whole milk}	0.010879512
<pre>## [538] {whipped/sour cream, ## whole milk}</pre>		{yogurt}	0.010879512
0.3375394 0.032231825 2.4196066 ## [539] {rolls/buns,	107		
## whipped/sour cream} 0.4583333 0.014641586 2.3687380	=> 66	{other vegetables}	0.006710727
## [540] {other vegetables, ## whipped/sour cream} 0.2323944 0.028876462 1.2634597	=> 66	{rolls/buns}	0.006710727
## [541] {rolls/buns, ## whipped/sour cream}		{whole milk}	0.007829181
0.5347222 0.014641586 2.0927151 ## [542] {whipped/sour cream,	77	(WHOLE HILLK)	0.007023101
## whole milk} 0.2429022 0.032231825 1.3205877	=> 77	{rolls/buns}	0.007829181
<pre>## [543] {other vegetables, ## whipped/sour cream}</pre>		{whole milk}	0.014641586
		-	

0.5070423 0.028876462 1.9843854	144		
## [544] {whipped/sour cream,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.014641586
0.4542587 0.032231825 2.3476795	144	,	
## [545] {pastry,			
		(la = 1 =	0 005003004
## pip fruit}		<pre>{whole milk}</pre>	0.005083884
0.4761905 0.010676157 1.8636424	50		
## [546] {citrus fruit,			
## pip fruit}	=>	{tropical fruit}	0.005592272
0.4044118 0.013828165 3.8540598	55		
## [547] {pip fruit,			
## tropical fruit}	->	{citrus fruit}	0.005592272
		(CICIUS Truit)	0.003332272
0.2736318 0.020437214 3.3061046	55		
## [548] {citrus fruit,			
## tropical fruit}	=>	{pip fruit}	0.005592272
0.2806122 0.019928826 3.7094374	55		
## [549] {citrus fruit,			
## pip fruit}	->	<pre>{other vegetables}</pre>	0 005897306
0.4264706 0.013828165 2.2040663	58	(other vegetables)	0.000077000
	50		
## [550] {other vegetables,			
## pip fruit}	=>	{citrus fruit}	0.005897306
0.2256809 0.026131164 2.7267469	58		
## [551] {citrus fruit,			
<pre>## other vegetables}</pre>	=>	{pip fruit}	0.005897306
0.2042254 0.028876462 2.6996725	58	(P-P 0-c)	0.003037300
	50		
## [552] {citrus fruit,		() 7 (7)	0.005405560
## pip fruit}		{whole milk}	0.005185562
0.3750000 0.013828165 1.4676184	51		
## [553] {pip fruit,			
## sausage}	=>	<pre>{whole milk}</pre>	0.005592272
0.5188679 0.010777834 2.0306669	55	,	
## [554] {pip fruit,			
		(mant varatables)	0 005207220
<pre>## tropical fruit}</pre>		<pre>{root vegetables}</pre>	0.00528/239
0.2587065 0.020437214 2.3734870	52		
## [555] {pip fruit,			
<pre>## root vegetables}</pre>	=>	{tropical fruit}	0.005287239
0.3398693 0.015556685 3.2389674	52		
## [556] {root vegetables,			
## tropical fruit}	->	Inin fouitl	0.005287239
		{pip fruit}	0.005267259
0.2512077 0.021047280 3.3207366	52		
## [557] {pip fruit,			
<pre>## tropical fruit}</pre>	=>	{yogurt}	0.006405694
0.3134328 0.020437214 2.2468017	63		
## [558] {pip fruit,			
## yogurt}	=>	{tropical fruit}	0.006405694
0.3559322 0.017996950 3.3920477	63	(c. opicar in aic)	0.000 - 000 -
	0.5		
## [559] {tropical fruit,			
## yogurt}	=>	{pip fruit}	0.006405694
0.2187500 0.029283172 2.8916751	63		
## [560] {pip fruit,			

<pre>## tropical fruit} 0.4626866 0.020437214 2.3912361 ## [561] {other vegetables,</pre>	<pre>=> {other vegetables} 93</pre>	0.009456024
## pip fruit} 0.3618677 0.026131164 3.4486132	<pre>=> {tropical fruit} 93</pre>	0.009456024
## [562] {other vegetables, ## tropical fruit} 0.2634561 0.035892222 3.4826487	=> {pip fruit} 93	0.009456024
## [563] {pip fruit, ## tropical fruit} 0.4129353 0.020437214 1.6160839	<pre>=> {whole milk} 83</pre>	0.008439248
## [564] {pip fruit, ## whole milk} 0.2804054 0.030096594 2.6722744	<pre>=> {tropical fruit} 83</pre>	0.008439248
## [565] {pip fruit, ## root vegetables} 0.3398693 0.015556685 2.4363079	=> {yogurt} 52	0.005287239
## [566] {pip fruit, ## yogurt} 0.2937853 0.017996950 2.6953158	<pre>=> {root vegetables} 52</pre>	0.005287239
<pre>## [567] {root vegetables, ## yogurt} 0.2047244 0.025826131 2.7062696</pre>	=> {pip fruit} 52	0.005287239
<pre>## [568] {pip fruit, ## root vegetables} 0.5228758 0.015556685 2.7023036</pre>	<pre>=> {other vegetables} 80</pre>	0.008134215
<pre>## [569] {other vegetables, ## pip fruit} 0.3112840 0.026131164 2.8558569</pre>	<pre>=> {root vegetables} 80</pre>	0.008134215
<pre>## [570] {pip fruit, ## root vegetables} 0.5751634 0.015556685 2.2509877</pre>	=> {whole milk} 88	0.008947636
## [571] {pip fruit, ## whole milk} 0.2972973 0.030096594 2.7275363	<pre>=> {root vegetables} 88</pre>	0.008947636
## [572] {pip fruit, ## yogurt}	<pre>=> {other vegetables}</pre>	0.008134215
<pre>0.4519774 0.017996950 2.3358895 ## [573] {other vegetables, ## pip fruit}</pre>	80 => {yogurt}	0.008134215
0.3112840 0.026131164 2.2313984 ## [574] {pip fruit, ## yogurt}	<pre>80 => {whole milk}</pre>	0.009557702
<pre>0.5310734 0.017996950 2.0784351 ## [575] {pip fruit, ## whole milk}</pre>	94 => {yogurt}	0.009557702
0.3175676 0.030096594 2.2764410 ## [576] {pip fruit, ## rolls/buns}	94 => {other vegetables}	0.005083884
0.3649635 0.013929842 1.8861882	50	

<pre>## [577] {pip fruit, ## rolls/buns} 0.4452555 0.013929842 1.7425</pre>		{whole milk}	0.006202339
<pre>## [578] {pip fruit, ## whole milk} 0.2060811 0.030096594 1.1204</pre>		{rolls/buns}	0.006202339
<pre>## [579] {other vegetables, ## pip fruit} 0.5175097 0.026131164 2.0253</pre>	=>	{whole milk}	0.013523132
## [580] {pip fruit, ## whole milk} 0.4493243 0.030096594 2.3221	=>	{other vegetables}	0.013523132
## [581] {pastry, ## sausage} 0.4552846 0.012506355 1.7818	=>	{whole milk}	0.005693950
## [582] {pastry, ## tropical fruit} 0.3846154 0.013218099 1.9877	=>	{other vegetables}	0.005083884
<pre>## [583] {other vegetables, ## pastry}</pre>	=>	{tropical fruit}	0.005083884
0.2252252 0.022572445 2.1464 ## [584] {pastry, ## tropical fruit}	=>	{whole milk}	0.006710727
0.5076923 0.013218099 1.9869 ## [585] {pastry, ## whole milk}		{tropical fruit}	0.006710727
<pre>0.2018349 0.033248602 1.9234 ## [586] {pastry, ## root vegetables}</pre>		<pre>{other vegetables}</pre>	0.005897306
0.5370370 0.010981190 2.7754 ## [587] {other vegetables,	1909 58		
## pastry} 0.2612613 0.022572445 2.3969 ## [588] {pastry,	9258 58	<pre>{root vegetables}</pre>	
## root vegetables} 0.5185185 0.010981190 2.0292 ## [589] {pastry,		<pre>{whole milk}</pre>	0.005693950
## soda} 0.2560386 0.021047280 1.3920 ## [590] {pastry,		{rolls/buns}	0.005388917
## rolls/buns} 0.2572816 0.020945602 1.4754 ## [591] {pastry,		{soda}	0.005388917
## soda} 0.2608696 0.021047280 1.3482		{other vegetables}	0.005490595
## [592] {other vegetables, ## pastry} 0.2432432 0.022572445 1.3949		{soda}	0.005490595
## [593] {pastry, ## soda}	=>	{whole milk}	0.008235892

0.3913043 0.021047280 1.5314279 ## [594] {pastry,	81		
## whole milk}	=>	{soda}	0.008235892
0.2477064 0.033248602 1.4205205	81	(5500)	0.000=3303=
## [595] {soda,			
<pre>## whole milk}</pre>		{pastry}	0.008235892
0.2055838 0.040061007 2.3107614	81		
## [596] {pastry, ## yogurt}		(nolls/buns)	0.005795628
0.3275862 0.017691917 1.7809897	= <i>></i> 57	{rolls/buns}	0.003/93020
## [597] {pastry,	37		
## rolls/buns}	=>	{yogurt}	0.005795628
0.2766990 0.020945602 1.9834803	57		
## [598] {pastry,			
## yogurt}		<pre>{other vegetables}</pre>	0.006609049
0.3735632 0.017691917 1.9306328 ## [599] {other vegetables,	65		
## pastry}	=>	{yogurt}	0.006609049
0.2927928 0.022572445 2.0988463	65	() 084. 0	0.0000000000000000000000000000000000000
## [600] {pastry,			
## yogurt}		<pre>{whole milk}</pre>	0.009150991
0.5172414 0.017691917 2.0243012	90		
## [601] {pastry,		(0.000150001
## whole milk} 0.2752294 0.033248602 1.9729451	=> 90	{yogurt}	0.009150991
## [602] {pastry,	30		
## rolls/buns}	=>	{other vegetables}	0.006100661
0.2912621 0.020945602 1.5052880	60	· · · · · · · · · · · · · · · · · · ·	
## [603] {other vegetables,			
## pastry}		{rolls/buns}	0.006100661
0.2702703 0.022572445 1.4693798	60		
<pre>## [604] {pastry, ## rolls/buns}</pre>	->	{whole milk}	0.008540925
0.4077670 0.020945602 1.5958569	84	(MIOTE MITK)	0.000540525
## [605] {pastry,	0.		
## whole milk}	=>	<pre>{rolls/buns}</pre>	0.008540925
0.2568807 0.033248602 1.3965849	84		
## [606] {other vegetables,		6	
## pastry}		{whole milk}	0.010574479
0.4684685 0.022572445 1.8334212 ## [607] {pastry,	104		
## whole milk}	=>	<pre>{other vegetables}</pre>	0.010574479
0.3180428 0.033248602 1.6436947	104	(comer regularization)	0102007 1 175
## [608] {bottled water,			
## citrus fruit}	=>	<pre>{other vegetables}</pre>	0.005083884
0.3759398 0.013523132 1.9429156	50		
## [609] {bottled water,		(aithmus Court)	0.00500004
## other vegetables} 0.2049180 0.024809354 2.4758831	=> 50	{citrus fruit}	0.005083884
## [610] {bottled water,	שכ		
"" [oro] (boccrea water)			

<pre>## citrus fruit} 0.4360902 0.013523132 1.7067041 ## [611] {citrus fruit,</pre>	=> 58	{whole milk}	0.005897306
## tropical fruit} 0.2857143 0.019928826 2.6212687	=> 56	<pre>{root vegetables}</pre>	0.005693950
## [612] {citrus fruit, ## root vegetables} 0.3218391 0.017691917 3.0671389	=> 56	{tropical fruit}	0.005693950
## [613] {root vegetables, ## tropical fruit} 0.2705314 0.021047280 3.2686441	=> 56	{citrus fruit}	0.005693950
## [614] {citrus fruit, ## tropical fruit} 0.3163265 0.019928826 2.2675448	=> 62	{yogurt}	0.006304016
## [615] {citrus fruit, ## yogurt} 0.2910798 0.021657346 2.7740019	=> 62	{tropical fruit}	0.006304016
## [616] {tropical fruit, ## yogurt} 0.2152778 0.029283172 2.6010528	=> 62	{citrus fruit}	0.006304016
## [617] {citrus fruit, ## tropical fruit} 0.4540816 0.019928826 2.3467645	=> 89	{other vegetables}	0.009049314
## [618] {citrus fruit, ## other vegetables} 0.3133803 0.028876462 2.9865262	=> 89	{tropical fruit}	0.009049314
<pre>## [619] {other vegetables, ## tropical fruit} 0.2521246 0.035892222 3.0462480</pre>	=> 89	{citrus fruit}	0.009049314
<pre>## [620] {citrus fruit, ## tropical fruit} 0.4540816 0.019928826 1.7771161</pre>	=> 89	{whole milk}	0.009049314
## [621] {citrus fruit, ## whole milk} 0.2966667 0.030503305 2.8272448		{tropical fruit}	0.009049314
<pre>## [622] {tropical fruit, ## whole milk}</pre>	=>	{citrus fruit}	0.009049314
<pre>0.2139423 0.042297916 2.5849172 ## [623] {citrus fruit, ## root vegetables}</pre>		{other vegetables}	0.010371124
<pre>0.5862069 0.017691917 3.0296084 ## [624] {citrus fruit, ## other vegetables}</pre>	102	<pre>{root vegetables}</pre>	0.010371124
0.3591549 0.028876462 3.2950455 ## [625] {other vegetables, ## root vegetables}	102	{citrus fruit}	0.010371124
0.2188841 0.047381800 2.6446257 ## [626] {citrus fruit,	102		
## root vegetables} 0.5172414 0.017691917 2.0243012	=> 90	{whole milk}	0.009150991

<pre>## [627] {citrus fruit, ## whole milk}</pre>	=>	<pre>{root vegetables}</pre>	0.009150991
0.3000000 0.030503305 2.7523321 ## [628] {citrus fruit,	90		
## yogurt}		<pre>{rolls/buns}</pre>	0.005795628
0.2676056 0.021657346 1.4548930	57		
<pre>## [629] {citrus fruit, ## rolls/buns}</pre>	=>	{yogurt}	0.005795628
0.3454545 0.016776817 2.4763451	57		
## [630] {citrus fruit,		(athanaatablaa)	0.007625026
## yogurt} 0.3521127 0.021657346 1.8197731	=> 75	{other vegetables}	0.00/025820
## [631] {citrus fruit,			
## other vegetables}		{yogurt}	0.007625826
0.2640845 0.028876462 1.8930548 ## [632] {citrus fruit,	75		
## yogurt}	=>	<pre>{whole milk}</pre>	0.010269446
0.4741784 0.021657346 1.8557678	101		
<pre>## [633] {citrus fruit, ## whole milk}</pre>	=>	{yogurt}	0.010269446
0.3366667 0.030503305 2.4133503	101	(Jogui e)	0.010203110
## [634] {citrus fruit,			
## rolls/buns} 0.3575758 0.016776817 1.8480071	=> 59	{other vegetables}	0.005998983
## [635] {citrus fruit,	55		
<pre>## other vegetables}</pre>		<pre>{rolls/buns}</pre>	0.005998983
0.2077465 0.028876462 1.1294564 ## [636] {citrus fruit,	59		
## rolls/buns}	=>	{whole milk}	0.007219115
0.4303030 0.016776817 1.6840550	71	•	
<pre>## [637] {citrus fruit, ## whole milk}</pre>	_\	{rolls/buns}	0.007219115
0.2366667 0.030503305 1.2866869	71	(10115/bull5)	0.00/219113
## [638] {citrus fruit,			
## other vegetables} 0.4507042 0.028876462 1.7638982	=> 128	{whole milk}	0.013014743
## [639] {citrus fruit,	120		
## whole milk}		<pre>{other vegetables}</pre>	0.013014743
0.4266667 0.030503305 2.2050797	128		
<pre>## [640] {sausage, ## shopping bags}</pre>	=>	{soda}	0.005693950
0.3636364 0.015658363 2.0853432	56		
## [641] {shopping bags, ## soda}		(caucago)	0.005602050
## soda} 0.2314050 0.024605999 2.4630604	= <i>></i> 56	{sausage}	0.005693950
## [642] {sausage,			
## soda}		<pre>{shopping bags}</pre>	0.005693950
0.2343096 0.024300966 2.3781580 ## [643] {sausage,	56		
## shopping bags}	=>	{rolls/buns}	0.005998983

0.3831169 0.015658363 2.0828936 ## [644] {rolls/buns,	59	
## shopping bags} 0.3072917 0.019522115 3.2707939	=> {sausage} 59	0.005998983
## [645] {sausage, ## shopping bags} 0.3441558 0.015658363 1.7786509 ## [646] {other vegetables,	<pre>=> {other vegetables} 53</pre>	0.005388917
## shopping bags} 0.2324561 0.023182511 2.4742491 ## [647] {other vegetables,	=> {sausage} 53	0.005388917
## sausage} 0.2000000 0.026944586 2.0299278 ## [648] {root vegetables,	<pre>=> {shopping bags} 53</pre>	0.005388917
## shopping bags} 0.5158730 0.012811388 2.6661120 ## [649] {other vegetables,	<pre>=> {other vegetables} 65</pre>	0.006609049
## shopping bags} 0.2850877 0.023182511 2.6155203 ## [650] {root vegetables,	<pre>=> {root vegetables} 65</pre>	0.006609049
## shopping bags} 0.4126984 0.012811388 1.6151567 ## [651] {shopping bags,	<pre>=> {whole milk} 52</pre>	0.005287239
## whole milk} 0.2157676 0.024504321 1.9795473 ## [652] {shopping bags,	<pre>=> {root vegetables} 52</pre>	0.005287239
## soda} 0.2561983 0.024605999 1.3928749 ## [653] {rolls/buns,	<pre>=> {rolls/buns} 62</pre>	0.006304016
## shopping bags} 0.3229167 0.019522115 1.8518282 ## [654] {shopping bags,	=> {soda} 62	0.006304016
## soda} 0.2190083 0.024605999 1.1318688	<pre>=> {other vegetables} 53</pre>	0.005388917
## [655] {other vegetables, ## shopping bags} 0.2324561 0.023182511 1.3330648	=> {soda} 53	0.005388917
## [656] {shopping bags, ## soda} 0.2768595 0.024605999 1.0835309	<pre>=> {whole milk} 67</pre>	0.006812405
## [657] {shopping bags, ## whole milk} 0.2780083 0.024504321 1.5942925	=> {soda} 67	0.006812405
## [658] {shopping bags, ## yogurt} 0.3533333 0.015251652 1.8260816	<pre>=> {other vegetables} 53</pre>	0.005388917
## [659] {other vegetables, ## shopping bags} 0.2324561 0.023182511 1.6663310	=> {yogurt} 53	0.005388917
## [660] {shopping bags,		

## yogurt} 0.3466667 0.015251652 1.3567317	=> 52	{whole milk}	0.005287239
## [661] {shopping bags, ## whole milk}		{yogurt}	0.005287239
0.2157676 0.024504321 1.5467017 ## [662] {rolls/buns,	52		
## shopping bags} 0.2708333 0.019522115 1.3997088	=> 52	{other vegetables}	0.00528/239
## [663] {other vegetables, ## shopping bags} 0.2280702 0.023182511 1.2399503	=> 52	{rolls/buns}	0.005287239
## [664] {rolls/buns, ## shopping bags}		{whole milk}	0.005287239
0.2708333 0.019522115 1.0599466 ## [665] {shopping bags,	52	(WHOLE HILLK)	0.003287233
## whole milk} 0.2157676 0.024504321 1.1730651	=> 52	{rolls/buns}	0.005287239
<pre>## [666] {other vegetables, ## shopping bags}</pre>		{whole milk}	0.007625826
0.3289474 0.023182511 1.2873845 ## [667] {shopping bags,	75	,	
## whole milk} 0.3112033 0.024504321 1.6083472	=> 75	{other vegetables}	0.007625826
<pre>## [668] {bottled water, ## sausage}</pre>	=>	<pre>{other vegetables}</pre>	0.005083884
0.4237288 0.011997966 2.1898964 ## [669] {bottled water,	50		
## other vegetables} 0.2049180 0.024809354 2.1811351	=> 50	{sausage}	0.005083884
<pre>## [670] {sausage, ## tropical fruit}</pre>		{other vegetables}	0.005998983
0.4306569 0.013929842 2.2257020 ## [671] {other vegetables,	59	(huaniaal (muit)	0.00500000
## sausage} 0.2226415 0.026944586 2.1217822	=> 59	{tropical fruit}	0.005998983
<pre>## [672] {sausage, ## tropical fruit} 0.5182482 0.013929842 2.0282415</pre>	=> 71	{whole milk}	0.007219115
## [673] {sausage, ## whole milk}		{tropical fruit}	0.007219115
0.2414966 0.029893238 2.3014719 ## [674] {root vegetables,	71	(cropical frait)	0.007213113
## sausage} 0.3469388 0.014946619 2.4869846	=> 51	{yogurt}	0.005185562
## [675] {sausage, ## yogurt}	=>	<pre>{root vegetables}</pre>	0.005185562
0.2642487 0.019623793 2.4243340 ## [676] {root vegetables,	51		
## yogurt} 0.2007874 0.025826131 2.1371689	=> 51	{sausage}	0.005185562

## [677] {root vegetables,		(athon wasatahlas)	0.000012405
## sausage} 0.4557823 0.014946619 2.3555539	=> 67	{other vegetables}	0.006812405
## [678] {other vegetables,	07		
## sausage}	=>	<pre>{root vegetables}</pre>	0.006812405
0.2528302 0.026944586 2.3195755	67	,	
## [679] {root vegetables,			
## sausage}	=>	<pre>{whole milk}</pre>	0.007727504
0.5170068 0.014946619 2.0233832	76		
## [680] {sausage,			
## whole milk}	=>	<pre>{root vegetables}</pre>	0.007727504
0.2585034 0.029893238 2.3716240	76		
## [681] {sausage,			
## soda}		{yogurt}	0.005592272
0.2301255 0.024300966 1.6496243	55		
## [682] {sausage,		(4-)	0.005503333
## yogurt}		{soda}	0.005592272
0.2849741 0.019623793 1.6342392	55		
## [683] {soda, ## yogurt}	_\	{sausage}	0.005592272
0.2044610 0.027351296 2.1762701	55	[sausage]	0.003332272
## [684] {sausage,))		
## soda}	=>	{rolls/buns}	0.009659380
0.3974895 0.024300966 2.1610335	95	(10113) build	0.000000000
## [685] {rolls/buns,			
## sausage}	=>	{soda}	0.009659380
0.3156146 0.030604982 1.8099532	95		
## [686] {rolls/buns,			
## soda}	=>	{sausage}	0.009659380
0.2519894 0.038332486 2.6821598	95		
## [687] {sausage,			
## soda}		<pre>{other vegetables}</pre>	0.007219115
0.2970711 0.024300966 1.5353098	71		
## [688] {other vegetables,		()	0.007040445
## sausage}		{soda}	0.007219115
0.2679245 0.026944586 1.5364652	71		
## [689] {other vegetables,	_,	(caucage)	0 007210115
## soda} 0.2204969 0.032740214 2.3469556	=> 71	{sausage}	0.007219115
## [690] {sausage,	/ 1		
## soda}	=>	{whole milk}	0.006710727
0.2761506 0.024300966 1.0807566	66	(WHOIC MIIK)	0.000/10/2/
## [691] {sausage,	00		
## whole milk}	=>	{soda}	0.006710727
0.2244898 0.029893238 1.2873803	66	(
## [692] {sausage,			
## yogurt}	=>	<pre>{rolls/buns}</pre>	0.005998983
0.3056995 0.019623793 1.6619980	59	-	
## [693] {sausage,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.008134215

0.4145078 0.019623793 2.1422406 ## [694] {other vegetables,	80		
## sausage}	=>	{yogurt}	0.008134215
0.3018868 0.026944586 2.1640354	80	(Jogui C)	0.00013 1213
## [695] {sausage,	00		
## yogurt}	=>	{whole milk}	0.008744281
0.4455959 0.019623793 1.7439058	86	(WHOLE IIILIN)	0.000711201
## [696] {sausage,	00		
## whole milk}	=>	{yogurt}	0.008744281
0.2925170 0.029893238 2.0968694	86	(Jogui e)	0.000711201
## [697] {rolls/buns,	00		
## sausage}	=>	<pre>{other vegetables}</pre>	0.008845958
0.2890365 0.030604982 1.4937858	87	(cent. regerales)	01000015550
## [698] {other vegetables,	0.		
## sausage}	=>	{rolls/buns}	0.008845958
0.3283019 0.026944586 1.7848806	87	(
## [699] {other vegetables,			
## rolls/buns}	=>	{sausage}	0.008845958
0.2076372 0.042602949 2.2100781	87	(
## [700] {rolls/buns,			
## sausage}	=>	{whole milk}	0.009354347
0.3056478 0.030604982 1.1961984	92	(
## [701] {sausage,			
## whole milk}	=>	{rolls/buns}	0.009354347
0.3129252 0.029893238 1.7012820	92	, ,	
## [702] {other vegetables,			
## sausage}	=>	<pre>{whole milk}</pre>	0.010167768
0.3773585 0.026944586 1.4768487	100	Ţ	
## [703] {sausage,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.010167768
0.3401361 0.029893238 1.7578760	100	-	
## [704] {bottled water,			
<pre>## tropical fruit}</pre>	=>	{soda}	0.005185562
0.2802198 0.018505338 1.6069747	51		
## [705] {soda,			
<pre>## tropical fruit}</pre>	=>	{bottled water}	0.005185562
0.2487805 0.020843925 2.2509256	51		
## [706] {bottled water,			
<pre>## tropical fruit}</pre>	=>	{yogurt}	0.007117438
0.3846154 0.018505338 2.7570644	70		
## [707] {bottled water,			
## yogurt}	=>	<pre>{tropical fruit}</pre>	0.007117438
0.3097345 0.022979156 2.9517819	70		
## [708] {tropical fruit,			
## yogurt}	=>	{bottled water}	0.007117438
0.2430556 0.029283172 2.1991273	70		
## [709] {bottled water,			
<pre>## tropical fruit}</pre>	=>	<pre>{rolls/buns}</pre>	0.005388917
0.2912088 0.018505338 1.5832164	53		
## [710] {bottled water,			

<pre>## rolls/buns} 0.2226891 0.024199288 2.1222355 ## [711] {rolls/buns,</pre>	=> 53	{tropical fruit}	0.005388917
## tropical fruit} 0.2190083 0.024605999 1.9815513	=> 53	{bottled water}	0.005388917
<pre>## [712] {bottled water, ## tropical fruit} 0.3351648 0.018505338 1.7321840 ## [713] {bottled water,</pre>	=> 61	{other vegetables}	0.006202339
## other vegetables} 0.2500000 0.024809354 2.3825097 ## [714] {bottled water,	=> 61	{tropical fruit}	0.006202339
## tropical fruit} 0.4340659 0.018505338 1.6987817 ## [715] {bottled water,	=> 79	{whole milk}	0.008032537
## whole milk} 0.2337278 0.034367056 2.2274351	=> 79	{tropical fruit}	0.008032537
## [716] {bottled water, ## root vegetables} 0.4480519 0.015658363 2.3156022	=> 69	{other vegetables}	0.007015760
## [717] {bottled water, ## other vegetables} 0.2827869 0.024809354 2.5944114	=> 69	{root vegetables}	0.007015760
## [718] {bottled water, ## root vegetables} 0.4675325 0.015658363 1.8297580	=> 72	{whole milk}	0.007320793
## [719] {bottled water, ## whole milk} 0.2130178 0.034367056 1.9543186	=> 72	<pre>{root vegetables}</pre>	0.007320793
## [720] {bottled water, ## soda} 0.2561404 0.028978139 1.8361081	=> 73	{yogurt}	0.007422471
## [721] {bottled water, ## yogurt} 0.3230088 0.022979156 1.8523569	=> 73	{soda}	0.007422471
## [722] {soda, ## yogurt} 0.2713755 0.027351296 2.4553613	=> 73	{bottled water}	0.007422471
## [723] {bottled water, ## soda} 0.2350877 0.028978139 1.2781027	=> 67	{rolls/buns}	0.006812405
## [724] {bottled water, ## rolls/buns} 0.2815126 0.024199288 1.6143886	=> 67	{soda}	0.006812405
## [725] {bottled water, ## other vegetables} 0.2295082 0.024809354 1.3161593	=> 56	{soda}	0.005693950
## [726] {bottled water, ## soda} 0.2596491 0.028978139 1.0161755	=> 74	{whole milk}	0.007524148

<pre>## [727] {bottled water, ## whole milk}</pre>	=>	{soda}	0.007524148
0.2189349 0.034367056 1.2555247 ## [728] {bottled water,	74		
## yogurt} 0.3097345 0.022979156 1.6839353	=> 70	<pre>{rolls/buns}</pre>	0.007117438
## [729] {bottled water,		()	0.007447420
## rolls/buns} 0.2941176 0.024199288 2.1083433	=> 70	{yogurt}	0.007117438
## [730] {rolls/buns, ## yogurt}	=>	{bottled water}	0.007117438
0.2071006 0.034367056 1.8738126 ## [731] {bottled water,	70	,	
## yogurt}		{other vegetables}	0.008134215
0.3539823 0.022979156 1.8294356 ## [732] {bottled water,	80		
## other vegetables} 0.3278689 0.024809354 2.3502844	=> 80	{yogurt}	0.008134215
<pre>## [733] {bottled water, ## yogurt}</pre>	=>	{whole milk}	0.009659380
0.4203540 0.022979156 1.6451180 ## [734] {bottled water,	95	(
## whole milk}		{yogurt}	0.009659380
0.2810651 0.034367056 2.0147778 ## [735] {bottled water,	95		
## rolls/buns} 0.3025210 0.024199288 1.5634756	=> 72	{other vegetables}	0.007320793
<pre>## [736] {bottled water, ## other vegetables}</pre>	=>	{rolls/buns}	0.007320793
0.2950820 0.024809354 1.6042737 ## [737] {bottled water,	72		
## rolls/buns}		{whole milk}	0.008744281
0.3613445 0.024199288 1.4141757 ## [738] {bottled water,	86		
## whole milk} 0.2544379 0.034367056 1.3833037	=> 86	{rolls/buns}	0.008744281
<pre>## [739] {bottled water, ## other vegetables}</pre>	=>	{whole milk}	0.010777834
0.4344262 0.024809354 1.7001918 ## [740] {bottled water,	106		
## whole milk} 0.3136095 0.034367056 1.6207825	=> 106	<pre>{other vegetables}</pre>	0.010777834
## [741] {root vegetables,		(vegunt)	0.000134345
## tropical fruit} 0.3864734 0.021047280 2.7703835	=> 80	{yogurt}	0.008134215
<pre>## [742] {tropical fruit, ## yogurt}</pre>	=>	<pre>{root vegetables}</pre>	0.008134215
0.2777778 0.029283172 2.5484556 ## [743] {root vegetables,	80		
## yogurt}	=>	{tropical fruit}	0.008134215

0.3149606 0.025826131 3.0015870	80		
## [744] {root vegetables,		(11 ())	0.005007306
## tropical fruit}		<pre>{rolls/buns}</pre>	0.00589/306
0.2801932 0.021047280 1.5233281	58		
## [745] {rolls/buns,			0.005007306
## tropical fruit}		<pre>{root vegetables}</pre>	0.005897306
0.2396694 0.024605999 2.1988328	58		
## [746] {rolls/buns,		(thereign)	0.005007306
## root vegetables}		{tropical fruit}	0.005897306
0.2426778 0.024300966 2.3127291	58		
## [747] {root vegetables,		(athon wasatables)	0.012202000
<pre>## tropical fruit} 0.5845411 0.021047280 3.0209991</pre>	121	<pre>{other vegetables}</pre>	0.012302999
	121		
<pre>## [748] {other vegetables, ## tropical fruit}</pre>		(noot vogotables)	0.012302999
0.3427762 0.035892222 3.1447798	121	<pre>{root vegetables}</pre>	0.012302999
## [749] {other vegetables,	121		
## root vegetables}		{tropical fruit}	0.012302999
0.2596567 0.047381800 2.4745380	121	(Cropical Truit)	0.012302999
## [750] {root vegetables,	121		
## tropical fruit}	->	{whole milk}	0.011997966
0.5700483 0.021047280 2.2309690	118	(MIIOTE IIITIK)	0.011997900
## [751] {tropical fruit,	110		
## whole milk}	=>	<pre>{root vegetables}</pre>	0.011997966
0.2836538 0.042297916 2.6023653	118	(1000 vegetubies)	0.011337300
## [752] {root vegetables,	110		
## whole milk}	=>	{tropical fruit}	0.011997966
0.2453222 0.048906965 2.3379305	118	(c. opical ale)	0.011337300
## [753] {soda,			
## tropical fruit}	=>	{yogurt}	0.006609049
0.3170732 0.020843925 2.2728970	65		
## [754] {tropical fruit,			
## yogurt}	=>	{soda}	0.006609049
0.2256944 0.029283172 1.2942885	65		
## [755] {soda,			
## yogurt}	=>	<pre>{tropical fruit}</pre>	0.006609049
0.2416357 0.027351296 2.3027975	65		
## [756] {soda,			
<pre>## tropical fruit}</pre>	=>	<pre>{rolls/buns}</pre>	0.005388917
0.2585366 0.020843925 1.4055872	53		
## [757] {rolls/buns,			
<pre>## tropical fruit}</pre>	=>	{soda}	0.005388917
0.2190083 0.024605999 1.2559454	53		
## [758] {soda,			
<pre>## tropical fruit}</pre>		<pre>{other vegetables}</pre>	0.007219115
0.3463415 0.020843925 1.7899466	71		
## [759] {other vegetables,			
<pre>## tropical fruit}</pre>		{soda}	0.007219115
0.2011331 0.035892222 1.1534370	71		
## [760] {other vegetables,			

