Wine Classification

hic sunt dracones

The Mapper Algorithm

What is the Mapper Algorithm?

1. From data set $\mathbb{X}$
2. We create a matrix of distances where the i, jth entry represents the Euclidean distance between points i and j.
3. From a quick analysis of our distance matrix, we reasonably choose ϵ
4. By using our chosen ϵ, we create an adjacency matrix.
   If the distance between two points i and j is less than ϵ we put a 1 in row i column j, otherwise 0
5. We proceed to cover our space by the following
   (a) We select a point at random
   (b) We draw a ball of radius ϵ around it (call it Cover i).
   (c) Points within ϵ of our chosen point are allocated to Cover i.
   (d) We select an uncovered/unallocated point and repeat the last two steps now allocating it to Cover $i + 1$, until all points have been covered.
6. We create a new adjacency matrix of covers where two covers are adjacent if there exists points in the intersection.
7. We select additional information for coloring and sizing our nodes
8. Visualize it

Wine Map

Here we have two separate sets of wine data, one for red and one for white. We take a sample of 100, merge the sets, then map them.

- Each set has twelve columns of numeric data, the last being a numeric ‘quality‘ column which we consider to be a response variable and is therefore not included in the distance calculation.
- We add a ‘type‘ column of 1’s and 0’s where 1 indicates red wine and 0 indicated white. This is not used in the distance calculation, it is only used after the fact for node coloring and as a description of the makeup of each open cover.

Other Notes:

- ϵ : 25
- Sample Size: 100

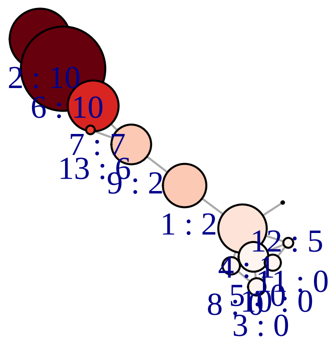
Mapped

Nodes in the map below have been labeled in the following manner:

Cover Number : Wine Proportion

Where the Wine Proportion is a scale from 0 to 10, 0 indicating a node of white wines and 10 indicating reds. This proportion is also used to shade each node on the white to red spectrum.

Node size was determined by the number of wines in each open cover.



