



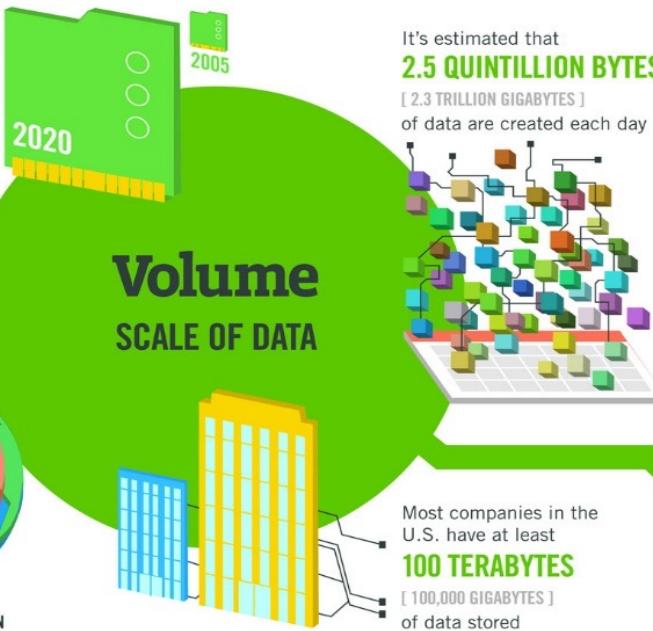
MACHINE LEARNING IN R
Research Technologies

Machine Learning
with **R**



40 ZETTABYTES

[43 TRILLION GIGABYTES]
of data will be created by
2020, an increase of 300
times from 2005



Velocity ANALYSIS OF STREAMING DATA

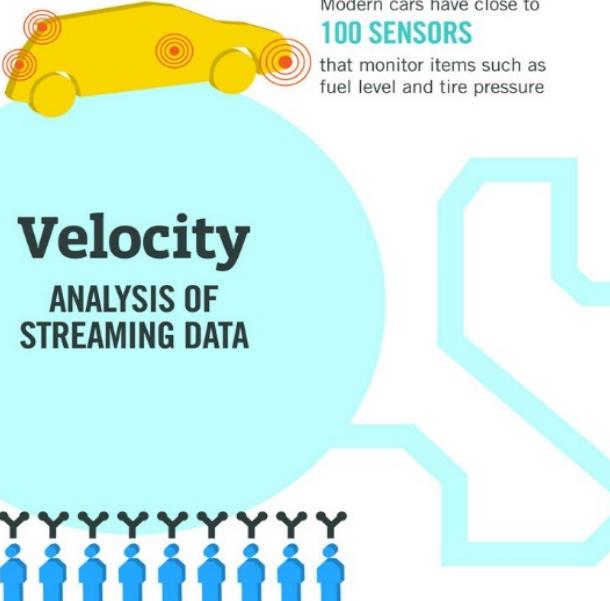
The New York Stock Exchange captures

1 TB OF TRADE INFORMATION

during each trading session



Modern cars have close to 100 SENSORS that monitor items such as fuel level and tire pressure



By 2016, it is projected there will be

18.9 BILLION NETWORK CONNECTIONS

– almost 2.5 connections per person on earth



The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume**, **Velocity**, **Variety** and **Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

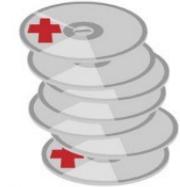
By 2015
4.4 MILLION IT JOBS

will be created globally to support big data, with 1.9 million in the United States



As of 2011, the global size of data in healthcare was estimated to be

150 EXABYTES
[161 BILLION GIGABYTES]



30 BILLION PIECES OF CONTENT

are shared on Facebook every month

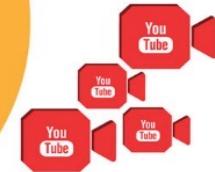


Variety DIFFERENT FORMS OF DATA

By 2014, it's anticipated there will be

420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month



400 MILLION TWEETS

are sent per day by about 200 million monthly active users



1 IN 3 BUSINESS LEADERS

don't trust the information they use to make decisions

27% OF RESPONDENTS

in one survey were unsure of how much of their data was inaccurate

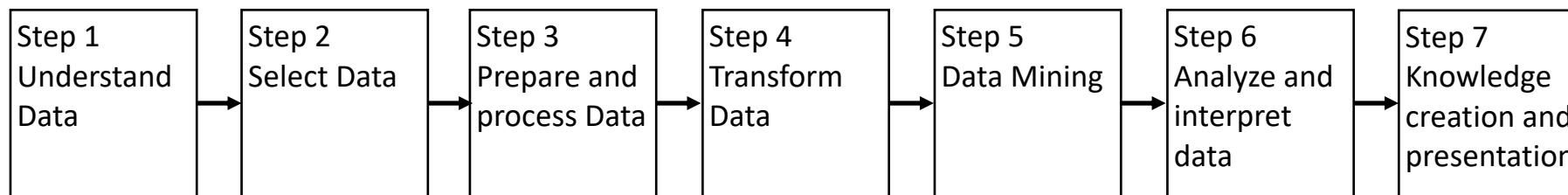


Veracity UNCERTAINTY OF DATA

IBM

Knowledge Discovery Process

- ▶ The Knowledge Discovery process takes 7 steps
 - ▶ Step 1: Understand your business goal and your data
 - ▶ Step 2: Select your data
 - ▶ Step 3: Prepare and process your data
 - ▶ Step 4: Transform your data
 - ▶ Step 5: Data mining
 - ▶ Step 6: Analyze, interpret the data and patterns
 - ▶ Step 7: Knowledge creation and presentation



Knowledge Discovery Process

- ▶ STEP 1: Understand your business goal and your data
 - ▶ Depending on your specific goal, the model you choose will be different
 - ▶ Exercise: For an electronics store, what type of factors lead to high return rate?

Knowledge Discovery Process

- ▶ STEP 2: Select your data
 - ▶ Figure out which data you need to solve your problem
 - ▶ This is important since preprocessing is very time consuming, you want to focus on that data that you will actually use
 - ▶ There will be problems when you get to this step (guaranteed!)
 - ▶ Incompatibilities in the databases or data files
 - ▶ Data may be incomplete
 - ▶ Data may be obscure
 - ▶ Note: this step can also come later (again)
 - ▶ Exercise: For the electronics store return rate problem, what data can we get (existing), could we create (start collecting)?