0.2204969 0.032740214 2.1013440		{tropical fruit}	0.007219115
## [761] {soda, ## tropical fruit} 0.3756098 0.020843925 1.4700048	=> 77	{whole milk}	0.007829181
<pre>## [762] {tropical fruit, ## yogurt} 0.2986111 0.029283172 1.6234606</pre>	=> 86	{rolls/buns}	0.008744281
<pre>## [763] {rolls/buns, ## tropical fruit} 0.3553719 0.024605999 2.5474363</pre>	=> 86	{yogurt}	0.008744281
## [764] {rolls/buns, ## yogurt}	=>	{tropical fruit}	0.008744281
<pre>0.2544379 0.034367056 2.4248028 ## [765] {tropical fruit, ## yogurt}</pre>	86 =>	<pre>{other vegetables}</pre>	0.012302999
0.4201389 0.029283172 2.1713431 ## [766] {other vegetables,	121	(vogunt)	A A122A2000
<pre>## tropical fruit} 0.3427762 0.035892222 2.4571457 ## [767] {other vegetables,</pre>	121	{yogurt}	0.012302999
## yogurt} 0.2833724 0.043416370 2.7005496 ## [768] {tropical fruit,	=> 121	{tropical fruit}	0.012302999
## yogurt} 0.5173611 0.029283172 2.0247698	=> 149	{whole milk}	0.015149975
<pre>## [769] {tropical fruit, ## whole milk} 0.3581731 0.042297916 2.5675162</pre>	=> 149	{yogurt}	0.015149975
## [770] {whole milk, ## yogurt}	=>	{tropical fruit}	0.015149975
<pre>0.2704174 0.056024403 2.5770885 ## [771] {rolls/buns, ## tropical fruit}</pre>	149 =>	<pre>{other vegetables}</pre>	0.007829181
0.3181818 0.024605999 1.6444131 ## [772] {other vegetables,	77		
<pre>## tropical fruit} 0.2181303 0.035892222 1.1859102 ## [773] {rolls/buns,</pre>	=> 77	{rolls/buns}	0.007829181
<pre>## tropical fruit} 0.4462810 0.024605999 1.7465872 ## [774] {tropical fruit,</pre>	=> 108	{whole milk}	0.010981190
## whole milk} 0.2596154 0.042297916 1.4114524	=> 108	{rolls/buns}	0.010981190
<pre>## [775] {other vegetables, ## tropical fruit} 0.4759207 0.035892222 1.8625865</pre>	=> 168	{whole milk}	0.017081851
<pre>## [776] {tropical fruit, ## whole milk}</pre>	=>	{other vegetables}	0.017081851
0.4038462 0.042297916 2.0871397	168		

## [777] {other vegetables,		6	
<pre>## whole milk}</pre>		{tropical fruit}	0.017081851
0.2282609 0.074834774 2.1753349	168		
## [778] {root vegetables,			
## soda}		<pre>{other vegetables}</pre>	0.008235892
0.4426230 0.018607016 2.2875443	81		
## [779] {other vegetables,		(tt.h1)	0.000335003
## soda}		<pre>{root vegetables}</pre>	0.008235892
0.2515528 0.032740214 2.3078561	81		
## [780] {root vegetables,		(ha] a m.*] [.]	0 000124215
## soda} 0.4371585 0.018607016 1.7108848		<pre>{whole milk}</pre>	0.008134215
	80		
## [781] {soda, ## whole milk}		<pre>{root vegetables}</pre>	0.008134215
0.2030457 0.040061007 1.8628305	=> 80	{root vegetables}	0.000134213
## [782] {root vegetables,	80		
## yogurt}	->	{rolls/buns}	0.007219115
0.2795276 0.025826131 1.5197090	71	(1 OIIS/ Dulis)	0.00/219113
## [783] {rolls/buns,	/ 1		
## root vegetables}	=>	{yogurt}	0.007219115
0.2970711 0.024300966 2.1295150	71	(yogur c)	0.007213113
## [784] {rolls/buns,	, _		
## yogurt}	=>	<pre>{root vegetables}</pre>	0.007219115
0.2100592 0.034367056 1.9271753	71	(. oot regetuores)	0.00,22,22
## [785] {root vegetables,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.012913066
0.5000000 0.025826131 2.5840778	127		
## [786] {other vegetables,			
<pre>## root vegetables}</pre>	=>	{yogurt}	0.012913066
0.2725322 0.047381800 1.9536108	127		
## [787] {other vegetables,			
## yogurt}	=>	<pre>{root vegetables}</pre>	0.012913066
0.2974239 0.043416370 2.7286977	127		
## [788] {root vegetables,			
## yogurt}	=>	{whole milk}	0.014539908
0.5629921 0.025826131 2.2033536	143		
## [789] {root vegetables,			
## whole milk}	=>	<pre>{yogurt}</pre>	0.014539908
0.2972973 0.048906965 2.1311362	143		
## [790] {whole milk,			
## yogurt}		<pre>{root vegetables}</pre>	0.014539908
0.2595281 0.056024403 2.3810253	143		
## [791] {rolls/buns,			0.04005:555
<pre>## root vegetables}</pre>		<pre>{other vegetables}</pre>	0.012201322
0.5020921 0.024300966 2.5948898	120		
## [792] {other vegetables,		(malla /km)	0.042204222
## root vegetables}		{rolls/buns}	0.012201322
0.2575107 0.047381800 1.4000100	120		
## [793] {other vegetables,		(noot vegetables)	0 012201222
## rolls/buns}	=>	<pre>{root vegetables}</pre>	0.012201322

0.2863962 0.042602949 2.6275247	120		
## [794] {rolls/buns,		(بالمام مسئال)	0 012700710
## root vegetables} 0.5230126 0.024300966 2.0468876		{whore wirk}	0.012/09/10
	125		
<pre>## [795] {root vegetables, ## whole milk}</pre>	_\$	(nolle/bune)	0.012709710
## whole milk} 0.2598753 0.048906965 1.4128652	=> 125	{rolls/buns}	0.012/09/10
## [796] {rolls/buns,	125		
## [/96] { OIIS/DUNS; ## whole milk}	->	<pre>{root vegetables}</pre>	0.012709710
0.2244165 0.056634469 2.0588959	125	{root vegetables}	0.012/09/10
## [797] {other vegetables,	123		
_	->	{whole milk}	0.023182511
0.4892704 0.047381800 1.9148326	228	(WHOLE WILK)	0.023102311
## [798] {root vegetables,	220		
## whole milk}	=>	<pre>{other vegetables}</pre>	0 023182511
0.4740125 0.048906965 2.4497702	228	(other vegetables)	0.023102311
## [799] {other vegetables,	220		
## whole milk}	=>	<pre>{root vegetables}</pre>	0.023182511
0.3097826 0.074834774 2.8420820	228	(, 555 , 585 , 555)	0101010101
## [800] {soda,			
## yogurt}	=>	{rolls/buns}	0.008642603
0.3159851 0.027351296 1.7179181	85	(. ===, ==)	
## [801] {rolls/buns,			
## soda}	=>	{yogurt}	0.008642603
0.2254642 0.038332486 1.6162101	85		
## [802] {rolls/buns,			
## yogurt}	=>	{soda}	0.008642603
0.2514793 0.034367056 1.4421567	85		
## [803] {soda,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.008337570
0.3048327 0.027351296 1.5754229	82		
## [804] {other vegetables,			
## soda}	=>	<pre>{yogurt}</pre>	0.008337570
0.2546584 0.032740214 1.8254849	82		
## [805] {soda,			
## yogurt}	=>	{whole milk}	0.010472801
0.3828996 0.027351296 1.4985348	103		
## [806] {soda,			
## whole milk}		{yogurt}	0.010472801
0.2614213 0.040061007 1.8739641	103		
## [807] {rolls/buns,			
## soda}		<pre>{other vegetables}</pre>	0.009862735
0.2572944 0.038332486 1.3297376	97		
## [808] {other vegetables,			
## soda}		<pre>{rolls/buns}</pre>	0.009862735
0.3012422 0.032740214 1.6377653	97		
## [809] {other vegetables,		()	0.000060707
## rolls/buns}		{soda}	0.009862735
0.2315036 0.042602949 1.3276022	97		
## [810] {rolls/buns,			

	0.038332486 0.90314		{whole milk}	0.008845958
	whole milk} 0.040061007 1.20049		{rolls/buns}	0.008845958
##	<pre>(other vegetables, soda) 0.032740214 1.66512</pre>		{whole milk}	0.013929842
## [813] { ##	[soda, whole milk}	=>	{other vegetables}	0.013929842
	0.040061007 1.79704 [rolls/buns, vogurt}		<pre>{other vegetables}</pre>	0.011489578
0.3343195 ## [815] {	0.034367056 1.72781 other vegetables,	53 113		
	yogurt} 0.043416370 1.43875 other vegetables,		{rolls/buns}	0.011489578
## 0.2696897	rolls/buns} 0.042602949 1.93323		{yogurt}	0.011489578
##	[rolls/buns, yogurt} 0.034367056 1.77156		{whole milk}	0.015556685
## [818] { ##	<pre>[whole milk, yogurt}</pre>	=>	{rolls/buns}	0.015556685
## [819]	<pre>0.056024403 1.50964 [rolls/buns, whole milk}</pre>		{yogurt}	0.015556685
0.2746858	0.056634469 1.96904 Other vegetables,		(yogur c)	0.015550005
	yogurt} 0.043416370 2.00723 whole milk,		{whole milk}	0.022267412
## 0.3974592	yogurt} 0.056024403 2.05413		{other vegetables}	0.022267412
##	<pre>(other vegetables, whole milk) 0.074834774 2.13297</pre>		{yogurt}	0.022267412
## [823] {	<pre>[other vegetables, rolls/buns}</pre>	=>	{whole milk}	0.017895272
	0.042602949 1.64391 [rolls/buns, whole milk}		<pre>{other vegetables}</pre>	0.017895272
0.3159785	0.056634469 1.63302 other vegetables,		(orner Aegeraptes)	0.01/0332/2
	whole milk} 0.074834774 1.30008 [fruit/vegetable jui	17 176	{rolls/buns}	0.017895272
## ##	other vegetables, yogurt}		{whole milk}	0.005083884

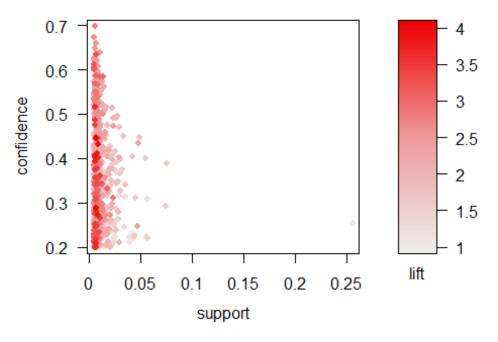
<pre>0.6172840 0.008235892 2.4158327 ## [827] {fruit/vegetable juice, ## whole milk,</pre>	50		
## yogurt} 0.5376344 0.009456024 2.7785782	=> 50	<pre>{other vegetables}</pre>	0.005083884
<pre>## [828] {fruit/vegetable juice,</pre>	30		
<pre>## other vegetables, ## whole milk}</pre>	=>	{yogurt}	0.005083884
0.4854369 0.010472801 3.4797900 ## [829] {other vegetables,	50		
## whole milk,		((()))	0.005000004
## yogurt} 0.2283105 0.022267412 3.1581347	=> 50	<pre>{fruit/vegetable juice}</pre>	0.005083884
<pre>## [830] {other vegetables, ## root vegetables,</pre>			
## whipped/sour cream}		<pre>{whole milk}</pre>	0.005185562
0.6071429 0.008540925 2.3761441 ## [831] {root vegetables,	51		
<pre>## whipped/sour cream, ## whole milk}</pre>	=>	<pre>{other vegetables}</pre>	0.005185562
0.5483871 0.009456024 2.8341498	51	(come regress)	
<pre>## [832] {other vegetables, ## whipped/sour cream,</pre>			
## whole milk} 0.3541667 0.014641586 3.2492809	=> 51	<pre>{root vegetables}</pre>	0.005185562
<pre>## [833] {other vegetables, ## root vegetables,</pre>			
## whole milk}		{whipped/sour cream}	0.005185562
0.2236842 0.023182511 3.1204741 ## [834] {other vegetables,	51		
<pre>## whipped/sour cream, ## yogurt}</pre>	-\	{whole milk}	0.005592272
0.5500000 0.010167768 2.1525070	55	/MILOTE HITTK}	0.003392272
<pre>## [835] {whipped/sour cream, ## whole milk,</pre>			
## yogurt} 0.5140187 0.010879512 2.6565286	=> 55	<pre>{other vegetables}</pre>	0.005592272
<pre>## [836] {other vegetables,</pre>	رر		
<pre>## whipped/sour cream, ## whole milk}</pre>	=>	{yogurt}	0.005592272
0.3819444 0.014641586 2.7379181 ## [837] {other vegetables,	55		
## whole milk,			0.005500000
## yogurt} 0.2511416 0.022267412 3.5035137	=> 55	{whipped/sour cream}	0.005592272
<pre>## [838] {other vegetables, ## pip fruit,</pre>			
<pre>## root vegetables}</pre>		{whole milk}	0.005490595
0.6750000 0.008134215 2.6417131 ## [839] {pip fruit,	54		

##	root vegetables, whole milk}		{other vegetables}	0.005490595
## [840]	0.008947636 3.1713682 {other vegetables, pip fruit,	54		
##	whole milk} 0.013523132 3.7249607	=> 54	<pre>{root vegetables}</pre>	0.005490595
## [841]	{other vegetables, root vegetables,	<i>3</i> .		
##	whole milk} . 0.023182511 3.1308362	=> 54	{pip fruit}	0.005490595
##	<pre>{other vegetables, pip fruit,</pre>			
0.6250000	yogurt} 0 0.008134215 2.4460306	=> 50	{whole milk}	0.005083884
##	<pre>{pip fruit, whole milk,</pre>		(ath an are satable 1 a 2)	0.005002004
0.5319149	yogurt} 0.009557702 2.7490189	=> 50	{other vegetables}	0.005083884
##	<pre>{other vegetables, pip fruit, whole milk}</pre>	->	{yogurt}	0.005083884
0.3759398	3 0.013523132 2.6948749 {other vegetables,	50	(yogur e)	0.003003004
	whole milk, yogurt}	=>	{pip fruit}	0.005083884
	0.022267412 3.0180562 {citrus fruit,	50		
##	other vegetables, root vegetables}		{whole milk}	0.005795628
## [847]	0.010371124 2.1870392 {citrus fruit,	57		
##	root vegetables, whole milk} 0.009150991 3.2731652	=> 57	{other vegetables}	0.005795628
	{citrus fruit, other vegetables,	37		
##	whole milk} 5 0.013014743 4.0854929	=> 57	<pre>{root vegetables}</pre>	0.005795628
## [849] ##	<pre>{other vegetables, root vegetables,</pre>			
	whole milk} 0.023182511 3.0205774	=> 57	{citrus fruit}	0.005795628
##	{root vegetables, tropical fruit,		(uhala millo	0.005603050
	yogurt} 0.008134215 2.7395543	=> 56	{whole milk}	0.005693950
## ## [851]	<pre>{root vegetables, tropical fruit, whole milk}</pre>	=>	{yogurt}	0.005693950
	miore mark)	-/	(Jobai c)	3.00505550

0.4745763 0.011997966 3.4019370 ## [852] {tropical fruit, ## whole milk,	56	
<pre>## yogurt} 0.3758389 0.015149975 3.4481118 ## [853] {root vegetables, ## whole milk,</pre>	<pre>=> {root vegetables} 56</pre>	0.005693950
## whole milk, ## yogurt} 0.3916084 0.014539908 3.7320432 ## [854] {other vegetables,	<pre>=> {tropical fruit} 56</pre>	0.005693950
<pre>## root vegetables, ## tropical fruit} 0.5702479 0.012302999 2.2317503 ## [855] {root vegetables,</pre>	<pre>=> {whole milk} 69</pre>	0.007015760
<pre>## tropical fruit, ## whole milk} 0.5847458 0.011997966 3.0220571</pre>	<pre>=> {other vegetables} 69</pre>	0.007015760
<pre>## [856] {other vegetables, ## tropical fruit, ## whole milk} 0.4107143 0.017081851 3.7680737</pre>	<pre>=> {root vegetables} 69</pre>	0.007015760
## [857] {other vegetables, ## root vegetables, ## whole milk}	=> {tropical fruit}	0.007015760
0.3026316 0.023182511 2.8840907 ## [858] {other vegetables, ## tropical fruit,	69	
<pre>## yogurt} 0.6198347 0.012302999 2.4258155 ## [859] {tropical fruit, ## whole milk,</pre>	=> {whole milk} 75	0.007625826
## yogurt} 0.5033557 0.015149975 2.6014206 ## [860] {other vegetables,	<pre>=> {other vegetables} 75</pre>	0.007625826
## tropical fruit, ## whole milk} 0.4464286 0.017081851 3.2001640	=> {yogurt} 75	0.007625826
## [861] {other vegetables, ## whole milk, ## yogurt}	=> {tropical fruit}	0.007625826
<pre>0.3424658 0.022267412 3.2637119 ## [862] {other vegetables, ## root vegetables, ## yogurt}</pre>	75 => {whole milk}	0.007829181
0.6062992 0.012913066 2.3728423 ## [863] {root vegetables, ## whole milk,	77	0.007.023101
## yogurt} 0.5384615 0.014539908 2.7828530 ## [864] {other vegetables,	<pre>=> {other vegetables} 77</pre>	0.007829181

## [865]	root vegetables, whole milk} 0.023182511 2.4208960 {other vegetables,	=> 77	{yogurt}	0.007829181
## [866]	whole milk, yogurt} 0.022267412 3.2257165 {other vegetables,	=> 77	{root vegetables}	0.007829181
## 0.5083333 ## [867]	rolls/buns, root vegetables} 0.012201322 1.9894383 {rolls/buns,	=> 61	{whole milk}	0.006202339
## 0.4880000 ## [868]	root vegetables, whole milk} 0.012709710 2.5220599 {other vegetables,	=> 61	{other vegetables}	0.006202339
	root vegetables, whole milk} 0.023182511 1.4545571 {other vegetables,	=> 61	{rolls/buns}	0.006202339
## 0.3465909	rolls/buns, whole milk} 0.017895272 3.1797776 {other vegetables,	=> 61	<pre>{root vegetables}</pre>	0.006202339
## 0.5221239	rolls/buns, yogurt} 0.011489578 2.0434097 {rolls/buns,	=> 59	{whole milk}	0.005998983
	whole milk, yogurt} 0.015556685 1.9929489 {other vegetables,	=> 59	{other vegetables}	0.005998983
## [873]	whole milk, yogurt} 0.022267412 1.4646832 {other vegetables,	=> 59	{rolls/buns}	0.005998983
## ## 0.3352273	rolls/buns, whole milk} 0.017895272 2.4030322	=> 59	{yogurt}	0.005998983
<pre>plot(bask</pre>	etrules)			

Scatter plot for 873 rules



```
## Choose a subset
inspect(subset(basketrules, subset=lift > 3))
##
        1hs
                                   rhs
                                                               support
confidence
              coverage
                           lift count
## [1] {herbs}
                                => {root vegetables}
                                                           0.007015760
0.4312500 0.016268429 3.956477
                                  69
## [2] {sliced cheese}
                                => {sausage}
                                                           0.007015760
0.2863071 0.024504321 3.047435
                                  69
                                => {whipped/sour cream}
## [3] {berries}
                                                           0.009049314
0.2721713 0.033248602 3.796886
## [4] {beef}
                                => {root vegetables}
                                                           0.017386884
0.3313953 0.052465684 3.040367
                                 171
## [5] {onions,
                                => {other vegetables}
##
         root vegetables}
                                                           0.005693950
0.6021505 0.009456024 3.112008
                                  56
## [6]
       {onions,
         other vegetables}
                                => {root vegetables}
                                                           0.005693950
##
0.4000000 0.014234875 3.669776
                                  56
## [7]
       {chicken,
                                => {root vegetables}
         whole milk}
                                                           0.005998983
0.3410405 0.017590239 3.128855
                                  59
## [8] {frozen vegetables,
                                => {root vegetables}
         other vegetables}
                                                           0.006100661
0.3428571 0.017793594 3.145522
                                  60
## [9] {beef,
       other vegetables} => {root vegetables}
                                                           0.007930859
```

0.4020619 0.019725470 3.688692 ## [10] {beef,		
## whole milk}	<pre>=> {root vegetables}</pre>	0.008032537
0.3779904 0.021250635 3.467851 ## [11] {curd,	79	
## whole milk}	=> {whipped/sour cream}	0.005897306
0.2256809 0.026131164 3.148329		
<pre>## [12] {curd, ## tropical fruit}</pre>	=> {vogurt}	0.005287239
0.5148515 0.010269446 3.690645	52	
## [13] {margarine,	-> (domostic oggs)	0 005105560
## whole milk} 0.2142857 0.024199288 3.377404	=> {domestic eggs} 51	0.003183302
## [14] {butter,		
## whole milk} 0.2177122 0.027554652 3.431409	<pre>=> {domestic eggs} 59</pre>	0.005998983
## [15] {domestic eggs,	39	
## whole milk}	=> {butter}	0.005998983
0.2000000 0.029994916 3.609174 ## [16] {butter,	59	
<pre>## other vegetables}</pre>		0.005795628
0.2893401 0.020030503 4.036397	57	
<pre>## [17] {other vegetables, ## whipped/sour cream}</pre>	=> {butter}	0.005795628
0.2007042 0.028876462 3.621883	57	
## [18] {butter, ## whole milk}	-> (whinned/soun speam)	0 006710727
0.2435424 0.027554652 3.397503		0.000/10/2/
## [19] {whipped/sour cream,		
## whole milk} 0.2082019 0.032231825 3.757185		0.006710727
## [20] {butter,	00	
<pre>## other vegetables}</pre>		0.006609049
0.3299492 0.020030503 3.027100 ## [21] {domestic eggs,	65	
## other vegetables}	<pre>=> {whipped/sour cream}</pre>	0.005083884
0.2283105 0.022267412 3.185012	50	
<pre>## [22] {domestic eggs, ## other vegetables}</pre>	<pre>=> {root vegetables}</pre>	0.007320793
0.3287671 0.022267412 3.016254	72	
<pre>## [23] {pip fruit, ## whipped/sour cream}</pre>	<pre>=> {other vegetables}</pre>	0.005592272
0.6043956 0.009252669 3.123610	55	0.003332272
## [24] {tropical fruit,		
## whipped/sour cream} 0.4485294 0.013828165 3.215224	=> {yogurt} 61	0.006202339
## [25] {other vegetables,	-	
## tropical fruit}	<pre>=> {whipped/sour cream}</pre>	0.007829181
0.2181303 0.035892222 3.042995 ## [26] {root vegetables,	77	

<pre>## yogurt} 0.2480315 0.025826131 3.460127 ## [27] {other vegetables,</pre>		0.006405694
## yogurt} 0.2341920 0.043416370 3.267062	<pre>=> {whipped/sour cream} 100</pre>	0.010167768
## [28] {citrus fruit, ## pip fruit} 0.4044118 0.013828165 3.854060	<pre>=> {tropical fruit} 55</pre>	0.005592272
## [29] {pip fruit, ## tropical fruit} 0.2736318 0.020437214 3.306105		0.005592272
## [30] {citrus fruit, ## tropical fruit} 0.2806122 0.019928826 3.709437		0.005592272
## [31] {pip fruit, ## root vegetables} 0.3398693 0.015556685 3.238967		0.005287239
<pre>## [32] {root vegetables, ## tropical fruit} 0.2512077 0.021047280 3.320737</pre>		0.005287239
<pre>## [33] {pip fruit, ## yogurt} 0.3559322 0.017996950 3.392048</pre>	<pre>=> {tropical fruit} 63</pre>	0.006405694
<pre>## [34] {other vegetables, ## pip fruit} 0.3618677 0.026131164 3.448613</pre>	<pre>=> {tropical fruit} 93</pre>	0.009456024
<pre>## [35] {other vegetables, ## tropical fruit} 0.2634561 0.035892222 3.482649</pre>		0.009456024
<pre>## [36] {citrus fruit, ## root vegetables} 0.3218391 0.017691917 3.067139</pre>		0.005693950
<pre>## [37] {root vegetables, ## tropical fruit} 0.2705314 0.021047280 3.268644</pre>		0.005693950
<pre>## [38] {other vegetables, ## tropical fruit} 0.2521246 0.035892222 3.046248</pre>	=> {citrus fruit} 89	0.009049314
<pre>## [39] {citrus fruit, ## root vegetables} 0.5862069 0.017691917 3.029608</pre>	,	0.010371124
<pre>## [40] {citrus fruit, ## other vegetables} 0.3591549 0.028876462 3.295045</pre>	• •	0.010371124
## [41] {rolls/buns, ## shopping bags} 0.3072917 0.019522115 3.270794	=> {sausage}	0.005998983
## [42] {root vegetables, ## yogurt} 0.3149606 0.025826131 3.001587	<pre>=> {tropical fruit}</pre>	0.008134215
0.3143000 0.023020131 3.00130/	00	

## [43] {root vegetables, ## tropical fruit} 0.5845411 0.021047280 3.020999	<pre>=> {other vegetables} 121</pre>	0.012302999
<pre>## [44] {other vegetables, ## tropical fruit} 0.3427762 0.035892222 3.144780 ## [45] {fruit/vegetable juice, ## property seems to be a seem to</pre>	<pre>=> {root vegetables} 121</pre>	0.012302999
## other vegetables, ## whole milk} 0.4854369 0.010472801 3.479790 ## [46] {other vegetables,	=> {yogurt} 50	0.005083884
## whole milk, ## yogurt} 0.2283105 0.022267412 3.158135 ## [47] {other vegetables,	<pre>=> {fruit/vegetable juice} 50</pre>	0.005083884
## whipped/sour cream, ## whole milk} 0.3541667 0.014641586 3.249281 ## [48] {other vegetables,	<pre>=> {root vegetables} 51</pre>	0.005185562
<pre>## root vegetables, ## whole milk} 0.2236842 0.023182511 3.120474 ## [49] {other vegetables,</pre>	<pre>=> {whipped/sour cream} 51</pre>	0.005185562
<pre>## whole milk, ## yogurt} 0.2511416 0.022267412 3.503514 ## [50] {pip fruit,</pre>	<pre>=> {whipped/sour cream} 55</pre>	0.005592272
<pre>## root vegetables, ## whole milk} 0.6136364 0.008947636 3.171368 ## [51] {other vegetables,</pre>	<pre>=> {other vegetables} 54</pre>	0.005490595
<pre>## pip fruit, ## whole milk} 0.4060150 0.013523132 3.724961 ## [52] {other vegetables,</pre>	<pre>=> {root vegetables} 54</pre>	0.005490595
<pre>## root vegetables, ## whole milk} 0.2368421 0.023182511 3.130836 ## [53] {other vegetables,</pre>	=> {pip fruit} 54	0.005490595
## whole milk, ## yogurt} 0.2283105 0.022267412 3.018056 ## [54] {citrus fruit,	=> {pip fruit} 50	0.005083884
<pre>## root vegetables, ## whole milk} 0.6333333 0.009150991 3.273165 ## [55] {citrus fruit,</pre>	<pre>=> {other vegetables} 57</pre>	0.005795628
## other vegetables, ## whole milk} 0.4453125 0.013014743 4.085493	<pre>=> {root vegetables} 57</pre>	0.005795628

```
## [56] {other vegetables,
##
         root vegetables,
##
         whole milk}
                                => {citrus fruit}
                                                            0.005795628
0.2500000 0.023182511 3.020577
                                  57
## [57] {root vegetables,
         tropical fruit,
##
         whole milk}
                                => {yogurt}
                                                            0.005693950
0.4745763 0.011997966 3.401937
                                  56
## [58] {tropical fruit,
##
         whole milk,
                                => {root vegetables}
##
         yogurt}
                                                            0.005693950
0.3758389 0.015149975 3.448112
                                  56
## [59] {root vegetables,
         whole milk,
##
                                => {tropical fruit}
                                                            0.005693950
         yogurt}
0.3916084 0.014539908 3.732043
## [60] {root vegetables,
##
         tropical fruit,
                                => {other vegetables}
##
         whole milk}
                                                            0.007015760
                                  69
0.5847458 0.011997966 3.022057
## [61] {other vegetables,
         tropical fruit,
                                => {root vegetables}
##
         whole milk}
                                                            0.007015760
0.4107143 0.017081851 3.768074
## [62] {other vegetables,
##
         tropical fruit,
##
         whole milk}
                                => {yogurt}
                                                            0.007625826
0.4464286 0.017081851 3.200164
                                  75
## [63] {other vegetables,
##
         whole milk,
                                => {tropical fruit}
##
         yogurt}
                                                            0.007625826
0.3424658 0.022267412 3.263712
                                  75
## [64] {other vegetables,
##
         whole milk,
##
                                => {root vegetables}
         yogurt}
                                                            0.007829181
0.3515982 0.022267412 3.225716
                                  77
## [65] {other vegetables,
##
         rolls/buns,
         whole milk}
                                => {root vegetables}
                                                            0.006202339
0.3465909 0.017895272 3.179778
                                  61
inspect(subset(basketrules, subset=confidence > 0.5))
##
         1hs
                                        rhs
                                                               support
              coverage
confidence
                           lift count
        {baking powder}
                                     => {whole milk}
                                                           0.009252669
0.5229885 0.017691917 2.046793
## [2]
         {oil,
          other vegetables}
                                     => {whole milk}
                                                           0.005083884
0.5102041 0.009964413 1.996760
                                   50
```

```
## [3]
        {onions,
##
          root vegetables}
                                   => {other vegetables} 0.005693950
0.6021505 0.009456024 3.112008
                                  56
## [4]
       {onions,
         whole milk}
##
                                   => {other vegetables} 0.006609049
0.5462185 0.012099644 2.822942
                                  65
## [5] {hygiene articles,
          other vegetables}
                                   => {whole milk}
                                                         0.005185562
0.5425532 0.009557702 2.123363
## [6]
        {other vegetables,
                                   => {whole milk}
##
          sugar}
                                                         0.006304016
0.5849057 0.010777834 2.289115
                                  62
## [7] {long life bakery product,
                                   => {whole milk}
         other vegetables}
                                                         0.005693950
0.5333333 0.010676157 2.087279
                                  56
## [8] {cream cheese,
##
         yogurt}
                                   => {whole milk}
                                                         0.006609049
0.5327869 0.012404677 2.085141
                                  65
## [9]
         {chicken,
##
          root vegetables}
                                   => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829
                                  56
## [10] {chicken,
                                   => {whole milk}
          root vegetables}
                                                         0.005998983
0.5514019 0.010879512 2.157993
## [11] {chicken,
          rolls/buns}
                                   => {whole milk}
                                                         0.005287239
0.5473684 0.009659380 2.142208
## [12] {coffee,
                                   => {whole milk}
                                                         0.005083884
         yogurt}
0.5208333 0.009761057 2.038359
                                  50
## [13] {frozen vegetables,
         root vegetables}
                                   => {other vegetables} 0.006100661
0.5263158 0.011591256 2.720082
                                  60
## [14] {frozen vegetables,
                                   => {whole milk}
##
          root vegetables}
                                                         0.006202339
0.5350877 0.011591256 2.094146
                                 61
## [15] {frozen vegetables,
##
          other vegetables}
                                   => {whole milk}
                                                         0.009659380
0.5428571 0.017793594 2.124552
## [16] {beef,
         yogurt}
                                   => {whole milk}
                                                         0.006100661
0.5217391 0.011692933 2.041904
                                  60
## [17] {curd,
          whipped/sour cream}
                                   => {whole milk}
                                                         0.005897306
0.5631068 0.010472801 2.203802
## [18] {curd,
         tropical fruit}
                                   => {yogurt}
                                                         0.005287239
0.5148515 0.010269446 3.690645
                                 52
## [19] {curd,
## tropical fruit} => {other vegetables} 0.005287239
```

0.5148515 0.010269446 2.660833 ## [20] {curd,	52		
## tropical fruit} 0.6336634 0.010269446 2.479936	=> 64	{whole milk} 0.006507372	
## [21] {curd,			
## root vegetables} 0.5046729 0.010879512 2.608228	=> 54	{other vegetables} 0.005490595	
<pre>## [22] {curd, ## root vegetables}</pre>	=>	{whole milk} 0.006202339	
0.5700935 0.010879512 2.231146	61	(WHOLE WILK) 0.000202333	
## [23] {curd, ## yogurt}	=>	{whole milk} 0.010066090	
0.5823529 0.017285206 2.279125 ## [24] {curd,	99		
## rolls/buns}		{whole milk} 0.005897306	
0.5858586 0.010066090 2.292845 ## [25] {curd,	58		
## other vegetables} 0.5739645 0.017183528 2.246296	=> 97	{whole milk} 0.009862735	
<pre>## [26] {pork, ## root vegetables}</pre>	->	{other vegetables} 0.007015760	
0.5149254 0.013624809 2.661214	69	(other vegetables) 0.00/015/00	
## [27] {pork, ## rolls/buns}	=>	{whole milk} 0.006202339	
0.5495495 0.011286223 2.150744 ## [28] {frankfurter,	61		
## tropical fruit} 0.5483871 0.009456024 2.146195	=> 51	{whole milk} 0.005185562	
## [29] {frankfurter,			
## yogurt} 0.5545455 0.011184545 2.170296	=> 61	{whole milk} 0.006202339	
<pre>## [30] {bottled beer, ## yogurt}</pre>	=>	{whole milk} 0.005185562	
0.5604396 0.009252669 2.193364	51	(
<pre>## [31] {brown bread, ## tropical fruit}</pre>		{whole milk} 0.005693950	
0.5333333 0.010676157 2.087279 ## [32] {brown bread,	56		
## root vegetables} 0.5600000 0.010167768 2.191643	=> 56	{whole milk} 0.005693950	
<pre>## [33] {domestic eggs, ## margarine}</pre>		{whole milk} 0.005185562	
0.6219512 0.008337570 2.434099	51	(WHOLE WILK) 0.005105502	
<pre>## [34] {margarine, ## root vegetables}</pre>	=>	{other vegetables} 0.005897306	
0.5321101 0.011082867 2.750028 ## [35] {margarine,	58		
## rolls/buns} 0.5379310 0.014743264 2.105273	=> 78	{whole milk} 0.007930859	
## [36] {butter,	, 0		

