CLASSIFICATIONS

Order from Chaos

Erik Kusch

erik.kusch@au.dk

Section for Ecoinformatics & Biodiversity
Center for Biodiversity and Dynamics in a Changing World (BIOCHANGE)
Aarhus University

01/04/2020

Types of Variables

Variables can be classed into a multitude of types. The most common classification system knows:

Categorical Variables

- also known as *Qualitative Variables*
- Scales can be either:
  - Nominal
  - Ordinal

Continuous Variables

- also known as *Quantitative Variables*
- Scales can be either:
  - Discrete
  - Continuous

Types of Variables

Variables can be classed into a multitude of types. The most common classification system knows:

Categorical Variables

- also known as *Qualitative Variables*
- Scales can be either:
  - Nominal
  - Ordinal

Continuous Variables

- also known as *Quantitative Variables*
- Scales can be either:
  - Discrete
  - Continuous

Types of Variables

Variables can be classed into a multitude of types. The most common classification system knows:

Categorical Variables

- also known as *Qualitative Variables*
- Scales can be either:
  - Nominal
  - Ordinal

Continuous Variables

- also known as *Quantitative Variables*
- Scales can be either:
  - Discrete
  - Continuous

Categorical Variables

Categorical variables are those variables which **establish and fall into distinct groups and classes**.

Categorical variables:

- can take on a finite number of values
- assign each unit of the population to one of a finite number of groups
- can *sometimes* be ordered

In R, categorical variables usually come up as object type `factor` or `character`.

Categorical Variables

Categorical variables are those variables which **establish and fall into distinct groups and classes**.

Categorical variables:

- can take on a finite number of values
- assign each unit of the population to one of a finite number of groups
- can *sometimes* be ordered

In R, categorical variables usually come up as object type `factor` or `character`.

Categorical Variables

Categorical variables are those variables which **establish and fall into distinct groups and classes**.

Categorical variables:

- can take on a finite number of values
- assign each unit of the population to one of a finite number of groups
- can *sometimes* be ordered

In R, categorical variables usually come up as object type `factor` or `character`.

Categorical Variables (Examples)

Examples of categorical variables:

- Biome Classifications (e.g. "Boreal Forest", "Tundra", etc.)
- Sex (e.g. "Male", "Female")
- Hierarchy Position (e.g. "α-Individual", "β-Individual", etc.)
- Soil Type (e.g. "Sandy", "Mud", "Permafrost", etc.)
- Leaf Type (e.g. "Compound", "Single Blade", etc.)
- Sexual Reproductive Stage (e.g. "Juvenile", "Mature", etc.)
- Species Membership
- Family Group Membership
- ...

Categorical Variables (Examples)

Examples of categorical variables:

- Biome Classifications (e.g. "Boreal Forest", "Tundra", etc.)
- Sex (e.g. "Male", "Female")
- Hierarchy Position (e.g. "α-Individual", "β-Individual", etc.)
- Soil Type (e.g. "Sandy", "Mud", "Permafrost", etc.)
- Leaf Type (e.g. "Compound", "Single Blade", etc.)
- Sexual Reproductive Stage (e.g. "Juvenile", "Mature", etc.)
- Species Membership
- Family Group Membership
- ...

Continuous Variables

Continuous variables are those variables which **establish a range of possible data values**.

Continuous variables:

- can take on an infinite number of values
- can take on a new value for each unit in the set-up
- can *always* be ordered

In R, continuous variables usually come up as object type `numeric`.

Continuous Variables

Continuous variables are those variables which **establish a range of possible data values**.

Continuous variables:

- can take on an infinite number of values
- can take on a new value for each unit in the set-up
- can *always* be ordered

In R, continuous variables usually come up as object type `numeric`.

Continuous Variables

Continuous variables are those variables which **establish a range of possible data values**.

Continuous variables:

- can take on an infinite number of values
- can take on a new value for each unit in the set-up
- can *always* be ordered

In R, continuous variables usually come up as object type `numeric`.

Continuous Variables (Examples)

Examples of categorical variables:

- Temperature
- Precipitation
- Weight
- pH
- Altitude
- Group Size
- Vegetation Indices
- Time
- ...

Continuous Variables (Examples)

Examples of categorical variables:

- Temperature
- Precipitation
- Weight
- pH
- Altitude
- Group Size
- Vegetation Indices
- Time
- ...

Binning Variables

Continuous variables can be converted into *categorical variables* via a method called **binning:**

Given a variable range, one can establish however many "bins" as one wants.

For example:

- Given a temperature range of $271K - 291K$, there may be 4 bins of equal size:
  - Bin A: $271K \leq X \leq 276K$
  - Bin B: $276K < X \leq 281K$
  - Bin C: $281K < X \leq 286K$
  - Bin D: $286K < X \leq 291K$

Whilst a **continuous variable** can be both *continuous* and *categorical*, a **categorical variable** can only ever be *categorical*!

