Project 1

STAT 437 Spring 2022

Christopher Mims
(10436827)

April 07, 2022

Introduction

This project will consist of two main parts, clustering and classification. Within the clustering section, our knowledge of K-means clustering and hierarchical clustering will be assessed, and within the classification section, our knowledge of Nearest-neighbor classifier and discriminant analysis for classification will be assessed. The visualizations we have learned will be shown throughout the project by our illustrations of the findings we uncover. The data set that we will be using contains over 800 observations of five types of cancer, with over 20,000 features.

The clustering section of this project will explore the use of K-means clustering and hierarchical clustering using varying amounts of gene expressions. This will explore the advantages and complications of adding additional features into such models. Use of the `clausGap` function will be used to estimate the amount of clusters within the data set and then its results will be compared to those we calculate using the total within-cluster sum of squares scores. This comparison will allow us to identify the best situations to use either such function to estimate clusters. Within hierarchical clustering, we will explore the accuracy of the clustering algorithm with each type of distance measure; average, single, or complete. Again, this will challenge us to recognize when it is best to use each such use case.

The classification section will focus on the use of the quadratic discriminate analysis (QDA) and k-nearest neighbors algorithms. With the use of the QDA model, we will reinforce the steps needed to ensure that the model can be appropriately used through tests that satisfy the Gaussian mixture of the samples and the removal of any highly correlated gene expression types. The Guassian mixture model will be tested through the use of Guassian density plots. In order to proceed, these plots will need to conform to normal Gaussian distributions. Accuracy of the model will be evaluated by the use of a 2-by-2 table showing the 0-1 loss. After building, training, testing and evaluating the QDA Model, we will switch our focus to a k-nearest neighbors model. The accuracy of this model will be evaluated and compared to the results of the QDA model. Using these two classification techniques will provide us with knowledge of when each model is best suited for different situations.

Methods and Results

The data set we will be using for this project is a collection of 801 samples of 20531 gene expression values, representing five (5) cancer types. These cancer types are represented by the labels “BRCA”, “KIRC”, “COAD”, “LAUD”, and “PRAD”. The data can be found here as a compressed gzip (‘tar.gz’) file. The two files contained in the compressed file represent the gene expression data and the cancer type labels. Each sample is labeled as ‘sample_x’, where ‘x’ is the number of the sample.

After downloading and decompressing the files, we can read in the data and labels. This will give us access to the data and its labels in tables, which can then be used to filter the data. We must ensure that the data is

read in properly to keep header labels and row names from becoming part of the data. The initial tables will then be filtered for use in other parts of the project.

Clustering

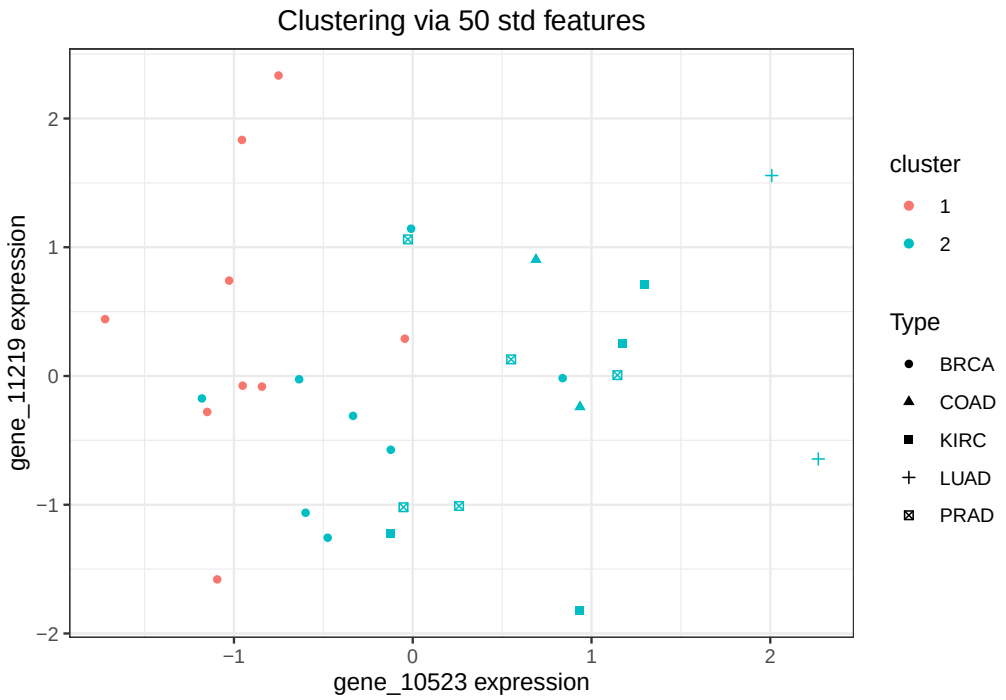
In this first section of the project, we will be implementing k-means and hierarchical clustering to divide the observations (samples) into their associated cancer types. We will then report our findings scientifically and professionally. In order to keep computational cost low and ensure our machine can process the clustering algorithm without issue, the data will be filtered to remove features that have a high instance of the value zero within their sample sets. This can be done due to the fact that when a cancer's gene expression consists of zeros for most subjects, it may have little informative value for a cancer type. As well as reduce computational cost.

The use of `set.seed(123)` for random sampling via the command `sample`, random initialization of `kmeans`, implementing the gap statistic, and any other process where artificial randomization is needed will take place. First, we will create a few objects that will be used in other parts of the project. The first object we will create will be `gexp2`, which will consist of the data where any genes whose expressions are zero for at least 300 subjects are removed. Then `gexp3` will be a random sample of 1000 genes and their expressions from `gexp2`. Next we will create an R object `labels1` as 30 randomly selected samples and R object `gexpProj1` as the corresponding samples from `gexp3`. The use of R objects here will decrease the amount of computational power and time for future processing of the code. Finally, we will scale the gene expression data in `gexpProj1` to have a sample mean of zero and a sample standard deviation of 1 for each gene expression, and saving as an R object `stdgexpProj1`.

K-means Clustering From Scaled Data

For this part of the project we will make use of the k-means clustering algorithm. We will be using 50 randomly picked samples of the scaled data we created above and perform the following tasks:

- Apply the "gap statistic" to estimate the number of clusters
  - Arguments `K.max = 10`, `B = 200`, and `iter.max = 100` must be used in the implementation of `clusGap`
- Apply the k-means clustering algorithm with number of clusters estimated by the gap statistic
  - Arguments `iter.max = 100`, `nstart = 25`, and `algorithm = c('Hartigan-Wong')` must be used in the implementation of `kmeans`
- Visualize the classification results using the techniques learned in 'LectureNotes3_notes.pdf'
- Provide a summary on the classification errors



Using the gap statistic `clausGap`, we end up with 2 clusters. Therefore, all samples that are not 'BRCA' or 'COAD' are misclassified. From this visual you can see that all samples have been classified into 2 clusters, with no other clusters available, while all five (5) types of cancer are present. Below you will find a table of the counts of the cancer types that were present in the data along with the counts of the types assigned during clustering, along with the count of errors within the clustering for each cancer type.

```
##   Cancer Type Actual Assigned Errors
## 1        BRCA     17        9      8
## 2        COAD      2       21     19
## 3        KIRC      4        0      4
## 4        LUAD      2        0      2
## 5        PRAD      5        0      5
```

Total Within-Cluster Sum of Squares

When performing k-means clustering, one way of determining the amount of clusters to use is by obtaining the total within-cluster sum of squares (twcss), $W(k)$, for a series of k in N, where $k = 1, 2, 3, \ldots, n$ is the number of clusters used in `kmeans`. Then the difference $\Delta_k = W(k) - W(k+1)$ for k ranging from 1 to k_n, is calculated. The K^* for which