Knowledge Discovery Process

- ▶ STEP 3: Prepare and process your data
 - ▶ Very time consuming
 - ▶ Data can come from legacy systems or external sources
 - ▶ The data will need to be
 - ▶ Cleaned, integrated
 - ▶ Selected (again)
- ▶ There will be problems/issues
 - ▶ Incomplete: missing values or aggregates only
 - ▶ Noisy: missing, duplicated, erroneous or outliers
 - ▶ Inconsistent: birthday is 1 Jan 1993, 010193, 01-01-1993
 - ▶ Intentional: default values if not specified

Knowledge Discovery Process

- ▶ STEP 3: Prepare and process your data
 - ▶ Nominal -> name
 - ▶ For a person: name, SSN, age, education
 - ▶ Some numbers can be used for calculation like age; and some cannot, like SSN
 - ▶ Some values are ranked: Cancer stages 1,2,3,4
 - ▶ Binary -> only two values
 - ▶ Boolean: True or False, organic or inorganic
 - ▶ Symmetric: if both values are equally important
 - ▶ Asymmetric: if one is more important – use 1 for more important
 - ▶ Numeric
 - ▶ Scale with units of equal size
 - ▶ Values can be +, -, or 0 (0 can be nothing or something like weight or temp)
 - ▶ Ratio scale
 - ▶ All characteristics of interval + true zero
 - ▶ Able to compute mean, median, standard deviation etc

Knowledge Discovery Process

- ▶ STEP 4: Transform your data
 - ▶ Different type of transformations exist
 - ▶ Feature construction = applying mathematical formulas to existing data features
 - ▶ Feature subset selection = selecting which feature to use, which database columns to use
 - ▶ Aggregating data = sometimes, you will want to use only the average, or sum, or maximum, or minimum, or other groupings (per store, region, time unit)
 - ▶ Bin the data = breaking up continuous ranging in discrete segments (e.g. per month)

Knowledge Discovery Process

- ▶ STEP 5: Data mining
- ▶ Different models exist:
 - ▶ Pattern identification and description = link analysis, market basket analysis
 - ▶ E.g. People who buy soda, usually buy chips
 - ▶ E.g. People who visit website A, usually visit website B
 - ▶ Classification, prediction = supervised learning
 - ▶ E.g. if you work hard and you are smart -> you will get an A
 - ▶ Database segmentation, clustering = unsupervised learning:
 - ▶ E.g. documents covering the same topic (e.g., for similar documents, click here)
- ▶ For each model, you can choose a method to build your model
 - ▶ Classification: decision trees, neural networks
 - ▶ Prediction: linear regression
- ▶ For each method, you need to choose the algorithm
 - ▶ E.g. decision trees: ID3, C4.5, CART

Knowledge Discovery Process

- ▶ STEP 6: Analyze, interpret the data and patterns
 - ▶ Once the model is build -> learn the implications
 - ▶ Sometimes visualization can help
 - ▶ e.g. clusters of documents, outliers
- ▶ STEP 7: Knowledge creation and presentation
 - ▶ Use other resources to understand and interpret the end product
 - ▶ Tie back to original goal

Classification of Algorithms - Based on Technique

	Characteristics	Pro	Con
Statistical Techniques e.g. regression analysis	<ul style="list-style-type: none"> ▶ usually involves numbers ▶ make strong assumptions about your model, e.g. normal distribution 	<ul style="list-style-type: none"> ▶ elegant, theory based ▶ usually fast to train and test ▶ accurate ▶ tools available (SPSS, SAS, Minitab) ▶ results easy to understand 	<ul style="list-style-type: none"> ▶ you need to know what to apply when ▶ need to be a good statistician (not an exact art :-))
Symbolic Learning e.g. decision trees (ID3)	<ul style="list-style-type: none"> ▶ roots in AI ▶ decision trees, rules ▶ no assumptions made ▶ intuitive ▶ meaningful 	<ul style="list-style-type: none"> ▶ very intuitive, easy to understand ▶ fast 	<ul style="list-style-type: none"> ▶ not as accurate
Connectionist = Neural Networks FF/BP, SOM	<ul style="list-style-type: none"> ▶ black box approach ▶ flexible ▶ powerful ▶ combination of specialty graphs (=the data structure) + statistics 	<ul style="list-style-type: none"> ▶ general purpose, can be applied to many problems ▶ can be super accurate (can also be a problem) 	<ul style="list-style-type: none"> ▶ black box approach ▶ no intuitive results ▶ slow to train
Evolutionary = Genetic Algorithms	<ul style="list-style-type: none"> ▶ based on 'survival of the fittest' ▶ stochastic process 	<ul style="list-style-type: none"> ▶ powerful for complex problems ▶ easy to implement ▶ relatively fast 	<ul style="list-style-type: none"> ▶ problem representation (translation into GA) is hard to do
Heuristics/human based	<ul style="list-style-type: none"> ▶ learn from experience ▶ rules of thumb ▶ sometimes translated into expert systems 	<ul style="list-style-type: none"> ▶ human judgment, insight (will be compatible) 	<ul style="list-style-type: none"> ▶ variation of performance ▶ memory limitations ▶ information overload problems

Classification of Algorithms - Based on Goal

- ▶ Descriptive Models

- ▶ Link Analysis
 - ▶ Apriori algorithm
- ▶ Clustering models
 - ▶ Hierarchical: Prim's and Kruskal's
 - ▶ Partitioning: K-Means
 - ▶ Density-based: DBSCAN
 - ▶ Grid-based: Self-Organizing Map (SOM)

- ▶ Predictive Models

- ▶ Classification (predict class)
 - ▶ Naïve Bayes
 - ▶ Decision Trees: ID3, C4.5
- ▶ Ensemble Methods:
Random Forest
- ▶ Neural Networks: FF/BP
- ▶ Support Vector Machines
- ▶ Regression (predict number)

- ▶ Optimization Models

- ▶ Evolutionary programming
(parallel search)
- ▶ Genetic Algorithm (GA)

