Homework 2 (ISYE 6501) — Question 3.1

Question 4.1 Answer

Sorry in advance to my peer reviewers! (— —) I like to write out my reasoning and can be… wordy… But I will try to be more brief this time!
Also please note I did the majority of this homework with gloves on due to my house being very cold throughout the winter storm in North America. If you notice typos I somehow didn't catch - this is why. Sorry!

Currently, a clustering model would be appropriate for showing how my work activities are distributed across my role in recent months.
In my job, I was originally hired to work across four main areas:
1. Salesforce migration
2. Research team support
3. Membership team support
4. Standalone data deduplication project

Each observation would represent a task or ticket I worked on, and clustering could reveal whether tasks naturally group by function or whether one area dominates my workload
Currently, this would help me demonstrate that my recent work has clustered heavily around Salesforce-related tasks rather than being balanced across responsibilities, and that I want it to be rebalanced once more so I don't end up being burnt-out.

The predictors I would use for this are:
1. System(s) involved (Salesforce, Aptify, both)
2. Requesting team (research, membership, IT)
3. Time spent on task (<1hr, <6hr, <12hr, <24hr, >24hr)
4. Task type (migration, question, report)
5. Project priority level (low, medium, high)

Question 4.2 Answer

Firstly, I need to do some setup, and load the data. `kmeans()` is built into R, so I don't need to load any libraries.
I noticed that the data file we were given this time was not tab-delimited, so I couldn't use `read.delim()` and instead chose to split it on whiterspace, which loaded the 5 columns as expected with 150 data points.

```
## [1] 150   5
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
```

```
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 12          4.8         3.4          1.6         0.2  setosa
## 13          4.8         3.0          1.4         0.1  setosa
## 14          4.3         3.0          1.1         0.1  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Now that I have checked the data quality, I need to identify the predictors and the response, because there are only 4 predictors, I felt I could easily write them out using `c()`. Then, because kmeans uses euclidean distance, we need to scale the predictor data.

```
i_y_true <- i_df$Species
i_x_all  <- i_df[, c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")]
i_x_all_scaled <- scale(i_x_all)
```

Because the `kmeans()` function shows cluster IDs instead of the species names, I wanted to asign a species to the cluster ID to make it easier to understand the results.

```
i_majority_map_predict <- function(i_cluster_id, i_y) {
  i_tab <- table(i_cluster_id, i_y)
  i_winners <- apply(i_tab, 1, function(row) names(row)[which.max(row)])
  i_y_pred <- i_winners[as.character(i_cluster_id)]
  return(i_y_pred)
}
```

Now, I actually run the `kmeans()` function, I have never used it before so I used the built in `help(kmeans)` to show me the proper structure of the function, and how to undestand the different arguments that go into it.

```
i_score_kmeans <- function(i_x_scaled, i_y, i_k, nstart = 50, seed = 16) {
  set.seed(seed)
  i_fit <- kmeans(i_x_scaled, centers = i_k, nstart = nstart)
  i_y_pred <- i_majority_map_predict(i_fit$cluster, i_y)
  i_acc <- mean(i_y_pred == i_y)
  list(acc = i_acc, fit = i_fit)
}
```

Then, I systematically tested every combination of the four predictors and a range of cluster counts, then tracked how well each kmeans clustering aligned with the true flower species.
I stored the results so I could more easily identify which predictors and value of k produced the best clustering performance.

```
i_predictor_names <- colnames(i_x_all_scaled)
i_subsets <- unlist(
  lapply(1:length(i_predictor_names),
         function(m) combn(i_predictor_names, m, simplify = FALSE)),
  recursive = FALSE
)
i_k_grid <- 2:6
i_results <- data.frame(
  predictors = character(),
  k = integer(),
  accuracy = numeric(),
  tot_withinss = numeric(),
  stringsAsFactors = FALSE
)
i_best <- list(acc = -Inf, predictors = NULL, k = NA, fit = NULL)
```

You might wonder "why did you test k values 2:6 when you know there are 3 species?"
To that I say, good question! Personally, I thought that testing `k = (2,3,4,5,6)` to show that `k = 3` performs the best, rather than assuming it, would be good practice for trickier data where the true number of clusters is not obvious in advance.