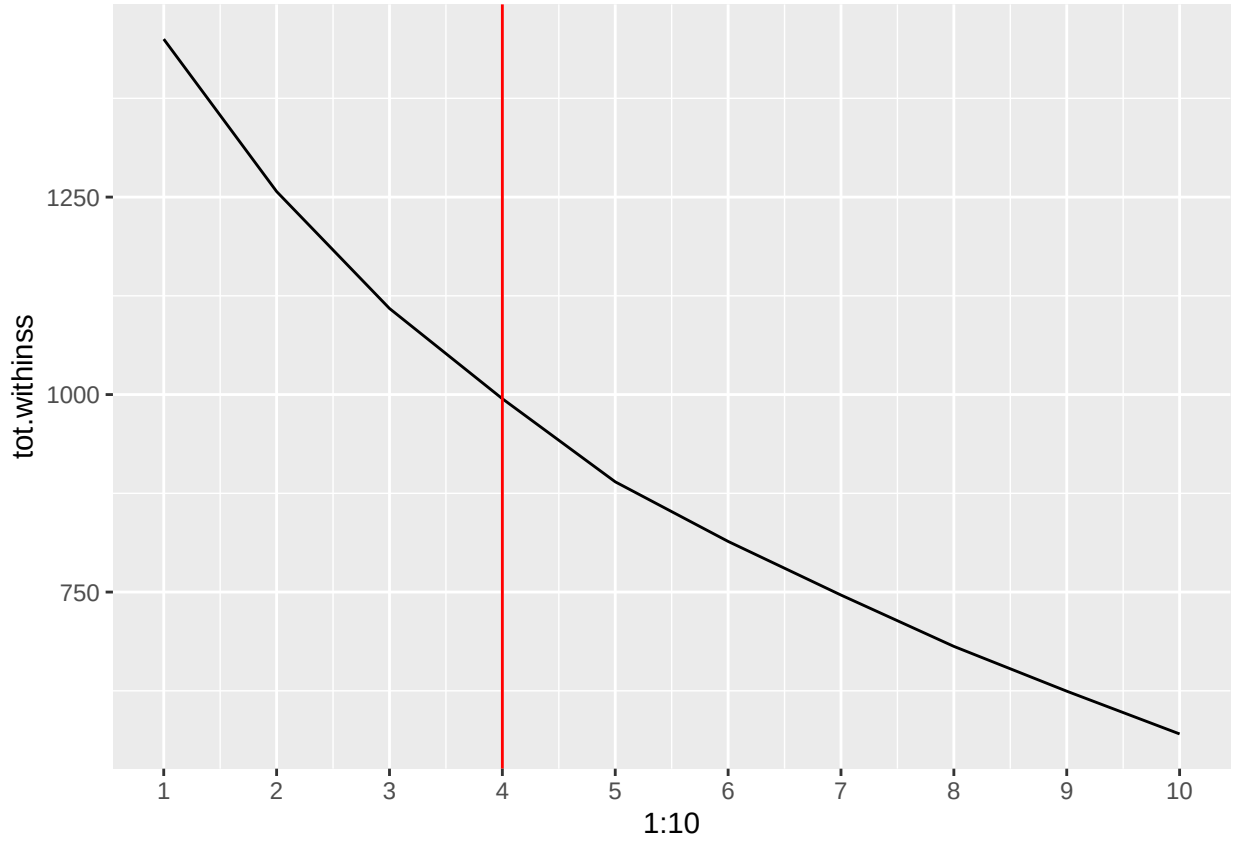
$$\{\Delta_k : k < K^*\} \gg \{\Delta_k : k \geq K^*\}$$

is an estimate of the true number K of clusters within the data, where $\gg$ means "much larger". In our current situation, if we obtain $W(k)$ for a sequence of $k = 1, 2, \ldots, 10$, we can calculate each Δ_k for the sequence, and estimate K. We will know which K by locating the Δ_k where $k < K^*$ is much larger than $k \geq K^*$. We can obtain the total within-cluster sum of squares from the output `tot.withinss` of `kmeans`. Upon trying to understand the concept of where K^* could be calculated, I ran across an article, *How to Automatically Determine the Number of Clusters in your Data - and more*, where the author had examples of how to calculate the 'elbow' within $W(k)$. It was very helpful and I used my findings to create the following graph.

The code for the calculation of the elbow at the best strength was based off the article *How to Automatically Determine the Number of Clusters in your Data - and more* by Vincent Granville, in Data Science Central.

```
##    tot.withinss   Delta k  Delta 2k Strength
```

```
## 1     1450.0000        NA        NA       NA
## 2     1257.0717 192.92829        NA       NA
## 3     1108.9970 148.07469 44.853606 80.43150
## 4      994.7439 114.25309 33.821596 96.15303
## 5      889.5409 105.20306  9.050029 45.63578
## 6      814.1214  75.41942 29.783639 60.39129
## 7      746.2161  67.90536  7.514065 62.06242
## 8      681.2322  64.98389  2.921470 48.42051
## 9      624.5300  56.70220  8.281688 51.56987
## 10     570.3940  54.13604  2.566164       NA
```

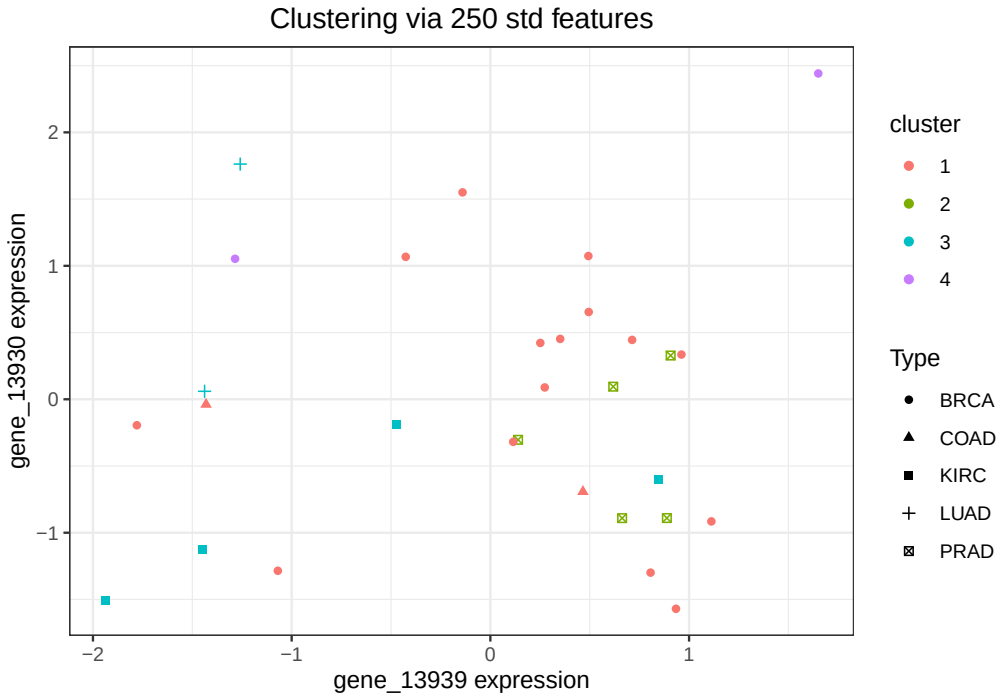


When one is trying to find the optimal clusters to use for k-means clustering, for relatively small data sets, they can run a series of the algorithm for increasing numbers of clusters. One aspect of the `kmeans()` function is that it calculates the twcss scores and stores them in the output. This output can then be captured and used to find the optimal number of clusters. When using the twcss scores, we must calculate Δ_k. As stated above we are looking for the score where for a value of Δ_k is much larger between two cluster values than all others. This shows that the value of k obtains the greatest reduction in the twcss score from the previous k and any further Δ_k will be lower. Graphically this will have a steep drop that leads to a more flat curve, creating the "elbow". From my readints, I was able to learn how to automate finding the best k to use by calculating the highest strength for each value of k. This is obtained by calculating primary and secondary Δ_k values, and then computing their "strength" by taking the difference between the primary and secondary Δ_k values for the k_{i+1}. The largest strength value achieves the best value for k, and therefore, the best number of clusters.

Using More Features

In this section we will explore if using more features for our observations will change the results of the clustering. We will use a total of 250 gene expression values as our features, and the values will come from

the standardized data set. First, we will perform k-means clustering on the new data set with the amount of clusters determined by the use of `clausGap`. The results will be visualized with the values of the same genes (for consistency) with color representing which cluster the sample was assigned to and shape representing which cancer type the sample actually belongs to.



