Classification Challenge Code

Miriam A Feldman

29 October 2022

```
library(glmnet)
library(caret)
library(data.table)
library(e1071)
library(mltest)
library(factoextra)
library(dplyr)
library(boot)
library("lightgbm")
library("pROC")
```

Read in Data

```
#Load Training and Test Data
songs_tr <- read.csv('../songs_train.csv')
songs_te <- read.csv('../songs_test.csv')

#Audio Features
songs_tr_sub <- songs_tr[,(colnames(songs_tr) %like% 'audio_')]
songs_te_sub <- songs_te[,(colnames(songs_te) %like% 'audio_')]

#Lyric Features
songs_tr_lyr <- songs_tr[,(colnames(songs_tr) %like% 'lyrics_')]
songs_te_lyr <- songs_te[,(colnames(songs_te) %like% 'lyrics_')]
```

Feature Construction

```
#Create New Base Data Frame
songs_tr_sub_2 <- songs_tr_sub
songs_te_sub_2 <- songs_te_sub

#Total Words
songs_tr_sub_2$totalwords <- rowSums(songs_tr_lyr[,1:1221]) #Training Set
songs_te_sub_2$totalwords <- rowSums(songs_te_lyr[,1:1221]) #Test Set

#Profanity Data Read In
```

```r
lewisprofanity <- read.csv("../writeup/Lewis2014_Profanity", sep = "", header = FALSE)

#remove lyrics_ prefix
colnames(songs_tr_lyr)<-gsub("lyrics_","",colnames(songs_tr_lyr))
colnames(songs_te_lyr)<-gsub("lyrics_","",colnames(songs_te_lyr))

#profanewords: Total number of profane words in the songs
songs_tr_sub_2$profanewords <- rowSums(songs_tr_lyr[, colnames(songs_tr_lyr) %in% lewisprofanity$V1])
songs_te_sub_2$profanewords <- rowSums(songs_te_lyr[, colnames(songs_te_lyr) %in% lewisprofanity$V1])

#profane: Binary Variable (1 if any profanity, 0 if clean)
songs_tr_sub_2$profane <- ifelse(songs_tr_sub_2$profanewords > 0, 1, 0)
songs_te_sub_2$profane <- ifelse(songs_te_sub_2$profanewords > 0, 1, 0)

#profanity_rate: Profane words as proportion of total words in song
songs_tr_sub_2$profanity_rate <- songs_tr_sub_2$profanewords / songs_tr_sub_2$totalwords
songs_te_sub_2$profanity_rate <- songs_te_sub_2$profanewords / songs_te_sub_2$totalwords
```

TF-IDF

Formula Source: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

```r
#Calculate Term Frequencies

#Training data
tfidf_tr_lyr <- as.data.frame(t(songs_tr_lyr[,1:1221]))
tfidf_tr_lyr <- tfidf_tr_lyr %>%
  mutate_if(is.numeric, funs(./sum(.)))
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```r
tfidf_tr_lyr <- as.data.frame(t(tfidf_tr_lyr))

#Test data
tfidf_te_lyr <- as.data.frame(t(songs_te_lyr[,1:1221]))
tfidf_te_lyr <- tfidf_te_lyr %>%
  mutate_if(is.numeric, funs(./sum(.)))
tfidf_te_lyr <- as.data.frame(t(tfidf_te_lyr))

#Calculate Inverse Document Frequencies
```

