INFS7203 Assignment Report

Student: Vincent Abbosh

Student ID: 45019218

Data Preparation:

In order to remove all records with missing values, I have searched for all columns that are of type “factor” and changed it to “integer”.
Changing from “factor” to “integer” directly changes the values themselves, therefore, it has to be changed first to “character” to preserve the actual values in the column.
After changing to “integer”, all “?” values become <NA> and can be easily removed using the “complete.cases()” function.

Clustering:

K-mean Clustering:

After running the k-mean clustering algorithm with k=2 and comparing it with the original labelled data I got:

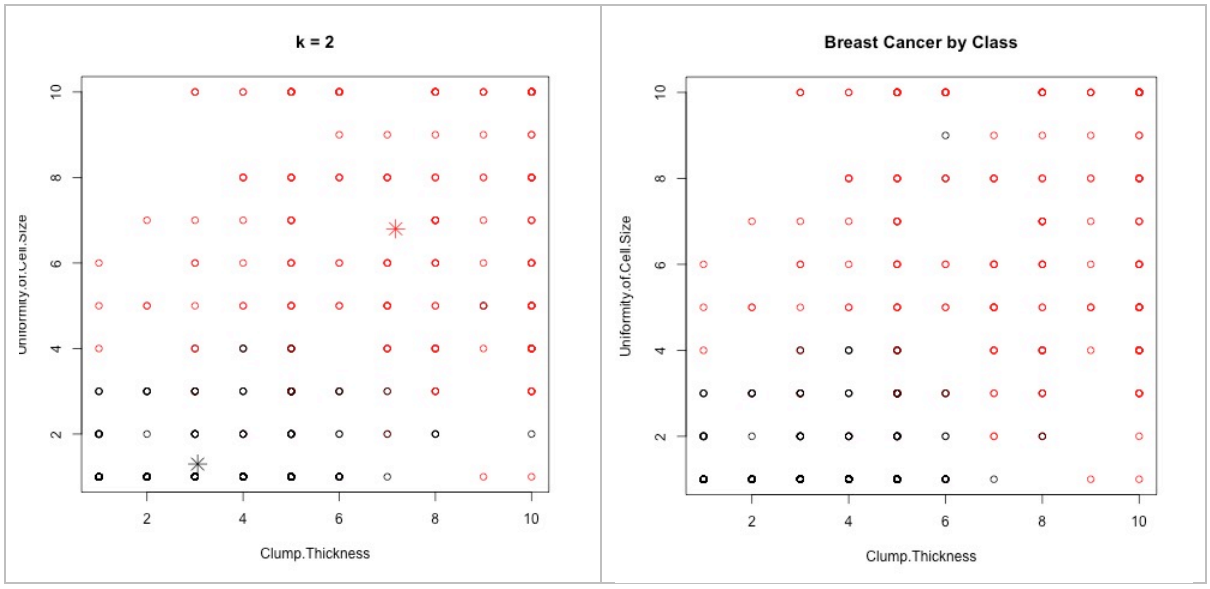


Fig.1 Comparison between k-mean clustering and original labelled data

We can see that the k-mean clustering has identified the original 2-classes data points fairly accurately.
We can check the accuracy by creating a table of the Actual class labels (2 and 4) vs the clusters labels (1 and 2):

> as.matrix(table(Actual = bcw$Class, Clusters = km.results$cluster))

	Clusters	
Actual	1	2
2	**435**	9
4	18	**221**

We can see that k-means has clustered the two different classes (benign vs malignant) with 96% accuracy.

Running k-means with k=3,4,5 and compare with k=2:

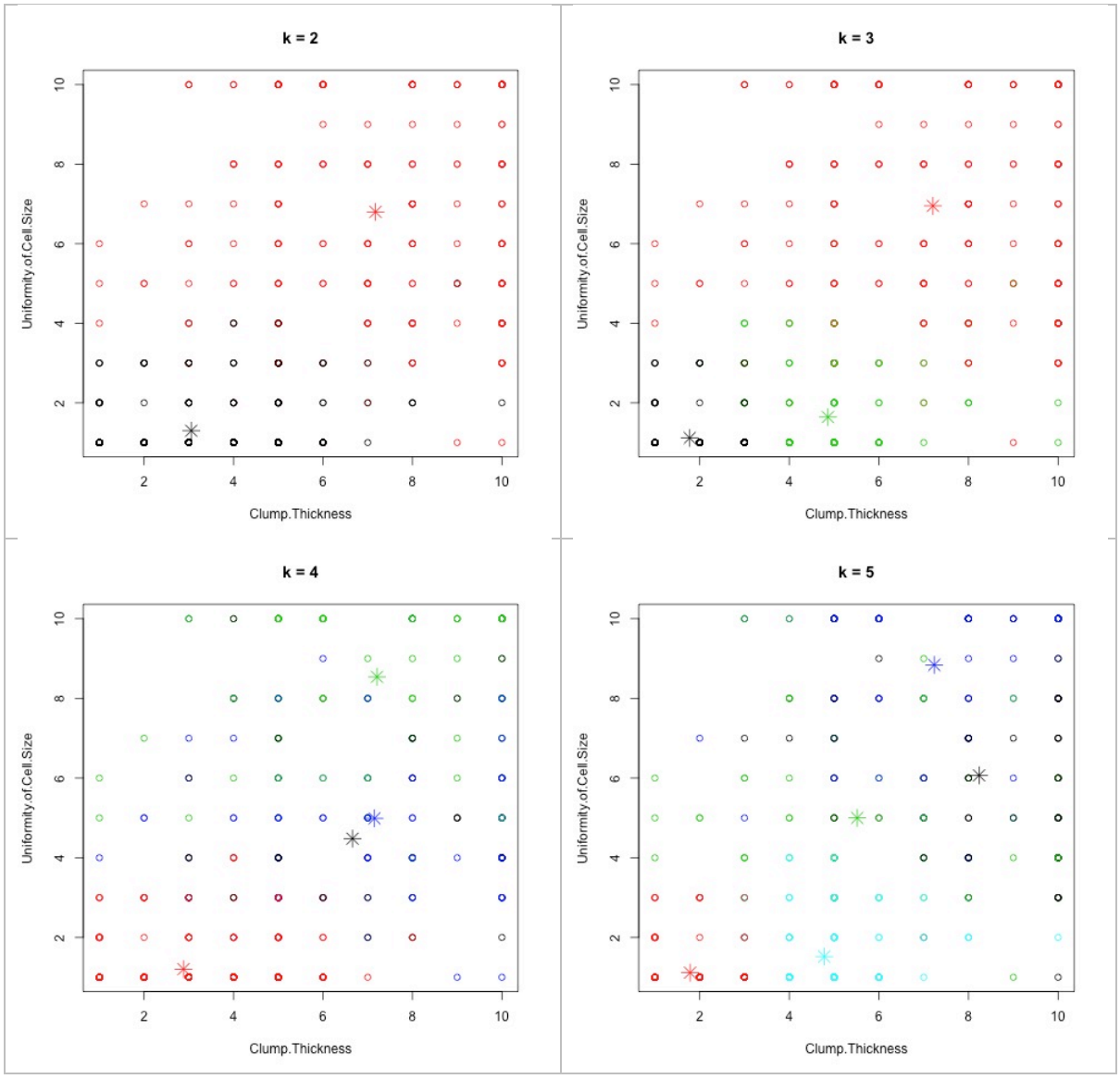
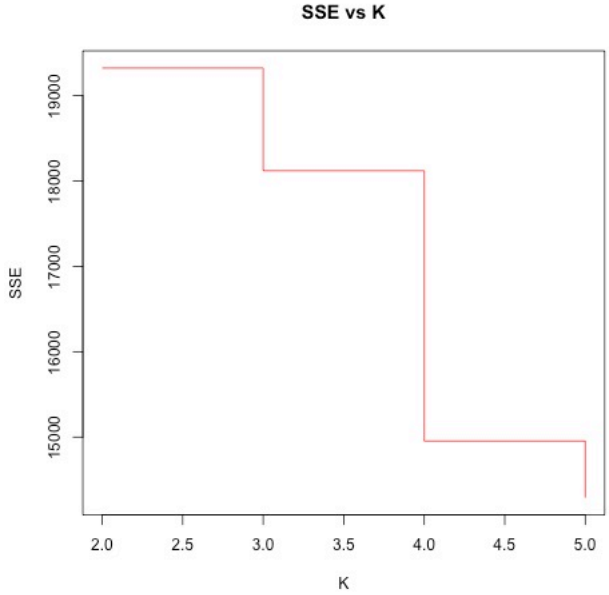


Fig. 2 differences in k-mean clustering results based on different k

Calculating Sum of Squared Errors (SSE) for each clustering and plot the results:



Which shows that we are getting the best reduction of SSE at k = 4.
But due to the limited dimensions of the plotted data, which is only 2, and due to the values being integers, I got a lot of overlapping points in the plots. Hence, I was not able to visually confirm whether k=4 is truly the best clustering of the data.

