Homework 2

2025-09-04

Boilerplate code

First, we load the necessary libraries:

```
library(kernlab, include.only = c("ksvm", "predict"))
library(kknn, include.only = 'kknn')
library(caret, include.only = 'confusionMatrix')
library(dplyr)
library(pracma, include.only = 'logspace')
library(huxtable) # pretty print dataframes
library(tidyr)
library(ggplot2)

# set seed for reproducibility
set.seed(42)
```

I wrote a function for plotting confusion matrices based on[1].

```
plot_confusion_matrix <- function(cm)
{
  t <- cm$table
  plt <- as.data.frame(t)
  labels <- colnames(t)

  plt$Prediction <- factor(plt$Prediction, levels=rev(levels(plt$Prediction)))


  ggplot(plt, aes(Prediction,Reference, fill= Freq)) +
          geom_tile() + geom_text(aes(label=Freq)) +
          scale_fill_gradient(low="white", high="#009194") +
          labs(x = "Reference",y = "Prediction") +
          scale_x_discrete(labels=rev(labels)) +
          scale_y_discrete(labels=labels)

}
```

[1] https://stackoverflow.com/a/64539733/1543042

Question 3.1

Using the same data set (`credit_card_data.txt` or `credit_card_data-headers.txt`) as in Question 2.2, use the `ksvm` or `kknn` function to find a good classifier:

(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and
(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

We start by loading the data:

```
data <- read.csv("./data 3.1/credit_card_data-headers.txt", sep = "\t" )
```

And split off 20% of the data into a test set:

```
test_fraction = 0.20
data_size = nrow(data)
test_count <- floor(test_fraction * data_size)
test_indicies <- sample(seq(data_size), size = test_count)

test_set <- data[test_indicies,]
train_val_set <- data[-test_indicies,]
```

Let's define convenience functions for features (`X`) and target (`Y`).

```
X <- function(df) { subset(df, select = -c(R1)) }
Y <- function(df) { df$R1 }
```

Let's wrap `kknn` and `ksvm` in convenience functions so they have a common input and output. This will reduce the repetition later.

```
knn <- function(
    train,
    x_test,
    param
) {
  # kknn has the weird syntax
  model <- kknn(
    R1 ~ .,
    train = train,
    test  = x_test,
    k = param,
    scale = TRUE
  )

  vote_total <- fitted(model)

  round(vote_total)
}
```

```
svm <- function(
    train,
    x_test,
    param,
    type="C-svc",
    kernel="vanilladot"
) {
  # to make a common interface this should return inference
  x_train <- X(train)
  y_train <- Y(train)

  # train the model
  invisible(capture.output(
    model <- ksvm(
        as.matrix(x_train),
        as.vector(y_train),
        C=param,
        type=type,
        kernel=kernel,
        scaled=TRUE,
    )
  ))

  # perform and return inference
  predict(model, x_test)
}
```

Question 3.1.A

With the train-validation set we generated above, we can split it into the 10 folds.

We need a function to split datasets into n folds for cross-validation. While a package might exist, I implemented a simple version[2].

```
rows <- nrow(train_val_set)
ind <- sample(seq(rows))
cross_val <- split(
  train_val_set,
  cut(ind, breaks = 10, label = FALSE)
)
```

Let's define the cross validation function. We can see that for each iteration of the loop, the training and validation dataframes are being created, and fed into the model. The results are then being collected to create a global confusion matrix.

```
split_apply_combine <- function(model, cross_val, param)
{
  Y_PRED <- list()
  Y_VAL <- list()

  for (i in seq(length(cross_val)))
  {
    train <- bind_rows(cross_val[-i])
    val <- bind_rows(cross_val[i])

    x_val <- X(val)
    Y_VAL <- unlist(list(Y_VAL, Y(val)))

    y_pred <- model(train, x_val, param)
    Y_PRED <- unlist(list(Y_PRED, y_pred))
  }

  confusionMatrix(data=as.factor(Y_PRED), reference=as.factor(Y_VAL))
}
```

[2] https://stackoverflow.com/a/60613546/1543042

SVM

Based on the knowledge from Homework 1, we choose $C = [10^{-3}, 10^{2}]$ for the SVM.

```
results <- data.frame(
  parameter = numeric(0),
  accuracy = numeric(0)
)

for (c in c(1E-3, 1E-2, 1E-1, 1, 10, 100))
{
  # get the confusion matrix
  iteration_results <- split_apply_combine(svm, cross_val, param=c)

  # load results into the dataframe
  results <- bind_rows(
    results,
      data.frame(
        parameter = c,
        accuracy = as.double(iteration_results[3]$overall["Accuracy"])
      )
    )
}
```

Now, let's look at the results.

```
ht <- as_hux(results)

ht <- set_all_borders(ht, 0.4) |>
      set_bold(1, everywhere, TRUE) |>        # bold headers
      set_align(everywhere, 2:ncol(ht), 'center') |>
      set_number_format(everywhere, 2:ncol(ht), 3)

ht
```

parameter	accuracy
0.001	0.823
0.01	0.872
0.1	0.872
1	0.872
10	0.872
100	0.872

Just as we saw in Homework 1 there is a wide range of stability, for the sake of a value let's choose $C = 0.1$. Doing so we can retrain the model on the entire train-validation set and then perform inference on the test set.

```
x_test <- X(test_set)
y_test <- Y(test_set)

y_pred <- svm(train_val_set, x_test, param=0.1)

conf_mat <- confusionMatrix(data=as.factor(y_pred), reference=as.factor(y_test))
```