0.6210526	domestic eggs} 0.009659380 2.430582	=> 59	{whole	milk}	0.005998983	
	<pre>{butter, whipped/sour cream} 0.010167768 2.945849</pre>	=> 57	{other	vegetables}	0.005795628	
	{butter, whipped/sour cream} 0.010167768 2.583008	=> 66	{whole	milk}	0.006710727	
## [39] ##	{butter, citrus fruit}	=>	{whole	milk}	0.005083884	
	0.009150991 2.174249 {bottled water, butter}	50 =>	{whole	milk}	0.005388917	
## [41]	·	53		-	0.005400505	
	tropical fruit} 0.009964413 2.847759 {butter,	=> 54	{otner	vegetables}	0.005490595	
0.6224490	tropical fruit} 0.009964413 2.436047	=> 61	{whole	milk}	0.006202339	
0.5118110	root vegetables} 0.012913066 2.645119	=> 65	{other	vegetables}	0.006609049	
	{butter, root vegetables} 0.012913066 2.496107	=> 81	{whole	milk}	0.008235892	
## [45] ##	{butter, yogurt}	=>	{whole	milk}	0.009354347	
## [46]	0.014641586 2.500387 {butter, other vegetables}	92	{whole	milk}	0.011489578	
0.5736041 ## [47]	0.020030503 2.244885 {newspapers,	113	•	Š		
0.5221239	root vegetables} 0.011489578 2.698417 {newspapers,	=> 59	{other	vegetables}	0.005998983	
## 0.5044248	root vegetables} 0.011489578 1.974142	=> 57	{whole	milk}	0.005795628	
##	{domestic eggs, whipped/sour cream} 0.009964413 2.636814	=> 50	{other	vegetables}	0.005083884	
##	<pre>{domestic eggs, whipped/sour cream} 0.009964413 2.236371</pre>	=> 56	{whole	milk}	0.005693950	
## [51] ##	{domestic eggs, pip fruit}	=>	{whole	milk}	0.005388917	
## [52]	0.008642603 2.440275 {citrus fruit, domestic eggs}	53 =>	{whole	milk}	0.005693950	
	0.010371124 2.148670	56	(

```
## [53] {domestic eggs,
          tropical fruit}
                                   => {whole milk}
                                                         0.006914082
##
0.6071429 0.011387900 2.376144
                                  68
## [54] {domestic eggs,
                                   => {other vegetables} 0.007320793
##
          root vegetables}
0.5106383 0.014336553 2.639058
                                 72
## [55] {domestic eggs,
          root vegetables}
                                   => {whole milk}
                                                         0.008540925
0.5957447 0.014336553 2.331536
                                  84
## [56] {domestic eggs,
         yogurt}
                                   => {whole milk}
                                                         0.007727504
0.5390071 0.014336553 2.109485
                                 76
## [57] {domestic eggs,
                                   => {whole milk}
          other vegetables}
                                                         0.012302999
0.5525114 0.022267412 2.162336
                                 121
## [58] {fruit/vegetable juice,
##
          root vegetables}
                                   => {other vegetables} 0.006609049
0.5508475 0.011997966 2.846865
                                  65
## [59] {fruit/vegetable juice,
                                   => {whole milk}
##
          root vegetables}
                                                         0.006507372
0.5423729 0.011997966 2.122657
## [60] {fruit/vegetable juice,
                                   => {whole milk}
##
         yogurt}
                                                         0.009456024
0.5054348 0.018708693 1.978094
                                  93
## [61] {pip fruit,
          whipped/sour cream}
                                   => {other vegetables} 0.005592272
0.6043956 0.009252669 3.123610
## [62] {pip fruit,
         whipped/sour cream}
                                   => {whole milk}
                                                         0.005998983
0.6483516 0.009252669 2.537421
## [63] {citrus fruit,
         whipped/sour cream}
                                   => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829
                                  56
## [64] {citrus fruit,
##
          whipped/sour cream}
                                   => {whole milk}
                                                         0.006304016
0.5794393 0.010879512 2.267722
                                  62
## [65] {sausage,
##
         whipped/sour cream}
                                   => {whole milk}
                                                         0.005083884
0.5617978 0.009049314 2.198679
                                  50
## [66] {tropical fruit,
         whipped/sour cream}
                                   => {other vegetables} 0.007829181
0.5661765 0.013828165 2.926088
                                 77
## [67] {tropical fruit,
         whipped/sour cream}
##
                                   => {whole milk}
                                                         0.007930859
0.5735294 0.013828165 2.244593
                                 78
## [68] {root vegetables,
         whipped/sour cream}
                                   => {whole milk}
                                                         0.009456024
0.5535714 0.017081851 2.166484
                                 93
## [69] {whipped/sour cream,
                                   => {whole milk}
                                                         0.010879512
## yogurt}
```

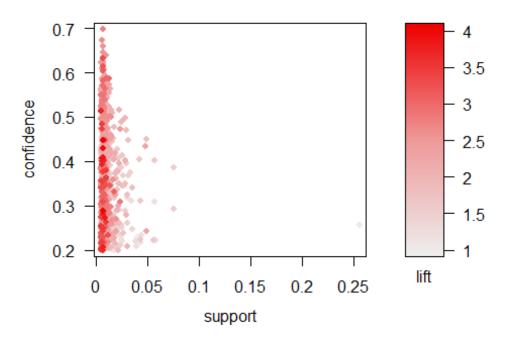
0.5245098 0.020742247 2.052747	107	
<pre>## [70] {rolls/buns, ## whipped/sour cream}</pre>	=>	{whole milk} 0.007829181
0.5347222 0.014641586 2.092715	77	(WHOIC HILK) 0.00/025101
## [71] {other vegetables,		
## whipped/sour cream}	=>	{whole milk} 0.014641586
0.5070423 0.028876462 1.984385	144	
## [72] {pip fruit,		
## sausage}		{whole milk} 0.005592272
0.5188679 0.010777834 2.030667	55	
<pre>## [73] {pip fruit, ## root vegetables}</pre>		{other vegetables} 0.008134215
0.5228758 0.015556685 2.702304	80	Tother vegetables, 0.000134213
## [74] {pip fruit,	00	
## root vegetables}	=>	{whole milk} 0.008947636
0.5751634 0.015556685 2.250988	88	
## [75] {pip fruit,		
## yogurt}		{whole milk} 0.009557702
0.5310734 0.017996950 2.078435	94	
## [76] {other vegetables,		(.h.l
## pip fruit} 0.5175097 0.026131164 2.025351	=> 133	{whole milk} 0.013523132
## [77] {pastry,	133	
## tropical fruit}	=>	{whole milk} 0.006710727
0.5076923 0.013218099 1.986930	66	(111012 11111)
## [78] {pastry,		
<pre>## root vegetables}</pre>	=>	{other vegetables} 0.005897306
0.5370370 0.010981190 2.775491	58	
## [79] {pastry,		
## root vegetables}		{whole milk} 0.005693950
0.5185185 0.010981190 2.029299	56	
## [80] {pastry, ## yogurt}	=>	{whole milk} 0.009150991
0.5172414 0.017691917 2.024301	90	(WHOIC HILK) 0.005150551
## [81] {citrus fruit,	20	
<pre>## root vegetables}</pre>	=>	{other vegetables} 0.010371124
0.5862069 0.017691917 3.029608	102	
## [82] {citrus fruit,		
<pre>## root vegetables}</pre>		{whole milk} 0.009150991
0.5172414 0.017691917 2.024301	90	
<pre>## [83] {root vegetables, ## shopping bags}</pre>		(athon vogatables) a aassaaaa
## shopping bags} 0.5158730 0.012811388 2.666112	65	{other vegetables} 0.006609049
## [84] {sausage,	05	
## tropical fruit}	=>	{whole milk} 0.007219115
0.5182482 0.013929842 2.028241	71	,
<pre>## [85] {root vegetables,</pre>		
## sausage}		{whole milk} 0.007727504
0.5170068 0.014946619 2.023383	76	
## [86] {root vegetables,		

```
## tropical fruit}
                                    => {other vegetables} 0.012302999
0.5845411 0.021047280 3.020999
                                 121
## [87] {root vegetables,
         tropical fruit}
                                    => {whole milk}
                                                          0.011997966
0.5700483 0.021047280 2.230969
                                 118
## [88] {tropical fruit,
                                    => {whole milk}
##
                                                          0.015149975
         yogurt}
                                 149
0.5173611 0.029283172 2.024770
## [89] {root vegetables,
##
                                    => {whole milk}
                                                          0.014539908
          yogurt }
0.5629921 0.025826131 2.203354
                                 143
## [90] {rolls/buns,
          root vegetables}
                                    => {other vegetables} 0.012201322
0.5020921 0.024300966 2.594890
                                 120
## [91] {rolls/buns,
          root vegetables}
                                    => {whole milk}
                                                          0.012709710
0.5230126 0.024300966 2.046888
                                 125
## [92] {other vegetables,
##
          yogurt }
                                    => {whole milk}
                                                          0.022267412
0.5128806 0.043416370 2.007235
                                 219
## [93] {fruit/vegetable juice,
##
          other vegetables,
                                    => {whole milk}
##
          yogurt }
                                                          0.005083884
0.6172840 0.008235892 2.415833
                                  50
## [94] {fruit/vegetable juice,
##
         whole milk,
##
         yogurt }
                                    => {other vegetables} 0.005083884
0.5376344 0.009456024 2.778578
                                  50
## [95] {other vegetables,
##
          root vegetables,
                                   => {whole milk}
##
         whipped/sour cream}
                                                          0.005185562
0.6071429 0.008540925 2.376144
                                  51
## [96] {root vegetables,
##
         whipped/sour cream,
##
         whole milk}
                                    => {other vegetables} 0.005185562
0.5483871 0.009456024 2.834150
                                  51
## [97] {other vegetables,
##
         whipped/sour cream,
                                    => {whole milk}
         yogurt}
##
                                                          0.005592272
0.5500000 0.010167768 2.152507
                                  55
## [98] {whipped/sour cream,
##
         whole milk,
##
         yogurt }
                                    => {other vegetables} 0.005592272
0.5140187 0.010879512 2.656529
                                  55
## [99] {other vegetables,
##
          pip fruit,
##
          root vegetables}
                                    => {whole milk}
                                                          0.005490595
0.6750000 0.008134215 2.641713
## [100] {pip fruit,
## root vegetables,
```

```
##
         whole milk}
                                    => {other vegetables} 0.005490595
                                  54
0.6136364 0.008947636 3.171368
## [101] {other vegetables,
          pip fruit,
##
          yogurt}
                                     => {whole milk}
                                                           0.005083884
0.6250000 0.008134215 2.446031
                                   50
## [102] {pip fruit,
##
          whole milk,
          yogurt}
                                    => {other vegetables} 0.005083884
0.5319149 0.009557702 2.749019
                                   50
## [103] {citrus fruit,
          other vegetables,
##
                                    => {whole milk}
                                                           0.005795628
##
          root vegetables}
0.5588235 0.010371124 2.187039
                                  57
## [104] {citrus fruit,
          root vegetables,
##
          whole milk}
                                    => {other vegetables} 0.005795628
0.6333333 0.009150991 3.273165
                                   57
## [105] {root vegetables,
##
          tropical fruit,
##
                                     => {whole milk}
          yogurt }
                                                           0.005693950
0.7000000 0.008134215 2.739554
                                   56
## [106] {other vegetables,
##
          root vegetables,
##
          tropical fruit}
                                    => {whole milk}
                                                           0.007015760
0.5702479 0.012302999 2.231750
                                   69
## [107] {root vegetables,
##
          tropical fruit,
##
          whole milk}
                                     => {other vegetables} 0.007015760
0.5847458 0.011997966 3.022057
                                   69
## [108] {other vegetables,
         tropical fruit,
##
          yogurt }
                                    => {whole milk}
                                                           0.007625826
0.6198347 0.012302999 2.425816
                                  75
## [109] {tropical fruit,
##
          whole milk,
##
          yogurt }
                                    => {other vegetables} 0.007625826
0.5033557 0.015149975 2.601421
                                  75
## [110] {other vegetables,
##
          root vegetables,
##
                                    => {whole milk}
                                                           0.007829181
          yogurt }
0.6062992 0.012913066 2.372842
                                  77
## [111] {root vegetables,
##
          whole milk,
##
          yogurt}
                                     => {other vegetables} 0.007829181
0.5384615 0.014539908 2.782853
                                  77
## [112] {other vegetables,
##
          rolls/buns,
                                    => {whole milk}
##
          root vegetables}
                                                           0.006202339
0.5083333 0.012201322 1.989438
```

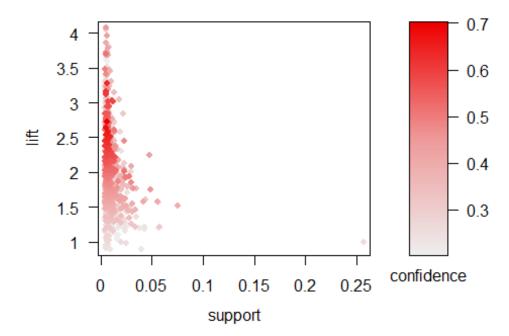
```
## [113] {other vegetables,
##
          rolls/buns,
##
          yogurt }
                                    => {whole milk}
                                                          0.005998983
0.5221239 0.011489578 2.043410
                                  59
inspect(subset(basketrules, subset=lift > 3 & confidence > 0.55))
##
                               rhs
                                                      support confidence
             lift count
coverage
## [1] {onions,
        root vegetables}
                            => {other vegetables} 0.005693950 0.6021505
0.009456024 3.112008
                       56
## [2] {pip fruit,
        whipped/sour cream} => {other vegetables} 0.005592272 0.6043956
0.009252669 3.123610
                        55
## [3] {citrus fruit,
##
        root vegetables}
                            => {other vegetables} 0.010371124 0.5862069
0.017691917 3.029608
                       102
## [4] {root vegetables,
                            => {other vegetables} 0.012302999 0.5845411
        tropical fruit}
0.021047280 3.020999
                     121
## [5] {pip fruit,
##
        root vegetables,
##
                            => {other vegetables} 0.005490595 0.6136364
        whole milk}
0.008947636 3.171368
                        54
## [6] {citrus fruit,
##
        root vegetables,
##
        whole milk}
                            => {other vegetables} 0.005795628 0.6333333
0.009150991 3.273165
                        57
## [7] {root vegetables,
##
        tropical fruit,
##
        whole milk}
                            => {other vegetables} 0.007015760 0.5847458
0.011997966 3.022057
                        69
# plot all the rules in (support, confidence) space
# notice that high lift rules tend to have low support
plot(basketrules)
```

Scatter plot for 873 rules

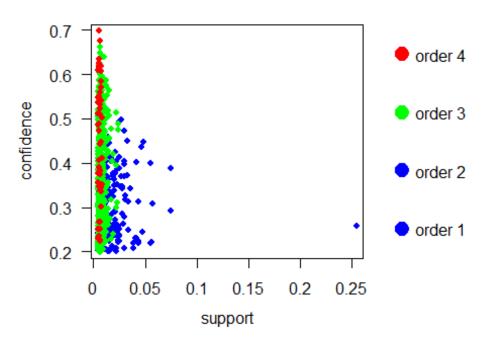


can swap the axes and color scales
plot(basketrules, measure = c("support", "lift"), shading = "confidence")

Scatter plot for 873 rules



Two-key plot



```
# can now look at subsets driven by the plot
inspect(subset(basketrules, support > 0.01))
##
         1hs
                                        rhs
                                                                 support
confidence
                            lift count
             coverage
## [1]
                                     => {whole milk}
                                                              0.25551601
         {}
0.2555160 1.00000000 1.0000000
                                 2513
                                     => {whole milk}
                                                              0.01006609
## [2]
         {hard cheese}
0.4107884 0.02450432 1.6076815
                                   99
       {butter milk}
                                     => {other vegetables}
                                                              0.01037112
## [3]
0.3709091 0.02796136 1.9169159
                                  102
## [4]
         {butter milk}
                                     => {whole milk}
                                                              0.01159126
0.4145455 0.02796136 1.6223854
                                  114
                                                              0.01148958
## [5]
        {ham}
                                     => {whole milk}
0.4414062 0.02602949 1.7275091
                                  113
## [6]
         {sliced cheese}
                                     => {whole milk}
                                                              0.01077783
0.4398340 0.02450432 1.7213560
                                  106
                                     => {whole milk}
                                                              0.01128622
## [7]
         {oil}
0.4021739 0.02806304 1.5739675
                                  111
         {onions}
                                     => {other vegetables}
                                                              0.01423488
## [8]
0.4590164 0.03101169 2.3722681
                                  140
## [9]
         {onions}
                                     => {whole milk}
                                                              0.01209964
0.3901639 0.03101169 1.5269647
                                  119
## [10] {berries}
                                     => {yogurt}
                                                              0.01057448
```

0.3180428 0.03324860 2.2798477 ## [11] {berries}	104	{other	vegetables}	0.01026945
0.3088685 0.03324860 1.5962805	101	(Ocher	vegetables	0.01020545
## [12] {berries}		{whole	milk}	0.01179461
0.3547401 0.03324860 1.3883281 ## [13] {hamburger meat}	116 =>	{other	vegetables}	0.01382816
0.4159021 0.03324860 2.1494470	136	-		
## [14] {hamburger meat}		{whole	milk}	0.01474326
0.4434251 0.03324860 1.7354101 ## [15] {hygiene articles}	145 =>	{whole	milk}	0.01281139
0.3888889 0.03294357 1.5219746	126	(WIIOIC		0.01201133
## [16] {salty snack}		{other	vegetables}	0.01077783
0.2849462 0.03782410 1.4726465	106	{whole	milkl	0.01118454
## [17] {salty snack} 0.2956989 0.03782410 1.1572618	110	(MIIOTE	IIIIK }	0.01110454
## [18] {sugar}		{other	vegetables}	0.01077783
0.3183183 0.03385867 1.6451186	106			
## [19] {sugar} 0.4444444 0.03385867 1.7393996	=> 148	{whole	milk}	0.01504830
## [20] {waffles}		{other	vegetables}	0.01006609
0.2619048 0.03843416 1.3535645	99	(-8	
## [21] {waffles}		{whole	milk}	0.01270971
0.3306878 0.03843416 1.2941961 ## [22] {long life bakery productions of the control of the contr	125	Sothon.	vogotablos?	0.01067616
0.2853261 0.03741739 1.4746096	105	locuei	vegetables	0.01007010
## [23] {long life bakery produc	ct} =>	{whole	milk}	0.01352313
0.3614130 0.03741739 1.4144438	133	6 11		0.04450406
## [24] {dessert} 0.3123288 0.03711235 1.6141636	=> 114	{otner	vegetables}	0.01159126
## [25] {dessert}		{whole	milk}	0.01372649
0.3698630 0.03711235 1.4475140	135	-	-	
## [26] {cream cheese}		{yogurt	t}	0.01240468
0.3128205 0.03965430 2.2424123 ## [27] {cream cheese}	122	{other	vegetables}	0.01372649
0.3461538 0.03965430 1.7889769	135	ionici	vegetables	0.013/2043
## [28] {cream cheese}		$\{ whole \}$	milk}	0.01647178
0.4153846 0.03965430 1.6256696	162	(200+)	(000+2hloc)	0 01007051
## [29] {chicken} 0.2535545 0.04290798 2.3262206	=> 107	froot v	vegetables}	0.01087951
## [30] {chicken}		{other	vegetables}	0.01789527
0.4170616 0.04290798 2.1554393	176	-		
## [31] {chicken}		{whole	milk}	0.01759024
0.4099526 0.04290798 1.6044106 ## [32] {white bread}	173 =>	{soda}		0.01026945
0.2439614 0.04209456 1.3990437	101	(Sour)		0.0101010
## [33] {white bread}		{other	vegetables}	0.01372649
0.3260870 0.04209456 1.6852681	135	(whole	mille)	0 01700105
## [34] {white bread} 0.4057971 0.04209456 1.5881474	=> 168	{whole	IIITTK }	0.01708185
## [35] {chocolate}		{soda}		0.01352313

0.2725410 0.04961871 1.5629391 ## [36] {chocolate}	133 => {re	olls/buns}	0.01179461
0.2377049 0.04961871 1.2923316	116	•	
## [37] {chocolate} 0.2561475 0.04961871 1.3238103	=> {o ¹	ther vegetables}	0.01270971
## [38] {chocolate}	_	hole milk}	0.01667514
0.3360656 0.04961871 1.3152427	164	than wagatablas	0 01242145
## [39] {coffee} 0.2311734 0.05805796 1.1947400	=> (0) 132	ther vegetables}	0.01342145
## [40] {coffee}	_	hole milk}	0.01870869
0.3222417 0.05805796 1.2611408 ## [41] {frozen vegetables}	184 => {re	oot vegetables}	0.01159126
0.2410148 0.04809354 2.2111759	114	ooc vegetables,	0.01133120
## [42] {frozen vegetables} 0.2579281 0.04809354 1.8489235	=> {yo	ogurt}	0.01240468
## [43] {frozen vegetables}		olls/buns}	0.01016777
0.2114165 0.04809354 1.1494092	100	•	
## [44] {frozen vegetables} 0.3699789 0.04809354 1.9121083	=> {o ¹	ther vegetables}	0.01779359
## [45] {frozen vegetables}		hole milk}	0.02043721
0.4249471 0.04809354 1.6630940	201	aat waaatablaa)	0.01730600
## [46] {beef} 0.3313953 0.05246568 3.0403668	=> {rd 171	oot vegetables}	0.01738688
## [47] {beef}		ogurt}	0.01169293
0.2228682 0.05246568 1.5976012 ## [48] {beef}	115 => {r(olls/buns}	0.01362481
0.2596899 0.05246568 1.4118576	134	0113/ 04113 j	0.01302-01
## [49] {beef} 0.3759690 0.05246568 1.9430662	=> {o ¹	ther vegetables}	0.01972547
## [50] {beef}		hole milk}	0.02125064
0.4050388 0.05246568 1.5851795	209	•	
## [51] {curd} 0.2041985 0.05327911 1.8734067	=> {ro	oot vegetables}	0.01087951
## [52] {curd}	=> {ye	ogurt}	0.01728521
0.3244275 0.05327911 2.3256154 ## [53] {curd}	170 -> \overline{1}	ther vegetables}	0 01719353
0.3225191 0.05327911 1.6668288	169	cher vegetables;	0.01/18555
## [54] {curd}	-	hole milk}	0.02613116
0.4904580 0.05327911 1.9194805 ## [55] {napkins}	257 => {so	oda}	0.01199797
0.2291262 0.05236401 1.3139687	118	•	
## [56] {napkins} 0.2349515 0.05236401 1.6842183	=> {yo	ogurt}	0.01230300
## [57] {napkins}		olls/buns}	0.01169293
0.2233010 0.05236401 1.2140216	115	+h	0.01442022
## [58] {napkins} 0.2757282 0.05236401 1.4250060	=> {0 ¹ 142	ther vegetables}	0.01443823
## [59] {napkins}	=> {wl	hole milk}	0.01972547
0.3766990 0.05236401 1.4742678 ## [60] {pork}	194 => {r	oot vegetables}	0.01362481
"" [00] (bork)	-> (1 (out repetubles,	J. J

0.2363316 0.05765125 2.1682099 ## [61] {pork}	134 =>	{soda}	0.01189629
0.2063492 0.05765125 1.1833495	117		
## [62] {pork} 0.3756614 0.05765125 1.9414764	=> 213	<pre>{other vegetables}</pre>	0.02165735
## [63] {pork}	_	{whole milk}	0.02216573
0.3844797 0.05765125 1.5047187	218	(miore mark)	0.02220373
## [64] {frankfurter}		<pre>{rolls/buns}</pre>	0.01921708
0.3258621 0.05897306 1.7716161	189	(athon vocatables)	0.01647170
## [65] {frankfurter} 0.2793103 0.05897306 1.4435193	=> 162	{other vegetables}	0.01647178
## [66] {frankfurter}		{whole milk}	0.02053889
0.3482759 0.05897306 1.3630295	202	,	
## [67] {bottled beer}		{soda}	0.01698017
0.2108586 0.08052872 1.2092094	167	(athanatablaa)	0.01616675
## [68] {bottled beer} 0.2007576 0.08052872 1.0375464	=> 159	{other vegetables}	0.01616675
## [69] {bottled beer}		{whole milk}	0.02043721
0.2537879 0.08052872 0.9932367	201	(
## [70] {brown bread}		<pre>{yogurt}</pre>	0.01453991
0.2241379 0.06487036 1.6067030	143		0.04070060
## [71] {brown bread} 0.2884013 0.06487036 1.4905025	=> 184	{other vegetables}	0.01870869
## [72] {brown bread}	_	{whole milk}	0.02521607
0.3887147 0.06487036 1.5212930	248	(0102222007
## [73] {margarine}	=>	{yogurt}	0.01423488
0.2430556 0.05856634 1.7423115	140	(11 ())	0 04 47 4226
## [74] {margarine} 0.2517361 0.05856634 1.3686151	=> 145	{rolls/buns}	0.01474326
## [75] {margarine}		{other vegetables}	0.01972547
0.3368056 0.05856634 1.7406635	194	(000. 10800020)	0102272017
## [76] {margarine}		<pre>{whole milk}</pre>	0.02419929
0.4131944 0.05856634 1.6170980	238	(, , , , , , , , , , , , , , , , , , ,	0.04204207
## [77] {butter} 0.2330275 0.05541434 2.1378971	=> 127	<pre>{root vegetables}</pre>	0.01291307
## [78] {butter}		{yogurt}	0.01464159
0.2642202 0.05541434 1.8940273	144	() -6)	
## [79] {butter}		<pre>{rolls/buns}</pre>	0.01342145
0.2422018 0.05541434 1.3167800	132	()	
## [80] {butter} 0.3614679 0.05541434 1.8681223	=> 197	<pre>{other vegetables}</pre>	0.02003050
## [81] {butter}		{whole milk}	0.02755465
0.4972477 0.05541434 1.9460530	271	(MIOLE MILLY)	0.02,33.03
	=>	<pre>{rolls/buns}</pre>	0.01972547
## [82] {newspapers}			
0.2471338 0.07981698 1.3435934	194		
0.2471338 0.07981698 1.3435934 ## [83] {newspapers}	=>	{other vegetables}	0.01931876
0.2471338 0.07981698 1.3435934 ## [83] {newspapers} 0.2420382 0.07981698 1.2508912	=> 190		
0.2471338 0.07981698 1.3435934 ## [83] {newspapers}	=> 190	<pre>{other vegetables} {whole milk}</pre>	0.01931876 0.02735130
<pre>0.2471338 0.07981698 1.3435934 ## [83] {newspapers} 0.2420382 0.07981698 1.2508912 ## [84] {newspapers}</pre>	=> 190 => 269		

0.2259615 0.06344687 2.0730706 ## [86] {domestic eggs}	141	{yogurt}	0.01433655
0.2259615 0.06344687 1.6197753	141	(Jogur c)	0.01.33033
## [87] {domestic eggs} 0.2467949 0.06344687 1.3417510	=> 154	{rolls/buns}	0.01565836
<pre>## [88] {domestic eggs}</pre>	=>	{other vegetables}	0.02226741
0.3509615 0.06344687 1.8138238 ## [89] {domestic eggs}	219 =>	{whole milk}	0.02999492
0.4727564 0.06344687 1.8502027	295		
## [90] {fruit/vegetable juice} 0.2545710 0.07229283 1.4598869	=> 181	{soda}	0.01840366
## [91] {fruit/vegetable juice}	_	{yogurt}	0.01870869
0.2587904 0.07229283 1.8551049	184		
## [92] {fruit/vegetable juice}		{rolls/buns}	0.01453991
<pre>0.2011252 0.07229283 1.0934583 ## [93] {fruit/vegetable juice}</pre>	143	{other vegetables}	0.02104728
0.2911392 0.07229283 1.5046529	207	(other vegetables)	0.02104720
<pre>## [94] {fruit/vegetable juice}</pre>		<pre>{whole milk}</pre>	0.02663955
0.3684951 0.07229283 1.4421604	262		
## [95] {whipped/sour cream}		<pre>{root vegetables}</pre>	0.01708185
0.2382979 0.07168277 2.1862496	168	(,,,,,,,,,,,)	0 02074225
## [96] {whipped/sour cream} 0.2893617 0.07168277 2.0742510	=> 204	{yogurt}	0.02074225
## [97] {whipped/sour cream}		{rolls/buns}	0.01464159
0.2042553 0.07168277 1.1104760	144	(,)	
<pre>## [98] {whipped/sour cream}</pre>		<pre>{other vegetables}</pre>	0.02887646
0.4028369 0.07168277 2.0819237	284	(1	0 02222402
## [99] {whipped/sour cream}		<pre>{whole milk}</pre>	0.03223183
0.4496454 0.07168277 1.7597542 ## [100] {pip fruit}	317	{tropical fruit}	0.02043721
0.2701613 0.07564820 2.5746476	201	(cropical fruit)	0.02043721
## [101] {pip fruit}	_	<pre>{root vegetables}</pre>	0.01555669
0.2056452 0.07564820 1.8866793	153	,	
## [102] {pip fruit}		<pre>{yogurt}</pre>	0.01799695
0.2379032 0.07564820 1.7053777	177		0.00510115
## [103] {pip fruit}		<pre>{other vegetables}</pre>	0.02613116
0.3454301 0.07564820 1.7852365 ## [104] {pip fruit}	257	{whole milk}	0.03009659
0.3978495 0.07564820 1.5570432	296	(MIIOTE IIITIK)	0.000000
## [105] {pastry}		{soda}	0.02104728
0.2365714 0.08896797 1.3566647	207		
## [106] {pastry}		{rolls/buns}	0.02094560
0.2354286 0.08896797 1.2799558	206	(other vegetables)	0 02257245
## [107] {pastry} 0.2537143 0.08896797 1.3112349	=> 222	{other vegetables}	0.02257245
## [108] {pastry}		{whole milk}	0.03324860
0.3737143 0.08896797 1.4625865	327	(
## [109] {citrus fruit}		{tropical fruit}	0.01992883
0.2407862 0.08276563 2.2947022	196		
## [110] {citrus fruit}	=>	<pre>{root vegetables}</pre>	0.01769192

0.2137592 0.08276563 1.9611211 ## [111] {citrus fruit}	174 =>	{yogurt}	0.02165735
0.2616708 0.08276563 1.8757521	213		
## [112] {citrus fruit}		<pre>{rolls/buns}</pre>	0.01677682
0.2027027 0.08276563 1.1020349 ## [113] {citrus fruit}	165	{other vegetables}	0.02887646
0.3488943 0.08276563 1.8031403	284	(other vegetables)	0.02007040
## [114] {citrus fruit}	=>	<pre>{whole milk}</pre>	0.03050330
0.3685504 0.08276563 1.4423768	300		
## [115] {shopping bags} 0.2497420 0.09852567 1.4321939	=> 242	{soda}	0.02460600
## [116] {shopping bags}		{other vegetables}	0.02318251
0.2352941 0.09852567 1.2160366	228	(other vegetables)	0.02310231
## [117] {shopping bags}	=>	<pre>{whole milk}</pre>	0.02450432
0.2487100 0.09852567 0.9733637	241		
## [118] {sausage}		{soda}	0.02430097
0.2586580 0.09395018 1.4833245 ## [119] {sausage}	239	{yogurt}	0.01962379
0.2088745 0.09395018 1.4972889	193	(yogur c)	0.01302373
## [120] {sausage}		{rolls/buns}	0.03060498
0.3257576 0.09395018 1.7710480	301		
## [121] {sausage}		<pre>{other vegetables}</pre>	0.02694459
0.2867965 0.09395018 1.4822091	265	(uhala mɨlk)	a a2000224
## [122] {sausage} 0.3181818 0.09395018 1.2452520	=> 294	{whole milk}	0.02989324
## [123] {bottled water}		{soda}	0.02897814
0.2621895 0.11052364 1.5035766	285		
## [124] {bottled water}		<pre>{yogurt}</pre>	0.02297916
0.2079117 0.11052364 1.4903873	226	6 77 (1)	0.00440000
## [125] {bottled water} 0.2189512 0.11052364 1.1903734	=> 238	{rolls/buns}	0.02419929
## [126] {bottled water}		{other vegetables}	0.02480935
0.2244710 0.11052364 1.1601012	244	(other vegetables)	0.02-00333
## [127] {bottled water}	=>	<pre>{whole milk}</pre>	0.03436706
0.3109476 0.11052364 1.2169396	338		
## [128] {tropical fruit}		<pre>{root vegetables}</pre>	0.02104728
0.2005814 0.10493137 1.8402220 ## [129] {tropical fruit}	207	(vogunt)	0.02928317
0.2790698 0.10493137 2.0004746	288	{yogurt}	0.02920317
## [130] {yogurt}		{tropical fruit}	0.02928317
0.2099125 0.13950178 2.0004746	288	,	
## [131] {tropical fruit}	=>	<pre>{rolls/buns}</pre>	0.02460600
0.2344961 0.10493137 1.2748863	242		0 03500333
## [132] {tropical fruit} 0.3420543 0.10493137 1.7677896	=> 353	{other vegetables}	0.03589222
## [133] {tropical fruit}		{whole milk}	0.04229792
0.4031008 0.10493137 1.5775950	416	(0.0.223,32
<pre>## [134] {root vegetables}</pre>	=>	{yogurt}	0.02582613
0.2369403 0.10899847 1.6984751	254		
## [135] {root vegetables}	=>	{rolls/buns}	0.02430097