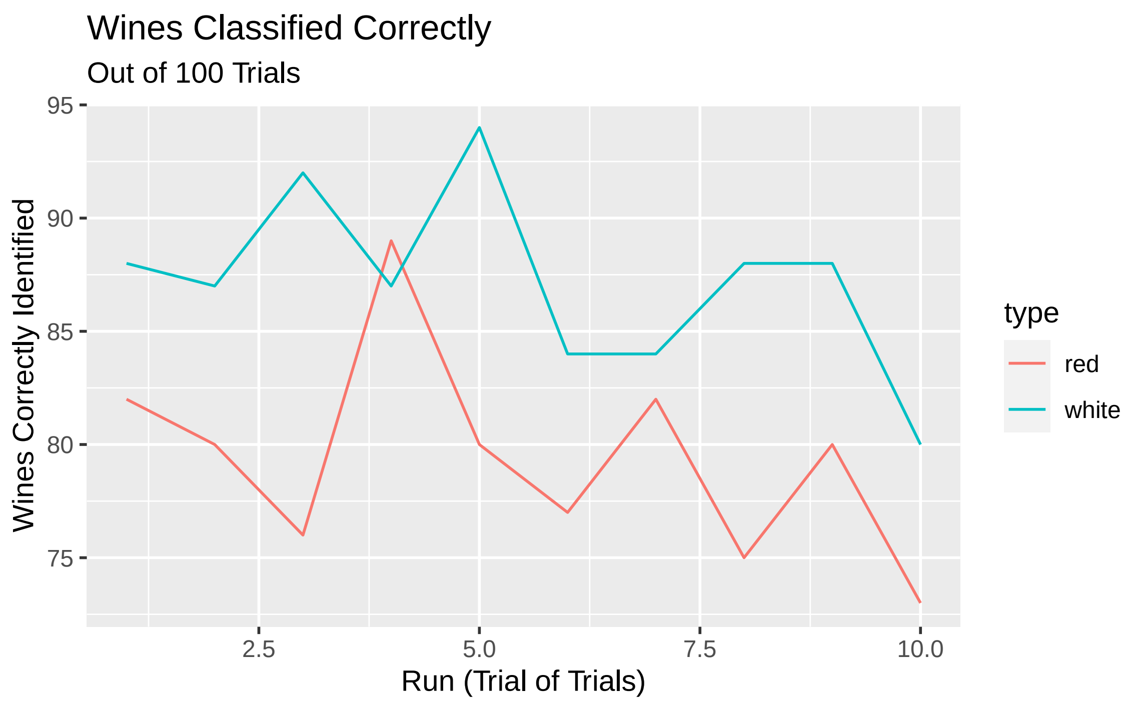
Using our Map

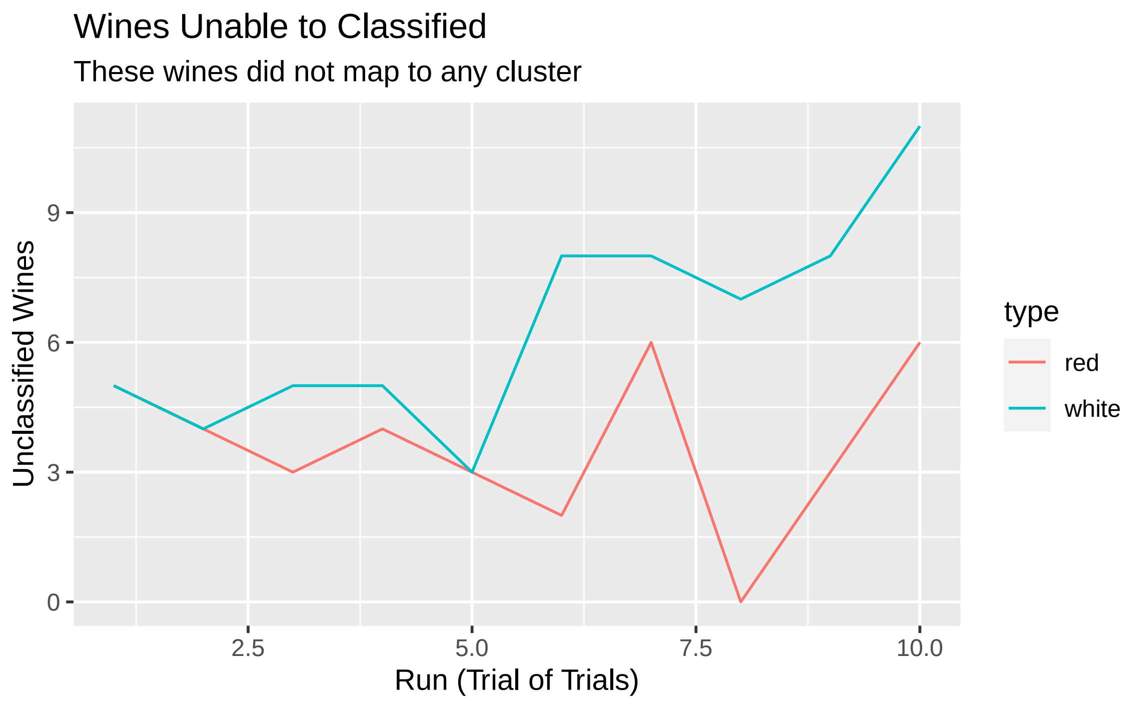
Can we use our map above as a classifier?

