David Smith
STAT 4630
Final Exam

1.

a.

In order to study the effect of the number of random projections on the accuracy of the distance, I used a simple nested for loop to iterate over six different values of p, and for each, calculated the reduced-dimension matrix 1000 times with different random matrices W, and evaluated a statistic which compares the average of the distances between each pair of observations of the reduced matrix to the average of each pair of observations in the original matrix. I decided not to use a bootstrap or training/test data technique for two major reasons. For one, we are using a fairly large randomly-generated data set, and thus results should be generally valid, especially since we are calculating 1000 output matrices using different random matrices W for each value of p. It is also better to use more values when feasible, instead of only looking at a subset of the data, as this leads to more accurate calculations. Also, we are only trying to compare obtained results to each other, and so do not need to know the exact uncertainty of our estimates. We are not really studying a population, but only the effect of a change of variable on our technique. Thus, using the entire data set to obtain results and comparing them to each other for different values of p seems to be the best option.

I chose to use 500 observations, as this number seems sufficient, and any significantly higher number slowed down the calculations too much. For the six values of p, which are the dimensions of the reduced observations, I chose to examine very small numbers up to ones very close to d, the dimension of the original observations. Specifically, I looked at p equal to 5, 20, 40, 60, 80, and 95, with d being equal to 100. For each value of p, I did the following 1000 times: I calculated the random matrix W, multiplied it by the original matrix to obtain the reduced-dimension matrix, and took the quotient of the mean distance between each pair of observations of the reduced matrix to the mean distance between each pair of observations of the original matrix. Then, having done this 1000 times for each value of p, I calculated the mean of the 1000 values to obtain six resulting statistics. The closer these numbers are to 1, the more accurate our random projection is. My results

are as follows: We have 0.9476 for p = 5, 0.9830 for p = 20, 0.9924 for p = 40, 0.9973 for p = 60, 0.9975 for p = 80, and 0.9981 for p = 95. We see that the accuracy is quite high for all values of p. The most significant change is from p = 5 to p = 20, with our statistic increasing from 94.76% to 98.3%. The accuracy seems to level out after around p = 40, with all values higher than 99%, but continuing to increase slightly as p grows. We conclude that random projections are accurate even when the desired dimension to reduce to is relatively small, but works better for reduced dimensions that are not very small.

b.

For this task, I followed pretty much the same procedure as in part a. I again used a nested for loop, this time iterating over six different values of d, all between 50 and 250 as suggested in the outline, with p fixed at 50. The main difference is that for each value of d, I first chopped off part of the original observation matrix X to match the required dimension d. I chose not to create a new random matrix for each different dimension, as my goal is to study only the effect of changing d on the accuracy of the random projection, and thus I wanted to keep as many other factors constant as possible. After calculating the accuracy statistic 1000 times for each value of d, I again took the mean of each to obtain six values, which are as follows: We have 0.9952 for d = 250, 0.9950 for d = 150, 0.9971 for d = 100, 0.9950 for d = 80, 0.9947 for d = 60, and 0.9965 for d = 55. We see that changing the value of d does not affect the accuracy of the random projections. The accuracy statistics all fluctuate between 99% and 100%, and do not steadily increase or steadily decrease, but instead seem to be shifting slightly mainly due to the randomness of the procedure. Thus, it seems that having very large values of the dimension of the original observations compared to values that are very close to the dimension of the reduced observations does not seem to make a difference. We conclude that the accuracy of the random projection technique is high, except when p is very small.

c.

The random projection technique seems to be useful for reducing the dimension of a data set when we are okay with settling for a reduced dimension that is not extremely small. It is likely the most useful when the distance between observations is important, as the goal of using random projections is to preserve this value. It seems to be a simple and fast technique, and might be

preferable to more complex algorithms such as PCA in instances where we want a quick and easy dimension-reduction solution.