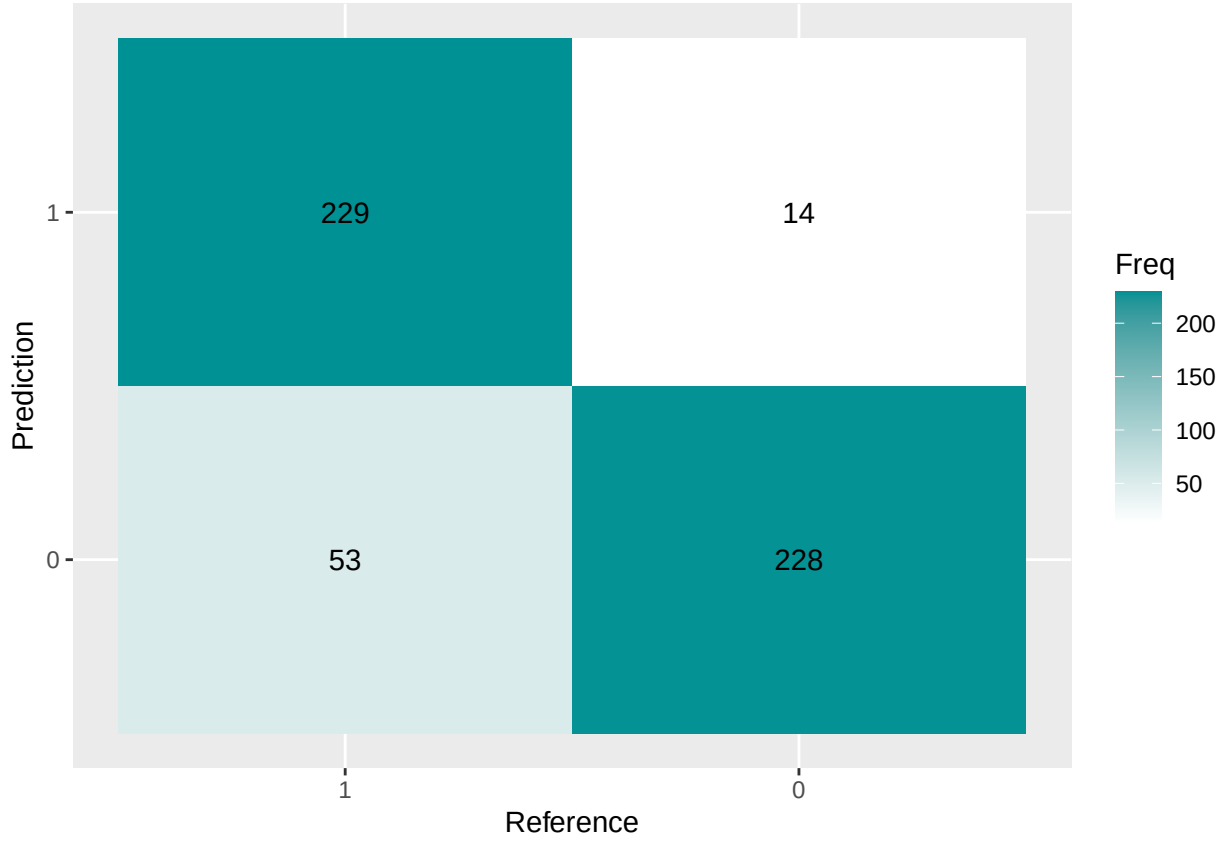
The accuracy of the SVM $C = 0.1$ classifier is

```
print(format(conf_mat[3]$overall["Accuracy"], digits=3))
```

```
## Accuracy
##  "0.823"
```

And the confusion matrix is

```
plot_confusion_matrix(iteration_results[2])
```



KNN

Doing the same for the KNN for parameters in the range $k = [6, 20]$

```
results <- data.frame(
  parameter = numeric(0),
  accuracy = numeric(0)
)

for (k in seq(6,20,2))
{
  # get the confusion matrix
  iteration_results <- split_apply_combine(knn, cross_val, param=k)

  # load results into the dataframe
  results <- bind_rows(
    results,
      data.frame(
        parameter = k,
        accuracy = as.double(iteration_results[3]$overall["Accuracy"])
      )
    )
}
```

Now, let's look at the results.

```
ht <- as_hux(results)

ht <- set_all_borders(ht, 0.4) |>
      set_bold(1, everywhere, TRUE) |>        # bold headers
      set_align(everywhere, 1:ncol(ht), 'center') |>
      set_number_format(everywhere, 2:ncol(ht), 3)

ht
```

parameter	accuracy
6	0.855
8	0.851
10	0.851
12	0.851
14	0.851
16	0.845
18	0.844
20	0.844

It appears that $k = 6$ performs slightly better than the others so let's choose that. We can retrain the model on the entire train-validation set and then perform inference on the test set.

```
x_test <- X(test_set)
y_test <- Y(test_set)

y_pred <- knn(train_val_set, x_test, param=6)

conf_mat <- confusionMatrix(data=as.factor(y_pred), reference=as.factor(y_test))
```

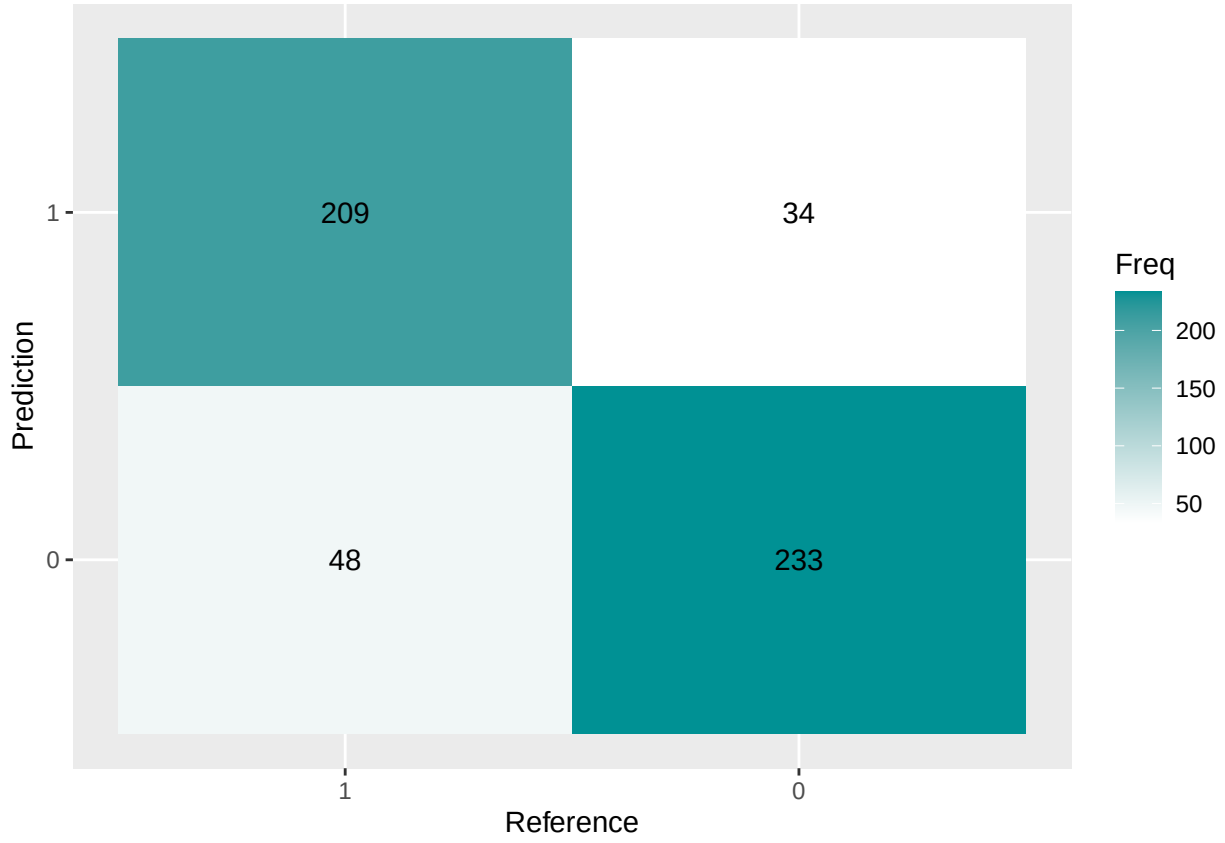
The accuracy of the KNN $k = 6$ classifier is

```
print(format(conf_mat[3]$overall["Accuracy"], digits=3))
```

```
## Accuracy
##  "0.831"
```

