unit-10

Agenda

- General Linear Models - logistic
- Networks
- Neural Network: image recognition
- Exercise: building networks with real data
- Additional exercise: application of k-means
- where do we go from here?

General Linear Model

So far we have modelled a response like this:

$$Y \sim N(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p, \sigma^2)$$

- The linear model says that the response is normally distributed with mean βX and residual variance σ^2
- The *general linear model* (GLM) says two things
  - the response doesn't need to be normally distributed, it could be any kind of probablity distribution
  - the mean of the response distribution doesn't need to be βX instead it can be some function of βX. For normal distribution (linear regression), $\beta X = \mu$

- For example, the response, Y, could be poisson distributed, that's called a Poisson regression
- It depends how you want to model your response, what kind of data is it? What kind of probability distribution describes it? *all that probability we learned ain't for nothin'!!*
- If Y is categorical, then modelling it as normally distributed does not make sense because range is the continuum $(-\infty, \infty)$ but categorical Y can only be 0 or 1, discrete etc.

Generalized linear models can be fitted in R using the `glm` function, similar to the `lm` function for fitting linear models

```
glm(formula, family = gaussian, data, weights, subset, na.action,
start = NULL, etastart, mustart, offset, control = glm.control(...),
model = TRUE, method = "glm.fit", contrasts = NULL,...)
```

Example: Logistic Regression

- let the response Y be bernoulli distributed, that means with probability p, $Y = 1$ and probability $1 - p$, $Y = 0$.
- Then $Y \sim bernoulli(p)$
- the mean of y, which is p for the bernoulli distribution comes from the data

$$p(X) = \frac{1}{1 + e^{-\beta X}}$$

- This can also be written as

$$\log(\frac{p}{1-p}) = \beta X$$

---

notation: $\beta X \equiv \eta(\mathbf{x}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$

- So given n observations of both y_i and x_i's, where each y_i is 1 or 0, the parameters β are determined by **maximum likelihood estimation** with no analytical solution
- R will give you the estimates $\hat{\beta}$, everything else works the same way as before, SE etc., p-values etc., the only difference is that we don't have an analytical formula for $\hat{\beta}$

For those who are curious the likelihood is

$$L(\beta) = \prod_i^n p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

The value of β that maximizes $L(\beta)$ is $\hat{\beta}$

We will simulate the following model

$$\eta(x) = -2 + 3x$$
$$p(x) = \frac{1}{1 + e^{-\eta(x)}}$$

```
sim_logistic_data = function(sample_size = 25, beta_0 = -2, beta_1 = 3) {
  x = rnorm(n = sample_size)
  eta = beta_0 + beta_1 * x
  p = 1 / (1 + exp(-eta))
  y = rbinom(n = sample_size, size = 1, prob = p)
  data.frame(y, x)
}
```

- take a moment to think about what we are doing in each line
- x is just some data, that it is normally distributed is irrelevant

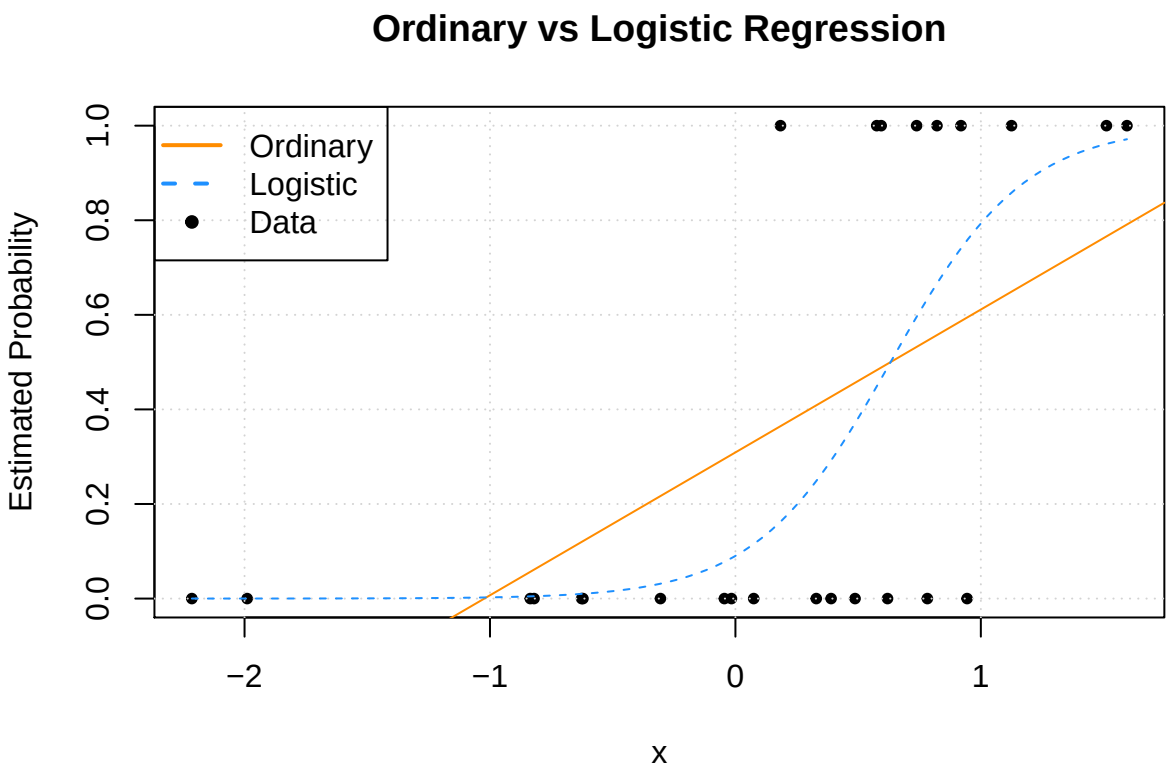
---

```
set.seed(1)
example_data = sim_logistic_data()
head(example_data)
```

```
##   y          x
## 1 0 -0.6264538
## 2 1  0.1836433
## 3 0 -0.8356286
## 4 1  1.5952808
## 5 0  0.3295078
## 6 0 -0.8204684
```

what is the effect on the probability of negative or positive β ?