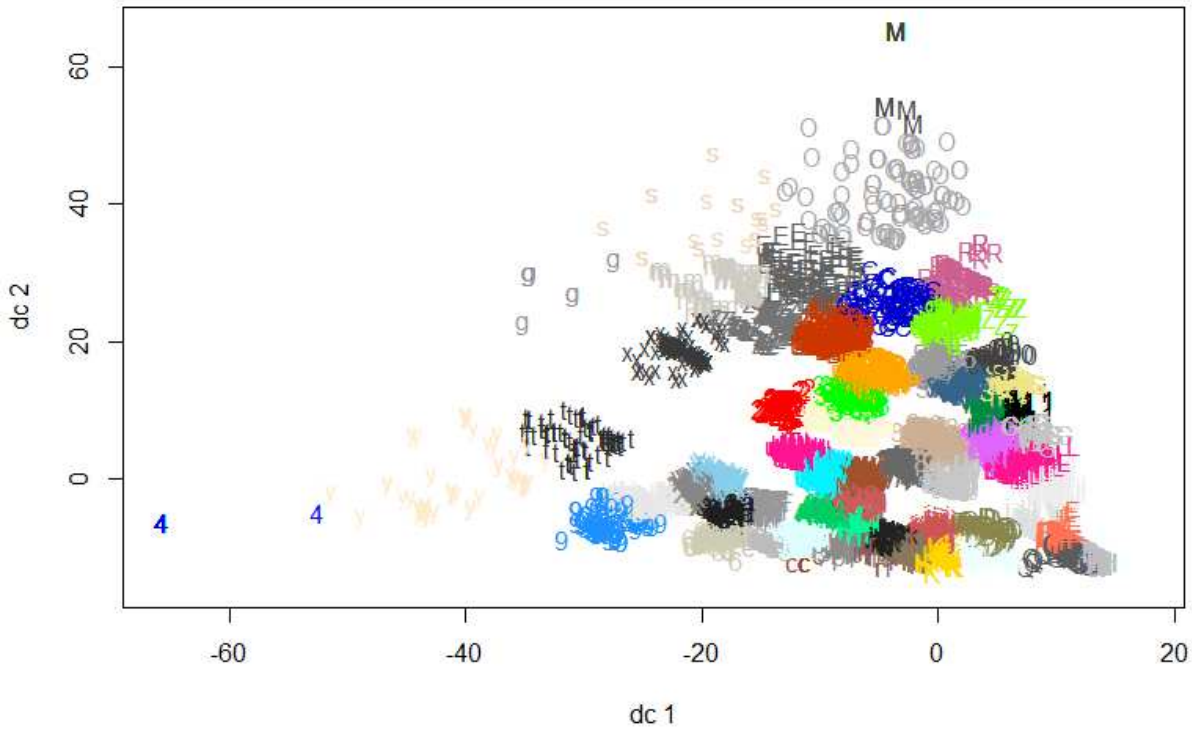
2.

For this task, we aim to group actors and actresses together based on movie genres they star in, possibly to help actors determine who they should work with on future projects. Thus, k-means and hierarchal clustering will be the ideal algorithms to try. We choose k-means clustering over hierarchal clustering as viewing the dendrogram would be very difficult due to the size and complexity of the dataset, and we would not know where to make the cut. Since we have 20 different genres, we will use PCA to represent the data by its first couple of principal components, making it less noisy.