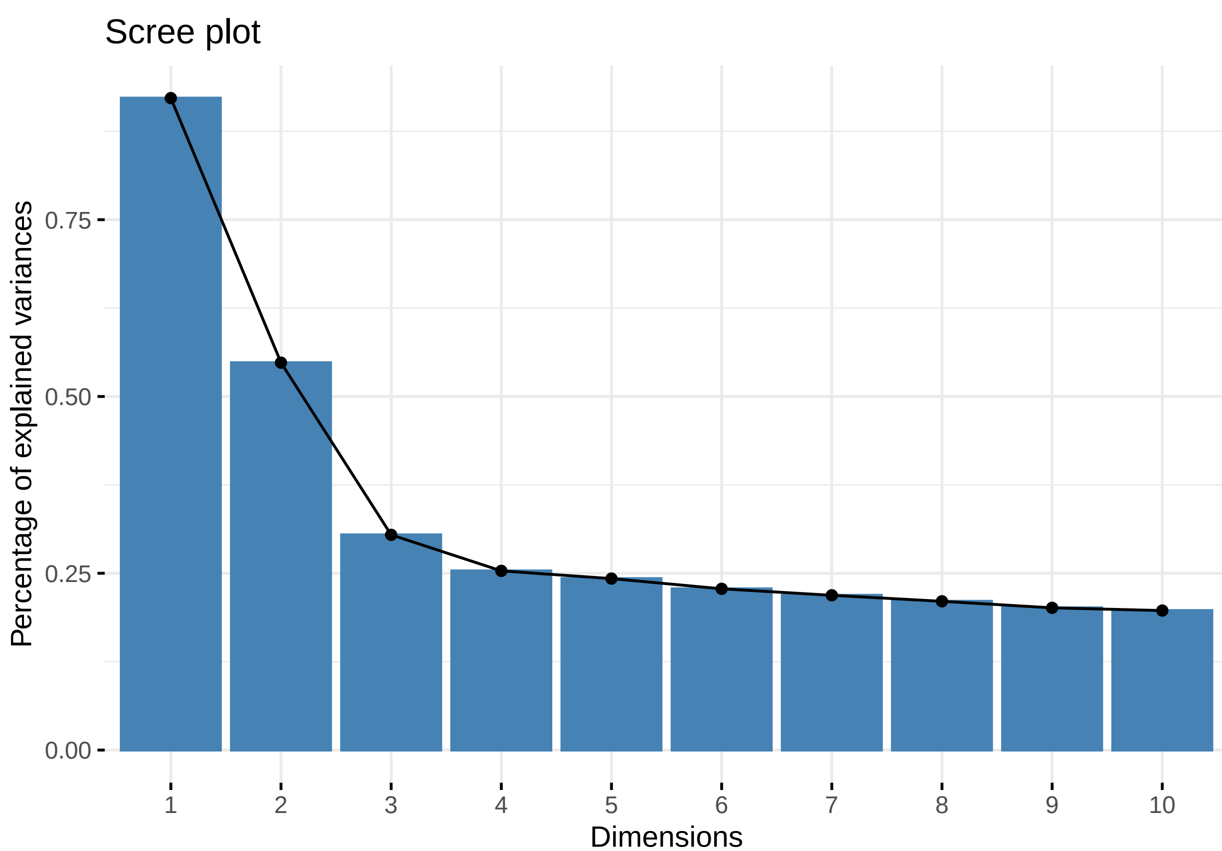
```r
N <- nrow(songs_tr_lyr)
idf <- colSums(songs_tr_lyr[,1:1221] != 0)
idf <- log(N/idf)

#Calculate TF-IDF
tfidf_tr_lyr <- t(t(tfidf_tr_lyr) * idf) #Training Set
tfidf_te_lyr <- t(t(tfidf_te_lyr) * idf) #Test Set
```

PCA 3: On TF-IDF Embedded Lyrics

Sources Referenced: http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/ https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/

```r
#Mitigating Inf, -Inf, NaN, and NAs
tfidf_tr_lyr[!is.finite(tfidf_tr_lyr)] <- 0
tfidf_te_lyr[!is.finite(tfidf_te_lyr)] <- 0
tfidf_tr_lyr[is.na(tfidf_tr_lyr)] <- 0
tfidf_te_lyr[is.na(tfidf_te_lyr)] <- 0
```

```r
#PCA Just on the TF-IDF lyric features
tr_pca_lyr <- prcomp(tfidf_tr_lyr, scale = TRUE)
fviz_eig(tr_pca_lyr)
```



```r
#Take Four Principal Components
new_tr_sub <- tr_pca_lyr$x[,1:4]

#Transform test into PCA
new_te_sub <- predict(tr_pca_lyr, newdata = tfidf_te_lyr)
new_te_sub <- as.data.frame(new_te_sub)
new_te_sub <- new_te_sub[,1:4]

#Combine with the audio and constructed features
new_tr_sub <- as.data.frame(cbind(songs_tr_sub_2, new_tr_sub))
new_te_sub <- cbind(songs_te_sub_2, new_te_sub)

#NAs
new_tr_sub[is.na(new_tr_sub)] <- 0
new_te_sub[is.na(new_te_sub)] <- 0 #Test
```

