Advanced Data Science Project 1

Yizhou (Alice) Mi ym2630

Sept 2018

Part I – Gender Disparity in Expressing Happiness

The aim of this project is to predict the gender of the author of each text using the basic skills of natural language processing and binary classification. More specifically, they are multinomial regression, Linear Discriminant Analysis, Decision tree, Naive Bayes, Random forest, Neural network, KNN, and Adaboost on both full text and PCA dimension reduced text. Accuracy level of each different method will be compared at the end of this document.

# Data Integration and Cleaning

# Read in Raw Data  
hm\_data <- read.csv("cleaned\_hm.csv", stringsAsFactors = FALSE)  
demo\_data <- read.csv("demographic.csv")  
  
sprintf("hm data has %s rows,%s cols", dim(hm\_data)[1],dim(hm\_data)[2])

## [1] "hm data has 100535 rows,9 cols"

sprintf("Demographic data has %s rows,%s cols", dim(demo\_data)[1],dim(demo\_data)[2])

## [1] "Demographic data has 10844 rows,6 cols"

# Merge demo data based on wid column  
hm\_data = merge(hm\_data,demo\_data,by = "wid")  
sprintf("Merged data has %s rows,%s cols", dim(hm\_data)[1],dim(hm\_data)[2])

## [1] "Merged data has 100535 rows,14 cols"

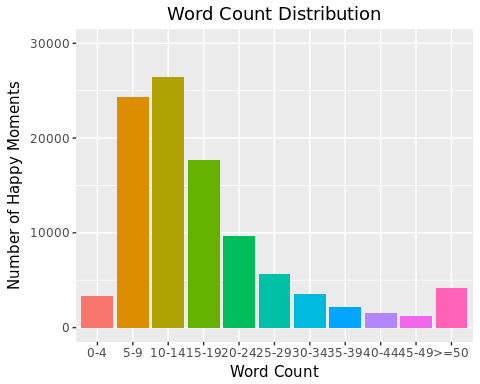
hm\_data = subset(hm\_data, gender %in% c("f","m")) # Only keep F and M as gender  
sprintf("Subsetted data has %s rows,%s cols", dim(hm\_data)[1],dim(hm\_data)[2])

## [1] "Subsetted data has 99759 rows,14 cols"

count <- sapply(hm\_data$cleaned\_hm, wordcount) # Counts number of words  
summary(count)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2.00 9.00 14.00 18.31 21.00 1155.00

category <- c("0-4","5-9","10-14","15-19","20-24","25-29","30-34","35-39",  
 "40-44","45-49",">=50")  
count\_class <- cut(count, breaks = c(0,4,9,14,19,24,29,34,39,44,49,Inf),   
 labels = category, include.lowest = TRUE)  
ggplot()+  
 geom\_bar(aes(x = count\_class, fill = count\_class))+  
 ylim(0,30000)+  
 labs(x = "Word Count", y = "Number of Happy Moments",   
 title = "Word Count Distribution")+  
 guides(fill = "none")



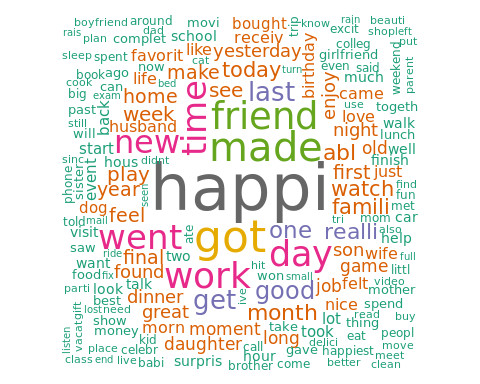
# Word Frequency

set.seed(0)  
n = 15000 # sample size  
random\_hm <- sample(1:nrow(hm\_data), n) # Working with only a random subset  
hm\_subset <- hm\_data[random\_hm,] # <- All analysis will be done on this subset  
gender <- hm\_subset$gender  
hm\_subset$gender\_int <- as.factor(ifelse(gender == "m",1,0))  
  
corpus <- Corpus(VectorSource(hm\_subset$cleaned\_hm))  
skipWords <- function(x) removeWords(x, words = c(stopwords(kind = "en")))  
funcs <- list(stripWhitespace,stemDocument,skipWords, removeNumbers, removePunctuation, tolower)  
?tm\_map  
a <- tm\_map(corpus, FUN = tm\_reduce, tmFuns = funcs)

