Final Project

Team 1

May 11, 2018

Data Exploration

```
str(trainRaw)
```

```
## Classes 'data.table' and 'data.frame':   109185 obs. of  4 variables:
##  $ PhraseId  : int  1 3 6 7 9 10 12 13 14 15 ...
##  $ SentenceId: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Phrase    : chr  "A series of escapades demonstrating the adage that what is
good for the goose is also good for the gander , som"| __truncated__ "A series" "of
escapades demonstrating the adage that what is good for the goose" "of" ...
##  $ Sentiment : int  1 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
summary(trainRaw)
```

```
##     PhraseId        SentenceId      Phrase            Sentiment
##  Min.   :     1   Min.   :   1   Length:109185      Min.   :0.000
##  1st Qu.: 38720   1st Qu.:1845   Class :character   1st Qu.:2.000
##  Median : 77770   Median :3999   Mode  :character   Median :2.000
##  Mean   : 77882   Mean   :4072                      Mean   :2.066
##  3rd Qu.:116950   3rd Qu.:6239                      3rd Qu.:3.000
##  Max.   :156060   Max.   :8544                      Max.   :4.000
```

```
table(trainRaw$Sentiment)
```

```
##
##     0     1     2     3     4
##  4936 18952 55717 23154  6426
```

```
summary(trainRaw$SentenceId)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1    1845    3999    4072    6239    8544
```

```
train <- copy(trainRaw)

train <- data.table(train)
```

Prepare data - get the whole reviews from data

```
setkeyv(train, c("SentenceId", "PhraseId"))

train <- train[, sent_start:=min(PhraseId), by=.(SentenceId)]

senTrain <- train[ sent_start == PhraseId][, sent_start := NULL]

setkeyv(test, c("SentenceId", "PhraseId"))

test <- test[, sent_start:=min(PhraseId), by=.(SentenceId)]

senTest <- test[ sent_start == PhraseId][, sent_start := NULL]
```

Text mining - Using tm package

```
senTrain.tm<- copy(senTrain)
setnames(senTrain.tm, "SentenceId", "doc_id")
setnames(senTrain.tm, "Phrase", "text")

reviewCorpus <- Corpus(DataframeSource(senTrain.tm))
inspect(reviewCorpus[[1]])
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 188
##
## A series of escapades demonstrating the adage that what is good for the goose is
also good for the gander , some of which occasionally amuses but none of which amou
nts to much of a story .
```

```
reviewCorpus <- tm_map(reviewCorpus, content_transformer(tolower))
```

What are stopwords?

```
stopwords("english")
```

```
##   [1] "i"          "me"         "my"         "myself"     "we"
##   [6] "our"        "ours"       "ourselves"  "you"        "your"
##  [11] "yours"      "yourself"   "yourselves" "he"         "him"
##  [16] "his"        "himself"    "she"        "her"        "hers"
##  [21] "herself"    "it"         "its"        "itself"     "they"
##  [26] "them"       "their"      "theirs"     "themselves" "what"
##  [31] "which"      "who"        "whom"       "this"       "that"
##  [36] "these"      "those"      "am"         "is"         "are"
##  [41] "was"        "were"       "be"         "been"       "being"
##  [46] "have"       "has"        "had"        "having"     "do"
##  [51] "does"       "did"        "doing"      "would"      "should"
##  [56] "could"      "ought"      "i'm"        "you're"     "he's"
##  [61] "she's"      "it's"       "we're"      "they're"    "i've"
##  [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
##  [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
##  [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
##  [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
##  [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
##  [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
##  [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
## [101] "who's"      "what's"     "here's"     "there's"    "when's"
## [106] "where's"    "why's"      "how's"      "a"          "an"
## [111] "the"        "and"        "but"        "if"         "or"
## [116] "because"    "as"         "until"      "while"      "of"
## [121] "at"         "by"         "for"        "with"       "about"
## [126] "against"    "between"    "into"       "through"    "during"
## [131] "before"     "after"      "above"      "below"      "to"
## [136] "from"       "up"         "down"       "in"         "out"
## [141] "on"         "off"        "over"       "under"      "again"
## [146] "further"    "then"       "once"       "here"       "there"
## [151] "when"       "where"      "why"        "how"        "all"
## [156] "any"        "both"       "each"       "few"        "more"
## [161] "most"       "other"      "some"       "such"       "no"
## [166] "nor"        "not"        "only"       "own"        "same"
## [171] "so"         "than"       "too"        "very"
```