MODEL 5: LightGBM

```r
#New Data Frame
tr_lgbm <- as.data.frame(cbind(songs_tr_sub_2, tfidf_tr_lyr, genre = songs_tr$genre))
te_lgbm <- as.data.frame(cbind(songs_te_sub_2, tfidf_te_lyr))

#Log Transformations
tr_lgbm$audio_speechiness <- log(tr_lgbm$audio_speechiness)
te_lgbm$audio_speechiness <- log(te_lgbm$audio_speechiness)

tr_lgbm$audio_liveness <- log(tr_lgbm$audio_liveness)
te_lgbm$audio_liveness <- log(te_lgbm$audio_liveness)

tr_lgbm$audio_acousticness <- log(tr_lgbm$audio_acousticness)
te_lgbm$audio_acousticness <- log(te_lgbm$audio_acousticness)
```

```r
 genrefactor <- levels(as.factor(tr_lgbm$genre))
# print(genrefactor)
```

Making LightGBM datasets

```r
set.seed(33200)

# Indices of training and validation observations
all_indices <- 1:nrow(tr_lgbm)
all_indices <- sample(all_indices)

training_indices <- all_indices[1:8000]
validation_indices <- all_indices[8001:10000]
```

```r
# Start Categories from 0
tr_lgbm$genre <- as.numeric(as.factor(tr_lgbm$genre)) - 1
```

```r
# Transforming features and the dataframe
dataset <- lgb.convert_with_rules(data = tr_lgbm)$data %>% as.matrix()

# In dataset with categoricals, add the feature names here
categoricals <- c("audio_key", "audio_mode", "profane")

# Training dataset
training_dataset <- lgb.Dataset(data = dataset[training_indices,-ncol(dataset)],
                                label = dataset[training_indices,ncol(dataset)],
                                categorical_feature = categoricals,
                                params = list(verbose = -1))

# Validation dataset
validation_dataset <- lgb.Dataset.create.valid(dataset = training_dataset,
                                               data = dataset[validation_indices,-ncol(dataset)],
                                               label = dataset[validation_indices,ncol(dataset)],
                                               params = list(verbose = -1))
```

Random Hyperparameter Search for LightGBM

Source: Week 9 MY474

```r
random_search_lgbm <- function(parameter_values_fixed,
                              n_draws,
                              training_dataset,
                              seed_int) {

  #
  #
  # Inputs
  #
  # - parameter_values_fixed: A dictionary of fixed lightgbm parameters
  # - n_draws: The amount of random parameter combinations to try
  # - training_dataset: A lightgbm dataset
  # - seed_int: An integer to set a pseudo random number seed
  #
  #
  # Output
  #
  # - A data.frame starting with the lowest CV loss that was found
  #
  #


  ## 1. Status update

  print("Starting the random hyper-parameter search ..")


  ## 2. Create a dataframe to store the outcomes

  search_output <- data.frame(learning_rate = numeric(),
```

```r
                              num_leaves = numeric(),
                              max_depth = numeric(),
                              feature_fraction = numeric(),
                              bagging_fraction = numeric(),
                              is_unbalance = logical(),
                              score = numeric())


## 3. Training the models

# Set seed
set.seed(seed_int)

# Iterations
for (ii in 1:n_draws) {

  # Picking a random point in the hyper-parameter space and storing it in the
  # search output dataframe

  # Runs from ca. 0.01 to 0.5, with smaller values being more likely
  search_output[ii, "learning_rate"] <- exp(runif(1, min = -4.6, max = -0.7))

  search_output[ii, "num_leaves"] <- round(runif(1, min = 2, max = 200))
  #lower max to prevent overfitting:
  search_output[ii, "max_depth"] <- round(runif(1, min = 1, max = 10))
  search_output[ii, "feature_fraction"] <- runif(1, min = 0, max = 1)
  search_output[ii, "bagging_fraction"] <- runif(1, min = 0, max = 1)
  search_output[ii, "is_unbalance"] <- sample(c(TRUE, FALSE), 1)


  # Transforming the parameter values into a list
  parameter_values_variable <- as.list(search_output[ii, !(colnames(search_output) %in% "score")])

  # Combining with fixed parameters and seed
  current_params <- c(parameter_values_fixed, parameter_values_variable, list(seed = seed_int))

  # Cross validation
  sink("/dev/null")
  current_cv <- lgb.cv(
    params = current_params,
    data = training_dataset,
    nrounds = 100,
    nfold = 5,
    verbose = -1
  )
  sink()

  # Storing the score in the final boosting round
  search_output[ii,"score"] <- current_cv$record_evals$valid$multi_logloss$eval[[100]]

  # Status update
  if (ii %% 25 == 0) {
```

```
      print(sprintf("%s/%s models trained.", ii, n_draws))

    }

  }


  # Sorting to obtain parameter combination with the best score first
  search_output <- search_output[order(search_output$score),]  #Ascending


  return(search_output)

}
```