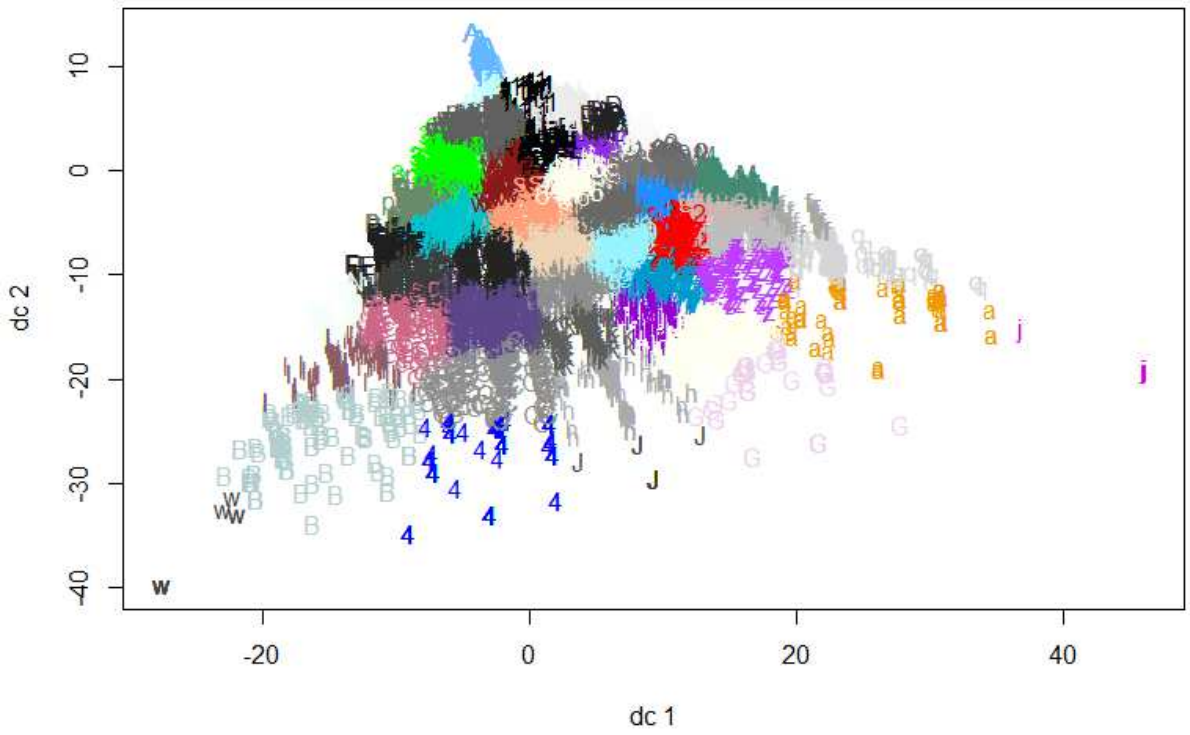
There seems to be only one way to categorize actors based on genres of movies they have starred in: we look at the number of movies of each genre they acted in. However, one important point to note is that we don't want actors to be separated into different groups simply because one starred in more movies than another. In other words, if one actor starred primarily in science fiction films, but only acted in five of them, and another actor also starred primarily in science fiction films, but acted in 20 of them, we want these actors to ideally be grouped together, despite the vast difference in the numbers. Additionally, we have that many movies fall into multiple genres, so that if we count the genres that actors starred in after merging the two datasets, we will have a higher total number than the number of movies they starred in. Of course, the number of genres each actor starred in is our variable of interest, but we don't want the fact that different movies have more genres than others to affect our results. Our solution to both of these problems is as follows: after counting the number of genres of movies that each actor acted in, we divide each number by the total number of genres they acted in. This scales our values, so the number of movies an actor acted in, and more specifically, the number of genres they acted in, does not affect how they are paired into clusters with other actors.

A key issue is how to determine the number of clusters to use. We want something not too small, but not too large. Counting the number of actors, we find it to be 54,562, so a number of clusters