Exclude some stopwords

```r
exceptions<- c('but','only','too','not','nor','most','again','because')
my_stopwords <- setdiff(stopwords("english"), exceptions)
```

Eliminate punctuation, white space, stopWords, numbers

```r
skipWords <- function(x) removeWords(x, my_stopwords)
funcs <- list(removePunctuation, removeNumbers, stripWhitespace, skipWords)
cleanText <- tm_map(reviewCorpus, FUN = tm_reduce, tmFuns = funcs)
inspect(cleanText[[1]])
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 115
##
##  series escapades demonstrating adage good goose also good gander  occasionally amuses but none amounts much story
```

Finding most frequent words with length from 3-20 characters

```
freqMatrix <- TermDocumentMatrix(cleanText, control = list(wordLengths = c(3,20)))
inspect(freqMatrix)
```

```
## <<TermDocumentMatrix (terms: 14924, documents: 8505)>>
## Non-/sparse entries: 72527/126856093
## Sparsity           : 100%
## Maximal term length: 20
## Weighting          : term frequency (tf)
## Sample             :
##        Docs
## Terms   1019 2527 2728 3151 403 4547 5535 5711 7670 8130
##   but      0    0    0    0   0    1    0    2    1    1
##   film     0    0    0    1   0    0    0    0    0    0
##   like     0    1    0    0   0    0    0    1    0    0
##   lrb      1    0    0    1   0    1    2    1    0    0
##   movie    1    0    0    0   1    0    0    0    0    1
##   not      0    0    0    0   0    0    0    0    1    1
##   one      0    0    1    0   0    1    0    0    0    0
##   rrb      1    0    0    1   0    1    2    1    0    0
##   story    0    0    0    0   0    0    0    0    0    0
##   too      0    0    0    0   0    1    0    0    0    0
```

Investigare rrb and lrb

```
senTrainsubset <- senTrain[ grep("lrb", senTrain$Phrase), ]
```

```
sentence
```

```
## [1] "the sensational true-crime hell-jaunt purists might like and more experimental in its storytelling -lrb- though no less horrifying for it -rrb- ."
```

Pick only 200 most frequent words

```
words.200
```

```
##   [1] "but"          "film"         "movie"        "not"
##   [5] "one"          "like"         "rrb"          "lrb"
##   [9] "story"        "too"          "just"         "most"
##  [13] "good"         "much"         "comedy"       "will"
##  [17] "can"          "even"         "time"         "characters"
##  [21] "only"         "funny"        "little"       "way"
##  [25] "never"        "enough"       "make"         "director"
##  [29] "may"          "work"         "love"         "bad"
##  [33] "makes"        "movies"       "best"         "life"
##  [37] "new"          "made"         "drama"        "many"
##  [41] "well"         "really"       "something"    "without"
##  [45] "better"       "plot"         "see"          "performances"
##  [49] "films"        "people"       "look"         "every"
##  [53] "two"          "action"       "great"        "nothing"
##  [57] "also"         "long"         "though"       "big"
##  [61] "cast"         "might"        "still"        "first"
##  [65] "another"      "get"          "feel"         "ever"
##  [69] "fun"          "character"    "audience"     "minutes"
##  [73] "humor"        "sense"        "world"        "yet"
##  [77] "performance"  "script"       "often"        "because"
##  [81] "thing"        "hard"         "kind"         "thriller"
##  [85] "real"         "comes"        "documentary"  "end"
##  [89] "entertaining" "less"         "seems"        "tale"
##  [93] "feels"        "man"          "lot"          "moments"
##  [97] "quite"        "far"          "picture"      "watching"
## [101] "seen"         "take"         "interesting"  "screen"
## [105] "almost"       "rather"       "family"       "hollywood"
## [109] "heart"        "full"         "things"       "original"
## [113] "right"        "find"         "worth"        "ultimately"
## [117] "year"         "romantic"     "back"         "acting"
## [121] "old"          "watch"        "times"        "american"
## [125] "material"     "dialogue"     "actors"       "despite"
## [129] "come"         "compelling"   "scenes"       "human"
## [133] "works"        "cinema"       "young"        "least"
## [137] "gets"         "seem"         "think"        "want"
## [141] "bit"          "piece"        "give"         "music"
## [145] "again"        "sometimes"    "going"        "making"
## [149] "takes"        "years"        "together"     "emotional"
## [153] "special"      "kids"         "say"          "gives"
## [157] "know"         "style"        "dark"         "fascinating"
## [161] "moving"       "women"        "subject"      "sweet"
## [165] "comic"        "last"         "dull"         "direction"
## [169] "anyone"       "show"         "need"         "matter"
## [173] "fans"         "flick"        "history"      "offers"
## [177] "anything"     "manages"      "star"         "everything"
## [181] "actually"     "point"        "goes"         "experience"
## [185] "whole"        "filmmakers"   "around"       "away"
## [189] "pretty"       "care"         "since"        "keep"
## [193] "place"        "war"          "clever"       "premise"
## [197] "plays"        "screenplay"   "short"        "probably"
## [201] "art"          "idea"
```