Neural Networks Relation to Biology

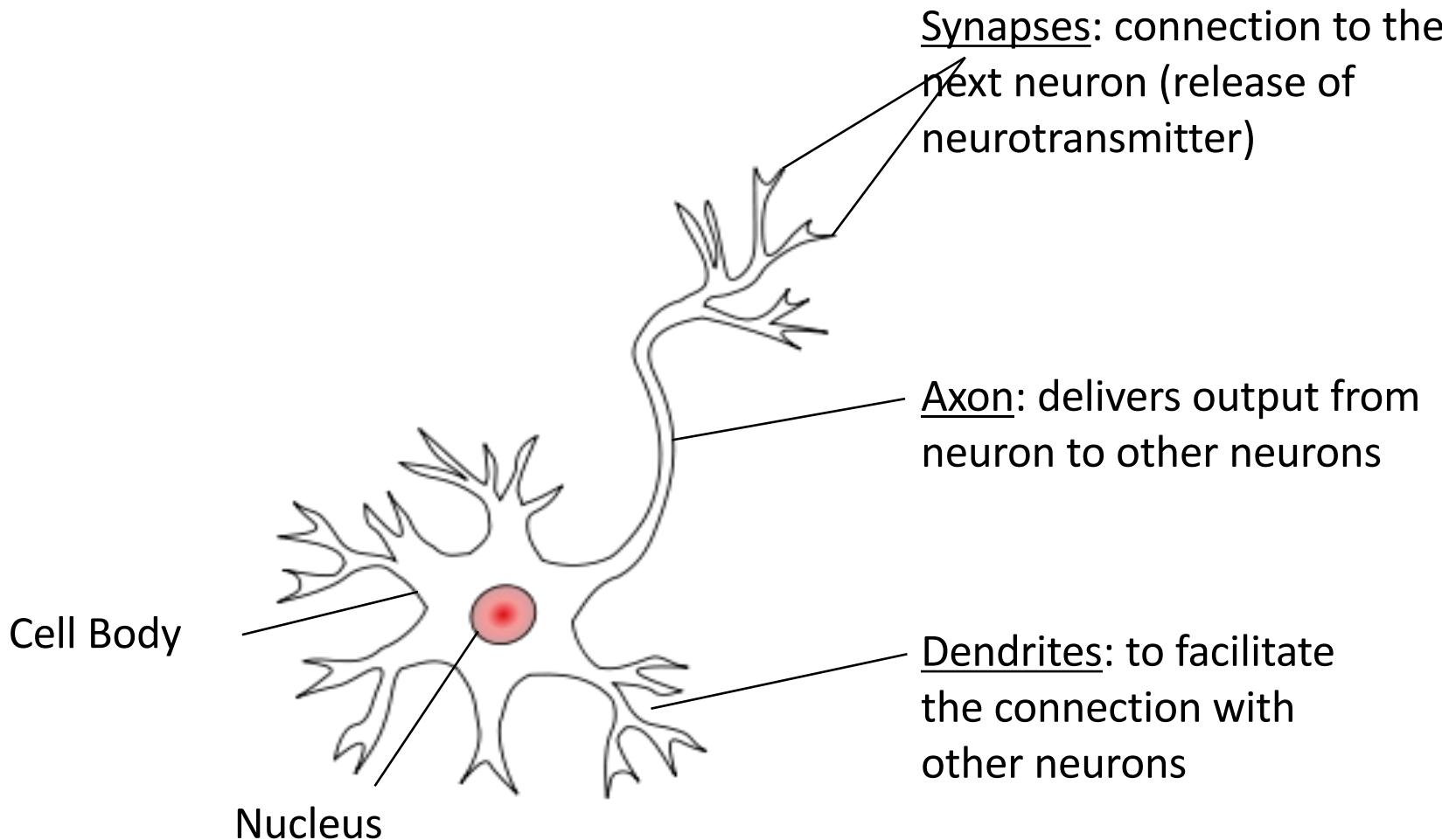
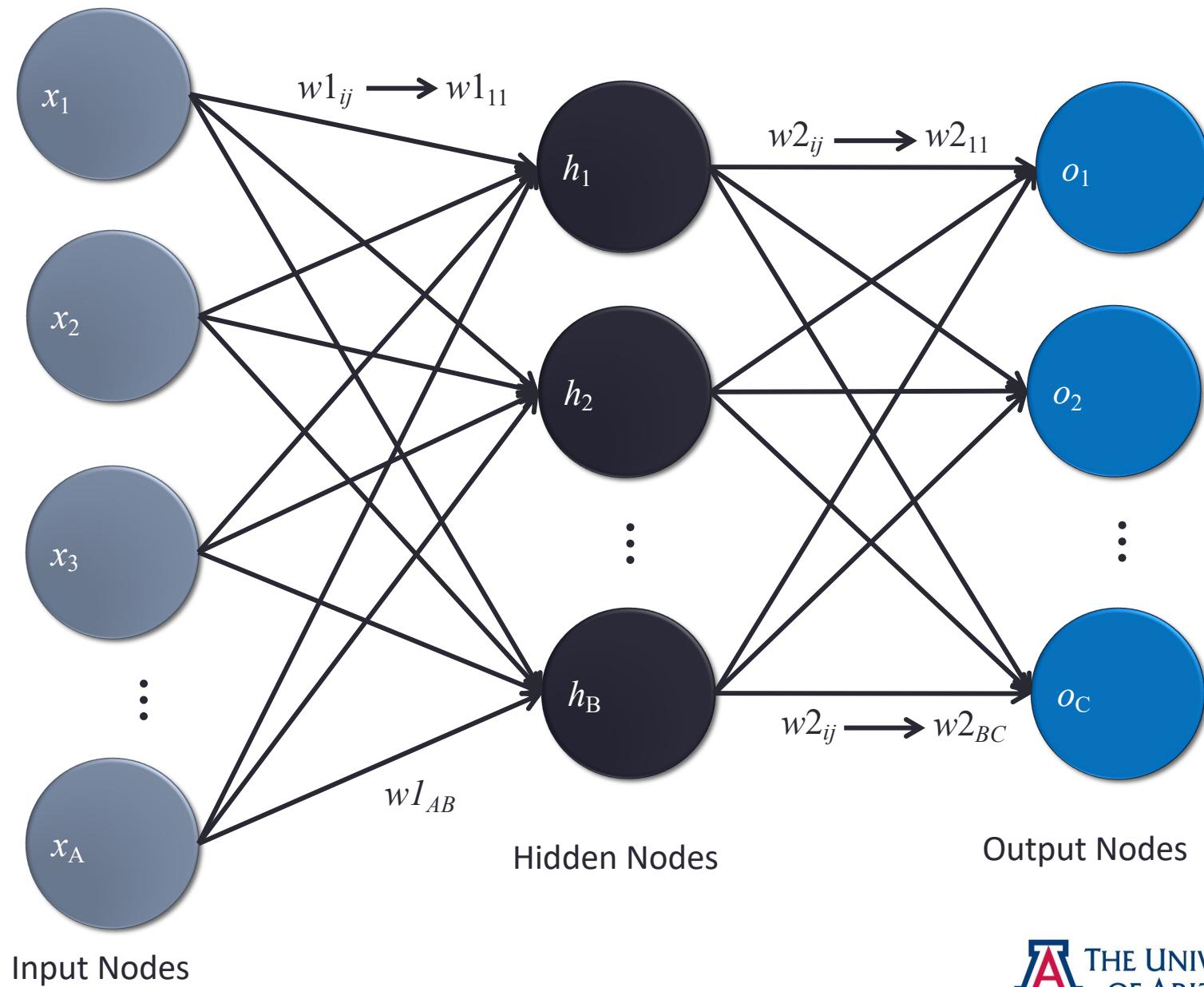


