Text Data Analytics in HappyDB

Jiun-Ying Chen, jc4878

2018/6/29

```r
# Loading all the necessary packages
library(ngram)
library(tm)
```

```
## Loading required package: NLP
```

```r
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```r
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-13
```

```r
library(text2vec)
library(data.table)
library(magrittr)
library(textmineR)
library(LiblineaR)

theme_update(plot.title = element_text(hjust = 0.5)) # Center-align ggplot title
```

```r
hm_data <- read.csv("~/Desktop/cleaned_hm.csv", stringsAsFactors = FALSE)
```

In this report, my works can be divided into two part. Start with supervised learning, I'll compare model performances under two scenarios (24 hour happy moment and 3 months happy moment) for forecasting the marital status using happy moment text data and also examine whether normalization on data can improve the model performance.

For the second part, I will do some simple hierarchical clustering analyses on the happy moment text data, basically focusing on married people and single people under two secenatios (24 hour happy moment and 3 months happy moment) and find out what are the popular words among a specific cluster using wordcloud.

Classification for marital status

In this section, my goal is to have a look at the model performance of logistic regression classifier and linear SVM classifier (L2 penalized with classic hinge-loss) for predicting the marital status which is a multi-calss response, besides, the text model of bag-of-word and n-gram (bigram and trigram) model will also be examined by applying them to multilogistic or linear SVM.

The data is divided by reflection_period (24 hour happy moment and 3 months happy moment), both of which would be further sampled out with 15,000 observations for the sake of computational efficiency. The DTM matrix data will also have additional normalized ones that applied to the models to see whether normalization in this case can improve the model performance.

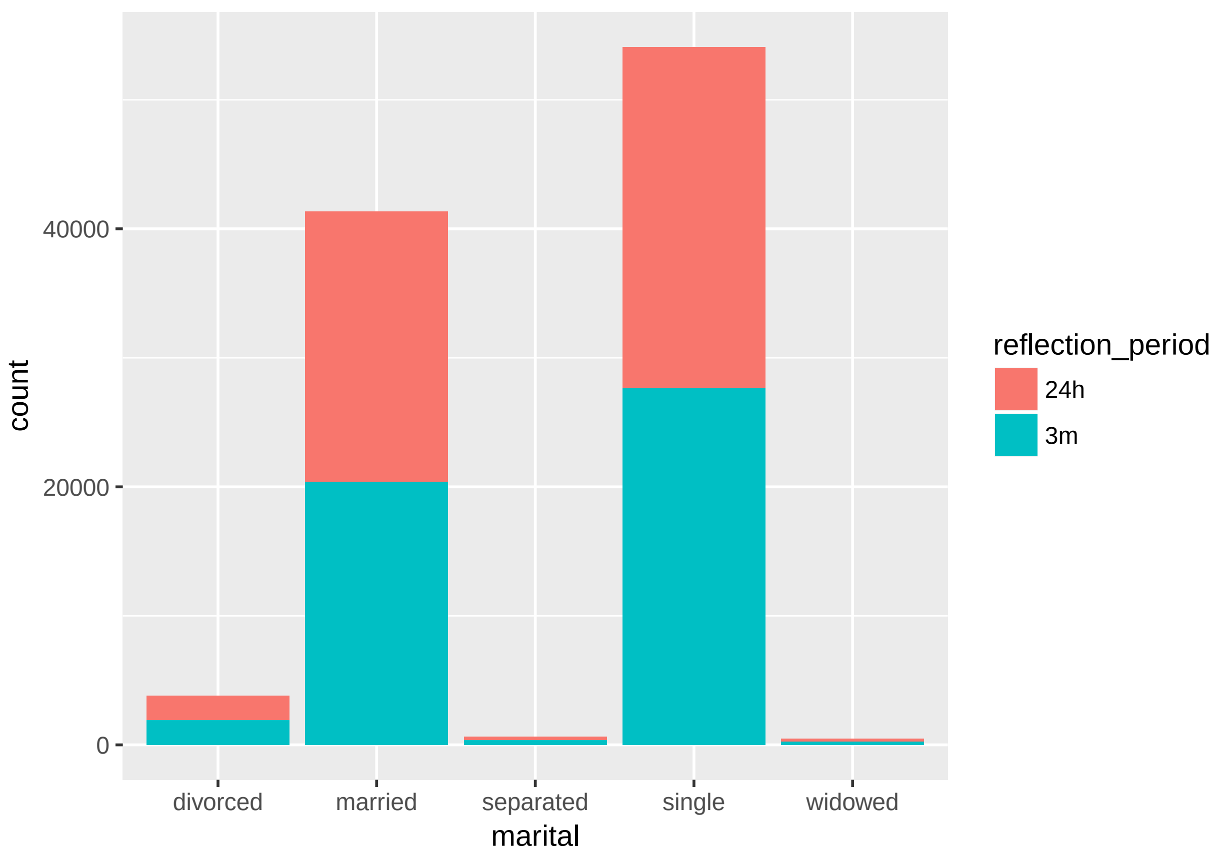
```
demo_data <- read.csv("~/Desktop/demographic.csv")
dim(demo_data)
```

```
## [1] 10844     6
```

```
# Joinging hm_data and demo_data based on "wid" column
merge_data <- merge(hm_data, demo_data, by = "wid")
dim(merge_data)
```

```
## [1] 100535     14
```

```
marital <- merge_data[c("reflection_period","cleaned_hm", "marital")]
marital_bin_data <- marital[-which(marital$marital == ""), ]
marital_bin_data <- marital[-which(factor(marital$marital) == ""), ]
marital_bin_data$marry_bin <- as.numeric(factor(marital_bin_data$marital))
ggplot(marital_bin_data, aes(marital, fill=reflection_period)) +
  stat_count()
```



As the above barplot shows, both happy moment data class with different reflection_period have roughly the same amount of data. But also notice that the data is unbalanced, where most observations belong to either married or single.

```r
set.seed(0)
marital_bin_24h = marital_bin_data[which(marital_bin_data$reflection_period=='24h'),]
marital_bin_3m = marital_bin_data[which(marital_bin_data$reflection_period=='3m'),]

marital_bin_24h = marital_bin_24h[sample(1:nrow(marital_bin_24h),15000),]
marital_bin_3m = marital_bin_3m[sample(1:nrow(marital_bin_3m),15000),]
rownames(marital_bin_24h) <- 1:nrow(marital_bin_24h)
rownames(marital_bin_3m) <- 1:nrow(marital_bin_3m)

table(marital_bin_24h$marital)
```

```
##
##              divorced    married separated    single    widowed
##          0        568       6307        88      7954         83
```

```r
table(marital_bin_3m$marital)
```

```
##
##              divorced    married separated    single    widowed
##          0        579       6055       123      8169         74
```

The above step sample out the data.

```r
# First 70% as training data, rest 30% as test data.
# 5-fold cross validation done later in the training set
set.seed(0)
sid=sample(1:(nrow(marital_bin_24h)),0.7*nrow(marital_bin_24h))
train_hm_24h <- marital_bin_24h[sid, ]
test_hm_24h <- marital_bin_24h[-sid, ]

rownames(train_hm_24h) <- 1:nrow(train_hm_24h)
rownames(test_hm_24h) <- 1:nrow(test_hm_24h)

sid=sample(1:(nrow(marital_bin_3m)),0.7*nrow(marital_bin_3m))
train_hm_3m <- marital_bin_3m[sid, ]
test_hm_3m <- marital_bin_3m[-sid, ]

rownames(train_hm_3m) <- 1:nrow(train_hm_3m)
rownames(test_hm_3m) <- 1:nrow(test_hm_3m)

table(train_hm_24h$marital)
```

```
##
##              divorced    married separated    single    widowed
##          0        385       4477        60      5518         60
```

```r
table(test_hm_24h$marital)
```

```
##
##              divorced    married separated    single    widowed
##          0        183       1830        28      2436         23
```

```r
table(train_hm_3m$marital)
```

```
##
##              divorced    married separated    single    widowed
##          0        406       4266        83      5702         43
```

```r
table(test_hm_3m$marital)
```

```
##
##            divorced   married separated    single   widowed
##         0       173      1789        40      2467        31
```

The above step divided training set and test set for each.

```r
# Helper functions:
# Text cleaning
funcs     <- function(x){
    x <- tolower(x)
    x <- sapply(x, gsub, patt = ",", replace = " ")
    x <- removePunctuation(x)
    x <- removeNumbers(x)
    x <- stripWhitespace(x)
    x <- removeWords(x, words = c(stopwords(kind = "en")))
    return(x)
}

clean_up_texts <- function(data){
  prepro_hm <- sapply(data$cleaned_hm, FUN = funcs)
  return(prepro_hm)
}

# Preprocessing function
prepro = function(data){
  pre <- clean_up_texts(data)
for(i in 1:nrow(data)){
  data$prepro[i] <- pre[[i]]
}
  return(data)
}

# Tokenization function (for bag-of-word model)
tokeniz = function(train,seed=0){
  set.seed(seed)
  prep_fun <- tolower
  tok_fun <- word_tokenizer

  it <- itoken(train$prepro,
                 preprocessor = prep_fun,
                 tokenizer = tok_fun,
                 progressbar = FALSE)
  vocab <- create_vocabulary(it)
  vectorizer <- vocab_vectorizer(vocab)
  dtm <- create_dtm(it, vectorizer)
  return(list(dtm,vectorizer))
}

tokenTest = function(test,train,seed=0){
  set.seed(seed)
  vectorizer = tokeniz(train,seed)[[2]]
  prep_fun <- tolower
  tok_fun <- word_tokenizer
```

```r
  it_test <- itoken(test$prepro,
                    preprocessor = prep_fun,
                    tokenizer = tok_fun,
                    progressbar = FALSE)

  dtm_test <- create_dtm(it_test, vectorizer)
  return(dtm_test)
}
```

```r
# Cleaned happy moments in training data
tr_hm_24h <- prepro(train_hm_24h)

# Cleaned happy moments in test data
tes_hm_24h <- prepro(test_hm_24h)
```

Bag-of-word model + multi-logistic (24h data)