And the confusion matrix is

```
plot_confusion_matrix(iteration_results[2])
```



Results

Just as we would expect, our model is not perfectly generalizable and so we see a lower accuracy in both the classifiers examining the test set than we saw in the cross-fold validation set.

Question 3.1.B

We will use the same test set as from Question 3.1.A; however, we now need to split the train-validation set into two parts. To have the train-validation-test be 60%:20%:20% we need to rescale our fraction to account for the already removed portion.

```
val_fraction = 0.20
rescaled_val_frac = val_fraction / ( 1 - test_fraction )
train_val_size = nrow(train_val_set)
val_count <- floor(rescaled_val_frac * train_val_size)
val_indicies <- sample(seq(train_val_size), size = val_count)

val_set <- train_val_set[val_indicies,]
train_set <- train_val_set[-test_indicies,]
```

Because we know that there is a large range of stability in the SVM, I am going to choose to only analyze the KNN classifier. First we need to choose a hyperparameter for the classifier, so we perform a parameter sweep of the space.

```
results <- data.frame(
  parameter = numeric(0),
  accuracy = numeric(0)
)

x_val <- X(val_set)
y_val <- Y(val_set)

for (k in seq(6,20,2))
{
  y_pred <- knn(train_set, x_val, param=k)
  conf_mat <- confusionMatrix(data=as.factor(y_pred), reference=as.factor(y_val))

  results <- bind_rows(
    results,
      data.frame(
        parameter = k,
        accuracy = as.double(iteration_results[3]$overall["Accuracy"])
      )
    )
}
```

Now, looking at the results we see that all the classifiers have the same accuracy, so we will choose is $k = 10$

```
ht <- as_hux(results)

ht <- set_all_borders(ht, 0.4) |>
      set_bold(1, everywhere, TRUE) |>       # bold headers
      set_align(everywhere, 1:ncol(ht), 'center') |>
      set_number_format(everywhere, 2:ncol(ht), 3)

ht
```

parameter	accuracy
6	0.844
8	0.844
10	0.844
12	0.844
14	0.844
16	0.844
18	0.844
20	0.844

Let's see if that generalizes to the test set.

```
x_test <- X(test_set)
y_test <- Y(test_set)

y_pred <- knn(train_set, x_val, param=10)
conf_mat <- confusionMatrix(data=as.factor(y_pred), reference=as.factor(y_val))
```

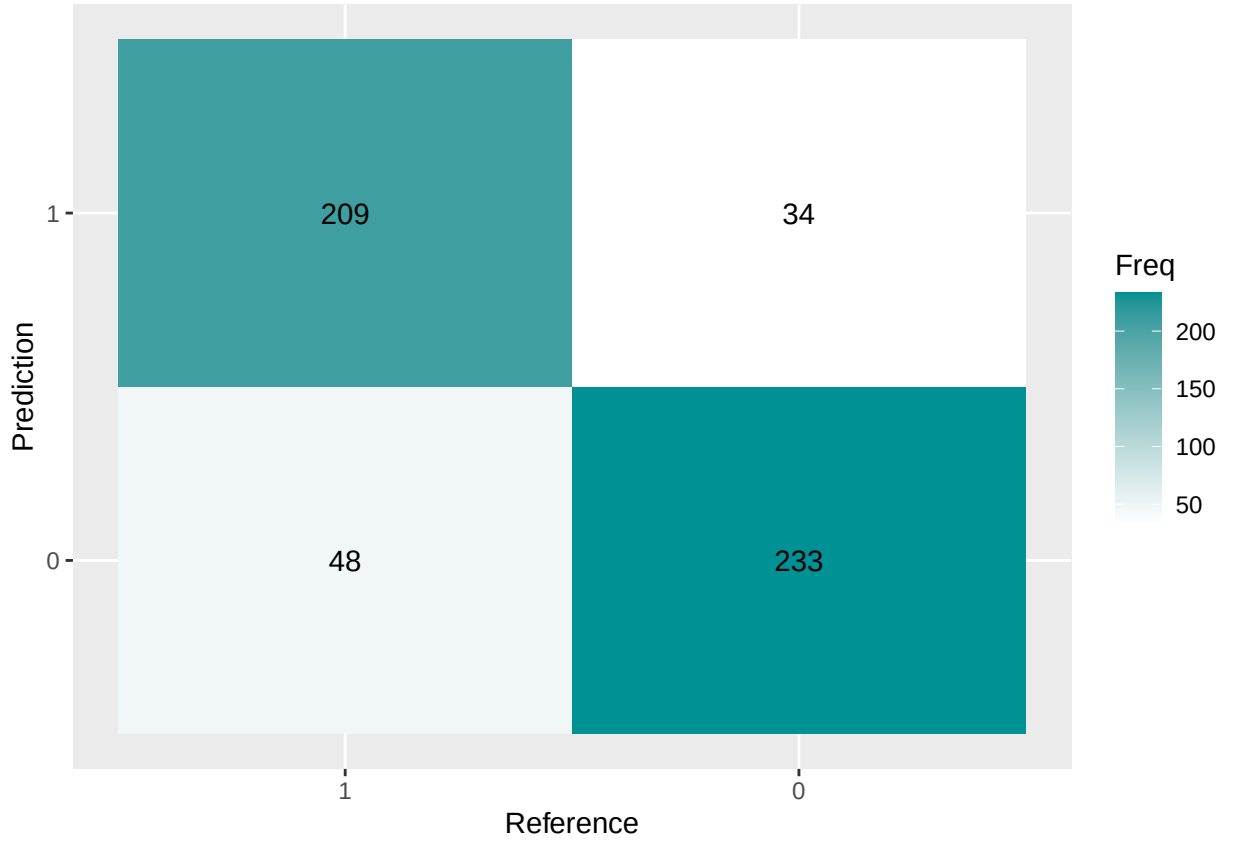
The accuracy of the KNN $k = 10$ classifier is

```
format(conf_mat[3]$overall["Accuracy"], digits=3)
```

```
## Accuracy
##  "0.878"
```