0.2229478 0.10899847 1.2121013 ## [136] {root vegetables}	239	<pre>{other vegetables}</pre>	0.04738180
0.4347015 0.10899847 2.2466049	466	(other vegetables)	0.04730100
<pre>## [137] {other vegetables}</pre>		<pre>{root vegetables}</pre>	0.04738180
0.2448765 0.19349263 2.2466049	466		
<pre>## [138] {root vegetables}</pre>	=>	<pre>{whole milk}</pre>	0.04890696
0.4486940 0.10899847 1.7560310	481		
## [139] {soda}		<pre>{rolls/buns}</pre>	0.03833249
0.2198251 0.17437722 1.1951242	377		
## [140] {rolls/buns}		{soda}	0.03833249
0.2084024 0.18393493 1.1951242	377	(ba]a m#]][)	0.04006101
## [141] {soda} 0.2297376 0.17437722 0.8991124	=> 394	{whole milk}	0.04006101
## [142] {yogurt}		{rolls/buns}	0.03436706
0.2463557 0.13950178 1.3393633	338	[1 0113/ buil3]	0.03430700
## [143] {yogurt}		{other vegetables}	0.04341637
0.3112245 0.13950178 1.6084566	427	(Jene: Vegetusies)	0.01312037
<pre>## [144] {other vegetables}</pre>	=>	{yogurt}	0.04341637
0.2243826 0.19349263 1.6084566	427	,	
## [145] {yogurt}	=>	<pre>{whole milk}</pre>	0.05602440
0.4016035 0.13950178 1.5717351	551		
## [146] {whole milk}	=>	<pre>{yogurt}</pre>	0.05602440
0.2192598 0.25551601 1.5717351	551		
## [147] {rolls/buns}		<pre>{other vegetables}</pre>	0.04260295
0.2316197 0.18393493 1.1970465	419	6 77 (1)	
## [148] {other vegetables}		<pre>{rolls/buns}</pre>	0.04260295
0.2201787 0.19349263 1.1970465	419	(ba]a m#]][)	0 05662447
## [149] {rolls/buns} 0.3079049 0.18393493 1.2050318	=> 557	{whole milk}	0.05663447
## [150] {whole milk}		{rolls/buns}	0.05663447
0.2216474 0.25551601 1.2050318	557	[1 0113/ buil3]	0.0000447
## [151] {other vegetables}		<pre>{whole milk}</pre>	0.07483477
0.3867578 0.19349263 1.5136341	736	(MIOLE MILLY)	0.07 103 177
## [152] {whole milk}		<pre>{other vegetables}</pre>	0.07483477
0.2928770 0.25551601 1.5136341	736	,	
## [153] {curd,			
## yogurt}	=>	<pre>{whole milk}</pre>	0.01006609
0.5823529 0.01728521 2.2791250	99		
## [154] {curd,			
## whole milk}		{yogurt}	0.01006609
0.3852140 0.02613116 2.7613555	99		
## [155] {other vegetables,		(0.01016777
## pork}		{whole milk}	0.01016777
0.4694836 0.02165735 1.8373939	100		
## [156] {pork, ## whole milk}	->	{other vegetables}	0.01016777
0.4587156 0.02216573 2.3707136	100	former Aegerantes?	0.01010///
## [157] {butter,	100		
## other vegetables}	=>	<pre>{whole milk}</pre>	0.01148958
0.5736041 0.02003050 2.2448850	113		

```
## [158] {butter,
          whole milk}
##
                                    => {other vegetables}
                                                            0.01148958
0.4169742 0.02755465 2.1549874
                                 113
## [159] {domestic eggs,
                                   => {whole milk}
         other vegetables}
                                                            0.01230300
0.5525114 0.02226741 2.1623358
                                 121
## [160] {domestic eggs,
                                    => {other vegetables}
         whole milk}
                                                            0.01230300
0.4101695 0.02999492 2.1198197
## [161] {fruit/vegetable juice,
         other vegetables}
                                    => {whole milk}
                                                            0.01047280
0.4975845 0.02104728 1.9473713
                                 103
## [162] {fruit/vegetable juice,
                                    => {other vegetables}
         whole milk}
                                                            0.01047280
0.3931298 0.02663955 2.0317558
                                 103
## [163] {whipped/sour cream,
         yogurt }
                                    => {other vegetables}
                                                            0.01016777
0.4901961 0.02074225 2.5334096
                                 100
## [164] {other vegetables,
         whipped/sour cream}
                                   => {yogurt}
                                                            0.01016777
0.3521127 0.02887646 2.5240730
                                 100
## [165] {other vegetables,
                                    => {whipped/sour cream} 0.01016777
         yogurt}
0.2341920 0.04341637 3.2670620
                                 100
## [166] {whipped/sour cream,
         yogurt }
                                   => {whole milk}
                                                            0.01087951
0.5245098 0.02074225 2.0527473
                                 107
## [167] {whipped/sour cream,
         whole milk}
                                                            0.01087951
                                    => {yogurt}
0.3375394 0.03223183 2.4196066
                                 107
## [168] {other vegetables,
                                   => {whole milk}
         whipped/sour cream}
                                                            0.01464159
0.5070423 0.02887646 1.9843854
                                 144
## [169] {whipped/sour cream,
                                    => {other vegetables}
         whole milk}
                                                            0.01464159
0.4542587 0.03223183 2.3476795
                                 144
## [170] {other vegetables,
##
                                    => {whole milk}
                                                            0.01352313
          pip fruit}
0.5175097 0.02613116 2.0253514
                                 133
## [171] {pip fruit,
         whole milk}
                                    => {other vegetables}
                                                            0.01352313
0.4493243 0.03009659 2.3221780
                                 133
## [172] {other vegetables,
##
         pastry}
                                    => {whole milk}
                                                            0.01057448
0.4684685 0.02257245 1.8334212
                                 104
## [173] {pastry,
         whole milk}
                                    => {other vegetables}
                                                            0.01057448
0.3180428 0.03324860 1.6436947
                                 104
## [174] {citrus fruit,
## root vegetables} => {other vegetables} 0.01037112
```

0.5862069 0.01769192 3.0296084 ## [175] {citrus fruit,	102		
## other vegetables} 0.3591549 0.02887646 3.2950455 ## [176] {other vegetables,	=> 102	<pre>{root vegetables}</pre>	0.01037112
## root vegetables} 0.2188841 0.04738180 2.6446257 ## [177] {citrus fruit,	=> 102	{citrus fruit}	0.01037112
## yogurt} 0.4741784 0.02165735 1.8557678 ## [178] {citrus fruit,	=> 101	{whole milk}	0.01026945
## whole milk} 0.3366667 0.03050330 2.4133503	=> 101	{yogurt}	0.01026945
## [179] {citrus fruit, ## other vegetables} 0.4507042 0.02887646 1.7638982	=> 128	{whole milk}	0.01301474
## [180] {citrus fruit, ## whole milk} 0.4266667 0.03050330 2.2050797	=> 128	{other vegetables}	0.01301474
## [181] {other vegetables, ## sausage} 0.3773585 0.02694459 1.4768487	=> 100	{whole milk}	0.01016777
<pre>## [182] {sausage, ## whole milk} 0.3401361 0.02989324 1.7578760</pre>	=> 100	{other vegetables}	0.01016777
<pre>## [183] {bottled water, ## other vegetables} 0.4344262 0.02480935 1.7001918</pre>	=> 106	{whole milk}	0.01077783
<pre>## [184] {bottled water, ## whole milk} 0.3136095 0.03436706 1.6207825</pre>	=> 106	{other vegetables}	0.01077783
<pre>## [185] {root vegetables, ## tropical fruit} 0.5845411 0.02104728 3.0209991</pre>		{other vegetables}	0.01230300
<pre>## [186] {other vegetables, ## tropical fruit}</pre>	=>	<pre>{root vegetables}</pre>	0.01230300
<pre>0.3427762 0.03589222 3.1447798 ## [187] {other vegetables, ## root vegetables}</pre>		{tropical fruit}	0.01230300
0.2596567 0.04738180 2.4745380 ## [188] {root vegetables, ## tropical fruit}	121	{whole milk}	0.01199797
0.5700483 0.02104728 2.2309690 ## [189] {tropical fruit, ## whole milk}	118	<pre>{root vegetables}</pre>	0.01199797
0.2836538 0.04229792 2.6023653 ## [190] {root vegetables, ## whole milk}	118	{tropical fruit}	0.01199797
0.2453222 0.04890696 2.3379305 ## [191] {tropical fruit,	118	(cropical indic)	0.01133/3/

<pre>## yogurt} 0.4201389 0.02928317 2.1713431 ## [192] {other vegetables,</pre>	=> 121	<pre>{other vegetables}</pre>	0.01230300
## tropical fruit} 0.3427762 0.03589222 2.4571457	=> 121	{yogurt}	0.01230300
## [193] {other vegetables, ## yogurt} 0.2833724 0.04341637 2.7005496	=> 121	{tropical fruit}	0.01230300
## [194] {tropical fruit, ## yogurt} 0.5173611 0.02928317 2.0247698	=> 149	{whole milk}	0.01514997
<pre>## [195] {tropical fruit, ## whole milk} 0.3581731 0.04229792 2.5675162</pre>	=> 149	{yogurt}	0.01514997
## [196] {whole milk, ## yogurt} 0.2704174 0.05602440 2.5770885	=> 149	{tropical fruit}	0.01514997
<pre>## [197] {rolls/buns, ## tropical fruit} 0.4462810 0.02460600 1.7465872</pre>	=> 108	{whole milk}	0.01098119
<pre>## [198] {tropical fruit, ## whole milk} 0.2596154 0.04229792 1.4114524</pre>	=> 108	{rolls/buns}	0.01098119
## [199] {other vegetables, ## tropical fruit} 0.4759207 0.03589222 1.8625865		{whole milk}	0.01708185
<pre>## [200] {tropical fruit, ## whole milk}</pre>	=>	{other vegetables}	0.01708185
<pre>0.4038462 0.04229792 2.0871397 ## [201] {other vegetables, ## whole milk}</pre>		{tropical fruit}	0.01708185
<pre>0.2282609 0.07483477 2.1753349 ## [202] {root vegetables, ## yogurt}</pre>	168	<pre>{other vegetables}</pre>	0.01291307
0.5000000 0.02582613 2.5840778 ## [203] {other vegetables, ## root vegetables}	127	{yogurt}	0.01291307
0.2725322 0.04738180 1.9536108 ## [204] {other vegetables,	127		
## yogurt} 0.2974239 0.04341637 2.7286977 ## [205] {root vegetables,	127	{root vegetables}	0.01291307
## yogurt} 0.5629921 0.02582613 2.2033536 ## [206] {root vegetables,	=> 143	{whole milk}	0.01453991
## whole milk} 0.2972973 0.04890696 2.1311362 ## [207] {whole milk,	=> 143	{yogurt}	0.01453991
## yogurt} 0.2595281 0.05602440 2.3810253	=> 143	<pre>{root vegetables}</pre>	0.01453991

<pre>## [208] {rolls/buns, ## root vegetables} 0.5020921 0.02430097 2.5948898 ## [209] {other vegetables,</pre>	=> {other vegetables}	0.01220132
## root vegetables} 0.2575107 0.04738180 1.4000100 ## [210] {other vegetables,	<pre>=> {rolls/buns} 120</pre>	0.01220132
## rolls/buns} 0.2863962 0.04260295 2.6275247 ## [211] {rolls/buns,	<pre>=> {root vegetables} 120</pre>	0.01220132
## root vegetables} 0.5230126 0.02430097 2.0468876 ## [212] {root vegetables,	<pre>=> {whole milk} 125</pre>	0.01270971
## whole milk} 0.2598753 0.04890696 1.4128652 ## [213] {rolls/buns,	<pre>=> {rolls/buns} 125</pre>	0.01270971
## whole milk} 0.2244165 0.05663447 2.0588959 ## [214] {other vegetables,	<pre>=> {root vegetables} 125</pre>	0.01270971
## root vegetables} 0.4892704 0.04738180 1.9148326	<pre>=> {whole milk} 228</pre>	0.02318251
## [215] {root vegetables, ## whole milk} 0.4740125 0.04890696 2.4497702	<pre>=> {other vegetables} 228</pre>	0.02318251
## [216] {other vegetables, ## whole milk} 0.3097826 0.07483477 2.8420820	<pre>=> {root vegetables} 228</pre>	0.02318251
## [217] {soda, ## yogurt} 0.3828996 0.02735130 1.4985348	<pre>=> {whole milk} 103</pre>	0.01047280
## [218] {soda, ## whole milk} 0.2614213 0.04006101 1.8739641	=> {yogurt} 103	0.01047280
## [219] {other vegetables, ## soda} 0.4254658 0.03274021 1.6651240	<pre>=> {whole milk} 137</pre>	0.01392984
## [220] {soda, ## whole milk} 0.3477157 0.04006101 1.7970490	<pre>=> {other vegetables} 137</pre>	0.01392984
## [221] {rolls/buns, ## yogurt} 0.3343195 0.03436706 1.7278153	<pre>=> {other vegetables} 113</pre>	0.01148958
## [222] {other vegetables, ## yogurt} 0.2646370 0.04341637 1.4387534	<pre>=> {rolls/buns} 113</pre>	0.01148958
<pre>## [223] {other vegetables, ## rolls/buns} 0.2696897 0.04260295 1.9332351</pre>	=> {yogurt} 113	0.01148958
## [224] {rolls/buns, ## yogurt}	=> {whole milk}	0.01555669

```
0.4526627 0.03436706 1.7715630
                                 153
## [225] {whole milk,
         yogurt}
                                   => {rolls/buns}
                                                            0.01555669
0.2776770 0.05602440 1.5096478
                                 153
## [226] {rolls/buns,
         whole milk}
                                    => {yogurt}
                                                            0.01555669
0.2746858 0.05663447 1.9690488
                                 153
## [227] {other vegetables,
                                                           0.02226741
         yogurt}
                                    => {whole milk}
0.5128806 0.04341637 2.0072345
                                 219
## [228] {whole milk,
##
         yogurt}
                                    => {other vegetables}
                                                           0.02226741
0.3974592 0.05602440 2.0541308
                                 219
## [229] {other vegetables,
         whole milk}
                                                            0.02226741
##
                                   => {yogurt}
0.2975543 0.07483477 2.1329789
## [230] {other vegetables,
          rolls/buns}
                                   => {whole milk}
                                                            0.01789527
0.4200477 0.04260295 1.6439194
                                 176
## [231] {rolls/buns,
                                    => {other vegetables}  0.01789527
         whole milk}
0.3159785 0.05663447 1.6330258
                                 176
## [232] {other vegetables,
         whole milk}
                                    => {rolls/buns}
                                                            0.01789527
0.2391304 0.07483477 1.3000817
                                 176
inspect(subset(basketrules, confidence > 0.2))
##
         1hs
                                       rhs
                                                                   support
confidence
                            lift count
             coverage
## [1]
                                    => {whole milk}
                                                              0.255516014
0.2555160 1.000000000 1.0000000 2513
## [2] {cake bar}
                                    => {whole milk}
                                                               0.005592272
0.4230769 0.013218099 1.6557746
                                   55
                                    => {other vegetables}
## [3]
       {dishes}
                                                               0.005998983
0.3410405 0.017590239 1.7625502
                                   59
## [4] {dishes}
                                   => {whole milk}
                                                               0.005287239
0.3005780 0.017590239 1.1763569
                                   52
## [5]
       {mustard}
                                   => {whole milk}
                                                               0.005185562
0.4322034 0.011997966 1.6914924
                                   51
## [6] {pot plants}
                                   => {whole milk}
                                                               0.006914082
0.4000000 0.017285206 1.5654596
                                   68
## [7]
       {chewing gum}
                                   => {soda}
                                                               0.005388917
0.2560386 0.021047280 1.4683033
                                   53
## [8] {chewing gum}
                                   => {whole milk}
                                                               0.005083884
0.2415459 0.021047280 0.9453259
                                   50
## [9] {canned fish}
                                   => {other vegetables}
                                                               0.005083884
0.3378378 0.015048297 1.7459985
                                    => {whole milk}
## [10] {pasta}
                                                               0.006100661
0.4054054 0.015048297 1.5866145
```

## [11] {herbs} 0.4312500 0.016268429 3.9564774	=> 69	<pre>{root vegetables}</pre>	0.007015760
## [12] {herbs} 0.4750000 0.016268429 2.4548739	=> 76	<pre>{other vegetables}</pre>	0.007727504
## [13] {herbs}	=>	{whole milk}	0.007727504
0.4750000 0.016268429 1.8589833 ## [14] {processed cheese}	76 =>	{soda}	0.005287239
0.3190184 0.016573462 1.8294729 ## [15] {processed cheese}	52 =>	<pre>{other vegetables}</pre>	0.005490595
0.3312883 0.016573462 1.7121497 ## [16] {processed cheese}	54	{whole milk}	0.007015760
0.4233129 0.016573462 1.6566981	69		
## [17] {semi-finished bread} 0.2931034 0.017691917 1.5148042	=> 51	<pre>{other vegetables}</pre>	0.005185562
## [18] {semi-finished bread} 0.4022989 0.017691917 1.5744565	=> 70	{whole milk}	0.007117438
## [19] {beverages} 0.2109375 0.026029487 1.5120775		{yogurt}	0.005490595
## [20] {beverages}	=>	{rolls/buns}	0.005388917
<pre>0.2070312 0.026029487 1.1255679 ## [21] {beverages}</pre>		{whole milk}	0.006812405
0.2617188 0.026029487 1.0242753 ## [22] {ice cream}	67 =>	{soda}	0.006100661
0.2439024 0.025012710 1.3987058 ## [23] {ice cream}	60	{other vegetables}	0.005083884
0.2032520 0.025012710 1.0504381	50		
## [24] {ice cream} 0.2357724 0.025012710 0.9227303	58	{whole milk}	0.005897306
## [25] {detergent} 0.3333333 0.019217082 1.7227185	=> 63	{other vegetables}	0.006405694
## [26] {detergent} 0.4656085 0.019217082 1.8222281	=> 88	<pre>{whole milk}</pre>	0.008947636
<pre>## [27] {pickled vegetables}</pre>	=>	{other vegetables}	0.006405694
0.3579545 0.017895272 1.8499648 ## [28] {pickled vegetables}	63 =>	{whole milk}	0.007117438
0.3977273 0.017895272 1.5565650 ## [29] {baking powder}	70 =>	<pre>{other vegetables}</pre>	0.007320793
0.4137931 0.017691917 2.1385471 ## [30] {baking powder}	72	{whole milk}	0.009252669
0.5229885 0.017691917 2.0467935	91		
## [31] {flour} 0.3625731 0.017386884 1.8738342	62	{other vegetables}	0.006304016
## [32] {flour} 0.4853801 0.017386884 1.8996074	=> 83	{whole milk}	0.008439248
## [33] {soft cheese} 0.3511905 0.017081851 2.5174623	=> 59	{yogurt}	0.005998983
## [34] {soft cheese}	=>	{rolls/buns}	0.005388917
0.3154762 0.017081851 1.7151511 ## [35] {soft cheese}		<pre>{other vegetables}</pre>	0.007117438
0.4166667 0.017081851 2.1533981	70		

## [36] {soft cheese} 0.4404762 0.017081851 1.7238692	=> 74	{whole milk}	0.007524148
## [37] {specialty bar}	=>	{soda}	0.007219115
0.2639405 0.027351296 1.5136181 ## [38] {specialty bar}	71	{rolls/buns}	0.005592272
0.2044610 0.027351296 1.1115940	55	(1 OII3/ Dull3)	0.003332272
## [39] {specialty bar}	=>	<pre>{other vegetables}</pre>	0.005592272
0.2044610 0.027351296 1.0566861 ## [40] {specialty bar}	55 ->	{whole milk}	0.006507372
0.2379182 0.027351296 0.9311284	64	(MIIOTE IIITIK)	0.000307372
## [41] {misc. beverages}	=>	{soda}	0.007320793
0.2580645 0.028368073 1.4799210	72	(1 11	0.007045760
## [42] {misc. beverages} 0.2473118 0.028368073 0.9678917	=> 69	{whole milk}	0.007015760
## [43] {grapes}		{tropical fruit}	0.006100661
0.2727273 0.022369090 2.5991015	60	(3. 24-23- 1. 4-3)	
## [44] {grapes}		<pre>{other vegetables}</pre>	0.009049314
0.4045455 0.022369090 2.0907538	89	(.deala målla)	0.007330703
## [45] {grapes} 0.3272727 0.022369090 1.2808306	=> 72	{whole milk}	0.007320793
## [46] {cat food}		{yogurt}	0.006202339
0.2663755 0.023284189 1.9094778	61		
## [47] {cat food}		<pre>{other vegetables}</pre>	0.006507372
0.2794760 0.023284189 1.4443753	64	(whole milk)	0.008845958
## [48] {cat food} 0.3799127 0.023284189 1.4868448	= > 87	{whole milk}	0.000045550
## [49] {specialty chocolate}		{soda}	0.006304016
0.2073579 0.030401627 1.1891338	62		
## [50] {specialty chocolate}		<pre>{other vegetables}</pre>	0.006100661
0.2006689 0.030401627 1.0370881 ## [51] {specialty chocolate}	60 ->	{whole milk}	0.008032537
0.2642140 0.030401627 1.0340410	79	(WHOLE WILK)	0.000032337
## [52] {meat}	=>	{sausage}	0.005287239
0.2047244 0.025826131 2.1790742	52		
## [53] {meat} 0.2125984 0.025826131 1.2191869		{soda}	0.005490595
## [54] {meat}	54 =>	{yogurt}	0.005287239
0.2047244 0.025826131 1.4675398	52	(yogu, c)	0.003207233
## [55] {meat}	=>	<pre>{rolls/buns}</pre>	0.006914082
0.2677165 0.025826131 1.4554959	68	6.11	0.000064443
## [56] {meat} 0.3858268 0.025826131 1.9940128	=> 98	{other vegetables}	0.009964413
## [57] {meat}		{whole milk}	0.009964413
0.3858268 0.025826131 1.5099906	98	,	
## [58] {frozen meals}		{soda}	0.006202339
0.2186380 0.028368073 1.2538220	61	(vogunt)	0 006202220
## [59] {frozen meals} 0.2186380 0.028368073 1.5672774	=> 61	{yogurt}	0.006202339
## [60] {frozen meals}	-	<pre>{other vegetables}</pre>	0.007524148
0.2652330 0.028368073 1.3707653	74	-	

## [61] {frozen meals} 0.3476703 0.028368073 1.3606593		<pre>{whole milk}</pre>	0.009862735
## [62] {hard cheese}	97 =>	{sausage}	0.005185562
0.2116183 0.024504321 2.2524519	51	(
## [63] {hard cheese}		<pre>{root vegetables}</pre>	0.005592272
0.2282158 0.024504321 2.0937519	55	(0.006405604
## [64] {hard cheese} 0.2614108 0.024504321 1.8738886	=> 63	{yogurt}	0.006405694
## [65] {hard cheese}		{rolls/buns}	0.005897306
0.2406639 0.024504321 1.3084187	58		
## [66] {hard cheese}		<pre>{other vegetables}</pre>	0.009456024
0.3858921 0.024504321 1.9943505	93	المادية والمادية	0.010066000
## [67] {hard cheese} 0.4107884 0.024504321 1.6076815	=> 99	{whole milk}	0.010066090
## [68] {butter milk}		{yogurt}	0.008540925
0.3054545 0.027961362 2.1896104	84	()-8	
## [69] {butter milk}		<pre>{rolls/buns}</pre>	0.007625826
0.2727273 0.027961362 1.4827378	75	()	0.040274404
## [70] {butter milk} 0.3709091 0.027961362 1.9169159	=> 102	{other vegetables}	0.010371124
## [71] {butter milk}		{whole milk}	0.011591256
0.4145455 0.027961362 1.6223854	114	(
## [72] {candy}		{soda}	0.008642603
0.2891156 0.029893238 1.6579897	85	(11 ())	0.007447420
## [73] {candy} 0.2380952 0.029893238 1.2944537	=> 70	{rolls/buns}	0.007117438
## [74] {candy}		{other vegetables}	0.006914082
0.2312925 0.029893238 1.1953557	68	(other vegetubies)	0.000311002
## [75] {candy}	=>	<pre>{whole milk}</pre>	0.008235892
0.2755102 0.029893238 1.0782502	81		
## [76] {ham}		{tropical fruit}	0.005388917
0.2070312 0.026029487 1.9730158 ## [77] {ham}	53 ->	{yogurt}	0.006710727
0.2578125 0.026029487 1.8480947	66	(yogur c)	0.000/10/2/
## [78] {ham}		{rolls/buns}	0.006914082
0.2656250 0.026029487 1.4441249	68	· ·	
## [79] {ham}		<pre>{other vegetables}</pre>	0.009150991
0.3515625 0.026029487 1.8169297	90	(ubala milk)	0 011400570
## [80] {ham} 0.4414062 0.026029487 1.7275091	=> 113	{whole milk}	0.011489578
## [81] {sliced cheese}		{sausage}	0.007015760
0.2863071 0.024504321 3.0474349	69	(
## [82] {sliced cheese}	=>	{tropical fruit}	0.005287239
0.2157676 0.024504321 2.0562739	52		
## [83] {sliced cheese} 0.2282158 0.024504321 2.0937519		<pre>{root vegetables}</pre>	0.005592272
## [84] {sliced cheese}	55 =>	{soda}	0.005083884
0.2074689 0.024504321 1.1897705	50	()	
## [85] {sliced cheese}	=>	<pre>{yogurt}</pre>	0.008032537
0.3278008 0.024504321 2.3497968	79		

## [86] {sliced cheese}		<pre>{rolls/buns}</pre>	0.007625826
0.3112033 0.024504321 1.6919208 ## [87] {sliced cheese}	75 =>	{other vegetables}	0.009049314
0.3692946 0.024504321 1.9085720	89	(000. 10800	
## [88] {sliced cheese}		{whole milk}	0.010777834
0.4398340 0.024504321 1.7213560	106		
## [89] {UHT-milk}		{bottled water}	0.007320793
0.2188450 0.033451957 1.9800740 ## [90] {UHT-milk}	72 ->	{soda}	0.007625826
0.2279635 0.033451957 1.3073010	75	(Soua)	0.007023020
## [91] {UHT-milk}		{yogurt}	0.007422471
0.2218845 0.033451957 1.5905496	73		
## [92] {UHT-milk}		<pre>{other vegetables}</pre>	0.008134215
0.2431611 0.033451957 1.2566944	80		0.007045740
## [93] {oil} 0.2500000 0.028063040 2.2936101		<pre>{root vegetables}</pre>	0.007015760
## [94] {oil}	69 =>	{other vegetables}	0.009964413
0.3550725 0.028063040 1.8350697	98	(other vegetables)	0.00000
## [95] {oil}		{whole milk}	0.011286223
0.4021739 0.028063040 1.5739675	111		
## [96] {onions}		<pre>{root vegetables}</pre>	0.009456024
0.3049180 0.031011693 2.7974523	93		0.007040445
## [97] {onions} 0.2327869 0.031011693 1.6687019	=> 71	{yogurt}	0.007219115
## [98] {onions}		{rolls/buns}	0.006812405
0.2196721 0.031011693 1.1942927	67	(10113/04113)	0.000012403
## [99] {onions}		<pre>{other vegetables}</pre>	0.014234875
0.4590164 0.031011693 2.3722681	140		
## [100] {onions}		{whole milk}	0.012099644
0.3901639 0.031011693 1.5269647	119	6.1.	0.000040344
## [101] {berries} 0.2721713 0.033248602 3.7968855	=> 89	{whipped/sour cream}	0.009049314
## [102] {berries}		{tropical fruit}	0.006710727
0.2018349 0.033248602 1.9234941	66	(cropical fruit)	0.000/10/2/
## [103] {berries}		{soda}	0.007320793
0.2201835 0.033248602 1.2626849	72		
## [104] {berries}		<pre>{yogurt}</pre>	0.010574479
0.3180428 0.033248602 2.2798477	104	6.11	0.040050445
## [105] {berries} 0.3088685 0.033248602 1.5962805	=> 101	<pre>{other vegetables}</pre>	0.010269446
## [106] {berries}		{whole milk}	0.011794611
0.3547401 0.033248602 1.3883281	116	(WHOLE WILK)	0.011/54011
<pre>## [107] {hamburger meat}</pre>	=>	{rolls/buns}	0.008642603
0.2599388 0.033248602 1.4132109	85		
## [108] {hamburger meat}	=>	<pre>{other vegetables}</pre>	0.013828165
0.4159021 0.033248602 2.1494470	136	()	0.044742064
## [109] {hamburger meat}		{whole milk}	0.014743264
0.4434251 0.033248602 1.7354101 ## [110] {hygiene articles}	145 =>	{tropical fruit}	0.006710727
0.2037037 0.032943569 1.9413042	66	(cropicul riule)	0.000/10/2/

## [111] {hygiene articles}		{soda}	0.007015760
0.2129630 0.032943569 1.2212774 ## [112] {hygiene articles}		{yogurt}	0.007320793
0.2222222 0.032943569 1.5929705	72		
## [113] {hygiene articles}		{other vegetables}	0.009557702
0.2901235 0.032943569 1.4994032 ## [114] {hygiene articles}		{whole milk}	0.012811388
0.3888889 0.032943569 1.5219746		,	
## [115] {salty snack}		{soda}	0.009354347
0.2473118 0.037824098 1.4182576 ## [116] {salty snack}		<pre>{other vegetables}</pre>	0.010777834
0.2849462 0.037824098 1.4726465		(other regetuotes)	0.020,,,03
## [117] {salty snack}		<pre>{whole milk}</pre>	0.011184545
0.2956989 0.037824098 1.1572618			
## [118] {sugar}		{soda}	0.007320793
0.2162162		(vogunt)	0.006014002
## [119] {sugar} 0.2042042 0.033858668 1.4638107		{yogurt}	0.006914082
## [120] {sugar}		{rolls/buns}	0.007015760
0.2072072 0.033858668 1.1265245		(1 0113/ bull3)	0.007013700
## [121] {sugar}		{other vegetables}	0.010777834
0.3183183 0.033858668 1.6451186		(**************************************	
## [122] {sugar}		<pre>{whole milk}</pre>	0.015048297
0.4444444 0.033858668 1.7393996		(4-)	0.000557703
## [123] {waffles}		{soda}	0.009557702
0.2486772 0.038434164 1.4260879 ## [124] {waffles}		{rolls/buns}	0.009150991
0.2380952 0.038434164 1.2944537		(1 0113/ buil3)	0.000100001
## [125] {waffles}		{other vegetables}	0.010066090
0.2619048 0.038434164 1.3535645		(come regeons	0.02000000
## [126] {waffles}		<pre>{whole milk}</pre>	0.012709710
0.3306878 0.038434164 1.2941961	L 125		
## [127] {long life bakery prod	_	{soda}	0.007625826
0.2038043 0.037417387 1.1687555			
## [128] {long life bakery prod	-	{yogurt}	0.008744281
0.2336957 0.037417387 1.6752163 ## [129] {long life bakery prod		Snolle/hunel	0.007930859
0.2119565 0.037417387 1.1523452		(10113/Dulls)	0.00793039
## [130] {long life bakery prod		{other vegetables}	0.010676157
0.2853261 0.037417387 1.4746096	-		
## [131] {long life bakery prod	•	<pre>{whole milk}</pre>	0.013523132
0.3614130 0.037417387 1.4144438			
## [132] {dessert}		{soda}	0.009862735
0.2657534 0.037112354 1.5240145		(,,,,,,,,,,,)	0 000063735
## [133] {dessert} 0.2657534 0.037112354 1.9050182		{yogurt}	0.009862735
## [134] {dessert}		<pre>{other vegetables}</pre>	0.011591256
0.3123288 0.037112354 1.6141636		(Jener Vegetables)	0.011991290
## [135] {dessert}		{whole milk}	0.013726487
0.3698630 0.037112354 1.4475146		,	

## [136] {cream cheese}		<pre>{yogurt}</pre>	0.012404677
0.3128205 0.039654296 2.2424123 ## [137] {cream cheese}	122	{rolls/buns}	0.009964413
0.2512821 0.039654296 1.3661465	98	(10113/04113)	0.005504415
## [138] {cream cheese}	=>	<pre>{other vegetables}</pre>	0.013726487
0.3461538 0.039654296 1.7889769	135		
## [139] {cream cheese}		{whole milk}	0.016471784
0.4153846 0.039654296 1.6256696	162	(neet vegetables)	0 010070510
## [140] {chicken} 0.2535545 0.042907982 2.3262206	=> 107	<pre>{root vegetables}</pre>	0.010879512
## [141] {chicken}		{rolls/buns}	0.009659380
0.2251185 0.042907982 1.2239029	95	(, ,	
## [142] {chicken}	=>	<pre>{other vegetables}</pre>	0.017895272
0.4170616 0.042907982 2.1554393	176		
## [143] {chicken}		<pre>{whole milk}</pre>	0.017590239
0.4099526 0.042907982 1.6044106 ## [144] {white bread}	173	{tropical fruit}	0.008744281
0.2077295 0.042094560 1.9796699	86	(cropical fruit)	0.000/44201
## [145] {white bread}		{soda}	0.010269446
0.2439614 0.042094560 1.3990437	101		
## [146] {white bread}	=>	{yogurt}	0.009049314
0.2149758 0.042094560 1.5410258	89		
## [147] {white bread}		<pre>{other vegetables}</pre>	0.013726487
0.3260870 0.042094560 1.6852681 ## [148] {white bread}	135	{whole milk}	0.017081851
0.4057971 0.042094560 1.5881474	168	(MIIOIE IIIIK)	0.01/001031
## [149] {chocolate}		{soda}	0.013523132
0.2725410 0.049618709 1.5629391	133		
## [150] {chocolate}	=>	<pre>{rolls/buns}</pre>	0.011794611
0.2377049 0.049618709 1.2923316	116		
## [151] {chocolate}		<pre>{other vegetables}</pre>	0.012709710
<pre>0.2561475 0.049618709 1.3238103 ## [152] {chocolate}</pre>	125	{whole milk}	0.016675140
0.3360656 0.049618709 1.3152427	164	(MIIOIE IIIIK)	0.0100/3140
## [153] {coffee}		<pre>{other vegetables}</pre>	0.013421454
0.2311734 0.058057956 1.1947400	132	,	
## [154] {coffee}	=>	{whole milk}	0.018708693
0.3222417 0.058057956 1.2611408	184		
## [155] {frozen vegetables}		<pre>{root vegetables}</pre>	0.011591256
0.2410148 0.048093543 2.2111759 ## [156] {frozen vegetables}	114	{yogurt}	0.012404677
0.2579281 0.048093543 1.8489235	122	(yogurc)	0.012404077
## [157] {frozen vegetables}		{rolls/buns}	0.010167768
0.2114165 0.048093543 1.1494092	100		
<pre>## [158] {frozen vegetables}</pre>	=>	<pre>{other vegetables}</pre>	0.017793594
0.3699789 0.048093543 1.9121083	175		
## [159] {frozen vegetables}		{whole milk}	0.020437214
0.4249471 0.048093543 1.6630940 ## [160] {beef}	201	<pre>{root vegetables}</pre>	0.017386884
0.3313953 0.052465684 3.0403668	=> 171	froot Aegeranies?	0.01/300004
0.00001 0.0000			