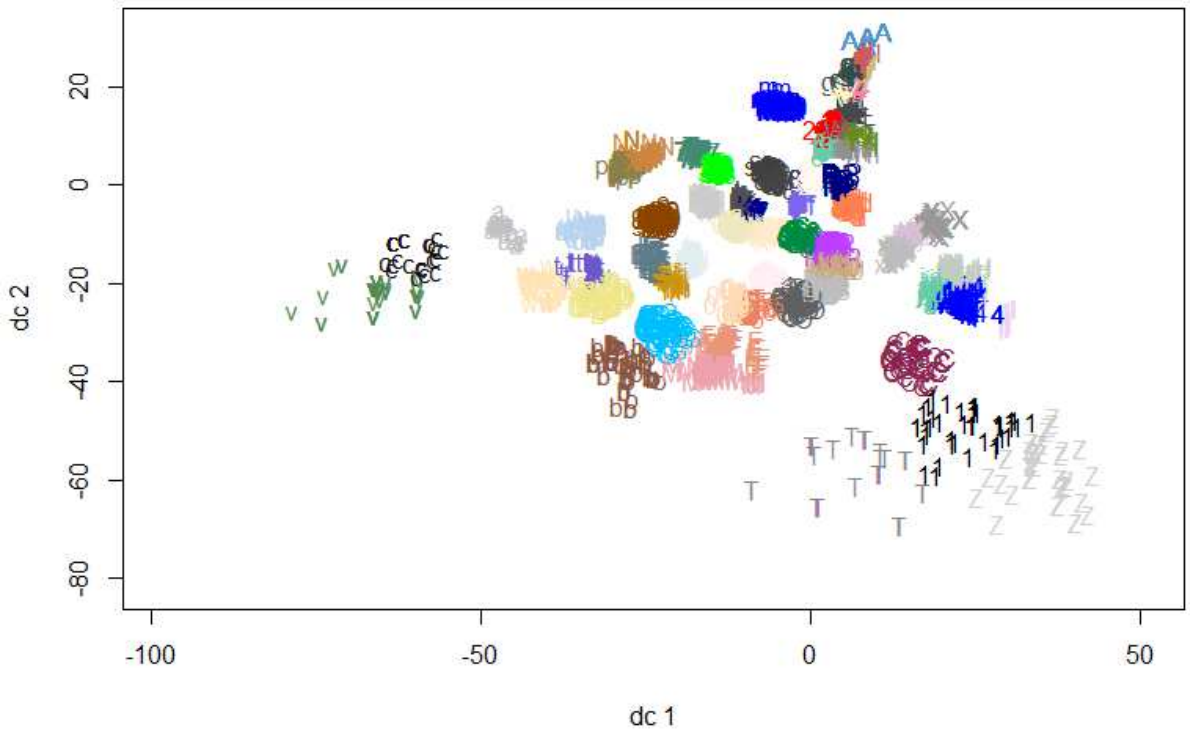
around 50-200 seems reasonable, which gives average cluster sizes of around 270 to 1090. I decide to try using 100, 50, and 200 clusters, and compare the plots and within-cluster sum of squares for each. Upon trying 100 clusters, we get the following plot of the first two principal components:



We see that many of the clusters do appear well-separated, but some overlap appears to exist. Also, some of the outlying clusters are spread very wide. The total within-cluster sum of squares is about 24.280. We now try using 50 clusters:



This appears to be worse, as the clusters are very tightly packed, and are spread out near the edge. The value of the total within-clusters sum of squares confirms this, as it is about 49.068, much higher than the value of 24.280 for 100 clusters. Thus, using only 50 clusters seems to be too small of a number. Finally, we compare these results with using 200 clusters. The plot of the first two principal components is as follows:



We see that many of the clusters are very well-distinguished, but some of them are tightly packed together. Also, we see that some of the clusters are a bit spread out, but this problem seems improved from before. The total within-cluster sum of squares for this model is about 10.877, the best value yet. Seeing as these clusters appear well-separated for the most part and aren't too spread out, with the sum of squares being much smaller than for the other two models, we decide to use 200 clusters of actors. Using more seems like a bad idea, as 200 is already a large number, and we don't want too many clusters with only a few actors.

Looking at actor 1, we see that they were placed into cluster 101, which contains 239 actors. This actor starred three times in genre 12, three times in genre 28, two times in genre 35, and one time in five other genres. Presumably, they were placed into a cluster of actors with a fairly similar distribution of genres for movies they have starred in.