Binning Variables

Continuous variables can be converted into *categorical variables* via a method called **binning:**

Given a variable range, one can establish however many "bins" as one wants. For example:

- Given a temperature range of $271K - 291K$, there may be 4 bins of equal size:
  - Bin A: $271K \leq X \leq 276K$
  - Bin B: $276K < X \leq 281K$
  - Bin C: $281K < X \leq 286K$
  - Bin D: $286K < X \leq 291K$

Whilst a **continuous variable** can be both *continuous* and *categorical*, a **categorical variable** can only ever be *categorical*!

Binning Variables

Continuous variables can be converted into *categorical variables* via a method called **binning:**

Given a variable range, one can establish however many "bins" as one wants.

For example:

- Given a temperature range of $271K - 291K$, there may be 4 bins of equal size:
  - Bin A: $271K \leq X \leq 276K$
  - Bin B: $276K < X \leq 281K$
  - Bin C: $281K < X \leq 286K$
  - Bin D: $286K < X \leq 291K$

Whilst a **continuous variable** can be both *continuous* and *categorical*, a **categorical variable** can only ever be *categorical*!

Binning Variables

Continuous variables can be converted into *categorical variables* via a method called **binning:**

Given a variable range, one can establish however many "bins" as one wants.

For example:

- Given a temperature range of $271K - 291K$, there may be 4 bins of equal size:
  - Bin A: $271K \leq X \leq 276K$
  - Bin B: $276K < X \leq 281K$
  - Bin C: $281K < X \leq 286K$
  - Bin D: $286K < X \leq 291K$

Whilst a **continuous variable** can be both *continuous* and *categorical*, a **categorical variable** can only ever be *categorical*!

Confusion Of Units

Theory

Logistic Regression

`glm(..., family=binomial(link='logit'))` in base `R`

Purpose: Understand how certain variables drive distinct outcomes.

Assumptions:

- Down to *Study-Design*:
  - Variable values are **independent** (not paired)
  - *Binary logistic regression*: response variable is **binary**
  - *Ordinal logistic regression*: response variable is **ordinal**
- Need for *Post-Hoc Tests*:
  - Absence of **influential outliers**
  - Absence of **multi-collinearity**
  - Predictor Variables and **log odds** are related in a **linear fashion**

Theory

Logistic Regression

`glm(..., family=binomial(link='logit'))` in base `R`

Purpose: Understand how certain variables drive distinct outcomes.

Assumptions:

- Down to *Study-Design*:
  - Variable values are **independent** (not paired)
  - *Binary logistic regression*: response variable is **binary**
  - *Ordinal logistic regression*: response variable is **ordinal**
- Need for *Post-Hoc Tests*:
  - Absence of **influential outliers**
  - Absence of **multi-collinearity**
  - Predictor Variables and **log odds** are related in a **linear fashion**

Example - The Data

```
library(titanic)
train_df <- na.omit(titanic_train) # remove NA rows
# retain first 50 rows for testing
test_df <- train_df[c(1:50),c(2,3,5,6)]
# delete first 50 rows for testing
train_df <- train_df[c(-1:-50),c(2,3,5,6)]
head(train_df)
```

```
##    Survived Pclass    Sex Age
## 68        0      3   male  19
## 69        1      3 female  17
## 70        0      3   male  26
## 71        0      2   male  32
## 72        0      3 female  16
## 73        0      2   male  21
```

→ Let's see if there is a good explanation for **Survival** (`Survived`) to be had based off of *Passenger class* (`Pclass`), *sex* (`Sex`), and *age* (`Age`). Was it really "Women and children first"?

Example - The Model

```
Logistic_Mod <- glm(Survived ~., family=binomial(link='logit'), data = train_df)
summary(Logistic_Mod)
```

```
##
## Call:
## glm(formula = Survived ~ ., family = binomial(link = "logit"),
##     data = train_df)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.745  -0.690  -0.405   0.651   2.455
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.11787    0.51998    9.84  < 2e-16 ***
## Pclass      -1.29567    0.14362   -9.02  < 2e-16 ***
## Sexmale     -2.45459    0.21484  -11.43  < 2e-16 ***
## Age         -0.03867    0.00794   -4.87  1.1e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 896.44  on 663  degrees of freedom
## Residual deviance: 604.61  on 660  degrees of freedom
## AIC: 612.6
##
## Number of Fisher Scoring iterations: 5
```

Example - Explanation & Prediction

Obviously, the coefficients of our model can't be **explained** the same way as typical regression coefficients since we are interested in survival probabilities between 0 and 1. However, we can interpret the **direction** and **strength** of effects as well as their statistical significance at a glance: Clearly, women of a young age in first class had the highest survival rate. How do we know this? As class increases (from 1 to 3), survival probability decreases (-1.2957). Furthermore, men had, on average, a much lower survival rate than women (-2.4546). Lastly, increasing age negatively affected survival chances (-0.0387).