1. Given a new data point
2. Determine distance from the center of each open cover
3. A point within ϵ of the center point is allocated to that cluster
4. Given the makeup of the clusters the new point maps to we should be able to classify our new data point

Run Trials

We do 10 runs of 100 trials





Thoughts:

- If a point maps to multiple clusters, can we choose one cluster by the minimum of the distances from each center point to our new one?
- Alternatively can we choose the most extreme cluster our new point maps to?
- How does our choice of sample size, ϵ, etc impact the accuracy of our classifier?

Appendix I: Code

Generating the Map

```r
##### Packages ####
library(tidyverse)
library(igraph)
library(RColorBrewer)
library(FredsVietorisRips)

##### Import Data ####
# Uncomment the below code to download the data directly from my github
# red <- read.csv(
#   "https://raw.githubusercontent.com/ftkjr/FredsVietorisRips/master/data/winequality-red.csv")
# white <- read.csv(
#   "https://raw.githubusercontent.com/ftkjr/FredsVietorisRips/master/data/winequality-white.csv")

# If you uncommented the above code, comment the below
red <- read.csv("../data/winequality-red.csv")
white <- read.csv("../data/winequality-white.csv")

##### Add column Type ####
# Column of
#   1's for Red
#   0's for White
red$type <- 1
white$type <- 0

##### Sample and Merge ####
# 1) Take two samples of size sample_size
# 2) Merge them into a data frame called df
# set.seed(2020)
sample_size <- 100
df <- red %>%
  sample_n(sample_size) %>%
  full_join(sample_n(white, sample_size))

##### Distance Matrix ####
# Populate a matrix with the
# Euclidean distance, so that [i, j]
# is the distance between the ith and jth points
d <- df[1:(ncol(df)-2)] %>%
  dist() %>%
  as.matrix()


##### Open Covers ####
# AdjacencyMatrix:
#   Points within epsilon of one another are adjacenct
# OpenCovers:
#   Select Point at random, points adjacent are allocated to
#     that "open cover"/node
#   Repeat until all points are in at least one open cover
```

```r
#
# Distance matrix -> Adjacency Matrix -> Open Covers
epsilon <- 25
covers <- d %>%
  AdjacencyMatrix(epsilon) %>%
  OpenCoverEballs()

##### Adjacent Covers ####
# CoverAdjacencies:
#   Covers which share points are adjacent
#
# Create a graph from an
# Adjacency matrix of our covers
covers.plot <- covers %>%
  CoverAdjacencies() %>%
  graph_from_adjacency_matrix(mode="undirected")
```

Visualizing the Map

```r
##### Graph ####
# Size the nodes by the number of wines within each cluster
# Color by the proportion of red to white wine
covers.size <- vector()
covers.color <- vector()

##### Node Features ####
for (i in 1:length(covers)) {
  ##### Node Color ####
  # Proportion of white to red on a 0 - 10 scale
  # 0 indicates all white wine and 10 indicates all red
  covers.color[i] <- round( mean( df$type[ covers[[i]] ]) * 10 )

  ##### Node Size ####
  covers.size[i] <- length(covers[[i]])
}

##### Coloring ####
redscale <- brewer.pal(9, "Reds")
redscale <- colorRampPalette(redscale)(11)
V(covers.plot)$color <- redscale[covers.color + 1]

##### Sizing ####
V(covers.plot)$size <- covers.size

##### Labeling #####
# We label by the following:
#   Cover_Number : Proportion of Red
V(covers.plot)$label <- paste("\n\n", 1:length(covers), ":", covers.color)

##### Plot ####
plot(covers.plot)
```