Remove ,()[].; from Phrase and make the

text to lower case

```r
senTrain$Phrase <- tolower(senTrain$Phrase)
senWords <- senTrain[, strsplit(Phrase,' ', fixed = T), by=.(SentenceId, Sentiment)]

setnames(senWords, "V1", "word")

wordBySent <- senWords[, (cnt=.N), by=.(word, Sentiment)]

setnames(wordBySent, "V1", "frequency")
```

Create density variable

```r
wordBySent1 <- copy(wordBySent)
wordBySent1$Sentiment <- as.factor(wordBySent1$Sentiment)
wordBySent1 <- merge(wordBySent1, data.frame(table(Sentiment = wordBySent1$Sentiment)), by = c("Sentiment"))
setnames(wordBySent1, "Freq", "class.width")

wordBySent1 <- wordBySent1[, density := (frequency/class.width)]
```

Subset the whole density table with only the “good” frequentwords from tm package

```r
densitytable.long <- wordBySent1[wordBySent1$word %chin% words.200]

densitytable <- dcast(densitytable.long, word ~ Sentiment, value.var = "density")
setnames(densitytable, c("0", "1", "2", "3", "4"), c("SN", "N", "NE", "P", "SP"))
```

Pick the sentiment with the highest density for each words

```r
densitytable <- cbind(densitytable, bestSent)
```

```r
densitytable$bestSent <- as.numeric(factor(densitytable$bestSent,
                 levels = c("SN", "N", "NE", "P", "SP"),
                 ordered = TRUE))
```