But how sure can we be of our model accuracy? We can test it by **predicting** some new data and **validating** our predictions:

```
fitted.results <- predict(Logistic_Mod, newdata=test_df, type='response') # predict on test data
fitted.results <- ifelse(fitted.results > 0.5 , 1, 0) # if predicted survival probability above .5 assume survival
misClasificError <- mean(fitted.results != test_df$Survived) # compare actual data with predictions
print(paste('Accuracy',1-misClasificError)) # output
```

```
## [1] "Accuracy 0.8"
```

Theory

K-Means Clustering

`Mclust()` in `mclust` package

Purpose: Identify a number of k clusters in our data.

Assumptions:

- Variance of the distribution of each variable is spherical
- All variables have the same variance
- Prior probability for all k clusters are the same

Theory

K-Means Clustering

`Mclust()` in `mclust` package

Purpose: Identify a number of k clusters in our data.

Assumptions:

- Variance of the distribution of each variable is spherical
- All variables have the same variance
- Prior probability for all k clusters are the same

Example - The Data

```
data("iris")
head(iris)
```

```
##   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
```

→ Let's see if we can accurately identify the `Species` contained within the data set by clustering according to `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`.

`mclust` is capable of identifying the statistically most appropriate number of clusters for the data set. Here, we decide to limit the number of clusters to the number of species present so we can test how well the prediction went.

Example - The Data

```
data("iris")
head(iris)
```

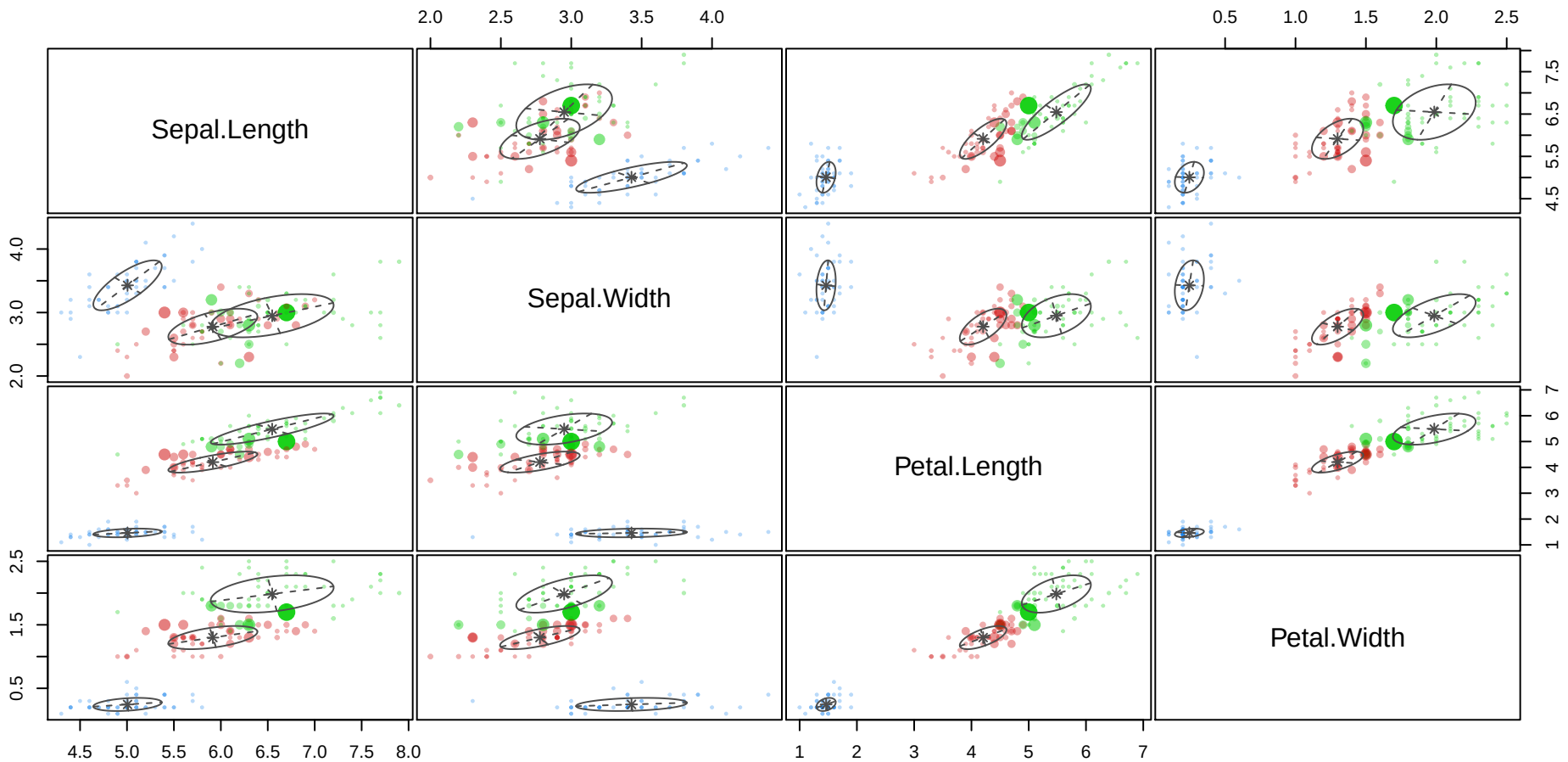
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

→ Let's see if we can accurately identify the `Species` contained within the data set by clustering according to `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`.