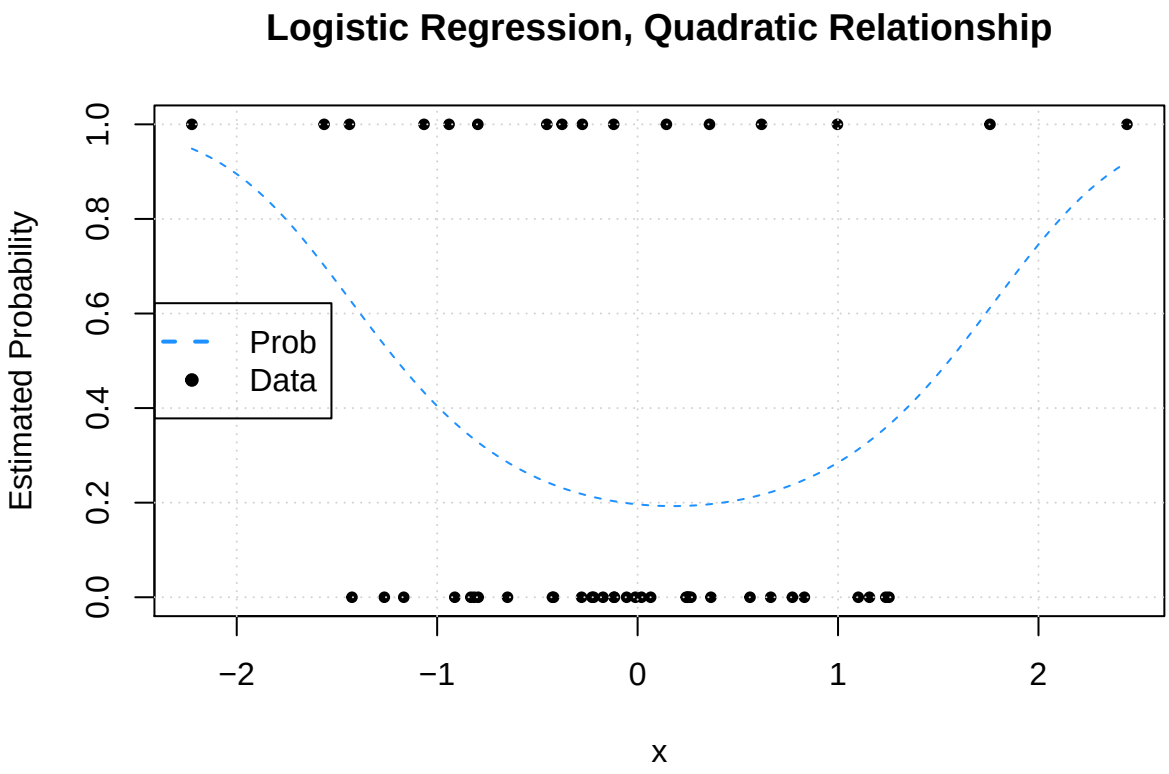
```
# ordinary linear regression
fit_lm  = lm(y ~ x, data = example_data)
# logistic regression
fit_glm = glm(y ~ x, data = example_data, family = binomial)
```



the estimated coefficients are $\hat{\beta}_0, \hat{\beta}_1$ = -2.3, 3.7

Here is another logistic model, showing different behavior

$$\eta(x) = -1.5 + 0.5x + x^2$$



Example: SAheart

```
library(ElemStatLearn)
data("SAheart")
```

sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
160	12.00	5.73	23.11	Present	49	25.30	97.20	52	1
144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
170	7.50	6.41	38.03	Present	51	31.99	24.26	58	1
134	13.60	3.50	27.78	Present	60	25.99	57.34	49	1
132	6.20	6.47	36.21	Present	62	30.77	14.14	45	0

$z = \frac{\hat{\beta}_j - \beta_j}{SE[\hat{\beta}_j]} \sim N(0,1)$, R will will figure out SE for us

```
chd_mod_additive = glm(chd ~ ., data = SAheart, family = binomial)
summary(chd_mod_additive)$coefficients
```

```
##                  Estimate  Std. Error     z value     Pr(>|z|)
## (Intercept)    -6.1507208650 1.308260018 -4.70145138 2.583188e-06
## sbp             0.0065040171 0.005730398  1.13500273 2.563742e-01
## tobacco         0.0793764457 0.026602843  2.98375801 2.847319e-03
## ldl             0.1739238981 0.059661738  2.91516648 3.554989e-03
```

```
## adiposity       0.0185865682 0.029289409  0.63458325 5.257003e-01
## famhistPresent  0.9253704194 0.227894010  4.06052980 4.896149e-05
## typea           0.0395950250 0.012320227  3.21382267 1.309805e-03
## obesity        -0.0629098693 0.044247743 -1.42176449 1.550946e-01
## alcohol         0.0001216624 0.004483218  0.02713729 9.783502e-01
## age             0.0452253496 0.012129752  3.72846442 1.926501e-04
```

---

let's do a step through just as before

```
chd_mod_selected = step(chd_mod_additive, trace = 0)
summary(chd_mod_selected)$coefficients
```

```
##                   Estimate Std. Error   z value     Pr(>|z|)
## (Intercept)    -6.44644451 0.92087165 -7.000372 2.552830e-12
## tobacco         0.08037533 0.02587968  3.105731 1.898095e-03
## ldl             0.16199164 0.05496893  2.946967 3.209074e-03
## famhistPresent  0.90817526 0.22575844  4.022774 5.751659e-05
## typea           0.03711521 0.01216676  3.050542 2.284290e-03
## age             0.05046038 0.01020606  4.944159 7.647325e-07
```

This gives you important predictors for heart disease

At this point we have developed a tool for **classification** since now given a new data point x^* we can make a prediction by determining if $p(x^*) > 0.5$ (or some other choice of cutoff)

Networks

A network is a collection of nodes (vertices) and edges (links)

- Links can be **unweighted** (binary: zero or one), or **weighted**
  - link weight $= 1$ means a link exists between two nodes
  - link weight $= 0$ means there is no link between them
- Links can be **undirected** or **directed** pointing from one node to another, sometimes indicated by the $\pm$ sign of link weight

The standard python module for networks is `networkx`.

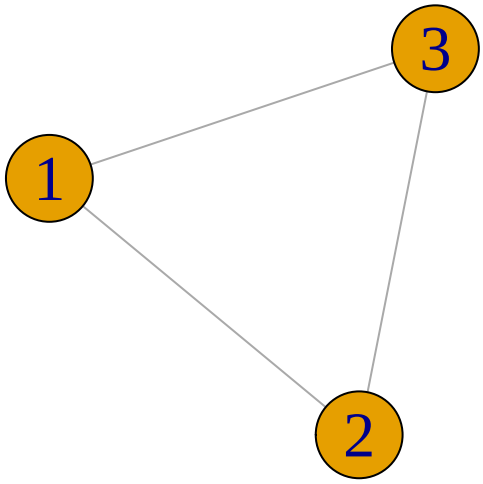
R has several similar tools we will use `igraph`

```
library("igraph")
```