Convert sentiment category

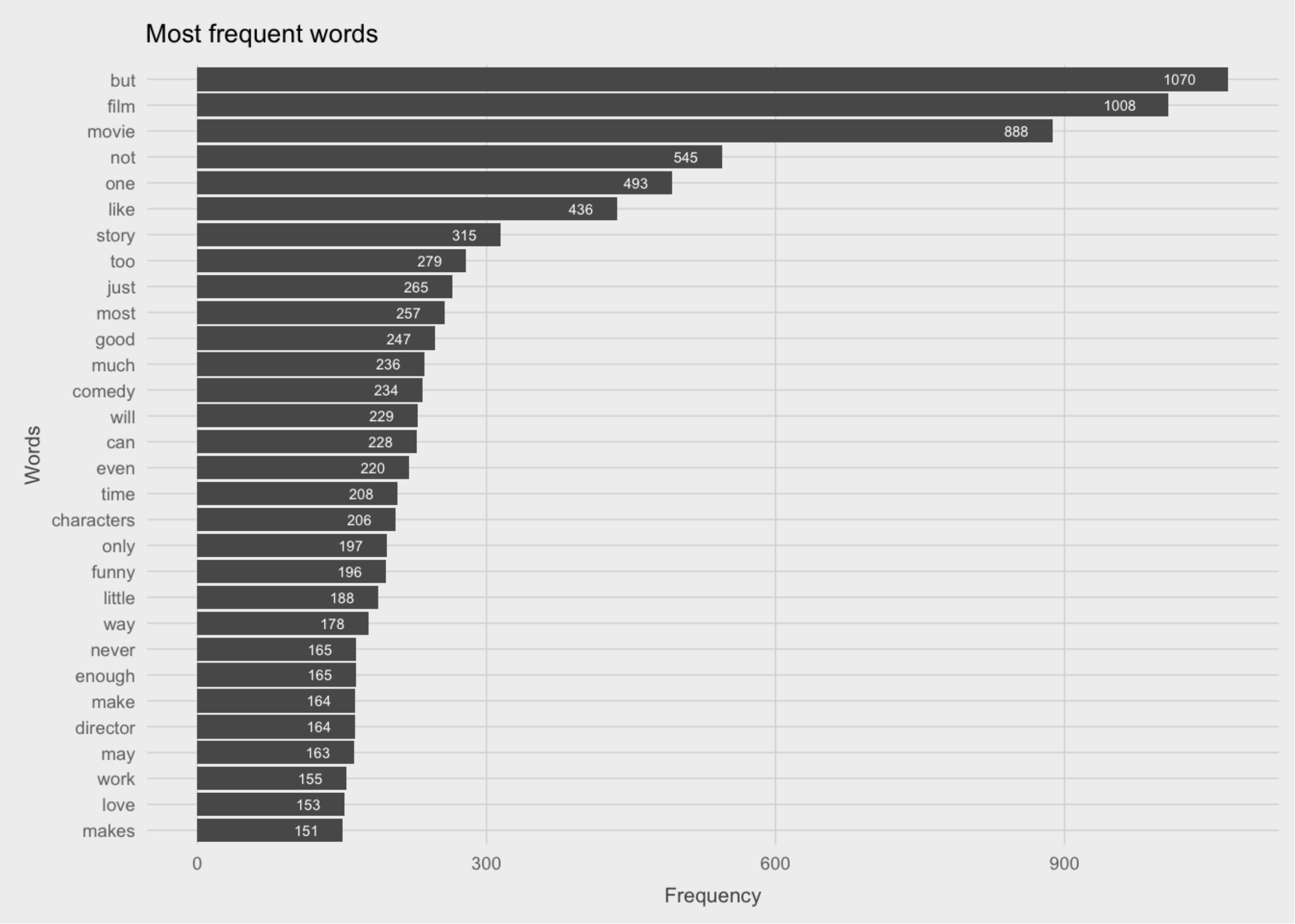
```
new <- densitytable[get("bestSent") == 1, eval("bestSent") := -5]
new <- new[get("bestSent") == 2, eval("bestSent") := -1]
new <- new[get("bestSent") == 3, eval("bestSent") := 0]
new <- new[get("bestSent") == 4, eval("bestSent") := 1]
```

```
table(densitytable$bestSent)
```

```
##
## -5 -1  0  1  5
## 26 45 24 58 47
```

Visualization 30 most frequent words

```
ggplot(data=wordfreq.plot, aes(reorder(word, frequency), frequency)) + geom_bar(stat="identity") + coord_flip() + geom_text(aes(label=round(frequency, 2)), hjust=2, size=2, color="white") + fte_theme() + labs(y="Frequency", x="Words", title="Most frequent words")
```



Remove some words

```
removewords <- c('acting','audience','back','care','come'
                 ,'dialogue','get','goes','making','minutes','movie'
                 ,'script','time','watch','thing', 'action','around'
                 ,'comes','first','gets','going','know','material'
                 ,'place','plays','plot','say','scenes','screenplay'
                 ,'things','think','watching', 'american','bit','can'
                 ,'character','director','film','find','give','gives'
                 ,'keep','kids','made','manages','may','something'
                 ,'sometimes','still','take','takes','two','war','way'
                 ,'will','without','women', 'actors','cast','films'
                 ,'makes','movies','one','piece','screen','see','seen'
                 ,'work','year','years')

new <- new[!new$word %chin% removewords]
```

