Part 1: Exercise 5b

Load Dataset

The dataset consists of 990 observations and 14 variables.

```
vowel <- read.arff("vowel.arff")
head(vowel, 3)
```

```
##   Train or Test Speaker Number  Sex Feature 0 Feature 1 Feature 2 Feature 3
## 1         Train         Andrew Male    -3.639     0.418    -0.670     1.779
## 2         Train         Andrew Male    -3.327     0.496    -0.694     1.365
## 3         Train         Andrew Male    -2.120     0.894    -1.576     0.147
##   Feature 4 Feature 5 Feature 6 Feature 7 Feature 8 Feature 9 Class
## 1    -0.168     1.627    -0.388     0.529    -0.874    -0.814   hid
## 2    -0.265     1.933    -0.363     0.510    -0.621    -0.488   hId
## 3    -0.707     1.559    -0.579     0.676    -0.809    -0.049   hEd
```

Drop columns

In the first part of our analysis, we will exclude the first 3 features, *Train or Test*, *Speaker Number* and *Sex* as well as the column *Class* and then run K-means clustering algorithm. We will tune k hyper-parameter using methods *elbow* and *silhouette*.

```
vowel_drop <- select(vowel, -1, -2, -3, -14)
head(vowel_drop, 3)
```

```
##   Feature 0 Feature 1 Feature 2 Feature 3 Feature 4 Feature 5 Feature 6
## 1    -3.639     0.418    -0.670     1.779    -0.168     1.627    -0.388
## 2    -3.327     0.496    -0.694     1.365    -0.265     1.933    -0.363
## 3    -2.120     0.894    -1.576     0.147    -0.707     1.559    -0.579
##   Feature 7 Feature 8 Feature 9
## 1     0.529    -0.874    -0.814
## 2     0.510    -0.621    -0.488
## 3     0.676    -0.809    -0.049
```

K Means

K-means algorithm works as presented below:

1. Choose groups in the feature plan randomly
2. Minimize the distance between the cluster center and the different observations (centroid). It results in groups with observations
3. Shift the initial centroid to the mean of the coordinates within a group.

4. Minimize the distance according to the new centroids. New boundaries are created. Thus, observations will move from one group to another
5. Repeat until no observation changes groups

Since M-means is based on distances of among data points, there few different measures it take into account. For example, *Euclidean* distance is the most common method, though we could also use *Manhattan* and *Minlowski* distances as well. Below, present the mathematical formulation of *Euclidean* distance.

$$distance(x, y) = \sum_{i}^{n} (x_i - y_i)^2 \tag{1}$$

Optimal K

One technique to choose the best k is called the *Elbow* method. This method uses within-group homogeneity or within-group heterogeneity to evaluate the variability. Another approach is called *Silhouette*. We will measure within groups sum of squares(variance) and the quality to clusters to determine the optimal value of k. For this, we will install the package *factoextra*.

Elbow

The elbow method is a method used to determime the number of clusters in a data set. It defines clusters in such a way that the total intra cluster variation (total within-cluster variation or total within-cluster sum of square) is minimized. We can formulate it as:

$$\min \sum_{k=1}^{k} W(C_k) \tag{2}$$

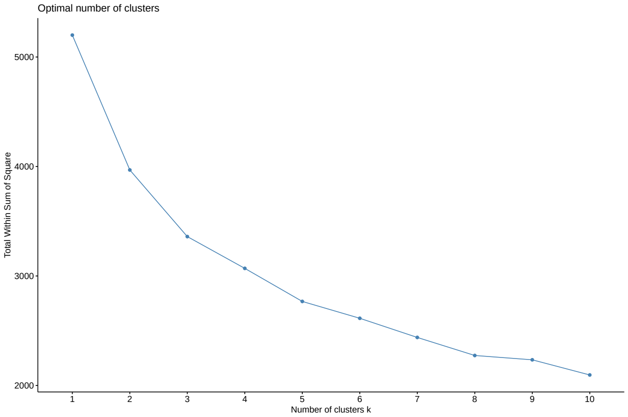
where C_k is the k^{th} cluster and $W(C_k)$ is the within-cluster variation. The total within-cluster sum of square measures the compactness of the clustering and we want it to be as *small* as possible