---

you can build networks with the syntax: `graph(...)`

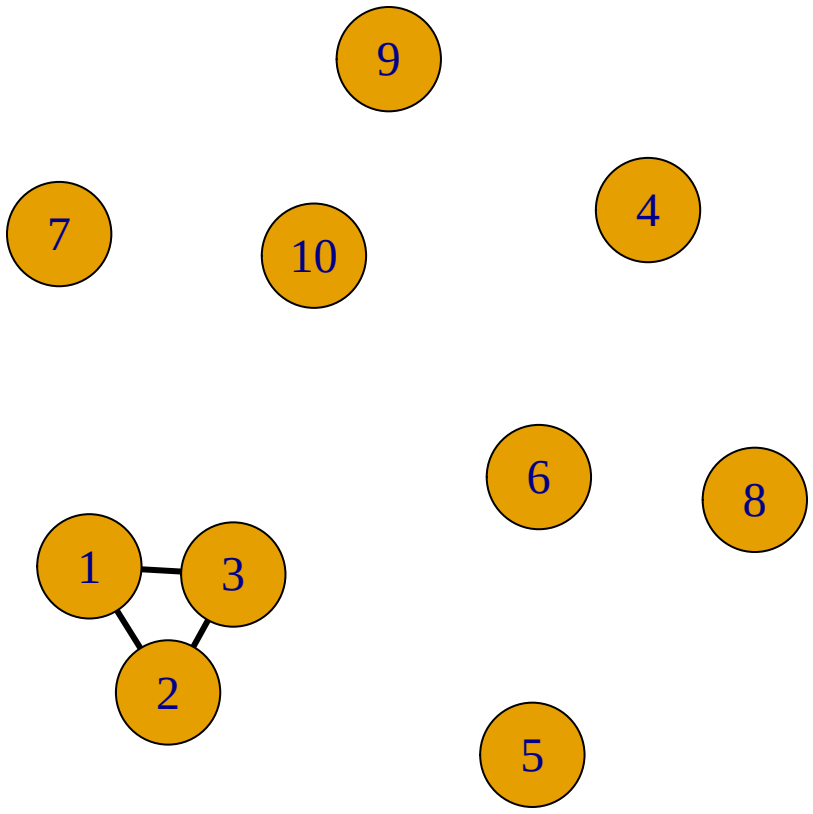
```
g1 <- graph(edges = c(1,2,2,3,3,1),n=3,directed=F)

plot(g1,vertex.size=c(45,45,45),vertex.label.cex=c(2,2,2))
```



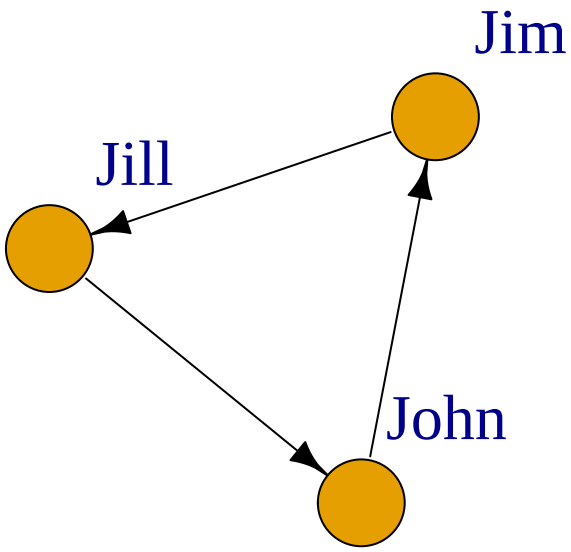
---

let's do a few more examples so you can get the hang of it

```
g2 <- graph(edges = c(1,2,2,3,3,1),n=10,directed=F)
```

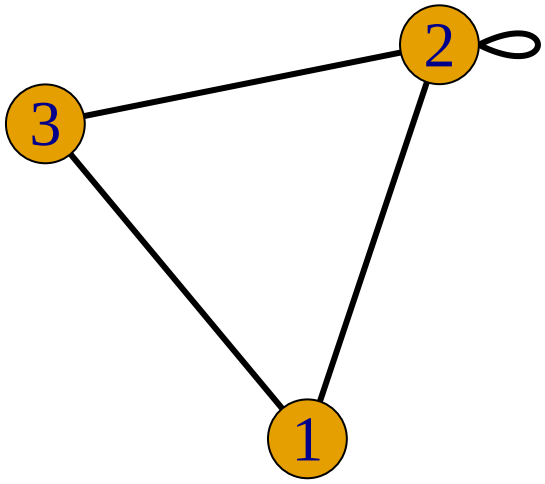


```
g3 <- graph(c("John","Jim","Jim","Jill","Jill","John")) # Directed

plot(g3, vertex.label.dist=8.5,vertex.size=c(45,45,45),
     vertex.label.cex=c(2,2,2),edge.color=rep("black",3))
```

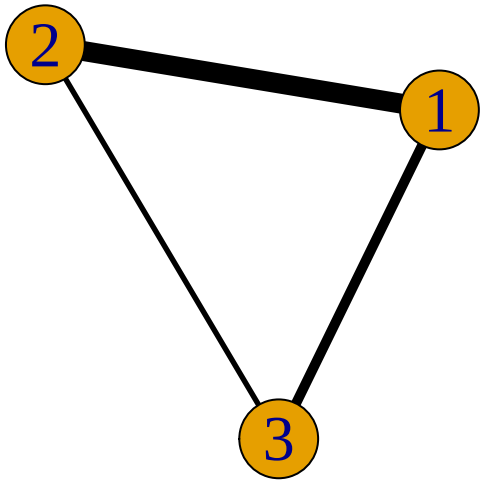


A link that points back to the same node is called a *loop*

```
g4 <- graph(edges = c(1,2,2,3,3,1,2,2),n=3,directed=F)
```




```
g5 <- graph(edges = c(1,2,2,3,3,1),n=3,directed=F)

E(g5)$weight <- c(10,2.72,5) # give a weight to each edge
```




```
E(g5)  # get the edges
```

```
## + 3/3 edges from 644c3ef:
## [1] 1--2 2--3 1--3
```

```
V(g5)  # show the vertices
```

```
## + 3/3 vertices, from 644c3ef:
## [1] 1 2 3
```

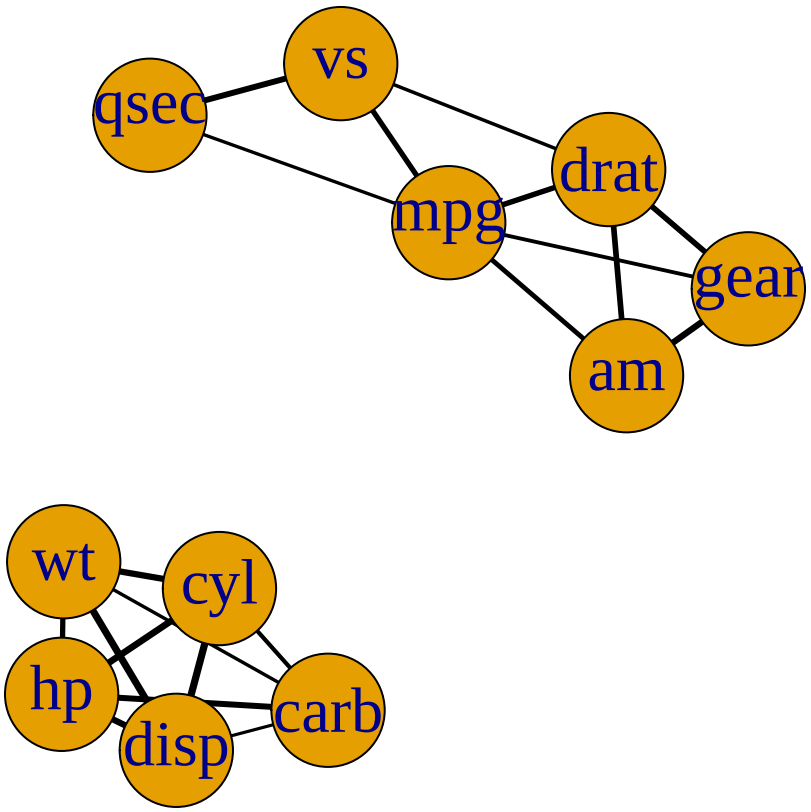
```
g5[]   # adjacency matrix
```

```
## 3 x 3 sparse Matrix of class "dgCMatrix"
##
## [1,]  . 10.00 5.00
## [2,] 10  .    2.72
## [3,]  5  2.72 .
```