We conclude that using k-means clustering to group together actors based on genres of movies they have starred in may be useful for categorizing them. 200 seems to be a good number of

clusters to use, as evidenced by our analysis above. However, we want to avoid attaching any interpretation or meaning to these clusters, as clustering is an inherently volatile technique. Clusters will be found, regardless of the situation, and there is no way to confirm that these clusters are actually useful—all we can do is try to minimize the within-cluster sum of squares and hope for the best. By refraining from trying to interpret the clusters, we keep them in the correct context. In our case, they are simply groups of actors that have starred in roughly similar proportions of the same genres of movies. This could indeed be useful to us, for example by finding actors that may work well together on future projects, so clustering can be an appropriate technique in this instance.

3.

The goal of this task is to develop an algorithm to predict the genre of a movie based on its title, actors, and director. We are dealing with a classification problem, and so we will be limited to certain options for our algorithm, namely logistic regression, LDA, QDA, tree-based methods, and support vector machines. We will see that some of these can be ruled out as inappropriate or unfeasible. Our plan will be to set up our data, split it into a training and test subset, train each algorithm on the training subset and predict for the test subject, and evaluate the misclassification rate for each, preferring the algorithm that gives the lowest test error. The context of our classification problem doesn't seem to indicate preference for any particular algorithm, so we will try all reasonable possibilities.

First, in setting up the data, we have movie title, actors, and directors to predict genre. We observe that although these predictors should be viewed as categorical, they have very high levels when converted to factor variables, and none of the above algorithms can deal with categorical data of this magnitude. Thus, we are forced to use these variables as numeric data. To accomplish this, we use the ID of each, and trim away the extraneous information. However, the ID's are somewhat random, and some values are much larger than others. To fix this, we relabel these ID's to consecutive integers starting with 1, while preserving their structure. Then, we take random training and test subsets of the data, with the test subset consisting of 10% of the data. We are now ready to examine which algorithms are feasible to perform.

Logistic regression and boosting are both inappropriate, as we have 20 possible genres that we are attempting to predict, and these methods, at least in the form presented in class and in the textbook, can only deal with binary classification problems. LDA and QDA can both be used, along with regular classification trees and bagging/random forests. I attempted to use a support vector machine, but it was computationally infeasible for this data set even without using cross-validation for the parameters, and so I was forced to omit it.

Proceeding with LDA, we obtain a somewhat surprising result: all observations in the test data set are classified as genre 18, 28, or 35. The misclassification rate is 80.94%, indicating that at least 20% of all of the observations in the test set belong to one of these three genres out of 20 possibilities. QDA does better, predicting between five different genres for all observations, with a misclassification rate of 80.72%. Turning to tree-based methods, a regular classification tree pruned at an optimal level of size 2 (according to cross-validation) gives every observation being classified as either genre 18 or 28, with a misclassification rate of 81.48%. For bagging and random forests we use 100 trees, as anything too much higher becomes computationally unfeasible. With bagging, using m equal to 3, the number of predictors, we find that many different classifications of genres are produced; we are no longer limited to predicting between a few different genres. However, the misclassification rate is now much worse, at 92.04%. Using a random forest with m equal to 2 again gives many different predicted genres, with a slightly lower misclassification rate of 89.16%. We conclude that LDA, QDA, and a pruned classification tree all gave the most accurate predictions for the test subset, with QDA coming out on top with the lowest misclassification rate. Thus, we choose to use QDA to predict movie genre. This algorithm, while our best option, is still performing quite poorly, classifying correctly only 20% of the time.

Looking at our data, the reason for our failure seems clear: we are attempting to predict between 20 different movie genres with only three pieces of information: the name of the movie, the actors starring in it, and the director. Considering this, the fact that we classified correctly 20% of the time in the best case is actually quite astounding. This is partially due to an anomaly in the data; the fact that at least 20% of our test subset genre labels fell into a small number of genre categories. Thus, these categories dominate, which explains why the best algorithms opted to predict each observation as one of these few. We can conclude that in order to better predict movie genres, we

require more information: having only the movie title, actors, and director is probably not enough. However, in the absence of additional data with which to refine our methods, we will choose to work with QDA for now.