Let's start with bag-of-word model as text model and see its performances by applying to multi-logistic and linear SVM

```r
# Bag-of-words model for feature selection
dtm_train = tokeniz(tr_hm_24h)[[1]]
dim(dtm_train)
```

```
## [1] 10500  9423
```

```r
ndtm_train = normalize(dtm_train)
dim(ndtm_train)
```

```
## [1] 10500  9423
```

```r
# Fitting the classifier using logistic regression
# Train original data
lg_bw_24h <- cv.glmnet(x = dtm_train, y = tr_hm_24h$marry_bin,
                       family = 'multinomial', alpha = 1,
                       nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

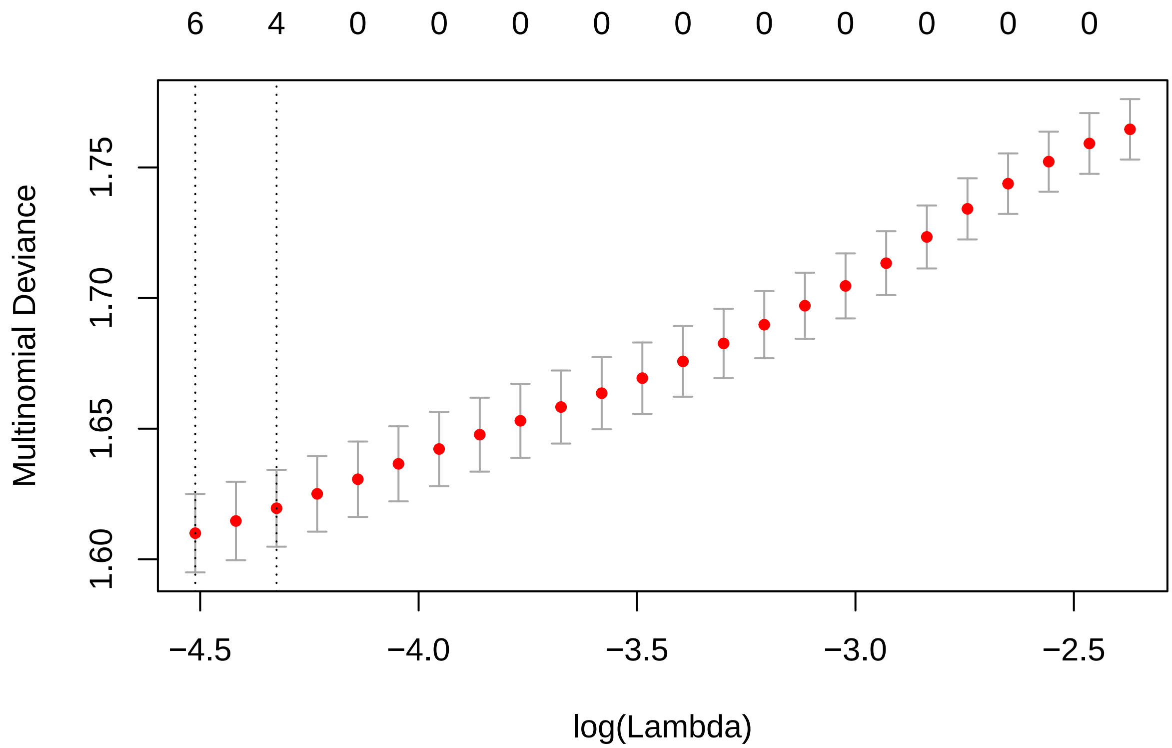
```
## Warning: from glmnet Fortran code (error code -28); Convergence for 28th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -93); Convergence for 93th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -49); Convergence for 49th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -89); Convergence for 89th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
# Train normalized data
nlg_bw_24h <- cv.glmnet(x = ndtm_train, y = tr_hm_24h$marry_bin,
```

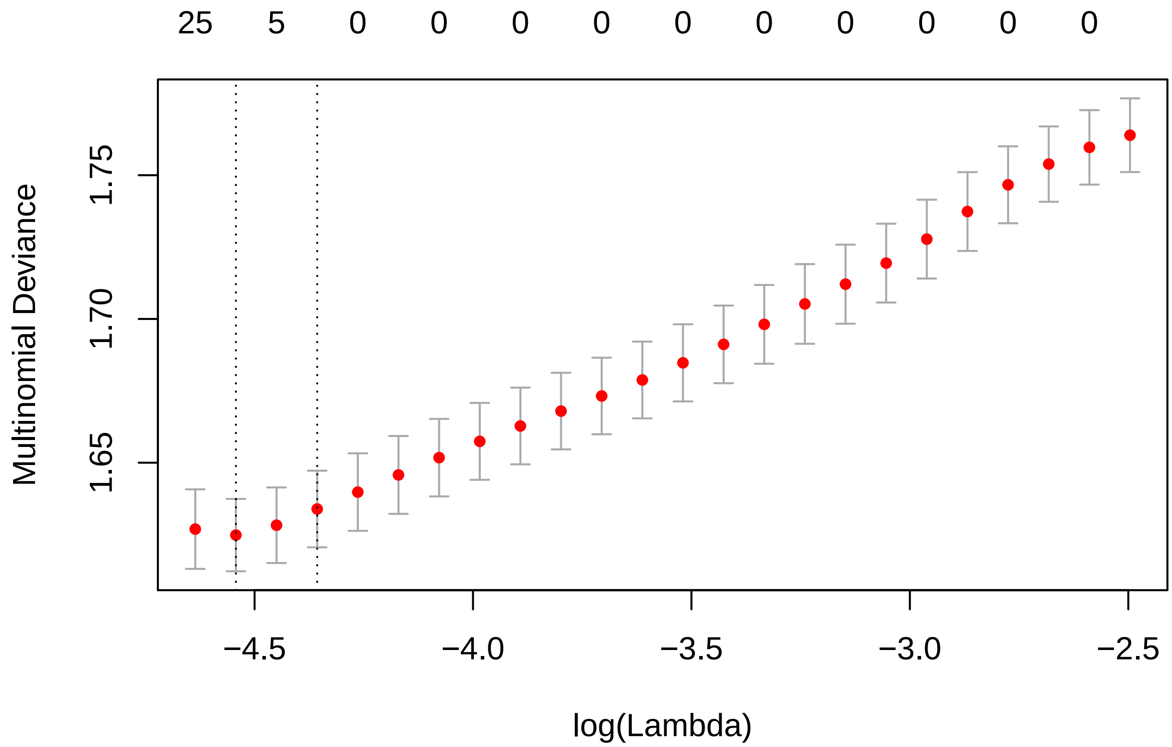
```r
                    family = 'multinomial', alpha = 1,
                    nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

```
## Warning: from glmnet Fortran code (error code -25); Convergence for 25th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -51); Convergence for 51th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -63); Convergence for 63th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -74); Convergence for 74th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
plot(lg_bw_24h)
```




```r
plot(nlg_bw_24h)
```




```
# Creating DocumentTerm Matrix for the test data
dtm_test = tokenTest(tes_hm_24h,tr_hm_24h)
ndtm_test = normalize(dtm_test)

# Making prediction of models
pred_lg_bw_24h = as.numeric(predict(lg_bw_24h, dtm_test, type = 'class'))
npred_lg_bw_24h = as.numeric(predict(nlg_bw_24h, ndtm_test, type = 'class'))
```

```
#Accuracy and tables
mean(pred_lg_bw_24h==tes_hm_24h$marry_bin)
```

```
## [1] 0.6371111
```

```
table(pred_lg_bw_24h,tes_hm_24h$marry_bin)
```

```
##
## pred_lg_bw_24h    1    2    3    4    5
##              2   34  503    8   72    7
##              4  149 1327   20 2364   16
```

```
mean(npred_lg_bw_24h==tes_hm_24h$marry_bin)
```

```
## [1] 0.6366667
```

```
table(npred_lg_bw_24h,tes_hm_24h$marry_bin)
```

```
##
## npred_lg_bw_24h    1    2    3    4    5
##               2   36  506    7   77    6
##               4  147 1324   21 2359   17
```

The model performance and prediction table of bag-of-word + multilogistic are as the above.

Bag-of-word model + Linear SVM (24h data)

```
# Train original data
svm_bw_24h = LiblineaR(data = as.matrix(dtm_train), target = tr_hm_24h$marry_bin,type = 2)
# Making predictions
pred_svm_bw_24h = predict(svm_bw_24h, as.matrix(dtm_test), type='class')
pred_svm_bw_24h = as.numeric(unlist(pred_svm_bw_24h))

# Train normalized data
nsvm_bw_24h = LiblineaR(data = as.matrix(ndtm_train), target = tr_hm_24h$marry_bin,type = 2)
# Making predictions
npred_svm_bw_24h = predict(nsvm_bw_24h, as.matrix(ndtm_test), type='class')
npred_svm_bw_24h = as.numeric(unlist(npred_svm_bw_24h))
```

Here I refer to the "LiblineaR" package for SVM modling, since the text data is really complex and large, this package is specifically suitable for large matrix training, specifically, for linear support vector machine and other linear models. For SVM throughout this report, linear SVM (primal) model with L2 regularization and hinge-loss is applied (code 2).

```
# Accuracy for SVM
mean(pred_svm_bw_24h==tes_hm_24h$marry_bin)
```

```
## [1] 0.6057778
```

```
table(pred_svm_bw_24h,tes_hm_24h$marry_bin)
```

```
##
## pred_svm_bw_24h    1    2    3    4    5
##               1    4   23    0   25    0
##               2   66 1003   11  686   13
##               3    0    2    0    4    0
##               4  112  802   17 1719   10
##               5    1    0    0    2    0
```

```
mean(npred_svm_bw_24h==tes_hm_24h$marry_bin)
```

```
## [1] 0.6448889
```

```
table(npred_svm_bw_24h,tes_hm_24h$marry_bin)
```

```
##
## npred_svm_bw_24h    1    2    3    4    5
##                1    0    0    0    1    0
##                2   59  905   10  438    8
##                4  124  925   18 1997   15
```

The model performance and prediction table of bag-of-word + linear SVM are shown as the above.

Bigram model + multi-logistic (24h data)

Now, let's examine the bigram model with combination to multi-logistic and linear SVM

```
#ngram function
token_ngram = function(train,seed=0,ngram=c(1L,2L)){
  set.seed(seed)
  prep_fun <- tolower
```

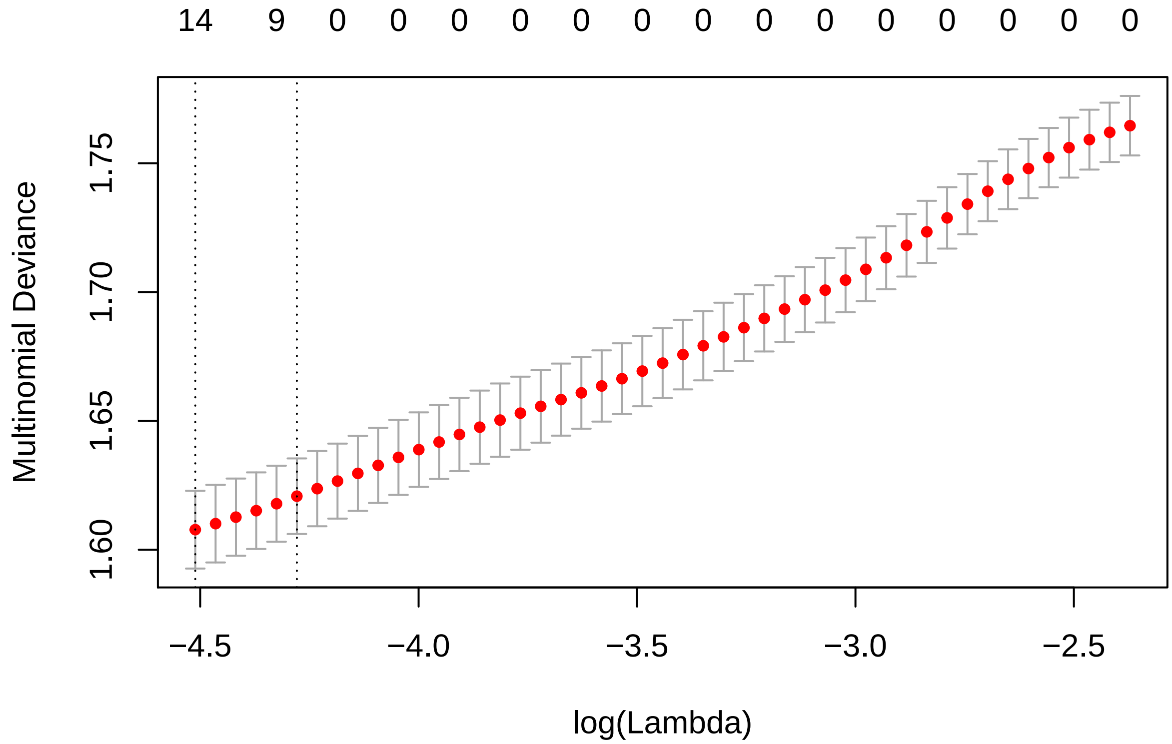
```r
  tok_fun <- word_tokenizer

  it_train <- itoken(train$prepro,
                     # We are using the processed data here. The steps are just to
                     # give you an example of usage of text2vec package
                     preprocessor = prep_fun,
                     tokenizer = tok_fun,
                     progressbar = FALSE)
  vocab = create_vocabulary(it_train, ngram = ngram)
  ngram_vectorizer = vocab_vectorizer(vocab)
  dtm_train <- create_dtm(it_train, ngram_vectorizer)
  return(list(dtm_train,ngram_vectorizer))
}

tokenTest_ngram = function(test,train,seed=0,ngram=c(1L,2L)){
  set.seed(seed)
  vectorizer = token_ngram(train,seed,ngram)[[2]]
  prep_fun <- tolower
  tok_fun <- word_tokenizer

  it_test <- itoken(test$prepro,
                     preprocessor = prep_fun,
                     tokenizer = tok_fun,
                     progressbar = FALSE)

  dtm_test <- create_dtm(it_test, vectorizer)
  return(dtm_test)
}
```

```r
dtm_train = token_ngram(tr_hm_24h)[[1]]
ndtm_train = normalize(dtm_train)

# Train original data
lg_bi_24h = cv.glmnet(x = dtm_train, y = tr_hm_24h$marry_bin,
                      family = 'multinomial', alpha = 1,
                      nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

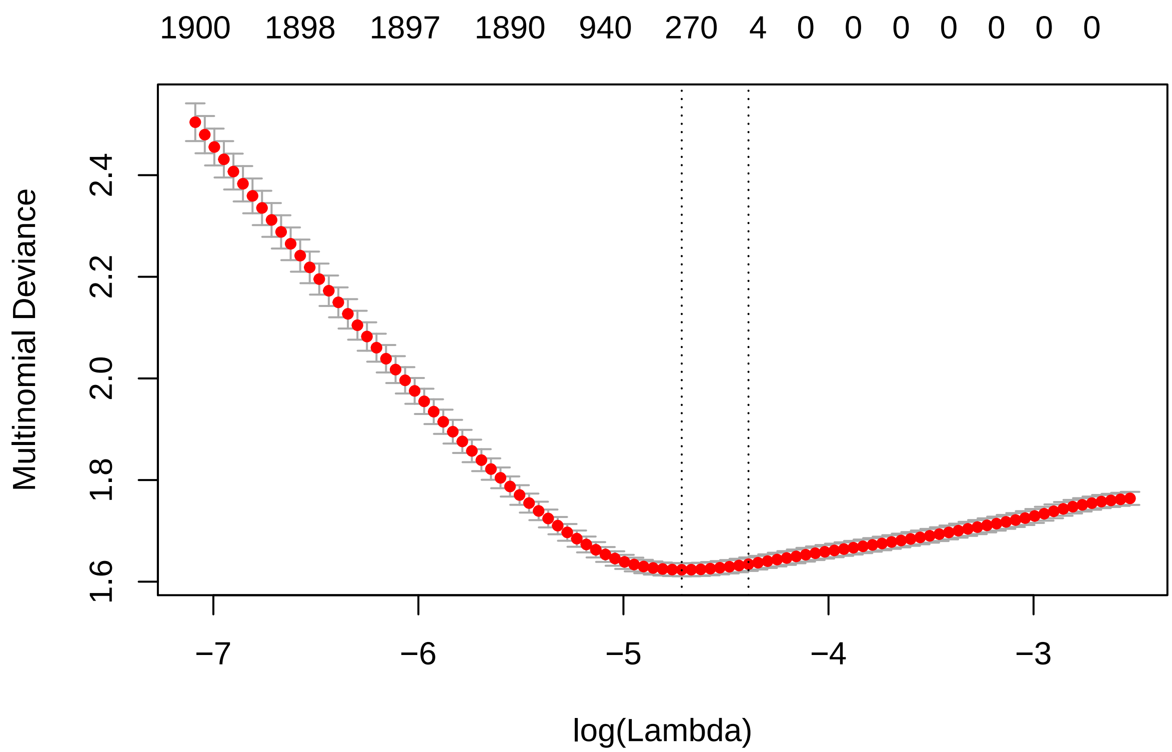
```
## Warning: from glmnet Fortran code (error code -50); Convergence for 50th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -95); Convergence for 95th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -49); Convergence for 49th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -49); Convergence for 49th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -51); Convergence for 51th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```
## Warning: from glmnet Fortran code (error code -48); Convergence for 48th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
# Train normalized data
nlg_bi_24h <- cv.glmnet(x = ndtm_train, y = tr_hm_24h$marry_bin,
                        family = 'multinomial', alpha = 1,
                        nfolds = 5, thresh = 1e-3, maxit = 1e3)
plot(lg_bi_24h)
```