Link weights from correlations

- A standard method of constructing networks is from correlations between nodes
- In this case nodes could be variables (predictors/features), or observations
- You can build a network where the links weights are the values in a matrix using `graph_from_adjacency_matrix()`

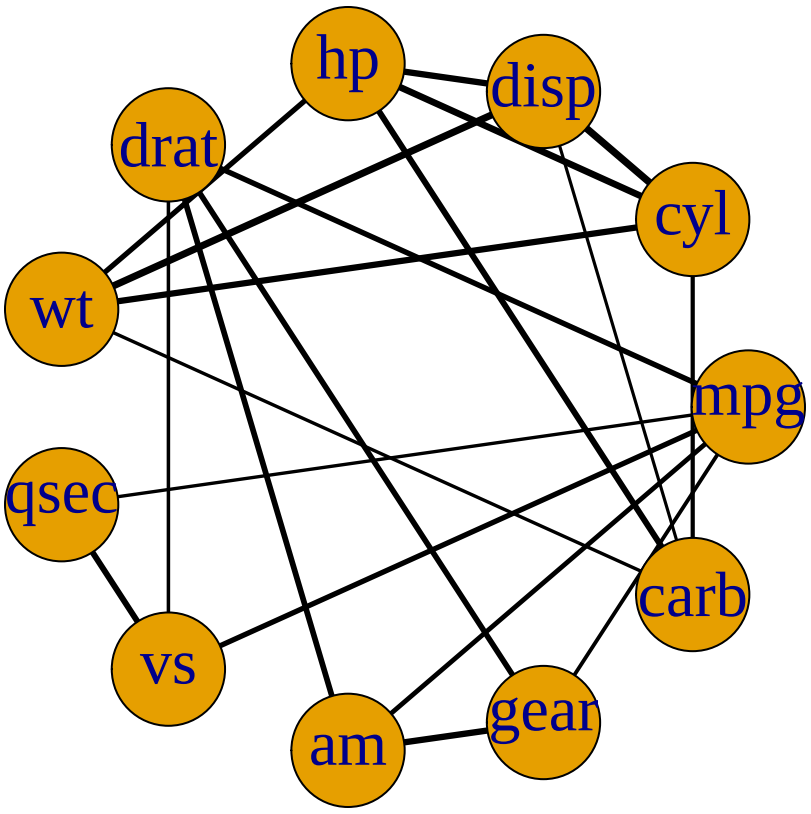
```r
x <- cor(mtcars)
x[x<0.3] <- 0 #keep only high correlations
g6 <- graph_from_adjacency_matrix(x,
                        weighted = T,mode="undirected",diag=F)
```



The network is just serving to visualize correlations between features

---

network visualization is generally a complex task You can do circular layout with: `layout=layout.circle(graph)`



---

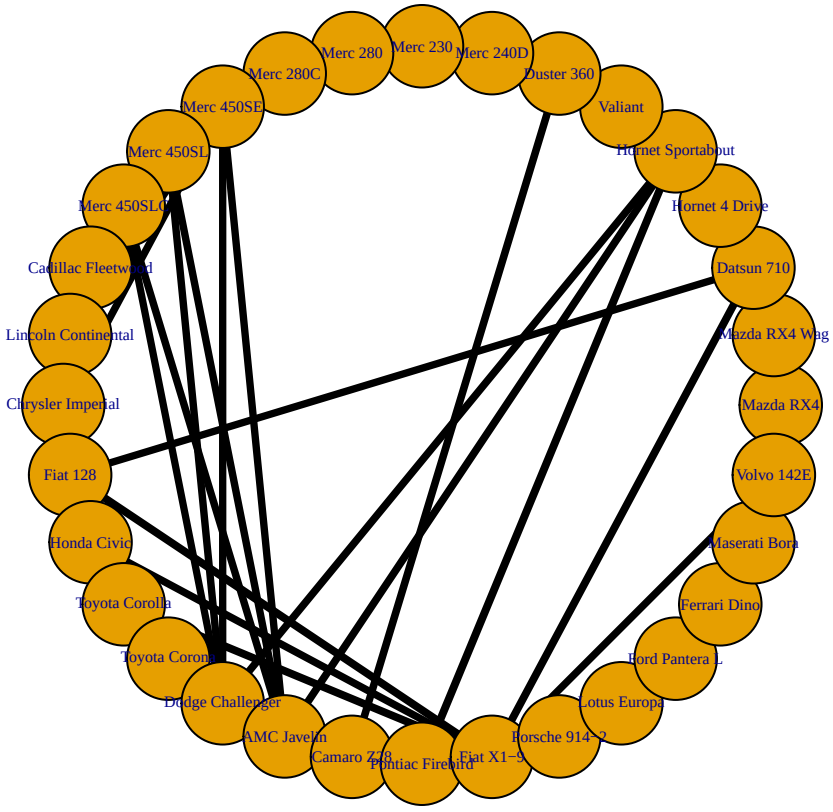
notice, we computed correlations between features.

If we wanted correlations between different observations we should compute `cor(t(mtcars))`

then the network nodes are the cars themselves, not features

we should probably scale the dataframe as well so that variables are comparable

```
scaled_mtcars <- scale(mtcars)
x <- cor(t(scaled_mtcars))
x[x<0.9] <- 0 #keep only high correlations
g7 <- graph_from_adjacency_matrix(x,
                      weighted = T,mode="undirected",diag=F)
```

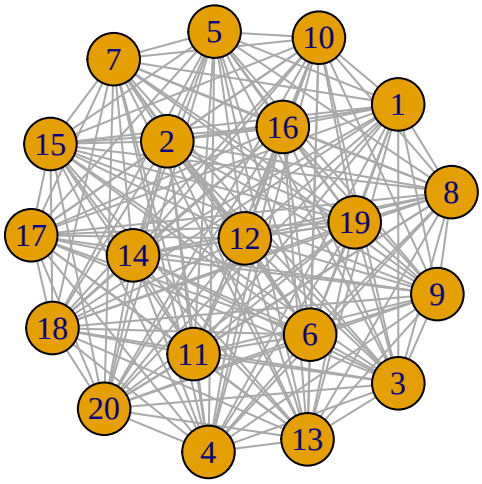


These are cars from 1973-1974, so I don't know much about these models, but inutitively the maseratti and porsche being correlated makes sense since they are both high end sports cars