Figure: Structure of a typical neuron

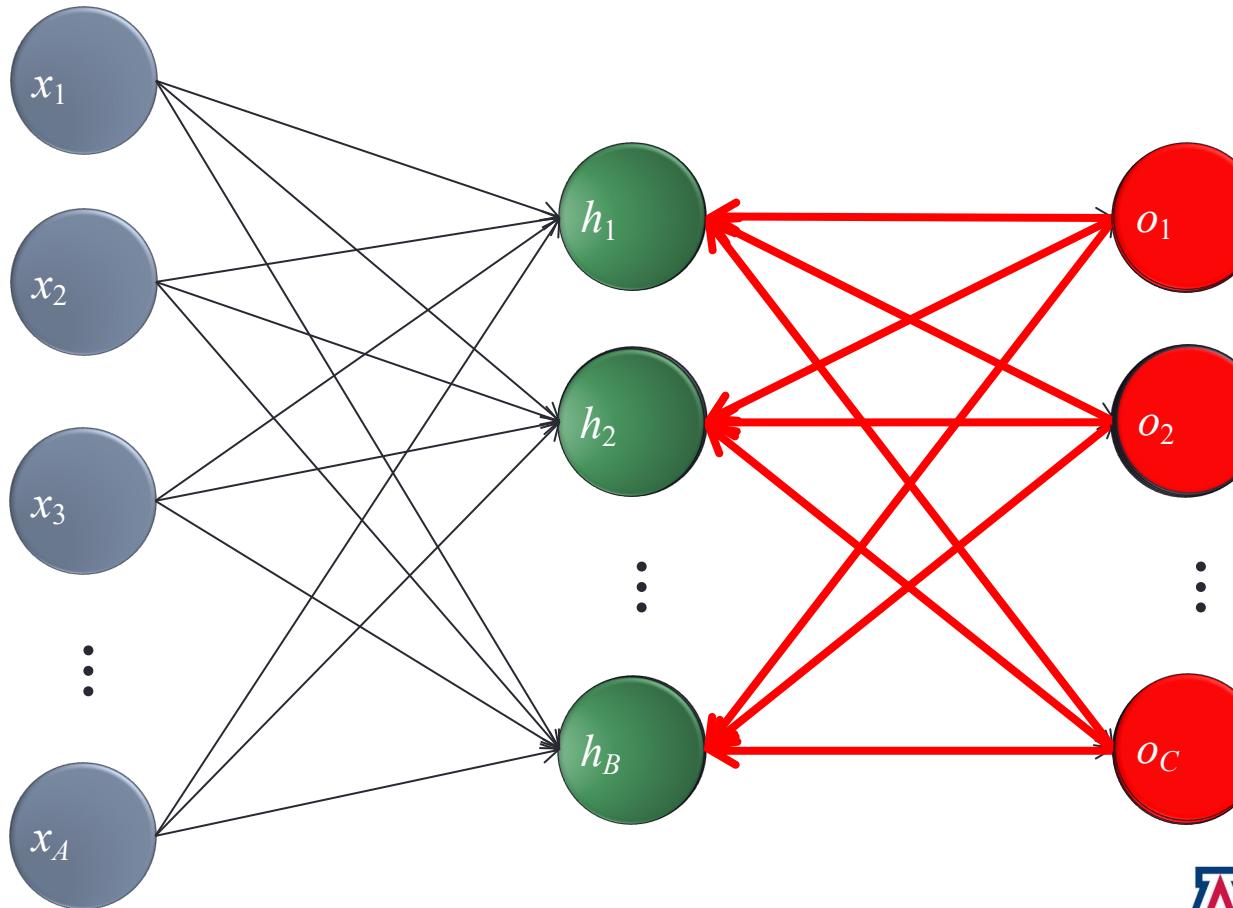
Neural Networks



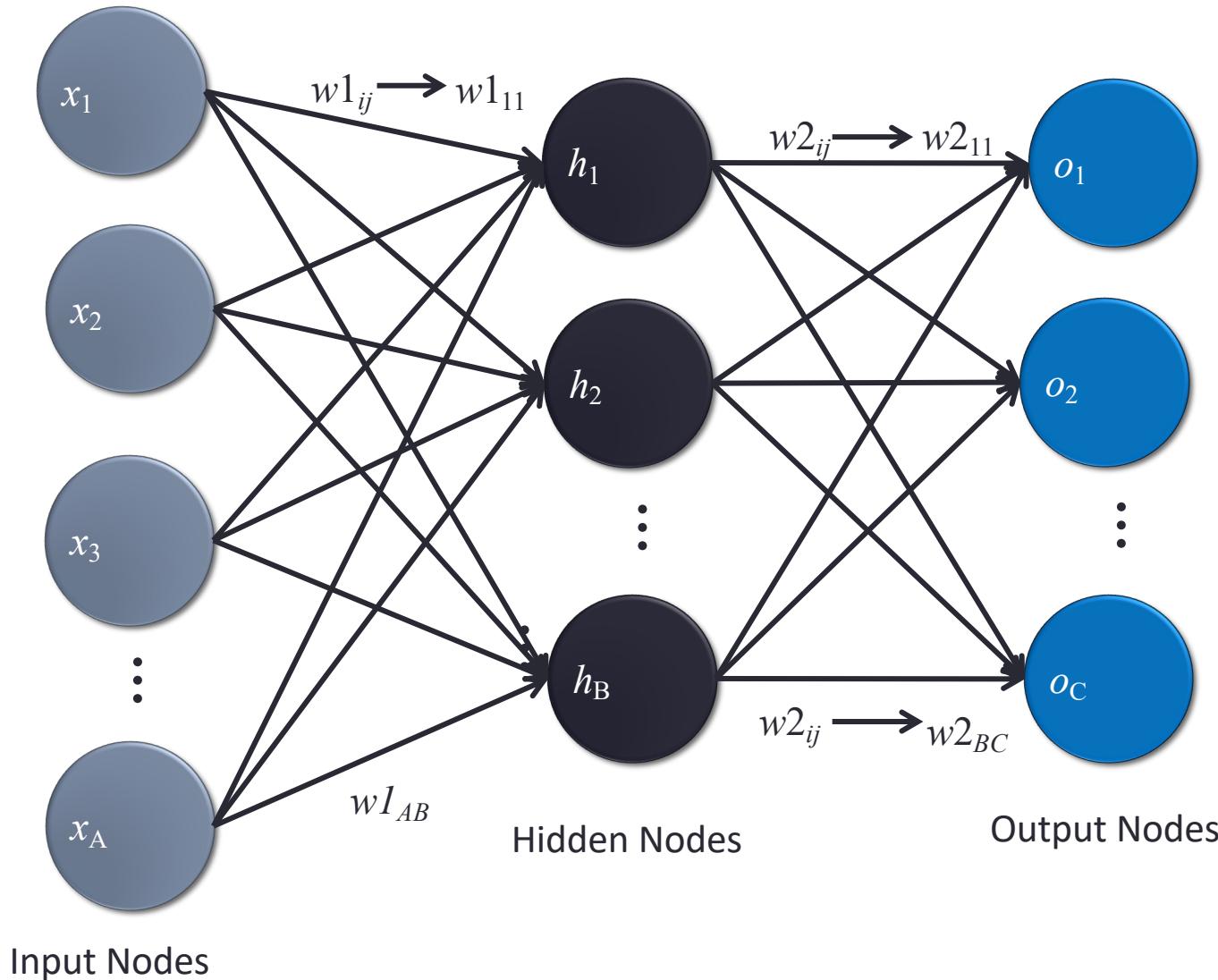
Backpropagation – the math

$$\text{Output Node Error} = \delta_2_j = o_j(1 - o_j)(y_j - o_j)$$