```r
plot(nlg_bi_24h)
```




```
# Evaluating the performance of our classifier on test data
dtm_test = tokenTest_ngram(tes_hm_24h,tr_hm_24h)
ndtm_test = normalize(dtm_test)
pred_lg_bi_24h = as.numeric(predict(lg_bi_24h, dtm_test, type = 'class'))
npred_lg_bi_24h = as.numeric(predict(nlg_bi_24h, ndtm_test, type = 'class'))
```

```
# Accuracy and tables
mean(pred_lg_bi_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.638
```

```
table(pred_lg_bi_24h,test_hm_24h$marry_bin)
```

```
##
## pred_lg_bi_24h    1    2    3    4    5
##              2   34  515    8   80    7
##              4  149 1315   20 2356   16
```

```
mean(npred_lg_bi_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.638
```

```
table(npred_lg_bi_24h,test_hm_24h$marry_bin)
```

```
##
## npred_lg_bi_24h    1    2    3    4    5
##               1    0    1    0    3    0
##               2   35  552    7  114    7
##               4  148 1277   21 2319   16
```

The model performance and prediction table of bigram + multilogistic are shown as the above.

Bigram model + Linear SVM (24h data)

```
# Train original data
svm_bi_24h = LiblineaR(data = as.matrix(dtm_train), target = tr_hm_24h$marry_bin,type = 2)
pred_svm_bi_24h = predict(svm_bi_24h, as.matrix(dtm_test), type='class')
pred_svm_bi_24h = as.numeric(unlist(pred_svm_bi_24h))

#Train normalized data
nsvm_bi_24h = LiblineaR(data = as.matrix(ndtm_train), target = tr_hm_24h$marry_bin,type = 2)
npred_svm_bi_24h = predict(nsvm_bi_24h, as.matrix(ndtm_test), type='class')
npred_svm_bi_24h = as.numeric(unlist(npred_svm_bi_24h))
```

```
#Accuracy for SVM
mean(pred_svm_bi_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.612
```

```
table(pred_svm_bi_24h,test_hm_24h$marry_bin)
```

```
##
## pred_svm_bi_24h    1    2    3    4    5
##               1    1   10    0   13    1
##               2   68  995   11  661    9
##               3    0    0    0    2    0
##               4  114  823   17 1758   13
##               5    0    2    0    2    0
```

```
mean(npred_svm_bi_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.6511111
```

```
table(npred_svm_bi_24h,test_hm_24h$marry_bin)
```

```
##
## npred_svm_bi_24h    1    2    3    4    5
##                2   52  872   11  378    8
##                4  131  958   17 2058   15
```

The model performance and prediction table of bigram + linear SVM are shown as the above.

Trigram model + multi-logistic (24h data)

Now, let's examine the trigram model with combination to multi-logistic and linear SVM

```
dtm_train <- token_ngram(tr_hm_24h,ngram=c(1L,3L))[[1]]
dim(dtm_train)
```

```
## [1]  10500 128339
```

```
ndtm_train = normalize(dtm_train)
dim(ndtm_train)
```

```
## [1]  10500 128339
```

```
# Train original data
lg_tri_24h = cv.glmnet(x = dtm_train, y = tr_hm_24h$marry_bin,
```

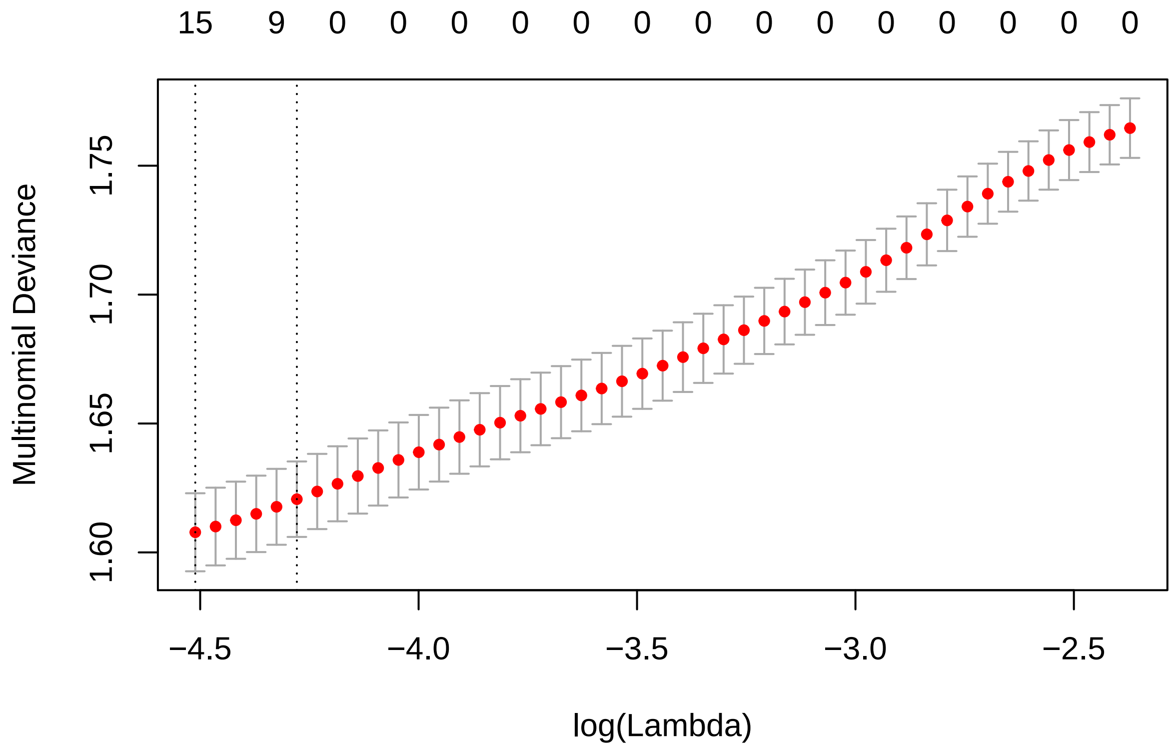
```
                     family = 'multinomial', alpha = 1,
                     nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

```
## Warning: from glmnet Fortran code (error code -50); Convergence for 50th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -96); Convergence for 96th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -50); Convergence for 50th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -49); Convergence for 49th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -51); Convergence for 51th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -48); Convergence for 48th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

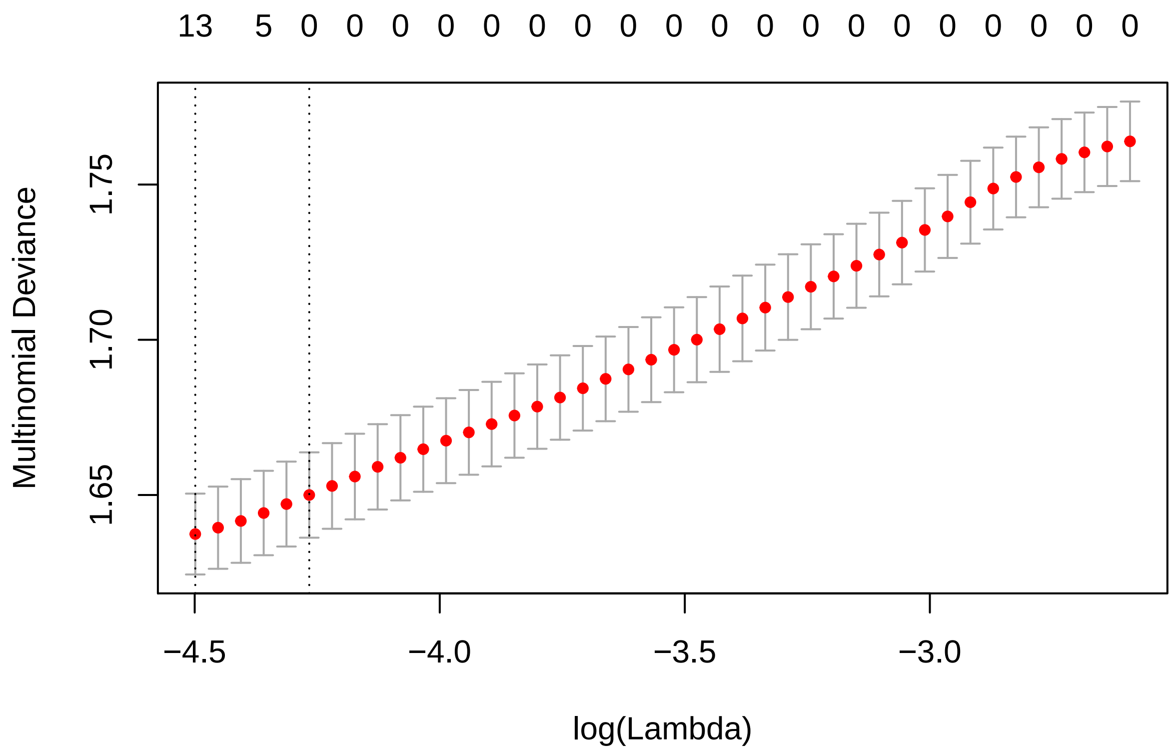
```
# Train normalized data
nlg_tri_24h = cv.glmnet(x = ndtm_train, y = tr_hm_24h$marry_bin,
                     family = 'multinomial', alpha = 1,
                     nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

```
## Warning: from glmnet Fortran code (error code -52); Convergence for 52th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -44); Convergence for 44th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -44); Convergence for 44th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -93); Convergence for 93th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```
plot(lg_tri_24h)
```




```
plot(nlg_tri_24h)
```




```
dtm_test = tokenTest_ngram(tes_hm_24h, tr_hm_24h,ngram=c(1L,3L))
ndtm_test = normalize(dtm_test)
pred_lg_tri_24h = as.numeric(predict(lg_tri_24h, dtm_test, type = 'class'))
npred_lg_tri_24h = as.numeric(predict(nlg_tri_24h, dtm_test, type = 'class'))
```

```r
mean(pred_lg_tri_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.638
```

```r
table(pred_lg_tri_24h,test_hm_24h$marry_bin)
```

```
##
## pred_lg_tri_24h    1    2    3    4    5
##               2   34  515    8   80    7
##               4  149 1315   20 2356   16
```

```r
mean(npred_lg_tri_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.6306667
```

```r
table(npred_lg_tri_24h,test_hm_24h$marry_bin)
```

```
##
## npred_lg_tri_24h    1    2    3    4    5
##                2   55  757    8  355    9
##                4  128 1073   20 2081   14
```

The model performance and prediction table of trigram + multilogistic are shown as the above.