Prepare data to train model

```
wlist <- data.table(word=new$word, weight=new$bestSent)
# create a word id
wlist[, temp_ord := .I] # maybe you already have some order
wlist[, wid := paste0("w",sprintf("%04d",temp_ord))]
wlist[, temp_ord := NULL]
```

Create phrase-word incidence (frequency) list

```
plistw <- plist[, strsplit(phrase, ' '), by=list(phraseId, sentiment)]
setnames(plistw, "V1", "word")
plistw <- plistw[, list(word_cnt=.N), by=list(phraseId, sentiment, word)]
```

Match list words on the phrase

```
plistw2 <- merge(plistw, wlist, by=c("word")) # inner join
plistw3 <- plistw2[,.(phraseId, sentiment, word_cnt, wid, weight)]
```

```
plistw3 <- plistw3[,.(phraseId, sentiment, wid, weight)]
plistwide <- reshape(plistw3,idvar = c("phraseId", "sentiment"), timevar= "wid",dir
ection = "wide" ,v.names = "weight")
```

```
tn <- names(plistwide)
#tn <- gsub("wcnt.","",tn, fixed= T)
tn <- gsub("weight.","",tn, fixed= T)
names(plistwide) <- tn
for(tcol in names(plistwide)){
  plistwide[is.na(get(tcol)), eval(tcol):=0]
}
```

=================

Model Building

=================

Do the same preparation on 30% data (test set)

Random forest

```
plistwideRF <- rbind(plistwide, plistwide.test)
RFmodel <- randomForest(as.factor(sentiment) ~  .-phraseId , data = plistwideRF)
ptrainvalue <- predict(RFmodel, data=plistwideRF)
```

```
m=as.matrix(table(ptrainvalue,as.factor(plistwideRF$sentiment)))
m
```

```
##            
## ptrainvalue    0    1    2    3    4
##           0  113  111   41   27    8
##           1  456 1039  627  443  176
##           2  356  889 1004  795  305
##           3  244  635  755 1521  776
##           4   23   75   80  192  261
```

```
sum(diag(m)/sum(m))
```

```
## [1] 0.359569
```

SVM (REGRESSION, OVE VS ALL APPROACH)

```
FN_trainSVM <- function(dtin, outcomeVar = "sentiment"){
  prepFormula <- as.formula(paste0(outcomeVar," ~ . -phraseId"))
  model <- svm(prepFormula, data= dtin, type = "eps-regression")
  model
}

FN_train.bestSVM <- function(dtin, outcomeVar = "sentiment"){
  prepFormula <- as.formula(paste0(outcomeVar," ~ . -phraseId"))
  tuneResult <- tune(svm, prepFormula,  data = dtin,
                      ranges = list(epsilon = seq(0,0.2,0.01), cost = 2^(2:9)), sca
le = F)
  tunedModel <- tuneResult$best.model
  tunedModel
}
```

```
# Function evaluates its performance via the confusionMatrix for a given cutoff
FN_evalmodel <- function(tmodel, tsData, whichDigit = workOn, cutoff = 0.5){
  x <- predict(tmodel, tsData)
  x <- as.numeric(x > cutoff)
  y <- confusionMatrix(as.factor(x), as.factor(tsData[,sentiment]))
  y
}
```