$$\text{Hidden Node Error} = \delta_1_j = h_j(1 - h_j) \sum_{i=1}^C \delta_2_i \times w_{2ji}$$



Artificial Neural Networks



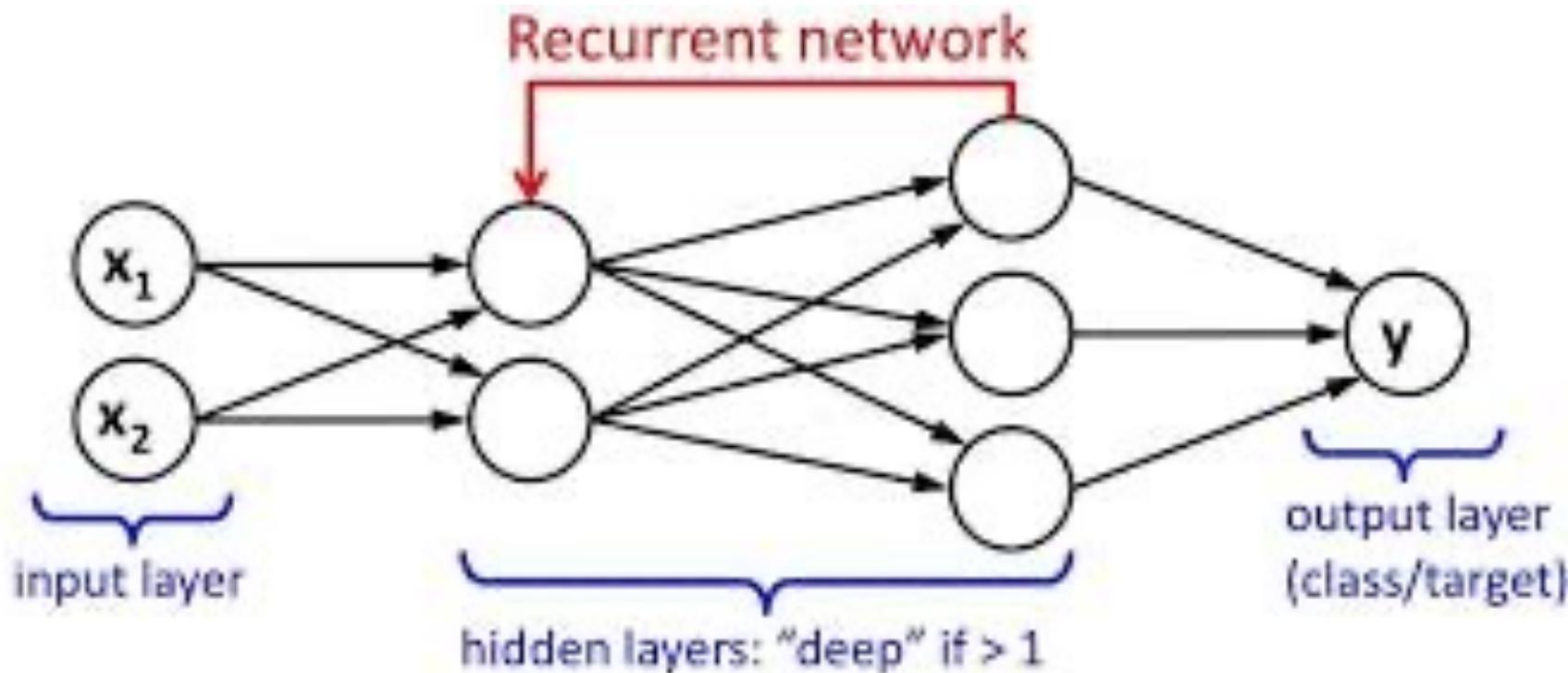
A single perceptron (or neuron) can be imagined as a Logistic Regression.