Trigram model + Linear SVM (24h data)

```r
svm_tri_24h = LiblineaR(data = as.matrix(dtm_train), target = tr_hm_24h$marry_bin,type = 2)
pred_svm_tri_24h = predict(svm_tri_24h, as.matrix(dtm_test), type='class')
pred_svm_tri_24h = as.numeric(unlist(pred_svm_tri_24h))

nsvm_tri_24h = LiblineaR(data = as.matrix(ndtm_train), target = tr_hm_24h$marry_bin,type = 2)
npred_svm_tri_24h = predict(nsvm_tri_24h, as.matrix(ndtm_test), type='class')
npred_svm_tri_24h = as.numeric(unlist(npred_svm_tri_24h))
```

```r
#Accuracy for SVM
mean(pred_svm_tri_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.6175556
```

```r
table(pred_svm_tri_24h,test_hm_24h$marry_bin)
```

```
##
## pred_svm_tri_24h    1    2    3    4    5
##                1    1   12    0   10    1
##                2   60  962   11  607    8
##                3    0    0    0    2    0
##                4  122  855   17 1816   14
##                5    0    1    0    1    0
```

```r
mean(npred_svm_tri_24h==test_hm_24h$marry_bin)
```

```
## [1] 0.6482222
```

```r
table(npred_svm_tri_24h,test_hm_24h$marry_bin)
```

```
##
## npred_svm_tri_24h    1    2    3    4    5
```

```
##                 2   54  858   11  377    8
##                 4  129  972   17 2059   15
```

The model performance and prediction table of trigram + linear SVM are shown as the above.

```r
# Cleaned happy moments in training data
tr_hm_3m <- prepro(train_hm_3m)

# Cleaned happy moments in test data
tes_hm_3m <- prepro(test_hm_3m)
```

Bag-of-word model + multi-logistic (3m data)

Now let's see the happy moment of last 3 months data:

```r
dtm_train <- tokeniz(tr_hm_3m)[[1]]
ndtm_train = normalize(dtm_train)
```

```r
# Fitting the classifier using logistic regression
lg_bw_3m <- cv.glmnet(x = dtm_train, y = tr_hm_3m$marry_bin,
                     family = 'multinomial', alpha = 1,
                     nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

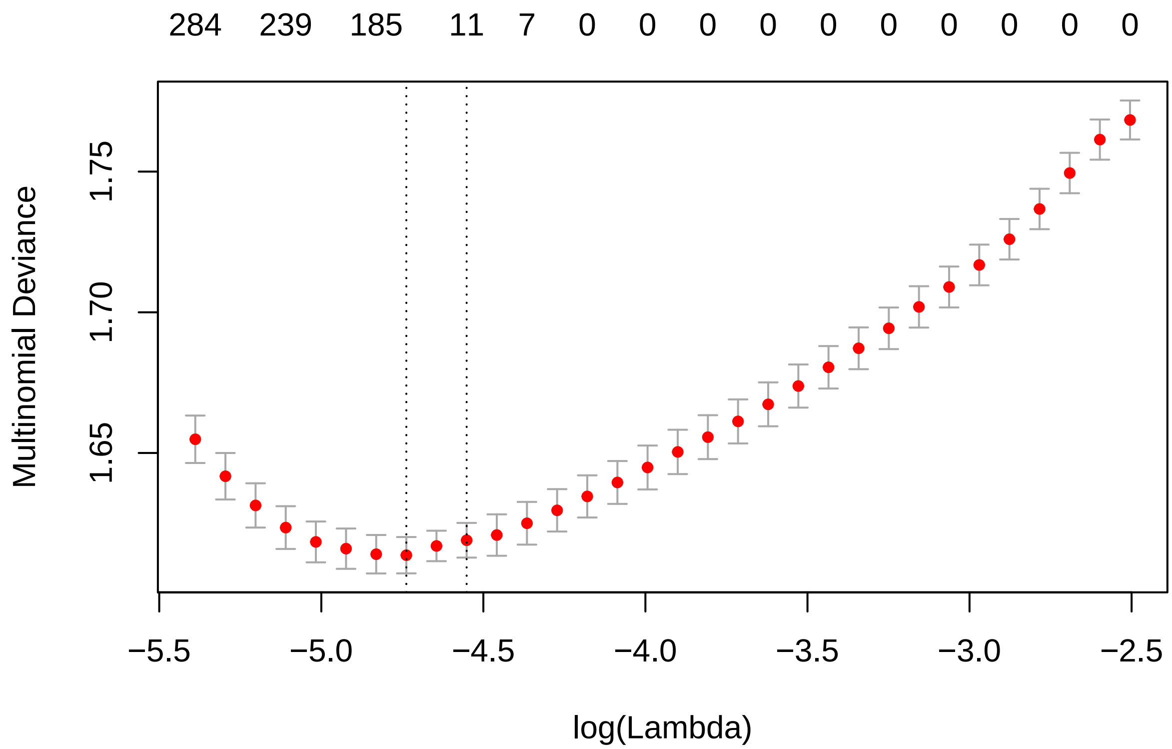
```
## Warning: from glmnet Fortran code (error code -33); Convergence for 33th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -68); Convergence for 68th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -90); Convergence for 90th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -97); Convergence for 97th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
nlg_bw_3m <- cv.glmnet(x = ndtm_train, y = tr_hm_3m$marry_bin,
                     family = 'multinomial', alpha = 1,
                     nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

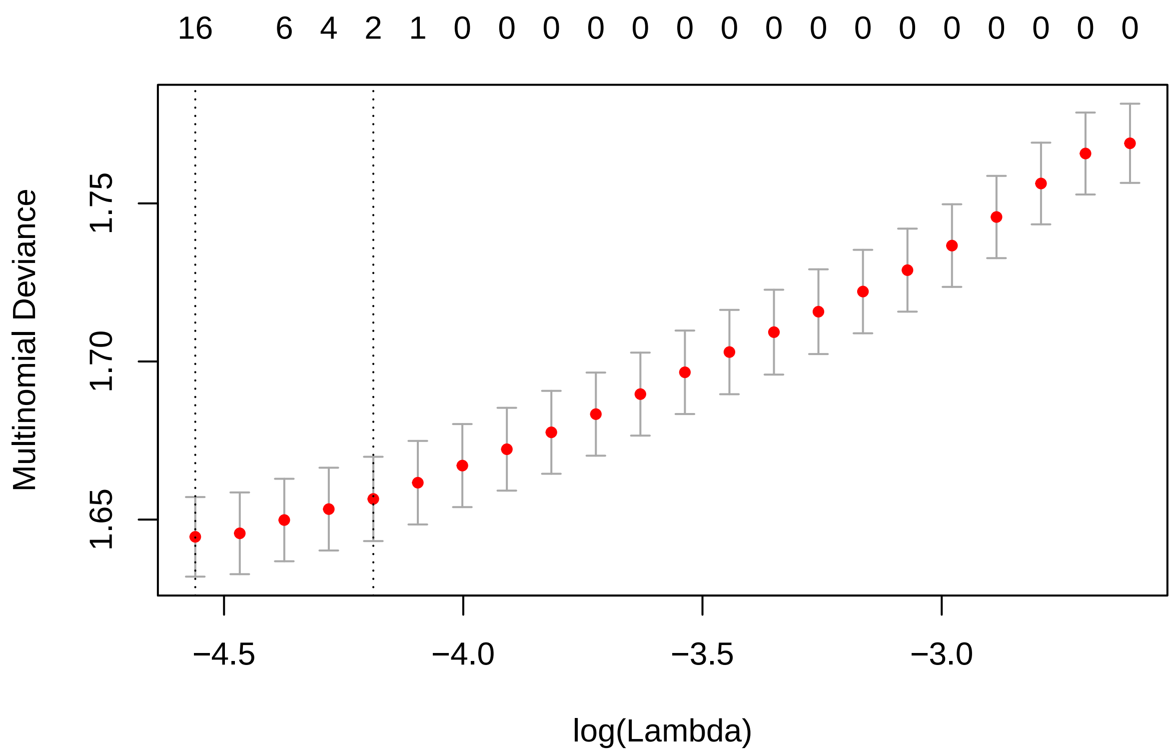
```
## Warning: from glmnet Fortran code (error code -26); Convergence for 26th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -44); Convergence for 44th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -47); Convergence for 47th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -53); Convergence for 53th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```
## Warning: from glmnet Fortran code (error code -89); Convergence for 89th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```
plot(lg_bw_3m)
```




```
plot(nlg_bw_3m)
```



```r
dtm_test <- tokenTest(tes_hm_3m, tr_hm_3m)
ndtm_test = normalize(dtm_test)
pred_lg_bw_3m = as.numeric(predict(lg_bw_3m, dtm_test, type = 'class'))
npred_lg_bw_3m = as.numeric(predict(nlg_bw_3m, ndtm_test, type = 'class'))
```

```r
mean(pred_lg_bw_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.6457778
```

```r
table(pred_lg_bw_3m,tes_hm_3m$marry_bin)
```

```
##
## pred_lg_bw_3m    1    2    3    4    5
##             2   26  524   10   84    8
##             3    0    1    0    1    0
##             4  147 1264   30 2382   23
```

```r
mean(npred_lg_bw_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.6366667
```

```r
table(npred_lg_bw_3m,tes_hm_3m$marry_bin)
```

```
##
## npred_lg_bw_3m    1    2    3    4    5
##              2   23  445    9   47    6
##              4  150 1344   31 2420   25
```

The model performance and prediction table of bag-of-word + multilogistic are shown as the above.

Bag-of-word model + Linear SVM (3m data)

```r
svm_bw_3m = LiblineaR(data = as.matrix(dtm_train), target = tr_hm_3m$marry_bin,type = 2)
pred_svm_bw_3m = predict(svm_bw_3m, as.matrix(dtm_test), type='class')
pred_svm_bw_3m = as.numeric(unlist(pred_svm_bw_3m))

nsvm_bw_3m = LiblineaR(data = as.matrix(ndtm_train), target = tr_hm_3m$marry_bin,type = 2)
npred_svm_bw_3m = predict(nsvm_bw_3m, as.matrix(ndtm_test), type='class')
npred_svm_bw_3m = as.numeric(unlist(npred_svm_bw_3m))
```

```r
# Accuracy for SVM
mean(pred_svm_bw_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.6048889
```

```r
table(pred_svm_bw_3m,tes_hm_3m$marry_bin)
```

```
##
## pred_svm_bw_3m    1    2    3    4    5
##              1    5   30    0   33    2
##              2   65  931   14  647   14
##              3    0    1    1    2    0
##              4  103  826   25 1784   14
##              5    0    1    0    1    1
```

```r
mean(npred_svm_bw_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.6451111
```

```r
table(npred_svm_bw_3m,tes_hm_3m$marry_bin)
```

```
##
## npred_svm_bw_3m    1    2    3    4    5
##               2   47  784   13  348   12
##               4  126 1005   27 2119   19
```

The model performance and prediction table of bag-of-word + linear SVM are shown as the above.