```
for (i_vars in i_subsets) {
  i_x_sub <- i_x_all_scaled[, i_vars, drop = FALSE]

  for (i_k in i_k_grid) {
    i_out <- i_score_kmeans(i_x_sub, i_y_true, i_k = i_k, nstart = 50)

    i_results <- rbind(
      i_results,
      data.frame(
        predictors = paste(i_vars, collapse = ", "),
        k = i_k,
        accuracy = i_out$acc,
        tot_withinss = i_out$fit$tot.withinss,
        stringsAsFactors = FALSE
      )
    )

    if (i_out$acc > i_best$acc) {
      i_best$acc <- i_out$acc
      i_best$predictors <- i_vars
      i_best$k <- i_k
      i_best$fit <- i_out$fit
    }
  }
}
```

This next section loops over every combination of predictors and every tested value of k, running kmeans for each pairing.
For each run, it records the predictors used, the chosen k, the resulting accuracy, and the sum of squares.
At the same time, it compares each result to the current best model and updates the stored "best" model whenever a higher accuracy is found.

```r
i_results <- i_results[order(-i_results$accuracy, i_results$tot_withinss), ]
print(head(i_results, 10))
```

```
##                                predictors k  accuracy tot_withinss
## 20                           Petal.Width 6 0.9600000     2.222153
## 19                           Petal.Width 5 0.9600000     2.896075
## 17                           Petal.Width 3 0.9600000     8.456319
## 47              Petal.Length, Petal.Width 3 0.9600000    17.906783
## 50              Petal.Length, Petal.Width 6 0.9533333     7.123243
## 49              Petal.Length, Petal.Width 5 0.9533333     9.077973
## 18                           Petal.Width 4 0.9466667     4.753221
## 12                          Petal.Length 3 0.9466667     7.867216
## 48              Petal.Length, Petal.Width 4 0.9466667    12.201483
## 70 Sepal.Width, Petal.Length, Petal.Width 6 0.9400000    45.671029
```