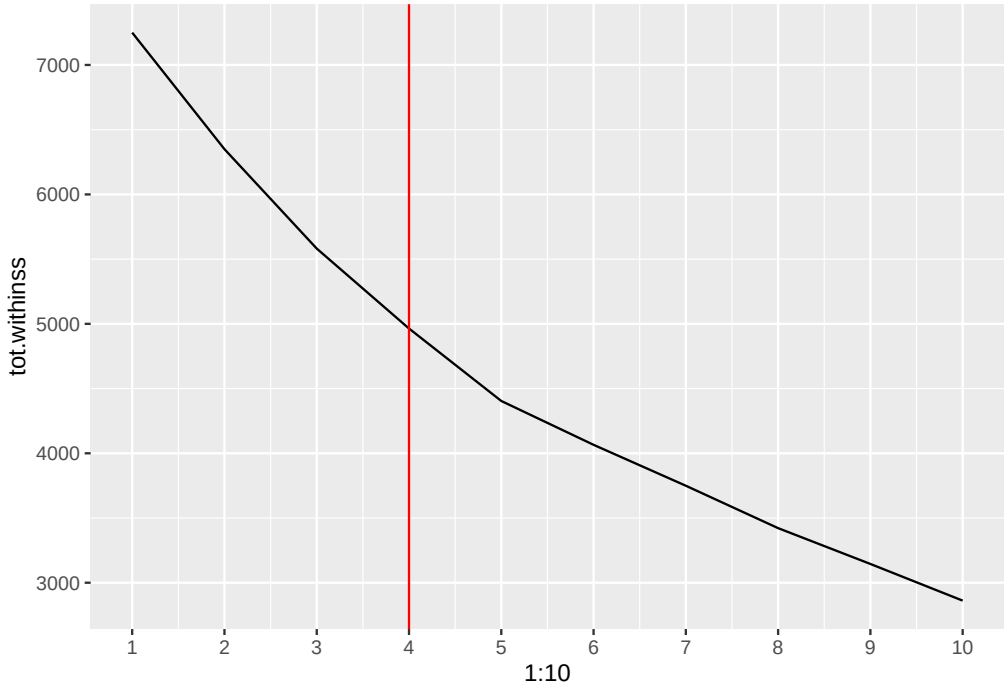
Now, using 250 standardized features, `clausGap` was used to calculate 4 clusters. By having 4 clusters, `kmeans` is much more accurate. Because the amount of clusters is not equal to the amount of cancer types, we cannot achieve 100% accuracy. Below is the same table from above, showing the counts and errors after clustering from the determined value of k from `clausGap`. It shows that for 'BRCA' and 'LUAD' cancer types, the clustering algorithm was able to accurately assign the corresponding samples. The remaining samples had to be clustered into two other groups while there were three types of cancer, causing the clustering algorithm to classify five samples incorrectly.

```
##   Cancer Type Actual Assigned Errors
## 1        BRCA     17       17      0
## 2        COAD      2        5      3
## 3        KIRC      4        6      2
## 4        LUAD      2        2      0
## 5        PRAD      5        0      5
```

Finally, we will verify the number of clusters obtained with `clausGap` through the calculation of the total within-cluster sum of squares and the corresponding Δ_k values.

```
##    tot.withinss  Delta k   Delta 2k Strength
## 1      7250.000       NA         NA       NA
## 2      6349.629 900.3710         NA       NA
## 3      5581.408 768.2206 132.150437 466.5806
## 4      4964.008 617.4006 150.820012 502.3144
## 5      4404.150 559.8575  57.543120 117.7029
## 6      4065.370 338.7802 221.077278 293.1958
## 7      3749.382 315.9880  22.792177 339.2819
## 8      3421.747 327.6349 -11.646928 225.7219
## 9      3145.069 276.6784  50.956533 290.0120
```

```
## 10     2861.724 283.3452  -6.666787       NA
```



Now that we have added more features and performed the same analysis, plotting the new total within-cluster scores gives us the same result as before. Based off of the "strength" score, we still obtain a value of 4 for number of optimal clusters. From this section, we can see that adding more features allows `clausGap` to obtain a better value for k but does not change the results for the twcss scores calculations.

Remarks on K-means Clustering

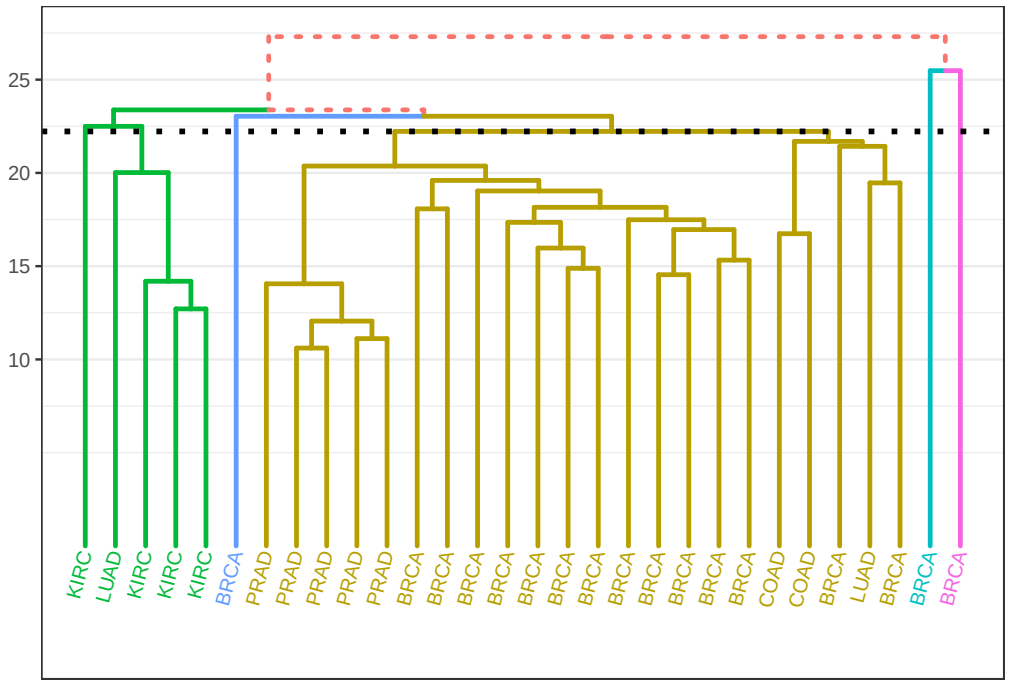
As we can see from these two examples, moving from 50 features to 250 features, the accuracy of the clustering and resulting classification was increased. Adding more features may not always have this affect on the results. If we were to run these same examples with all 1000 features from our standardized data set, the accuracy may not be improved. We still have to think about the curse of dimensionality, which can cause a detriment to the overall accuracy of the clustering algorithm. This is due to the compression of the Euclidean space in increasingly higher dimensions. In this case, adding the extra 200 genes led to an increase in accuracy.

Hierarchical Clustering with 250 Standardized Features

In this section we will use hierarchical clustering on the standardized values with 250 features to cluster the 30 samples. We will use average linkage, single linkage, and complete linkage to show the different results.

Average Linkage

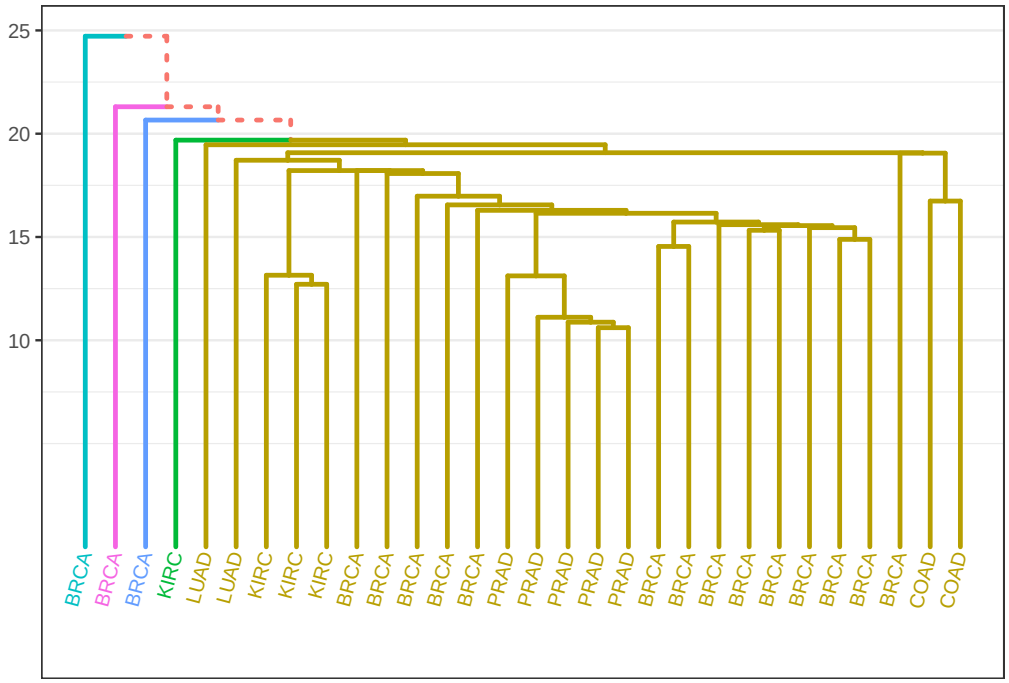
This dendrogram will be produced by using average linkage. Also, for this dendrogram, we will find the height at which we can cut the dendrogram to obtain the same number of groups that are found in our 'labels1' list. We will then compare the results of the clustering to the actual group assignment for each sample.