Bigram model + multi-logistic (3m data)

```r
dtm_train <- token_ngram(tr_hm_3m)[[1]]
ndtm_train = normalize(dtm_train)
```

```r
lg_bi_3m = cv.glmnet(x = dtm_train, y = tr_hm_3m$marry_bin,
                     family = 'multinomial', alpha = 1,
                     nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

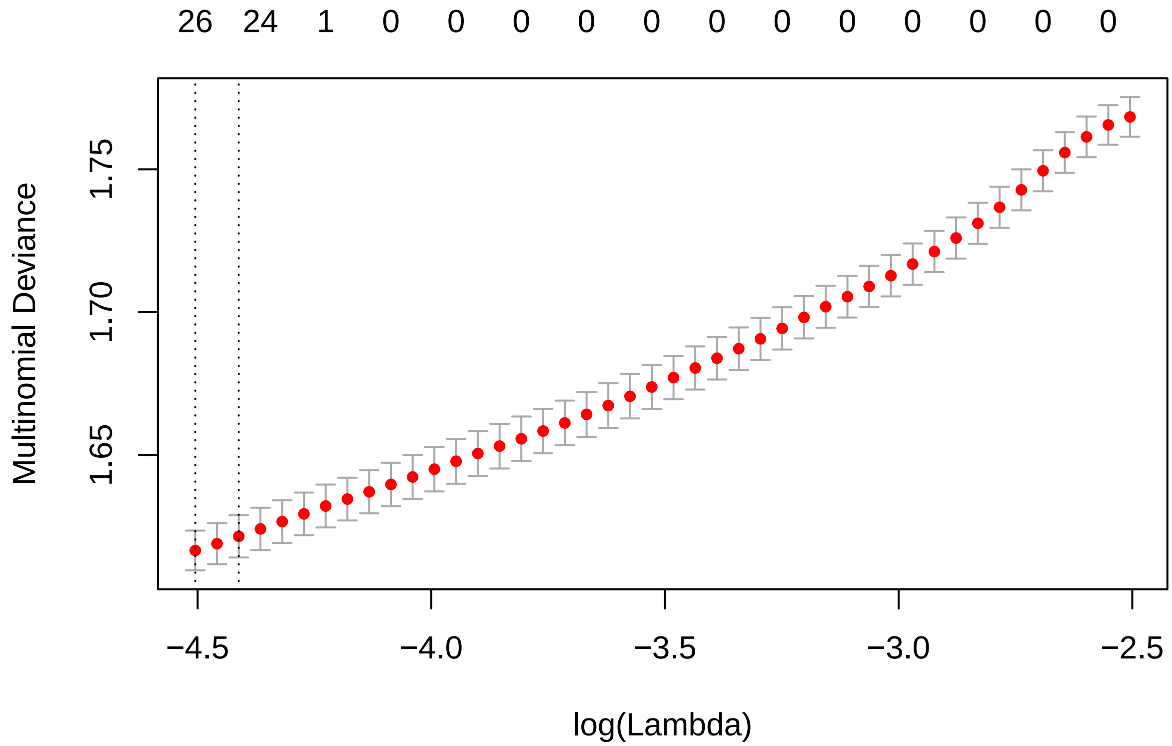
```
## Warning: from glmnet Fortran code (error code -93); Convergence for 93th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -45); Convergence for 45th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -46); Convergence for 46th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -88); Convergence for 88th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -47); Convergence for 47th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -83); Convergence for 83th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
nlg_bi_3m = cv.glmnet(x = ndtm_train, y = tr_hm_3m$marry_bin,
                      family = 'multinomial', alpha = 1,
                      nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

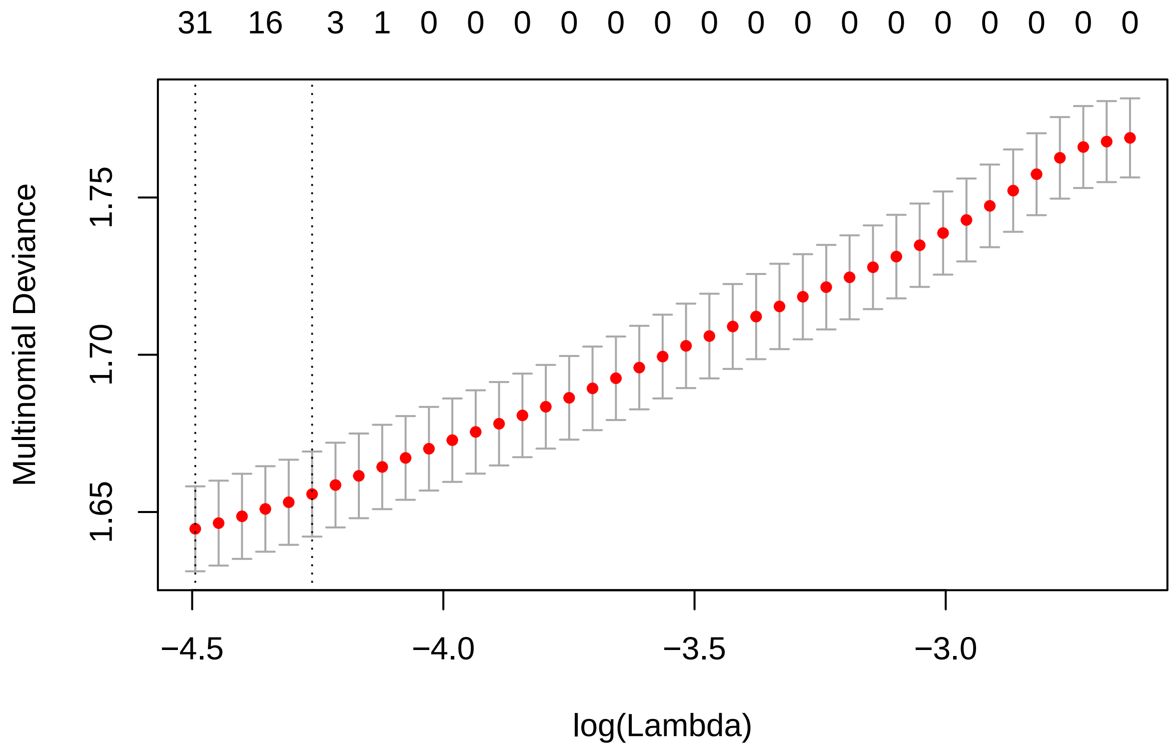
```
## Warning: from glmnet Fortran code (error code -45); Convergence for 45th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -43); Convergence for 43th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```
## Warning: from glmnet Fortran code (error code -42); Convergence for 42th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -41); Convergence for 41th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
plot(lg_bi_3m)
```




```r
plot(nlg_bi_3m)
```




```r
# Evaluating the performance of our classifier on test data
dtm_test = tokenTest_ngram(tes_hm_3m, tr_hm_3m)
ndtm_test = normalize(dtm_test)
pred_lg_bi_3m = as.numeric(predict(lg_bi_3m, dtm_test, type = 'class'))
npred_lg_bi_3m = as.numeric(predict(nlg_bi_3m, dtm_test, type = 'class'))
```

```r
mean(pred_lg_bi_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.646
```

```r
table(pred_lg_bi_3m,tes_hm_3m$marry_bin)
```

```
##
## pred_lg_bi_3m    1    2    3    4    5
##             2   24  507   10   67    8
##             4  149 1282   30 2400   23
```

```r
mean(npred_lg_bi_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.648
```

```r
table(npred_lg_bi_3m,tes_hm_3m$marry_bin)
```

```
##
## npred_lg_bi_3m    1    2    3    4    5
##              1    2    3    0    1    1
##              2   36  660   13  212   14
##              4  135 1126   27 2254   16
```

The model performance and prediction table of bigram + multilogistic are shown as the above.

Bigram model + Linear SVM (3m data)

```
svm_bi_3m = LiblineaR(data = as.matrix(dtm_train), target = train_hm_3m$marry_bin,type = 2)
pred_svm_bi_3m = predict(svm_bi_3m, as.matrix(dtm_test), type='class')
pred_svm_bi_3m = as.numeric(unlist(pred_svm_bi_3m))

nsvm_bi_3m = LiblineaR(data = as.matrix(ndtm_train), target = train_hm_3m$marry_bin,type = 2)
npred_svm_bi_3m = predict(nsvm_bi_3m, as.matrix(ndtm_test), type='class')
npred_svm_bi_3m = as.numeric(unlist(npred_svm_bi_3m))
```

```
#Accuracy for SVM
mean(pred_svm_bi_3m==test_hm_3m$marry_bin)
```

```
## [1] 0.6164444
```

```
table(pred_svm_bi_3m,test_hm_3m$marry_bin)
```

```
##
## pred_svm_bi_3m    1    2    3    4    5
##              1    1   15    1   23    1
##              2   67  964   12  634   12
##              3    0    2    2    3    0
##              4  105  808   25 1807   18
```

```
mean(npred_svm_bi_3m==test_hm_3m$marry_bin)
```

```
## [1] 0.6451111
```

```
table(npred_svm_bi_3m,test_hm_3m$marry_bin)
```

```
##
## npred_svm_bi_3m    1    2    3    4    5
##               2   40  723   13  287    9
##               4  133 1066   27 2180   22
```

The model performance and prediction table of bigram + linear SVM are shown as the above.

Trigram model + multi-logistic (3m data)

Now, let's examine the trigram model with combination to multi-logistic and linear SVM

```
dtm_train <- token_ngram(tr_hm_3m,ngram=c(1L,3L))[[1]]
ndtm_train = normalize(dtm_train)
```

```
lg_tri_3m = cv.glmnet(x = dtm_train, y = tr_hm_3m$marry_bin,
                      family = 'multinomial', alpha = 1,
                      nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

```
## Warning: from glmnet Fortran code (error code -93); Convergence for 93th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```
## Warning: from glmnet Fortran code (error code -45); Convergence for 45th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

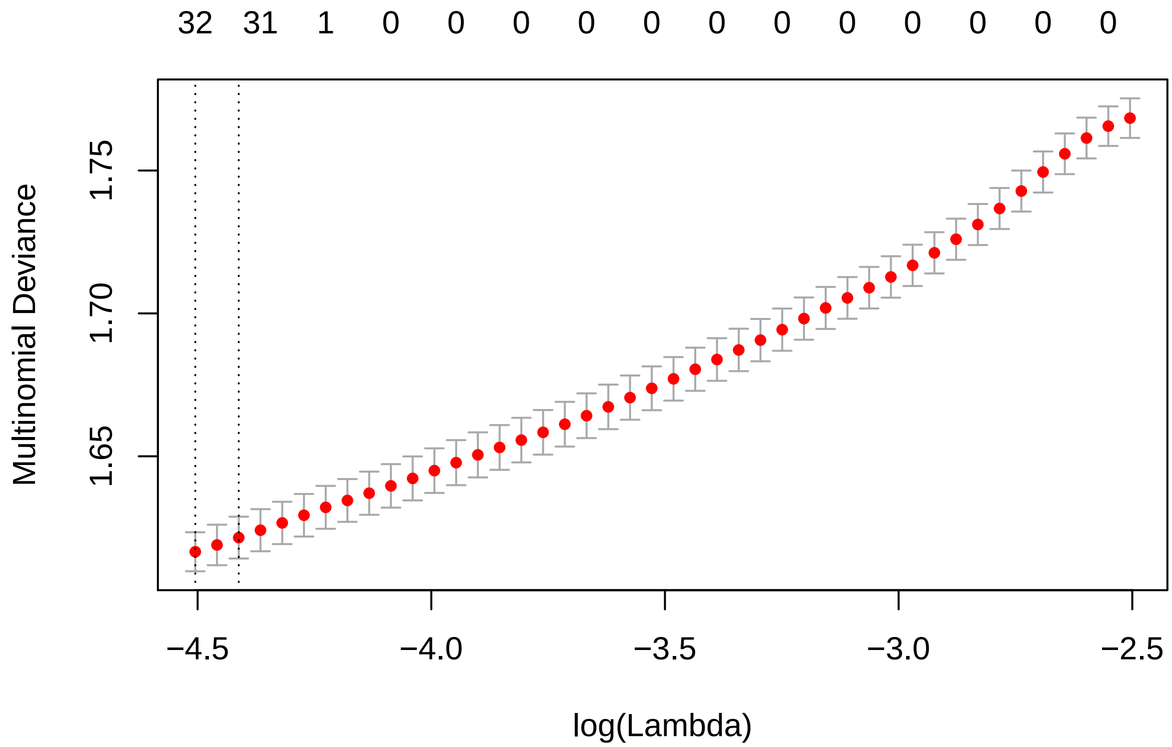
```
## Warning: from glmnet Fortran code (error code -46); Convergence for 46th
```

```
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -91); Convergence for 91th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -47); Convergence for 47th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -86); Convergence for 86th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

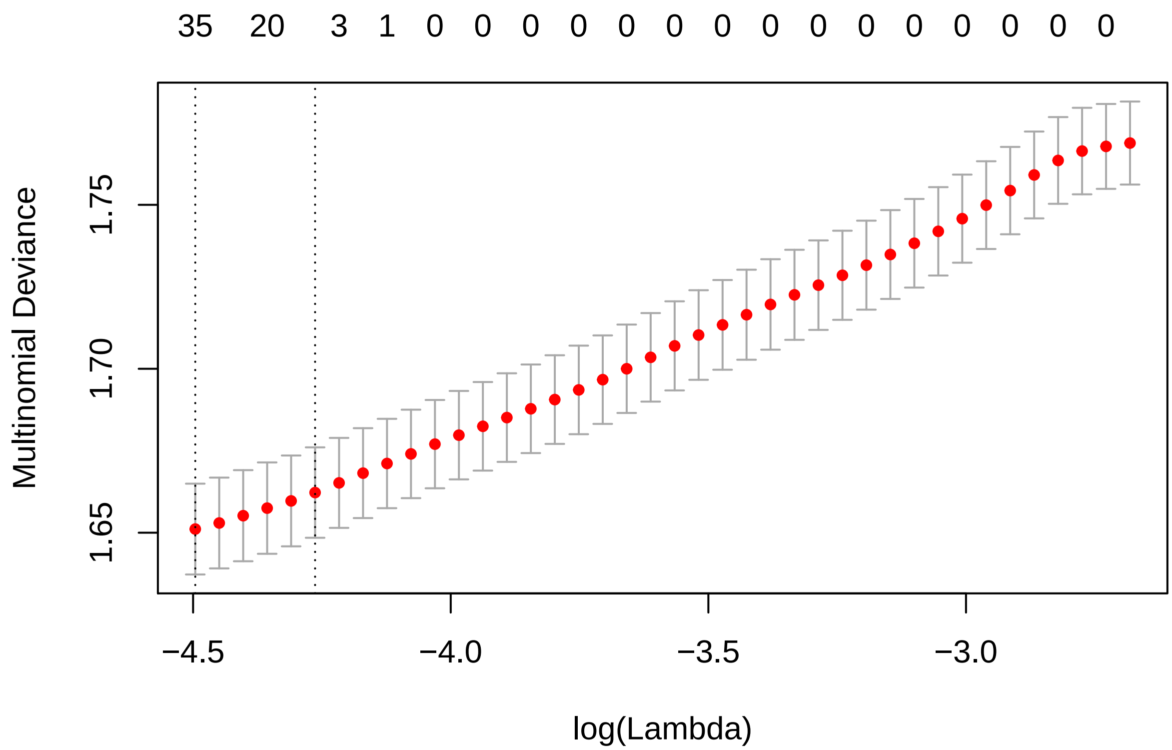
```r
nlg_tri_3m = cv.glmnet(x = ndtm_train, y = tr_hm_3m$marry_bin,
                     family = 'multinomial', alpha = 1,
                     nfolds = 5, thresh = 1e-3, maxit = 1e3)
```

