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

18/11/2020

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

```r
library(titanic)
titanic_df <- na.omit(titanic_train) # remove NA rows
set.seed(42)
Rows <- sample(1:dim(titanic_df)[1], 50, replace = FALSE)
test_df <- titanic_df[Rows,c(2,3,5,6)] # 50 data for testing
train_df <- titanic_df[-Rows,c(2,3,5,6)] # remaining data for training
head(train_df)
```

```
##   Survived Pclass    Sex Age
## 1        0      3   male  22
## 2        1      1 female  38
## 4        1      1 female  35
## 5        0      3   male  35
## 7        0      1   male  54
## 8        0      3   male   2
```

Can we explain **Survival** (‘Survived‘) based on *Passenger class* (‘Pclass‘), *sex* (‘Sex‘), and *age* (‘Age‘). Was it really "Women and children first"?

Example - The Model

```
Logistic_Mod <- glm(Survived ~., # use all variables in the data frame
                    family = binomial(link = 'logit'), # logistic
                    data = train_df # where to take the data from
                    )
summary(Logistic_Mod)[["coefficients"]]
```

```
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept)  5.11787   0.519980   9.842 7.390e-23
## Pclass      -1.29567   0.143615  -9.022 1.850e-19
## Sexmale     -2.45459   0.214837 -11.425 3.123e-30
## Age         -0.03867   0.007937  -4.872 1.105e-06
```

Logistic Regression Coefficients can't be interpreted the same way as regular linear model coefficients since we are interested in survival probabilities between 0 and 1.

Example - Explanation & Prediction

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 (*sexmale*) 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:

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

```
## [1] 0.2
```

In reality, one would fine-tune the probability at which to assume survivorship!

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

‘mclust‘ is capable of identifying the statistically most appropriate number of clusters for the data set.

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

‘mclust‘ is capable of identifying the statistically most appropriate number of clusters for the data set.

Example - The Data I

```r
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
```

Can we accurately identify the ‘Species‘ contained within the data set by clustering according to ‘Sepal.Length‘, ‘Sepal.Width‘, ‘Petal.Length‘, and ‘Petal.Width‘?

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 II

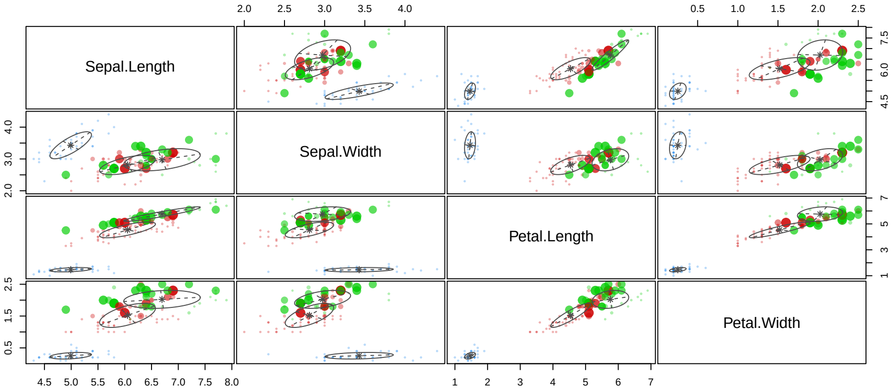
When building a *training* and *test* data set for identification of discrete values, we need to identify data of each group in both data sets. We do so via **stratified sampling**.

```r
library(splitstackshape) # access to the stratified function
set.seed(42) # make sampling reproducible
test_df <- stratified(indt = iris, # input data
           group = "Species", # what the strata are
           size = 7, # how many samples per strata
           keep.rownames = TRUE) # keep the original rownames
training_df <- iris[-as.numeric(test_df$rn), ] # training data
```

Doing this assures that we have data for each group to build a classifier as well as test the validity of our grouping.

Example - The Model I

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



Example - The Model II

Looking at the cluster centres and/or spreads can help with some **biological interpretation**.

```
Mclust_means <- Mclust_mod[["parameters"]][["mean"]] # extract means
colnames(Mclust_means) <- unique(training_df$Species) # set columns
Mclust_means
```

```
##              setosa versicolor virginica
## Sepal.Length 4.9907      6.052     6.696
## Sepal.Width  3.4302      2.811     2.974
## Petal.Length 1.4628      4.539     5.759
## Petal.Width  0.2535      1.521     2.024
```

I prefer a visualization as seen on the previous slide.