Classifier Code

```r
ClassifyNewPoint <- function(df, list_of_covers, new_point) {

  ##### Initialize Empty Frame ####
  center_points <- data.frame(
    cover = 1:length(list_of_covers),
    point = 0
  )

  ##### Create index column ####
  df$point <- 1:nrow(df)

  ##### Find the center of each cover ####
  # For each open cover
  #   find the center point
  #   put it in row cover, column 2 of center_points
  # Each cover's center point is in the first position
  for ( cover in 1:length(list_of_covers) ) {
    center_points[cover, 2] <- list_of_covers[[cover]][1]
  }

  ##### Center Point Data ####
  center_points <- left_join(center_points, df, by = "point")
  # center_points

  ##### Which Center Points are we adjacent to? ####
  adjacencies <- center_points[, 3:13] %>%
    rbind(new_point[, 1:11]) %>%
    dist() %>%
    as.matrix() %>%
    AdjacencyMatrix(epsilon)

  ##### Which Covers are we in? ####
  cover_allocation <- which(adjacencies[nrow(adjacencies), ] == 1)

  ##### Return which covers we're in ####
  return(cover_allocation)
}


ClassifyWine <- function(df, sample_frame, frame_name,
                         list_of_covers, cover_makeup, trials,
                         talktome = FALSE) {
  ##### Initialize Results Vectors ####
  # How many correct
  # How many couldn't we classify (NOT THE NUMBER OF INCORRECTLY CLASSIFIED)
  positives <- 0
  unclassified <- 0

  ##### For each of our trials ####
  # Pick a random Wine
  # Classify it
  for (trial in 1:trials) {
    if (talktome == TRUE) cat("\nTrial: ", trial, ":\n")

    ##### Pick a random wine from one of the data frames ####
    sample_point <- sample_frame[sample(1:nrow(sample_frame), 1), ]

    ##### Classify the Wine ####
    classified_covers <- ClassifyNewPoint(df, list_of_covers, sample_point)

    ##### Results ####
    # If it didn't map to a cluster, it's unclassified
    #   or if the cluster value is 5 exactly then we can't classify it
    # If the average of the cover values is over 5, it's probably red
    # If the average of the cover values is less than 5, it's probably white
    if (length(classified_covers) == 0 | mean(cover_makeup[classified_covers]) == 5) {
          unclassified <- unclassified + 1
      # cat("\nCannot Classify. Sorry.\n")
        } else {
          if (mean(cover_makeup[classified_covers]) > 5) {
            if (frame_name == "red") positives <- positives + 1
            # cat("\nIt's probably Red\n")
          } else if (mean(cover_makeup[classified_covers]) < 5) {
            if (frame_name == "white") positives <- positives + 1
            # cat("\nIt's probably White\n")
          }
    }

  }
  results <- c(positives, unclassified)
  names(results) <- c("positives", "unclassified")
  return(results)
}

PopulateResults <- function(runs, trials) {
  ##### Data Frame of Results ####
  # So we can plot them later
  results_frame <- data.frame(
    type = c(rep("white", runs), rep("red", runs)),
    trial = rep(1:runs, 2),
    result = 0,
    unclassified = 0
  )

  ##### Runs of Trials ####
  # Each Run has Trials
  # (I hope the distinction provides some clarification)
  for (r in 1:runs) {
    results_frame[r, c(3,4)] <- ClassifyWine(df, white, "white", covers, covers.color, trials)

    results_frame[r + runs, c(3,4)] <- ClassifyWine(df, red, "red", covers, covers.color, trials)
  }

  ##### Return a data frame of our results ####
```

```r
  return(results_frame)
}
```

Visualize Classification Results

```r
##### Run our Trials ####
trialresults <- PopulateResults(runs = 10, trials = 100)


##### Chart Skeleton ####
chart <- trialresults %>%
  ggplot(aes( x = trial ))

##### Correct Classifications ####
chart +
  geom_line(aes( y = result, color = type )) +
  xlab("Run (Trial of Trials)") +
  ylab("Wines Correctly Identified") +
  ggtitle("Wines Classified Correctly", "Out of 100 Trials")

##### Unclassifiable Points ####
chart +
  geom_line(aes( y = unclassified, color = type )) +
  xlab("Run (Trial of Trials)") +
  ylab("Unclassified Wines") +
  ggtitle("Wines Unable to Classified",
          "These wines did not map to any cluster")
```