```
## Warning: from glmnet Fortran code (error code -44); Convergence for 44th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -42); Convergence for 42th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -41); Convergence for 41th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -76); Convergence for 76th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -47); Convergence for 47th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned

## Warning: from glmnet Fortran code (error code -42); Convergence for 42th
## lambda value not reached after maxit=1000 iterations; solutions for larger
## lambdas returned
```

```r
plot(lg_tri_3m)
```




```
plot(nlg_tri_3m)
```




```
dtm_test = tokenTest_ngram(tes_hm_3m, tr_hm_3m,ngram=c(1L,3L))
ndtm_test = normalize(dtm_test)
pred_lg_tri_3m = as.numeric(predict(lg_tri_3m, dtm_test, type = 'class'))
npred_lg_tri_3m = as.numeric(predict(nlg_tri_3m, dtm_test, type = 'class'))
```

```r
mean(pred_lg_tri_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.646
```

```r
table(pred_lg_tri_3m,tes_hm_3m$marry_bin)
```

```
##
## pred_lg_tri_3m    1    2    3    4    5
##              2   24  507   10   67    8
##              4  149 1282   30 2400   23
```

```r
mean(npred_lg_tri_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.65
```

```r
table(npred_lg_tri_3m,tes_hm_3m$marry_bin)
```

```
##
## npred_lg_tri_3m    1    2    3    4    5
##               1    2    3    0    1    1
##               2   35  657   12  200   13
##               4  136 1129   28 2266   17
```

The model performance and prediction table of trigram + multilogistic are shown as the above.

Trigram model + Linear SVM (3m data)

```r
svm_tri_3m = LiblineaR(data = as.matrix(dtm_train), target = tr_hm_3m$marry_bin,type = 2)
pred_svm_tri_3m = predict(svm_tri_3m, as.matrix(dtm_test), type='class')
pred_svm_tri_3m = as.numeric(unlist(pred_svm_tri_3m))

nsvm_tri_3m = LiblineaR(data = as.matrix(ndtm_train), target = tr_hm_3m$marry_bin,type = 2)
npred_svm_tri_3m = predict(nsvm_tri_3m, as.matrix(ndtm_test), type='class')
npred_svm_tri_3m = as.numeric(unlist(npred_svm_tri_3m))
```

```r
#Accuracy for SVM
mean(pred_svm_tri_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.6284444
```

```r
table(pred_svm_tri_3m,tes_hm_3m$marry_bin)
```

```
##
## pred_svm_tri_3m    1    2    3    4    5
##               1    1    6    1   12    1
##               2   63  931   12  559   12
##               3    0    2    2    2    0
##               4  109  850   25 1894   18
```

```r
mean(npred_svm_tri_3m==tes_hm_3m$marry_bin)
```

```
## [1] 0.6482222
```

```r
table(npred_svm_tri_3m,tes_hm_3m$marry_bin)
```

```
##
## npred_svm_tri_3m    1    2    3    4    5
```

```
##                 2   38  711   12  261   10
##                 4  135 1078   28 2206   21
```

The model performance and prediction table of trigram + linear SVM are shown as the above.

Summary of classification for marital status

For happy moment of last 24hr data:

```r
data.frame(model=c("Bag-of-word logistic","Bag-of-word linear svm",
                   "bigram logistic","bigram linear svm",
                   "trigram logistic", "trigram linear svm"),Accuracy=
             c(mean(pred_lg_bw_24h==tes_hm_24h$marry_bin),
               mean(pred_svm_bw_24h==tes_hm_24h$marry_bin),
               mean(pred_lg_bi_24h==tes_hm_24h$marry_bin),
               mean(pred_svm_bi_24h==tes_hm_24h$marry_bin),
               mean(pred_lg_tri_24h==tes_hm_24h$marry_bin),
               mean(pred_svm_tri_24h==tes_hm_24h$marry_bin)),
           Normalized_Accuracy = c(mean(npred_lg_bw_24h==tes_hm_24h$marry_bin),
               mean(npred_svm_bw_24h==tes_hm_24h$marry_bin),
               mean(npred_lg_bi_24h==tes_hm_24h$marry_bin),
               mean(npred_svm_bi_24h==tes_hm_24h$marry_bin),
               mean(npred_lg_tri_24h==tes_hm_24h$marry_bin),
               mean(npred_svm_tri_24h==tes_hm_24h$marry_bin)))
```

```
##                    model  Accuracy Normalized_Accuracy
## 1   Bag-of-word logistic 0.6371111           0.6366667
## 2 Bag-of-word linear svm 0.6057778           0.6448889
## 3        bigram logistic 0.6380000           0.6380000
## 4      bigram linear svm 0.6120000           0.6511111
## 5       trigram logistic 0.6380000           0.6306667
## 6     trigram linear svm 0.6175556           0.6482222
```

For happy moment of last 3 months data:

```r
data.frame(model=c("Bag-of-word logistic","Bag-of-word linear svm",
                   "bigram logistic","bigram linear svm",
                   "trigram logistic", "trigram linear svm"),Accuracy=
             c(mean(pred_lg_bw_3m==tes_hm_3m$marry_bin),
               mean(pred_svm_bw_3m==tes_hm_3m$marry_bin),
               mean(pred_lg_bi_3m==tes_hm_3m$marry_bin),
               mean(pred_svm_bi_3m==tes_hm_3m$marry_bin),
               mean(pred_lg_tri_3m==tes_hm_3m$marry_bin),
               mean(pred_svm_tri_3m==tes_hm_3m$marry_bin)),
           Normalized_Accuracy = c(mean(npred_lg_bw_24h==tes_hm_24h$marry_bin),
               mean(npred_svm_bw_24h==tes_hm_24h$marry_bin),
               mean(npred_lg_bi_24h==tes_hm_24h$marry_bin),
               mean(npred_svm_bi_24h==tes_hm_24h$marry_bin),
               mean(npred_lg_tri_24h==tes_hm_24h$marry_bin),
               mean(npred_svm_tri_24h==tes_hm_24h$marry_bin)))
```

```
##                    model  Accuracy Normalized_Accuracy
## 1   Bag-of-word logistic 0.6457778           0.6366667
## 2 Bag-of-word linear svm 0.6048889           0.6448889
## 3        bigram logistic 0.6460000           0.6380000
```

```
## 4      bigram linear svm 0.6164444          0.6511111
## 5       trigram logistic 0.6460000          0.6306667
## 6     trigram linear svm 0.6284444          0.6482222
```

The accuracy of all tested models are shown as the above tables, where ngram models (bigram or trigram) perform better than bag-of-word models in general. For machine learning models, multilogistic models perform better than linear SVMs if the DTM matrix data is not normalized, however, as data is normalized, we can see linear SVM outperforms multilogistic model, and there's an obvious improvement of linear SVM performance after normalization. On the other hand, normalization doesn't seem to be able to improve model performance of multilogistic models.

Hierarchical clustering for happy moment text data

In this part, I would like to perform basic document clustering, where a commonly used technique for information retrieval TFIDF is applied. Specifically, I will focuse on group of married people and singled, and divide data with respect to happy moment of last 24 hours and 3 months.

```
#Data preparation
new = marital_bin_data[,1:3]
set.seed(0)
n =3000

single_24h = marital_bin_data[which(marital_bin_data$reflection_period=='24h'&marital_bin_data$marital=
single_24h = single_24h[sample(1:nrow(single_24h),n),]
rownames(single_24h) = seq(1:nrow(single_24h))

married_24h = marital_bin_data[which(marital_bin_data$reflection_period=='24h'&marital_bin_data$marital
married_24h = married_24h[sample(1:nrow(married_24h),n),]
rownames(married_24h) = seq(1:nrow(married_24h))

single_3m = marital_bin_data[which(marital_bin_data$reflection_period=='3m'&marital_bin_data$marital=="
single_3m = single_3m[sample(1:nrow(single_3m),n),]
rownames(single_3m) = seq(1:nrow(single_3m))

married_3m = marital_bin_data[which(marital_bin_data$reflection_period=='3m'&marital_bin_data$marital==
married_3m = married_3m[sample(1:nrow(married_3m),n),]
rownames(married_3m) = seq(1:nrow(married_3m))
```

Data is subset as: "moment for last 24 hours of single", "moment for last 3 months of single", "moment for last 24 hours of married" and "moment for last 3 months of married". For the sake of computational efficiency, 3,000 samples from each subset are drawn with random seed zero.

Preprocessing data:

```
married_24h = prepro(married_24h)
single_24h = prepro(single_24h)

married_3m = prepro(married_3m)
single_3m = prepro(single_3m)
```

In order to obtain IDFs, need to create DTM matrices and their matrix transformed terms. The IDF is a measure of how much information the word provides whether common or rare across all documents:

```
dtm_married_24h = as.matrix(tokeniz(married_24h)[[1]])
dtm_single_24h = as.matrix(tokeniz(single_24h)[[1]])
```

```r
dtm_married_3m = as.matrix(tokeniz(married_3m)[[1]])
dtm_single_3m = as.matrix(tokeniz(single_3m)[[1]])


mat_married_24h = TermDocFreq(dtm_married_24h)
mat_single_24h = TermDocFreq(dtm_single_24h)


mat_married_3m = TermDocFreq(dtm_married_3m)
mat_single_3m = TermDocFreq(dtm_single_3m)


head(mat_married_24h)
```

```
##                  term term_freq doc_freq      idf
## citizens   citizens         1        1 8.006368
## scaring     scaring         1        1 8.006368
## packet       packet         1        1 8.006368
## tater         tater         1        1 8.006368
## hacking     hacking         1        1 8.006368
## crashersa crashersa         1        1 8.006368
```

```r
head(mat_married_3m)
```

```
##            term term_freq doc_freq      idf
## folding folding         1        1 8.006368
## rules     rules         1        1 8.006368
## scaring scaring         1        1 8.006368
## packet   packet         1        1 8.006368
## tater     tater         1        1 8.006368
## cowgirl cowgirl         1        1 8.006368
```

```r
head(mat_single_24h)
```

```
##                    term term_freq doc_freq      idf
## canadiens     canadiens         1        1 8.006368
## shakespeare shakespeare         1        1 8.006368
## recharge       recharge         1        1 8.006368
## partyafter   partyafter         1        1 8.006368
## champagne     champagne         1        1 8.006368
## turns             turns         1        1 8.006368
```