## [161] {beef} 0.2228682 0.052465684	1 5076013	=> 115	{yogurt}	0.011692933
## [162] {beef}	1.59/6012		{rolls/buns}	0.013624809
0.2596899 0.052465684	1.4118576	134		
## [163] {beef}	4 0430550		<pre>{other vegetables}</pre>	0.019725470
0.3759690 0.052465684 ## [164] {beef}	1.9430662	194	{whole milk}	0.021250635
0.4050388 0.052465684	1.5851795	209	/MILOTE HITTK	0.021230033
## [165] {curd}		=>	<pre>{root vegetables}</pre>	0.010879512
0.2041985 0.053279105	1.8734067	107		
## [166] {curd} 0.3244275 0.053279105	2 2256154	=> 170	{yogurt}	0.017285206
## [167] {curd}	2.3230134		{other vegetables}	0.017183528
0.3225191 0.053279105	1.6668288	169	(oene. regetables)	0.01,103320
## [168] {curd}			<pre>{whole milk}</pre>	0.026131164
0.4904580 0.053279105	1.9194805	257	6 13	0.044007066
## [169] {napkins} 0.2291262 0.052364006	1 2120697	=> 118	{soda}	0.011997966
## [170] {napkins}	1.3139007		{yogurt}	0.012302999
0.2349515 0.052364006	1.6842183	121	() - 6)	0102202
## [171] {napkins}		=>	<pre>{rolls/buns}</pre>	0.011692933
0.2233010 0.052364006	1.2140216	115		0.044400004
## [172] {napkins} 0.2757282 0.052364006	1 /250060	=> 142	{other vegetables}	0.014438231
## [173] {napkins}	1.4230000		{whole milk}	0.019725470
0.3766990 0.052364006	1.4742678	194	(<u>)</u>	
## [174] {pork}			<pre>{root vegetables}</pre>	0.013624809
0.2363316 0.057651246	2.1682099	134	(4-)	0.011006300
## [175] {pork} 0.2063492 0.057651246	1 1933/05	=> 117	{soda}	0.011896289
## [176] {pork}	1.1055455		{other vegetables}	0.021657346
0.3756614 0.057651246	1.9414764	213	,	
## [177] {pork}			<pre>{whole milk}</pre>	0.022165735
0.3844797 0.057651246		218	(1 1 - //s)	0 010317003
## [178] {frankfurter] 0.3258621 0.058973055		=> 189	{rolls/buns}	0.019217082
## [179] {frankfurter}		_	{other vegetables}	0.016471784
0.2793103 0.058973055	1.4435193	162	,	
## [180] {frankfurter}			{whole milk}	0.020538892
0.3482759 0.058973055		202	(anda)	0.016000173
## [181] {bottled been 0.2108586 0.080528724	•	=> 167	{soda}	0.016980173
## [182] {bottled been		-	<pre>{other vegetables}</pre>	0.016166751
0.2007576 0.080528724	=	159		
## [183] {bottled been	=		{whole milk}	0.020437214
0.2537879 0.080528724 ## [184] {brown bread]		201	{yogurt}	0.014539908
0.2241379 0.064870361		=> 143	(Angri cl	0.014333300
## [185] {brown bread}			<pre>{other vegetables}</pre>	0.018708693
0.2884013 0.064870361	1.4905025	184		

## [186] {brown bread}	=> 248	<pre>{whole milk}</pre>	0.025216065
0.3887147 0.064870361 1.5212930 ## [187] {margarine}		{yogurt}	0.014234875
0.2430556 0.058566345 1.7423115	140	(yogur e)	0.01-25-075
## [188] {margarine}	=>	{rolls/buns}	0.014743264
0.2517361 0.058566345 1.3686151	145		
## [189] {margarine}		<pre>{other vegetables}</pre>	0.019725470
0.3368056 0.058566345 1.7406635	194		
## [190] {margarine}		{whole milk}	0.024199288
0.4131944 0.058566345 1.6170980 ## [191] {butter}	238	<pre>{root vegetables}</pre>	0.012913066
0.2330275 0.055414337 2.1378971	127	(100c vegetables)	0.012313000
## [192] {butter}		{yogurt}	0.014641586
0.2642202 0.055414337 1.8940273	144	() - 8 7	
## [193] {butter}	=>	<pre>{rolls/buns}</pre>	0.013421454
0.2422018 0.055414337 1.3167800	132		
## [194] {butter}		<pre>{other vegetables}</pre>	0.020030503
0.3614679 0.055414337 1.8681223	197	(.d1	0.027554652
## [195] {butter} 0.4972477 0.055414337 1.9460530	=> 271	{whole milk}	0.027554652
## [196] {newspapers}		{rolls/buns}	0.019725470
0.2471338 0.079816980 1.3435934	194	(1 0113/ buil3)	0.013/234/0
## [197] {newspapers}		{other vegetables}	0.019318760
0.2420382 0.079816980 1.2508912	190		
## [198] {newspapers}	=>	{whole milk}	0.027351296
0.3426752 0.079816980 1.3411103	269		
## [199] {domestic eggs}		<pre>{root vegetables}</pre>	0.014336553
0.2259615 0.063446873 2.0730706	141	(0.014226552
## [200] {domestic eggs} 0.2259615 0.063446873 1.6197753	=> 141	{yogurt}	0.014336553
## [201] {domestic eggs}		{rolls/buns}	0.015658363
0.2467949 0.063446873 1.3417510	154	(. 0113, 003)	0.013030303
## [202] {domestic eggs}	=>	<pre>{other vegetables}</pre>	0.022267412
0.3509615 0.063446873 1.8138238	219		
## [203] {domestic eggs}		<pre>{whole milk}</pre>	0.029994916
0.4727564 0.063446873 1.8502027	295		
## [204] {fruit/vegetable juice}		{soda}	0.018403660
0.2545710 0.072292832 1.4598869 ## [205] {fruit/vegetable juice}	181	{yogurt}	0.018708693
0.2587904 0.072292832 1.8551049	184	(yogurc)	0.010/00093
## [206] {fruit/vegetable juice}		{rolls/buns}	0.014539908
0.2011252 0.072292832 1.0934583	143	(
<pre>## [207] {fruit/vegetable juice}</pre>	=>	<pre>{other vegetables}</pre>	0.021047280
0.2911392 0.072292832 1.5046529	207		
<pre>## [208] {fruit/vegetable juice}</pre>		{whole milk}	0.026639553
0.3684951 0.072292832 1.4421604	262	(masttable.	0.047004054
## [209] {whipped/sour cream} 0.2382979 0.071682766 2.1862496	=> 168	<pre>{root vegetables}</pre>	0.017081851
## [210] {whipped/sour cream}		{yogurt}	0.020742247
0.2893617 0.071682766 2.0742510	204	(7.580, 5)	J. J. Z. J. T. Z. T.

## [211] {whipped/sour cream} 0.2042553 0.071682766 1.1104760		{rolls/buns}	0.014641586
## [212] {whipped/sour cream}	144 =>	{other vegetables}	0.028876462
0.4028369 0.071682766 2.0819237	284		
<pre>## [213] {whipped/sour cream}</pre>	=>	<pre>{whole milk}</pre>	0.032231825
0.4496454 0.071682766 1.7597542	317		
## [214] {pip fruit}		{tropical fruit}	0.020437214
0.2701613 0.075648195 2.5746476	201		
## [215] {pip fruit}		<pre>{root vegetables}</pre>	0.015556685
0.2056452 0.075648195 1.8866793	153	(0.017006050
## [216] {pip fruit} 0.2379032 0.075648195 1.7053777		{yogurt}	0.017996950
## [217] {pip fruit}	177	{other vegetables}	0.026131164
0.3454301 0.075648195 1.7852365	257	(Other Vegetables)	0.020131104
## [218] {pip fruit}		{whole milk}	0.030096594
0.3978495 0.075648195 1.5570432	296	(miere mrrk)	
## [219] {pastry}		{soda}	0.021047280
0.2365714 0.088967972 1.3566647	207		
## [220] {pastry}	=>	<pre>{rolls/buns}</pre>	0.020945602
0.2354286 0.088967972 1.2799558	206		
## [221] {pastry}	=>	<pre>{other vegetables}</pre>	0.022572445
0.2537143 0.088967972 1.3112349	222		
## [222] {pastry}		{whole milk}	0.033248602
0.3737143 0.088967972 1.4625865	327		
## [223] {citrus fruit}		{tropical fruit}	0.019928826
0.2407862 0.082765633 2.2947022	196	(mant vocatables)	0.017601017
## [224] {citrus fruit} 0.2137592 0.082765633 1.9611211	= <i>></i> 174	<pre>{root vegetables}</pre>	0.017691917
## [225] {citrus fruit}		{yogurt}	0.021657346
0.2616708 0.082765633 1.8757521	213	(yogur e)	0.021037340
## [226] {citrus fruit}		{rolls/buns}	0.016776817
0.2027027 0.082765633 1.1020349	165	(,,	
## [227] {citrus fruit}	=>	<pre>{other vegetables}</pre>	0.028876462
0.3488943 0.082765633 1.8031403	284		
## [228] {citrus fruit}	=>	<pre>{whole milk}</pre>	0.030503305
0.3685504 0.082765633 1.4423768	300		
## [229] {shopping bags}		{soda}	0.024605999
0.2497420 0.098525674 1.4321939	242		
## [230] {shopping bags}		<pre>{other vegetables}</pre>	0.023182511
0.2352941 0.098525674 1.2160366	228	(h.a.] a	0 024504224
## [231] {shopping bags} 0.2487100 0.098525674 0.9733637	=> 241	{whole milk}	0.024504321
## [232] {sausage}		{soda}	0.024300966
0.2586580 0.093950178 1.4833245	239	(Soua)	0.024300300
## [233] {sausage}		{yogurt}	0.019623793
0.2088745 0.093950178 1.4972889	193	() 080. 2)	0,0100101010
## [234] {sausage}		{rolls/buns}	0.030604982
0.3257576 0.093950178 1.7710480	301		
## [235] {sausage}	=>	<pre>{other vegetables}</pre>	0.026944586
0.2867965 0.093950178 1.4822091	265		

## [236] {sausage} 0.3181818 0.093950178 1.2452520	=> 294	<pre>{whole milk}</pre>	0.029893238
## [237] {bottled water}		{soda}	0.028978139
0.2621895 0.110523640 1.5035766	285		
## [238] {bottled water}		{yogurt}	0.022979156
0.2079117 0.110523640 1.4903873	226	() -8)	
## [239] {bottled water}		{rolls/buns}	0.024199288
0.2189512 0.110523640 1.1903734	238	(. 0110, 000)	0102122200
## [240] {bottled water}		{other vegetables}	0.024809354
0.2244710 0.110523640 1.1601012	244	(other vegetuoits)	0.021003331
## [241] {bottled water}		{whole milk}	0.034367056
0.3109476 0.110523640 1.2169396	338	(WHOLE IIILIN)	0.03 1307 030
## [242] {tropical fruit}		<pre>{root vegetables}</pre>	0.021047280
0.2005814 0.104931368 1.8402220	207	(100c vegetables)	0.02104/200
## [243] {tropical fruit}		{yogurt}	0.029283172
0.2790698 0.104931368 2.0004746	288	(yogui c)	0.029203172
## [244] {yogurt}		{tropical fruit}	0.029283172
0.2099125 0.139501779 2.0004746	288	(cropical fruit)	0.029203172
## [245] {tropical fruit}		{rolls/buns}	0.024605999
0.2344961 0.104931368 1.2748863	242	{1.0112/pail2}	0.024005999
		(other vegetables)	A A25002222
## [246] {tropical fruit} 0.3420543 0.104931368 1.7677896		<pre>{other vegetables}</pre>	0.035892222
	353	(bala milk)	0.042207016
## [247] {tropical fruit}		{whole milk}	0.042297916
0.4031008 0.104931368 1.5775950	416	(0 025026121
## [248] {root vegetables}		{yogurt}	0.025826131
0.2369403 0.108998475 1.6984751	254	(malla /ha)	0.024200066
## [249] {root vegetables}		<pre>{rolls/buns}</pre>	0.024300966
0.2229478 0.108998475 1.2121013	239	6 11	0.047204000
## [250] {root vegetables}		<pre>{other vegetables}</pre>	0.047381800
0.4347015 0.108998475 2.2466049	466	6 1 1 1 2 3	0.047204000
## [251] {other vegetables}		<pre>{root vegetables}</pre>	0.047381800
0.2448765 0.193492628 2.2466049	466	6 1 7 1713	0.040006065
<pre>## [252] {root vegetables}</pre>		{whole milk}	0.048906965
0.4486940 0.108998475 1.7560310	481	6 77 (1)	
## [253] {soda}		<pre>{rolls/buns}</pre>	0.038332486
0.2198251 0.174377224 1.1951242	377		
## [254] {rolls/buns}		{soda}	0.038332486
0.2084024 0.183934926 1.1951242	377	6 1 7 1713	0.040064007
## [255] {soda}		{whole milk}	0.040061007
0.2297376 0.174377224 0.8991124	394	6 77 ()	
## [256] {yogurt}		{rolls/buns}	0.034367056
0.2463557 0.139501779 1.3393633	338		
## [257] {yogurt}		<pre>{other vegetables}</pre>	0.043416370
0.3112245 0.139501779 1.6084566	427		
## [258] {other vegetables}		{yogurt}	0.043416370
0.2243826 0.193492628 1.6084566	427	6 1 7 1713	0.056001111
## [259] {yogurt}		{whole milk}	0.056024403
0.4016035 0.139501779 1.5717351	551		
## [260] {whole milk}		{yogurt}	0.056024403
0.2192598 0.255516014 1.5717351	551		

## [261] {rolls/buns} 0.2316197 0.183934926 1.1970465	=> 419	<pre>{other vegetables}</pre>	0.042602949
<pre>## [262] {other vegetables}</pre>	=>	{rolls/buns}	0.042602949
0.2201787 0.193492628 1.1970465 ## [263] {rolls/buns}	419 =>	{whole milk}	0.056634469
0.3079049 0.183934926 1.2050318	557	(MIIOTE IIITK)	0.00004409
## [264] {whole milk}		<pre>{rolls/buns}</pre>	0.056634469
0.2216474 0.255516014 1.2050318 ## [265] {other vegetables}	557 ->	{whole milk}	0.074834774
0.3867578 0.193492628 1.5136341	736	(WHOIC MIIK)	0.074054774
## [266] {whole milk}		<pre>{other vegetables}</pre>	0.074834774
0.2928770 0.255516014 1.5136341 ## [267] {oil,	736		
## other vegetables}	=>	{whole milk}	0.005083884
0.5102041 0.009964413 1.9967597	50		
## [268] {oil, ## whole milk}	=>	{other vegetables}	0.005083884
0.4504505 0.011286223 2.3279980	50	(other vegetables)	0.003003004
## [269] {onions,			
## root vegetables} 0.6021505 0.009456024 3.1120076	=> 56	{other vegetables}	0.005693950
## [270] {onions,	50		
<pre>## other vegetables}</pre>		<pre>{root vegetables}</pre>	0.005693950
0.4000000 0.014234875 3.6697761 ## [271] {onions,	56		
## other vegetables}	=>	{whole milk}	0.006609049
0.4642857 0.014234875 1.8170513	65	•	
## [272] {onions, ## whole milk}	->	{other vegetables}	0.006609049
0.5462185 0.012099644 2.8229421	65	focues regerantes?	0.000009049
## [273] {hamburger meat,			
## other vegetables} 0.4558824 0.013828165 1.7841635	=> 62	{whole milk}	0.006304016
## [274] {hamburger meat,	02		
## whole milk}	=>	<pre>{other vegetables}</pre>	0.006304016
0.4275862 0.014743264 2.2098320	62		
<pre>## [275] {hygiene articles, ## other vegetables}</pre>	=>	{whole milk}	0.005185562
0.5425532 0.009557702 2.1233628	51		
<pre>## [276] {hygiene articles, ## whole milk}</pre>	->	<pre>{other vegetables}</pre>	0.005185562
0.4047619 0.012811388 2.0918725	51	{orner, Aegeraptes}	0.005165562
## [277] {other vegetables,			
## sugar} 0.5849057 0.010777834 2.2891155		{whole milk}	0.006304016
## [278] {sugar,	62		
## whole milk}		<pre>{other vegetables}</pre>	0.006304016
0.4189189 0.015048297 2.1650381	62		
<pre>## [279] {long life bakery product ## other vegetables}</pre>		{whole milk}	0.005693950
5		•	

0.5333333 0.010676157 2.0872795	56		
## [280] {long life bakery product			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.005693950
0.4210526 0.013523132 2.1760655	56		
## [281] {cream cheese,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.005287239
0.4262295 0.012404677 2.2028204	52		
## [282] {cream cheese,			
## other vegetables}	=>	{yogurt}	0.005287239
0.3851852 0.013726487 2.7611489	52	() -8)	01005207257
## [283] {cream cheese,			
## yogurt}	->	{whole milk}	0.006609049
0.5327869 0.012404677 2.0851409	65	(WHOLE WILK)	0.0000000
## [284] {cream cheese,	05		
		(vogunt)	0.006609049
## whole milk}		{yogurt}	0.000009049
0.4012346 0.016471784 2.8761968	65		
## [285] {cream cheese,		6 1 2 1213	
<pre>## other vegetables}</pre>		{whole milk}	0.006710727
0.4888889 0.013726487 1.9133395	66		
## [286] {cream cheese,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.006710727
0.4074074 0.016471784 2.1055449	66		
## [287] {chicken,			
<pre>## root vegetables}</pre>	=>	<pre>{other vegetables}</pre>	0.005693950
0.5233645 0.010879512 2.7048291	56	-	
## [288] {chicken,			
<pre>## other vegetables}</pre>	=>	<pre>{root vegetables}</pre>	0.005693950
0.3181818 0.017895272 2.9191401	56		
## [289] {chicken,			
## root vegetables}	=>	{whole milk}	0.005998983
0.5514019 0.010879512 2.1579934	59	(WHOLE WILK)	0.003330303
## [290] {chicken,	,,,		
## whole milk}	->	<pre>{root vegetables}</pre>	0 005000003
0.3410405 0.017590239 3.1288554	59	(100c vegetables)	0.0055505
	29		
## [291] {chicken,		(b.a.] a	0 005307330
## rolls/buns}		{whole milk}	0.005287239
0.5473684 0.009659380 2.1422079	52		
## [292] {chicken,			
## whole milk}		{rolls/buns}	0.005287239
0.3005780 0.017590239 1.6341542	52		
## [293] {chicken,			
<pre>## other vegetables}</pre>	=>	{whole milk}	0.008439248
0.4715909 0.017895272 1.8456413	83		
## [294] {chicken,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.008439248
0.4797688 0.017590239 2.4795197	83	,	
## [295] {other vegetables,	_		
## white bread}	=>	{whole milk}	0.005897306
0.4296296 0.013726487 1.6814196	58	(miore mark)	0.005057500
## [296] {white bread,	50		
יים ובסטן לאוודרב מו בממי			

<pre>## whole milk} 0.3452381 0.017081851 1.7842442 ## [297] {chocolate,</pre>	<pre>=> {other vegetables} 58</pre>	0.005897306
## soda} 0.3759398 0.013523132 1.4712966	=> {whole milk} 50	0.005083884
<pre>## [298] {chocolate, ## whole milk}</pre>	=> {soda}	0.005083884
0.3048780 0.016675140 1.7483823 ## [299] {chocolate,	50	
## other vegetables} 0.4320000 0.012709710 1.6906964	=> {whole milk} 54	0.005490595
## [300] {chocolate, ## whole milk}	=> {other vegetables}	0.005490595
0.3292683 0.016675140 1.7017098 ## [301] {coffee,	54	0.005083884
## yogurt} 0.5208333 0.009761057 2.0383589 ## [302] {coffee,	=> {whole milk} 50	0.003003004
## whole milk} 0.2717391 0.018708693 1.9479259	=> {yogurt} 50	0.005083884
<pre>## [303] {coffee, ## other vegetables}</pre>		0.006405694
0.4772727 0.013421454 1.8678779 ## [304] {coffee,	63	
## whole milk} 0.3423913 0.018708693 1.7695315	<pre>=> {other vegetables} 63</pre>	0.006405694
<pre>## [305] {frozen vegetables, ## root vegetables}</pre>	<pre>=> {other vegetables}</pre>	0.006100661
0.5263158 0.011591256 2.7200819 ## [306] {frozen vegetables,	60	
## other vegetables} 0.3428571 0.017793594 3.1455224	<pre>=> {root vegetables} 60</pre>	0.006100661
<pre>## [307] {frozen vegetables, ## root vegetables}</pre>	=> {whole milk}	0.006202339
0.5350877 0.011591256 2.0941455 ## [308] {frozen vegetables,	61	0.006202220
## whole milk} 0.3034826 0.020437214 2.7842829 ## [309] {frozen vegetables,	<pre>=> {root vegetables} 61</pre>	0.006202339
## yogurt} 0.4262295 0.012404677 2.2028204	<pre>=> {other vegetables} 52</pre>	0.005287239
<pre>## [310] {frozen vegetables, ## other vegetables}</pre>	=> {yogurt}	0.005287239
0.2971429 0.017793594 2.1300292 ## [311] {frozen vegetables,	52	
## yogurt} 0.4918033 0.012404677 1.9247454	<pre>=> {whole milk} 60</pre>	0.006100661
<pre>## [312] {frozen vegetables, ## whole milk}</pre>	=> {yogurt}	0.006100661
0.2985075 0.020437214 2.1398111	60	

<pre>## [313] {frozen vegetables, ## rolls/buns}</pre>	=> {whole milk}	0.005083884
0.5000000 0.010167768 1.9568245 ## [314] {frozen vegetables,	50	
## whole milk} 0.2487562 0.020437214 1.3524143	<pre>=> {rolls/buns} 50</pre>	0.005083884
<pre>## [315] {frozen vegetables, ## other vegetables}</pre>	=> {whole milk}	0.009659380
0.5428571 0.017793594 2.1245523	95	0.0000000000
<pre>## [316] {frozen vegetables, ## whole milk}</pre>	=> {other vegetables	0.009659380
0.4726368 0.020437214 2.4426606 ## [317] {beef,	95	
## root vegetables} 0.4561404 0.017386884 2.3574043		0.007930859
## [318] {beef,		0.007030050
## other vegetables} 0.4020619 0.019725470 3.6886925	=> {root vegetables} 78	0.007930859
<pre>## [319] {beef, ## root vegetables}</pre>	=> {whole milk}	0.008032537
0.4619883 0.017386884 1.8080601 ## [320] {beef,	79	
## whole milk} 0.3779904 0.021250635 3.4678506	<pre>=> {root vegetables} 79</pre>	0.008032537
## [321] {beef,		
## yogurt} 0.4434783 0.011692933 2.2919646	<pre>=> {other vegetables;</pre>	} 0.005185562
<pre>## [322] {beef, ## other vegetables}</pre>	=> {yogurt}	0.005185562
0.2628866 0.019725470 1.8844677 ## [323] {beef,	51	
## yogurt} 0.5217391 0.011692933 2.0419038	<pre>=> {whole milk} 60</pre>	0.006100661
## [324] {beef,		0.00440044
## whole milk} 0.2870813 0.021250635 2.0579045	=> {yogurt} 60	0.006100661
## [325] {beef, ## rolls/buns}	=> {other vegetables	} 0.005795628
0.4253731 0.013624809 2.1983945 ## [326] {beef,	57	
## other vegetables} 0.2938144 0.019725470 1.5973825	<pre>=> {rolls/buns} 57</pre>	0.005795628
## [327] {beef,		0.000013405
## rolls/buns} 0.5000000 0.013624809 1.9568245	<pre>=> {whole milk} 67</pre>	0.006812405
## [328] {beef, ## whole milk}	=> {rolls/buns}	0.006812405
0.3205742 0.021250635 1.7428673 ## [329] {beef,	67	
## other vegetables}	<pre>=> {whole milk}</pre>	0.009252669

0.4690722 0.019725470 1.8357838 ## [330] {beef,	91		
## whole milk}	=>	{other vegetables}	0.009252669
0.4354067 0.021250635 2.2502495	91	(55	
## [331] {curd,			
## whipped/sour cream}	=>	{whole milk}	0.005897306
0.5631068 0.010472801 2.2038024	58	•	
## [332] {curd,			
## whole milk}	=>	<pre>{whipped/sour cream}</pre>	0.005897306
0.2256809 0.026131164 3.1483291	58		
## [333] {curd,			
<pre>## tropical fruit}</pre>		{yogurt}	0.005287239
0.5148515 0.010269446 3.6906446	52		
## [334] {curd,		(topping) (muit)	0 005207220
## yogurt}		{tropical fruit}	0.00528/239
0.3058824 0.017285206 2.9150707 ## [335] {curd,	52		
## [555] {curu, ## tropical fruit}	- \	{other vegetables}	0 005287230
0.5148515 0.010269446 2.6608326	52	(other vegetables)	0.003267239
## [336] {curd,	72		
## other vegetables}	=>	{tropical fruit}	0.005287239
0.3076923 0.017183528 2.9323196		(c. opicui uic)	0.003207233
## [337] {curd,			
<pre>## tropical fruit}</pre>	=>	{whole milk}	0.006507372
0.6336634 0.010269446 2.4799360	64		
## [338] {curd,			
## whole milk}	=>	{tropical fruit}	0.006507372
0.2490272 0.026131164 2.3732392	64		
## [339] {curd,			
<pre>## root vegetables}</pre>		<pre>{other vegetables}</pre>	0.005490595
0.5046729 0.010879512 2.6082280	54		
## [340] {curd,			
## other vegetables}		{root vegetables}	0.005490595
0.3195266 0.017183528 2.9314780	54		
## [341] {curd,	_,	(uhala milk)	0.006202220
## root vegetables} 0.5700935 0.010879512 2.2311457		{whore wirk}	0.006202339
## [342] {curd,	61		
## [542] (curu, ## whole milk)	->	<pre>{root vegetables}</pre>	0.006202339
0.2373541 0.026131164 2.1775909	61	(1000 vegetables)	0.000202333
## [343] {curd,	0-		
## yogurt}	=>	<pre>{other vegetables}</pre>	0.006100661
0.3529412 0.017285206 1.8240549	60		
## [344] {curd,			
<pre>## other vegetables}</pre>	=>	{yogurt}	0.006100661
0.3550296 0.017183528 2.5449825	60		
## [345] {curd,			
## yogurt}	=>	<pre>{whole milk}</pre>	0.010066090
0.5823529 0.017285206 2.2791250	99		
## [346] {curd,			

<pre>## whole milk} 0.3852140 0.026131164 2.7613555 ## [347] {curd,</pre>	=> 99	{yogurt}	0.010066090
## rolls/buns} 0.5858586 0.010066090 2.2928449	=> 58	{whole milk}	0.005897306
## [348] {curd, ## whole milk} 0.2256809 0.026131164 1.2269607	=> 58	{rolls/buns}	0.005897306
## [349] {curd, ## other vegetables} 0.5739645 0.017183528 2.2462956	=> 97	{whole milk}	0.009862735
## [350] {curd, ## whole milk}	=>	{other vegetables}	0.009862735
0.3774319 0.026131164 1.9506268 ## [351] {napkins, ## yogurt}	97	<pre>{whole milk}</pre>	0.006100661
0.4958678 0.012302999 1.9406524 ## [352] {napkins,	60	,	
<pre>## whole milk} 0.3092784 0.019725470 2.2170208 ## [353] {napkins,</pre>	=> 60	{yogurt}	0.006100661
## rolls/buns} 0.4521739 0.011692933 1.7696500	=> 52	<pre>{whole milk}</pre>	0.005287239
<pre>## [354] {napkins, ## whole milk} 0.2680412 0.019725470 1.4572612</pre>	=> 52	{rolls/buns}	0.005287239
<pre>## [355] {napkins, ## other vegetables}</pre>	=>	{whole milk}	0.006812405
<pre>0.4718310 0.014438231 1.8465809 ## [356] {napkins, ## whole milk}</pre>	67 =>	<pre>{other vegetables}</pre>	0.006812405
0.3453608 0.019725470 1.7848785 ## [357] {pork,	67		
<pre>## root vegetables} 0.5149254 0.013624809 2.6612144 ## [358] {other vegetables,</pre>	=> 69	<pre>{other vegetables}</pre>	0.007015760
## pork} 0.3239437 0.021657346 2.9720018	=> 69	<pre>{root vegetables}</pre>	0.007015760
<pre>## [359] {pork, ## root vegetables} 0.5000000 0.013624809 1.9568245</pre>	=> 67	{whole milk}	0.006812405
<pre>## [360] {pork, ## whole milk} 0.3073394 0.022165735 2.8196674</pre>	=> 67	<pre>{root vegetables}</pre>	0.006812405
## [361] {pork, ## rolls/buns}		{other vegetables}	0.005592272
<pre>0.4954955 0.011286223 2.5607978 ## [362] {other vegetables,</pre>	55 =>	<pre>{rolls/buns}</pre>	0.005592272
0.2582160 0.021657346 1.4038441	55	[1 0113/ 04113]	0.003332272

## [363] {pork, ## rolls/buns}		{whole milk}	0.006202339
0.5495495 0.011286223 2.1507441 ## [364] {pork,	61		
## whole milk} 0.2798165 0.022165735 1.5212799	=> 61	{rolls/buns}	0.006202339
<pre>## [365] {other vegetables, ## pork}</pre>	=>	{whole milk}	0.010167768
0.4694836 0.021657346 1.8373939 ## [366] {pork,	100		
## whole milk} 0.4587156 0.022165735 2.3707136	=> 100	{other vegetables}	0.010167768
<pre>## [367] {frankfurter, ## tropical fruit}</pre>	=>	{whole milk}	0.005185562
0.5483871 0.009456024 2.1461946 ## [368] {frankfurter,	51	,	
## whole milk} 0.2524752 0.020538892 2.4060989	=> 51	{tropical fruit}	0.005185562
<pre>## [369] {frankfurter, ## root vegetables}</pre>		{whole milk}	0.005083884
0.5000000 0.010167768 1.9568245 ## [370] {frankfurter,	50	(
## whole milk} 0.2475248 0.020538892 2.2709011	=> 50	<pre>{root vegetables}</pre>	0.005083884
## [371] {frankfurter, ## yogurt}		{whole milk}	0.006202339
0.5545455 0.011184545 2.1702963 ## [372] {frankfurter,	61	(WHOIC IIIIK)	0.000202333
## whole milk} 0.3019802 0.020538892 2.1647050	=> 61	{yogurt}	0.006202339
## [373] {frankfurter, ## rolls/buns}		<pre>{other vegetables}</pre>	0 005502272
0.2910053 0.019217082 1.5039606	55	(other vegetables)	0.003392272
## [374] {frankfurter, ## other vegetables}		{rolls/buns}	0.005592272
0.3395062 0.016471784 1.8457950 ## [375] {frankfurter,	55	(ba]a	0.005008083
## rolls/buns} 0.3121693 0.019217082 1.2217211	= <i>></i> 59	{whole milk}	0.005998983
## [376] {frankfurter, ## whole milk}		{rolls/buns}	0.005998983
0.2920792 0.020538892 1.5879486 ## [377] {frankfurter,	59	(ubala mille)	0.007635036
## other vegetables} 0.4629630 0.016471784 1.8118745	=> 75	{whole milk}	0.007625826
<pre>## [378] {frankfurter, ## whole milk}</pre>		{other vegetables}	0.007625826
0.3712871 0.020538892 1.9188696 ## [379] {bottled beer,	75		
## bottled water}	=>	{soda}	0.005083884

0.3225806 0.015760041 1.8499013 ## [380] {bottled beer,	50		
## soda}	=>	{bottled water}	0 005083884
0.2994012 0.016980173 2.7089336	50	(boccica water)	0.005005001
	30		
## [381] {bottled beer,			
## bottled water}	=>	{whole milk}	0.006100661
0.3870968 0.015760041 1.5149609	60		
## [382] {bottled beer,			
<pre>## whole milk}</pre>	=>	{bottled water}	0.006100661
0.2985075 0.020437214 2.7008472	60	(,	
## [383] {bottled beer,			
		{whole milk}	0.005185562
## yogurt}		(whose milk)	0.003103302
0.5604396 0.009252669 2.1933637	51		
## [384] {bottled beer,			
## whole milk}	=>	{yogurt}	0.005185562
0.2537313 0.020437214 1.8188395	51		
## [385] {bottled beer,			
## rolls/buns}	=>	{whole milk}	0.005388917
0.3955224 0.013624809 1.5479358	53	(
## [386] {bottled beer,	33		
## whole milk}		(nolls/huns)	0.005388917
		<pre>{rolls/buns}</pre>	0.005588917
0.2636816 0.020437214 1.4335591	53		
## [387] {bottled beer,			
<pre>## other vegetables}</pre>	=>	{whole milk}	0.007625826
0.4716981 0.016166751 1.8460609	75		
## [388] {bottled beer,			
<pre>## whole milk}</pre>	=>	<pre>{other vegetables}</pre>	0.007625826
0.3731343 0.020437214 1.9284162	75		
## [389] {brown bread,			
## tropical fruit}	=>	{whole milk}	0.005693950
0.5333333 0.010676157 2.0872795	56	(WHOIC MIIK)	0.0000000000
	50		
## [390] {brown bread,		(+	0.005603050
## whole milk}		{tropical fruit}	0.005693950
0.2258065 0.025216065 2.1519442	56		
## [391] {brown bread,			
<pre>## root vegetables}</pre>	=>	{whole milk}	0.005693950
0.5600000 0.010167768 2.1916435	56		
## [392] {brown bread,			
## whole milk}	=>	<pre>{root vegetables}</pre>	0.005693950
0.2258065 0.025216065 2.0716478	56	(out regressies)	0.00505550
## [393] {brown bread,	50		
		(ba] a m#] ₄]	0 005003004
## soda}		<pre>{whole milk}</pre>	0.005083884
0.4032258 0.012608033 1.5780843	50		
## [394] {brown bread,			
## whole milk}	=>	{soda}	0.005083884
0.2016129 0.025216065 1.1561883	50		
## [395] {brown bread,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.005185562
0.3566434 0.014539908 1.8431883	51	, ,	
## [396] {brown bread,			
[350] (3.0Mil bi caa)			