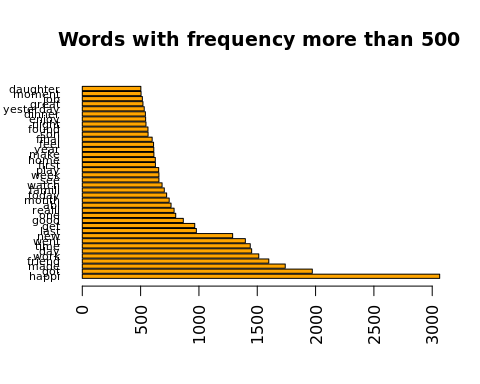
## Warning in tm\_map.SimpleCorpus(corpus, FUN = tm\_reduce, tmFuns = funcs):  
## transformation drops documents

a\_dtm <- DocumentTermMatrix(a)  
m <- as.data.frame(as.matrix(a\_dtm))  
v <- sort(colSums(m), decreasing = TRUE)  
d <- data.frame(word = names(v), freq = v)

# Word cloud  
options(warn=-1)  
wordcloud(words = d$word, freq = d$freq, colors = brewer.pal(8, "Dark2"),   
 random.order = FALSE, min.freq = 5,scale=c(4,.5),max.words=300)



# Extra - Bar chart of high frequency words  
barplot(height = d$freq[d$freq %in% 500:max(d$freq)],   
 names.arg = d$word[d$freq %in% 500:max(d$freq)],   
 horiz = TRUE, col = "orange", las = 2, cex.names = 0.7,  
 main = "Words with frequency more than 500")



# Check whether high frequency words are noise

# Check whether high frequency data is noise data  
agg1 = aggregate(. ~ hm\_subset$gender, m[,as.character(d$word[d$freq >= 500])],sum)  
f\_ratio = agg1[1,-1] / colSums(agg1[,-1])  
m\_ratio = agg1[2,-1] / colSums(agg1[,-1])  
  
ratio = rbind(f\_ratio,m\_ratio)  
ratio

## happi got made friend work day time  
## 1 0.4676893 0.4137931 0.4643678 0.3996248 0.386649 0.4387052 0.4256944  
## 2 0.5323107 0.5862069 0.5356322 0.6003752 0.613351 0.5612948 0.5743056  
## went new last get good one realli  
## 1 0.4170243 0.3630721 0.3558282 0.4585062 0.4099307 0.3957553 0.4841169  
## 2 0.5829757 0.6369279 0.6441718 0.5414938 0.5900693 0.6042447 0.5158831  
## abl month today famili watch see week  
## 1 0.4388962 0.4107383 0.4536653 0.428165 0.3918129 0.4322679 0.414003  
## 2 0.5611038 0.5892617 0.5463347 0.571835 0.6081871 0.5677321 0.585997  
## play first home make year feel final  
## 1 0.3419847 0.4258373 0.4392971 0.465798 0.4486134 0.4255319 0.4307179  
## 2 0.6580153 0.5741627 0.5607029 0.534202 0.5513866 0.5744681 0.5692821  
## son found night enjoy dinner yesterday great  
## 1 0.5992908 0.4547069 0.4047619 0.4235727 0.4713494 0.4632768 0.3865385  
## 2 0.4007092 0.5452931 0.5952381 0.5764273 0.5286506 0.5367232 0.6134615  
## job moment daughter  
## 1 0.415534 0.4642857 0.5836653  
## 2 0.584466 0.5357143 0.4163347

# High Frequency Words that can be predictive about gender   
ratio[,ratio[1,] < 0.45 | ratio[1,] > 0.55]

## got friend work day time went new  
## 1 0.4137931 0.3996248 0.386649 0.4387052 0.4256944 0.4170243 0.3630721  
## 2 0.5862069 0.6003752 0.613351 0.5612948 0.5743056 0.5829757 0.6369279  
## last good one abl month famili watch  
## 1 0.3558282 0.4099307 0.3957553 0.4388962 0.4107383 0.428165 0.3918129  
## 2 0.6441718 0.5900693 0.6042447 0.5611038 0.5892617 0.571835 0.6081871  
## see week play first home year feel  
## 1 0.4322679 0.414003 0.3419847 0.4258373 0.4392971 0.4486134 0.4255319  
## 2 0.5677321 0.585997 0.6580153 0.5741627 0.5607029 0.5513866 0.5744681  
## final son night enjoy great job daughter  
## 1 0.4307179 0.5992908 0.4047619 0.4235727 0.3865385 0.415534 0.5836653  
## 2 0.5692821 0.4007092 0.5952381 0.5764273 0.6134615 0.584466 0.4163347

# Noise words  
ratio[,ratio[1,] >= 0.45 & ratio[1,] <= 0.55]

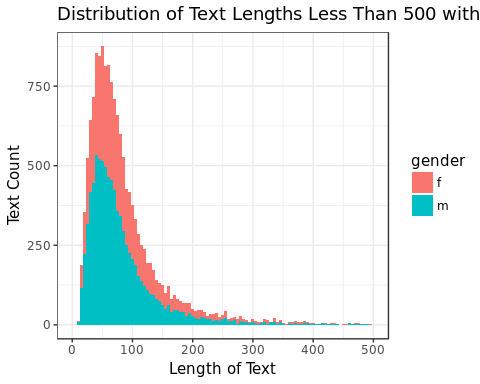
## happi made get realli today make found  
## 1 0.4676893 0.4643678 0.4585062 0.4841169 0.4536653 0.465798 0.4547069  
## 2 0.5323107 0.5356322 0.5414938 0.5158831 0.5463347 0.534202 0.5452931  
## dinner yesterday moment  
## 1 0.4713494 0.4632768 0.4642857  
## 2 0.5286506 0.5367232 0.5357143

# Set skipwords to be those of which are not so distinguishable between genders

skipwords = colnames(ratio[,ratio[1,] >= 0.45 & ratio[1,] <= 0.55])

Females tend to be more expressive about their feelings.