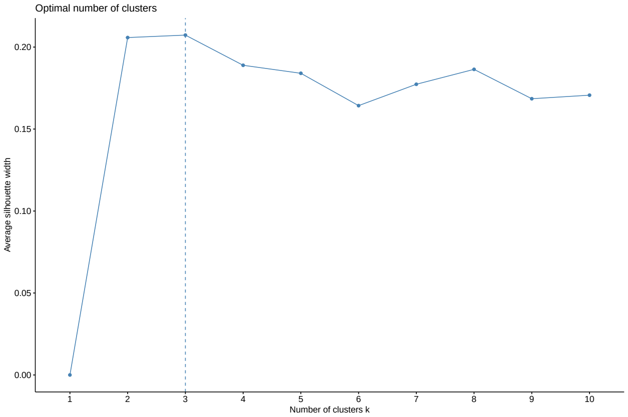
Silhouette

The average silhouette method measures the quality of a clustering. It determines how well each object lies within its cluster. A high average silhouette width indicates a good clustering. The average silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximizes the average silhouette over a range of possible values for k.

Both of the above process can be computed by the following code snippet.

```r
set.seed(123)
fviz_nbclust(vowel_drop, kmeans, method = "wss")
fviz_nbclust(vowel_drop, kmeans, method = "silhouette")
```





We see the best value of k for the given dataset is 3.

Add Class attribute

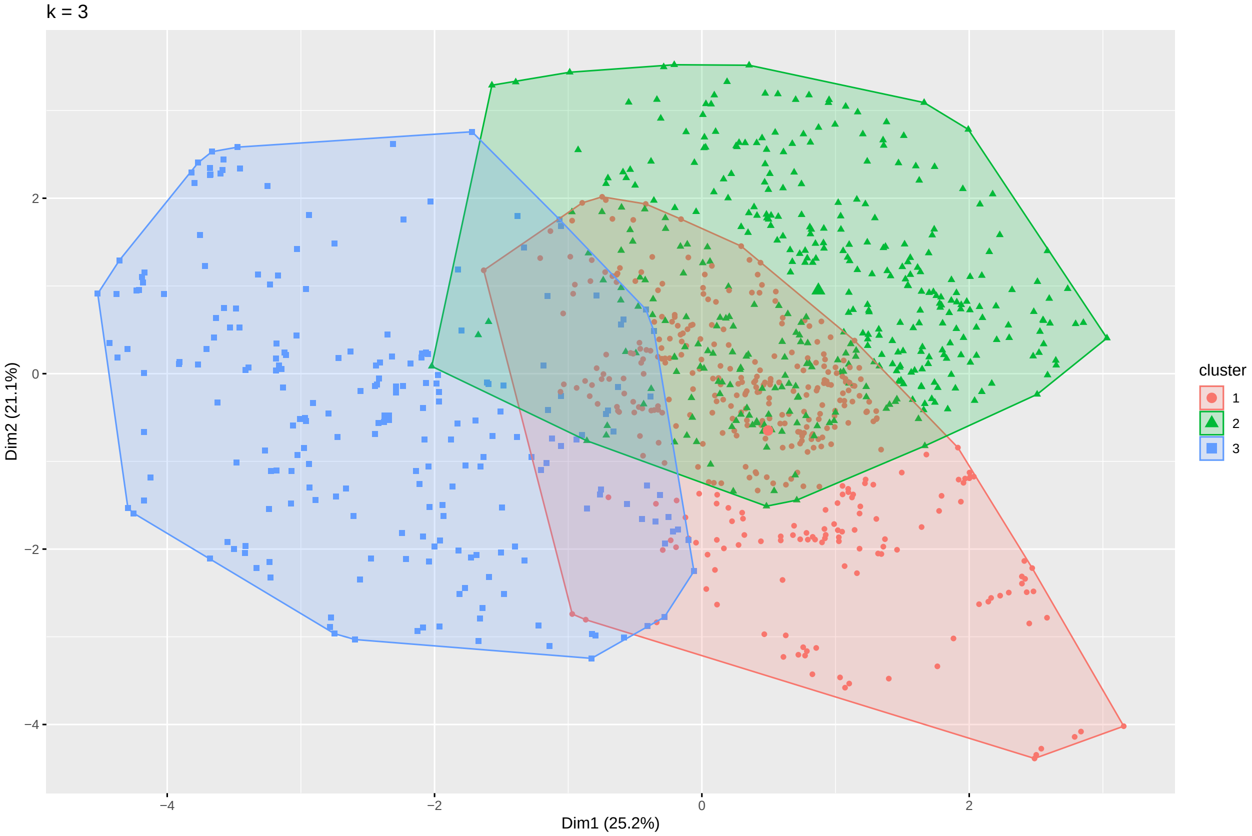
Now, we will add the *class* attribute to the dataset *vowel_drop* and will repeat the same process of previous section.

```
vowel_drop_2 <- select(vowel, -1, -2, -3)
unique(vowel_drop_2$Class)
```

```
##  [1] hid hId hEd hAd hYd had hOd hod hUd hud hed
## Levels: hid hId hEd hAd hYd had hOd hod hUd hud hed
```