## other vegetables} 0.2771739 0.018708693 1.9868844 ## [397] {brown bread,	=> { <u>y</u> 51	yogurt}	0.005185562
## yogurt} 0.4895105 0.014539908 1.9157723	=> {\ 70	whole milk}	0.007117438
## [398] {brown bread, ## whole milk} 0.2822581 0.025216065 2.0233295	=> {y	yogurt}	0.007117438
## [399] {brown bread, ## rolls/buns} 0.4193548 0.012608033 1.6412077	=> {\\ 52	whole milk}	0.005287239
## [400] {brown bread, ## whole milk} 0.2096774 0.025216065 1.1399544	=> {I	rolls/buns}	0.005287239
## [401] {brown bread, ## other vegetables} 0.5000000 0.018708693 1.9568245	=> {\\ 92	whole milk}	0.009354347
<pre>## [402] {brown bread, ## whole milk} 0.3709677 0.025216065 1.9172190</pre>	=> {0	other vegetables}	0.009354347
<pre>## [403] {domestic eggs, ## margarine} 0.6219512 0.008337570 2.4340988</pre>	=> {\ 51	whole milk}	0.005185562
## [404] {margarine, ## whole milk} 0.2142857 0.024199288 3.3774038		domestic eggs}	0.005185562
<pre>## [405] {margarine, ## root vegetables}</pre>	=> {(other vegetables}	0.005897306
<pre>0.5321101 0.011082867 2.7500277 ## [406] {margarine, ## other vegetables}</pre>	58 => {ı	root vegetables}	0.005897306
0.2989691 0.019725470 2.7428739 ## [407] {margarine, ## yogurt}	58 => {(other vegetables}	0.005693950
0.4000000 0.014234875 2.0672622 ## [408] {margarine,	56	g .	
## other vegetables} 0.2886598 0.019725470 2.0692194 ## [409] {margarine,	56	yogurt}	0.005693950
## yogurt} 0.4928571 0.014234875 1.9288699 ## [410] {margarine,	=> {\\ 69	whole milk}	0.007015760
## whole milk} 0.2899160 0.024199288 2.0782241 ## [411] {margarine,	=> { <u>y</u>	yogurt}	0.007015760
## rolls/buns} 0.3517241 0.014743264 1.8177651	=> {0 51	other vegetables}	0.005185562
## [412] {margarine, ## other vegetables} 0.2628866 0.019725470 1.4292370	=> {ı 51	rolls/buns}	0.005185562

<pre>## [413] {margarine, ## rolls/buns} 0.5379310 0.014743264 2.1052733</pre>	=> {whole milk}	0.007930859
<pre>## [414] {margarine, ## whole milk} 0.3277311 0.024199288 1.7817774</pre>	=> {rolls/buns} 78	0.007930859
<pre>## [415] {margarine, ## other vegetables} 0.4690722 0.019725470 1.8357838</pre>	=> {whole milk} 91	0.009252669
<pre>## [416] {margarine, ## whole milk} 0.3823529 0.024199288 1.9760595</pre>	<pre>=> {other vegetables} 91</pre>	0.009252669
## [417] {butter, ## domestic eggs} 0.6210526 0.009659380 2.4305820	=> {whole milk} 59	0.005998983
## [418] {butter, ## whole milk} 0.2177122 0.027554652 3.4314091	=> {domestic eggs} 59	0.005998983
## [419] {butter, ## whipped/sour cream} 0.5700000 0.010167768 2.9458487	<pre>=> {other vegetables}</pre>	0.005795628
<pre>## [420] {butter, ## other vegetables}</pre>	<pre>57 => {whipped/sour cream}</pre>	0.005795628
0.2893401 0.020030503 4.0363970 ## [421] {other vegetables, ## whipped/sour cream}		0.005795628
0.2007042 0.028876462 3.6218827 ## [422] {butter, ## whipped/sour cream}		0.006710727
0.6600000 0.010167768 2.5830084 ## [423] {butter, ## whole milk}	<pre>66 => {whipped/sour cream}</pre>	0.006710727
0.2435424 0.027554652 3.3975033 ## [424] {whipped/sour cream, ## whole milk}	66	
0.2082019 0.032231825 3.7571846 ## [425] {butter,	=> {butter} 66	0.006710727
## citrus fruit} 0.5555556 0.009150991 2.1742495 ## [426] {bottled water,	=> {whole milk} 50	0.005083884
## butter} 0.6022727 0.008947636 2.3570841 ## [427] {butter,	<pre>=> {whole milk} 53</pre>	0.005388917
## tropical fruit} 0.5510204 0.009964413 2.8477592 ## [428] {butter,	<pre>=> {other vegetables} 54</pre>	0.005490595
## other vegetables} 0.2741117 0.020030503 2.6122949 ## [429] {butter,	<pre>=> {tropical fruit} 54</pre>	0.005490595
## tropical fruit}	=> {whole milk}	0.006202339

0.6224490 0.009964413 2.4360468 ## [430] {butter,	61		
## [430] {Ducter, ## whole milk}	->	{tropical fruit}	0 006202220
0.2250923 0.027554652 2.1451379	61	(cropical fruit)	0.000202333
	01		
<pre>## [431] {butter, ## root vegetables}</pre>	->	(other vegetables)	0.006609049
0.5118110 0.012913066 2.6451190	65	(other vegetables)	0.000009049
## [432] {butter,	05		
## other vegetables}	->	front vegetables	0.006609049
0.3299492 0.020030503 3.0270996		(100c vegecables)	0.000009049
## [433] {butter,	05		
## root vegetables}	=>	{whole milk}	0.008235892
0.6377953 0.012913066 2.4961069	81	(WHOLE IIILK)	0.000233032
## [434] {butter,	01		
## whole milk}	=>	<pre>{root vegetables}</pre>	0.008235892
0.2988930 0.027554652 2.7421759	81	(. oot regetables)	0.000233032
## [435] {butter,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.006405694
0.4375000 0.014641586 2.2610681	63	(cent. regetables)	
## [436] {butter,			
<pre>## other vegetables}</pre>	=>	{vogurt}	0.006405694
0.3197970 0.020030503 2.2924220		() -81	
## [437] {butter,			
## yogurt}	=>	<pre>{whole milk}</pre>	0.009354347
0.6388889 0.014641586 2.5003869	92		
## [438] {butter,			
## whole milk}	=>	{yogurt}	0.009354347
0.3394834 0.027554652 2.4335417	92		
## [439] {butter,			
## rolls/buns}	=>	<pre>{other vegetables}</pre>	0.005693950
0.4242424 0.013421454 2.1925508	56		
## [440] {butter,			
<pre>## other vegetables}</pre>	=>	{rolls/buns}	0.005693950
0.2842640 0.020030503 1.5454594	56		
## [441] {butter,			
		{whole milk}	0.006609049
0.4924242 0.013421454 1.9271757	65		
## [442] {butter,			
## whole milk}		{rolls/buns}	0.006609049
0.2398524 0.027554652 1.3040068	65		
## [443] {butter,			
<pre>## other vegetables}</pre>		<pre>{whole milk}</pre>	0.011489578
0.5736041 0.020030503 2.2448850	113		
## [444] {butter,		(ath an are 1 1 2 2	0.044400530
## whole milk}		<pre>{other vegetables}</pre>	0.011489578
0.4169742 0.027554652 2.1549874	113		
## [445] {newspapers,		(ubolo milla)	0 00500004
## tropical fruit} 0.4310345 0.011794611 1.6869177		(MIIOTE MITTK)	0.005083884
	50		
## [446] {newspapers,			

## root vegetables} 0.5221239 0.011489578 2.6984175	=> 59	<pre>{other vegetables}</pre>	0.005998983
## [447] {newspapers, ## other vegetables} 0.3105263 0.019318760 2.8489051	=> 59	<pre>{root vegetables}</pre>	0.005998983
<pre>## [448] {newspapers, ## root vegetables} 0.5044248 0.011489578 1.9741415</pre>	=> 57	{whole milk}	0.005795628
<pre>## [449] {newspapers, ## whole milk} 0.2118959 0.027351296 1.9440264</pre>	=> 57	<pre>{root vegetables}</pre>	0.005795628
## [450] {newspapers, ## yogurt}	=>	{rolls/buns}	0.005083884
<pre>0.3311258 0.015353330 1.8002336 ## [451] {newspapers, ## rolls/buns}</pre>	50 =>	{yogurt}	0.005083884
0.2577320 0.019725470 1.8475174 ## [452] {newspapers,	50		0 005502272
## yogurt} 0.3642384 0.015353330 1.8824408 ## [453] {newspapers,	= <i>></i> 55	{other vegetables}	0.005592272
<pre>## other vegetables} 0.2894737 0.019318760 2.0750537 ## [454] {newspapers,</pre>	=> 55	{yogurt}	0.005592272
## yogurt} 0.4304636 0.015353330 1.6846834	=> 65	{whole milk}	0.006609049
<pre>## [455] {newspapers, ## whole milk} 0.2416357 0.027351296 1.7321334</pre>	=> 65	{yogurt}	0.006609049
<pre>## [456] {newspapers, ## rolls/buns}</pre>	=>	{other vegetables}	0.005490595
0.2783505 0.019725470 1.4385588 ## [457] {newspapers, ## other vegetables}	54 =>	{rolls/buns}	0.005490595
<pre>0.2842105 0.019318760 1.5451689 ## [458] {newspapers, ## rolls/buns}</pre>	54 =>	{whole milk}	0.007625826
0.3865979 0.019725470 1.5130086 ## [459] {newspapers,	75	,	
## whole milk} 0.2788104 0.027351296 1.5158100 ## [460] {newspapers,	=> 75	{rolls/buns}	0.007625826
## other vegetables} 0.4315789 0.019318760 1.6890485	=> 82	{whole milk}	0.008337570
<pre>## [461] {newspapers, ## whole milk} 0.3048327 0.027351296 1.5754229</pre>	=> 82	{other vegetables}	0.008337570
## [462] {domestic eggs, ## whipped/sour cream} 0.5102041 0.009964413 2.6368141	=> 50	{other vegetables}	0.005083884
1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.	23		

<pre>## [463] {domestic eggs, ## other vegetables} => {whipped/sour cream} 0.00508388 0.2283105 0.022267412 3.1850125 50 ## [464] {domestic eggs,</pre>	4
## [464] {domestic eggs,	
## whipped/sour cream} => {whole milk} 0.00569395 0.5714286 0.009964413 2.2363709 56	0
<pre>## [465] {domestic eggs, ## pip fruit} => {whole milk} 0.00538891 0.6235294 0.008642603 2.4402753 53</pre>	7
<pre>## [466] {citrus fruit, ## domestic eggs} => {whole milk} 0.00569395 0.5490196 0.010371124 2.1486701 56</pre>	0
<pre>## [467] {domestic eggs, ## tropical fruit} => {whole milk} 0.00691408 0.6071429 0.011387900 2.3761441 68</pre>	2
<pre>## [468] {domestic eggs, ## whole milk} => {tropical fruit} 0.00691408 0.2305085 0.029994916 2.1967547 68</pre>	2
## [469] {domestic eggs, ## root vegetables} => {other vegetables} 0.00732079 0.5106383 0.014336553 2.6390582 72	3
<pre>## [470] {domestic eggs, ## other vegetables} => {root vegetables} 0.00732079</pre>	3
<pre>## [471] {domestic eggs, ## root vegetables} => {whole milk} 0.00854092</pre>	5
<pre>0.5957447 0.014336553 2.3315356 84 ## [472] {domestic eggs, ## whole milk} => {root vegetables} 0.00854092</pre>	5
0.2847458 0.029994916 2.6123830 84 ## [473] {domestic eggs, ## soda} => {other vegetables} 0.00508388	4
0.4098361 0.012404677 2.1180965 50 ## [474] {domestic eggs,	
<pre>## other vegetables} => {soda} 0.00508388 0.2283105 0.022267412 1.3092908 50 ## [475] {domestic eggs,</pre>	
<pre>## soda} => {whole milk} 0.00518556 0.4180328 0.012404677 1.6360336 51 ## [476] {domestic eggs,</pre>	2
<pre>## yogurt} => {other vegetables} 0.00579562 0.4042553 0.014336553 2.0892544 57 ## [477] {domestic eggs,</pre>	.8
## other vegetables} => {yogurt} 0.00579562 0.2602740 0.022267412 1.8657394 57	8
## [478] {domestic eggs, ## yogurt} => {whole milk} 0.00772750 0.5390071 0.014336553 2.1094846 76	4
## [479] {domestic eggs, ## whole milk} => {yogurt} 0.00772750	4

0.2576271 0.029994916 1.8467658	76		
## [480] {domestic eggs,		6.11	0.005007306
## rolls/buns}		<pre>{other vegetables}</pre>	0.005897306
0.3766234 0.015658363 1.9464482	58		
<pre>## [481] {domestic eggs, ## other vegetables}</pre>		{rolls/buns}	0.005897306
0.2648402 0.022267412 1.4398580	58	{1.0112/pail2}	0.005097500
## [482] {domestic eggs,	56		
## rolls/buns}	=>	{whole milk}	0.006609049
0.4220779 0.015658363 1.6518648	65	(WHOLE MILK)	0.0000000
## [483] {domestic eggs,	0,5		
## whole milk}	=>	{rolls/buns}	0.006609049
0.2203390 0.029994916 1.1979181	65		
## [484] {domestic eggs,			
<pre>## other vegetables}</pre>	=>	<pre>{whole milk}</pre>	0.012302999
0.5525114 0.022267412 2.1623358	121		
## [485] {domestic eggs,			
## whole milk}		<pre>{other vegetables}</pre>	0.012302999
0.4101695 0.029994916 2.1198197	121		
## [486] {bottled water,			
<pre>## fruit/vegetable juice}</pre>		{soda}	0.005185562
0.3642857 0.014234875 2.0890671	51		
<pre>## [487] {fruit/vegetable juice,</pre>		(144]-141	0.005405563
## soda} 0.2817680 0.018403660 2.5493908	=> 51	{bottled water}	0.005185562
	21		
<pre>## [488] {bottled water, ## fruit/vegetable juice}</pre>	->	{whole milk}	0.005795628
0.4071429 0.014234875 1.5934142	57	(MIIOTE IIITK)	0.003733028
## [489] {fruit/vegetable juice,	57		
## whole milk}	=>	{bottled water}	0.005795628
0.2175573 0.026639553 1.9684228	57	(5555254 11456.)	0,000,750=0
<pre>## [490] {fruit/vegetable juice,</pre>			
## tropical fruit}	=>	<pre>{other vegetables}</pre>	0.006609049
0.4814815 0.013726487 2.4883712	65		
<pre>## [491] {fruit/vegetable juice,</pre>			
<pre>## other vegetables}</pre>	=>	{tropical fruit}	0.006609049
0.3140097 0.021047280 2.9925242	65		
<pre>## [492] {fruit/vegetable juice,</pre>			
<pre>## tropical fruit}</pre>		{whole milk}	0.005998983
0.4370370 0.013726487 1.7104096	59		
<pre>## [493] {fruit/vegetable juice,</pre>		6	
<pre>## whole milk}</pre>		{tropical fruit}	0.005998983
0.2251908 0.026639553 2.1460774	59		
<pre>## [494] {fruit/vegetable juice,</pre>		(athan astables)	0.006600040
## root vegetables} 0.5508475 0.011997966 2.8468653		{other vegetables}	0.006609049
## [495] {fruit/vegetable juice,	65		
## [495] {Trult/vegetable juice, ## other vegetables}		<pre>{root vegetables}</pre>	0.006609049
0.3140097 0.021047280 2.8808629	65	[1 OOC VEGECADIES]	0.00000000
## [496] {fruit/vegetable juice,	0,5		
[o] (old consider Juree)			

## root vegetables} 0.5423729 0.011997966 2.1226571	=> 64	{whole milk}	0.006507372
<pre>## [497] {fruit/vegetable juice,</pre>		(most vesstables)	0.000507373
## whole milk} 0.2442748 0.026639553 2.2410847	=> 64	<pre>{root vegetables}</pre>	0.006507372
## [498] {fruit/vegetable juice,	04		
## soda}	=>	{yogurt}	0.005083884
0.2762431 0.018403660 1.9802120	50	() -8)	
<pre>## [499] {fruit/vegetable juice,</pre>			
## yogurt}	=>	{soda}	0.005083884
0.2717391 0.018708693 1.5583407	50		
<pre>## [500] {fruit/vegetable juice,</pre>			
## soda}		{whole milk}	0.006100661
0.3314917 0.018403660 1.2973422	60		
<pre>## [501] {fruit/vegetable juice, ##</pre>		(codo)	0.006100661
## whole milk} 0.2290076 0.026639553 1.3132887	= <i>></i> 60	{soda}	0.000100001
## [502] {fruit/vegetable juice,	00		
## yogurt}	=>	<pre>{other vegetables}</pre>	0.008235892
0.4402174 0.018708693 2.2751120	81	(a since it against a since it	
<pre>## [503] {fruit/vegetable juice,</pre>			
<pre>## other vegetables}</pre>	=>	<pre>{yogurt}</pre>	0.008235892
0.3913043 0.021047280 2.8050133	81		
<pre>## [504] {fruit/vegetable juice,</pre>			
## yogurt}		<pre>{whole milk}</pre>	0.009456024
0.5054348 0.018708693 1.9780943	93		
<pre>## [505] {fruit/vegetable juice, ##</pre>		(vogunt)	0 000456034
## whole milk} 0.3549618 0.026639553 2.5444968	=> 93	{yogurt}	0.009456024
## [506] {fruit/vegetable juice,	93		
## rolls/buns}	=>	{whole milk}	0.005592272
0.3846154 0.014539908 1.5052496	55	,	
<pre>## [507] {fruit/vegetable juice,</pre>			
## whole milk}	=>	<pre>{rolls/buns}</pre>	0.005592272
0.2099237 0.026639553 1.1412931	55		
<pre>## [508] {fruit/vegetable juice,</pre>			
<pre>## other vegetables}</pre>		<pre>{whole milk}</pre>	0.010472801
0.4975845 0.021047280 1.9473713	103		
<pre>## [509] {fruit/vegetable juice, ## whole milk}</pre>		(athan yagatahlas)	0 010472901
## whole milk} 0.3931298 0.026639553 2.0317558	=> 103	{other vegetables}	0.010472801
## [510] {pip fruit,	103		
## whipped/sour cream}	=>	<pre>{other vegetables}</pre>	0.005592272
0.6043956 0.009252669 3.1236105	55		
## [511] {other vegetables,			
<pre>## pip fruit}</pre>	=>	<pre>{whipped/sour cream}</pre>	0.005592272
0.2140078 0.026131164 2.9854844	55		
## [512] {pip fruit,			
## whipped/sour cream}		{whole milk}	0.005998983
0.6483516 0.009252669 2.5374208	59		

<pre>## [513] {citrus fruit, ## whipped/sour cream} 0.5233645 0.010879512 2.7048291 ## [514] {citrus fruit,</pre>	<pre>=> {other vegetables} 56</pre>	0.005693950
## [314] {Citrus Hult, ## whipped/sour cream} 0.5794393 0.010879512 2.2677219 ## [515] {citrus fruit,	<pre>=> {whole milk} 62</pre>	0.006304016
## whole milk} 0.2066667 0.030503305 2.8830733 ## [516] {sausage,	<pre>=> {whipped/sour cream} 62</pre>	0.006304016
## whipped/sour cream} 0.5617978 0.009049314 2.1986792 ## [517] {tropical fruit,	<pre>=> {whole milk} 50</pre>	0.005083884
## whipped/sour cream} 0.4485294 0.013828165 3.2152236 ## [518] {whipped/sour cream,	=> {yogurt} 61	0.006202339
## yogurt} 0.2990196 0.020742247 2.8496685 ## [519] {tropical fruit,	<pre>=> {tropical fruit} 61</pre>	0.006202339
## yogurt} 0.2118056 0.029283172 2.9547626 ## [520] {tropical fruit,	<pre>=> {whipped/sour cream} 61</pre>	0.006202339
## whipped/sour cream} 0.5661765 0.013828165 2.9260881	<pre>=> {other vegetables} 77</pre>	0.007829181
## [521] {other vegetables, ## whipped/sour cream} 0.2711268 0.028876462 2.5838485 ## [522] {other vegetables,	<pre>=> {tropical fruit} 77</pre>	0.007829181
## tropical fruit} 0.2181303 0.035892222 3.0429952 ## [523] {tropical fruit,	<pre>=> {whipped/sour cream} 77</pre>	0.007829181
## whipped/sour cream} 0.5735294 0.013828165 2.2445928 ## [524] {whipped/sour cream,	<pre>=> {whole milk} 78</pre>	0.007930859
## whole milk} 0.2460568 0.032231825 2.3449307	<pre>=> {tropical fruit} 78</pre>	0.007930859
## [525] {root vegetables, ## whipped/sour cream} 0.3750000 0.017081851 2.6881378	=> {yogurt} 63	0.006405694
## [526] {whipped/sour cream, ## yogurt} 0.3088235 0.020742247 2.8332830	<pre>=> {root vegetables} 63</pre>	0.006405694
## [527] {root vegetables, ## yogurt} 0.2480315 0.025826131 3.4601273	<pre>=> {whipped/sour cream} 63</pre>	0.006405694
## [528] {root vegetables, ## whipped/sour cream} 0.5000000 0.017081851 2.5840778	<pre>=> {other vegetables} 84</pre>	0.008540925
<pre>## [529] {other vegetables, ## whipped/sour cream}</pre>	<pre>=> {root vegetables}</pre>	0.008540925

0.2957746 0.028876462 2.7135668 ## [530] {root vegetables,	84		
## whipped/sour cream}	=>	{whole milk}	0.009456024
0.5535714 0.017081851 2.1664843	93	(WHOLE IIILIN)	0.003 13002 1
## [531] {whipped/sour cream,	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		
## whole milk}	=>	<pre>{root vegetables}</pre>	0.009456024
0.2933754 0.032231825 2.6915550	93	(100c vegetables)	0.003 13002 1
## [532] {soda,	23		
## whipped/sour cream}	=>	{whole milk}	0.005490595
0.4736842 0.011591256 1.8538337	54	(Milete mility)	01003120323
## [533] {whipped/sour cream,			
## yogurt}	=>	{other vegetables}	0.010167768
0.4901961 0.020742247 2.5334096	100	(come regresser)	
## [534] {other vegetables,			
## whipped/sour cream}	=>	{yogurt}	0.010167768
0.3521127 0.028876462 2.5240730	100		
## [535] {other vegetables,			
## yogurt}	=>	<pre>{whipped/sour cream}</pre>	0.010167768
0.2341920 0.043416370 3.2670620	100	, ,	
## [536] {whipped/sour cream,			
## yogurt}	=>	{whole milk}	0.010879512
0.5245098 0.020742247 2.0527473	107	,	
## [537] {whipped/sour cream,			
## whole milk}	=>	{yogurt}	0.010879512
0.3375394 0.032231825 2.4196066	107		
## [538] {rolls/buns,			
## whipped/sour cream}	=>	<pre>{other vegetables}</pre>	0.006710727
0.4583333 0.014641586 2.3687380	66		
## [539] {other vegetables,			
<pre>## whipped/sour cream}</pre>	=>	<pre>{rolls/buns}</pre>	0.006710727
0.2323944 0.028876462 1.2634597	66		
## [540] {rolls/buns,			
<pre>## whipped/sour cream}</pre>	=>	{whole milk}	0.007829181
0.5347222 0.014641586 2.0927151	77		
<pre>## [541] {whipped/sour cream,</pre>			
## whole milk}	=>	{rolls/buns}	0.007829181
0.2429022 0.032231825 1.3205877	77		
<pre>## [542] {other vegetables,</pre>			
<pre>## whipped/sour cream}</pre>	=>	{whole milk}	0.014641586
0.5070423 0.028876462 1.9843854	144		
<pre>## [543] {whipped/sour cream,</pre>			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.014641586
0.4542587 0.032231825 2.3476795	144		
## [544] {pastry,			
## pip fruit}	=>	{whole milk}	0.005083884
0.4761905 0.010676157 1.8636424	50		
## [545] {citrus fruit,			
## pip fruit}		{tropical fruit}	0.005592272
0.4044118 0.013828165 3.8540598	55		
## [546] {pip fruit,			

<pre>## tropical fruit} 0.2736318 0.020437214 3.3061046 ## [547] {citrus fruit,</pre>		{citrus fruit}	0.005592272
## tropical fruit} 0.2806122 0.019928826 3.7094374 ## [548] {citrus fruit,	=> 55	{pip fruit}	0.005592272
## pip fruit} 0.4264706 0.013828165 2.2040663 ## [549] {other vegetables,	=> 58	<pre>{other vegetables}</pre>	0.005897306
## pip fruit} 0.2256809 0.026131164 2.7267469 ## [550] {citrus fruit,	=> 58	{citrus fruit}	0.005897306
## other vegetables} 0.2042254 0.028876462 2.6996725	=> 58	{pip fruit}	0.005897306
## [551] {citrus fruit, ## pip fruit} 0.3750000 0.013828165 1.4676184	=> 51	{whole milk}	0.005185562
## [552] {pip fruit, ## sausage} 0.5188679 0.010777834 2.0306669	=> 55	{whole milk}	0.005592272
## [553] {pip fruit, ## tropical fruit} 0.2587065 0.020437214 2.3734870	=> 52	{root vegetables}	0.005287239
<pre>## [554] {pip fruit, ## root vegetables} 0.3398693 0.015556685 3.2389674</pre>	=> 52	{tropical fruit}	0.005287239
<pre>## [555] {root vegetables, ## tropical fruit} 0.2512077 0.021047280 3.3207366</pre>	=> 52	{pip fruit}	0.005287239
<pre>## [556] {pip fruit, ## tropical fruit} 0.3134328 0.020437214 2.2468017</pre>		{yogurt}	0.006405694
## [557] {pip fruit, ## yogurt} 0.3559322 0.017996950 3.3920477		{tropical fruit}	0.006405694
<pre>## [558] {tropical fruit, ## yogurt}</pre>	=>	{pip fruit}	0.006405694
<pre>0.2187500 0.029283172 2.8916751 ## [559] {pip fruit, ## tropical fruit}</pre>	63 =>	<pre>{other vegetables}</pre>	0.009456024
<pre>0.4626866 0.020437214 2.3912361 ## [560] {other vegetables, ## pip fruit}</pre>	93	{tropical fruit}	0.009456024
0.3618677 0.026131164 3.4486132 ## [561] {other vegetables,	93		
## tropical fruit} 0.2634561 0.035892222 3.4826487 ## [562] {pip fruit,	93	{pip fruit}	0.009456024
## tropical fruit} 0.4129353 0.020437214 1.6160839	=> 83	{whole milk}	0.008439248

## [563] {pip fruit, ## whole milk}		{tropical fruit}	0.008439248
0.2804054 0.030096594 2.6722744 ## [564] {pip fruit,	83	(veguet)	0.005287239
## root vegetables} 0.3398693 0.015556685 2.4363079 ## [565] {pip fruit,	52	{yogurt}	0.005287259
## yogurt} 0.2937853 0.017996950 2.6953158	=> 52	<pre>{root vegetables}</pre>	0.005287239
<pre>## [566] {root vegetables, ## yogurt}</pre>		{pip fruit}	0.005287239
0.2047244 0.025826131 2.7062696 ## [567] {pip fruit,	52		
## root vegetables} 0.5228758 0.015556685 2.7023036	=> 80	{other vegetables}	0.008134215
<pre>## [568] {other vegetables, ## pip fruit} 0.3112840 0.026131164 2.8558569</pre>	=> 80	<pre>{root vegetables}</pre>	0.008134215
## [569] {pip fruit, ## root vegetables}		{whole milk}	0.008947636
0.5751634 0.015556685 2.2509877 ## [570] {pip fruit,	88	(., .
## whole milk} 0.2972973 0.030096594 2.7275363	=> 88	<pre>{root vegetables}</pre>	0.008947636
## [571] {pip fruit, ## yogurt}		{other vegetables}	0.008134215
0.4519774 0.017996950 2.3358895 ## [572] {other vegetables,	80	(veguet)	0 000124215
## pip fruit} 0.3112840 0.026131164 2.2313984 ## [573] {pip fruit,	80	{yogurt}	0.008134215
## yogurt} 0.5310734 0.017996950 2.0784351	=> 94	<pre>{whole milk}</pre>	0.009557702
<pre>## [574] {pip fruit, ## whole milk}</pre>	=>	{yogurt}	0.009557702
0.3175676 0.030096594 2.2764410 ## [575] {pip fruit,	94		
## rolls/buns} 0.3649635 0.013929842 1.8861882	=> 50	<pre>{other vegetables}</pre>	0.005083884
<pre>## [576] {pip fruit, ## rolls/buns} 0.4452555 0.013929842 1.7425737</pre>	=> 61	{whole milk}	0.006202339
## [577] {pip fruit, ## whole milk}		{rolls/buns}	0.006202339
0.2060811 0.030096594 1.1204021 ## [578] {other vegetables,	61	(. 5225, 545)	1,000202000
## pip fruit} 0.5175097 0.026131164 2.0253514	=> 133	{whole milk}	0.013523132
## [579] {pip fruit, ## whole milk}	=>	{other vegetables}	0.013523132

0.4493243 0.030096594 2.3221780	133		
## [580] {pastry,		(7 .7)	0.005603050
## sausage}		<pre>{whole milk}</pre>	0.005693950
0.4552846 0.012506355 1.7818239	56		
## [581] {pastry,			
<pre>## tropical fruit}</pre>		<pre>{other vegetables}</pre>	0.005083884
0.3846154 0.013218099 1.9877521	50		
## [582] {other vegetables,			
## pastry}		{tropical fruit}	0.005083884
0.2252252 0.022572445 2.1464051	50		
## [583] {pastry,			
<pre>## tropical fruit}</pre>	=>	{whole milk}	0.006710727
0.5076923 0.013218099 1.9869295	66		
## [584] {pastry,			
## whole milk}	=>	{tropical fruit}	0.006710727
0.2018349 0.033248602 1.9234941	66		
## [585] {pastry,			
<pre>## root vegetables}</pre>	=>	<pre>{other vegetables}</pre>	0.005897306
0.5370370 0.010981190 2.7754909	58		
<pre>## [586] {other vegetables,</pre>			
## pastry}	=>	<pre>{root vegetables}</pre>	0.005897306
0.2612613 0.022572445 2.3969258	58	-	
## [587] {pastry,			
<pre>## root vegetables}</pre>	=>	<pre>{whole milk}</pre>	0.005693950
0.5185185 0.010981190 2.0292995	56		
## [588] {pastry,			
## soda}	=>	<pre>{rolls/buns}</pre>	0.005388917
0.2560386 0.021047280 1.3920067	53		
## [589] {pastry,			
## rolls/buns}	=>	{soda}	0.005388917
0.2572816 0.020945602 1.4754309	53		
## [590] {pastry,			
## soda}	=>	<pre>{other vegetables}</pre>	0.005490595
0.2608696 0.021047280 1.3482145	54		
## [591] {other vegetables,	_		
## pastry}	=>	{soda}	0.005490595
0.2432432 0.022572445 1.3949255	54	(
## [592] {pastry,			
## soda}	=>	{whole milk}	0.008235892
0.3913043 0.021047280 1.5314279	81	(11.1010 11.111)	0.000233032
## [593] {pastry,	01		
## whole milk}	=>	{soda}	0.008235892
0.2477064 0.033248602 1.4205205	81	(Soud)	0.000233032
## [594] {soda,	01		
## whole milk}	=>	{pastry}	0.008235892
0.2055838 0.040061007 2.3107614	81	(pastry)	3.000233032
## [595] {pastry,	01		
## yogurt}	->	{rolls/buns}	0.005795628
0.3275862 0.017691917 1.7809897	57	[1 0113/ buils]	0.005/55020
## [596] {pastry,	57		
"" [JJO] (pastry)			

## rolls/buns} 0.2766990 0.020945602 1.9834803 ## [597] {pastry,	=> {yogurt} 57	0.005795628
## yogurt} 0.3735632 0.017691917 1.9306328	<pre>=> {other vegetables} 65</pre>	0.006609049
## [598] {other vegetables, ## pastry} 0.2927928 0.022572445 2.0988463	=> {yogurt} 65	0.006609049
## [599] {pastry, ## yogurt} 0.5172414 0.017691917 2.0243012	<pre>=> {whole milk} 90</pre>	0.009150991
## [600] {pastry, ## whole milk} 0.2752294 0.033248602 1.9729451	=> {yogurt} 90	0.009150991
<pre>## [601] {pastry, ## rolls/buns} 0.2912621 0.020945602 1.5052880</pre>	<pre>=> {other vegetables} 60</pre>	0.006100661
<pre>## [602] {other vegetables, ## pastry} 0.2702703 0.022572445 1.4693798</pre>	=> {rolls/buns} 60	0.006100661
<pre>## [603] {pastry, ## rolls/buns} 0.4077670 0.020945602 1.5958569</pre>	=> {whole milk}	0.008540925
<pre>## [604] {pastry, ## whole milk} 0.2568807 0.033248602 1.3965849</pre>	=> {rolls/buns} 84	0.008540925
<pre>## [605] {other vegetables, ## pastry}</pre>	=> {whole milk}	0.010574479
<pre>0.4684685 0.022572445 1.8334212 ## [606] {pastry, ## whole milk}</pre>	<pre>104 => {other vegetables}</pre>	0.010574479
0.3180428 0.033248602 1.6436947 ## [607] {bottled water, ## citrus fruit}	<pre>104 => {other vegetables}</pre>	0.005083884
0.3759398 0.013523132 1.9429156 ## [608] {bottled water, ## other vegetables}	<pre>50 => {citrus fruit}</pre>	0.005083884
0.2049180 0.024809354 2.4758831 ## [609] {bottled water, ## citrus fruit}	50 => {whole milk}	0.005897306
0.4360902 0.013523132 1.7067041 ## [610] {citrus fruit,	58	
<pre>## tropical fruit} 0.2857143 0.019928826 2.6212687 ## [611] {citrus fruit,</pre>	<pre>=> {root vegetables} 56</pre>	
<pre>## root vegetables} 0.3218391 0.017691917 3.0671389 ## [612] {root vegetables,</pre>	<pre>=> {tropical fruit} 56</pre>	0.005693950
## tropical fruit} 0.2705314 0.021047280 3.2686441	<pre>=> {citrus fruit} 56</pre>	0.005693950