After cutting the dendrogram to produce 5 clusters, with the average linkage distance metric, we obtain three clusters with only a single sample. Then we obtain a cluster with five samples but contains three different cancer types. The last cluster is very large compared to the other clusters and contains four different cancer types. With the visual help of coloring each cluster with its own color, we can see that the use of average linkage for the distance metric does not properly cluster the samples into their correct clusters.

Single Linkage

Below you will find the dendrogram produced by the use of single linkage as the distance metric.

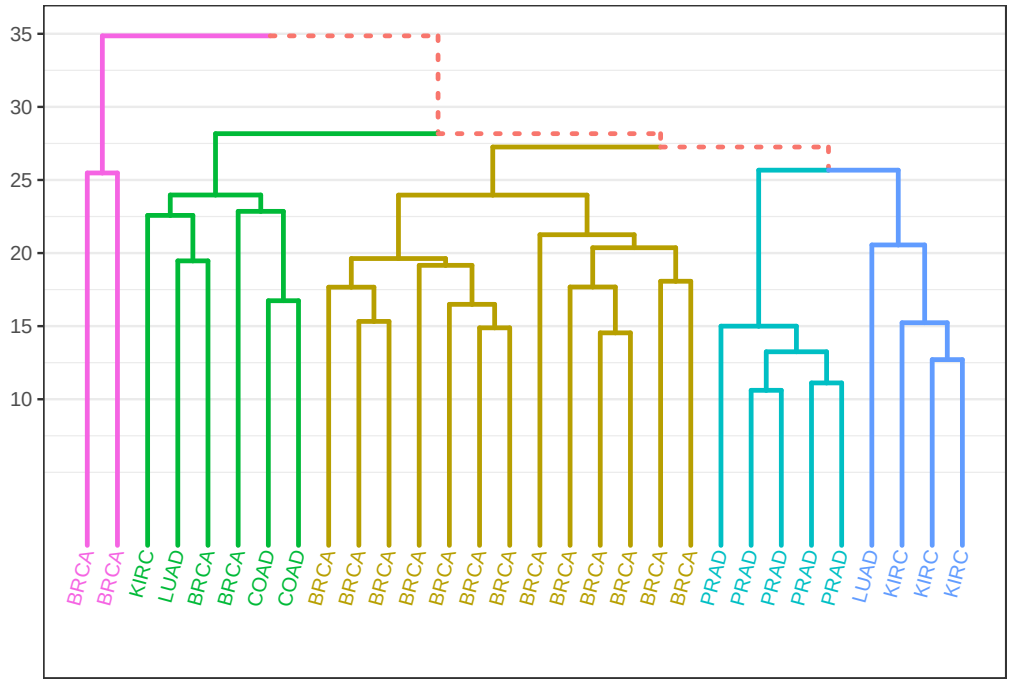


The use of single linkage as the distance metric seems to have less accurate results than the average linkage

distance metric. We can see here that only one sample from differing cancer types were clustered into their own cluster and all other samples were lumped into one large cluster. This is highly inaccurate and the use of single linkage as a distance metric should not be used in this instance.

Complete Linkage

Below you will find the dendrogram produced by the use of complete linkage as the distance metric.



From this plot, we can see that the complete linkage distance metric was the most accurate of the three distance metrics. Three out of the five clusters contain only one type of cancer, one other cluster only contains one type that does not match the others in the cluster, and the last cluster is a combination of four cancer types. Though the clustering was not 100% accurate, it was the most accurate out of the three distance metrics used. This goes to show that only using one distance metrics without exploring others could lead to poor results.

Remarks on Hierarchical Clustering

Average linkage is the most commonly used distance metric for hierarchical clustering, as it produces the best results, generally. Through the examination of each distance metric, we can see that average linkage is not always the the optimal distance metric. This emphasizes that the best distance metric is contextual. Each distance metric should be evaluated to ensure that the proper distance metric is chosen.

Classification

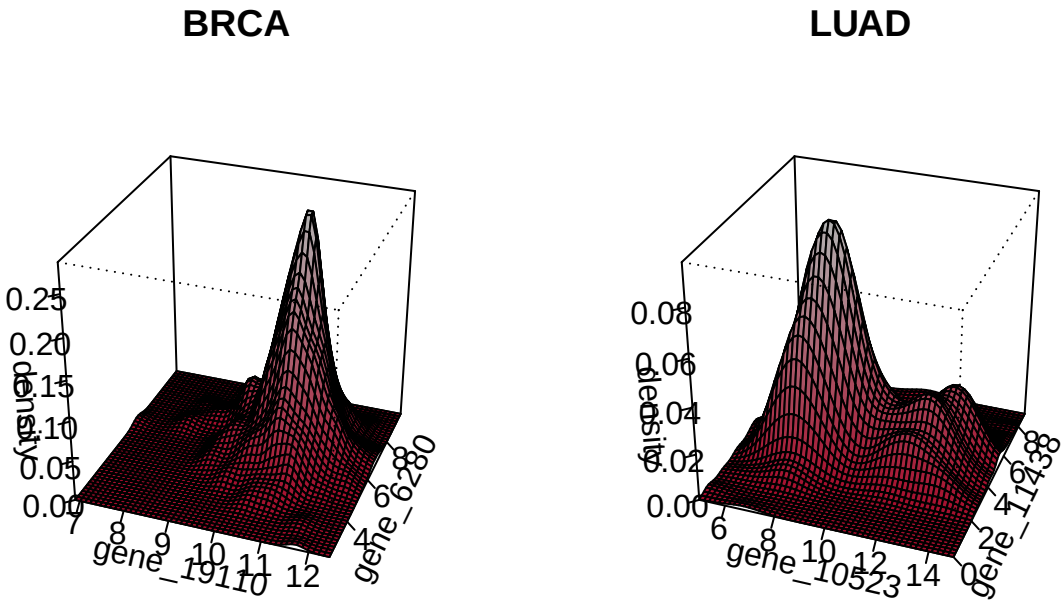
In this section, we will be exploring the use of quadratic discriminant analysis and k-nearest-neighbors to classify our samples from the cancer dataset from earlier. We will subset the data and break it up into training and testing sets. A model will be built and trained with the training data and then its accuracy will be measured by how well the model performs on the unseen testing data.

Quadratic Discriminant Analysis

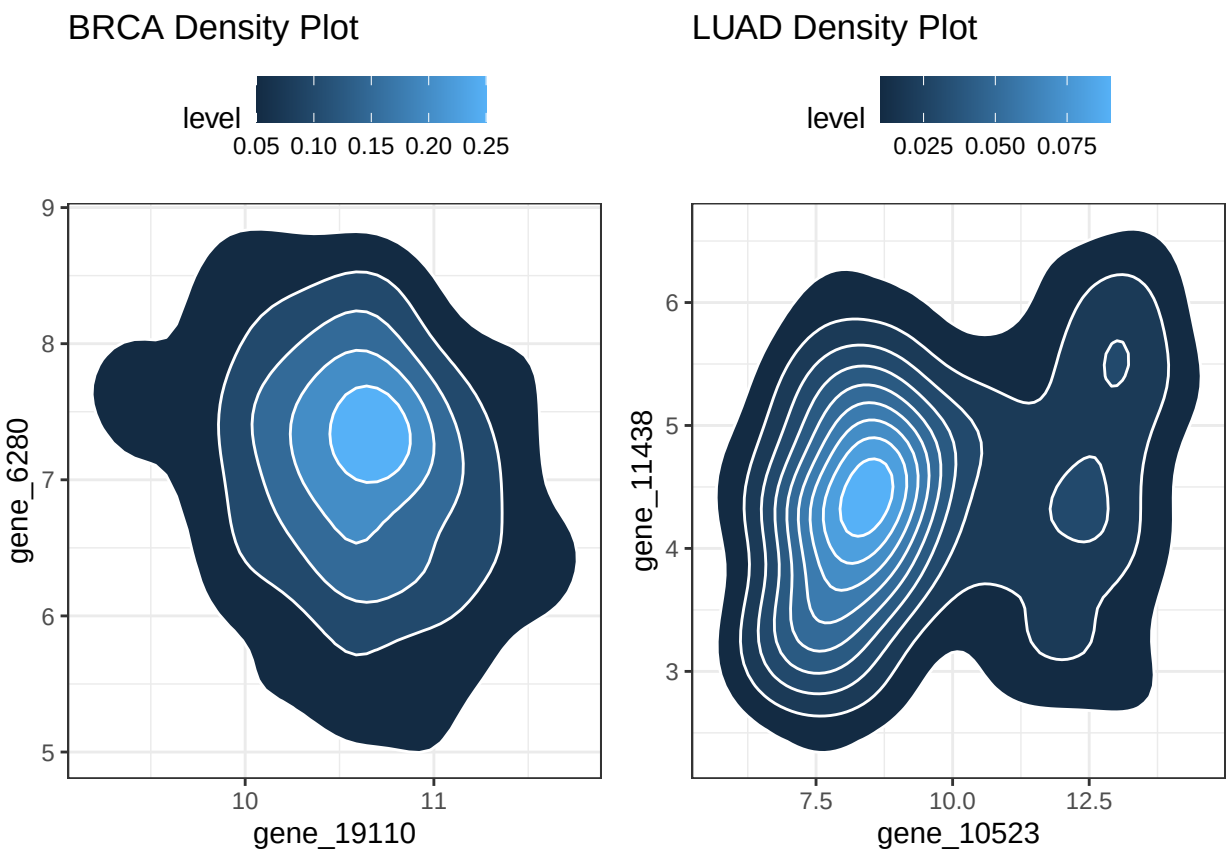
With either linear or quadratic discriminant analysis (QDA), there are some requirements in order for the assumptions to hold true. First, each observation must follow a Gaussian distribution, given the class or group of membership of the observation, and second, each observation must follow a Gaussian mixture model. For our project, each sample (observation) within a group, either 'BRCA' or 'LUAD', would follow a Gaussian distribution with a dimensionality equal to the number of features, gene expressions. Therefore, the more gene expressions (features) we use, the higher the dimension of Gaussian distribution will be.