```r
cat("\nBEST SETUP\n")
```

```
##
## BEST SETUP
```

```r
cat("Predictors:", paste(i_best$predictors, collapse = ", "), "\n")
```

```
## Predictors: Petal.Width
```

```r
cat("k:", i_best$k, "\n")
```

```
## k: 3
```

```r
cat("Accuracy:", round(i_best$acc, 4), "\n\n")
```

```
## Accuracy: 0.96
```

Here, I sort all the tested models by highest accuracy first and then by lowest sum of squares, also printing the top results so it's easy to see which predictor and k combinations performed best overall.
Interestingly, I found that `k = 6`, `k = 5` and `k = 3` using the single predictor of `Petal.Width`, and `k = 3` with two predictors `Petal.Width` and `Petal.Length` had the highest accuracy, tied at [0.9600000] (96%).
Immediately, I will rule out `k = 3` using two predictors, because I have now proven I can get the same accuracy using only one predictor, so I had to decide between `k = (3,5,6)` using `Petal.Width` as the only predictor. As we learned in the lesson, the sum of squares gets smaller the larger k gets (generally), so it makes sense that `k = (5,6)` has a smaller sum of squares.
For this dataset, knowing that there are 3 species, and knowing that a higher k value would not improve accuracy, and might cause confusion to a reader (peer), I selected `k = 3` to be the best pick.
I would say though, as a side note, this could be used in other scenarios where you are not only splitting species, but maybe trying to identify genes? Could be a long-shot, but you could say, for example, you want to cluster togther the 3 species, but you want it to be "species A likely with AAA gene." This could be determined by the Petal.Width, where a width over a certain mm or cm (the data itself does not say what the widths are measered by) possibly has gene AAA.

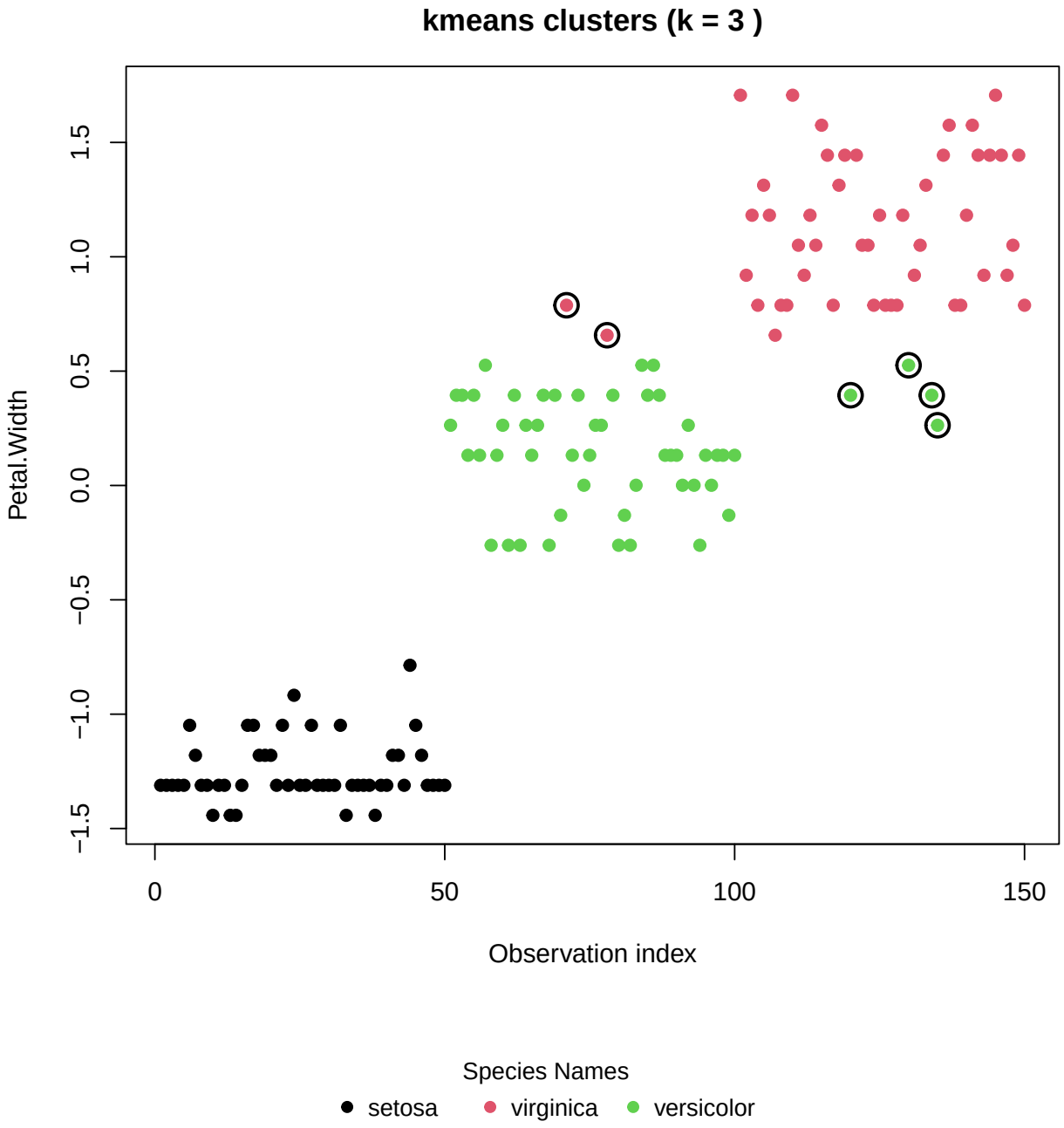
```r
i_x_best <- i_x_all_scaled[, i_best$predictors, drop = FALSE]
i_y_best_pred <- i_majority_map_predict(i_best$fit$cluster, i_y_true)

print(table(Predicted = i_y_best_pred, Actual = i_y_true))
```

```
##             Actual
## Predicted    setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         4
##   virginica       0          2        46
```

Here, I extracted the predictor data used by the best model and convert its kmeans cluster assignments into predicted species labels. I then compare those predicted labels to the true species labels in a table, which shows where the clustering was correct and where misclassifications occurred.
It seems I was able to separate [setosa] perfectly, no misclassifications. The errors occured between [versicolor] and [virginica], with [2] [versicolor] identified as [virginica], and [4] [virginica] being identified as [versicolor]. I believe this is because [virginica] and [versicolor] have petal widths that are, in general, much closer together, so it's a lot more difficult to separate them.