And the confusion matrix is

```
plot_confusion_matrix(iteration_results[2])
```



Interestingly we see that the test set actually had a higher accuracy than the validation set. This can happen due to how the data was distributed between the three sets.

Question 4.1

> Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

One situation that we interact with on a daily basis are recommendation systems. For example Netflix wants to recommend movies and shows that I will want to watch. To do that they want to group similar people together so titles one person in the cluster likes would probably be a good recommendation for someone else. Some predictors they might use are:

1. Demographic information (sex, age, etc.)
2. Location
3. Previously viewed titles and ranking info

Question 4.2

The *iris* data set `iris.txt` contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). *The response values are only given to see how well a specific method performed and should not be used to build the model.*

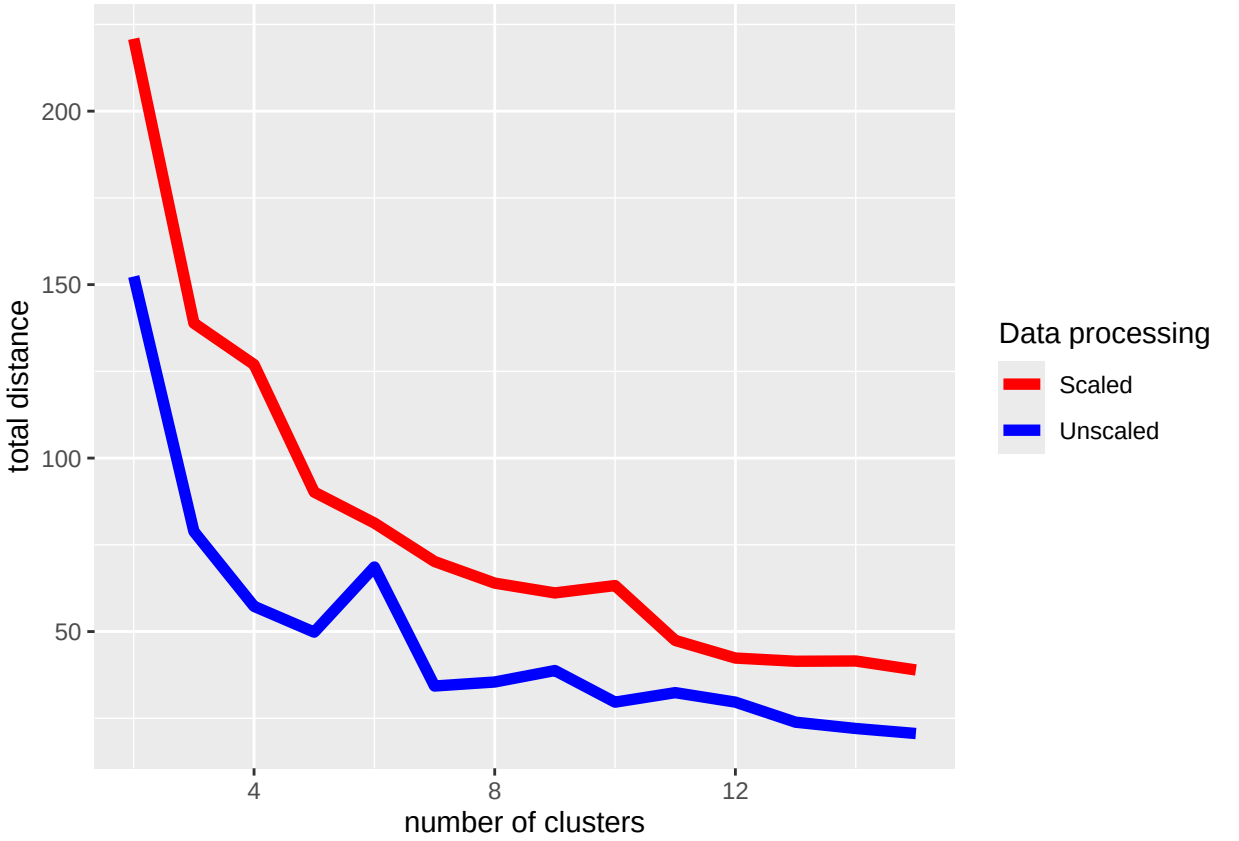
Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