`mclust` is capable of identifying the statistically most appropriate number of clusters for the data set. Here, we decide to limit the number of clusters to the number of species present so we can test how well the prediction went.

Example - The Model

```
library(mclust)
Mclust_mod <- Mclust(iris[,-5], # data for the cluster model
                     G = length(unique(iris[,5]))) # number of species
plot(Mclust_mod, what = "uncertainty")
```



Example - Explanation & Prediction

K-means clusters can be interpreted to a certain degree. For example, by calling `Mclust_mod[["parameters"]][["mean"]]` we can assess the mean value of each cluster for each variable. These allow for some **biological interpretation**. Personally, I prefer a visualization as seen on the previous slide. Clearly, `Petal.Length`, and `Petal.Width` are extremely good separators for our different clusters with the green and red clusters overlapping a lot in `Sepal.Length` and `Sepal.Width` space.

But how sure can we be of our model accuracy? We can test it by **predicting** the cluster membership and **validating** our predictions against the real data:

```
Mclust_pred <- predict.Mclust(Mclust_mod, iris[,-5]) # prediction
# extract predicted species number
fitted.results <- Mclust_pred$classification
# compare actual data with predictions
misClasificError <- mean(fitted.results != as.numeric(iris$Species))
print(paste('Accuracy',1-misClasificError)) # output
```

```
## [1] "Accuracy 0.966666666666667"
```

Attention! We wouldn't want to do this in a real analysis. There, we would like to split the data in training and test data like we did with the logistic regression example.

Example - Explanation & Prediction

K-means clusters can be interpreted to a certain degree. For example, by calling `Mclust_mod[["parameters"]][["mean"]]` we can assess the mean value of each cluster for each variable. These allow for some **biological interpretation**. Personally, I prefer a visualization as seen on the previous slide. Clearly, `Petal.Length`, and `Petal.Width` are extremely good separators for our different clusters with the green and red clusters overlapping a lot in `Sepal.Length` and `Sepal.Width` space.

But how sure can we be of our model accuracy? We can test it by **predicting** the cluster membership and **validating** our predictions against the real data:

```
Mclust_pred <- predict.Mclust(Mclust_mod, iris[,-5]) # prediction
# extract predicted species number
fitted.results <- Mclust_pred$classification
# compare actual data with predictions
misClasificError <- mean(fitted.results != as.numeric(iris$Species))
print(paste('Accuracy',1-misClasificError)) # output
```

```
## [1] "Accuracy 0.966666666666667"
```

Attention! We wouldn't want to do this in a real analysis. There, we would like to split the data in training and test data like we did with the logistic regression example.

Theory

Hierarchical Clustering

`hclust()` in base `R` or `rpart()` in `rpart` package and many others

Purpose: Build a decision tree for classification of our data.

Advantages:

- Very easy to **explain and interpret**.
- Easy to **visualize**.
- Easily handle qualitative predictors without the need to create dummy variables.

Disadvantages:

- Very **sensitive to** the **choice of linkage**.
- Generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
- Trees can be very **non-robust**.

Theory

Hierarchical Clustering

`hclust()` in base `R` or `rpart()` in `rpart` package and many others

Purpose: Build a decision tree for classification of our data.

Advantages:

- Very easy to **explain and interpret**.
- Easy to **visualize**.
- Easily handle qualitative predictors without the need to create dummy variables.

Disadvantages:

- Very **sensitive to** the **choice of linkage**.
- Generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
- Trees can be very **non-robust**.

Example - The Data

```
data("iris")
head(iris)
```

```
##   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
```

→ Again, let's see if we can accurately identify the `Species` contained within the data set by clustering according to `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`.

`hclust()` can only handle distance matrices. We generate one between the numeric components of our data set as seen below.

```
dist_mat <- dist(iris[, -5])
```

A distance matrix stores information about the dissimilarity of different observations.

Example - The Data

```
data("iris")
head(iris)
```

```
##   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
```

→ Again, let's see if we can accurately identify the `Species` contained within the data set by clustering according to `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`.