```r
head(mat_single_3m)
```

```
##                      term term_freq doc_freq      idf
## hangout           hangout         1        1 8.006368
## rules               rules         1        1 8.006368
## monarch           monarch         1        1 8.006368
## bubbly             bubbly         1        1 8.006368
## champagne       champagne         1        1 8.006368
## longexposure longexposure         1        1 8.006368
```

For this step, calculate the cosine similarities for text and then converting similarities back to distances in order to make it work for clustering in R:

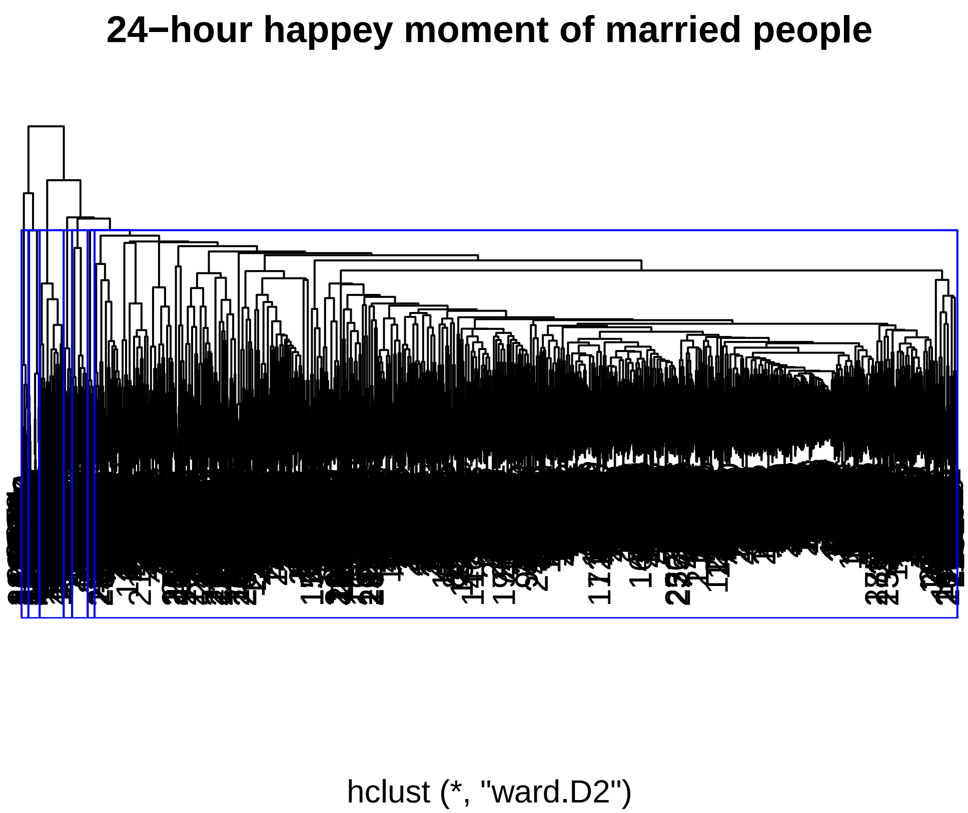
```r
# Transpose the matrix for calculation:
t_mar_24h = t(t(dtm_married_24h[,mat_married_24h$term])*mat_married_24h$idf)
t_sin_24h = t(t(dtm_single_24h[,mat_single_24h$term])*mat_single_24h$idf)

t_mar_3m = t(t(dtm_married_3m[,mat_married_3m$term])*mat_married_3m$idf)
t_sin_3m = t(t(dtm_single_3m[,mat_single_3m$term])*mat_single_3m$idf)



# Calculate cosine similarity and convert to distances:
mar_24h = t_mar_24h/sqrt(rowSums(t_mar_24h*t_mar_24h))
mar_24h = as.dist(1-mar_24h%*%t(mar_24h))
sin_24h = t_sin_24h/sqrt(rowSums(t_sin_24h*t_sin_24h))
sin_24h = as.dist(1-sin_24h%*%t(sin_24h))

mar_3m = t_mar_3m/sqrt(rowSums(t_mar_3m*t_mar_3m))
mar_3m = as.dist(1-mar_3m%*%t(mar_3m))
sin_3m = t_sin_3m/sqrt(rowSums(t_sin_3m*t_sin_3m))
sin_3m = as.dist(1-sin_3m%*%t(sin_3m))
```

```r
# Helper functions:
# Hierarchical Clustering, the default distance matrix is Ward's distance
HC = function(distance,method = "ward.D",num_group=7){
  hc = hclust(distance,method)
  clustering = cutree(hc,k=num_group)
  return(list(clustering,hc))
}

# Draw Clusters
DrawHC = function(distance, method = "ward.D", num_group=7, word = "Hierarchical clustering"){
  hc = HC(distance,method)[[2]]
  clustering = HC(distance, method)[[1]]

  plot(hc, main = word,
     ylab = "", xlab = "", yaxt = "n")

  rect.hclust(hc, num_group, border = "blue")
}

# Calculate probability of word
prob = function(data){
  colSums(data) / sum(data)
}

# Process cluster output
cluster_process <- function(data,distance,method="ward.D2"){
 clustering = HC(distance,method)[[1]]
 p_words <- prob(data)
 out = lapply(unique(clustering), function(x){
  rows <- data[clustering == x , ]

  # Drop words that don't appear in the cluster
  rows <- rows[ ,colSums(rows)>0]
```

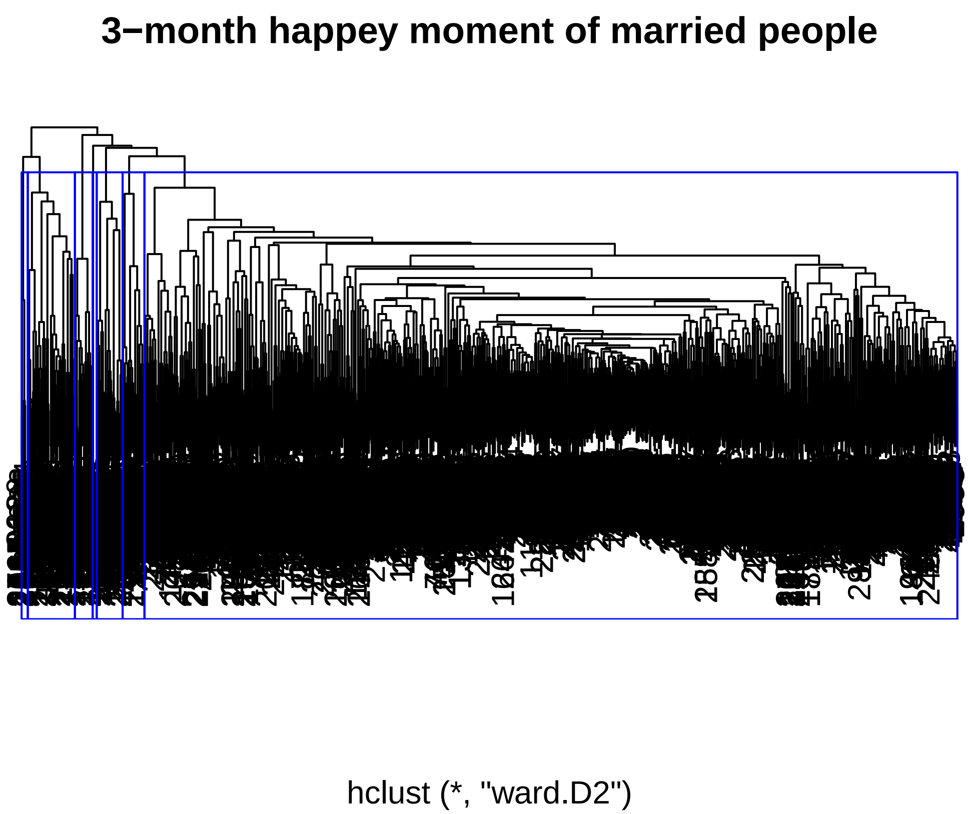
```r
  colSums(rows) / sum(rows) - p_words[colnames(rows)]
})
 return(out)
}

# Create a summary table of the top N words defining each cluster
cluster_summary = function(clustering, cluster_words, N){
      data.frame(cluster = unique(clustering),
                          size = as.numeric(table(clustering)),
                          top_words = sapply(cluster_words, function(d){
                                 paste(
                                 names(d)[ order(d, decreasing = TRUE) ][1:N],
                                 collapse = ", ")
                               }),
                           stringsAsFactors = FALSE)
}
```

For hierarchical clustering, I arbitrarily set K equal to 7, distance measure be Ward's distance and the hierarchical clustering trees can be shown as follows:

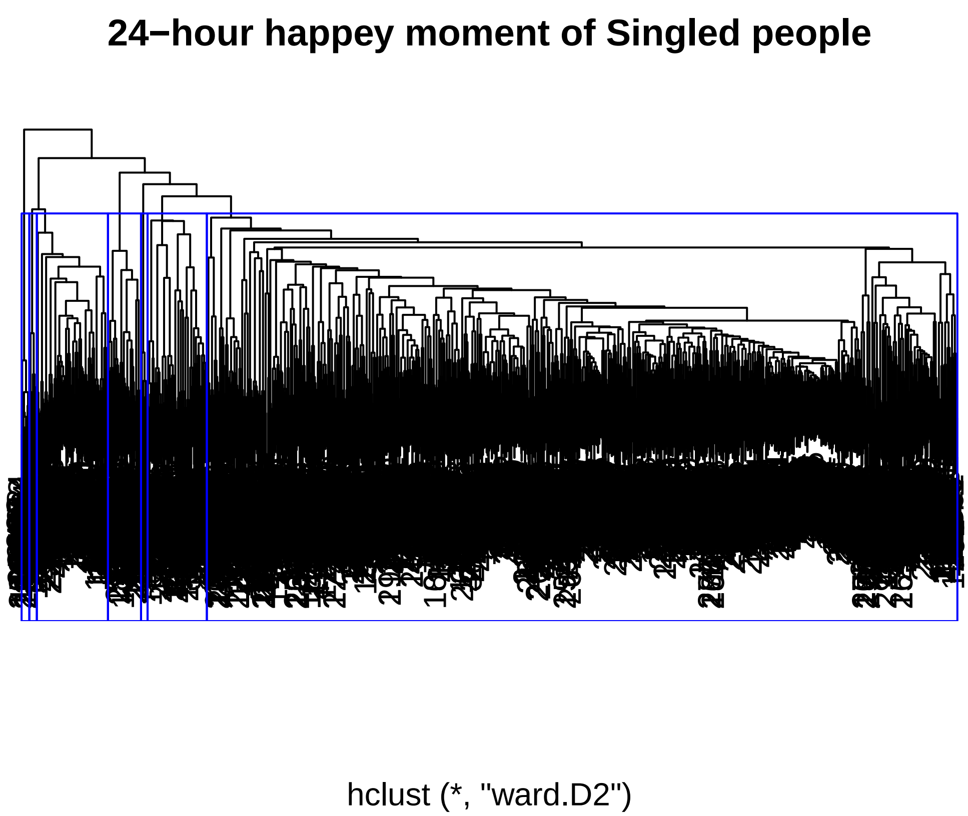
```r
DrawHC(mar_24h,method = "ward.D2",word="24-hour happey moment of married people")
```



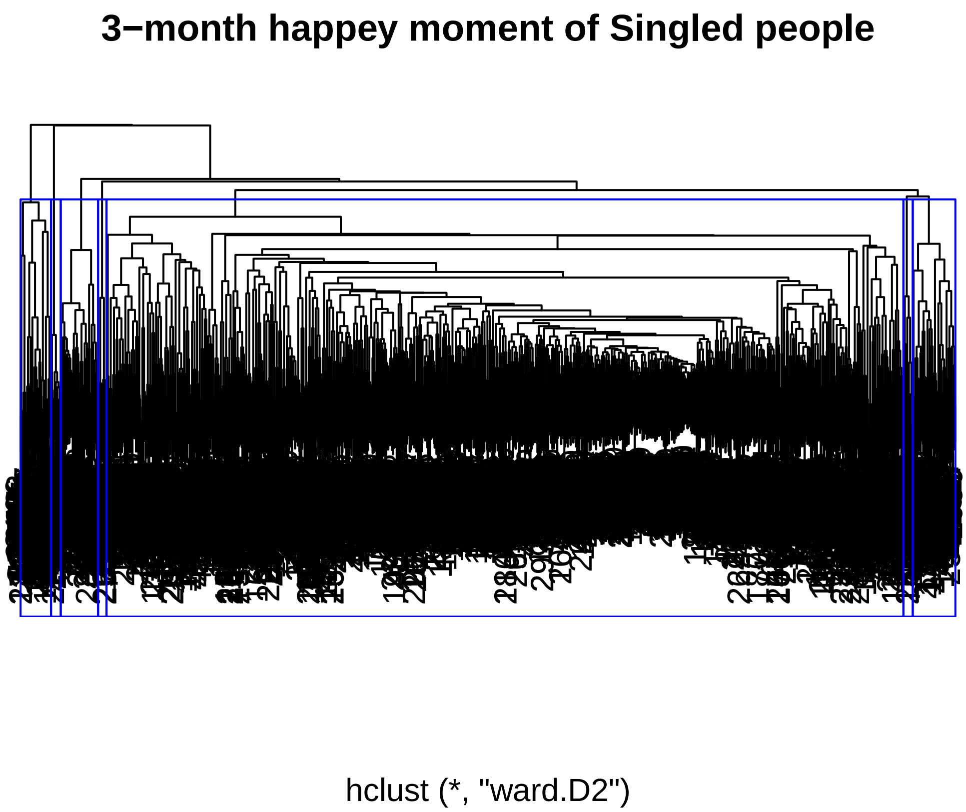