Hierarchical Clustering:

Using the default "hclust" on the data has produced the following dendrogram:

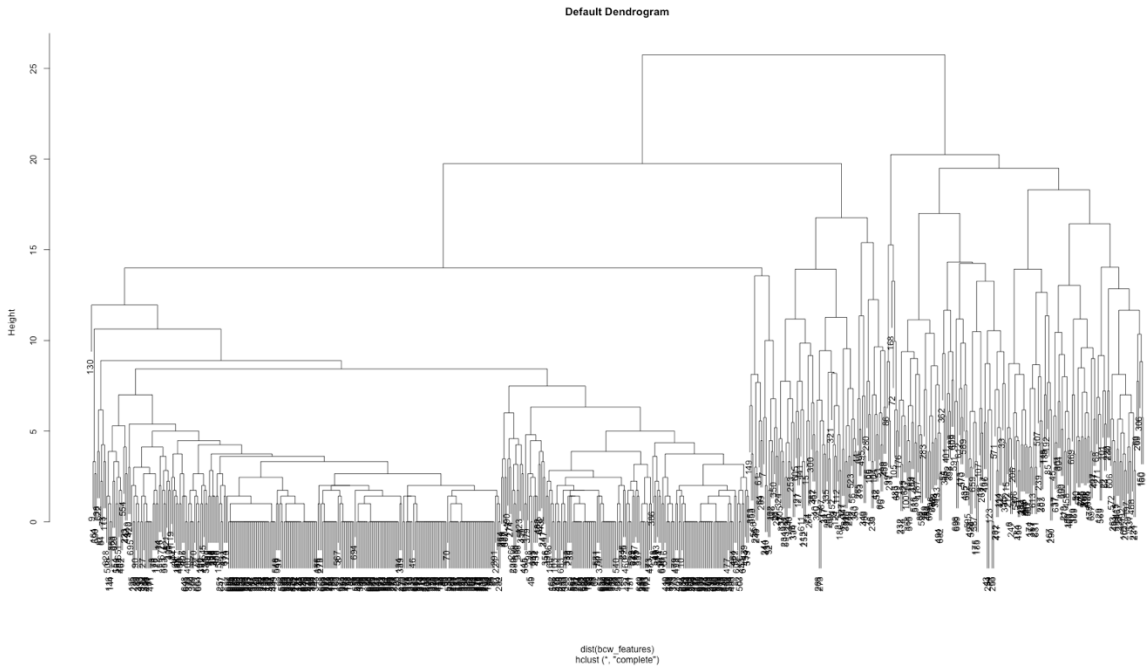


Fig. 4 Default Dendrogram

Using "cutree" command, to cut the tree at different levels, I was able to produce the different clustering of the data:

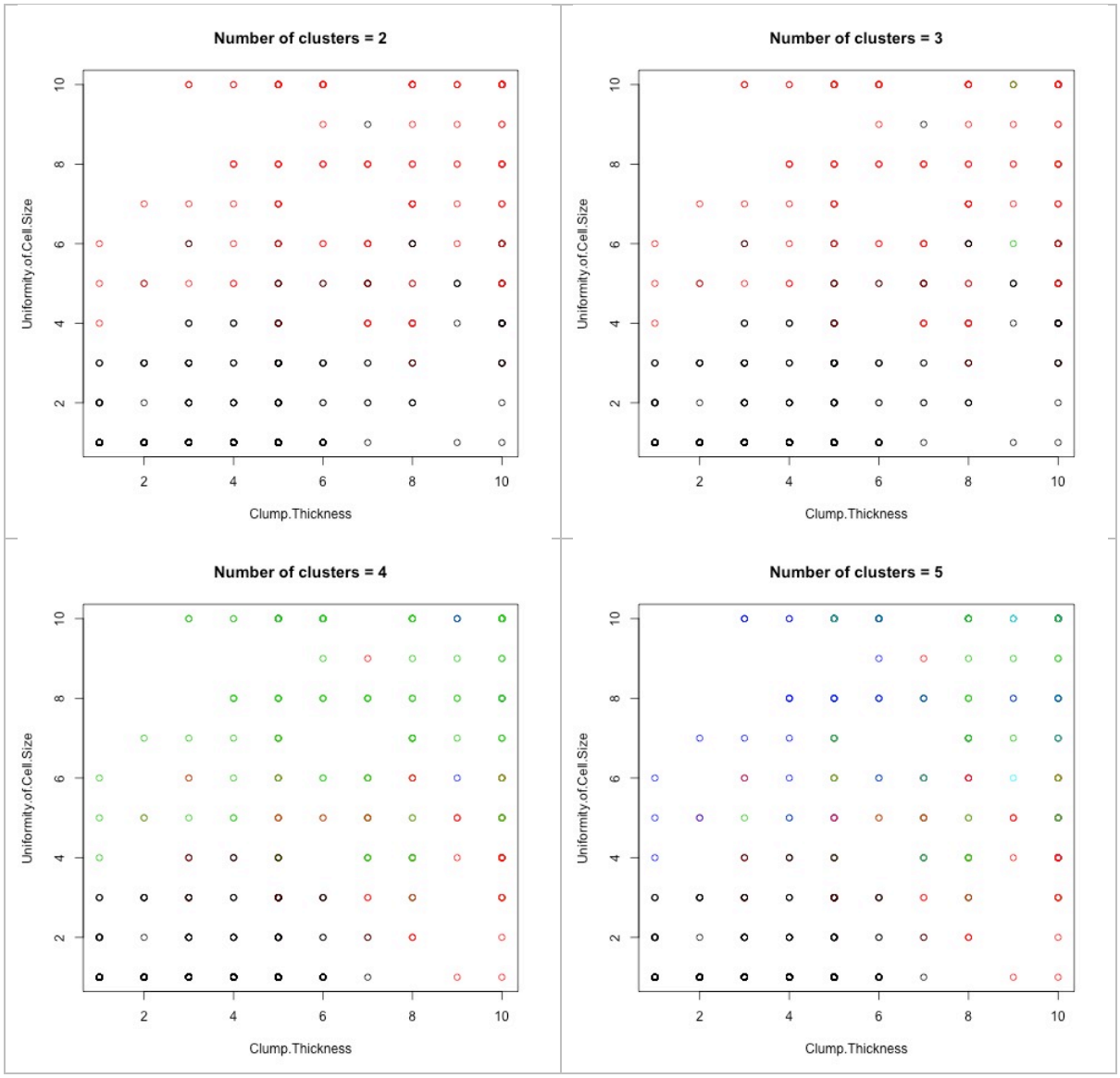
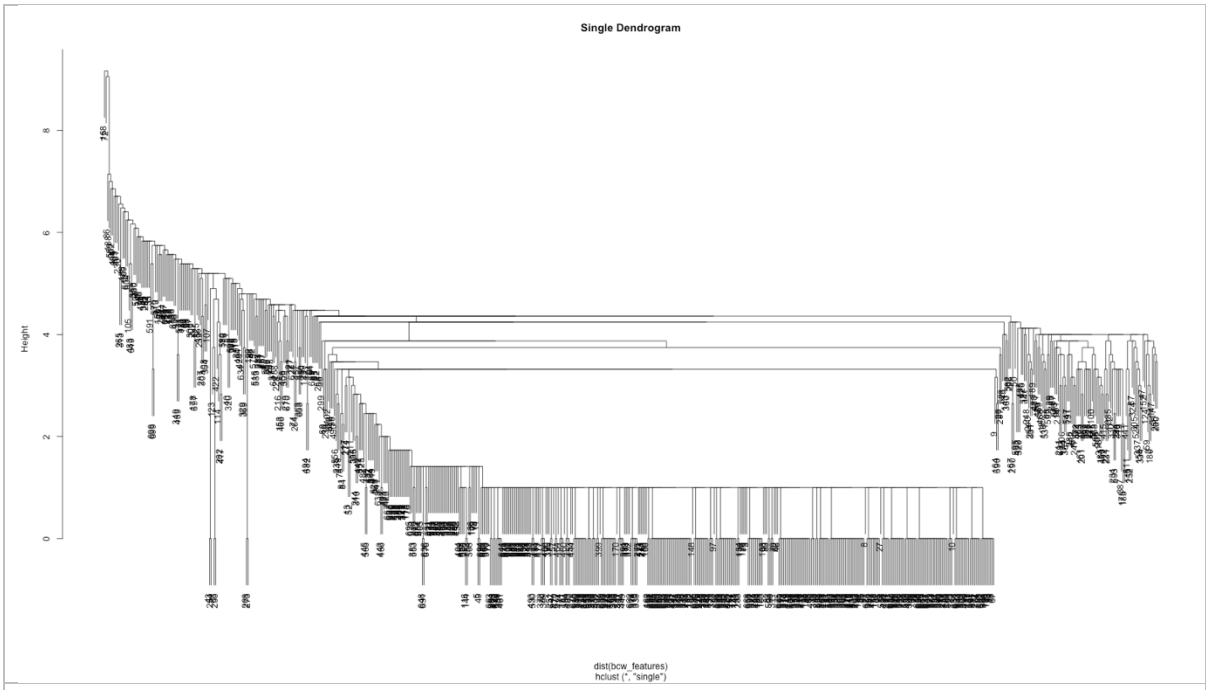