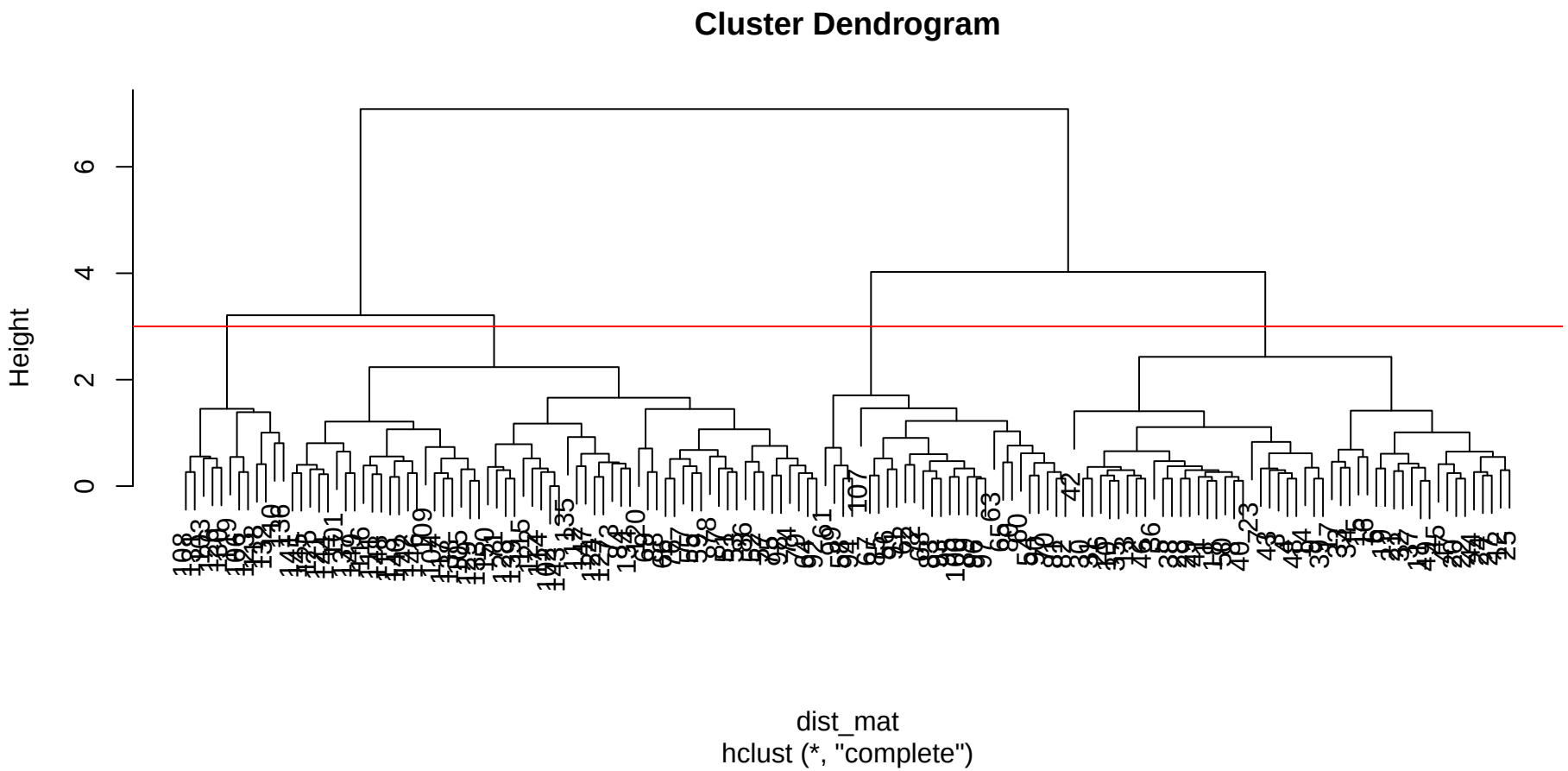
`hclust()` can only handle distance matrices. We generate one between the numeric components of our data set as seen below.

```
dist_mat <- dist(iris[, -5])
```

A distance matrix stores information about the dissimilarity of different observations.

Example - The Model

```
clusters <- hclust(dist_mat)
plot(clusters)
abline(h = 3, col = "red")
```



Example - Explanation & Prediction

Obviously, our initial dendrogramm recognizes way more clusters than we are interested in. In fact, it recognizes as many clusters as their are observations. We may wish to cut the tree down to a proper size. We know that we have three species in our data, so we may want to cut the tree at a height of 3 - not because that's the number of species, but because the tree just so happens to recognize three clusters at that level of decision-making.

```
clusterCut <- cutree(clusters, 3) # cut tree
table(clusterCut, iris$Species) # assess fit
```

```
##
## clusterCut setosa versicolor virginica
##          1     50          0         0
##          2      0         23        49
##          3      0         27         1
```

As we can see here, our decision tree has had no issue identifying *setosa* and *virginica* into clusters 1 and 2 respectively. However, it is struggling with placing the species *versicolor*.

Note that we did not take into account any special linkages here!