An Artificial Neural Network is a group of multiple perceptrons at each layer.

ANN can be used to solve problems related to:

- Tabular data
- Image data
- Text data

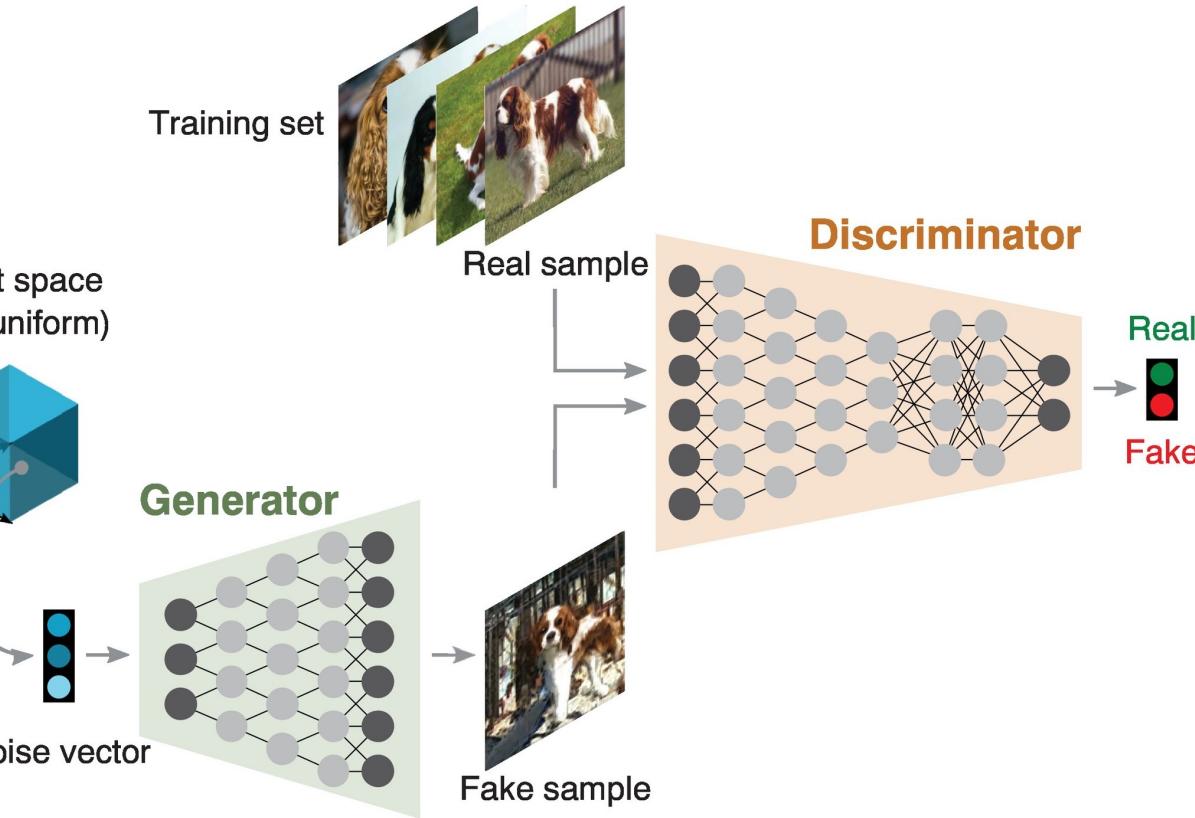
Recurrent Neural Networks



A looping constraint on the hidden layer of ANN turns it into RNN.
Parameter sharing results in fewer parameters to train and so reduces cost
RNN can be used for problems related to: Time series, Text or Audio data

Generative Adversarial Networks

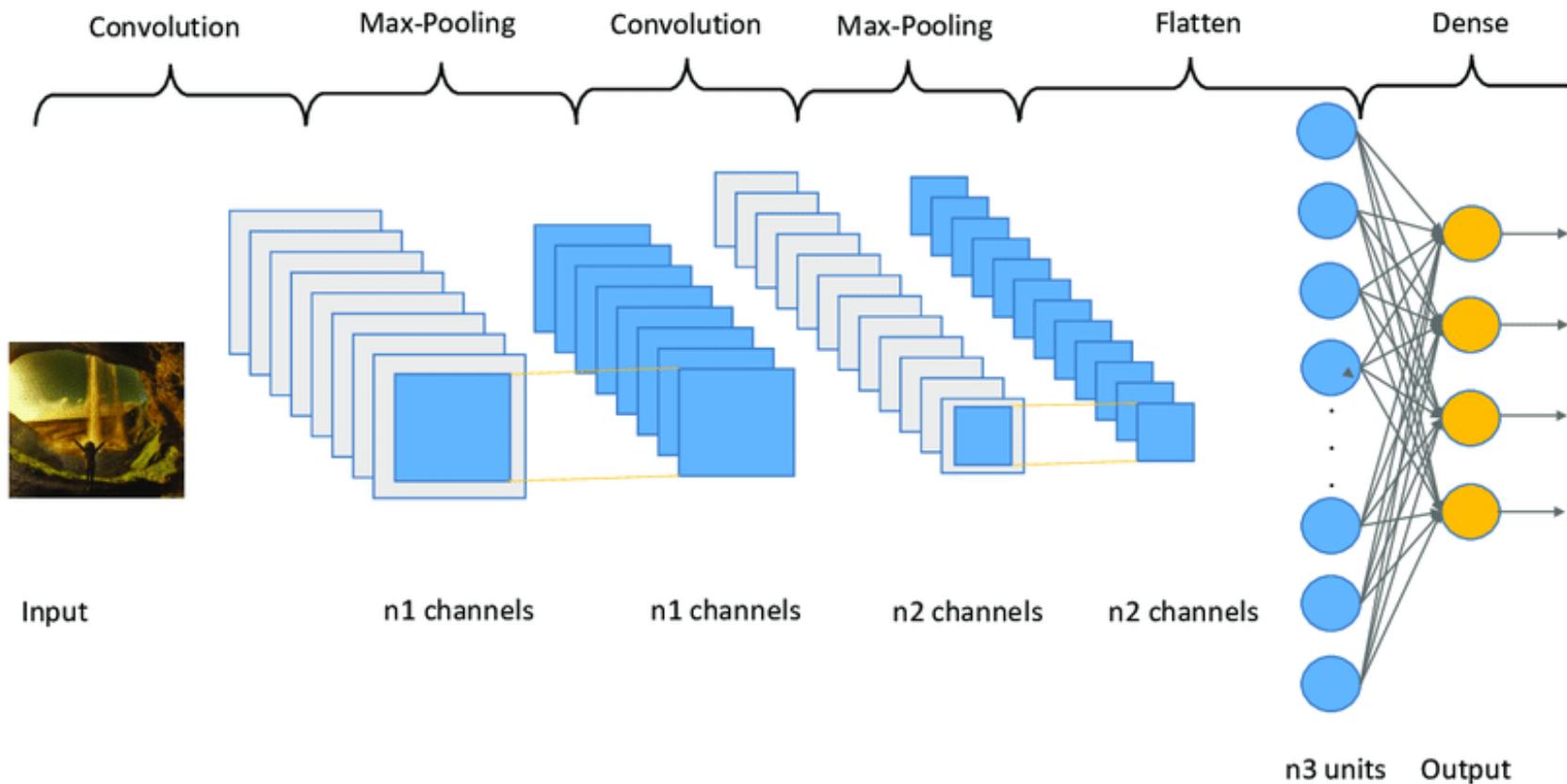
(A)