Specific graph models

A full graph

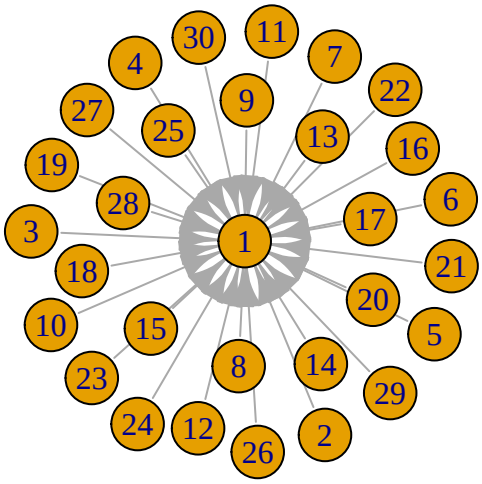
```
fcg <- make_full_graph(20)  # 20 nodes
```



---

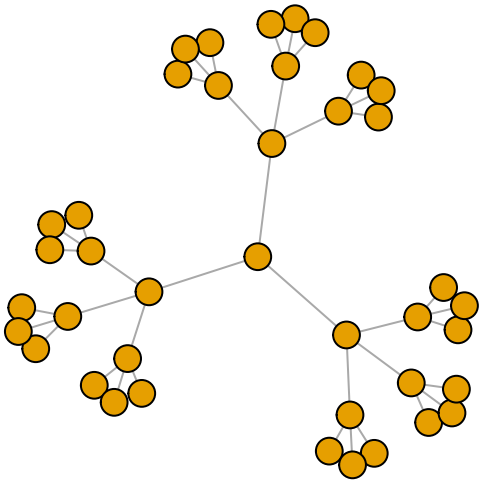
star graph

```
st <- make_star(30)  # 30 nodes
```



Tree graph

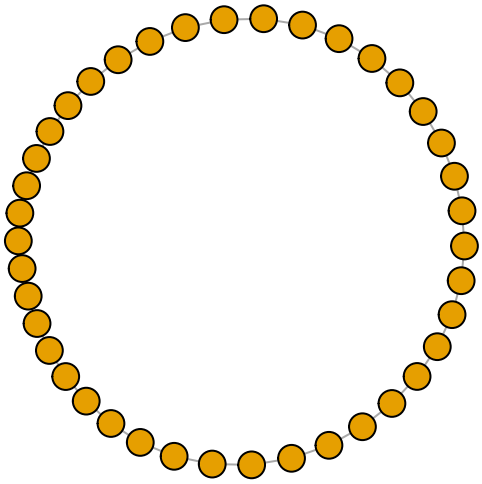
```
tr <- make_tree(40,children=3,mode="undirected")  # 40 nodes, 3 childs per node
```



---

ring graph

```
rn <- make_ring(40)
```



Random networks

There are two, essentially same, types of random networks

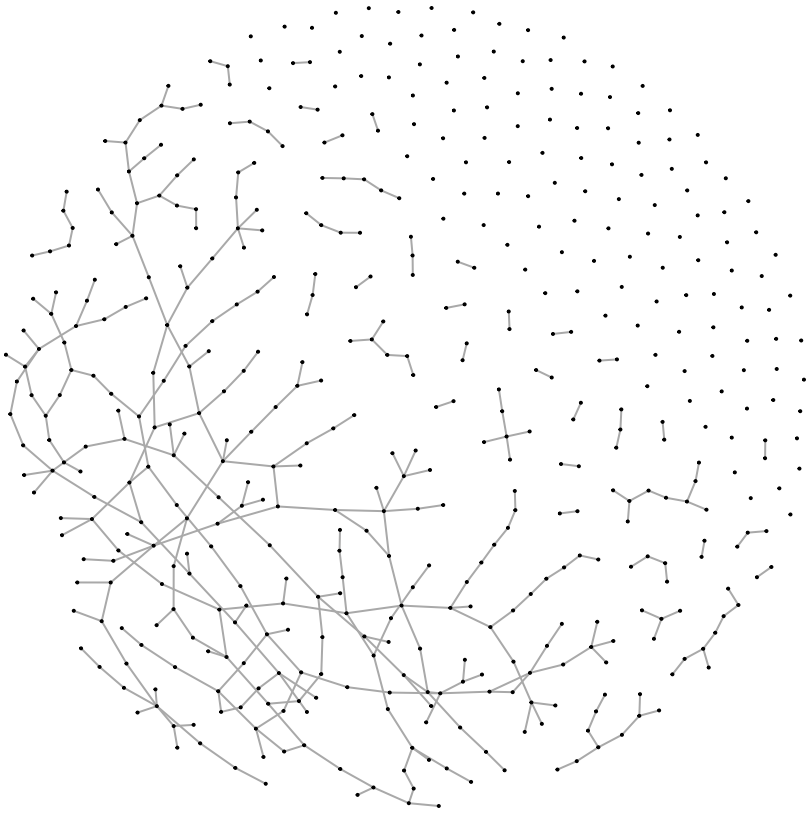
- The Gilbert random graph $G(N, p)$: each pair of N nodes is connected with probability p
- The Erdos-Renyi random graph $G(N, L)$: N nodes are connected with L randomly placed links

Paul Erdos showed that these types of graphs undergo a *phase transition* between full connected and not fully connected at a critical probability p_c/L_c

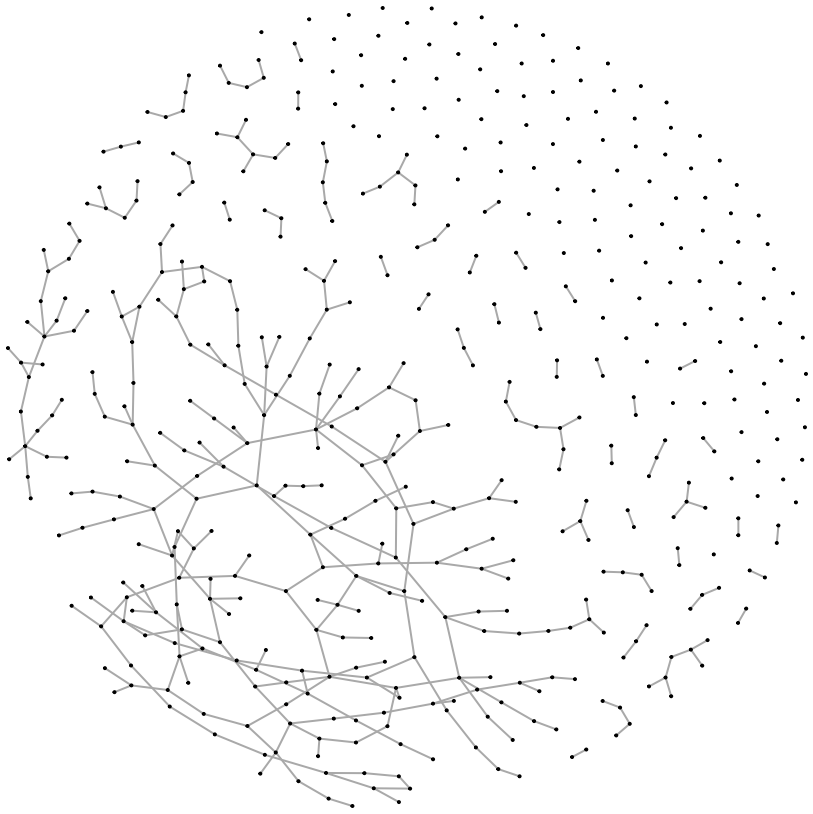
- There is a difference between a **full graph** and a ***fully connected graph*** - the latter meaning there exists a path from every node to every other node, whereas in a full graph there is a direct link between every pair of nodes