Scaling investigation

Let's load the dataset in and sweep through the parameter space using `kmeans` then plot the total distance to find the elbow. Note that the total distance for the scaled iris dataset is actually larger than for the unscaled and so I will use the unscaled for the rest of the analysis.

```
data("iris")

# including 1 just so index matches number of clusters
num_clusters <- seq(1,15)

iris_without_label <- subset(iris, select = -c(Species))
scaled_iris = scale(iris_without_label)

parameter_sweep <- function(clusters) {
  kmeans(iris_without_label, centers=clusters)
}

total_distance_scaled <- lapply(
  num_clusters,
  function(clusters) {
    kmeans(scaled_iris, centers=clusters)$tot.withinss
  }
)

parameter_sweep_results <- lapply( num_clusters , parameter_sweep )

total_distance <- lapply( parameter_sweep_results, function(x) {x$tot.withinss} )

scaled_vs_unscaled <- data.frame(
  num_clusters = num_clusters,
  unscaled_iris = unlist(total_distance),
  scaled_iris = unlist(total_distance_scaled)
)[-1, ]

ggplot(scaled_vs_unscaled, aes(num_clusters)) +
  geom_line(aes(y = unscaled_iris, colour = "Unscaled"), linewidth = 2) +
  geom_line(aes(y = scaled_iris, colour = "Scaled"), linewidth = 2) +
  scale_colour_manual(
    name = "Data processing",
    values = c("Unscaled" = "blue", "Scaled" = "red")
  ) +
  scale_x_continuous(name="number of clusters") +
```

```
  scale_y_continuous(name="total distance")
```



Number of clusters

Furthermore, notice the distinct kink at $k = 4$ and a second less pronounced kink at $k = 7$ if this were truly unlabeled data one of these would be our choice; however, since we have the labels let's map each cluster to it's modal category and see how what the accuracy is. This involves several steps

1. Matching the per row the cluster and true label,
2. Counting the frequency of each label,
3. Finding the most frequent true label for each cluster
4. Using the most frequent label as the mapping from cluster to label

```
get_true_mapping <- function(clustered) {
  data.frame( cluster=clustered$cluster, true_label=iris$Species )
}

get_true_mapping_frequency <- function(clustered) {
  true_label <- get_true_mapping(clustered)

  cluster_frequency <- true_label %>%
      rename(inferred_label = true_label) %>%
      count(cluster, inferred_label)

  cluster_frequency
}
```

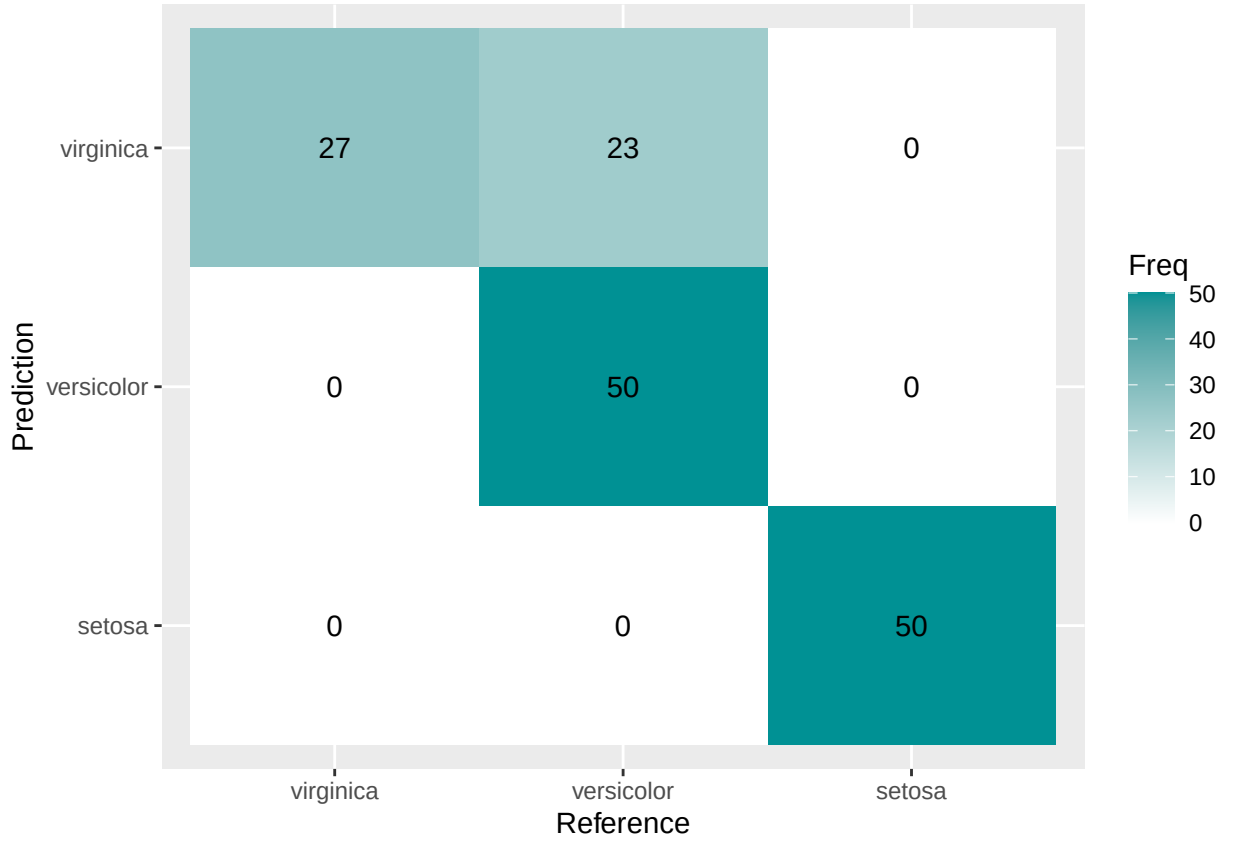
```
infer_cluster_mapping <- function(clustered) {
  cluster_frequency <- get_true_mapping_frequency(clustered)

  inferred_mapping <- cluster_frequency %>%
      group_by(cluster) %>%
      filter(n == max(n)) %>%
      select(cluster, inferred_label)

  inferred_mapping
}

get_cluster_label <- function(clustered) {
  true_label <- get_true_mapping(clustered)
  inferred_mapping <- infer_cluster_mapping(clustered)

  labeled_cluster <- merge(true_label, inferred_mapping)
  labeled_cluster
}
```