CODE:

```
## Problem 1


## Part a

# Get the data
X <- simulate_dataset(100, 500)

# Vector of values for p
p <- c(5, 20, 40, 60, 80, 95)

out <- numeric()
result <- numeric()

for (i in 1:6) {
  for (j in 1:1000) {
    # Generate the random matrix W
    W <- (1/sqrt(p[i])) * matrix(rnorm(100*p[i]), nrow=100)

    # Calculate the reduced-dimensional observation matrix
    Y <- X%*%W

    # Return the distance statistic
```

```
    out[j] <- mean(dist(Y))/mean(dist(X))
  }

  result[i] <- mean(out)
}

result

# We have 0.9476 for p = 5, 0.9830 for p = 20, 0.9924 for p = 40,
# 0.9973 for p = 60, 0.9975 for p = 80, and 0.9981 for p = 95.



## Part b

# Simulate the data set
X <- simulate_dataset(250, 500)

# Vector of values for d
d <- c(250, 150, 100, 80, 60, 55)

out <- numeric()
result <- numeric()

for (i in 1:6) {
  # Adjust the original data
  X <- X[,1:d[i]]

  for (j in 1:1000) {
    # Generate the random matrix W
    W <- (1/sqrt(50)) * matrix(rnorm(d[i]*50), nrow=d[i])
```

```
    # Calculate the reduced-dimensional observation matrix
    Y <- X%*%W

    # Return the distance statistic
    out[j] <- mean(dist(Y))/mean(dist(X))
  }

  result[i] <- mean(out)
}

result

# We have 0.9952 for d = 250, 0.9950 for d = 150, 0.9971 for d = 100,
# 0.9950 for d = 80, 0.9947 for d = 60, and 0.9965 for d = 55.



## Problem 2

library(tidyverse)

load("acting.Rdata")
load("genres.Rdata")

# Merge the data by movie title
data <- acting %>% rename(actor=name, act_id=id, movie_title=title, id=movie_id) %>%
inner_join(genres_readable)

# Get the number of actors
n <- length(unique(data$act_id))
```

```
# Get the number of movies of each genre that each actor acted in, with the genres forming the columns
# and the actors the rows
data_counts <- group_by(data, act_id, id1) %>% summarise(count=n()) %>% spread(id1, count)
data_counts[is.na(data_counts)] <- 0
data_counts <- data_counts[,2:21]

# Divide each element by the sum of its row
data_counts <- t(apply(t(data_counts), 2, function(x) x/sum(x)))

# Run PCA
data_counts_pca <- prcomp(data_counts)

# Run k-means with 100 clusters on the first two principal components of the data
data_counts_kmeans <- kmeans(data_counts_pca$x[,1:2], 100, nstart=20)

# Plot the clusters on the first two principal components
library(fpc)
plotcluster(data_counts, data_counts_kmeans$cluster)

# Get the total within-cluster sum of squares
data_counts_kmeans$tot.withinss

# 24.280

# Run k-means with 50 clusters on the first two principal components of the data
data_counts_kmeans <- kmeans(data_counts_pca$x[,1:2], 50, nstart=20)

# Plot the clusters on the first two principal components
plotcluster(data_counts, data_counts_kmeans$cluster)
```

```
# Get the total within-cluster sum of squares
data_counts_kmeans$tot.withinss

# 49.068

# Run k-means with 200 clusters on the first two principal components of the data
data_counts_kmeans <- kmeans(data_counts_pca$x[,1:2], 200, nstart=20)

# Plot the clusters on the first two principal components
plotcluster(data_counts, data_counts_kmeans$cluster)

# Get the total within-cluster sum of squares
data_counts_kmeans$tot.withinss

# 10.877

# Look at actor 1
data_counts_kmeans$cluster