Check Truth of Gaussian Mixture Assumption

To begin, we will check that the Gaussian mixture assumption is satisfied. We can visually tell if this assumption is satisfied if there are two distinct bumps, because we have two distinct groups, that appear when the density is plotted. Below you will find the density plot for our data.



Below you will find the 2D density contour plot of two random samples of each cancer type, "BRCA" and "LUAD".



Looking at the previous two types of plots, we can see that the "BRCA" data seems to be consistent with a true Guassian distribution, as seen by a mostly circular 2D plot. We can see from the 3D plot that there seems to be some outliers but does not mimic the plots of bivariate data. When looking at the "LUAD" data, there is more data that effects the Guassian distribution, but not in a significant way that would negate the incorporation of the "LUAD" data into the QDA model. Also, if we look at the mode of each density plot, we can see that each cancer type has a different location of its center. This allows the QDA model to be effective at classifying each type of model, since we do not have a large overlap of the two types.

Building a Quadratic Discriminate Analysis Model

Use of 3 Gene Expressions for the Model Next, we will build a quadratic discriminate analysis model for classification of our subset data of "BRCA" and "LUAD" cancer types. First, we will create a subset of the data that only contains 3 genes and their expression values for each sample. We will then pair each sample with its label, coding "BRCA" as '0' and "LUAD" as '1'. Then, we will divide the subset into two further subsets, one containing 60% of the samples as a training set and the other containing the remaining 40 % of the samples for the test set. We will check to make sure no two columns are highly correlated within the training subset. If any two columns are highly correlated, with a sample correlation value grater than 0.9 in absolute value, one column will be removed, both in the training and test sets.

Now that we have obtained subsets that should give us the most reliable results, we will build and train the model. Once the model has been trained with the train subset, we will use the model to predict the labels, cancer types, of the samples in the the test set. Then we will check for classification errors by creating a 2-by-2 table showing actual classification versus predicted classification.

```
##                       True Class Label
## QDA Est. Class Label   0   1
##                     0 124   8
##                     1   4  40
```

As we can see from this table, the QDA model was 96.88 percent correct when classifying the "BRCA" cancer type, and 83.33 percent correct when classifying the "LUAD" cancer type based off the use of three gene types. This gives the overall accuracy of the model to be 93.18 percent.

Use of 100 Gene Expressions for the Model Now we will investigate if using more gene expressions, from three to 100, will enhance the accuracy of the model. We will also be dividing the subset into two further subsets with 75% of the samples as the training set and the remaining 25% of the samples for the test set.

With the creation of the new subsets containing 100 gene expressions, we can build a QDA model. After the model is trained, we will apply the trained model on the test samples and predict their cancer type, "BRCA" or "LUAD". Again, we will check for classification errors by building a 2-by-2 table showing actual classification versus predicted classification.

```
##                      True Class Label
## QDA Est. Class Label  0  1
##                     0 72 38
```

As we can see from this table, the QDA model with 100 features, gene expressions, classified all samples as cancer type "BRCA". Therefore the model was 100% correct in identifying all samples of "BRCA" cancer type and 0% correct in identifying all samples of "LUAD" cancer type. The model with more features, gene expressions, was much less accurate than the model with only three gene expressions. Since we know that in Euclidean space, the higher the dimension the space is, the smaller the space. This leads to a much more compact arrangement of the samples, making it much harder to perform QDA accurately.

Building a k-Nearest-Neighbor Model

Now we will look at building a k-nearest-neighbor (k-NN) model using a random sample of 100 gene expressions from our previous subset of "BRCA" and "LUAD" cancer type samples. Again, we will create two further subsets, one containing 75% of the samples to be used for training and the remaining 25% to be used for a test set. The prediction of a sample belonging to the "BRCA" cancer type will occur when the k-NN model has a prediction value greater than 0.5, and "LUAD" otherwise. Again, we will examine the accuracy of the model by utilizing the 2-by-2 table of classification errors.

```
##        tb4.test.labs
## tb4.kNN BRCA LUAD
##    BRCA   72    3
##    LUAD    0   35
```

Looking at this table of classification errors, we can see that the k-NN model was, again, 100% correct when classifying cancer type "BRCA", while it was only 92.11 percent correct for classification of cancer type "LUAD". The overall accuracy of the model is 97.27 percent. Out of all the classification models, the k-NN model was the most accurate. As compared to those from the QDA model using 100 features, there is a drastic improvement. The QDA model predicted all samples to belong to the "BRCA" cancer type, leaving all "LUAD" cancer type samples to be misclassified. Here, only three samples were misclassified.

Discussion

This project has taken us through the many aspects of using clustering algorithms and classification models. Within the clustering section, we saw the impact of increasing the amount of features used to determine how many clusters would be optimal. Using more features made a difference when utilizing the `clausGap` function for the determination of the number of clusters to use in the `kmeans` algorithm. On the other hand, when using the total within-cluster sum of squares scores to calculate the optimal number of clusters, the amount of features had no effect on the value to use for optimal clusters within the `kmeans` algorithm. The concept of ensuring the correct function or calculation used to determine the amount of clusters should take into account the situation and the amount of features used in the data.

For the exploration of hierarchical clustering, the comparison of each distance metric, single linkage, complete linkage, and average linkage, showed their varying results. Starting with the average linkage, which is known to be the most commonly used distance metric showed very poor results for our data and the single linkage, though slightly better, was also not accurate. Complete linkage showed the best results and provided insights into the two cancer types that could possibly be very similar. Further investigation into these two cancer types my be needed to develop a sub clustering algorithm to assist in the intricacies of their proper clustering.

We explored the use of quadratic discriminate analysis for classification of two specific cancer types. In order to use the QDA model, we had to first ensure that our data satisfied the Guassian mixture assumption and that highly correlated gene expressions were removed from the data. These preparation steps ensured that the data used would be appropriate to use in the model and receive the most accurate results. After training the model with 60% of the samples and then testing the model on the remaining 40%. The results showed poor results when using more features, due to the model classifying all samples as only one type of cancer. And the model did better when using less features. This was the opposite of what we found with the clustering algorithms.

Finishing things with the k-nearest neighbors algorithm, we were able to see how amazing this algorithm is at classification. Since we only had two different types of cancer to use here, the algorithm did quite well. Using a larger training set, with 75% of samples, the algorithm had more samples to train on and fine tune the model.

After completing this project, there are areas within these sections that would be interesting to investigate further. It would be interesting to see how well the k-nearest neighbors algorithm would do as the number of cancer types increased. Would k-nearest neighbors be able to classify cancer types better than complete linkage hierarchical clustering? Also, at what point does the QDA model break down and not return accurate results. The use of three gene expressions provided accurate results while 100 did not. Is the optimal number of features closer to 3 or 100?

This project was a great exercise to help fully understand the concepts that I had not fully grasped in previous courses. It was great to be able to research and play around with the code in order to visualize the results for many aspects. I loved the contrast of when the `k-means` algorithm did better with more features for the use of `clausGap`, while less features used for the QDA model provided the better results. Being able to put all three of the distance metrics used in hierarchical clustering really drove home the point that just because something is 'usually' the best, doesn't mean it is 'usually' best for your data. I can't wait to dive deeper into these algorithms and see how I can incorporate them into my future work.