From this we can convert the clusters into labels and look at accuracy.

```
parameter_sweep_labeled_clusters <- lapply(
  parameter_sweep_results, get_cluster_label
)

parameter_sweep_labeled_confusion_matrix <- lapply(
  parameter_sweep_labeled_clusters,
  function(x) {
    confusionMatrix(
      data=as.factor(x$inferred_label),
      reference=as.factor(x$true_label)
    )
  }
)
```

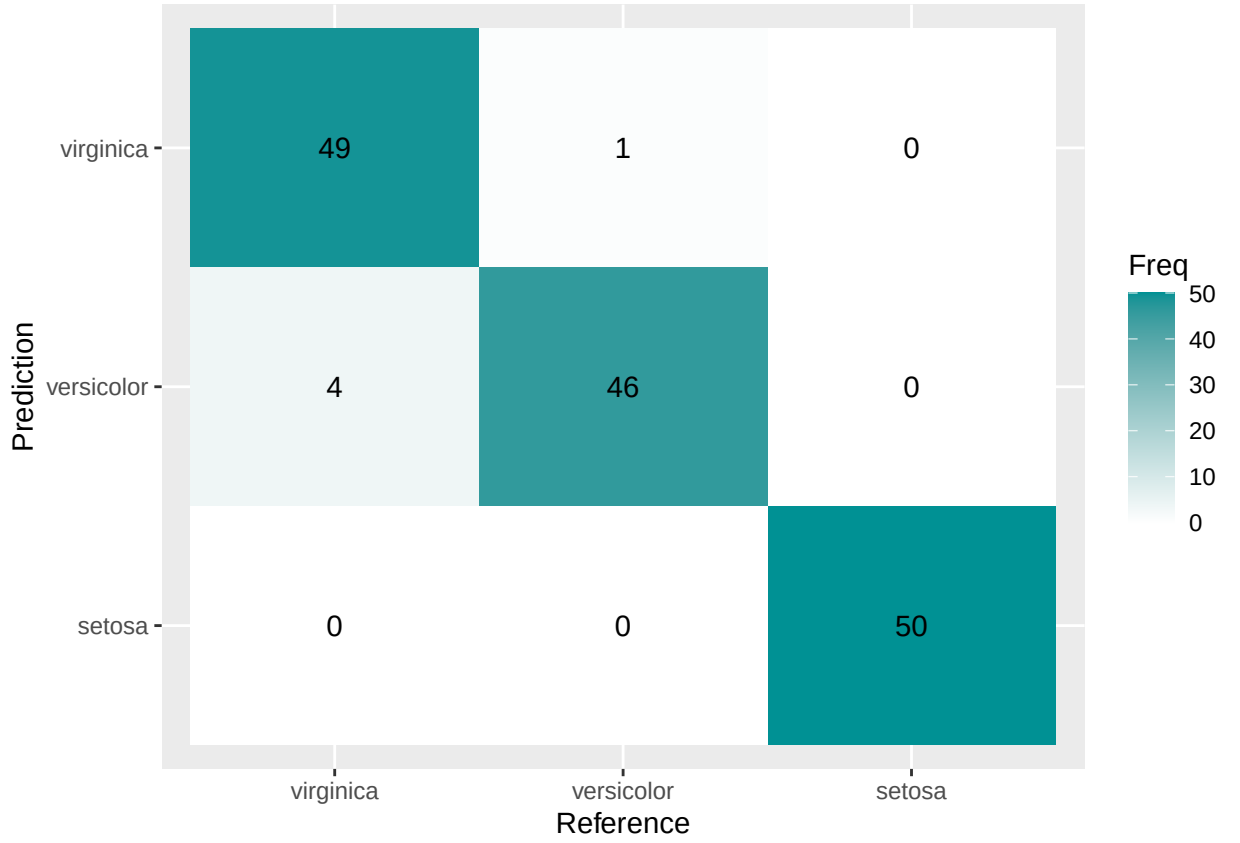
For example the $k = 4$ still has a fair amount of difficulty distinguishing between versicolor and virginica species.

```
plot_confusion_matrix(parameter_sweep_labeled_confusion_matrix[[4]])
```



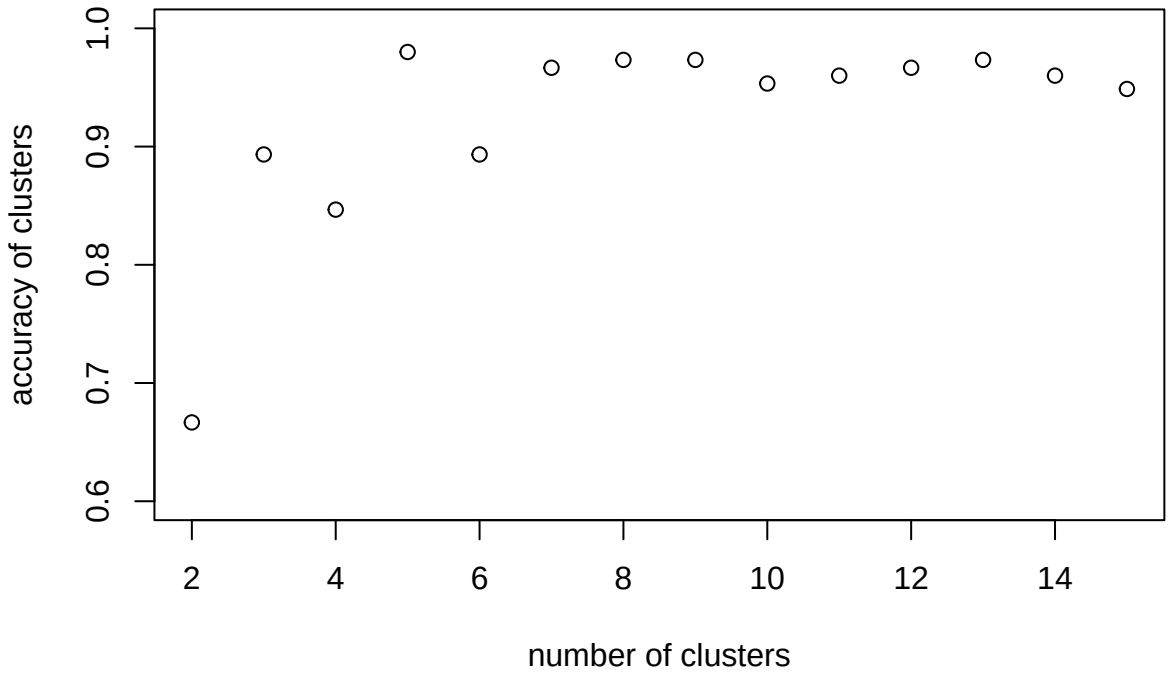
This is resolved at $k = 7$ and the accuracy is significantly improved.

```
plot_confusion_matrix(parameter_sweep_labeled_confusion_matrix[[7]])
```