Example - Explanation & Prediction

Obviously, our initial dendrogramm recognizes way more clusters than we are interested in. In fact, it recognizes as many clusters as their are observations. We may wish to cut the tree down to a proper size. We know that we have three species in our data, so we may want to cut the tree at a height of 3 - not because that's the number of species, but because the tree just so happens to recognize three clusters at that level of decision-making.

```
clusterCut <- cutree(clusters, 3) # cut tree
table(clusterCut, iris$Species) # assess fit
```

```
##
## clusterCut setosa versicolor virginica
##          1     50          0         0
##          2      0         23        49
##          3      0         27         1
```

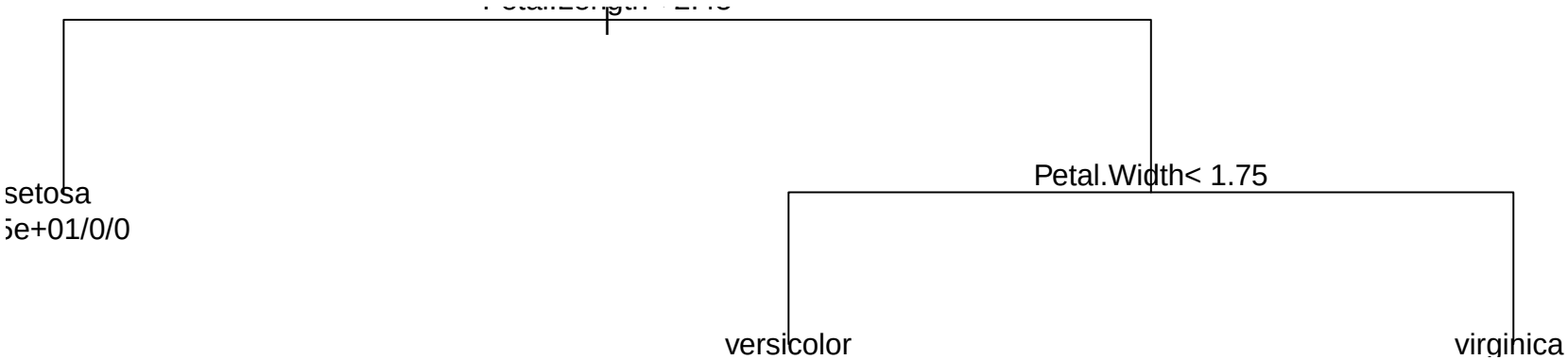
As we can see here, our decision tree has had no issue identifying *setosa* and *virginica* into clusters 1 and 2 respectively. However, it is struggling with placing the species *versicolor*.

Note that we did not take into account any special linkages here!

Example - Decisions

So far we weren't able to tell the actual decision rules of how to cluster our data. Let's do this:

```
library(rpart)
fit <- rpart(Species ~. , data = iris)
plot(fit)
text(fit, use.n = TRUE)
```

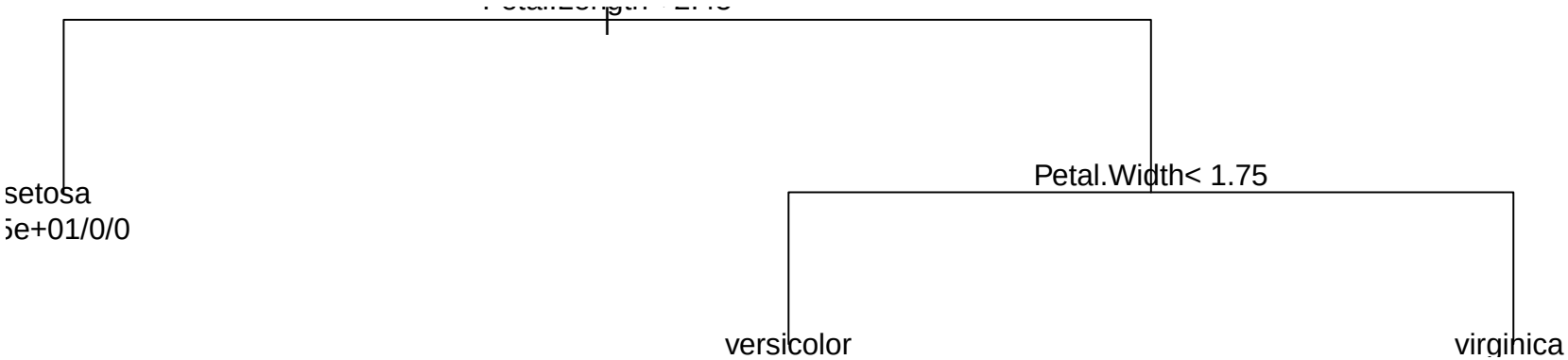


Despite the slightly awkward plotting, we can tell that our decisions for assigning species membership build on `Petal.Length` and `Petal.Width` in this example (remember the K-mean clustering which showed roughly the same?)!

Example - Decisions

So far we weren't able to tell the actual decision rules of how to cluster our data. Let's do this:

```
library(rpart)
fit <- rpart(Species ~. , data = iris)
plot(fit)
text(fit, use.n = TRUE)
```



Despite the slightly awkward plotting, we can tell that our decisions for assigning species membership build on `Petal.Length` and `Petal.Width` in this example (remember the K-mean clustering which showed roughly the same?)!

Theory

Random Forests

`tuneRF()` in `randomForest` package

Purpose:

Identify which variables to use for clustering our data and build a tree.

Advantages:

- Extremely **powerful**.
- Very **robust**.
- Easy to **interpret**.

Disadvantages:

- A **black box** algorithm.
- **Computationally expensive**.