---

```
g <- erdos.renyi.game(500,350,type="gnm")
par(mar=c(0,0,0,0))
plot(g,vertex.label=NA,edge.arrow.size=0.02,vertex.size=0.5)
```



Note that if all connections were present $L_f = \frac{N(N-1)}{2}$, then we can create an equivalent Gilbert graph with $p = 350/L_f = 350/124750 = 0.0028$



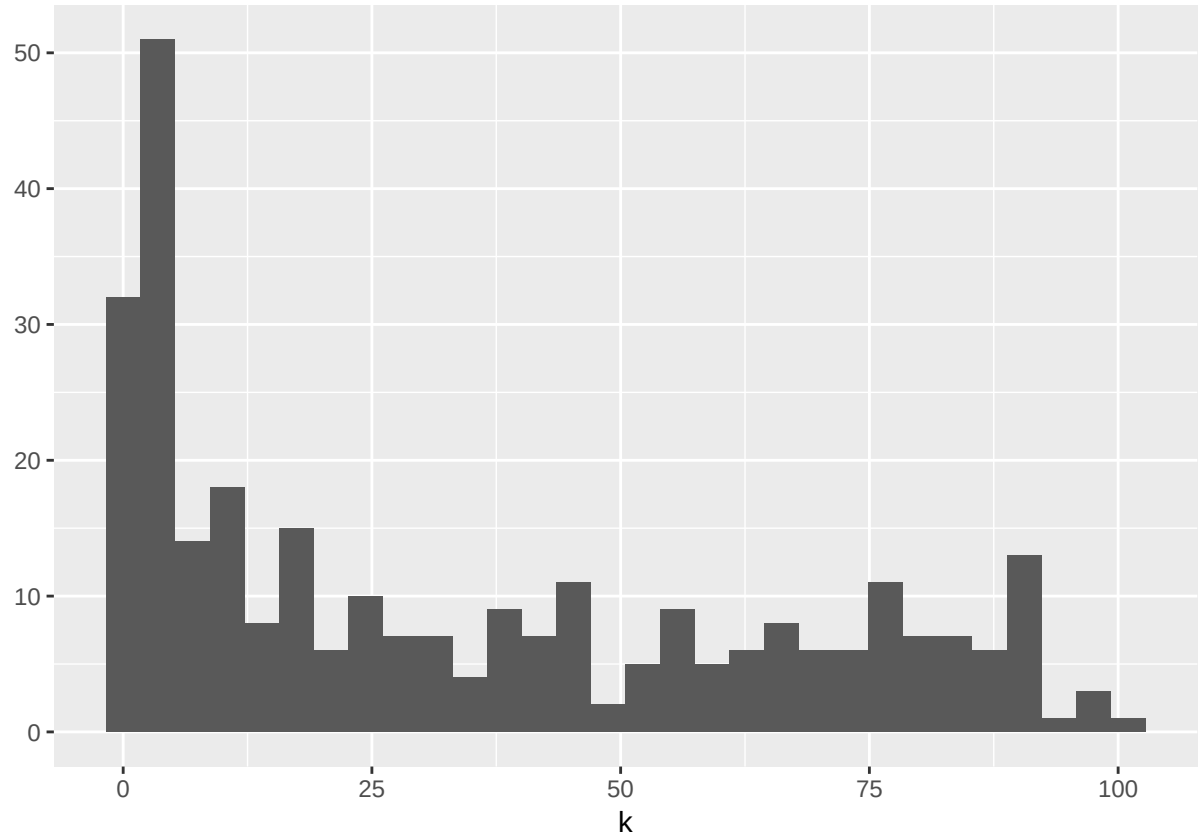
Network properties

- The degree, k, of a vertex counts the number of links to it
- The *degree distribution* is normalized histogram of degrees
- For a random graph, $G(N, p)$,

$$P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}$$

- The Internet, and some social networks have degree distributions that approximately are power law: $P(k) \sim k^{-\gamma}$, where γ is a constant. These are called **scale-free** networks

```
g <- erdos.renyi.game(100000,0.5,type="gnp")
g.degrees <- degree(g) # get degrees of all nodes
```



For the random network, the degree distribution is Poisson in the limit of large n.

Small world networks

- A network is **small world** if the average *shortest path distance* between any two nodes is small (in some sense)
  - For example, the average shortest path distance between people on earth is about 6 (six degrees of separation)
- the `distance(G)` function gives shortest path distances between all connected nodes

`sample_smallworld()` creates a WS small world network

```
small_world <- sample_smallworld(1, 100, 5, 0.05)
mean_distance(small_world)
```

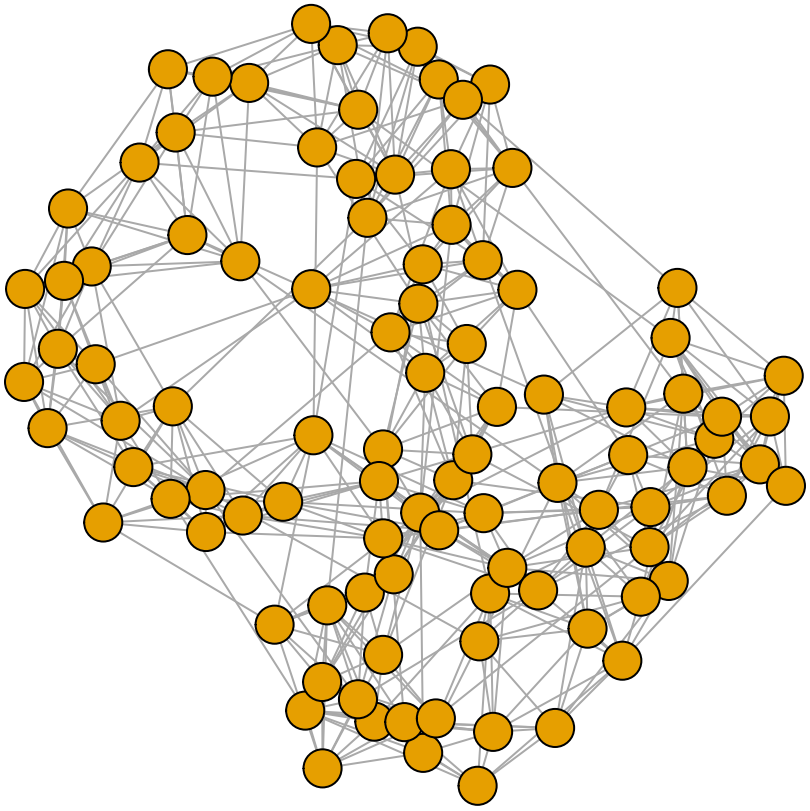
```
## [1] 2.702626
```

The Watts-strogatz model locally clusters nodes with mean degree K on the lattice within a neighborhood $K/2$. Links are then randomly rewired with probability β

- in 1D we chose $K = 10$ or a neighborhood of $K/2 = 5$
- thus there will be $NK/2$ links (edges) in the graph
- If β (the rewiring probability) =1 then you approach the random graph $G(N, p = \frac{K}{N-1})$

- random graphs also have the small world property, **BUT** one property of real world graphs, such as social networks, that is absent from random graphs is presence of a large degree of transitivity.