Works Cited

1. Granville, V. (2019, March 13). *How to Automatically Determine the Number of Clusters in your Data—And more*—DataScienceCentral.com. Data Science Central. https://www.datasciencecentral.com/how-to-automatically-determine-the-number-of-clusters-in-your-dat/

Appendix

```
knitr::opts_chunk$set(tidy=FALSE)
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, fig.align =
'center')
# Load Packages
library(knitr)
library(formatR)
library(tidyr)
library(dplyr)
library(ggplot2)
library(cluster)
```

```r
library(ggdendro)
source('Plotggdendro.r')
library(MASS)
library(plot3D)
library(gridExtra)
library(class)
# Load in data from download
tempdir <- tempdir()
url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/00401/
         TCGA-PANCAN-HiSeq-801x20531.tar.gz'
file <- 'compdata.tar.gz'
download.file(url, file)

untar(file, exdir = '/Users/chris24/Documents/WSU/Spring\ 2022/
        STAT\ 437/Projects/1')
# Put data and labels into relevant objects
gexp <- read.csv('TCGA-PANCAN-HiSeq-801x20531/data.csv',
                 row.names = 1, header = TRUE)
ctype <- read.csv('TCGA-PANCAN-HiSeq-801x20531/labels.csv',
                  row.names = 1, header = TRUE)
saveRDS(ctype, 'ctype.rds') # Save as R object for faster execution
# Task-A1
# Remove any columns that have 300 or more zero values
gexp2 <- gexp[, colSums(gexp == 0) < 300]

# Set seed to ensure reproducible results
set.seed(123)

# Randomly select 1000 genes and their expressions from `gexp2`
gexp3 <- sample(gexp2, 1000)
saveRDS(gexp3, 'gexp3.rds') # For faster loading in Task B

# Randomly select 30 samples and their labels (from 'labels.csv')
gexpProj1 <- sample_n(gexp3, 30)
labels1 <- ctype[rownames(gexpProj1), ] %>% as.data.frame()
colnames(labels1) <- c('Type')

# Transfer the row names for the labels
rownames <- rownames(gexpProj1)
row.names(labels1) <- rownames

# Scale the gene expression data with mean = 0 and std = 1
stdgexpProj1 <- as.data.frame(scale(gexpProj1))

# Save necessary objects as rda files for faster loading
saveRDS(labels1, file = 'labels1.rds')
saveRDS(stdgexpProj1, file = 'stdgexpProj1.rds')
# Task A2 Part 1
# Load data from rds file
labels1 <- readRDS('labels1.rds')
stdgexpProj1 <- readRDS('stdgexpProj1.rds')

# Set seed to ensure reproducible results
```

```r
set.seed(123)

# Subset `stdgexProj1` to only 50 genes
ta2.p1.sub <- as.data.frame(sample(stdgexpProj1, 50))

# Use `clusGap` function on standardized data
# to estimate the true number of clusters
gap = clusGap(t(ta2.p1.sub), kmeans, K.max = 10, B = 200, iter.max = 100)
k = maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"], method = "Tibs2001SEmax")

# Perform k-means clustering with the number of clusters calculated above
ta2.p1.k <- kmeans(ta2.p1.sub, k, iter.max = 100, nstart = 25,
                   algorithm = 'Hartigan-Wong')
# Augment `stdgexProj1` with the k-means clustering results
ta2.p1 <- as.data.frame(ta2.p1.sub)
ta2.p1$Type <- factor(labels1$Type)
ta2.p1$cluster <- factor(ta2.p1.k$cluster)

# Get 2 random genes for plotting
genes.plt <- sample(colnames(ta2.p1[,-2]), 2)

# Visualize the clustering results
ta2.p1.plt <- ggplot(ta2.p1, aes(ta2.p1[,genes.plt[1]], ta2.p1[, genes.plt[2]])) +
                xlab(paste0(genes.plt[1], ' expression')) +
                ylab(paste0(genes.plt[2], ' expression')) +
                theme_bw() +
                geom_point(aes(shape = Type, color = cluster), na.rm = T) +
                theme(legend.position = 'right') +
                ggtitle('Clustering via 50 std features') +
                theme(plot.title = element_text(hjust = 0.5))
ta2.p1.plt
# Task A2 Part 1 Classification Error Counts
# Get counts for each cancer type present in subset
cnts <- table(labels1) %>% as.data.frame()
colnames(cnts) <- c('Cancer Type', 'Actual')

# Get counts for each cancer type as clustered by k-means
c.types <- list(BRCA = c(1), COAD = c(2), KIRC = c(3), LUAD = c(4), PRAD = c(5))
ta2.p1$cluster <- as.numeric(ta2.p1$cluster)
ta2.p1 <- ta2.p1 %>% left_join(., stack(c.types), by = c(cluster = "values"))
acts.list <- as.data.frame(ta2.p1$ind)
acts <- table(acts.list) %>% as.data.frame()

# Combine Actual and Clustered counts
cnts$Assigned <- acts$Freq
cnts$Errors <- with(cnts, abs(Actual - Assigned))
cnts
# Task A2 Part 2 - Total Within-Cluster Sum of Squares
# Create an empty list to store our values
twcss.scores <- c()
# Obtain the 'total within-cluster sum of squares' values
# for each number of clusters used
for (i in 1:10){
```

```
  set.seed(123)
  k.result <- kmeans(ta2.p1.sub, i, iter.max = 100, nstart = 25,
                     algorithm = 'Hartigan-Wong')
  twcss.scores[i] <- k.result$tot.withinss
}
# Calculate delta_k
deltak <- c()
for (i in 2:10){
  deltak[i] <- twcss.scores[i - 1] - twcss.scores[i]
}

# Calculate 2nd order delta_k
delta2k <- c()
for (i in 3:10){
  delta2k[i] <- deltak[i - 1] - deltak[i]
}

# Calculate strength of elbow
strength <- c()
for (i in 3:9){
  strength[i] <- deltak[i + 1] - delta2k[i + 1]
}

# Determine largest strength value
elbow <- which.max(strength)

# Put all values into a dataframe
twcss.scores <- rbind(twcss.scores, deltak, delta2k, strength)
twcss.scores <- as.data.frame(twcss.scores)
twcss.scores <- as.data.frame(t(twcss.scores))
colnames(twcss.scores) <- c('tot.withinss', 'Delta k', 'Delta 2k', 'Strength')
rownames(twcss.scores) <- c(1:10)
twcss.scores.p2 <- as.data.frame(twcss.scores)
twcss.scores.p2
# Task A2 Part 2 - Total Within-Cluster Sum of Squares Plot
# Create a plot of 'total within-cluster sum of squares' values against
# the number of clusters
twcss.ta2.p2.plt <- ggplot(twcss.scores.p2, aes(1:10, tot.withinss)) +
                      geom_line() +
                      scale_x_continuous(breaks = c(1:10)) +
                      geom_vline(xintercept = elbow, col = 'red')
twcss.ta2.p2.plt
# Task A2 Part 3
# Set seed to ensure reproducible results
set.seed(123)

# Obtain a subset of the standardized data
ta2.p3.sub <- as.data.frame(sample(stdgexpProj1, 250))

# Use `clusGap` function on standardized data
# to estimate the true number of clusters
gap.p3 = clusGap(t(ta2.p3.sub), kmeans, K.max = 10, B = 200, iter.max = 100)
k.p3 = maxSE(gap.p3$Tab[, "gap"], gap.p3$Tab[, "SE.sim"], method = "Tibs2001SEmax")
```

```r
# Perform k-means clustering with the number of clusters calculated above
ta2.p3.k <- kmeans(ta2.p3.sub, k.p3, iter.max = 100, nstart = 25,
                   algorithm = 'Hartigan-Wong')

# Augment `stdgexProj1` with the k-means clustering results
ta2.p3 <- as.data.frame(ta2.p3.sub)
ta2.p3$Type <- factor(labels1$Type)
ta2.p3$cluster <- factor(ta2.p3.k$cluster)

# Get 2 random genes for plotting
genes.p3.plt <- sample(colnames(ta2.p3[,-2]), 2)

# Visualize the clustering results
ta2.p3.plt <- ggplot(ta2.p3, aes(ta2.p3[, genes.p3.plt[1]],
                                 ta2.p3[, genes.p3.plt[2]])) +
                xlab(paste0(genes.p3.plt[1], ' expression')) +
                ylab(paste0(genes.p3.plt[2], ' expression')) +
                theme_bw() +
                geom_point(aes(shape = Type, color = cluster), na.rm = T) +