# Check whether word length differ by gender  
ggplot(hm\_subset, aes(x = nchar(cleaned\_hm), fill = gender)) +  
 xlim(0,500) +  
 theme\_bw() +  
 geom\_histogram(binwidth = 5) +  
 labs(y = "Text Count", x = "Length of Text",  
 title = "Distribution of Text Lengths Less Than 500 with Class Labels")



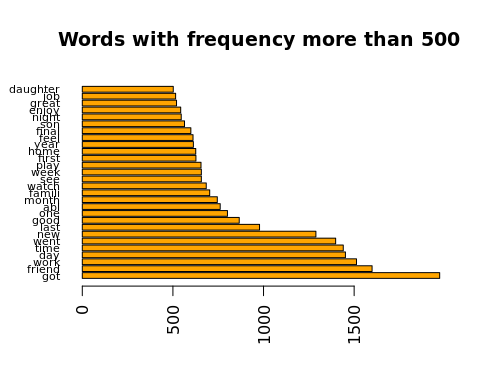
# Removing noise words

# Filter out noise words  
new\_a <- tm\_map(a,removeWords, words = c(stopwords(kind = "en"),skipwords))  
new\_a\_dtm <- DocumentTermMatrix(new\_a)  
new\_m <- as.matrix(new\_a\_dtm)  
new\_v <- sort(colSums(new\_m), decreasing = TRUE)  
new\_d <- data.frame(word = names(new\_v), freq = new\_v)

# Word cloud after stripping extra skipwords   
wordcloud(words = new\_d$word, freq = new\_d$freq, colors = brewer.pal(8, "Dark2"),   
 random.order = FALSE, min.freq = 5,max.words = 300,scale=c(4,.5))

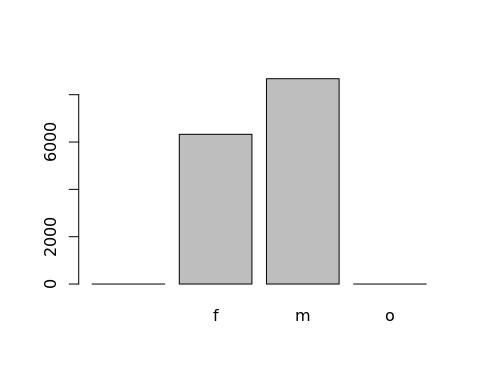


# Extra - High frequency words after stripping unnecessary words  
barplot(height = new\_d$freq[new\_d$freq %in% 500:max(new\_d$freq)],   
 names.arg = new\_d$word[new\_d$freq %in% 500:max(new\_d$freq)],   
 horiz = TRUE, col = "orange", las = 2, cex.names = 0.7,  
 main = "Words with frequency more than 500")



# Generalized Linear Models with Lasso Penalty with gender

# Quick visualization of distribution of gender  
barplot(table(hm\_subset$gender))



# First 70% as training data, rest 30% as test data.   
# 5-fold cross validation done later in the training set  
# Sample train id  
train\_id <- sample(1:dim(hm\_subset)[1],dim(hm\_subset)[1] \* 0.7)  
  
# Filter out infrequent words  
new\_a\_dtm <- removeSparseTerms(new\_a\_dtm, 0.999)  
dtm <- as.data.frame(as.matrix(new\_a\_dtm))  
dtm$gender\_int = hm\_subset$gender\_int  
  
sprintf("Subsetted data has %s rows,%s cols", dim(dtm)[1],dim(dtm)[2])

## [1] "Subsetted data has 15000 rows,1119 cols"

dtm\_train <- dtm[train\_id, ]  
dtm\_test <- dtm[-train\_id, ]  
  
train\_X <- dtm\_train[,!colnames(dtm\_train) %in% c("gender\_int")]  
test\_X <- dtm\_test[,!colnames(dtm\_test) %in% c("gender\_int")]  
  
train\_Y <- dtm\_train$gender\_int  
test\_Y <- dtm\_test$gender\_int

# Use PCA to reduce dimensionality  
start\_time = Sys.time()  
pca = preProcess(x = train\_X, method = 'pca', thresh = 0.95)  
pca\_train = predict(pca, train\_X)  
pca\_test = predict(pca, test\_X)  
  
end\_time = Sys.time()  
# Processing Time  
end\_time - start\_time

## Time difference of 54.17838 secs

sprintf("PCA train has %s rows,%s cols", dim(pca\_train)[1],dim(pca\_train)[2])