Visualization of the Watts-Strogatz small-world network



- **transitivity**: if A is connected to B and B is connected to C what is the probability that A is connected to C?
- Since your friends are likely to be friends, the social network describing these friendship is likely to have many more triangles than predicted by the random network model.

```
random_net <- erdos.renyi.game(100,0.1,type="gnp")
transitivity(random_net)
```

```
## [1] 0.09762131
```

```
transitivity(small_world)
```

```
## [1] 0.4968111
```

watts-strogatz: **both** high transitivity and small world property

---

For comparison, and to test that you understand:

What is an example of a graph that would NOT be small world?

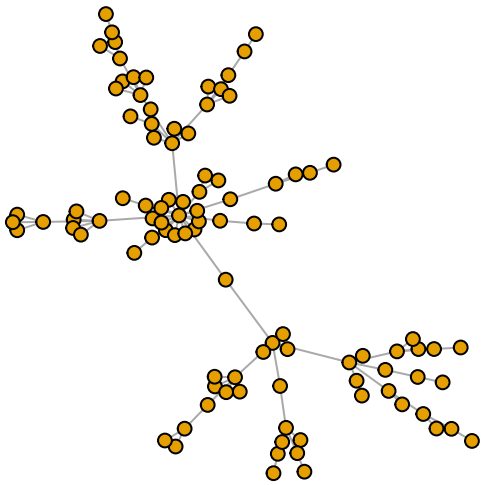
Hint:

```
rn = make_ring(100)
mean_distance(rn)
```

```
## [1] 25.25253
```

- WS has the small world property but lacks **hubs**
- Through method of preferential attachment, the barabassi-albert network creates a **scale-free** network with hubs

```
ba <-  sample_pa(n=100, power=1, m=1,  directed=F)
plot(ba, vertex.size=6, vertex.label=NA)
```



```
mean_distance(ba)
```

```
## [1] 5.356566
```

```
transitivity(ba)
```

```
## [1] 0
```

The BA network does not have small world or transitivity but it does have nodes with very high degree or a **long-tailed power law degree distribution** which WS networks don't have

- real networks are not like WS or BA but rather some *combination* of the two having all kinds of properties.

- You will get a chance to load a real data set to explore networks in the exercises
- not covered: centrality measures, community detection,...

Switching gears

Building a Neural Network in R

I will go over a simple neural network, in R using keras and tensorflow to do image recognition

- The workflow is exactly the same as python, no real difference. Your preference really.
- I will be unable to describe CNNs fully though I will try a little bit (I am not an expert). Hopefully, your ML/Deep Learning course will (have already) explained more

What is a neural network doing?

- Really, it is putting all the pixels as an independent predictor
- so a 20×20 pixel image has $p = 400$ predictors, one for each pixel
- these pixels are considered nodes of a network -> next links are created to new nodes in the hidden layer
- what are these "hidden layer nodes"
- they are just combinations of the pixels
- In theory, it should be possible to represent a neural network by a linear model

Image classifier in R

Image recognition is similar to a logistic regression problem.

But the dimensionality of the data is much greater and feature selection is far more difficult problem - this is why Neural Networks are used to build the model.

An additive model where each pixel has independent coefficient is just not going to work.

Doing image classification in R is basically the same as in python

I will skip some of the theoretical aspects of neural networks and how they work (e.g. backpropogation etc.)

```
library(keras)       # install in anaconda prompt
library(tensorflow) # pip install tensorflow
```

---

data: images of fruits from kaggle.com

```
fruit_list <- c("Kiwi", "Banana", "Apricot", "Avocado","Clementine",
                "Orange","Limes", "Lemon","Peach", "Plum",
               "Raspberry", "Strawberry", "Pineapple", "Pomegranate")

# number of output classes (i.e. fruits)
output_n <- length(fruit_list)
```

Image properties

```
# image size to scale down to (original images are 100 x 100 px)
img_width <- 20
img_height <- 20
target_size <- c(img_width, img_height)

# RGB = 3 channels
channels <- 3
```

set-up data

```
# path to image folders
train_image_files_path <- "fruits/Training/"
valid_image_files_path <- "fruits/Test/"     #validation
```

```
train_data_gen = image_data_generator(
  rescale = 1/255)

valid_data_gen <- image_data_generator(
  rescale = 1/255)
```

next, we will load the images into memory

```
train_image_array_gen <-
  flow_images_from_directory(train_image_files_path,
                                          train_data_gen,
                                          target_size = target_size,
                                          class_mode = "categorical",
                                          classes = fruit_list,
                                          seed = 42)

# validation images
valid_image_array_gen <-
  flow_images_from_directory(valid_image_files_path,
                                          valid_data_gen,
                                          target_size = target_size,
                                          class_mode = "categorical",
                                          classes = fruit_list,
                                          seed = 42)
```

```
table(factor(train_image_array_gen$classes)) # number of each fruits
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13
## 466 474 260 427 330 479 426 428 300 351 490 228 362 492
```