```r
DrawHC(mar_3m,method = "ward.D2",word="3-month happey moment of married people")
```



```r
DrawHC(sin_24h,method = "ward.D2",word="24-hour happey moment of Singled people")
```



```r
DrawHC(sin_3m,method = "ward.D2",word="3-month happey moment of Singled people")
```



Here, the cluster summary shows the top 5 words in the cluster and the size of that cluster:

```r
clng_mar_24h = HC(mar_24h)[[1]]
clwo_mar_24h = cluster_process(dtm_married_24h,mar_24h)
m_24h = cluster_summary(clng_mar_24h,clwo_mar_24h,5)
m_24h
```

```
##   cluster size                          top_words
## 1       1 2820       new, one, moment, found, gave
## 2       2   35   dinner, night, last, wife, cooked
## 3       3   16 went, temple, shopping, mall, saint
## 4       4   50        sleep, time, spend, able, got
## 5       5   48 lunch, friend, yesterday, ate, took
## 6       6   17     movie, went, watched, wife, good
## 7       7   14       walk, went, weather, dog, take
```

```r
clng_mar_3m = HC(mar_3m)[[1]]
clwo_mar_3m = cluster_process(dtm_married_3m,mar_3m)
m_3m = cluster_summary(clng_mar_3m,clwo_mar_3m,5)
m_3m
```

```
##   cluster size                                             top_words
## 1       1 2853                    went, vacation, movie, wife, dinner
## 2       2   13                           day, able, life, months, one
## 3       3   58 offsite, discussions, stimulating, colleagues, fun
## 4       4   19                           got, work, job, bonus, raise
## 5       5   21                     shopping, went, wend, mall, tempel
```

```
## 6       6   28     birthday, party, marriage, celebrated, friends
## 7       7    8                new, car, bought, purchased, laptop
```

```r
clng_sin_24h = HC(sin_24h)[[1]]
clwo_sin_24h = cluster_process(dtm_single_24h,sin_24h)
s_24h = cluster_summary(clng_sin_24h,clwo_sin_24h,5)
s_24h
```

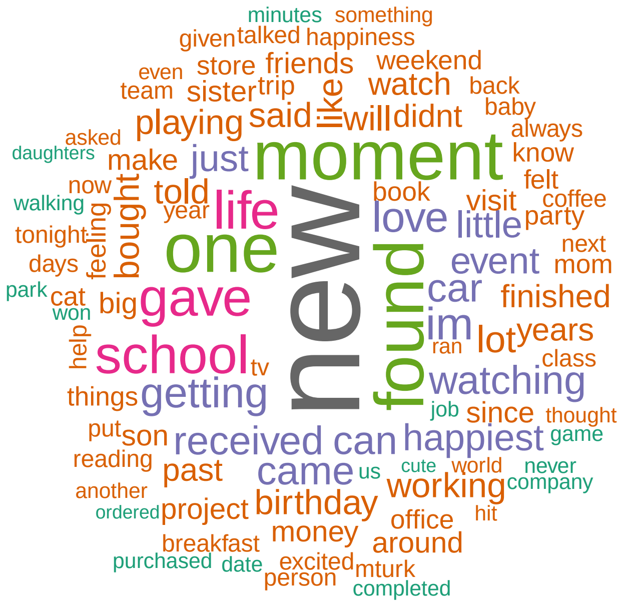
```
##   cluster size                             top_words
## 1       1 2433         new, time, found, moment, dog
## 2       2  219 ate, favorite, dinner, delicious, eat
## 3       3  174         work, got, money, hours, sleep
## 4       4   31 talked, friend, called, phone, havent
## 5       5   99   game, video, won, movement, playing
## 6       6   19   dinner, family, went, enjoyed, nice
## 7       7   25  movie, watched, theater, good, watch
```

```r
clng_sin_3m = HC(sin_3m)[[1]]
clwo_sin_3m = cluster_process(dtm_single_3m,sin_3m)
s_3m = cluster_summary(clng_sin_3m,clwo_sin_3m,5)
s_3m
```

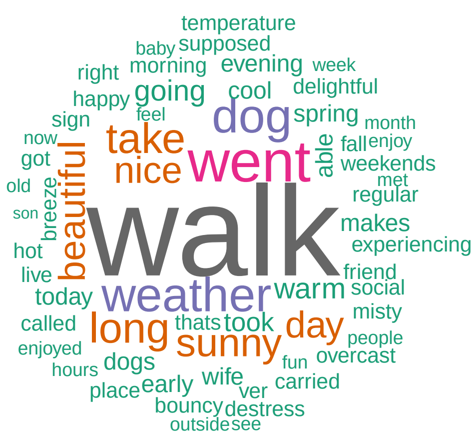
```
##   cluster size                                 top_words
## 1       1 2606                day, get, found, made, life
## 2       2   67             game, won, video, played, team
## 3       3  143       dinner, restaurant, ate, lunch, eat
## 4       4   39                   job, got, new, work, raise
## 5       5   95        seen, havent, hadnt, friend, years
## 6       6   25             movie, went, watch, see, movies
## 7       7   25 party, birthday, friends, surprise, threw
```

Now, I will visualize the top words of the cluster of largest size and smallest size with respect to four different group, and start with married people's last 24 hour happy moment:

```r
max.m_24h = which(m_24h$size==max(m_24h$size))[1]
min.m_24h = which(m_24h$size==min(m_24h$size))[1]
wordcloud::wordcloud(words = names(clwo_mar_24h[[ max.m_24h ]]),
                     freq = clwo_mar_24h[[ max.m_24h ]],
                     max.words = 100,
                     random.order = FALSE,
                     scale=c(4,.2),
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```



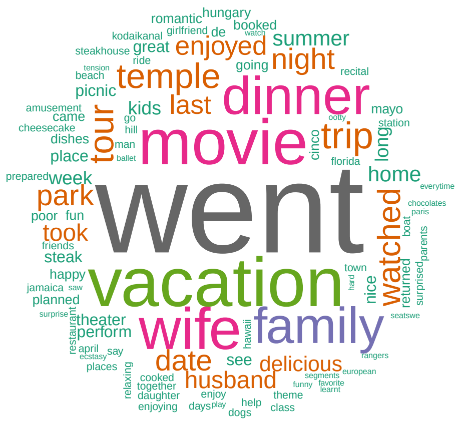

```r
wordcloud::wordcloud(words = names(clwo_mar_24h[[ min.m_24h ]]),
                     freq = clwo_mar_24h[[ min.m_24h ]],
                     max.words = 100,
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```



For the largest size cluster, it seems to be more like daily family life as well as the new discoveries as can seen words like “new”, “found”, “moment”, “life”, “school”, “car”. For the smallest size cluster, it looks like casual relaxing event, for example, walking a dog in a beautiful sunny day.

Now, let’s look at married people’s happy moment for last 3 months:

```r
max.m_3m = which(m_3m$size==max(m_3m$size))[1]
min.m_3m = which(m_3m$size==min(m_3m$size))[1]
wordcloud::wordcloud(words = names(clwo_mar_3m[[ max.m_3m ]]),
                     freq = clwo_mar_3m[[ max.m_3m ]],
                     max.words = 100,
                     scale = c(4,.2),
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```



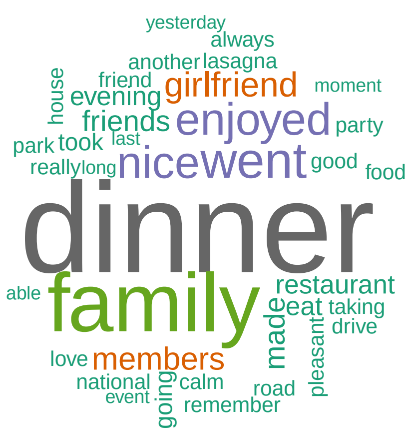

```r
wordcloud::wordcloud(words = names(clwo_mar_3m[[ min.m_3m ]]),
                     freq = clwo_mar_3m[[ min.m_3m ]],
                     max.words = 100,
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```

For largest size cluster, a longer term happy moment can be outing or having a trip as can see words include “went”, “vacation”, “trip”, “tour”. For smallest size cluster, it deals with consuming or fixing a new item like buying a new car, laptop or words like “expensive”.

Now, turn to singled people’s happy moment of last 24 hours:

```r
max.s_24h = which(s_24h$size==max(s_24h$size))[1]
min.s_24h = which(s_24h$size==min(s_24h$size))[1]
wordcloud::wordcloud(words = names(clwo_sin_24h[[ max.s_24h ]]),
                     freq = clwo_sin_24h[[ max.s_24h ]],
                     max.words = 100,
                     scale = c(4,.2),
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```


```
wordcloud::wordcloud(words = names(clwo_sin_24h[[ min.s_24h ]]),
                     freq = clwo_sin_24h[[ min.s_24h ]],
                     max.words = 100,
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```

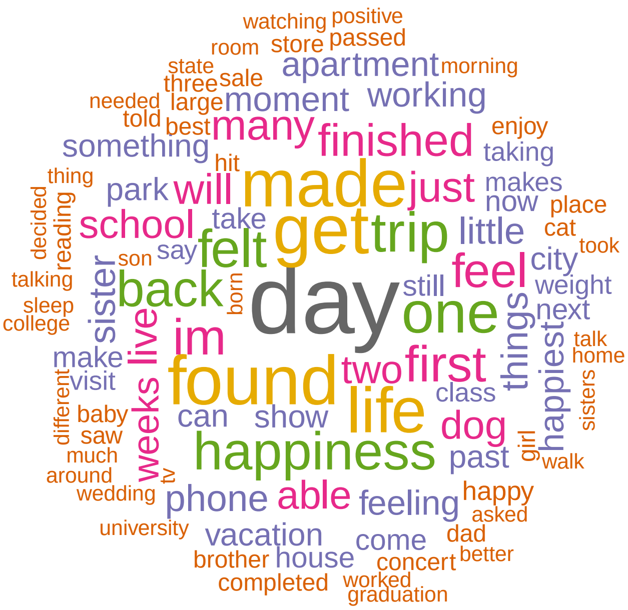


For largest size cluster, general words such as “new”,“moment”,“found” or “dog” are not quite different from married people of happy moment last 24 hours, however, we do see that there’s differnce between them like “boyfriend” would be for singled only, and interestingly, word “time” appeared. Maybe it indicates that single people enjoy more free time than married people. For smallest size cluster, this one looks like having great time with family, members or firends.

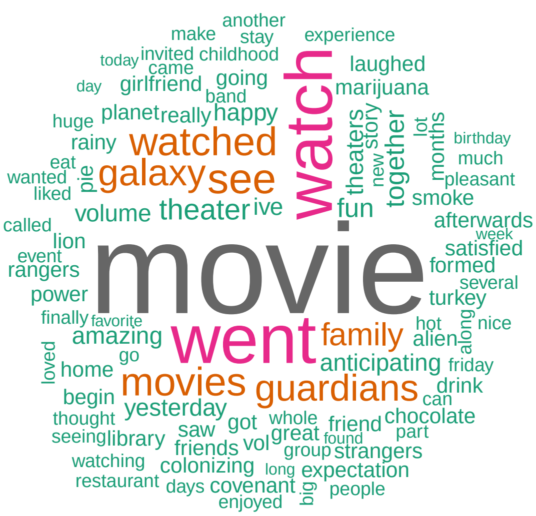
Finally, examine the single people’s happey moment of last 3 months:

```
max.s_3m = which(s_3m$size==max(s_3m$size))[1]
min.s_3m = which(s_3m$size==min(s_3m$size))[1]
wordcloud::wordcloud(words = names(clwo_sin_3m[[ max.s_3m ]]),
                     freq = clwo_sin_3m[[ max.s_3m ]],
                     max.words = 100,
                     scale=c(3,.1),
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
```

```r
                       main = "Top words in cluster")
```




```r
wordcloud::wordcloud(words = names(clwo_sin_3m[[ min.s_3m ]]),
                     freq = clwo_sin_3m[[ min.s_3m ]],
                     max.words = 100,
                     random.order = FALSE,
                     colors = brewer.pal(8, "Dark2"),
                     main = "Top words in cluster")
```



For largest size cluster, we can see common word for a longer term happy moment not so different from married people including “trip” and “vacation”. For smaller size cluster, it’s like recreational activities as words like “movie”, “watch” and maybe refer to the new movie " guardians of the galaxy" then.

Summary of hierarchical clustering for happy moment text data

To sum up, for this specific case, the largest size of cluster may be quite similar among married or single people, but for the smallest size of cluster there are significantly more differences between the two groups, more interesting results may come out when increasing the sample size, using different distance metrics or choosing different cluster parameters.