Theory

Random Forests

`tuneRF()` in `randomForest` package

Purpose:

Identify which variables to use for clustering our data and build a tree.

Advantages:

- Extremely **powerful**.
- Very **robust**.
- Easy to **interpret**.

Disadvantages:

- A **black box** algorithm.
- **Computationally expensive**.

Example - The Data

```
library(randomForest)
data("iris")
head(iris)
```

```
##   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
```

→ Once more, let's see if we can accurately identify the `Species` contained within the data set by clustering according to `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`.

Example - The Model

```
set.seed(42) # set seed because the process is random
RF_Mod <- tuneRF(x = iris[,-5], # variables which to use for clustering
                 y = iris[,5], # correct cluster assignment
                 strata = iris[,5], # consider this for stratified sampling
                 doBest = TRUE, # run the best overall tree
                 ntreeTry = 20000, # consider this number of trees
                 improve = 0.0001, # use an improvement for tuning if this margin is exceeded
                 trace = FALSE, plot = FALSE)
```

```
## -0.1429 0.0001
## 0 0.0001
```

```
RF_Mod
```

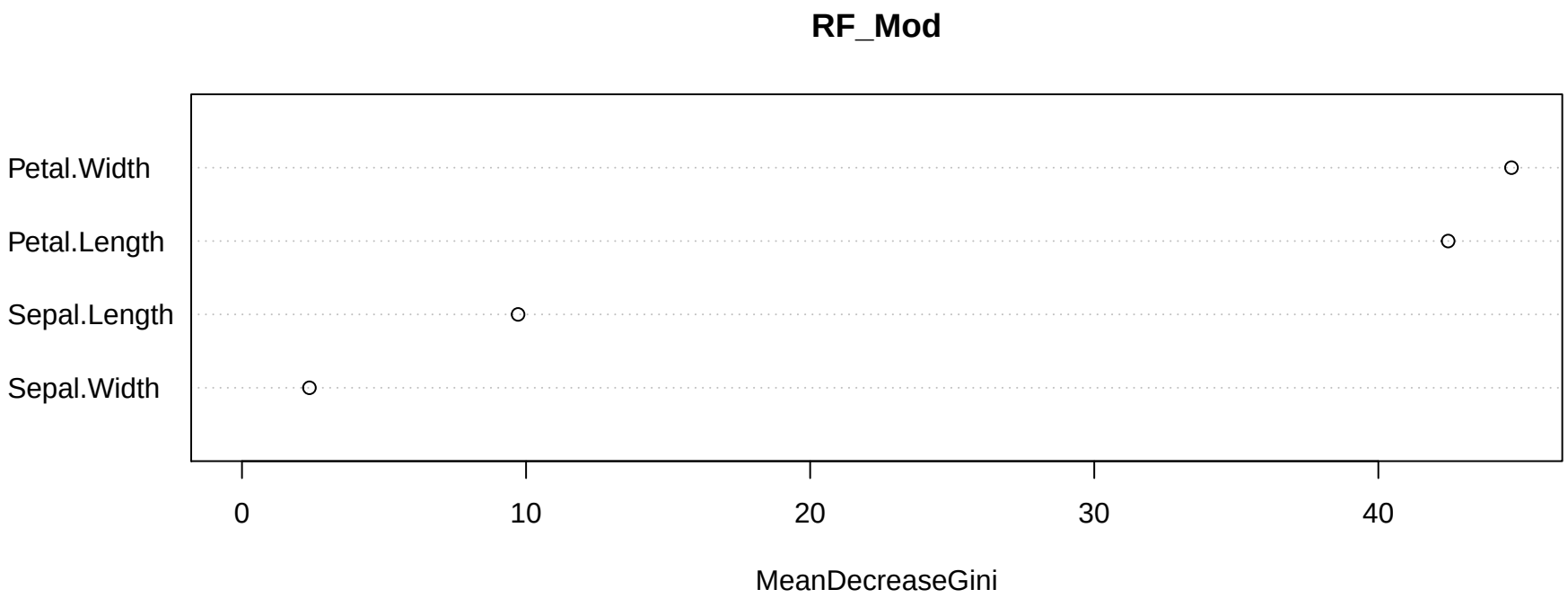
```
##
## Call:
##  randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1],      strata = ..1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 4%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

That is one **stunningly accurate** tree!

Example - Explanation

Let's see which variables where actually the most useful when making our clustering decisions:

```
varImpPlot(RF_Mod)
```

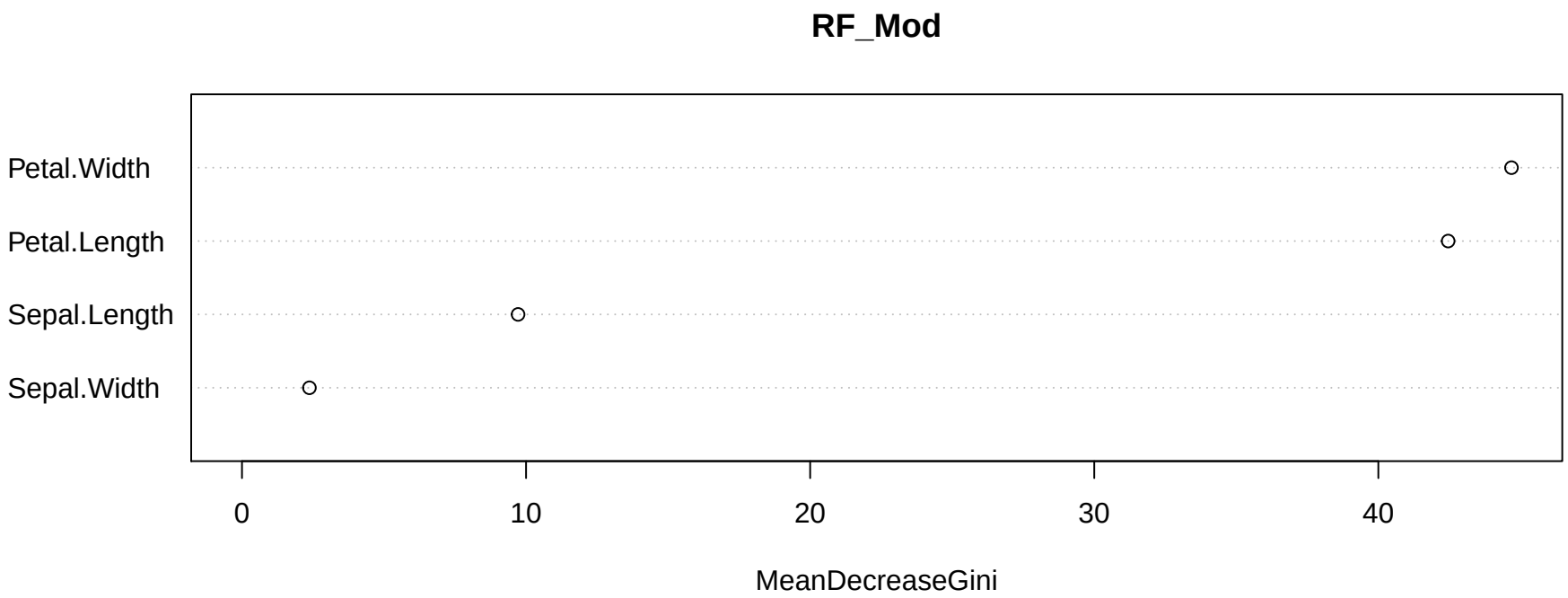


Again, `Petal.Width` and `Petal.Length` alone seem to be almost enough to accurately classify all of our `iris` data in their respective species memberships!

Example - Explanation

Let's see which variables where actually the most useful when making our clustering decisions:

```
varImpPlot(RF_Mod)
```



Again, `Petal.Width` and `Petal.Length` alone seem to be almost enough to accurately classify all of our `iris` data in their respective species memberships!

Theory

Network Clustering

`cluster_optimal()`, etc. in `igraph` package and many others

Purpose:

Identify compartments which are strongly connected within, but not between each other.

Advantages:

- Highly **flexible** approaches.
- Network analyses **offer much more** than clustering.
- Allow for clustering of **very different data** and identification relationships than other approaches.

Disadvantages:

- **Steep learning curve**.
- Tricky in **formatting data correctly**.
- Choices can become **overwhelming**

Theory

Network Clustering

`cluster_optimal()`, etc. in `igraph` package and many others

Purpose:

Identify compartments which are strongly connected within, but not between each other.

Advantages:

- Highly **flexible** approaches.
- Network analyses **offer much more** than clustering.
- Allow for clustering of **very different data** and identification relationships than other approaches.

Disadvantages:

- **Steep learning curve**.
- Tricky in **formatting data correctly**.
- Choices can become **overwhelming**

Example - The Data

Here, we take a foodweb contained within the `foodwebs` data collection of the `igraphdata` package. We are using the Middle Chesapeake Bay in Summer foodweb (Hagy, J.D. (2002) Eutrophication, hypoxia and trophic transfer efficiency in Chesa-peake Bay PhD Dissertation, University of Maryland at College Park (USA), 446 pp.).

```
library(igraph)
library(igraphdata)
data("foodwebs")
Foodweb_ig <- foodwebs[[2]]
```

→ Let's see what kind of network-internal clusters we can make out.

Example - A Directed Network

A **directed network** is one in which we know which node/vertex is acting one which other node/vertex.

We identify the clusters as follows:

```
Clusters <- cluster_optimal(Foodweb_ig)
Colours <- Clusters$membership
Colours <- rainbow(max(Colours))[Colours]
plot(Foodweb_ig,
     vertex.color = Colours,
     vertex.size = degree(Foodweb_ig)*0.5,
     layout=layout.grid, edge.arrow.size=0.001)
```

This identifies sub-networks/clusters by optimizing the modularity score of the overall network (i.e. optimizing connections within vs. between clusters).