Example - Explanation & Prediction

Clearly, `Petal.Length`, and `Petal.Width` are extremely good separators for our different clusters with the green and red clusters (`versicolor` and `virginica`) 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:

```r
Mclust_pred <- predict.Mclust(Mclust_mod, test_df[,-c(1,6)]) # prediction
fitted <- Mclust_pred$classification # predicted species number
# compare actual data with predictions --> ERROR RATE
mean(fitted != as.numeric(test_df$Species))
```

```
## [1] 0.09524
```

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 I

```r
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‘.

Example - The Data II & Model I

‘hclust()‘ can only handle distance matrices.

We a distance matrix between the numeric components of our data like so:

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

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

Now, we can build our initial model:

```
clusters <- hclust(dist_mat, method = "complete")
```

Example - The Data II & Model I

‘hclust()‘ can only handle distance matrices.

We a distance matrix between the numeric components of our data like so:

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

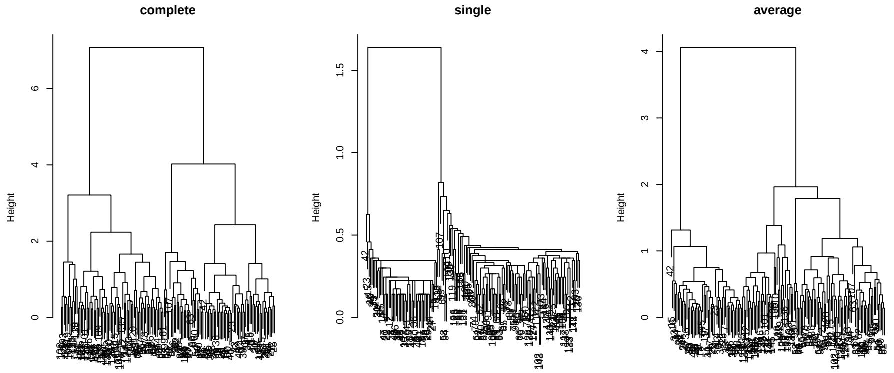
A distance matrix stores information about the dissimilarity of different observations.

Now, we can build our initial model:

```
clusters <- hclust(dist_mat, method = "complete")
```

Example - The Model II

```r
par(mfrow = c(1,3))
plot(clusters, main = "complete")
plot(hclust(dist_mat, method = "single"), main = "single")
plot(hclust(dist_mat, method = "average"), main = "average")
```



Example - Explanation & Prediction

Hierarchical clustering recognises as many groups as there are observation and we may wish to **prune** the decision tree to a meaningful split level.

We know that we have three species in our data, so we may want to cut the complete 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
```

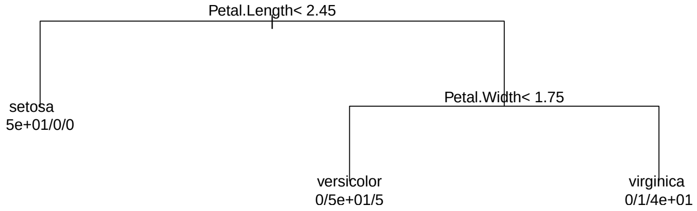
```
##
## clusterCut setosa versicolor virginica
##          1     50          0         0
##          2      0         23        49
##          3      0         27         1
```

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

Example - Decisions

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

```r
library(rpart)
fit <- rpart(Species ~. , data = iris)
plot(fit, margin = .1); text(fit, use.n = TRUE)
```

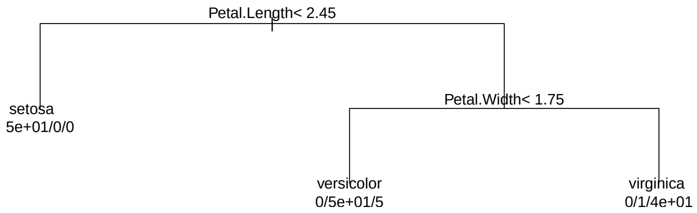


We can tell that our decisions for assigning species membership build on `Petal.Length` and `Petal.Width` in this example (remember the K-mean clustering)!