MODEL 5B: Random Hyperparameter Search (No PCA)

```
# parameter_values_fixed <- list(objective = "multiclass",
#                                  metric = "multi_logloss",
#                                  num_class = 4)
#
# random_search_output <- random_search_lgbm(parameter_values_fixed = parameter_values_fixed,
#                                              n_draws = 200,
#                                              training_dataset = training_dataset,
#                                              seed_int = 33200)
```

```
# head(random_search_output, 5)
# write.csv(random_search_output, 'random_search_out_nopca.csv', row.names=FALSE)
```

```
best_params_nopca <- list(objective = "multiclass",
                    metric = "multi_logloss",
                    num_class = 4,
                    learning_rate = 0.07631250  ,
                    num_leaves = 57,
                    max_depth = 7,
                    feature_fraction = 0.6513432,
                    bagging_fraction = 0.8861957,
                    is_unbalance = FALSE,
                    early_stopping = 50)


model_nopca <- lgb.train(
  params = best_params_nopca,
  data = training_dataset,
  nrounds = 10000, # note: needs to be larger for very small learning rates
  valids = list(training = training_dataset, validation = validation_dataset),
  verbose = -1)

# Test set predictions
test_pred <- predict(model_nopca, as.matrix(te_lgbm), reshape=T)
pred_test_y = max.col(test_pred)
```

```r
# print(genrefactor)

# Output answers for submission to Kaggle
answers <- data.frame(10001:(10000+nrow(te_lgbm)), pred_test_y)
colnames(answers) <- c('song_id', 'genrefac')

answers$genre <- genrefactor[answers$genre]
answers$genrefac <- NULL

#write.csv(answers, 'answers10.csv', row.names=FALSE)


# Validation set
val_X <- dataset[validation_indices,-ncol(dataset)]
val_y <- dataset[validation_indices,ncol(dataset)]
val_pred <- predict(model_nopca, val_X, reshape=T)
pred_val_y = max.col(val_pred)-1

confusionMatrix(as.factor(val_y), as.factor(pred_val_y))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3
##          0 249  34 129  21
##          1  25 378  37  75
##          2 132  65 313   7
##          3  15  75   2 443
##
## Overall Statistics
##
##                Accuracy : 0.6915
##                  95% CI : (0.6707, 0.7117)
##     No Information Rate : 0.276
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5877
##
##  Mcnemar's Test P-Value : 0.04512
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3
## Sensitivity            0.5914   0.6848   0.6507   0.8114
## Specificity            0.8835   0.9054   0.8657   0.9367
## Pos Pred Value         0.5751   0.7340   0.6054   0.8280
## Neg Pred Value         0.8902   0.8828   0.8867   0.9297
## Prevalence             0.2105   0.2760   0.2405   0.2730
## Detection Rate         0.1245   0.1890   0.1565   0.2215
## Detection Prevalence   0.2165   0.2575   0.2585   0.2675
## Balanced Accuracy      0.7375   0.7951   0.7582   0.8740
```

```r
ml_test(pred_val_y, val_y)$F1 # Validation Set
```

```
##         0         1         2         3
## 0.5831382 0.7085286 0.6272545 0.8196115
```

Test F1: 0.68422