## [1] "PCA train has 10500 rows,955 cols"

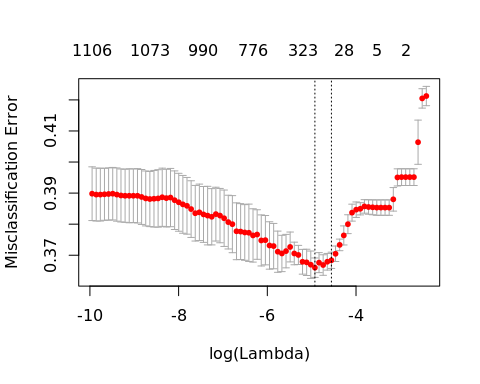
sprintf("PCA test has %s rows,%s cols", dim(pca\_test)[1],dim(pca\_test)[2])

## [1] "PCA test has 4500 rows,955 cols"

# Creating 5 Folds Training and Testing ids  
flds <- createFolds(1:dim(train\_X)[1], k = 5, list = TRUE, returnTrain = FALSE)  
  
# Create Result Holder  
result = NULL  
# Store Predicted Y value  
Ypreds = NULL  
  
# Voting function based on predicted results  
vote = function(pred\_dataframe){  
 for(i in 1:dim(pred\_dataframe)[2]){  
 pred\_dataframe[,i] = as.numeric(as.character(pred\_dataframe[,i]))  
 }  
 votes = rowSums(pred\_dataframe)  
 votes = votes / dim(pred\_dataframe)[2]  
 final\_vote = ifelse(votes > 0.5,1,0)  
 return(final\_vote)  
}

## Logistic regression with Lasso Penalty

# Fitting the classifier using logistic regression  
start\_time = Sys.time()  
lg\_classifier <- cv.glmnet(x = as.matrix(train\_X), y = train\_Y,  
 family = 'binomial', alpha = 1, type.measure = "class",  
 nfolds = 5, thresh = 1e-3, maxit = 1e3)  
plot(lg\_classifier)



end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 55.39568 secs

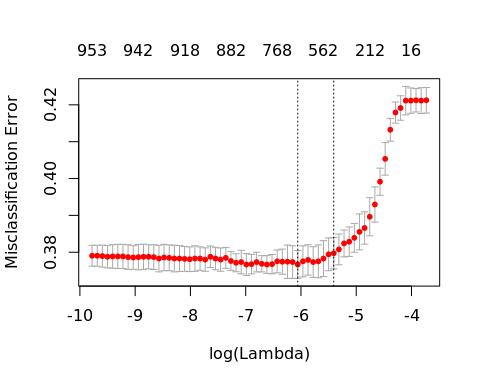
# Evaluating the performance of our classifier on test data  
preds <- predict(lg\_classifier, as.matrix(train\_X), type = "class")  
train\_accuracy <- sum(preds == train\_Y,1,0) / length(train\_Y)  
  
preds <- predict(lg\_classifier, as.matrix(test\_X), type = "class")  
Ypreds = data.frame(glm\_full = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = data.frame(type = "Glmnet",var = "Full",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F)  
  
sprintf("Glmnet With LASSO Penalty")

## [1] "Glmnet With LASSO Penalty"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.639, Testing accuracy: 0.630"

# Fitting the classifier using logistic regression  
start\_time = Sys.time()  
lg\_classifier\_pca <- cv.glmnet(x = as.matrix(pca\_train), y = train\_Y,  
 family = 'binomial', alpha = 1, type.measure = "class",  
 nfolds = 5, thresh = 1e-3, maxit = 1e3)  
plot(lg\_classifier\_pca)



end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 47.72582 secs

# Evaluating the performance of our classifier on test data  
preds <- predict(lg\_classifier\_pca, as.matrix(pca\_train), type = "class")  
train\_accuracy <- sum(preds == train\_Y,1,0) / length(train\_Y)  
  
preds <- predict(lg\_classifier\_pca, as.matrix(pca\_test), type = "class")  
Ypreds = data.frame(Ypreds,glm\_pca = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Glmnet",var = "PCA",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Glmnet With LASSO Penalty and PCA")

## [1] "Glmnet With LASSO Penalty and PCA"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.681, Testing accuracy: 0.630"

## Linear Discriminant Analysis after PCA

LDA complains about muticolinearity issue, so we only fit PCA version.

start\_time = Sys.time()  
# 5-Fold cross validation on LDA after PCA  
  
cv\_orig = NULL  
cv\_pred = NULL  
lda\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "MASS") %dopar%{  
 train\_Fold = pca\_train[-flds[[i]],]  
 cv\_Fold = pca\_train[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
   
lda = lda(formula = train\_Y[-flds[[i]]] ~ ., data = train\_Fold)  
  