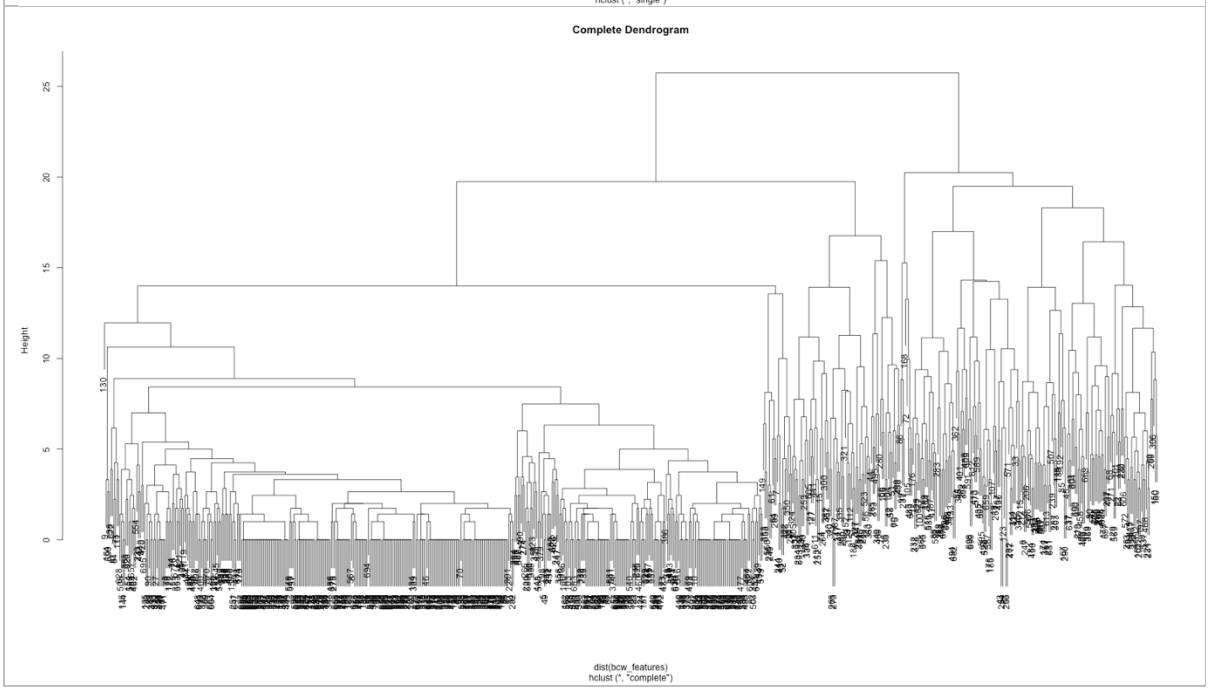