We observe that the levels of the column *Class* are 11. Though, it appears that there are duplicates in classes cause of typos, case-sensitivity etc. The unique classes in the dataset are 6. We have found previously that the optimal value of k is 3. We will plot the clusters.

```
k3 <- kmeans(vowel_drop, centers = 3, nstart = 25)
fviz_cluster(k3, geom = "point",  data = vowel_drop) + ggtitle("k = 3")
```



The clusters overlap quite a bit cause of the nature of the dataset. Especially the red and green clusters appear to be misplaced. It does not seem that there is a good correlation among true-clusters and the 3 clusters of k-means.

Classification

In this section we will try to address the same problem from a different perspective. We will try two classifiers(Naive Bayes, SVM) and will measure their performace.

Naive Bayes

The fundamental Naive Bayes assumption is that each feature is independent and equal with each other.

```
library(e1071)
nb_model = naiveBayes(as.factor(Class) ~., data=vowel_drop_2)
```

```
modelPred <- predict(nb_model, vowel_drop_2)
cMatrix <- table(modelPred, vowel_drop_2$Class)
confusionMatrix(cMatrix)
```

```
## Confusion Matrix and Statistics
##
##
## modelPred hid hId hEd hAd hYd had hOd hod hUd hud hed
##       hid  76  10   0   0   0   0   0   0   0   1   0
##       hId   3  58   3   0   0   0   3   0   1   7   0
##       hEd   0   6  78   8   0   0   0   0   0   0   0
##       hAd   0   0   6  67   0   7   0   0   0   0   0
##       hYd   0   0   0   3  71  19  16   0   0   0   8
##       had   0   0   0   6   5  49   0   0   0   0  13
##       hOd   0   0   0   0  12   0  56  12  11   0   1
##       hod   0   0   0   0   0   0   8  70  12   1   0
##       hUd   0   0   0   0   1   1   7   6  52  13   0
##       hud  11  12   1   0   0   0   0   2  10  66   4
##       hed   0   4   2   6   1  14   0   0   4   2  64
##
## Overall Statistics
##
##                Accuracy : 0.7141
##                  95% CI : (0.6849, 0.7421)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6856
##
##  Mcnemar’s Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hid Class: hId Class: hEd Class: hAd Class: hYd
## Sensitivity             0.84444    0.64444    0.86667    0.74444    0.78889
## Specificity             0.98778    0.98111    0.98444    0.98556    0.94889
## Pos Pred Value          0.87356    0.77333    0.84783    0.83750    0.60684
## Neg Pred Value          0.98450    0.96503    0.98664    0.97473    0.97824
## Prevalence              0.09091    0.09091    0.09091    0.09091    0.09091
## Detection Rate          0.07677    0.05859    0.07879    0.06768    0.07172
## Detection Prevalence    0.08788    0.07576    0.09293    0.08081    0.11818
## Balanced Accuracy       0.91611    0.81278    0.92556    0.86500    0.86889
##                      Class: had Class: hOd Class: hod Class: hUd Class: hud
## Sensitivity             0.54444    0.62222    0.77778    0.57778    0.73333
## Specificity             0.97333    0.96000    0.97667    0.96889    0.95556
## Pos Pred Value          0.67123    0.60870    0.76923    0.65000    0.62264
## Neg Pred Value          0.95529    0.96214    0.97775    0.95824    0.97285
## Prevalence              0.09091    0.09091    0.09091    0.09091    0.09091
## Detection Rate          0.04949    0.05657    0.07071    0.05253    0.06667
## Detection Prevalence    0.07374    0.09293    0.09192    0.08081    0.10707
## Balanced Accuracy       0.75889    0.79111    0.87722    0.77333    0.84444
##                      Class: hed
## Sensitivity             0.71111
## Specificity             0.96333
## Pos Pred Value          0.65979
## Neg Pred Value          0.97088
## Prevalence              0.09091
## Detection Rate          0.06465
## Detection Prevalence    0.09798
## Balanced Accuracy       0.83722
```