# Predict cross validation part  
preds = as.data.frame(predict(lda, cv\_Fold))$class  
cv\_orig = as.numeric(as.character(cv\_Y))  
cv\_pred = as.numeric(as.character(preds))  
  
# Get testing prediction  
preds = as.data.frame(predict(lda, pca\_test))$class  
list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, lda\_pred = as.numeric(as.character(preds)))  
}  
  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(lda\_preds)){  
 lda\_preds = res[[i]]$lda\_pred  
 }else{  
 lda\_preds = cbind(lda\_preds,res[[i]]$lda\_pred)  
 }  
}  
  
end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 49.94935 secs

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(lda\_preds)  
Ypreds = data.frame(Ypreds,lda\_pca = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "LDA",var = "PCA",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Linear Discriminant Analysis with PCA")

## [1] "Linear Discriminant Analysis with PCA"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.617, Testing accuracy: 0.631"

## Decision Tree

Decision Tree

start\_time = Sys.time()  
# 5-Fold cross validation Decision Tree model  
  
cv\_orig = NULL  
cv\_pred = NULL  
tree\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "rpart") %dopar%{  
 train\_Fold = train\_X[-flds[[i]],]  
 cv\_Fold = train\_X[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
  
tree = rpart(formula = train\_Y[-flds[[i]]] ~ ., data = train\_Fold)  
  
# Predict cross validation part  
preds = predict(tree, cv\_Fold,type = "class")  
cv\_orig = as.numeric(as.character(cv\_Y))  
cv\_pred = as.numeric(as.character(preds))  
  
# Get testing prediction  
preds = predict(tree, test\_X,type = "class")  
preds = as.numeric(as.character(preds))  
  
list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, tree\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(tree\_preds)){  
 tree\_preds = res[[i]]$tree\_preds  
 }else{  
 tree\_preds = cbind(tree\_preds,res[[i]]$tree\_preds)  
 }  
}  
end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 1.405228 mins

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(tree\_preds)  
Ypreds = data.frame(Ypreds,tree = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Tree",var = "Full",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Decision Tree")

## [1] "Decision Tree"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.613, Testing accuracy: 0.613"

# Decision Tree with PCA

start\_time = Sys.time()  
# 5-Fold cross validation Decision Tree model  
  
cv\_orig = NULL  
cv\_pred = NULL  
tree\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "rpart") %dopar%{  
 train\_Fold = pca\_train[-flds[[i]],]  
 cv\_Fold = pca\_train[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
  
tree = rpart(formula = train\_Y[-flds[[i]]] ~ ., data = train\_Fold)  
  
# Predict cross validation part  
preds = predict(tree, cv\_Fold,type = "class")  
cv\_orig = as.numeric(as.character(cv\_Y))  
cv\_pred = as.numeric(as.character(preds))  
# Get testing prediction  
preds = predict(tree, pca\_test,type = "class")  
preds = as.numeric(as.character(preds))  
  
list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, tree\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(tree\_preds)){  
 tree\_preds = res[[i]]$tree\_preds  
 }else{  
 tree\_preds = cbind(tree\_preds,res[[i]]$tree\_preds)  
 }  
}  
end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 1.431357 mins

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(tree\_preds)  
Ypreds = data.frame(Ypreds,tree = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Decision Tree",var = "PCA",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Decision Tree with PCA")

## [1] "Decision Tree with PCA"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.578, Testing accuracy: 0.584"

## Naive Bayes

start\_time = Sys.time()  
# 5-Fold cross validation Naive Bayes  
cv\_orig = NULL  
cv\_pred = NULL  
nb\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "e1071") %dopar%{  
 print(i)  
 train\_Fold = train\_X[-flds[[i]],]  
 cv\_Fold = train\_X[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
  
naiveBayes = naiveBayes(formula = train\_Y[-flds[[i]]] ~ ., data = train\_Fold)  
  
# Predict cross validation part  
preds = predict(naiveBayes, cv\_Fold,type = "class", threshold = 0.01)  
cv\_orig = as.numeric(as.character(cv\_Y))  
cv\_pred = as.numeric(as.character(preds))  
# Get testing prediction  
preds = predict(naiveBayes, test\_X,type = "class",threshold = 0.01)  
preds = as.numeric(as.character(preds))  
  
list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, nb\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(tree\_preds)){  
 nb\_preds = res[[i]]$nb\_preds  
 }else{  
 nb\_preds = cbind(nb\_preds,res[[i]]$nb\_preds)  
 }  
}  
end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 3.916792 mins

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(nb\_preds)  
Ypreds = data.frame(Ypreds,nb = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Naive Bayes",var = "Full",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Naive Bayes")