Fig.5 Different clustering of *cutree* command

We are now faced with the same issue as k-mean clustering. The inability to visually identify meaningful clustering of the data due to the points overlap and limited resolution of the values. Although, the dendrogram in Fig.4 seemed to indicate that there are 3 meaningful clusters.

By trying different agglomerations like “single”, “complete”, and “average” and see if the extra cluster can become more visible:


Single Dendrogram
Height
dist(bcw_features)
hclust (*, "single")


Complete Dendrogram
Height
dist(bcw_features)
hclust (*, "complete")

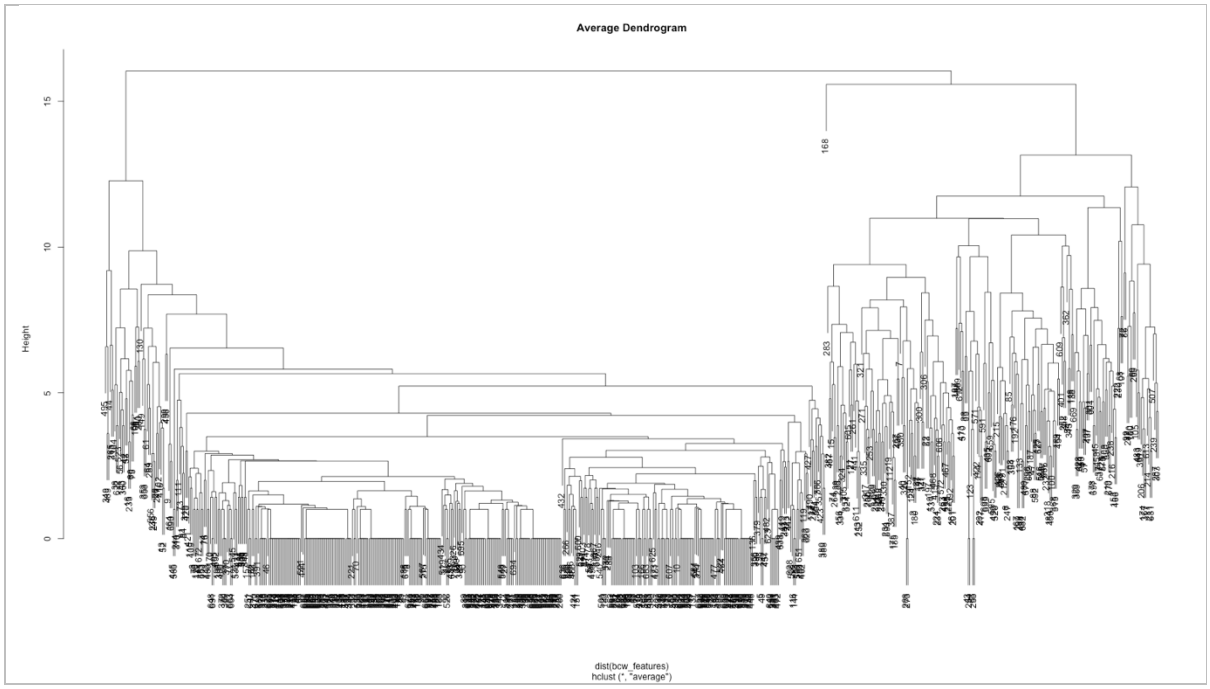


Fig.6 Single, Complete, and Average Dendrograms

The data is clearly sensitive to the agglomeration method used, as it had produced very different clusters.

The “Single” dendrogram did not add any insights into how the data is clustered as it failed to properly aggregate data into larger clusters.

The “Complete” dendrogram is similar to the Default dendrogram produced earlier in Fig.4, which means that the default “*hclust”* dendrogram uses the “Complete” agglomeration method.

The “Average” dendrogram did show a better insight by clearly identifying 3 different clusters. So, there is a possibility that a 3rd class of diseases might exist.

Classification:

Decision Trees:

I used Conditional inference trees, through “ctree” command, to fit a binary tree to the data.
After splitting the data randomly as 70% training and 30% testing sets, and then fitting the default tree to the training set, I got the following tree diagram:

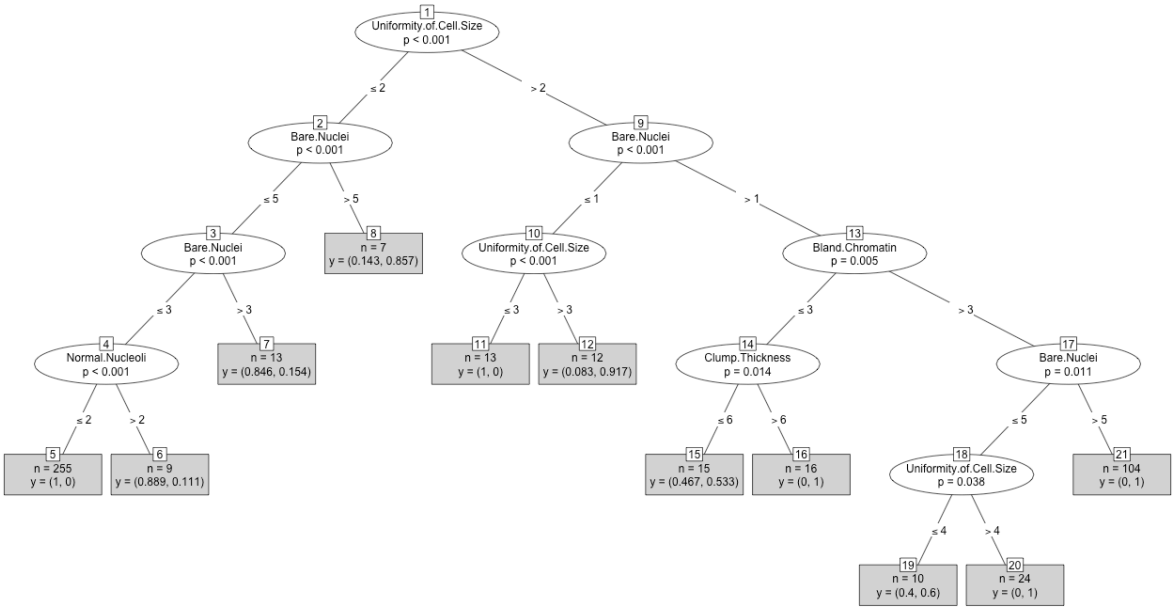


Fig.7 Default ctree

The tree shows a few redundant nodes that are not adding any value to the classification. For example, nodes 4, 17, and 18 are not changing the decision at all.

Looking at the tree, we could see that the most important features are: “Uniformity of Cell size”, “Bare Nuclei”, and “Clump Thickness”. Their decision nodes at the root of tree provided the most insight into the classification.

The function “calculatePerformance(confusion_matrix)” calculates the performance of the classification in term of accuracy, precision, and recall:

> *calculatePerformance(confu_m)*

	accuracy	**precision**	**recall**	**f1**
2	0.9560976	0.9856115	0.9513889	0.9681979
4	0.9560976	0.8939394	0.9672131	0.9291339

These are very good numbers. Let’s see if it is possible to remove the redundancies in the tree while at least keeping the performance as it is.

By trying different parameters of “ctree” command, like changing testtype to “MonteCarlo” instead of the default value “Bonferroni”, and raising the 1-p value to 0.97 instead of the default 0.95, I got this simplified tree:

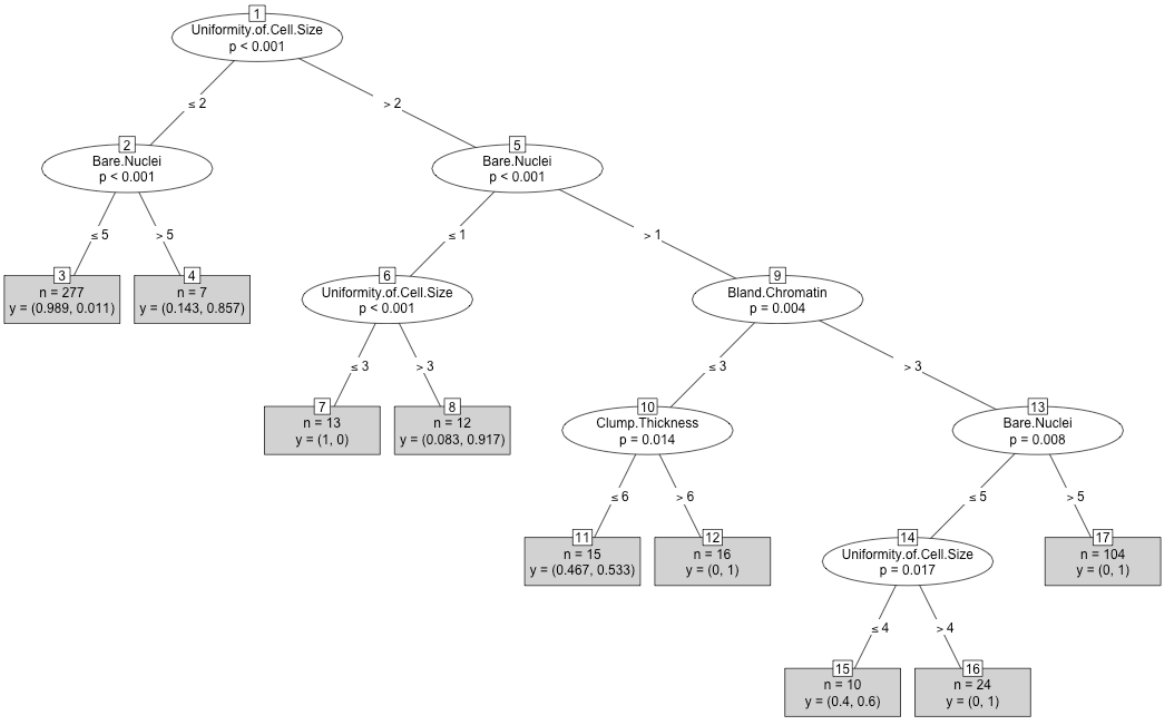


Fig. 8 MonteCarlo testtype with 0.97 mincriterion

A number of redundant decision nodes have been removed. But we still have nodes 13 and 14 that are not adding any real value to the classification as they both classifies the data into one particular label (y = (0,1).
Reducing the maximum depth of the tree by setting the parameter maxdepth = 3 produced this tree:

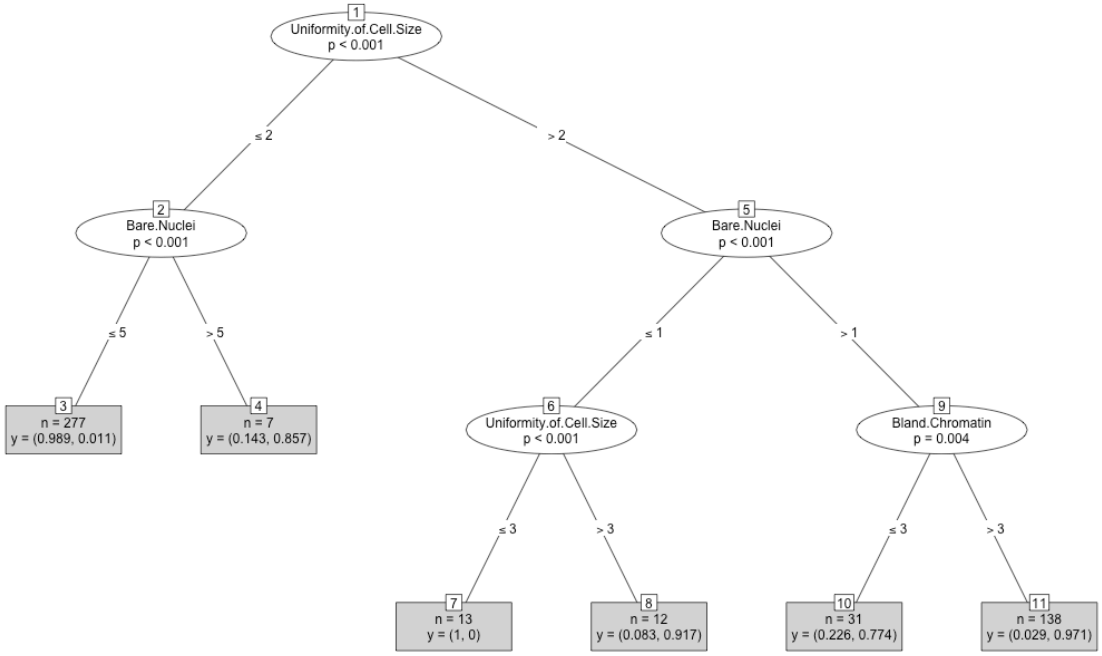


Fig.9 MonteCarlo testtype, with 0.97 mincriterion, and 3 maxdepth

This is now a more reasonably looking tree. And it has the same classification performance as the default tree in Fig.7:

> *calculatePerformance(confu_m4)*

	accuracy	precision	recall	f1
2	0.9560976	0.9856115	0.9513889	0.9681979
4	0.9560976	0.8939394	0.9672131	0.9291339

So, we have reduced the complexity of the tree while keeping its classification performance as it is.

K-NN:

Using K-NN, with K taking values from 1 to 5, on the test set produced these classification performances:

> *perf_results*

[[1]]

	accuracy	precision	recall	f1
2	0.9560976	0.9655172	0.9722222	0.9688581
4	0.9560976	0.9333333	0.9180328	0.9256198

[[2]]

	accuracy	precision	recall	f1
2	0.9560976	0.9655172	0.9722222	0.9688581
4	0.9560976	0.9333333	0.9180328	0.9256198

[[3]]

	accuracy	precision	recall	f1
2	0.9707317	0.9859155	0.9722222	0.9790210
4	0.9707317	0.9365079	0.9672131	0.9516129

[[4]]

	accuracy	precision	recall	f1
2	0.9804878	0.9929577	0.9791667	0.9860140
4	0.9804878	0.9523810	0.9836066	0.9677419

[[5]]

	accuracy	precision	recall	f1
2	0.9804878	0.9929577	0.9791667	0.9860140
4	0.9804878	0.9523810	0.9836066	0.9677419

We can see that the best K-NN classifier is when K = 4. The K-NN accuracy, precision, and recall were a little better than the Decision Tree in Fig.9