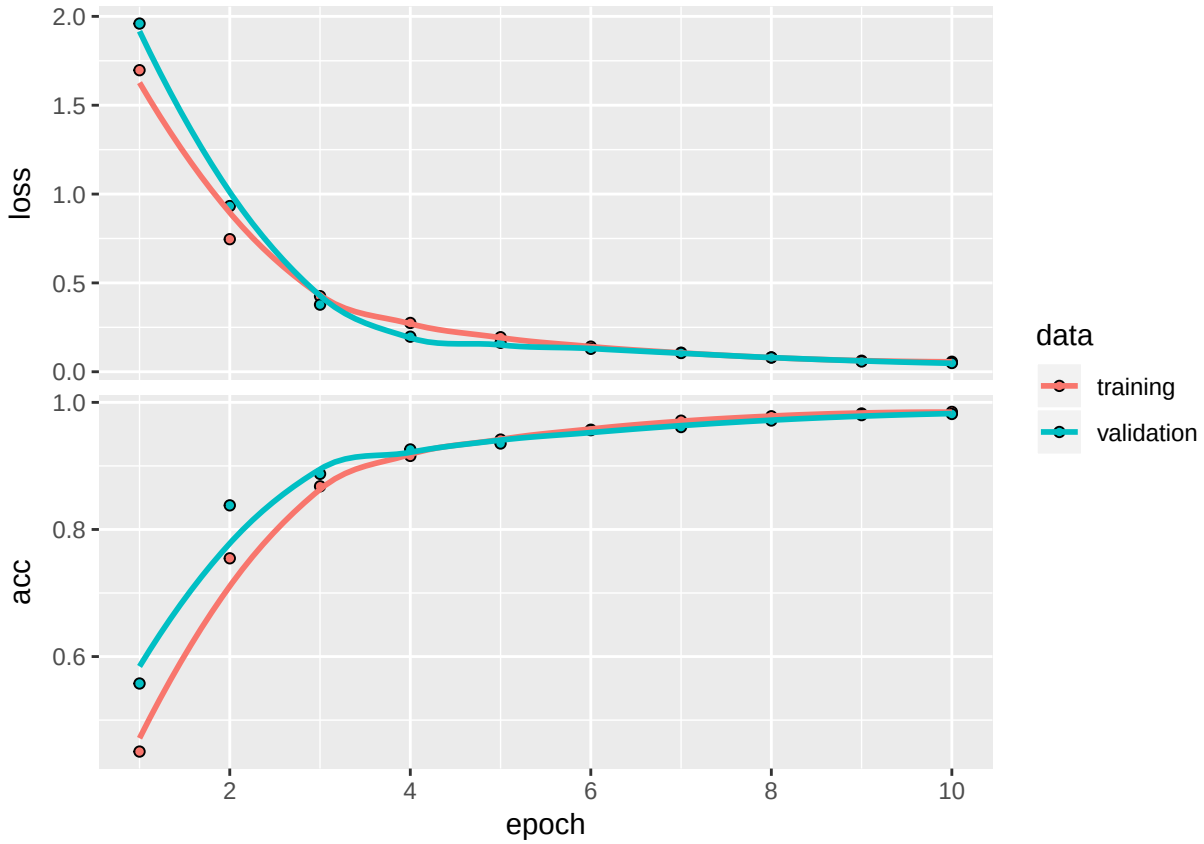
```
train_samples <- train_image_array_gen$n #Number of training samples

valid_samples <- valid_image_array_gen$n # number of validation samples

batch_size <- 32
```

```
epochs <- 10

model <- keras_model_sequential()
```

Here we build the neural network, setting all the parameters, this is a complex process about which I do not know enough about, nor do I have time to explain here

```
model %>%
  layer_conv_2d(filter=32,kernel_size=c(3,3),padding="same",
          input_shape = c(img_width,img_height,channels)) %>%
          layer_activation("relu") %>%
          layer_conv_2d(filter = 16,kernel_size=c(3,3),padding="same") %>%
          layer_activation_leaky_relu(0.5) %>%
          layer_batch_normalization() %>%
          layer_max_pooling_2d(pool_size = c(2,2)) %>%
          layer_dropout(0.25) %>%
          layer_flatten() %>%
          layer_dense(100) %>%
          layer_activation("relu") %>%
          layer_dropout(0.5) %>%      # dropout is related to overfitting
          layer_dense(output_n) %>%
          layer_activation("softmax")
```

```
model %>% compile(loss="categorical_crossentropy",
                  optimizer = optimizer_rmsprop(lr=0.0001,decay=1e-6),
                  metrics="accuracy")

hist <- model %>% fit_generator(
  train_image_array_gen,
  steps_per_epoch = as.integer(train_samples/batch_size),
  epochs=epochs,
  validation_data = valid_image_array_gen,
  validation_steps = as.integer(valid_samples/batch_size),
  verbose=2
)
```



The validation accuracy is better than the training accuracy due to the use of a dropout, removes neurons to prevent overfitting (like LOOCV)

Network Exercise

To build your understanding and practice with networks of real data, please **follow all instructions, and execute all the code below**. The dataset is about media outlets and links between them

An **edge list** is a format for describing the links of a network where each row is of the form: node1 node2 weight. Other attributes can also be included in subsequent columns

1. Download the data from my github lecture notes- unit 10 folder, and then load the files and then inspect

```
nodes <- read.csv("Dataset1-Media-Example-NODES.csv", header=T, as.is=T)

links <- read.csv("Dataset1-Media-Example-EDGES.csv", header=T, as.is=T)
```

click on the objects in the environment panel and inspect

```
head(nodes)

head(links)

nrow(nodes);

length(unique(nodes$id))
```

```
nrow(links);

nrow(unique(links[,c("from", "to")]))
```

Notice that there are more links than unique from-to combinations. That means we have cases in the data where there are multiple links between the same two nodes. We will collapse all links of the same type between the same two nodes by summing their weights, using aggregate() by "from", "to", & "type". We don't use simplify() here so as not to collapse different link types.

```
links <- aggregate(x=links[,3], by=links[,-3], FUN=sum)

links <- links[order(links$from, links$to),]

colnames(links)[4] <- "weight"

rownames(links) <- NULL
```

We start by converting the raw data to an igraph network object. Here we use igraph's `graph.data.frame` function, which takes two data frames: **d** and **vertices**.

- **d** describes the edges of the network. Its first two columns are the IDs of the source and the target node for each edge. The following columns are edge attributes (weight, type, label, or anything else).
- **vertices** starts with a column of node IDs. Any following columns are interpreted as node attributes.

```
library(igraph)

net <- graph_from_data_frame(d=links, vertices=nodes, directed=T)

class(net)

net
```

We also have easy access to nodes, edges, and their attributes with:

```
E(net)       # The edges of the "net" object

V(net)       # The vertices of the "net" object

E(net)$type  # Edge attribute "type"

V(net)$media # Vertex attribute "media"
```

---

try plotting the network

```
plot(net, edge.arrow.size=.4,vertex.label=NA)
```

If you need them, you can extract an edge list or a matrix from igraph networks.

```
as_edgelist(net, names=T)

as_adjacency_matrix(net, attr="weight")
```

Or data frames describing nodes and edges:

```
as_data_frame(net, what="edges")

as_data_frame(net, what="vertices")
```

Additional Exercise: k-means

we will cluster data of household income. Download the data below it could take some time because it is big data

setup:

```
install.packages("reldist")
library(tidyverse)
library("reldist")
library(readxl)
if (!file.exists("SCFP2016.xlsx")){
  download.file(   "https://www.federalreserve.gov/econres/files/scfp2016excel.zip",   "SCFP2016.zip")
  unzip("scfp2016.zip")}

df <- read_excel("SCFP2016.xlsx") # takes a while
```

WGT column represents the weight of the observation based on whether the household is representative of a typical US household.

```
# How many survey participants?

nrow(df)

# How many households does the survey represent?

floor(sum(df$WGT)))

# What is the weighted mean of household net worth?

floor(sum(df$NETWORTH*df$WGT)/sum(df$WGT)))

# what is the median NW in US?

reldist::wtd.quantile(df$NETWORTH, q=0.5, weight = df$WGT)
```

```
# who is the 1%?

reldist::wtd.quantile(df$NETWORTH, q=0.99, weight = df$WGT)

#top 0.1% Ultra-high net worth households

reldist::wtd.quantile(df$NETWORTH, q=0.999, weight = df$WGT)
```

Now, let's try to cluster, we need to reduce the data a bit

```
df2 <- select(df,INCOME,NETWORTH,STOCKS)

k6 <- kmeans(df2, centers=6)
```

Finally plot the result

```
fviz_cluster(k6, geom = "point", data = df2) + ggtitle("k = 6")
```