                theme(legend.position = 'right') +
                ggtitle('Clustering via 250 std features') +
                theme(plot.title = element_text(hjust = 0.5))
ta2.p3.plt
# Task A2 Part 3
# Get counts for each cancer type present in subset
cnts.p3 <- table(labels1) %>% as.data.frame()
colnames(cnts.p3) <- c('Cancer Type', 'Actual')

# Get counts for each cancer type as clustered by k-means
ta2.p3$cluster <- as.numeric(ta2.p3$cluster)
ta2.p3 <- ta2.p3 %>% left_join(., stack(c.types), by = c(cluster = "values"))
acts.p3.list <- as.data.frame(ta2.p3$ind)
acts.p3 <- table(acts.p3.list) %>% as.data.frame()

# Combine Actual and Clustered counts
cnts.p3$Assigned <- acts.p3$Freq
cnts.p3$Errors <- with(cnts.p3, abs(Actual - Assigned))
colnames(cnts.p3) <- c('Cancer Type', 'Actual', 'Assigned', 'Errors')
cnts.p3
# Task A2 Part 3 - Total Within-Cluster Sum of Squares
# Create an empty list to store our values
twcss.scores.p3 <- c()
# Obtain the 'total within-cluster sum of squares' values
# for each number of clusters used
for (i in 1:10){
  set.seed(123)
  k.result.p3 <- kmeans(ta2.p3.sub, i, iter.max = 100, nstart = 25,
                   algorithm = 'Hartigan-Wong')
  twcss.scores.p3[i] <- k.result.p3$tot.withinss
}
# Calculate delta_k
deltak.p3 <- c()
```

```r
for (i in 2:10){
  deltak.p3[i] <- twcss.scores.p3[i - 1] - twcss.scores.p3[i]
}
# Calculate 2nd order delta_k
delta2k.p3 <- c()
for (i in 3:10){
  delta2k.p3[i] <- deltak.p3[i - 1] - deltak.p3[i]
}
# Calculate strength of elbow
strength.p3 <- c()
for (i in 3:9){
  strength.p3[i] <- deltak.p3[i + 1] - delta2k.p3[i + 1]
}
# Calculate elbow
elbow.p3 <- which.max(strength.p3)

# Put values into dataframe
twcss.scores.p3 <- rbind(twcss.scores.p3, deltak.p3, delta2k.p3, strength.p3)
twcss.scores.p3 <- as.data.frame(twcss.scores.p3)
twcss.scores.p3 <- t(twcss.scores.p3)
colnames(twcss.scores.p3) <- c('tot.withinss', 'Delta k', 'Delta 2k', 'Strength')
rownames(twcss.scores.p3) <- c(1:10)
twcss.scores.p3 <- as.data.frame(twcss.scores.p3)
twcss.scores.p3
# Task A2 Part 3 - Total Within-Cluster Sum of Squares Plot
# Create a plot of 'total within-cluster sum of squares' values against
# the number of clusters
twcss.ta2.p3.plt <- ggplot(twcss.scores.p3, aes(1:10, tot.withinss)) +
                    geom_line() +
                    scale_x_continuous(breaks = c(1:10)) +
                    geom_vline(xintercept = elbow.p3, col = 'red')

twcss.ta2.p3.plt
# Task A3 - Dataset Setup
# Set seed to ensure reproducible results
set.seed(123)

# Create a subset of the standardized data
ta3.base <- as.data.frame(sample(stdgexpProj1, 250))

# Assign the cancer type as a feature
ta3.base$Type <- labels1$Type

# Transpose dataset and assign cancer type as column name
ta3.base.t <- t(ta3.base)
colnames(ta3.base.t) <- ta3.base.t[251,]
# Obtain only the relevant data
ta3.base.t <- ta3.base.t[1:250,]
# Transpose back to a working dataset for hierarchical clustering
ta3.base <- t(ta3.base.t)
# Task A3 - Average Linkage Dendrogram Plot
# Obtain the count of distinct cancer types in dataset
ctype.cnt <- length(unique(labels1[['Type']]))
```

```r
# Perform hierarchical clustering with average distance metric
hc.ta3.avg <- hclust(dist(ta3.base), method = 'average')
# Calculate the height that gives the same number of clusters as that
# of unique cancer types contained in 'labels1'
cut.height <- hc.ta3.avg$height[length(hc.ta3.avg$height) - ctype.cnt]
# Plot the dendrogram through the use of 'Plotggdendro.r'
plt.hc.ta3.avg <- plot_ggdendro(dendro_data_k(hc.ta3.avg, ctype.cnt),
                                direction = 'tb',
                                heightReferece = cut.height,
                                expand.y = 0.2)
plt.hc.ta3.avg
# Task A3 - Single Linkage Dendrogram Plot
# Perform hierarchical clustering with single linkage as the distance metric
hc.pa3.sgl <- hclust(dist(ta3.base), method = 'single')

# Plot the dendrogram with the use of 'Plotggdendro.r'
plt.hc.ta3.sgl <- plot_ggdendro(dendro_data_k(hc.pa3.sgl, ctype.cnt),
                                direction = 'tb',
                                expand.y = 0.2)
plt.hc.ta3.sgl
# Task A3 - Complete Linkage Plot
# Perform hierarchical clustering with complete linkage as the distance metric
hc.pa3.cmp <- hclust(dist(ta3.base), method = 'complete')

# Plot the dendrogram with the use of 'Plotggdendro.r'
plt.hc.ta3.cmp <- plot_ggdendro(dendro_data_k(hc.pa3.cmp, ctype.cnt),
                                direction = 'tb',
                                expand.y = 0.2)
plt.hc.ta3.cmp
# Task B1 - Dataset Setup
# Filtering and random sampling of data was done in Task-A1
# Obtain the sample names whose cancer type is either 'LUAD' or 'BRCA'
tb.ctypes <- c('LUAD', 'BRCA')
ctype <- readRDS('ctype.rds')
labels2 <- ctype %>% filter(Class %in% tb.ctypes)

# Obtain the data for these samples using the row names
gexp3 <- readRDS('gexp3.rds') # Load data from saved object
tb.base <- gexp3[rownames(labels2), ]

# Standardize the data column-wise (for each gene expression)
stdgexp2 <- as.data.frame(tb.base)

# Save object as rds to minimize compute power when knitting
saveRDS(stdgexp2, 'stdgexp2.rds')
saveRDS(labels2, 'labels2.rds')
# Task B2
# Load in the data from our saved objects
stdgexp2 <- readRDS('stdgexp2.rds')
labels2 <- readRDS('labels2.rds')

# Set seed for reproducible results
```

```r
set.seed(123)

# Randomly select 2 samples from each cancer type
brca <- sample(stdgexp2[rownames(labels2["Class"] == "BRCA"), ], 2) %>%
          as.data.frame()
luad <- sample(stdgexp2[rownames(labels2["Class"] == "LUAD"), ], 2) %>%
           as.data.frame()

# Obtain 2D density estimate
brca.fde <- with(brca, MASS::kde2d(brca[, 1], brca[, 2], n = 50),
                 lims = c(min(brca[, 1]), max(brca[, 1]),
                          min(brca[, 2]), max(brca[, 2])))
luad.fde <- with(luad, MASS::kde2d(luad[, 1], luad[, 2], n = 50),
                 lims = c(min(luad[, 1]), max(luad[, 1]),
                          min(luad[, 2]), max(luad[, 2])))

# Create a color palette (100 colors)
col.pal <- colorRampPalette(c('#A60F2D', '#b3b3b3'))
colors <- col.pal(100)

# Obtain centers fo surface facets
nrz <- ncz <- 50
brca.facet.center <- (brca.fde$z[-1, -1] + brca.fde$z[-1, -ncz] +
                      brca.fde$z[-nrz, -1] + brca.fde$z[-nrz, -ncz]) / 4

# Range of colors
brca.facet.range <- cut(brca.facet.center, 100)

# Apply to LUAD data as well
luad.facet.center <- (luad.fde$z[-1, -1] + luad.fde$z[-1, -ncz] +
                      luad.fde$z[-nrz, -1] + luad.fde$z[-nrz, -ncz]) / 4
luad.facet.range <- cut(luad.facet.center, 100)

# Generate plots and display side-by-side
par(mfrow = c(1, 2))
persp(brca.fde, phi = 30, theta = 20, d = 5,
      xlab = colnames(brca)[1], ylab = colnames(brca)[2],
      zlab = 'density', main = "BRCA", r = sqrt(0.5),
      ticktype = 'detailed', col = colors[brca.facet.range])
persp(luad.fde, phi = 30, theta = 20, d = 5,
      xlab = colnames(luad)[1], ylab = colnames(luad)[2],
      zlab = 'density', main = "LUAD", r = sqrt(0.5),
      ticktype = 'detailed', col = colors[luad.facet.range])