## [1] "Naive Bayes"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.555, Testing accuracy: 0.558"

# Naive Bayes with PCA

start\_time = Sys.time()  
# 5-Fold cross validation Naive Bayes  
  
cv\_orig = NULL  
cv\_pred = NULL  
nb\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "e1071") %dopar%{  
 train\_Fold = pca\_train[-flds[[i]],]  
 cv\_Fold = pca\_train[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
  
naiveBayes = naiveBayes(formula = train\_Y[-flds[[i]]] ~ ., data = train\_Fold)  
  
# Predict cross validation part  
preds = predict(naiveBayes, cv\_Fold,type = "class",threshold = 0.01)  
cv\_orig = as.numeric(as.character(cv\_Y))  
cv\_pred = as.numeric(as.character(preds))  
   
# Get testing prediction  
preds = predict(naiveBayes, pca\_test,type = "class",threshold = 0.01)  
preds = as.numeric(as.character(preds))  
   
  
list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, nb\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(tree\_preds)){  
 nb\_preds = res[[i]]$nb\_preds  
 }else{  
 nb\_preds = cbind(nb\_preds,res[[i]]$nb\_preds)  
 }  
}  
end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 3.261237 mins

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(nb\_preds)  
Ypreds = data.frame(Ypreds,nb\_pca = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Naive Bayes",var = "PCA",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Naive Bayes with PCA")

## [1] "Naive Bayes with PCA"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.542, Testing accuracy: 0.543"

## Random Forest

Random Forest is known to be robust because it’s an ensemble of week learners. So we fit all variables to random forest model.

start\_time = Sys.time()  
# 5-Fold cross validation Random Forest  
  
cv\_orig = NULL  
cv\_pred = NULL  
rf\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "randomForest") %dopar%{  
 train\_Fold = train\_X[-flds[[i]],]  
 cv\_Fold = train\_X[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
  
rf <- randomForest(y = train\_Y[-flds[[i]]],x = train\_Fold, mtry = 100, data = train\_Fold, ntree = 500, maxnodes = 15)  
  
# Predict cross validation part  
preds = predict(rf, cv\_Fold,type = "class")  
cv\_orig = as.numeric(as.character(cv\_Y))  
cv\_pred = as.numeric(as.character(preds))  
  
# Get testing prediction  
preds = predict(rf, test\_X,type = "class")  
preds = as.numeric(as.character(preds))  
  
list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, rf\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(tree\_preds)){  
 rf\_preds = res[[i]]$rf\_preds  
 }else{  
 rf\_preds = cbind(rf\_preds,res[[i]]$rf\_preds)  
 }  
}  
end\_time = Sys.time()  
  
# Processing time  
end\_time - start\_time

## Time difference of 17.90224 mins

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(rf\_preds)  
Ypreds = data.frame(Ypreds,rf = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Random Forest",var = "Full",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Random Forest")

## [1] "Random Forest"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.581, Testing accuracy: 0.580"

start\_time = Sys.time()  
# 5-Fold cross validation KNN  
  
cv\_orig = NULL  
cv\_pred = NULL  
knn\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "class") %dopar%{  
 train\_Fold = train\_X[-flds[[i]],]  
 cv\_Fold = train\_X[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
   
   
 # Predict cross validation part  
 preds = knn(train = train\_Fold,  
 test = cv\_Fold,  
 cl = train\_Y[-flds[[i]]],  
 k = 5,  
 prob = F)  
 cv\_orig = as.numeric(as.character(cv\_Y))  
 cv\_pred = as.numeric(as.character(preds))  
   
 # Predict Test  
 preds = knn(train = train\_Fold,  
 test = test\_X,  
 cl = train\_Y[-flds[[i]]],  
 k = 5,  
 prob = F)  
 preds = as.numeric(as.character(preds))  
   
 list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, knn\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(knn\_preds)){  
 knn\_preds = res[[i]]$knn\_preds  
 }else{  
 knn\_preds = cbind(knn\_preds,res[[i]]$knn\_preds)  
 }  
}  
end\_time = Sys.time()

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(knn\_preds)  
Ypreds = data.frame(Ypreds,knn = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "KNN",var = "Full",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("K-Nearest Neighbour")

## [1] "K-Nearest Neighbour"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.588, Testing accuracy: 0.590"

### Adaboosting

start\_time = Sys.time()  
# 5-Fold cross validation Adaboost  
  
cv\_orig = NULL  
cv\_pred = NULL  
adb\_preds = NULL # <- result holder to save test prediction  
res = foreach(i = 1:length(flds),.packages = "fastAdaboost") %dopar%{  
 train\_Fold = train\_X[-flds[[i]],]  
 cv\_Fold = train\_X[flds[[i]],]  
 cv\_Y = train\_Y[flds[[i]]]  
 train\_Fold$Y = train\_Y[-flds[[i]]]  
 adaboost = adaboost(formula = Y ~ ., data = train\_Fold,nIter = 10)  
   
 # Predict cross validation part  
 preds = predict(adaboost, cv\_Fold)$class  
 cv\_orig = as.numeric(as.character(cv\_Y))  
 cv\_pred = as.numeric(as.character(preds))  
   