<pre>## [613] {citrus fruit, ## tropical fruit} 0.3163265 0.019928826 2.2675448</pre>		{yogurt}	0.006304016
## [614] {citrus fruit, ## yogurt}		{tropical fruit}	0.006304016
0.2910798 0.021657346 2.7740019 ## [615] {tropical fruit,	62	(cropical reals)	0.000304010
## yogurt} 0.2152778 0.029283172 2.6010528	=> 62	{citrus fruit}	0.006304016
<pre>## [616] {citrus fruit, ## tropical fruit} 0.4540816 0.019928826 2.3467645</pre>	=> 89	{other vegetables}	0.009049314
<pre>## [617] {citrus fruit, ## other vegetables} 0.3133803 0.028876462 2.9865262</pre>	=> 89	{tropical fruit}	0.009049314
<pre>## [618] {other vegetables, ## tropical fruit} 0.2521246 0.035892222 3.0462480</pre>	=> 89	{citrus fruit}	0.009049314
<pre>## [619] {citrus fruit, ## tropical fruit} 0.4540816 0.019928826 1.7771161</pre>		{whole milk}	0.009049314
<pre>## [620] {citrus fruit, ## whole milk}</pre>	=>	{tropical fruit}	0.009049314
0.2966667 0.030503305 2.8272448 ## [621] {tropical fruit, ## whole milk}	89 =>	{citrus fruit}	0.009049314
0.2139423 0.042297916 2.5849172 ## [622] {citrus fruit,	89		
<pre>## root vegetables} 0.5862069 0.017691917 3.0296084 ## [623] {citrus fruit,</pre>	=> 102	<pre>{other vegetables}</pre>	0.010371124
## other vegetables} 0.3591549 0.028876462 3.2950455	=> 102	<pre>{root vegetables}</pre>	0.010371124
## [624] {other vegetables, ## root vegetables} 0.2188841 0.047381800 2.6446257	=> 102	{citrus fruit}	0.010371124
<pre>## [625] {citrus fruit, ## root vegetables} 0.5172414 0.017691917 2.0243012</pre>	=> 90	{whole milk}	0.009150991
<pre>## [626] {citrus fruit, ## whole milk} 0.3000000 0.030503305 2.7523321</pre>	=> 90	<pre>{root vegetables}</pre>	0.009150991
## [627] {citrus fruit, ## yogurt} 0.2676056 0.021657346 1.4548930	=> 57	{rolls/buns}	0.005795628
<pre>## [628] {citrus fruit, ## rolls/buns}</pre>	=>	{yogurt}	0.005795628
0.3454545 0.016776817 2.4763451 ## [629] {citrus fruit, ## yogurt}	57 =>	<pre>{other vegetables}</pre>	0.007625826
, ,		0	

0.3521127 0.021657346 1.8197731	75		
	/5		
## [630] {citrus fruit,		6 13	0.007635036
<pre>## other vegetables}</pre>		{yogurt}	0.007625826
0.2640845 0.028876462 1.8930548	75		
## [631] {citrus fruit,			
## yogurt}	=>	{whole milk}	0.010269446
0.4741784 0.021657346 1.8557678	101		
## [632] {citrus fruit,			
## whole milk}	=>	{yogurt}	0.010269446
0.3366667 0.030503305 2.4133503	101	0 - 8 3	
## [633] {citrus fruit,			
## rolls/buns}	=>	<pre>{other vegetables}</pre>	0 005998983
0.3575758 0.016776817 1.8480071	59	(other vegetables)	0.0000000
	23		
## [634] {citrus fruit,		(malla /h.mal	0 00500000
<pre>## other vegetables}</pre>		<pre>{rolls/buns}</pre>	0.005998983
0.2077465 0.028876462 1.1294564	59		
## [635] {citrus fruit,			
## rolls/buns}	=>	{whole milk}	0.007219115
0.4303030 0.016776817 1.6840550	71		
## [636] {citrus fruit,			
## whole milk}	=>	<pre>{rolls/buns}</pre>	0.007219115
0.2366667 0.030503305 1.2866869	71		
## [637] {citrus fruit,			
## other vegetables}	=>	{whole milk}	0.013014743
0.4507042 0.028876462 1.7638982			
## [638] {citrus fruit,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.013014743
0.4266667 0.030503305 2.2050797	128	-8	
## [639] {sausage,			
## shopping bags}	=>	{soda}	0.005693950
0.3636364 0.015658363 2.0853432	56	(
## [640] {shopping bags,			
## soda}	->	{sausage}	0.005693950
0.2314050 0.024605999 2.4630604	56	(Sausage)	0.00000000000
	50		
## [641] {sausage,			0.005603050
## soda}		<pre>{shopping bags}</pre>	0.005693950
0.2343096 0.024300966 2.3781580	56		
## [642] {sausage,			
<pre>## shopping bags}</pre>	=>	{rolls/buns}	0.005998983
0.3831169 0.015658363 2.0828936	59		
## [643] {rolls/buns,			
<pre>## shopping bags}</pre>	=>	{sausage}	0.005998983
0.3072917 0.019522115 3.2707939	59		
## [644] {sausage,			
## shopping bags}	= \	{other vegetables}	0.005388917
0.3441558 0.015658363 1.7786509	53	(orner vegerantes)	0.00000011
	23		
## [645] {other vegetables,		(0.005300047
## shopping bags}		{sausage}	0.005388917
0.2324561 0.023182511 2.4742491	53		
## [646] {root vegetables,			

<pre>## shopping bags} 0.5158730 0.012811388 2.6661120 ## [647] {other vegetables,</pre>	<pre>=> {other vegetables} 65</pre>	0.006609049
## shopping bags} 0.2850877 0.023182511 2.6155203	<pre>=> {root vegetables} 65</pre>	0.006609049
## [648] {root vegetables, ## shopping bags} 0.4126984 0.012811388 1.6151567	<pre>=> {whole milk} 52</pre>	0.005287239
<pre>## [649] {shopping bags, ## whole milk} 0.2157676 0.024504321 1.9795473</pre>	<pre>=> {root vegetables} 52</pre>	0.005287239
## [650] {shopping bags, ## soda} 0.2561983 0.024605999 1.3928749	=> {rolls/buns} 62	0.006304016
<pre>## [651] {rolls/buns, ## shopping bags} 0.3229167 0.019522115 1.8518282</pre>	=> {soda} 62	0.006304016
## [652] {shopping bags, ## soda} 0.2190083 0.024605999 1.1318688	<pre>=> {other vegetables} 53</pre>	0.005388917
<pre>## [653] {other vegetables, ## shopping bags}</pre>	=> {soda}	0.005388917
<pre>0.2324561 0.023182511 1.3330648 ## [654] {shopping bags, ## soda}</pre>	<pre>53 => {whole milk}</pre>	0.006812405
<pre>0.2768595 0.024605999 1.0835309 ## [655] {shopping bags, ## whole milk}</pre>	67 => {soda}	0.006812405
0.2780083 0.024504321 1.5942925 ## [656] {shopping bags,	67	
## yogurt} 0.3533333 0.015251652 1.8260816 ## [657] {other vegetables,	<pre>=> {other vegetables} 53</pre>	0.005388917
## shopping bags} 0.2324561 0.023182511 1.6663310 ## [658] {shopping bags,	=> {yogurt} 53	0.005388917
## yogurt} 0.3466667 0.015251652 1.3567317	<pre>=> {whole milk} 52</pre>	0.005287239
<pre>## [659] {shopping bags, ## whole milk} 0.2157676 0.024504321 1.5467017</pre>	=> {yogurt} 52	0.005287239
<pre>## [660] {rolls/buns, ## shopping bags} 0.2708333 0.019522115 1.3997088</pre>	<pre>=> {other vegetables} 52</pre>	0.005287239
<pre>## [661] {other vegetables, ## shopping bags} 0.2280702 0.023182511 1.2399503</pre>	=> {rolls/buns} 52	0.005287239
<pre>## [662] {rolls/buns, ## shopping bags}</pre>	=> {whole milk}	0.005287239
0.2708333 0.019522115 1.0599466	52	

<pre>## [663] {shopping bags, ## whole milk} 0.2157676 0.024504321 1.1730651</pre>	=> {rolls/buns}	0.005287239
## [664] {other vegetables,		0.007625026
## shopping bags} 0.3289474 0.023182511 1.2873845 ## [665] {shopping bags,	<pre>=> {whole milk} 75</pre>	0.007625826
## whole milk} 0.3112033 0.024504321 1.6083472	<pre>=> {other veget 75</pre>	ables} 0.007625826
<pre>## [666] {bottled water, ## sausage}</pre>	=> {other veget	ables} 0.005083884
0.4237288 0.011997966 2.1898964 ## [667] {bottled water,	50	,
## other vegetables} 0.2049180 0.024809354 2.1811351	=> {sausage} 50	0.005083884
## [668] {sausage, ## tropical fruit}		ables} 0.005998983
0.4306569 0.013929842 2.2257020	59	autes? 0.003990903
<pre>## [669] {other vegetables, ## sausage}</pre>	=> {tropical fr	uit} 0.005998983
0.2226415 0.026944586 2.1217822 ## [670] {sausage,	59	
## tropical fruit} 0.5182482 0.013929842 2.0282415	<pre>=> {whole milk} 71</pre>	0.007219115
<pre>## [671] {sausage, ## whole milk}</pre>	=> {tropical fr	uit} 0.007219115
<pre>0.2414966 0.029893238 2.3014719 ## [672] {root vegetables,</pre>	71	
## sausage} 0.3469388 0.014946619 2.4869846	=> {yogurt} 51	0.005185562
## [673] {sausage, ## yogurt}	=> {root vegeta	bles} 0.005185562
0.2642487 0.019623793 2.4243340 ## [674] {root vegetables,	51	
## yogurt} 0.2007874 0.025826131 2.1371689	=> {sausage} 51	0.005185562
<pre>## [675] {root vegetables, ## sausage}</pre>	=> {other veget	ables} 0.006812405
0.4557823 0.014946619 2.3555539 ## [676] {other vegetables,	67	,
## sausage} 0.2528302 0.026944586 2.3195755	<pre>=> {root vegeta 67</pre>	bles} 0.006812405
<pre>## [677] {root vegetables, ## sausage}</pre>	=> {whole milk}	0.007727504
0.5170068 0.014946619 2.0233832 ## [678] {sausage,	76	
## whole milk} 0.2585034 0.029893238 2.3716240	<pre>=> {root vegeta 76</pre>	bles} 0.007727504
## [679] {sausage, ## soda}	-> {yogurt}	0.005592272
ππ 30ua ſ	-> (yoguit)	0.003332272

0.2301255 0.024300966 1.6496243	55		
## [680] {sausage,			
## yogurt}	=>	{soda}	0.005592272
0.2849741 0.019623793 1.6342392	55	•	
## [681] {soda,			
		(caucado)	0.005592272
, ,		{sausage}	0.005592272
0.2044610 0.027351296 2.1762701	55		
## [682] {sausage,			
## soda}	=>	{rolls/buns}	0.009659380
0.3974895 0.024300966 2.1610335	95		
## [683] {rolls/buns,			
## sausage}	=>	{soda}	0.009659380
0.3156146 0.030604982 1.8099532	95	(
## [684] {rolls/buns,			
## soda}	_\$	(caucage)	0.009659380
		{sausage}	0.009039360
0.2519894 0.038332486 2.6821598	95		
## [685] {sausage,			
## soda}	=>	<pre>{other vegetables}</pre>	0.007219115
0.2970711 0.024300966 1.5353098	71		
## [686] {other vegetables,			
## sausage}	=>	{soda}	0.007219115
0.2679245 0.026944586 1.5364652	71	,	
## [687] {other vegetables,	, _		
## soda}	->	{sausage}	0.007219115
0.2204969 0.032740214 2.3469556	71	(sausage)	0.00/219113
	/ 1		
## [688] {sausage,		6 1 7 1712	
## soda}		{whole milk}	0.006710727
0.2761506 0.024300966 1.0807566	66		
## [689] {sausage,			
## whole milk}	=>	{soda}	0.006710727
0.2244898 0.029893238 1.2873803	66		
## [690] {sausage,			
## yogurt}	=>	{rolls/buns}	0.005998983
0.3056995 0.019623793 1.6619980	59	(10113/04113)	0.005550505
	23		
## [691] {sausage,		6 (1	0 000434345
## yogurt}		<pre>{other vegetables}</pre>	0.008134215
0.4145078 0.019623793 2.1422406	80		
## [692] {other vegetables,			
## sausage}	=>	{yogurt}	0.008134215
0.3018868 0.026944586 2.1640354	80		
## [693] {sausage,			
## yogurt}	->	{whole milk}	0.008744281
0.4455959 0.019623793 1.7439058	86	(WHOIC IIIIK)	0.000744201
	80		
## [694] {sausage,		(0.000744664
## whole milk}		{yogurt}	0.008744281
0.2925170 0.029893238 2.0968694	86		
## [695] {rolls/buns,			
## sausage}	=>	<pre>{other vegetables}</pre>	0.008845958
0.2890365 0.030604982 1.4937858	87		
## [696] {other vegetables,			

## sausage} 0.3283019 0.026944586 1.7848806 ## [697] {other vegetables,	=> 87	{rolls/buns}	0.008845958
## rolls/buns} 0.2076372 0.042602949 2.2100781	=> 87	{sausage}	0.008845958
## [698] {rolls/buns, ## sausage} 0.3056478 0.030604982 1.1961984	=> 92	{whole milk}	0.009354347
## [699] {sausage, ## whole milk} 0.3129252 0.029893238 1.7012820	=> 92	{rolls/buns}	0.009354347
## [700] {other vegetables, ## sausage} 0.3773585 0.026944586 1.4768487	=> 100	{whole milk}	0.010167768
<pre>## [701] {sausage, ## whole milk} 0.3401361 0.029893238 1.7578760</pre>	=> 100	{other vegetables}	0.010167768
<pre>## [702] {bottled water, ## tropical fruit} 0.2802198 0.018505338 1.6069747</pre>	=> 51	{soda}	0.005185562
<pre>## [703] {soda, ## tropical fruit}</pre>	=>	{bottled water}	0.005185562
0.2487805 0.020843925 2.2509256 ## [704] {bottled water, ## tropical fruit}	51 =>	{yogurt}	0.007117438
<pre>0.3846154 0.018505338 2.7570644 ## [705] {bottled water, ## yogurt}</pre>	70 =>	{tropical fruit}	0.007117438
0.3097345 0.022979156 2.9517819 ## [706] {tropical fruit,	70		
## yogurt} 0.2430556 0.029283172 2.1991273 ## [707] {bottled water,		{bottled water}	0.007117438
<pre>## tropical fruit} 0.2912088 0.018505338 1.5832164 ## [708] {bottled water,</pre>	=> 53	{rolls/buns}	0.005388917
## rolls/buns} 0.2226891 0.024199288 2.1222355	=> 53	{tropical fruit}	0.005388917
<pre>## [709] {rolls/buns, ## tropical fruit} 0.2190083 0.024605999 1.9815513</pre>	=> 53	{bottled water}	0.005388917
<pre>## [710] {bottled water, ## tropical fruit} 0.3351648 0.018505338 1.7321840</pre>	=> 61	{other vegetables}	0.006202339
<pre>## [711] {bottled water, ## other vegetables} 0.2500000 0.024809354 2.3825097</pre>	=> 61	{tropical fruit}	0.006202339
<pre>## [712] {bottled water, ## tropical fruit}</pre>	=>	{whole milk}	0.008032537
0.4340659 0.018505338 1.6987817	79		

<pre>## [713] {bottled water, ## whole milk} 0.2337278 0.034367056 2.2274351</pre>	=> 79	{tropical fruit}	0.008032537
## [714] {bottled water, ## root vegetables} 0.4480519 0.015658363 2.3156022	=> 69	{other vegetables}	0.007015760
## [715] {bottled water, ## other vegetables} 0.2827869 0.024809354 2.5944114	=> 69	{root vegetables}	0.007015760
## [716] {bottled water, ## root vegetables} 0.4675325 0.015658363 1.8297580	=> 72	{whole milk}	0.007320793
## [717] {bottled water, ## whole milk} 0.2130178 0.034367056 1.9543186	=> 72	<pre>{root vegetables}</pre>	0.007320793
## [718] {bottled water, ## soda} 0.2561404 0.028978139 1.8361081	=> 73	{yogurt}	0.007422471
## [719] {bottled water, ## yogurt} 0.3230088 0.022979156 1.8523569	=> 73	{soda}	0.007422471
## [720] {soda, ## yogurt} 0.2713755 0.027351296 2.4553613	=> 73	{bottled water}	0.007422471
## [721] {bottled water, ## soda} 0.2350877 0.028978139 1.2781027	=> 67	{rolls/buns}	0.006812405
## [722] {bottled water, ## rolls/buns} 0.2815126 0.024199288 1.6143886	=> 67	{soda}	0.006812405
<pre>## [723] {bottled water, ## other vegetables} 0.2295082 0.024809354 1.3161593</pre>	=> 56	{soda}	0.005693950
<pre>## [724] {bottled water, ## soda} 0.2596491 0.028978139 1.0161755</pre>	=> 74	{whole milk}	0.007524148
<pre>## [725] {bottled water, ## whole milk} 0.2189349 0.034367056 1.2555247</pre>	=> 74	{soda}	0.007524148
<pre>## [726] {bottled water, ## yogurt} 0.3097345 0.022979156 1.6839353</pre>	=> 70	{rolls/buns}	0.007117438
<pre>## [727] {bottled water, ## rolls/buns} 0.2941176 0.024199288 2.1083433</pre>	=> 70	{yogurt}	0.007117438
## [728] {rolls/buns, ## yogurt} 0.2071006 0.034367056 1.8738126		{bottled water}	0.007117438
## [729] {bottled water, ## yogurt}		{other vegetables}	0.008134215

0.3539823 0.022979156 1.8294356 ## [730] {bottled water,	80		
## other vegetables}	=>	{yogurt}	0.008134215
0.3278689 0.024809354 2.3502844	80	(yogur e)	0.00013 1213
## [731] {bottled water,			
## yogurt}	=>	{whole milk}	0.009659380
0.4203540 0.022979156 1.6451180	95	,	
## [732] {bottled water,			
## whole milk}	=>	{yogurt}	0.009659380
0.2810651 0.034367056 2.0147778	95		
## [733] {bottled water,			
## rolls/buns}		<pre>{other vegetables}</pre>	0.007320793
0.3025210 0.024199288 1.5634756	72		
## [734] {bottled water,			
<pre>## other vegetables}</pre>		<pre>{rolls/buns}</pre>	0.007320793
0.2950820 0.024809354 1.6042737	72		
## [735] {bottled water,		(h-1	0.000744301
## rolls/buns} 0.3613445 0.024199288 1.4141757		{whole milk}	0.008744281
## [736] {bottled water,	86		
## [/36] {bottled water, ## whole milk}	->	{rolls/buns}	0.008744281
0.2544379 0.034367056 1.3833037	86	(10113/buils)	0.000/44201
## [737] {bottled water,	00		
## other vegetables}	=>	{whole milk}	0.010777834
0.4344262 0.024809354 1.7001918	106	(
## [738] {bottled water,			
## whole milk}	=>	<pre>{other vegetables}</pre>	0.010777834
0.3136095 0.034367056 1.6207825	106		
## [739] {root vegetables,			
<pre>## tropical fruit}</pre>	=>	<pre>{yogurt}</pre>	0.008134215
0.3864734 0.021047280 2.7703835	80		
## [740] {tropical fruit,			
## yogurt}		<pre>{root vegetables}</pre>	0.008134215
0.2777778 0.029283172 2.5484556	80		
## [741] {root vegetables,		(torreignal County)	0.000434345
## yogurt}		{tropical fruit}	0.008134215
0.3149606 0.025826131 3.0015870	80		
<pre>## [742] {root vegetables, ## tropical fruit}</pre>		(nolle/hune)	0.005897306
## tropical fruit} 0.2801932 0.021047280 1.5233281	58	<pre>{rolls/buns}</pre>	0.003097300
## [743] {rolls/buns,	20		
## tropical fruit}	=>	<pre>{root vegetables}</pre>	0.005897306
0.2396694 0.024605999 2.1988328	58	(1000 vegetables)	0.005057500
## [744] {rolls/buns,	30		
## root vegetables}	=>	{tropical fruit}	0.005897306
0.2426778 0.024300966 2.3127291	58		
## [745] {root vegetables,			
## tropical fruit}	=>	<pre>{other vegetables}</pre>	0.012302999
0.5845411 0.021047280 3.0209991	121		
## [746] {other vegetables,			

0	# tropical fruit} .3427762 0.035892222 3.1447798 # [747] {other vegetables,	=> 121	<pre>{root vegetables}</pre>	0.012302999
#	<pre># root vegetables} .2596567 0.047381800 2.4745380</pre>	=> 121	{tropical fruit}	0.012302999
#	# [748] {root vegetables, # tropical fruit} .5700483 0.021047280 2.2309690	=> 118	{whole milk}	0.011997966
#	# [749] {tropical fruit, # whole milk} .2836538 0.042297916 2.6023653	=> 118	<pre>{root vegetables}</pre>	0.011997966
#	# [750] {root vegetables, # whole milk} .2453222 0.048906965 2.3379305	=> 118	{tropical fruit}	0.011997966
#	# [751] {soda, # tropical fruit} .3170732 0.020843925 2.2728970	=> 65	{yogurt}	0.006609049
#	# [752] {tropical fruit, # yogurt} .2256944 0.029283172 1.2942885	=> 65	{soda}	0.006609049
#	# [753] {soda, # yogurt} .2416357 0.027351296 2.3027975		{tropical fruit}	0.006609049
#	# [754] {soda, # tropical fruit}	=>	{rolls/buns}	0.005388917
#	.2585366 0.020843925 1.4055872 # [755] {rolls/buns, # tropical fruit}	53 =>	{soda}	0.005388917
#	.2190083 0.024605999 1.2559454 # [756] {soda, # tropical fruit}	53 =>	<pre>{other vegetables}</pre>	0 007219115
e #	.3463415 0.020843925 1.7899466 # [757] {other vegetables,	71		
0	# tropical fruit} .2011331 0.035892222 1.1534370 # [758] {other vegetables,	=> 71	{soda}	0.007219115
#	# soda} .2204969 0.032740214 2.1013440 # [759] {soda,	=> 71	{tropical fruit}	0.007219115
#	# tropical fruit} .3756098 0.020843925 1.4700048	=> 77	{whole milk}	0.007829181
#	# [760] {tropical fruit, # yogurt} .2986111 0.029283172 1.6234606	=> 86	{rolls/buns}	0.008744281
#	# [761] {rolls/buns, # tropical fruit} .3553719 0.024605999 2.5474363	=> 86	{yogurt}	0.008744281
#	# [762] {rolls/buns, # yogurt} .2544379 0.034367056 2.4248028		{tropical fruit}	0.008744281
0	.2.44373 0.034307030 2.4248028	86		

<pre>## [763] {tropical fruit, ## yogurt}</pre>		{other vegetables}	0.012302999
0.4201389 0.029283172 2.1713431 ## [764] {other vegetables,	121		
## tropical fruit} 0.3427762 0.035892222 2.4571457	=> 121	{yogurt}	0.012302999
<pre>## [765] {other vegetables, ## yogurt}</pre>	=>	{tropical fruit}	0.012302999
0.2833724 0.043416370 2.7005496 ## [766] {tropical fruit,	121		
## yogurt} 0.5173611 0.029283172 2.0247698	=> 149	{whole milk}	0.015149975
<pre>## [767] {tropical fruit, ## whole milk}</pre>	=>	{yogurt}	0.015149975
0.3581731 0.042297916 2.5675162 ## [768] {whole milk,	149		
## yogurt} 0.2704174 0.056024403 2.5770885	=> 149	{tropical fruit}	0.015149975
<pre>## [769] {rolls/buns, ## tropical fruit}</pre>	=>	<pre>{other vegetables}</pre>	0.007829181
0.3181818 0.024605999 1.6444131 ## [770] {other vegetables,	77		
## tropical fruit} 0.2181303 0.035892222 1.1859102	=> 77	{rolls/buns}	0.007829181
<pre>## [771] {rolls/buns, ## tropical fruit}</pre>	=>	{whole milk}	0.010981190
0.4462810 0.024605999 1.7465872 ## [772] {tropical fruit,	108		
## whole milk} 0.2596154 0.042297916 1.4114524	=> 108	{rolls/buns}	0.010981190
<pre>## [773] {other vegetables, ## tropical fruit}</pre>	=>	{whole milk}	0.017081851
0.4759207 0.035892222 1.8625865 ## [774] {tropical fruit,	168		
## whole milk} 0.4038462 0.042297916 2.0871397	=> 168	<pre>{other vegetables}</pre>	0.017081851
<pre>## [775] {other vegetables, ## whole milk}</pre>		{tropical fruit}	0.017081851
0.2282609 0.074834774 2.1753349 ## [776] {root vegetables,	168		
## soda} 0.4426230 0.018607016 2.2875443	=> 81	{other vegetables}	0.008235892
## [777] {other vegetables, ## soda}		<pre>{root vegetables}</pre>	0.008235892
0.2515528 0.032740214 2.3078561 ## [778] {root vegetables,	81	(c.k1., m/11.)	0.000134345
## soda} 0.4371585 0.018607016 1.7108848	=> 80	{whole milk}	0.008134215
## [779] {soda, ## whole milk}	=>	{root vegetables}	0.008134215

0.2030457 0.040061007 1.8628305 ## [780] {root vegetables,	80		
## yogurt}	=>	<pre>{rolls/buns}</pre>	0.007219115
0.2795276 0.025826131 1.5197090	71	•	
## [781] {rolls/buns,			
<pre>## root vegetables}</pre>	=>	{yogurt}	0.007219115
0.2970711 0.024300966 2.1295150	71		
## [782] {rolls/buns,			
## yogurt}	=>	<pre>{root vegetables}</pre>	0.007219115
0.2100592 0.034367056 1.9271753	71		
## [783] {root vegetables,			
## yogurt}	=>	<pre>{other vegetables}</pre>	0.012913066
0.5000000 0.025826131 2.5840778	127		
## [784] {other vegetables,			
<pre>## root vegetables}</pre>		{yogurt}	0.012913066
0.2725322 0.047381800 1.9536108	127		
## [785] {other vegetables,			
## yogurt}		<pre>{root vegetables}</pre>	0.012913066
0.2974239 0.043416370 2.7286977	127		
## [786] {root vegetables,		(h1	0.014530000
## yogurt}		<pre>{whole milk}</pre>	0.014539908
0.5629921 0.025826131 2.2033536	143		
## [787] {root vegetables,	_\$	(vegunt)	0 014520000
## whole milk} 0.2972973 0.048906965 2.1311362	=> 143	{yogurt}	0.014539908
## [788] {whole milk,	143		
## [/88] {\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	->	<pre>{root vegetables}</pre>	0.014539908
0.2595281 0.056024403 2.3810253	143	(1000 vegetables)	0.014333300
## [789] {rolls/buns,	143		
## root vegetables}	=>	<pre>{other vegetables}</pre>	0.012201322
0.5020921 0.024300966 2.5948898	120	(other vegetables)	0.012201322
## [790] {other vegetables,			
## root vegetables}	=>	{rolls/buns}	0.012201322
0.2575107 0.047381800 1.4000100	120	, ,	
## [791] {other vegetables,			
## rolls/buns}	=>	<pre>{root vegetables}</pre>	0.012201322
0.2863962 0.042602949 2.6275247	120		
## [792] {rolls/buns,			
<pre>## root vegetables}</pre>	=>	<pre>{whole milk}</pre>	0.012709710
0.5230126 0.024300966 2.0468876	125		
<pre>## [793] {root vegetables,</pre>			
## whole milk}	=>	{rolls/buns}	0.012709710
0.2598753 0.048906965 1.4128652	125		
## [794] {rolls/buns,			
<pre>## whole milk}</pre>		<pre>{root vegetables}</pre>	0.012709710
0.2244165 0.056634469 2.0588959	125		
## [795] {other vegetables,		(7 1712	0.000100511
## root vegetables}		<pre>{whole milk}</pre>	0.023182511
0.4892704 0.047381800 1.9148326	228		
## [796] {root vegetables,			

## whole milk} 0.4740125 0.048906965 2.4497702 ## [797] {other vegetables,	<pre>=> {other vegetables} 228</pre>	0.023182511
## whole milk} 0.3097826 0.074834774 2.8420820	<pre>=> {root vegetables} 228</pre>	0.023182511
<pre>## [798] {soda, ## yogurt} 0.3159851 0.027351296 1.7179181</pre>	=> {rolls/buns} 85	0.008642603
## [799] {rolls/buns, ## soda}	=> {yogurt}	0.008642603
0.2254642 0.038332486 1.6162101 ## [800] {rolls/buns,	85	
## yogurt} 0.2514793 0.034367056 1.4421567	=> {soda} 85	0.008642603
## [801] {soda, ## yogurt} 0.3048327 0.027351296 1.5754229	<pre>=> {other vegetables} 82</pre>	0.008337570
<pre>## [802] {other vegetables, ## soda}</pre>	=> {yogurt}	0.008337570
0.2546584 0.032740214 1.8254849 ## [803] {soda, ## yogurt}	<pre>82 => {whole milk}</pre>	0.010472801
0.3828996 0.027351296 1.4985348 ## [804] {soda,	103	0.010472301
## whole milk} 0.2614213 0.040061007 1.8739641	=> {yogurt} 103	0.010472801
<pre>## [805] {rolls/buns, ## soda} 0.2572944 0.038332486 1.3297376</pre>	<pre>=> {other vegetables} 97</pre>	0.009862735
## [806] {other vegetables, ## soda}	=> {rolls/buns}	0.009862735
0.3012422 0.032740214 1.6377653 ## [807] {other vegetables,	97	
## rolls/buns} 0.2315036 0.042602949 1.3276022	=> {soda} 97	0.009862735
<pre>## [808] {rolls/buns, ## soda} 0.2307692 0.038332486 0.9031498</pre>	=> {whole milk} 87	0.008845958
## [809] {soda, ## whole milk}	=> {rolls/buns}	0.008845958
0.2208122 0.040061007 1.2004908 ## [810] {other vegetables,	87	0.042020042
## soda} 0.4254658 0.032740214 1.6651240 ## [811] {soda,	<pre>=> {whole milk} 137</pre>	0.013929842
## whole milk} 0.3477157 0.040061007 1.7970490	<pre>=> {other vegetables} 137</pre>	0.013929842
## [812] {rolls/buns, ## yogurt}	=> {other vegetables}	0.011489578
0.3343195 0.034367056 1.7278153	113	

<pre>## [813] {other vegetables, ## yogurt}</pre>	->	{rolls/buns}	0.011489578
0.2646370 0.043416370 1.4387534	113	(10113/bulls)	0.011409378
## [814] {other vegetables,			
## rolls/buns}	=>	{yogurt}	0.011489578
0.2696897 0.042602949 1.9332351	113		
## [815] {rolls/buns,			
## yogurt}		<pre>{whole milk}</pre>	0.015556685
0.4526627 0.034367056 1.7715630	153		
## [816] {whole milk, ## yogurt}	->	{rolls/buns}	0.015556685
0.2776770 0.056024403 1.5096478	153	(1 0113/ bull3)	0.0100000
## [817] {rolls/buns,	133		
## whole milk}	=>	{yogurt}	0.015556685
0.2746858 0.056634469 1.9690488	153		
## [818] {other vegetables,			
## yogurt}		{whole milk}	0.022267412
0.5128806 0.043416370 2.0072345	219		
## [819] {whole milk,	>	(athon vagatables)	0 022267412
## yogurt} 0.3974592 0.056024403 2.0541308	=> 219	<pre>{other vegetables}</pre>	0.02220/412
## [820] {other vegetables,	219		
## whole milk}	=>	{yogurt}	0.022267412
0.2975543 0.074834774 2.1329789	219	() -8)	
## [821] {other vegetables,			
## rolls/buns}		<pre>{whole milk}</pre>	0.017895272
0.4200477 0.042602949 1.6439194	176		
## [822] {rolls/buns,		6	0.047005070
## whole milk}		{other vegetables}	0.017895272
0.3159785 0.056634469 1.6330258 ## [823] {other vegetables,	176		
## whole milk}	=>	{rolls/buns}	0.017895272
0.2391304 0.074834774 1.3000817	176	(10113) 54113)	0.01/0332/2
<pre>## [824] {fruit/vegetable juice,</pre>			
## other vegetables,			
## yogurt}	=>	{whole milk}	0.005083884
0.6172840 0.008235892 2.4158327	50		
<pre>## [825] {fruit/vegetable juice,</pre>			
## whole milk,	>	(athon wagatables)	0 005003004
## yogurt} 0.5376344 0.009456024 2.7785782	= <i>></i> 50	<pre>{other vegetables}</pre>	0.005083884
## [826] {fruit/vegetable juice,	50		
## other vegetables,			
## whole milk}	=>	{yogurt}	0.005083884
0.4854369 0.010472801 3.4797900	50		
## [827] {other vegetables,			
## whole milk,			
## yogurt}		<pre>{fruit/vegetable juice}</pre>	0.005083884
0.2283105 0.022267412 3.1581347	50		
## [828] {other vegetables,			