```r
if (ncol(i_x_best) == 1) {

  opar <- par(no.readonly = TRUE)
  on.exit(par(opar), add = TRUE)

  layout(matrix(c(1, 2), nrow = 2), heights = c(4, 1))

  par(mar = c(4, 4, 3, 1))
  plot(
    seq_along(i_x_best[, 1]),
    i_x_best[, 1],
    col = i_best$fit$cluster,
    pch = 19,
    xlab = "Observation index",
    ylab = i_best$predictors[1],
    main = paste("kmeans clusters (k =", i_best$k, ")")
  )

  i_mis <- which(i_y_best_pred != i_y_true)
  points(i_mis, i_x_best[i_mis, 1], pch = 1, cex = 2, lwd = 2)

  i_cluster_species <- tapply(i_y_best_pred, i_best$fit$cluster, function(x) x[1])

  par(mar = c(0, 0, 0, 0))
  plot.new()
  legend(
    "center",
    legend = i_cluster_species,
    col = as.numeric(names(i_cluster_species)),
    pch = 19,
    title = "Species Names",
    horiz = TRUE,
    bty = "n",
    cex = 0.9
```

```
  )

  layout(1)
}
```



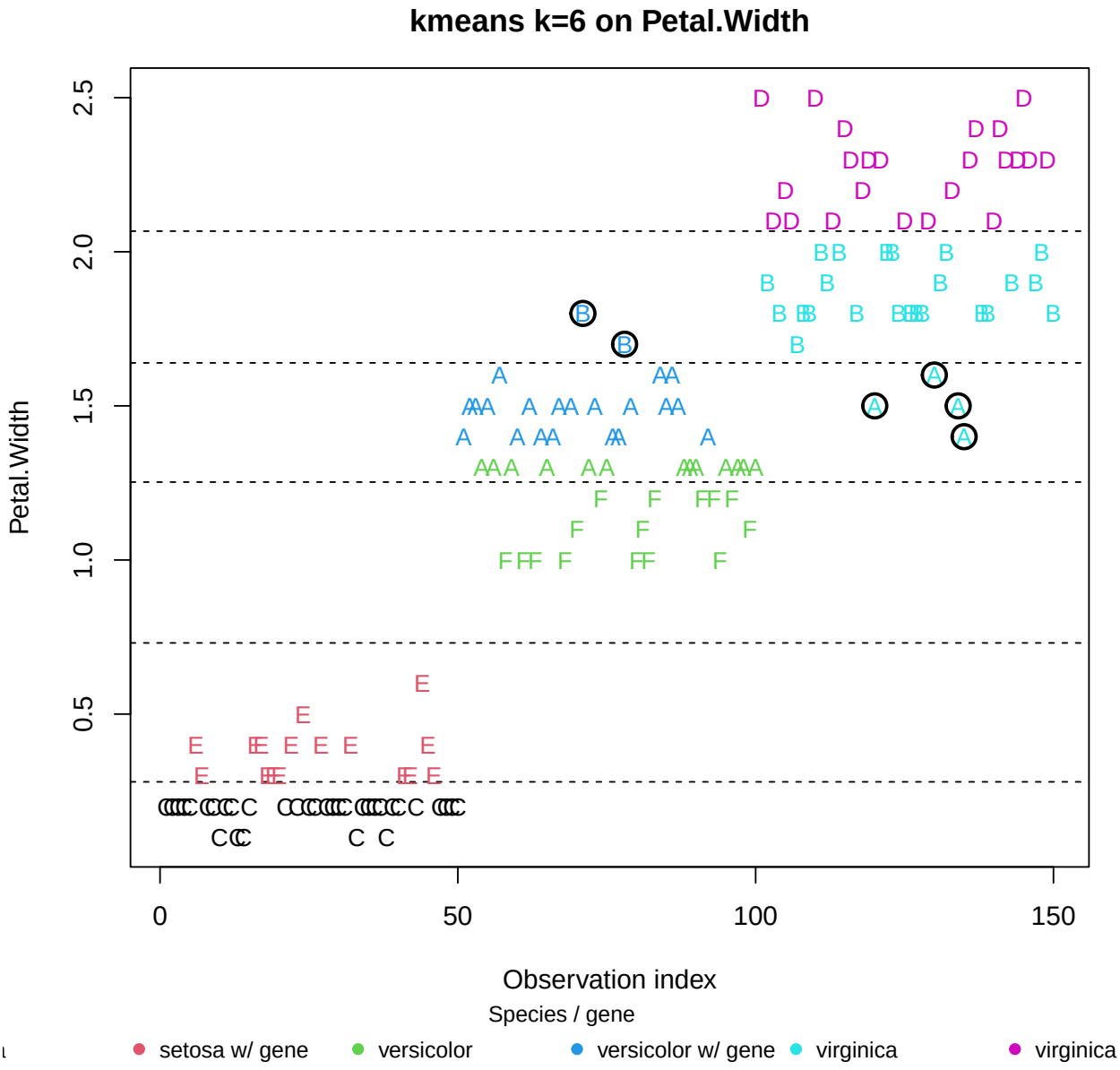
To better show the results, I made this plot which shows the kmeans clusters for the best model using a single predictor (`Petal.Width`), with points colored by their assigned cluster.
The circled points highlight the six observations that were misclassified when cluster labels were compared to the true species, making it easy to see where the model makes mistakes between [versicolor] and [virginica].

Just For Fun

Listen... I studied genetics at Texas A&M University, so I couldn't help but try out my little hypothesis of using `k = 6` `Petal.Width` to try to separate out the species by imaginary gene, so "what species is this flower" and aldo "does this flower have the imaginarty gene that makes the petal width larger than average?" This is completely unnecessary and silly, so please just skip it if you don't want to read it.
Because `Petal.Width` shows strong separation both between and within species, it is the most plausible single predictor for an imagined gene that increases trait magnitude across all three species. Please note I am using all the data I have already found above, so I won't explain it in detail here. Although I did not directly copy/paste any one line of code directly, I did use a lot of references to figure out how to use the Voronoi Diagram within R, including AI, and https://stackoverflow.com/questions/66946196/how-to-build-voronoi-polygons-for-point-coordinates-in-r, and would not have been able to do this without those references.