Looking more systematically by plotting the accuracy, we can see that like we saw using the elbow method without using the labels. There is a large jump in the accuracy of the classifier for $k = 4$, then a smaller jump at $k = 7$. For $k > 7$ the change in accuracy appears to just be random fluctuations.

```
parameter_sweep_labeled_accuracy <- lapply(
  parameter_sweep_labeled_confusion_matrix,
  function(x) {
    as.numeric(x[3]$overall[1])
  }
)

# exclude 1 cluster from the plot because it's so bad
plot(
  num_clusters[-1],
  parameter_sweep_labeled_accuracy[-1],
  xlab="number of clusters",
  ylab="accuracy of clusters",
  ylim=c(0.6,1.0)
)
```



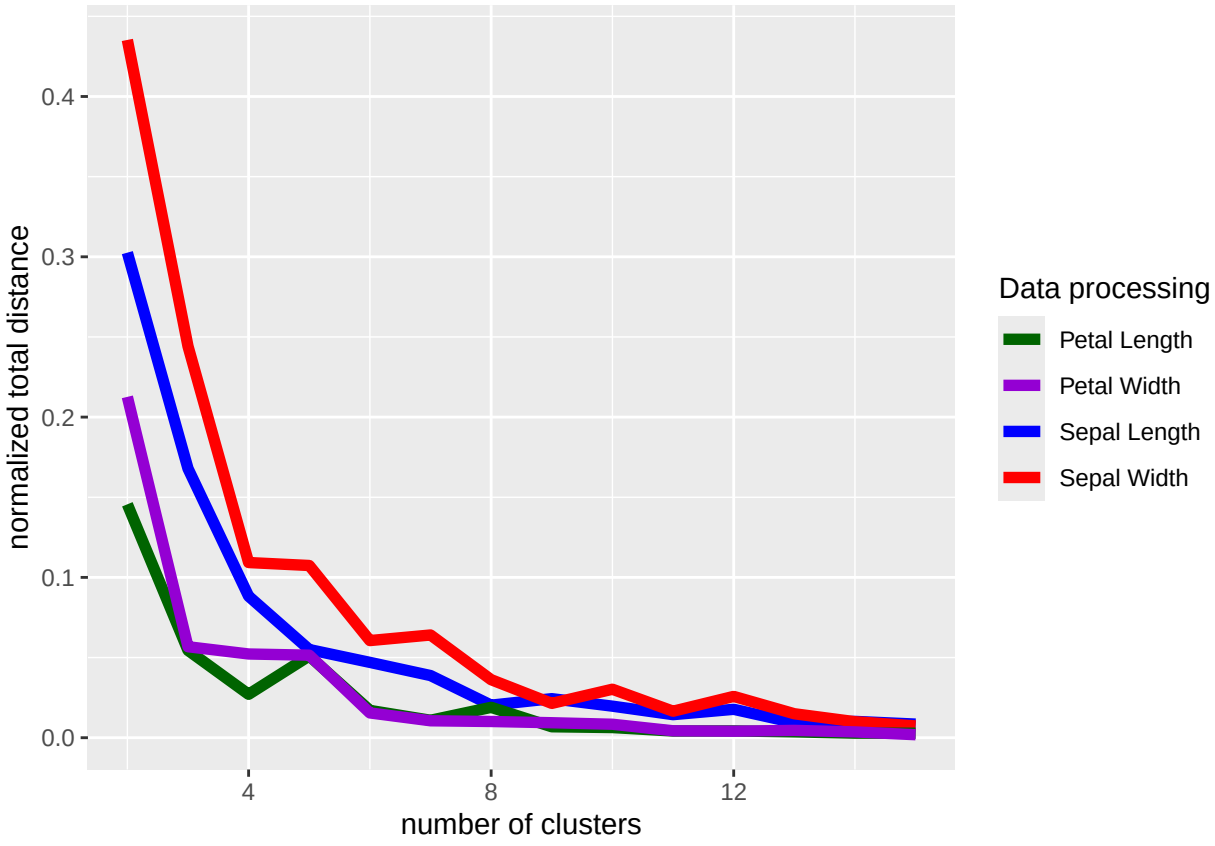
Feature importance

Since the dimensions are changing, a way to normalize the error term is to look at the ratio of the distance within the clusters compared to the total sum of squares[3] (this author actually looked at the opposite but they are just complements of each other). Let's look at the normalized error just including one factor.

```
results <- data.frame(
  num_clusters = num_clusters
)

for ( n in names(iris_without_label) ) {
  leave_one_out <- iris_without_label %>% select(all_of(n))


  parameter_sweep <- function(clusters) {
    r <- kmeans(leave_one_out, centers=clusters)

    r$tot.withinss / r$totss
  }

  total_distance <- lapply( num_clusters , parameter_sweep )

  results[[n]] = unlist(total_distance)
}
```

[3] https://andrea-grianti.medium.com/kmeans-parameters-in-rstudio-explained-c493ec5a05df

```
ggplot(results[-c(1),], aes(num_clusters)) +
  geom_line(aes(y = Sepal.Length, colour = "Sepal Length"), linewidth = 2) +
  geom_line(aes(y = Sepal.Width, colour = "Sepal Width"), linewidth = 2) +
  geom_line(aes(y = Petal.Length, colour = "Petal Length"), linewidth = 2) +
  geom_line(aes(y = Petal.Width, colour = "Petal Width"), linewidth = 2) +
  scale_colour_manual(
    name = "Data processing",
    values = c(
      "Sepal Length" = "blue",
      "Sepal Width" = "red",
      "Petal Length" = "darkgreen",
      "Petal Width" = "darkviolet"
    )
  ) +
  scale_x_continuous(name="number of clusters") +
  scale_y_continuous(name="normalized total distance")
```