# Create 2D density contour plot for BRCA and LUAD
brca.2d.plt <- ggplot(brca, aes(brca[, 1], brca[, 2])) + theme_bw() +
                 stat_density_2d(aes(fill = ..level..),
                                 geom = 'polygon', color = 'white') +
                 labs(title = "BRCA Density Plot") +
                 xlab(colnames(brca)[1]) +
                 ylab(colnames(brca)[2]) +
                 theme(legend.position = 'top', legend.direction = 'horizontal')
luad.2d.plt <- ggplot(luad, aes(luad[, 1], luad[, 2])) + theme_bw() +
                 stat_density_2d(aes(fill = ..level..),
```

```
                                  geom = 'polygon', color = 'white') +
                labs(title = "LUAD Density Plot") +
                xlab(colnames(luad)[1]) +
                ylab(colnames(luad)[2]) +
                theme(legend.position = 'top', legend.direction = 'horizontal')
grid.arrange(brca.2d.plt, luad.2d.plt, ncol = 2)
# Task B2 - Dataset Setup
# Set seed for reproducible results
set.seed(123)

# Choose 3 random genes and their expression values for all samples
stdgexp2a <- as.data.frame(sample(stdgexp2, 3))

# Apply labels to the samples and encode them with numerical values
# BRCA = 0, LUAD = 1
stdgexp2a$Type <- labels2$Class
stdgexp2a <- stdgexp2a %>% mutate(Class = case_when (
               Type == "BRCA" ~ 0,
               Type == "LUAD" ~ 1))
stdgexp2a <- subset(stdgexp2a, select = -Type)
colnames(stdgexp2a) <- c("Gene1", "Gene2", "Gene3", "Class")

# Choose 60% of dataset for training set
stdgexp2a.train <- stdgexp2a %>% sample_frac(0.6)
stdgexp2a.test <- stdgexp2a[!(rownames(stdgexp2a) %in%
                              rownames(stdgexp2a.train)), ]

# Obtain the correlations between each column and all others
tb2.corr <- stdgexp2a.train[, 1:3] %>% cor() %>% abs() %>% data.frame()

# Remove any highly correlated columns in both train and test sets
stdgexp2a.train <- stdgexp2a.train[, colSums(tb2.corr > 0.9 &
                                             tb2.corr < 1) == 0]
colnames(stdgexp2a.train) <- colnames(stdgexp2a)
stdgexp2a.test <- stdgexp2a.test[, colnames(stdgexp2a.train)]
colnames(stdgexp2a.test) <- colnames(stdgexp2a)
# Task B2 - Train, Test, and Process Results of QDA Model w/3 Gene Expressions
# Build and train the qda model
tb2.qda.fit <- qda(factor(Class) ~ Gene1 + Gene2 + Gene3,
                   data = stdgexp2a,
                   subset = rownames(stdgexp2a.train))

# Create predictions on the test set
tb2.qda.pred <- predict(tb2.qda.fit, stdgexp2a.test)

# Obtain estimated class label based off of posterior probabilities
# where Pr(Class = 0|X) > 0.5, as the threshold rule
# First create a vector of all zeros for all test samples
tb2.qda.estlab <- rep(1, nrow(stdgexp2a.test))
tb2.qda.estBRCA <- which(tb2.qda.pred$posterior[, 1] > 0.5)
tb2.qda.estlab[tb2.qda.estBRCA] = 0

# Calculate classification errors in 2-by-2 table
```

```r
`True Class Label` <- stdgexp2a.test$Class
`QDA Est. Class Label` <- tb2.qda.estlab
table(`QDA Est. Class Label`, `True Class Label`)
# Task B3 - Dataset Setup
# Set seed for reproducible results
set.seed(123)

# Choose 3 random genes and their expression values for all samples
stdgexp2b <- as.data.frame(sample(stdgexp2, 100))

# Apply labels to the samples and encode them with numerical values
# BRCA = 0, LUAD = 1
stdgexp2b$Type <- labels2$Class
stdgexp2b <- stdgexp2b %>% mutate(Class = case_when (
               Type == "BRCA" ~ 0,
               Type == "LUAD" ~ 1))
stdgexp2b <- subset(stdgexp2b, select = -Type)

# Choose 75% of dataset for training set
stdgexp2b.train <- stdgexp2b %>% sample_frac(0.75)
stdgexp2b.test <- stdgexp2b[!(rownames(stdgexp2b) %in%
                              rownames(stdgexp2b.train)), ]

# Obtain the correlations between each column and all others
tb3.corr <- stdgexp2b.train[, -1] %>% cor() %>% abs() %>% data.frame()

# Remove any highly correlated columns in both train and test sets
stdgexp2b.train <- stdgexp2b.train[, colSums(tb3.corr > 0.9 &
                                             tb3.corr < 1) == 0]
stdgexp2b.test <- stdgexp2b.test[, colnames(stdgexp2b.train)]
# Task B2 - Train, Test, & Process Results of QDA Model w/100 Gene Expressions
# Build and train the qda model
tb3.qda.fit <- qda(Class ~ .,
                   data = stdgexp2b,
                   subset = rownames(stdgexp2b.train))

# Create predictions on the test set
tb3.qda.pred <- predict(tb3.qda.fit, stdgexp2b.test)

# Obtain estimated class label based off of posterior probabilities
# where Pr(Class = 0|X) > 0.5, as the threshold rule
# First create a vector of all zeros for all test samples
tb3.qda.estlab <- rep(1, nrow(stdgexp2b.test))
tb3.qda.estBRCA <- which(tb3.qda.pred$posterior[, 1] > 0.5)
tb3.qda.estlab[tb3.qda.estBRCA] = 0

# Calculate classification errors in 2-by-2 table
`True Class Label` <- stdgexp2b.test$Class
`QDA Est. Class Label` <- tb3.qda.estlab
table(`QDA Est. Class Label`, `True Class Label`)
# Task B4 - Train, Test, & Process Results of k-NN Model w/100 Gene Expressions
# Set seed to ensure reproducible results
set.seed(123)
```

```r
# Choose 3 random genes and their expression values for all samples
stdgexp2c <- as.data.frame(sample(stdgexp2, 100))

# Choose 75% of dataset for training set
# Remaining samples for test set
tb4.train <- stdgexp2c %>% sample_frac(0.75)
tb4.test <- stdgexp2c[!(rownames(stdgexp2c) %in%
                            rownames(tb4.train)), ]

# Obtain labels for train and test sets
tb4.train.labs <- labels2[rownames(tb4.train), ]
tb4.test.labs <- labels2[rownames(tb4.test), ]

# Create k-NN model
tb4.kNN <- knn(tb4.train, tb4.test, cl = tb4.train.labs, k = 2)

# Report results in 2-by-2 classification error table
table(tb4.kNN, tb4.test.labs)

library(devtools)
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] devtools_2.4.3  usethis_2.1.5   class_7.3-19    gridExtra_2.3
##  [5] plot3D_1.4      MASS_7.3-54     ggdendro_0.1.23 cluster_2.1.2
##  [9] ggplot2_3.3.5   dplyr_1.0.7     tidyr_1.1.4     formatR_1.11
## [13] knitr_1.34
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.1  xfun_0.26         remotes_2.4.2     purrr_0.3.4
##  [5] tcltk_4.1.1       testthat_3.0.4    colorspace_2.0-2  vctrs_0.3.8
##  [9] generics_0.1.0    htmltools_0.5.2   yaml_2.2.1        utf8_1.2.2
## [13] rlang_0.4.11      pkgbuild_1.3.0    isoband_0.2.5     pillar_1.6.2
## [17] glue_1.4.2        withr_2.4.2       DBI_1.1.1         sessioninfo_1.2.2
## [21] lifecycle_1.0.0   stringr_1.4.0     munsell_0.5.0     gtable_0.3.0
## [25] evaluate_0.14     memoise_2.0.0     labeling_0.4.2    misc3d_0.9-1
## [29] callr_3.7.0       fastmap_1.1.0     ps_1.6.0          fansi_0.5.0
## [33] scales_1.1.1      cachem_1.0.6      desc_1.4.0        pkgload_1.2.2
```

```
## [37] farver_2.1.0      fs_1.5.0          digest_0.6.27     stringi_1.7.4
## [41] processx_3.5.2    rprojroot_2.0.2   grid_4.1.1        cli_3.1.0
## [45] tools_4.1.1       magrittr_2.0.1    tibble_3.1.4      crayon_1.4.1
## [49] pkgconfig_2.0.3   ellipsis_0.3.2    prettyunits_1.1.1 assertthat_0.2.1
## [53] rmarkdown_2.11    rstudioapi_0.13   R6_2.5.1          compiler_4.1.1
```