 # Get testing prediction  
 preds = predict(adaboost, test\_X)$class  
 preds = as.numeric(as.character(preds))  
   
 list(cv\_orig = cv\_orig, cv\_pred = cv\_pred, adb\_preds = preds)  
}  
  
for(i in 1:length(res)){  
 cv\_orig = c(cv\_orig,res[[i]]$cv\_orig)  
 cv\_pred = c(cv\_pred,res[[i]]$cv\_pred)  
 if(is.null(adb\_preds)){  
 adb\_preds = res[[i]]$adb\_preds  
 }else{  
 adb\_preds = cbind(adb\_preds,res[[i]]$adb\_preds)  
 }  
}  
end\_time = Sys.time()

# Evaluating the performance of our classifier on test data  
train\_accuracy <- sum(cv\_pred == cv\_orig,1,0) / length(cv\_orig)  
  
preds <- vote(adb\_preds)  
Ypreds = data.frame(Ypreds,adb = preds) # <- Save for future use  
test\_accuracy <- sum(preds == test\_Y,1,0) / length(test\_Y)  
  
result = rbind(result,  
 data.frame(type = "Adaboost",var = "Full",train\_accuracy = train\_accuracy, test\_accuracy = test\_accuracy,stringsAsFactors = F))  
  
sprintf("Adaboost")

## [1] "Adaboost"

sprintf("Training accuracy: %.3f, Testing accuracy: %.3f", train\_accuracy, test\_accuracy)

## [1] "Training accuracy: 0.631, Testing accuracy: 0.629"

### Result Summary

result

## type var train\_accuracy test\_accuracy  
## 1 Glmnet Full 0.6391429 0.6297778  
## 2 Glmnet PCA 0.6810476 0.6295556  
## 3 LDA PCA 0.6167619 0.6308889  
## 4 Tree Full 0.6125714 0.6128889  
## 5 Decision Tree PCA 0.5782857 0.5842222  
## 6 Naive Bayes Full 0.5553333 0.5582222  
## 7 Naive Bayes PCA 0.5424762 0.5431111  
## 8 Random Forest Full 0.5814286 0.5800000  
## 9 KNN Full 0.5876190 0.5904444  
## 10 Adaboost Full 0.6313333 0.6291111

# Ensemble of all models  
fin = vote(Ypreds)  
test\_accuracy <- sum(fin == test\_Y,1,0) / length(test\_Y)  
test\_accuracy

## [1] 0.6273333

Models using PCA dim reduced dataset are in general perform worse than their corresponding models using the full dataset with a lower accuracy level on test dataset. The only exception is the LDA model, of which was not able to perform on R due to a warning of multicolinearity issue, and it gave the best prediction accuracy level on the test dataset. Naïve Bayes is the worst model for this specific binary classification of gender using this dataset including both full text and PCA dim reduced text.

LDA on PCA text, Adaboost on full text, and Lasso on both versions of text outperform all other models in predicting the gender of the author who created this happy moments dataset.

There is some difference in expressing happiness by different genders, but not as much difference as expected. Gender classification is better than random guess.

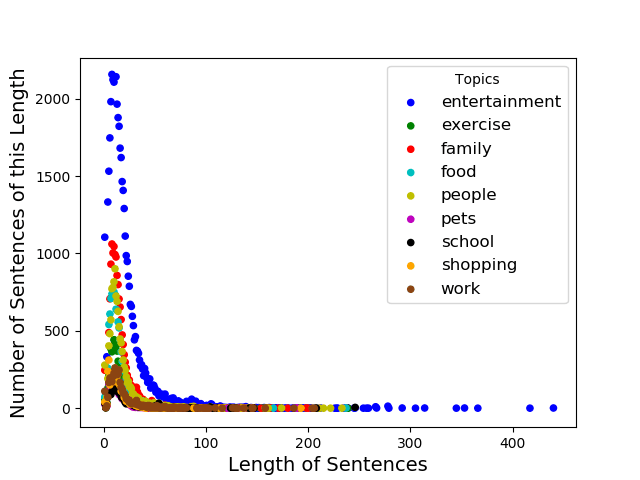
Technology used: rstudio server on Google Cloud Platform

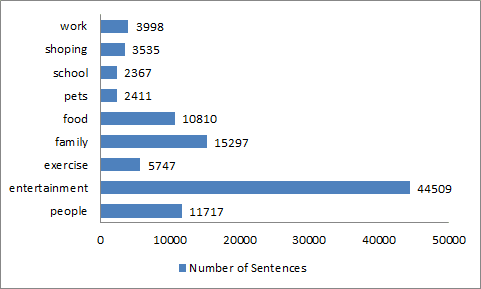
It would take approximately an hour to run the whole rmd file on Google Cloud Platform.

Part II – Topic Analysis

Ten topics are entertainment, exercise, family, food, overall, people, pets, school, shopping, and work respectively.