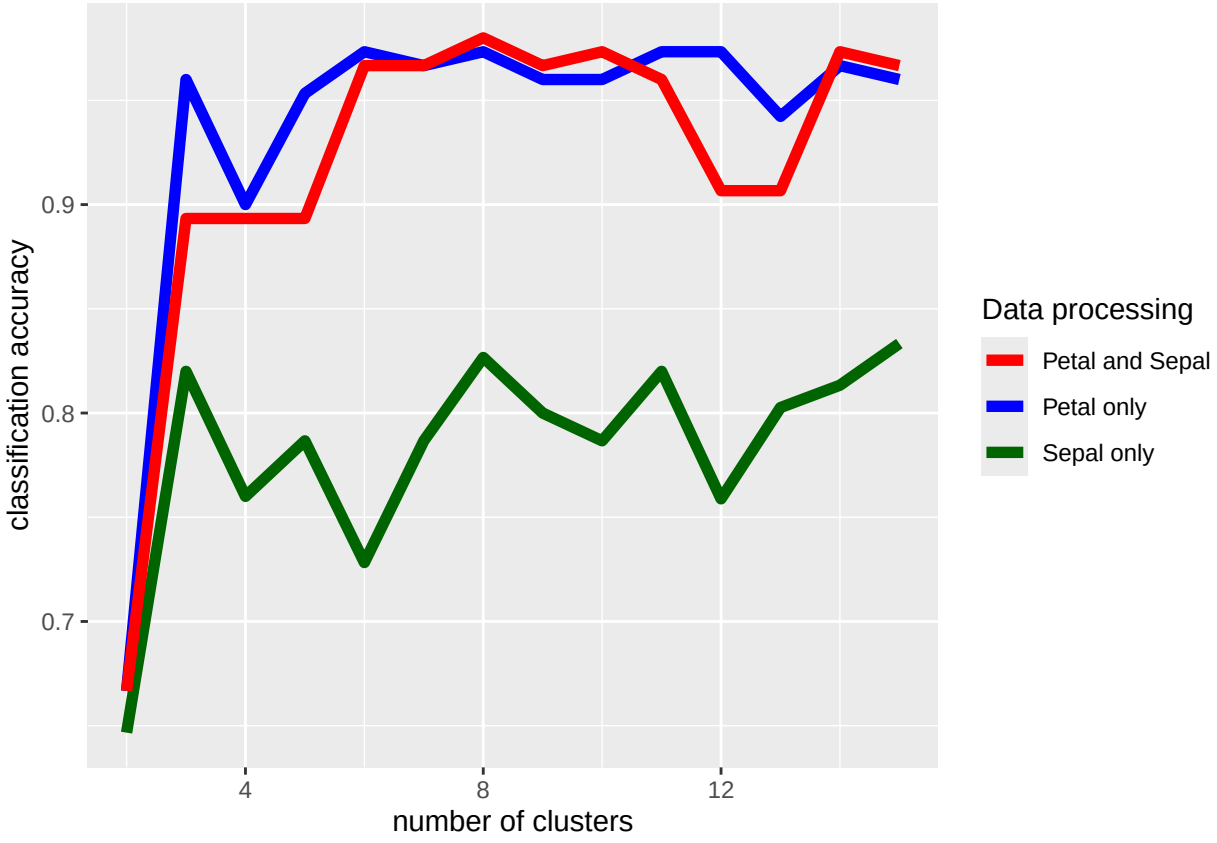


From this, we can see that the most important factors are petal length and width as they drive down the error much faster than the sepal dimensions. To test this, let's compare the accuracy of a petal only clustering model with the full petal and sepal model.

```
# combined function for determining accuracy
determine_accuracy <- function(dataset, clusters) {
  clustered <- kmeans(dataset, centers=clusters)

  labeled <- get_cluster_label(clustered)

  confusion_matrix <- confusionMatrix(
    data=as.factor(labeled$inferred_label),
    reference=as.factor(labeled$true_label)
  )

  as.numeric(confusion_matrix[3]$overall[1])
}

# define datasets
petals_only_dataset <- subset(iris, select = c(Petal.Length, Petal.Width))
sepal_only_dataset <- subset(iris, select = c(Sepal.Length, Sepal.Width))

petals_and_sepal_dataset <- subset(
  iris,
  select = c(Petal.Length, Petal.Width, Sepal.Length, Sepal.Width)
)

# get accuracy
results <- data.frame(
  num_clusters = num_clusters
)

results$petal_only_accuracy <- unlist(lapply(
  num_clusters,
  function(x) {determine_accuracy(petals_only_dataset, x)}
))

results$sepal_only_accuracy <- unlist(lapply(
  num_clusters,
  function(x) {determine_accuracy(sepal_only_dataset, x)}
))

results$petals_and_sepal_accuracy <- unlist(lapply(
  num_clusters,
  function(x) {determine_accuracy(petals_and_sepal_dataset, x)}
))
```

```
ggplot(
  results[-c(1),],
  aes(num_clusters)
) +
  geom_line(
    aes(y = petal_only_accuracy, colour = "Petal only"),
    linewidth = 2
  ) +
```

```
  geom_line(
    aes(y = sepal_only_accuracy, colour = "Sepal only"),
    linewidth = 2
  ) +
  geom_line(
    aes(y = petals_and_sepal_accuracy, colour = "Petal and Sepal"),
    linewidth = 2
  ) +
  scale_colour_manual(
    name = "Data processing",
    values = c(
      "Petal only" = "blue",
      "Sepal only" = "darkgreen",
      "Petal and Sepal" = "red"
    )
  ) +
  scale_x_continuous(name="number of clusters") +
  scale_y_continuous(name="classification accuracy")
```



As we suspected from the clustering analysis, the petal factors are significantly more important for segmenting the population than the sepal factors.

Comments

From this analysis we have come to several conclusions

1. The petal dimensions are much more important in predicting iris species than the sepal dimensions.

This can be shown using other methods also.[4]

2. To segment the population while maintaining a parsimonious model between 4 and 7 clusters are needed.

[4]https://scikit-learn.org/stable/_images/sphx_glr_plot_pca_iris_001.png