Output of SVM Regression with the best cutoff

```
SVMAccuracy
```

```
##                [,1]      [,2]      [,3]      [,4]      [,5]
## Accuracy 0.8969377 0.7537487 0.7499472 0.7292503 0.8648363
```

```
confusionMatrix(as.factor(realSent$final), as.factor(realSent$sentiment))
```

```
## Warning in levels(reference) != levels(data): longer object length is not a
## multiple of shorter object length
```

```
## Warning in confusionMatrix.default(as.factor(realSent$final),
## as.factor(realSent$sentiment)): Levels are not in the same order for
## reference and data. Refactoring data to match.
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1    2    3    4
##          0    3    5    2    2    0
##          1    0    0    0    0    0
##          2  473 1153 1172 1239  575
##          3    0    0    0    3    3
##          4    6    8   10   38   43
## 
## Overall Statistics
##                                           
##                Accuracy : 0.2579          
##                  95% CI : (0.2455, 0.2706)
##     No Information Rate : 0.2707          
##     P-Value [Acc > NIR] : 0.9783          
##                                           
##                   Kappa : 0.0143          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: 0 Class: 1 Class: 2  Class: 3 Class: 4
## Sensitivity          0.0062241   0.0000  0.98986 0.0023401 0.069243
## Specificity          0.9978838   1.0000  0.03126 0.9991312 0.984930
## Pos Pred Value       0.2500000      NaN  0.25412 0.5000000 0.409524
## Neg Pred Value       0.8985814   0.7537  0.90244 0.7295411 0.875162
## Prevalence           0.1017951   0.2463  0.25005 0.2707497 0.131151
## Detection Rate       0.0006336   0.0000  0.24752 0.0006336 0.009081
## Detection Prevalence 0.0025343   0.0000  0.97402 0.0012672 0.022175
## Balanced Accuracy    0.5020540   0.5000  0.51056 0.5007356 0.527086
```

========================

SVM One vs one

========================

Create a function to build model

```r
FN_trainSVMClass <- function(dtin, gamma = 1, cost = 1){
  # SVM
  datain <- copy(dtin)
  datain[, sentiment := as.factor(sentiment)]
  set.seed(2018)
  model <- svm(sentiment ~ . -phraseId, data= datain,
               kernel = "radial",
               gamma = 1, cost =1, scale = F)
  model
}
```

Tried tune but Build a simple model with gamma = 1 and cost = 1

```
FN_trainSVMBestClass <- function(dtin){
  # SVM
  datain <- copy(dtin)
  datain[, sentiment := as.factor(sentiment)]
  set.seed(2018)
  tune.out=tune(svm, sentiment ~ .-phraseId, data=datain, kernel ="radial",
                ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),
                             gamma=c(0.5,1,2,3,4)))

  model.tuned = svm(sentiment ~ . -phraseId, data = datain
                    , kernel ="radial"
                    ,gamma = tune.out$best.parameters$gamma
                    , cost = tune.out$best.parameters$cost)
  model.tuned
}
```

Create a function to test model

```
FN_evalmodelClass <- function(tmodel, datain) {
  tsData <- copy(datain)
  tsData[, sentiment := as.factor(sentiment)]
  x <- predict(tmodel, tsData, type="class")
  y <- confusionMatrix(x, tsData[,sentiment])
  y
}
```

Run model

```
tmodel <- FN_trainSVMClass(plistwide)
# predict on train
```

```
SVMClass.result <- FN_evalmodelClass(tmodel,plistwide.test)
SVMClass.result
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1   2   3   4
##          0  63  52  26   9   4
##          1 186 483 325 178  75
##          2 117 339 395 248  73
##          3 109 266 408 741 326
##          4   7  26  30 106 143
## 
## Overall Statistics
##                                           
##                Accuracy : 0.3854          
##                  95% CI : (0.3715, 0.3995)
##     No Information Rate : 0.2707          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.1866          
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           0.13071   0.4142  0.33361   0.5780  0.23027
## Specificity           0.97860   0.7859  0.78119   0.6788  0.95892
## Pos Pred Value        0.40909   0.3873  0.33703   0.4005  0.45833
## Neg Pred Value        0.90854   0.8042  0.77856   0.8125  0.89193
## Prevalence            0.10180   0.2463  0.25005   0.2707  0.13115
## Detection Rate        0.01331   0.1020  0.08342   0.1565  0.03020
## Detection Prevalence  0.03252   0.2634  0.24752   0.3907  0.06589
## Balanced Accuracy     0.55465   0.6001  0.55740   0.6284  0.59460
```

38.54% - the most reasonable so far