Example - Decisions

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

```r
library(rpart)
fit <- rpart(Species ~. , data = iris)
plot(fit, margin = .1); text(fit, use.n = TRUE)
```



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)!

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

```
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
```

One final time, we ask whether 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

```
library(randomForest)
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], # stratified sampling
                 doBest = TRUE, # run the best overall tree
                 ntreeTry = 20000, # consider this number of trees
                 improve = 0.0001, # improvement if this is exceeded
                 trace = FALSE, plot = FALSE)
```

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

```
RF_Mod[["confusion"]]
```

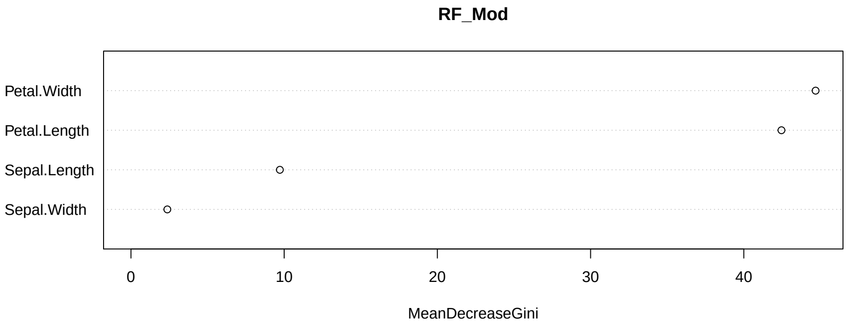
```
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

Example - Explanation

That is one **stunningly accurate** classification!

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

```
varImpPlot(RF_Mod)
```



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.

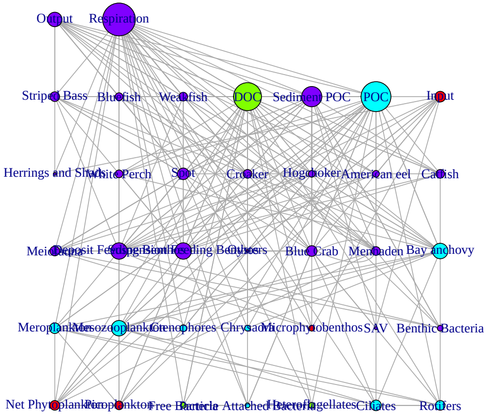
Example - A Directed Network

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).



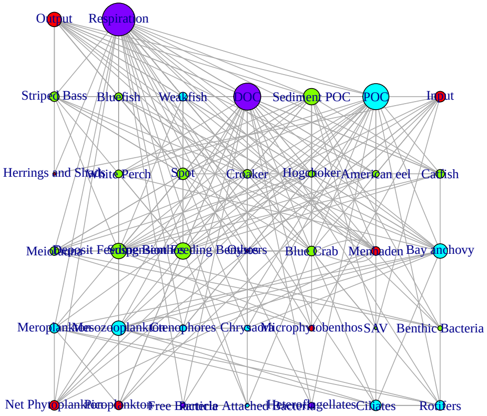
Example - An Undirected Network

An **undirected network** is one in which we don't know which node/vertex is acting one which other node/vertex.

We identify the clusters as follows (there are more options):

```r
Foodweb_ig <- as.undirected(Foodweb_ig)
Clusters <- cluster_fast_greedy(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).


Output
Respiration
Striped Bass
Bluefish
Weakfish
DOC
Sediment POC
POC
Input
Herrings and Shads
White Perch
Spot
Croaker
Hogchoker
American eel
Catfish
Meiofauna
Deposit Feeding Benthos
Suspension Feeding Benthos
Others
Blue Crab
Menhaden
Bay anchovy
Meroplankton
Mesozooplankton
Ctenophores
Chrysaora
Microphytobenthos
SAV
Benthic Bacteria
Net Phytoplankton
Picoplankton
Free Bacteria
Particle Attached Bacteria
Heteroflagellates
Ciliates
Rotifers