head(group_by(data, act_id, id1) %>% summarise(count=n()) %>% spread(id1, count))
length(data_counts_kmeans$cluster[data_counts_kmeans$cluster == 101])

# Actor 1 is in cluster 101, which contains 239 actors.


## Problem 3


load("directors.Rdata")

# Trim extraneous data
acting <- acting[,c(1,3)]
```

```
genres <- genres_readable[,c(1,3)]
directors <- directors[c(1,3)]

# Rename the variables for convenience
acting <- acting %>% rename(actor_id=id)
genres <- genres %>% rename(movie_id=id, genre_id=id1)
directors <- directors %>% rename(director_id=id)

# Join the three data sets
data <- inner_join(acting, genres)
data <- inner_join(data, directors)

# Convert the response to a categorical variable
data$genre_id <- as.factor(data$genre_id)

# Reassign values
j <- 1
for (i in unique(data$movie_id)) {
  data$movie_id[data$movie_id == i] <- j
  j <- j + 1
}

k <- 1
for (i in unique(data$actor_id)) {
  data$actor_id[data$actor_id == i] <- k
  k <- k + 1
}

l <- 1
for (i in unique(data$director_id)) {
  data$director_id[data$director_id == i] <- l
```

```
  l <- l + 1
}

# Split into training and test subsets
samp <- sample(nrow(data), floor(nrow(data)*0.9))

data_train <- data[samp,]
data_test <- data[-samp,]

# Try LDA
library(MASS)

lda_fit <- lda(genre_id ~ ., data=data_train)
lda_pred <- predict(lda_fit, newdata=dplyr::select(data_test, -genre_id))

lda_pred$class
length(unique(lda_pred$class))

# LDA classified everything as genre 18, 28, or 35

accuracy <- table(lda_pred$class, data_test$genre_id)
1 - (sum(diag(accuracy))/sum(accuracy))

# The misclassification rate is 0.8094

# Try QDA
qda_fit <- qda(genre_id ~ ., data=data_train)
qda_pred <- predict(qda_fit, newdata=dplyr::select(data_test, -genre_id))

qda_pred$class
length(unique(qda_pred$class))
```

```
# QDA classified in five different genres

accuracy <- table(qda_pred$class, data_test$genre_id)
1 - (sum(diag(accuracy))/sum(accuracy))

# The misclassification rate is 0.8072

# Try pruned classification tree
library(tree)

tree_fit <- tree(genre_id ~ ., data=data_train)

# Use cross-validation to determine the optimal size of the tree
tree_cv <- cv.tree(tree_fit)

tree_cv$size
tree_cv$dev

# A size of 2 is best

# Prune the tree
tree_prune <- prune.misclass(tree_fit, best=2)

tree_pred <- predict(tree_prune, newdata=data_test, type="class")
tree_pred

length(unique(tree_pred))

# Using a pruned tree results in everything being classified as genre
# 18 or 28
```

```
accuracy <- table(tree_pred, data_test$genre_id)
1 - (sum(diag(accuracy))/sum(accuracy))

# The misclassification rate is 0.8148

# Try bagging and random forests
library(randomForest)

bag_fit <- randomForest(genre_id ~ ., data=data_train, mtry=3, ntree=100)
bag_pred <- predict(bag_fit, newdata=data_test)

accuracy <- table(bag_pred, data_test$genre_id)
1 - (sum(diag(accuracy))/sum(accuracy))

# More genres were predicted, but the misclassification rate is worse, at 0.9204

forest_fit <- randomForest(genre_id ~ ., data=data_train, mtry=2, ntree=100)
forest_pred <- predict(forest_fit, newdata=data_test)

accuracy <- table(forest_pred, data_test$genre_id)
1 - (sum(diag(accuracy))/sum(accuracy))

# The misclassification rate is slightly better, at 0.8916, but still worse
# than other methods
```