Here is a scatter plot showing the relationship between number of sentences of each length and the length of sentences when people express different topics of which make them happy.

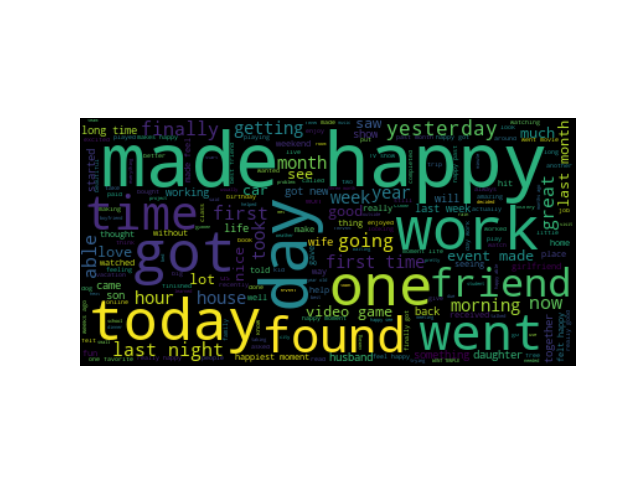




As shown by the graph above, people’s happiness is the most often related to entertainment, and the least related to school.

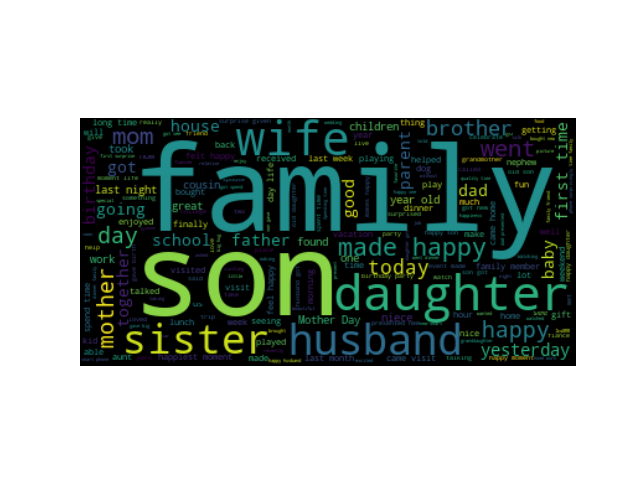
In a descending order of happiness, these topics are respectively entertainment, family, people, food, exercise, work, shopping, pets, and school. Surprising, shopping does not bring people much more happiness to people as expected.

Here are the word clouds of each category above:



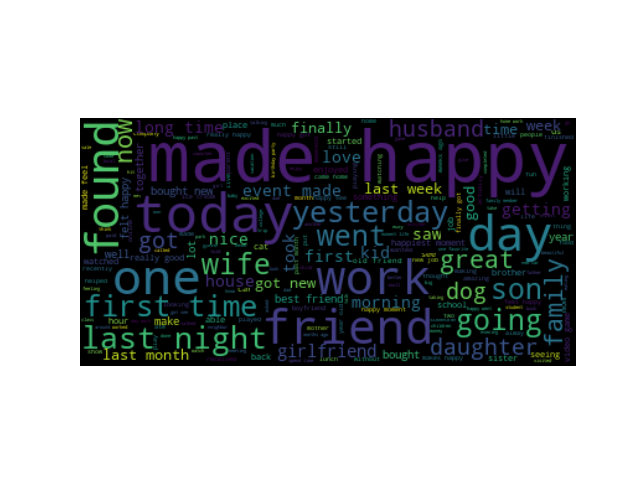
Word Cloud of Entertainment

Word Cloud of Exercise



Word Cloud of Family

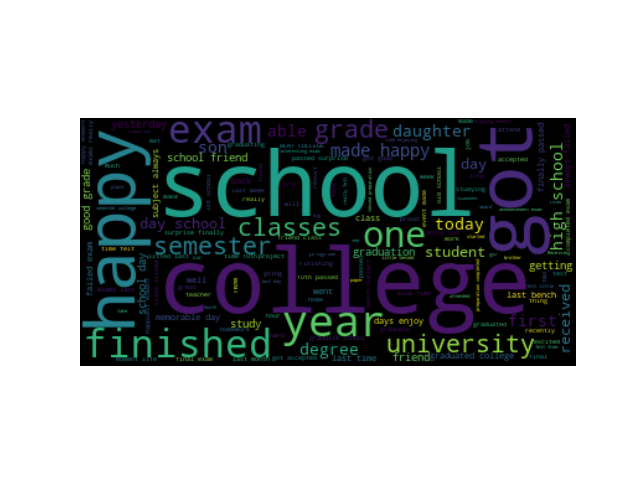
 Word Cloud of Food

Word Cloud of Overall

 Word Cloud of People



Word Cloud of Pets



Word Cloud of School



Word Cloud of Shopping



Word Cloud of Work

Technology used: Python