шш				
## ##	<pre>root vegetables, whipped/sour cream}</pre>	-\	{whole milk}	0.005185562
	0.008540925 2.3761441	51	(MILOTE HITTK)	0.000100002
	{root vegetables,	71		
## [023]	whipped/sour cream,			
##	whole milk}	=>	{other vegetables}	0.005185562
	0.009456024 2.8341498	51	(Other Vegetables)	0.003103302
	{other vegetables,	7-		
	whipped/sour cream,			
##	whole milk}	=>	<pre>{root vegetables}</pre>	0.005185562
	0.014641586 3.2492809	51	(. ooc vegetables)	0.003203302
	{other vegetables,			
	root vegetables,			
	whole milk}	=>	<pre>{whipped/sour cream}</pre>	0.005185562
	0.023182511 3.1204741	51		
## [832]	{other vegetables,			
##	whipped/sour cream,			
##	yogurt}	=>	<pre>{whole milk}</pre>	0.005592272
0.5500000	0.010167768 2.1525070	55		
## [833]	<pre>{whipped/sour cream,</pre>			
##	whole milk,			
##	yogurt}	=>	<pre>{other vegetables}</pre>	0.005592272
	0.010879512 2.6565286	55		
	{other vegetables,			
	whipped/sour cream,			
	whole milk}		{yogurt}	0.005592272
	0.014641586 2.7379181	55		
	{other vegetables,			
##	whole milk,			
##	yogurt}		{whipped/sour cream}	0.005592272
	6 0.022267412 3.5035137	55		
_	{other vegetables,			
	pip fruit,		(uhala milk)	0 005400505
	root vegetables} 0 0.008134215 2.6417131	=> 54	{whole milk}	0.005490595
	{pip fruit,	54		
## [03/]	root vegetables,			
##	whole milk}	->	{other vegetables}	0.005490595
	4 0.008947636 3.1713682	54	(Other Vegetables)	0.005450555
	{other vegetables,	5-		
##	pip fruit,			
##	whole milk}	=>	<pre>{root vegetables}</pre>	0.005490595
	0 0.013523132 3.7249607	54	-6	
	{other vegetables,			
##	root vegetables,			
##	whole milk}	=>	{pip fruit}	0.005490595
	0.023182511 3.1308362	54		
## [840]	{other vegetables,			
##	pip fruit,			
##	yogurt}	=>	{whole milk}	0.005083884

0.6250000 0.008134215 2.4460306 ## [841] {pip fruit, ## whole milk,	50	
<pre>## yogurt} 0.5319149 0.009557702 2.7490189 ## [842] {other vegetables, ## pip fruit,</pre>	<pre>=> {other vegetables} 50</pre>	0.005083884
## whole milk} 0.3759398 0.013523132 2.6948749	=> {yogurt} 50	0.005083884
<pre>## [843] {other vegetables, ## whole milk, ## vogurt}</pre>	=> {pip fruit}	0.005083884
0.2283105 0.022267412 3.0180562 ## [844] {citrus fruit,	=> {bib 11.dic}	0.003003004
## other vegetables, ## root vegetables} 0.5588235 0.010371124 2.1870392	<pre>=> {whole milk} 57</pre>	0.005795628
<pre>## [845] {citrus fruit, ## root vegetables, ## whole milk}</pre>	<pre>=> {other vegetables}</pre>	0.005795628
0.6333333 0.009150991 3.2731652 ## [846] {citrus fruit,	57	0.003733028
<pre>## other vegetables, ## whole milk} 0.4453125 0.013014743 4.0854929</pre>	<pre>=> {root vegetables} 57</pre>	0.005795628
<pre>## [847] {other vegetables, ## root vegetables, ## whole milk}</pre>	=> {citrus fruit}	0.005795628
0.2500000 0.023182511 3.0205774 ## [848] {root vegetables,	57	0.003733028
<pre>## tropical fruit, ## yogurt} 0.7000000 0.008134215 2.7395543</pre>	=> {whole milk} 56	0.005693950
<pre>## [849] {root vegetables, ## tropical fruit, ## whole milk)</pre>	(vegunt)	0.005602050
## whole milk} 0.4745763 0.011997966 3.4019370 ## [850] {tropical fruit,	=> {yogurt} 56	0.005693950
## whole milk, ## yogurt} 0.3758389 0.015149975 3.4481118	<pre>=> {root vegetables} 56</pre>	0.005693950
<pre>## [851] {root vegetables, ## whole milk,</pre>		
## yogurt} 0.3916084 0.014539908 3.7320432 ## [852] {other vegetables,	<pre>=> {tropical fruit} 56</pre>	0.005693950
<pre>## root vegetables, ## tropical fruit} 0.5702479 0.012302999 2.2317503</pre>	<pre>=> {whole milk} 69</pre>	0.007015760
## [853] {root vegetables,		

## ## 0.5847458	<pre>tropical fruit, whole milk} 0.011997966 3.0220571</pre>	=> 69	{other vegetables}	0.007015760
##	<pre>{other vegetables, tropical fruit, whole milk}</pre>		<pre>{root vegetables}</pre>	0.007015760
## [855]	<pre>3 0.017081851 3.7680737 {other vegetables, root vegetables, whole milk}</pre>	69	{tropical fruit}	0.007015760
0.3026316 ## [856]	other vegetables, tropical fruit,	69	(cropical riule)	0.007013700
## 0.6198347 ## [857]	yogurt} 7 0.012302999 2.4258155 {tropical fruit,	=> 75	{whole milk}	0.007625826
	whole milk, yogurt} 7 0.015149975 2.6014206 {other vegetables,	=> 75	{other vegetables}	0.007625826
	tropical fruit, whole milk} 0.017081851 3.2001640 {other vegetables,	=> 75	{yogurt}	0.007625826
## ##	whole milk, yogurt} 3 0.022267412 3.2637119	=> 75	{tropical fruit}	0.007625826
## ##	<pre>{other vegetables, root vegetables, yogurt} 2 0.012913066 2.3728423</pre>	=> 77	{whole milk}	0.007829181
	{root vegetables, whole milk, yogurt}		<pre>{other vegetables}</pre>	0.007829181
	0.014539908 2.7828530 {other vegetables, root vegetables,	77	(0.000.00.000.000)	
## [863]	whole milk} 3 0.023182511 2.4208960 {other vegetables,	=> 77	{yogurt}	0.007829181
	whole milk, yogurt} 2 0.022267412 3.2257165 {other vegetables,	=> 77	<pre>{root vegetables}</pre>	0.007829181
	rolls/buns, root vegetables} 0.012201322 1.9894383 {rolls/buns,	=> 61	{whole milk}	0.006202339
## ##	root vegetables, whole milk}	=>	{other vegetables}	0.006202339

```
0.4880000 0.012709710 2.5220599
                                   61
## [866] {other vegetables,
##
          root vegetables,
##
          whole milk}
                                    => {rolls/buns}
                                                                0.006202339
0.2675439 0.023182511 1.4545571
                                   61
## [867] {other vegetables,
##
          rolls/buns,
          whole milk}
                                    => {root vegetables}
##
                                                                0.006202339
0.3465909 0.017895272 3.1797776
## [868] {other vegetables,
##
          rolls/buns,
##
          yogurt}
                                    => {whole milk}
                                                                0.005998983
0.5221239 0.011489578 2.0434097
                                   59
## [869] {rolls/buns,
##
          whole milk,
                                    => {other vegetables}
##
          yogurt }
                                                                0.005998983
0.3856209 0.015556685 1.9929489
                                   59
## [870] {other vegetables,
          whole milk,
##
                                    => {rolls/buns}
##
          yogurt}
                                                                0.005998983
0.2694064 0.022267412 1.4646832
                                   59
## [871] {other vegetables,
##
          rolls/buns,
##
          whole milk}
                                    => {yogurt}
                                                                0.005998983
0.3352273 0.017895272 2.4030322
                                   59
```

I then determined that I wanted only associate rules that were higher than 0.5 confidence level and around .005 support. I created a subset based on this criteria. Again, you can see that the number of rules is still high.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)</pre>
```

```
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
##
           0.2
                  0.1
                                                  TRUE
                                                             5
                                                                 0.005
## maxlen target ext
##
         5 rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
                                         TRUE
##
                                    2
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [873 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
summary(sub1)
## set of 113 rules
## rule length distribution (lhs + rhs):sizes
## 2 3 4
## 1 91 21
##
##
      Min. 1st Ou. Median
                              Mean 3rd Ou.
                                               Max.
     2.000
             3.000
                     3.000
                             3.177 3.000
##
                                              4.000
## summary of quality measures:
```

```
##
       support
                         confidence
                                                                  lift
                                            coverage
                       Min.
                               :0.5021
##
   Min.
           :0.005084
                                         Min.
                                                :0.008134
                                                             Min.
                                                                    :1.974
                                                             1st Qu.:2.109
##
    1st Qu.:0.005694
                       1st Qu.:0.5221
                                         1st Qu.:0.009964
##
   Median :0.006202
                       Median :0.5484
                                         Median :0.011388
                                                             Median :2.268
                               :0.5570
                                                             Mean
##
   Mean
           :0.007315
                       Mean
                                         Mean
                                                :0.013268
                                                                    :2.394
##
    3rd Qu.:0.007931
                       3rd Qu.:0.5824
                                         3rd Qu.:0.014642
                                                             3rd Qu.:2.657
##
   Max.
          :0.022267
                       Max.
                              :0.7000
                                         Max. :0.043416
                                                             Max.
                                                                    :3.691
##
        count
##
   Min.
           : 50.00
    1st Qu.: 56.00
##
   Median : 61.00
##
##
          : 71.95
   Mean
    3rd Qu.: 78.00
##
##
   Max.
           :219.00
##
## mining info:
##
           data ntransactions support confidence
##
   baskettrans
                         9835
                                 0.005
                                              0.2
plot(sub1, method='graph')
```

Graph for 100 rules

size: support (0.005 - 0.022) color: lift (1.974 - 3.691)



```
## [2] {oil,
##
          other vegetables}
                                   => {whole milk}
                                                         0.005083884
0.5102041 0.009964413 1.996760
## [3]
       {onions,
         root vegetables}
                                   => {other vegetables} 0.005693950
##
0.6021505 0.009456024 3.112008
                                 56
## [4]
      {onions,
         whole milk}
                                   => {other vegetables} 0.006609049
0.5462185 0.012099644 2.822942
                                 65
## [5]
        {hygiene articles,
         other vegetables}
                                   => {whole milk}
                                                         0.005185562
0.5425532 0.009557702 2.123363
                                 51
## [6] {other vegetables,
                                   => {whole milk}
##
         sugar}
                                                         0.006304016
0.5849057 0.010777834 2.289115
                                 62
## [7] {long life bakery product,
##
         other vegetables}
                                   => {whole milk}
                                                         0.005693950
0.5333333 0.010676157 2.087279
                                 56
## [8]
        {cream cheese,
                                   => {whole milk}
##
         yogurt }
                                                         0.006609049
0.5327869 0.012404677 2.085141
                                 65
## [9]
       {chicken,
         root vegetables}
                                   => {other vegetables} 0.005693950
##
0.5233645 0.010879512 2.704829
## [10] {chicken,
         root vegetables}
                                   => {whole milk}
                                                         0.005998983
0.5514019 0.010879512 2.157993
## [11] {chicken,
                                   => {whole milk}
         rolls/buns}
                                                         0.005287239
0.5473684 0.009659380 2.142208
                                 52
## [12] {coffee,
         yogurt}
                                   => {whole milk}
                                                         0.005083884
0.5208333 0.009761057 2.038359
                                 50
## [13] {frozen vegetables,
          root vegetables}
##
                                   => {other vegetables} 0.006100661
0.5263158 0.011591256 2.720082
                                 60
## [14] {frozen vegetables,
##
          root vegetables}
                                   => {whole milk}
                                                         0.006202339
0.5350877 0.011591256 2.094146
                                 61
## [15] {frozen vegetables,
         other vegetables}
                                   => {whole milk}
                                                         0.009659380
0.5428571 0.017793594 2.124552
                                 95
## [16] {beef,
                                   => {whole milk}
##
         yogurt}
                                                         0.006100661
0.5217391 0.011692933 2.041904
## [17] {curd,
                                   => {whole milk}
         whipped/sour cream}
                                                         0.005897306
0.5631068 0.010472801 2.203802
                                 58
## [18] {curd,
## tropical fruit} => {yogurt}
                                                         0.005287239
```

0.5148515 0.010269446 3.690645 ## [19] {curd,	52			
## tropical fruit}	=>	{other	vegetables}	0.005287239
0.5148515 0.010269446 2.660833	52		,	
## [20] {curd,				
## tropical fruit}	=>	{whole	milk}	0.006507372
0.6336634 0.010269446 2.479936	64		•	
## [21] {curd,				
<pre>## root vegetables}</pre>	=>	{other	<pre>vegetables}</pre>	0.005490595
0.5046729 0.010879512 2.608228	54	•	o ,	
## [22] {curd,				
<pre>## root vegetables}</pre>	=>	{whole	milk}	0.006202339
0.5700935 0.010879512 2.231146	61			
## [23] {curd,				
## yogurt}	=>	{whole	milk}	0.010066090
0.5823529 0.017285206 2.279125	99			
## [24] {curd,				
## rolls/buns}	=>	{whole	milk}	0.005897306
0.5858586 0.010066090 2.292845	58			
## [25] {curd,				
<pre>## other vegetables}</pre>	=>	{whole	milk}	0.009862735
0.5739645 0.017183528 2.246296	97			
## [26] {pork,				
<pre>## root vegetables}</pre>		{other	vegetables}	0.007015760
0.5149254 0.013624809 2.661214	69			
## [27] {pork,				
## rolls/buns}		{whole	milk}	0.006202339
0.5495495 0.011286223 2.150744	61			
## [28] {frankfurter,				
<pre>## tropical fruit}</pre>		{whole	milk}	0.005185562
0.5483871 0.009456024 2.146195	51			
## [29] {frankfurter,				
## yogurt}		{whole	milk}	0.006202339
0.5545455 0.011184545 2.170296	61			
## [30] {bottled beer,		Cla = 1 =		0.005105563
## yogurt}		{wnore	milk}	0.005185562
0.5604396 0.009252669 2.193364	51			
## [31] {brown bread,		(who] o	m: 1L1	0 005602050
## tropical fruit} 0.5333333 0.010676157 2.087279	= <i>></i> 56	{whole	IIITTK }	0.005693950
## [32] {brown bread,	90			
## root vegetables}	->	{whole	milkl	0.005693950
0.5600000 0.010167768 2.191643	56	MIIOTE	IIIIIK \	0.003093930
## [33] {domestic eggs,	50			
## margarine}	= \	{whole	milkl	0.005185562
0.6219512 0.008337570 2.434099	51	CMILOTE	mark)	0.007107702
## [34] {margarine,	J.			
## root vegetables}	= >	{other	vegetables}	0.005897306
0.5321101 0.011082867 2.750028	58	(O CITCI	regerables	0.005057500
## [35] {margarine,	50			
[] (0)				

## rolls, 0.5379310 0.0147	743264 2.105273	=> 78	{whole	milk}	0.007930859	
## [36] {butter ## domest 0.6210526 0.0096	tic eggs}	=> 59	{whole	milk}	0.005998983	
	ed/sour cream}		{other	vegetables}	0.005795628	
0.5700000 0.0103 ## [38] {butter ## whippe		57 =>	{whole	milk}	0.006710727	
0.6600000 0.0103 ## [39] {butter	167768 2.583008 ^,	66		-		
## citrus 0.5555556 0.0093 ## [40] {bottle	150991 2.174249	=> 50	{whole	milk}	0.005083884	
## butter 0.6022727 0.0089	^} 947636 2.357084	=> 53	{whole	milk}	0.005388917	
## [41] {butter ## tropic 0.5510204 0.0099	cal fruit}	=> 54	{other	vegetables}	0.005490595	
## [42] {butter ## tropic	r, cal fruit}	=>	{whole	milk}	0.006202339	
0.6224490 0.0099 ## [43] {butter ## root v	^,	61	{other	vegetables}	0 006609049	
0.5118110 0.0129 ## [44] {butter	913066 2.645119 ^,	65	Cocher	vegetables	0.0000000	
## root v 0.6377953 0.0129 ## [45] {butter	913066 2.496107	=> 81	{whole	milk}	0.008235892	
## yoguri 0.6388889 0.0146	t}	=> 92	{whole	milk}	0.009354347	
## [46] {butter ## other 0.5736041 0.0200	vegetables}	=> 113	{whole	milk}	0.011489578	
## [47] {newspa	apers, vegetables}		{other	vegetables}	0.005998983	
0.5221239 0.0114 ## [48] {newspa		59	[who]o	milk}	0.005795628	
0.5044248 0.0114 ## [49] {domest	489578 1.974142	57	MIIOTE	IIIIIK }	0.003/93028	
0.5102041 0.0099		=> 50	{other	vegetables}	0.005083884	
## [50] {domest ## whippe 0.5714286 0.0099	ed/sour cream}	=> 56	{whole	milk}	0.005693950	
## [51] {domest ## pip fr	ruit}		{whole	milk}	0.005388917	
0.6235294 0.0086	042003 2.4402/5	53				

```
## [52] {citrus fruit,
         domestic eggs}
                                   => {whole milk}
                                                         0.005693950
0.5490196 0.010371124 2.148670
                                 56
## [53] {domestic eggs,
                                   => {whole milk}
##
         tropical fruit}
                                                         0.006914082
0.6071429 0.011387900 2.376144
                                 68
## [54] {domestic eggs,
                                   => {other vegetables} 0.007320793
         root vegetables}
0.5106383 0.014336553 2.639058
                                 72
## [55] {domestic eggs,
         root vegetables}
                                   => {whole milk}
                                                         0.008540925
0.5957447 0.014336553 2.331536
                                 84
## [56] {domestic eggs,
##
         yogurt }
                                   => {whole milk}
                                                         0.007727504
0.5390071 0.014336553 2.109485
                                 76
## [57] {domestic eggs,
         other vegetables}
                                   => {whole milk}
                                                         0.012302999
0.5525114 0.022267412 2.162336
                                121
## [58] {fruit/vegetable juice,
##
          root vegetables}
                                   => {other vegetables} 0.006609049
0.5508475 0.011997966 2.846865
## [59] {fruit/vegetable juice,
                                   => {whole milk}
##
         root vegetables}
                                                         0.006507372
0.5423729 0.011997966 2.122657
## [60] {fruit/vegetable juice,
         yogurt }
                                   => {whole milk}
                                                         0.009456024
0.5054348 0.018708693 1.978094
                                 93
## [61] {pip fruit,
         whipped/sour cream}
                                   => {other vegetables} 0.005592272
0.6043956 0.009252669 3.123610
                                 55
## [62] {pip fruit,
                                  => {whole milk}
         whipped/sour cream}
                                                         0.005998983
0.6483516 0.009252669 2.537421
                                 59
## [63] {citrus fruit,
##
         whipped/sour cream}
                                   => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829
                                 56
## [64] {citrus fruit,
##
         whipped/sour cream}
                                   => {whole milk}
                                                         0.006304016
0.5794393 0.010879512 2.267722
                                 62
## [65] {sausage,
         whipped/sour cream}
                                   => {whole milk}
                                                         0.005083884
0.5617978 0.009049314 2.198679
                                 50
## [66] {tropical fruit,
         whipped/sour cream}
                                  => {other vegetables} 0.007829181
##
0.5661765 0.013828165 2.926088
                                 77
## [67] {tropical fruit,
         whipped/sour cream}
                                   => {whole milk}
                                                         0.007930859
0.5735294 0.013828165 2.244593
                                 78
## [68] {root vegetables,
## whipped/sour cream} => {whole milk}
                                                         0.009456024
```

0.5535714 0.017081851 2.166484 ## [69] {whipped/sour cream,	93
## yogurt} 0.5245098 0.020742247 2.052747	=> {whole milk} 0.010879512
## [70] {rolls/buns,	
## whipped/sour cream} 0.5347222 0.014641586 2.092715	=> {whole milk} 0.007829181 77
<pre>## [71] {other vegetables, ## whipped/sour cream}</pre>	=> {whole milk} 0.014641586
0.5070423 0.028876462 1.984385 ## [72] {pip fruit,	144
## sausage} 0.5188679 0.010777834 2.030667	=> {whole milk} 0.005592272 55
<pre>## [73] {pip fruit, ## root vegetables}</pre>	=> {other vegetables} 0.008134215
0.5228758 0.015556685 2.702304 ## [74] {pip fruit,	80
## root vegetables} 0.5751634 0.015556685 2.250988	=> {whole milk} 0.008947636
## [75] {pip fruit,	
## yogurt} 0.5310734 0.017996950 2.078435	=> {whole milk} 0.009557702 94
<pre>## [76] {other vegetables, ## pip fruit}</pre>	=> {whole milk} 0.013523132
0.5175097 0.026131164 2.025351 ## [77] {pastry,	133
## tropical fruit} 0.5076923 0.013218099 1.986930	=> {whole milk} 0.006710727
<pre>## [78] {pastry, ## root vegetables}</pre>	=> {other vegetables} 0.005897306
0.5370370 0.010981190 2.775491 ## [79] {pastry,	58
## root vegetables} 0.5185185 0.010981190 2.029299	=> {whole milk} 0.005693950
## [80] {pastry,	56
## yogurt} 0.5172414 0.017691917 2.024301	=> {whole milk} 0.009150991 90
<pre>## [81] {citrus fruit, ## root vegetables}</pre>	=> {other vegetables} 0.010371124
0.5862069 0.017691917 3.029608 ## [82] {citrus fruit,	102
## root vegetables} 0.5172414 0.017691917 2.024301	=> {whole milk} 0.009150991 90
<pre>## [83] {root vegetables, ## shopping bags}</pre>	=> {other vegetables} 0.006609049
0.5158730 0.012811388 2.666112 ## [84] {sausage,	65
## tropical fruit} 0.5182482 0.013929842 2.028241	=> {whole milk} 0.007219115 71
## [85] {root vegetables,	

```
## sausage}
                                   => {whole milk}
                                                          0.007727504
0.5170068 0.014946619 2.023383
                                  76
## [86] {root vegetables,
                                    => {other vegetables} 0.012302999
         tropical fruit}
0.5845411 0.021047280 3.020999
                                 121
## [87] {root vegetables,
                                    => {whole milk}
##
          tropical fruit}
                                                          0.011997966
0.5700483 0.021047280 2.230969
                                 118
## [88] {tropical fruit,
##
                                    => {whole milk}
                                                          0.015149975
          yogurt}
                                 149
0.5173611 0.029283172 2.024770
## [89] {root vegetables,
                                    => {whole milk}
##
                                                          0.014539908
         yogurt}
0.5629921 0.025826131 2.203354
                                 143
## [90] {rolls/buns,
                                    => {other vegetables} 0.012201322
          root vegetables}
0.5020921 0.024300966 2.594890
                                 120
## [91] {rolls/buns,
          root vegetables}
##
                                    => {whole milk}
                                                          0.012709710
0.5230126 0.024300966 2.046888
                                 125
## [92] {other vegetables,
##
                                    => {whole milk}
                                                          0.022267412
         yogurt}
0.5128806 0.043416370 2.007235
                                 219
## [93] {fruit/vegetable juice,
##
          other vegetables,
##
         yogurt }
                                    => {whole milk}
                                                          0.005083884
0.6172840 0.008235892 2.415833
                                  50
## [94] {fruit/vegetable juice,
##
         whole milk,
         yogurt }
##
                                    => {other vegetables} 0.005083884
0.5376344 0.009456024 2.778578
                                  50
## [95] {other vegetables,
##
          root vegetables,
##
         whipped/sour cream}
                                    => {whole milk}
                                                          0.005185562
0.6071429 0.008540925 2.376144
                                  51
## [96] {root vegetables,
##
         whipped/sour cream,
##
         whole milk}
                                    => {other vegetables} 0.005185562
0.5483871 0.009456024 2.834150
                                  51
## [97] {other vegetables,
##
          whipped/sour cream,
                                    => {whole milk}
##
          yogurt }
                                                          0.005592272
0.5500000 0.010167768 2.152507
                                  55
## [98] {whipped/sour cream,
##
         whole milk,
                                    => {other vegetables} 0.005592272
##
          yogurt }
0.5140187 0.010879512 2.656529
                                  55
## [99] {other vegetables,
##
          pip fruit,
          root vegetables} => {whole milk} 0.005490595
```

```
0.6750000 0.008134215 2.641713
                                   54
## [100] {pip fruit,
##
          root vegetables,
                                    => {other vegetables} 0.005490595
##
          whole milk}
0.6136364 0.008947636 3.171368
                                   54
## [101] {other vegetables,
##
          pip fruit,
##
          yogurt}
                                     => {whole milk}
                                                           0.005083884
0.6250000 0.008134215 2.446031
                                   50
## [102] {pip fruit,
##
          whole milk,
##
          yogurt }
                                     => {other vegetables} 0.005083884
0.5319149 0.009557702 2.749019
                                   50
## [103] {citrus fruit,
##
          other vegetables,
          root vegetables}
                                    => {whole milk}
                                                           0.005795628
0.5588235 0.010371124 2.187039
                                   57
## [104] {citrus fruit,
##
          root vegetables,
                                     => {other vegetables} 0.005795628
##
          whole milk}
0.6333333 0.009150991 3.273165
                                   57
## [105] {root vegetables,
##
          tropical fruit,
##
          yogurt }
                                     => {whole milk}
                                                           0.005693950
0.7000000 0.008134215 2.739554
                                   56
## [106] {other vegetables,
##
          root vegetables,
                                    => {whole milk}
          tropical fruit}
##
                                                           0.007015760
0.5702479 0.012302999 2.231750
                                   69
## [107] {root vegetables,
##
          tropical fruit,
##
          whole milk}
                                    => {other vegetables} 0.007015760
0.5847458 0.011997966 3.022057
                                   69
## [108] {other vegetables,
##
          tropical fruit,
##
                                    => {whole milk}
          yogurt}
                                                           0.007625826
0.6198347 0.012302999 2.425816
                                   75
## [109] {tropical fruit,
##
          whole milk,
          yogurt}
                                     => {other vegetables} 0.007625826
##
0.5033557 0.015149975 2.601421
                                   75
## [110] {other vegetables,
##
          root vegetables,
                                    => {whole milk}
##
          yogurt}
                                                           0.007829181
0.6062992 0.012913066 2.372842
                                   77
## [111] {root vegetables,
##
          whole milk,
          yogurt}
                                    => {other vegetables} 0.007829181
##
0.5384615 0.014539908 2.782853
                                   77
## [112] {other vegetables,
```

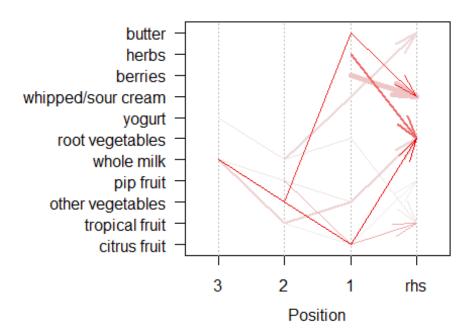
```
##
          rolls/buns,
                                    => {whole milk}
##
          root vegetables}
                                                           0.006202339
0.5083333 0.012201322 1.989438
                                  61
## [113] {other vegetables,
##
          rolls/buns,
                                     => {whole milk}
##
          yogurt}
                                                           0.005998983
0.5221239 0.011489578 2.043410
                                   59
```

To get a better idea about the association rules, I plotted the top ten association rules based on lift in a parallel coordinate plot. The y value represents the basket item or the rule itself while the x axis is the position in the association rule. As you can see, curd seems to be what people purchase with other items. Almost all of the top 10 rules have curd as the RHS. It is also interesting that if you bought curd and then tropical fruit. You are likely to buy whole milk.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)
baskets <- cbind(Baskets = rownames(baskets), baskets)</pre>
rownames(baskets) <- 1:nrow(baskets)</pre>
library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")</pre>
# baskets1= split(x=baskets 1[,-1], f=baskets$Baskets)
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                       parameter=list(support=.005, confidence=.2, maxlen=5))
## Apriori
##
```

```
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                         1 none FALSE
                                                  TRUE
                                                             5
                                                                 0.005
##
           0.2
                  0.1
## maxlen target ext
         5
           rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
                                          TRUE
##
## Absolute minimum support count: 49
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [873 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
subRules2<-head(sub1, n=20, by="lift")</pre>
subRules2 <- head(sort(basketrules, by="lift"), 10)</pre>
plot(subRules2, method="paracoord",reorder=TRUE)
```

Parallel coordinates plot for 10 rules



To get a better idea about the association rules, I plotted the top 20 rules based on lift in circular format. In addition, I also plotted the top 20 rules unformatted too as I believe that the different views provided an interesting perspective.

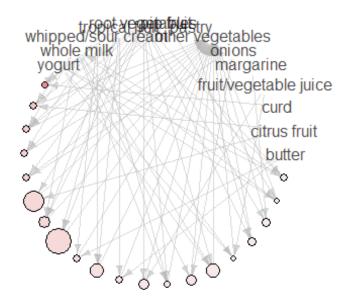
```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t", header = FALSE)
baskets <- cbind(Baskets = rownames(baskets), baskets)</pre>
rownames(baskets) <- 1:nrow(baskets)</pre>
library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")</pre>
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##
           0.2
                  0.1
                          1 none FALSE
                                                   TRUE
                                                                  0.005
## maxlen target ext
         5 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##
       0.1 TRUE TRUE FALSE TRUE
                                          TRUE
                                     2
## Absolute minimum support count: 49
```

```
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [873 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].

sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
subRules2<-head(sub1, n=20, by="lift")
subRules2 <- head(sort(basketrules, by="lift"), 10)
plot(head(sub1, 20, by='lift'), method='graph',
control=list(layout=igraph::in circle()))</pre>
```

Graph for 20 rules

size: support (0.005 - 0.012) color: lift (2.74 - 3.691)



```
plot(head(sub1, 20, by='lift'), method='graph')
```

Graph for 20 rules

size: support (0.005 - 0.012)
color: lift (2.74 - 3.691)

pastry
fruit/vegetable juice

onions

yogurt
curd
root vegetablesical fruit

pip fruit

pip fruit

margarine
whipped/sour cream

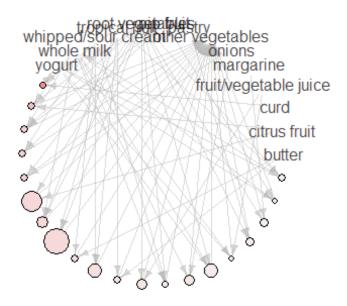
I also included grouped matrix that provided a different view of the top 20 rules. It shows very similar information as the parallel coordinate plot. I also plotted the basket items that would usually not be purchased together along with an inspection of these association rules.

```
#lower end of the lift
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules) # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t", header = FALSE)
baskets <- cbind(Baskets = rownames(baskets), baskets)</pre>
rownames(baskets) <- 1:nrow(baskets)</pre>
library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")</pre>
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)
mbaskets <- nbaskets[,-2]</pre>
mbaskets <- melt(mbaskets, id=c("Baskets"))</pre>
```

```
mbaskets <- mbaskets[,-2]</pre>
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)</pre>
baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
                  0.1
                         1 none FALSE
                                                  TRUE
                                                            5
                                                                 0.005
##
## maxlen target ext
         5 rules TRUE
##
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
       0.1 TRUE TRUE FALSE TRUE
##
                                    2
                                          TRUE
##
## Absolute minimum support count: 49
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [873 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
subRules2<-head(sub1, n=20, by="lift")</pre>
subRules2 <- head(sort(basketrules, by="lift"), 10)</pre>
plot(head(sub1, 20, by='lift'), method='graph',
control=list(layout=igraph::in circle()))
```

Graph for 20 rules

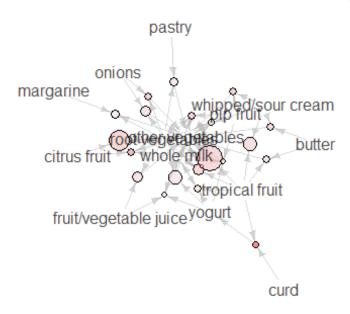
size: support (0.005 - 0.012) color: lift (2.74 - 3.691)



plot(head(sub1, 20, by='lift'), method='graph')

Graph for 20 rules

size: support (0.005 - 0.012) color: lift (2.74 - 3.691)



plot(subRules2, method = 'grouped')

Grouped Matrix for 10 Rules

```
Size: support
                          tropical fruit, other vegetables, +1 items}
                                                                    Color: lift
       :-{citrus fruit, other vegetables, +1 items}
                                      pip fruit, other vegetables, +1 items}
                                  root vegetables, yogurt, +1 items}
                              whipped/sour cream, whole milk)
                      berries, abrasive cleaner}
           butter, other vegetables}
               herbs, abrasive cleaner
   ns in LHS Group
                                         {tropical fruit, citrus fruit}
                  pip fruit, citrus fruit}
                                             {pip fruit}
inspect(tail(sort(basketrules, by = "lift")))
##
         1hs
                                      rhs
                                                        support
                                                                        confidence coverage
## [1] {misc. beverages} => {whole milk} 0.007015760 0.2473118
                                                                                       0.02836807
## [2] {chewing gum}
                                  => {whole milk} 0.005083884 0.2415459
                                                                                      0.02104728
                                  => {whole milk} 0.006507372 0.2379182
## [3] {specialty bar}
                                                                                       0.02735130
                                  => {whole milk} 0.005897306 0.2357724
## [4] {ice cream}
                                                                                       0.02501271
## [5] {rolls/buns,soda} => {whole milk} 0.008845958 0.2307692
                                                                                       0.03833249
                                  => {whole milk} 0.040061007 0.2297376 0.17437722
## [6] {soda}
##
         lift
                       count
## [1] 0.9678917
                        69
## [2] 0.9453259
                        50
## [3] 0.9311284
                        64
## [4] 0.9227303
                        58
                        87
## [5] 0.9031498
## [6] 0.8991124 394
inspect(head(sort(basketrules, by = "lift")))
##
         lhs
                                       rhs
                                                                         support confidence
                 lift count
coverage
## [1] {citrus fruit,
##
           other vegetables,
                                   => {root vegetables}
##
           whole milk}
                                                                    0.005795628
                                                                                     0.4453125
0.01301474 4.085493
                               57
## [2] {butter,
          other vegetables} => {whipped/sour cream} 0.005795628 0.2893401
```

```
0.02003050 4.036397
                      57
## [3] {herbs}
                         => {root vegetables}
                                                0.007015760 0.4312500
0.01626843 3.956477
                      69
## [4] {citrus fruit,
                         => {tropical fruit} 0.005592272 0.4044118
##
       pip fruit}
0.01382816 3.854060
                      55
                         => {whipped/sour cream} 0.009049314 0.2721713
## [5] {berries}
0.03324860 3.796886
                      89
## [6] {other vegetables,
       tropical fruit,
##
                         => {root vegetables} 0.007015760 0.4107143
##
       whole milk}
0.01708185 3.768074
                      69
```

Lastly, I plotted the larger, filtered network of basket items.