A double network with a generator that tries to fool the discriminator.

Used to generate images but also used in large particle physics on the LHC to create results close to the simulations without the very high compute cost.

Convolutional Neural Networks



A filter extracts part of the visual data and creates a 3D array or convolution which are passed to the pooling layer which reduces unnecessary features.

For image recognition, first layer detects gradients, second lines, third shapes etc

CNN can be used for problems related to: Image processing, Computer vision, Speech recognition

Using RStudio on HPC

The screenshot shows the OnDemand HPC interface. At the top, there is a navigation bar with links for Apps, Files, Jobs, Clusters, Interactive Apps (which is currently selected), and My Interactive Sessions. A note about "windfall" jobs is displayed. Below the navigation bar, there is a large "OPEN OnDemand" button and a brief description of what OnDemand provides. A "Pinned Apps" section lists SIMULIA Abaqus and ANSYS Workbench GUI as system-installed apps. The main area features a dropdown menu for "Interactive Apps" which is currently open, showing categories for Desktops (Interactive Desktop), GUIs (ABAQUS GUI, ANSYS Workbench GUI, MATLAB GUI, Mathematica GUI, VSCode GUI), and Servers (Jupyter Notebook, RStudio Server). The RStudio Server option is highlighted with a mouse cursor.

Please NOTE: "windfall" jobs will be re

Interactive Apps ▾ My Interactive Sessions

Desktops

Interactive Desktop

GUIs

- ABAQUS GUI
- ANSYS Workbench GUI
- MATLAB GUI
- Mathematica GUI
- VSCode GUI

Servers

- Jupyter Notebook
- RStudio Server

OnDemand provides an integrated, efficient way to access and manage your HPC resources.

Pinned Apps A featured subset of pinned applications.

SIMULIA
Abaqus

ABAQUS GUI

System Installed App

ANSYS
Workbench

GUI

System Installed App

RStudio Server

MATLAB GUI

System Installed App

Using RStudio on HPC

Interactive Apps

- Desktops
- Interactive Desktop
- GUIs
- Abaqus GUI
- Ansys Workbench GUI
- Mathematica GUI
- Matlab GUI
- Stata GUI
- VSCode GUI
- Servers
- Jupyter Notebook
- RStudio Server

RStudio Server

This app will launch [RStudio Server](#), an IDE for [R](#), on a [UAz cluster](#).

Cluster

ElGato Cluster

RStudio Server version

R 4.2.2

This defines the version of the application you want to use

Run Time

3

Enter maximum number of wall clock hours the job is allowed to run.

Core count on a single node

1

Enter the number of cores on a single node that the job is allowed to use.

Memory per core

6

Enter the number of Gigabytes of RAM needed per core.

GPUs required

0

Enter number of gpus needed per node, if any.

PI Group

chrisreidy

Enter an HPC PI group to be charged for time used.

Using RStudio on HPC

RStudio Server (638047)

1 node | 1 core | Running

Host: >[_i16n0.ocelote.hpc.arizona.edu](https://i16n0.ocelote.hpc.arizona.edu)

Created at: 2022-03-27 09:03:00 MST

Time Remaining: 1 hour and 59 minutes

Session ID: 6183a471-561b-498d-bd0b-06531ca63348

[④ Connect to RStudio Server](#)

[Delete](#)

Using RStudio on HPC

The screenshot shows the RStudio interface running on a High-Performance Computing (HPC) system. The top navigation bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The top right shows the user 'chrisreidy' and a project dropdown set to '(None)'. The left sidebar has tabs for Console, Terminal (disabled), and Jobs (disabled). The main area consists of three panes:

- Environment:** Shows the Global Environment with a table of variables:

mydata	15 obs. of 2 variables
a	chr [1:5] "aa" "bb" "cc" "dd" "ee"
A	num [1:8] 32311 32624 32908 33219 33499 ...
b	num [1:4] 1 2 4 8
B	num [1:7] 313 284 311 280 322 324 302
C	num [1:10] 27 26 30 34 32 25 31 25 27 32
commute	num [1:10] 27 26 30 34 32 25 31 25 27 32
- Files:** Shows the file structure under 'Home':

Name	Size	Modified
.RData	941 B	Jul 20, 2019, 8:03 AM
.Rhistory	2.4 KB	Aug 6, 2021, 8:53 AM
__pycache__	3.3 KB	Mar 21, 2011, 12:05 AM
activate.ini	2.1 MB	Feb 19, 2022, 4:50 PM
alpine_latest.sif	3.7 KB	Sep 21, 2021, 9:52 AM
autamus-notes	3.5 KB	Oct 23, 2019, 4:07 PM
conda-bash.sh		
- Console:** Displays the R startup message and workspace loading information:

```
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Exercise: Data Pre-processing

In this exercise, download package VIM and mice. The dataset 'sleep' is included in the package VIM. It comes from a study about the sleeping pattern of 62 mammals. It wants to identify the relationships between sleep, and some physical characteristics of mammals, such as brain and body weight, life span, gestation time, time sleeping, and predation and danger indices.