The overall accuracy of Naive Bayes classifier is 71.4%

Support Vector Machines (SVM)

We will use SVM with linear kernel and compare its accuracy to Naive Bayes.

```
svmfit = svm(as.factor(Class) ~ ., data = vowel_drop_2, kernel = "linear", cost = 10, scale = FALSE)
modelPred <- predict(svmfit, vowel_drop_2)
cMatrix <- table(modelPred, vowel_drop_2$Class)
confusionMatrix(cMatrix)
```

```
## Confusion Matrix and Statistics
##
##
## modelPred hid hId hEd hAd hYd had hOd hod hUd hud hed
##       hid  89   2   1   0   0   0   3   0   0   0   0
##       hId   1  84  10   0   0   1   0   0   0   0   0
##       hEd   0   4  77   2   0   3   0   0   0   0   0
##       hAd   0   0   2  80   0   7   0   0   0   0   2
##       hYd   0   0   0   0  67  13   4   0   1   0   0
##       had   0   0   0   8  15  63   0   0   0   0   7
##       hOd   0   0   0   0   7   0  81   5   2   0   4
##       hod   0   0   0   0   0   0   0  76   7   2   0
##       hUd   0   0   0   0   0   0   0   9  78   3   1
##       hud   0   0   0   0   0   0   0   0   1  85   0
##       hed   0   0   0   0   1   3   2   0   1   0  76
##
## Overall Statistics
##
##                Accuracy : 0.8646
##                  95% CI : (0.8417, 0.8854)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8511
##
##  Mcnemar’s Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hid Class: hId Class: hEd Class: hAd Class: hYd
## Sensitivity             0.98889    0.93333    0.85556    0.88889    0.74444
## Specificity             0.99333    0.98667    0.99000    0.98778    0.98000
## Pos Pred Value          0.93684    0.87500    0.89535    0.87912    0.78824
## Neg Pred Value          0.99888    0.99329    0.98562    0.98888    0.97459
## Prevalence              0.09091    0.09091    0.09091    0.09091    0.09091
## Detection Rate          0.08990    0.08485    0.07778    0.08081    0.06768
## Detection Prevalence    0.09596    0.09697    0.08687    0.09192    0.08586
## Balanced Accuracy       0.99111    0.96000    0.92278    0.93833    0.86222
##                      Class: had Class: hOd Class: hod Class: hUd Class: hud
## Sensitivity             0.70000    0.90000    0.84444    0.86667    0.94444
## Specificity             0.96667    0.98000    0.99000    0.98556    0.99889
## Pos Pred Value          0.67742    0.81818    0.89412    0.85714    0.98837
## Neg Pred Value          0.96990    0.98990    0.98453    0.98665    0.99447
## Prevalence              0.09091    0.09091    0.09091    0.09091    0.09091
## Detection Rate          0.06364    0.08182    0.07677    0.07879    0.08586
## Detection Prevalence    0.09394    0.10000    0.08586    0.09192    0.08687
## Balanced Accuracy       0.83333    0.94000    0.91722    0.92611    0.97167
##                      Class: hed
## Sensitivity             0.84444
## Specificity             0.99222
## Pos Pred Value          0.91566
## Neg Pred Value          0.98456
## Prevalence              0.09091
## Detection Rate          0.07677
## Detection Prevalence    0.08384
## Balanced Accuracy       0.91833
```

Accuracy is 86.4% which is better than Naive Bayes.

Add first 3 features

In the first section we excluded 3 features of our analysis. Now we will see whether they could impact our analysis.

```
unique(vowel$`Train or Test`)
```

```
## [1] Train Test
## Levels: Train Test
```

```
unique(vowel$`Speaker Number`)
```

```
##  [1] Andrew Bill   David  Mark   Jo     Kate   Penny  Rose   Mike   Nick
## [11] Rich   Tim    Sarah  Sue    Wendy
## 15 Levels: Andrew Bill David Mark Jo Kate Penny Rose Mike Nick Rich ... Wendy
```

```
unique(vowel$Sex)
```

```
## [1] Male   Female
## Levels: Male Female
```

References

kmeans [@https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a/].

naive bayes [@https://en.wikipedia.org/wiki/Support-vector_machine].

svm [@https://scikit-learn.org/stable/modules/naive_bayes.html].