```
opar <- par(no.readonly = TRUE)
on.exit(par(opar), add = TRUE)

i_pw_species_mean <- ave(i_df$Petal.Width, i_df$Species, FUN = mean)
i_gene <- i_df$Petal.Width > i_pw_species_mean

i_species_gene <- ifelse(
  i_gene,
  paste(i_df$Species, "w/ gene"),
  as.character(i_df$Species)
)

i_species_gene <- factor(
  i_species_gene,
  levels = c("setosa", "setosa w/ gene",
             "versicolor", "versicolor w/ gene",
             "virginica", "virginica w/ gene")
)

set.seed(16)
i_pw_scaled <- as.numeric(scale(i_df$Petal.Width))
i_k6_fit <- kmeans(i_pw_scaled, centers = 6, nstart = 50)

i_cluster_letters <- LETTERS[i_k6_fit$cluster]

i_centers <- sort(as.numeric(i_k6_fit$centers))
i_bounds_scaled <- (i_centers[-1] + i_centers[-length(i_centers)]) / 2
i_pw_center <- attr(scale(i_df$Petal.Width), "scaled:center")
i_pw_scale  <- attr(scale(i_df$Petal.Width), "scaled:scale")
i_bounds_pw <- i_bounds_scaled * i_pw_scale + i_pw_center

layout(matrix(c(1, 2), nrow = 2), heights = c(4, 1))

par(mar = c(4, 4, 3, 1))
plot(
  seq_along(i_df$Petal.Width),
  i_df$Petal.Width,
  type = "n",
  xlab = "Observation index",
  ylab = "Petal.Width",
```

```
  main = "kmeans k=6 on Petal.Width"
)

abline(h = i_bounds_pw, lty = 2)

text(
  seq_along(i_df$Petal.Width),
  i_df$Petal.Width,
  labels = i_cluster_letters,
  col = as.integer(i_species_gene),
  cex = 0.9
)

i_mis <- which(i_y_best_pred != i_y_true)
points(i_mis, i_df$Petal.Width[i_mis], pch = 1, cex = 2, lwd = 2)

par(mar = c(0, 0, 0, 0))
plot.new()

legend(
  "top",
  legend = levels(i_species_gene),
  col = seq_along(levels(i_species_gene)),
  pch = 19,
  title = "Species / gene",
  horiz = TRUE,
  bty = "n",
  cex = 0.85
)

legend(
  "bottom",
  legend = paste("Cluster", LETTERS[1:6]),
  title = "kmeans (k=6)",
  horiz = TRUE,
  bty = "n",
  cex = 0.85
)
```


kmeans k=6 on Petal.Width
Petal.Width
2.5
2.0
1.5
1.0
0.5
0
50
100
150
Observation index
Species / gene
setosa w/ gene
versicolor
versicolor w/ gene
virginica
virginica
kmeans (k=6)
Cluster A
Cluster B
Cluster C
Cluster D
Cluster E
Cluster F

```
layout(1)
```