```
# install package "VIM"
install.packages("VIM") # Hint: use multiple cores. It goes much faster
# To use the package in an R session, we need to load it in an R session via
library()
library(VIM)
# Load dataset "sleep", which comes within the package "VIM"
data(sleep, package ="VIM")
# call function head() to get a feeling about data, or call sleep to see all values
head(sleep)
# download package "mice" and load it into R
install.packages("mice")
library(mice)
```

Exercise: Data Pre-processing

In R, we use “NA” stands for missing value; NaN(Not a Number) stands for an impossible value; And symbol “Inf” and “-Inf” stands for infinity and negative infinity respectively. In order to tell whether a value belongs to any case above, we use function is.na(), is.nan(), and is.infinite(). The return value of each function is Boolean

First, we need to know how many rows in “sleep”

```
nrow(sleep)
```

```
## [1] 62
```

We use complete.cases() or na.omit() to see tuples without missing value.

```
sleep[complete.cases(sleep),]
```

or try

```
na.omit(sleep)
```

Count the number of rows without missing value

```
nrow(sleep[complete.cases(sleep),])
```

```
## [1] 42
```

To reverse the condition logic (rows containing one or more missing value), we use the exclamation mark highlighted in Red

```
sleep[!complete.cases(sleep),]
```

```
nrow(sleep[!complete.cases(sleep),])
```

```
## [1] 20
```

Exercise: Data Pre-processing

We tell R a Boolean value by inputting a TRUE or a FALSE. However, R can treat them as integer 1 or 0 respectively, which means information about missing value can be captured by using function sum() and mean().

```
# Check how many observations contain missing value in column "Dream"  
sum(is.na(sleep$Dream))  
## [1] 12  
# About 19% of obs (observations) in column Dream contain missing value  
mean(is.na(sleep$Dream))  
## [1] 0.1935484  
# 32% obs in data frame sleep containing one or more missing value  
mean(!complete.cases(sleep))  
## [1] 0.3225806
```

Exercise: Data Pre-processing

Checking missing values is important, but little information about distribution and pattern can be observed via functions we called above. It will be a convenient alternative if we can see how the missing values distribute in datasets. To do so, we can use the function `md.pattern()` to generate a matrix that shows the pattern of missing value.

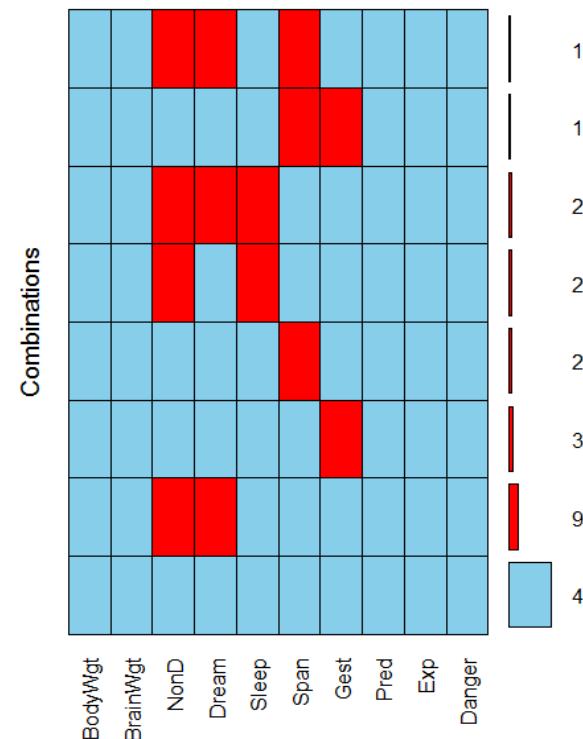
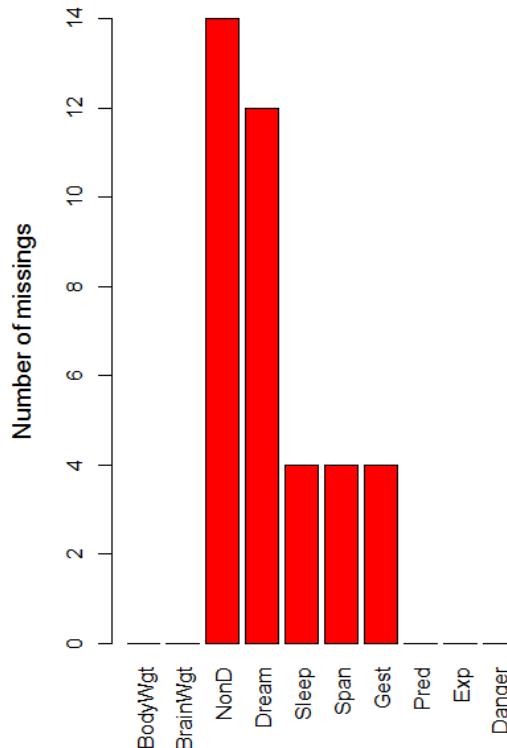
```
# call function md.pattern(). Make sure you loaded package mice into R first  
md.pattern(sleep)
```

BodyWgt	BrainWgt	Pred	Exp	Danger	Sleep	Span	Gest	Dream	NonD
42	1	1	1	1	1	1	1	1	1 0
2	1	1	1	1	1	1	0	1	1 1

Exercise: Visualization – aggr

Instead of seeing the sleep data in table via function `md.pattern()`, a visualization usually makes more sense to users. The package `VIM` provides several functions that visualize the pattern of missing values. Here we will apply `aggr()`, to our dataset.

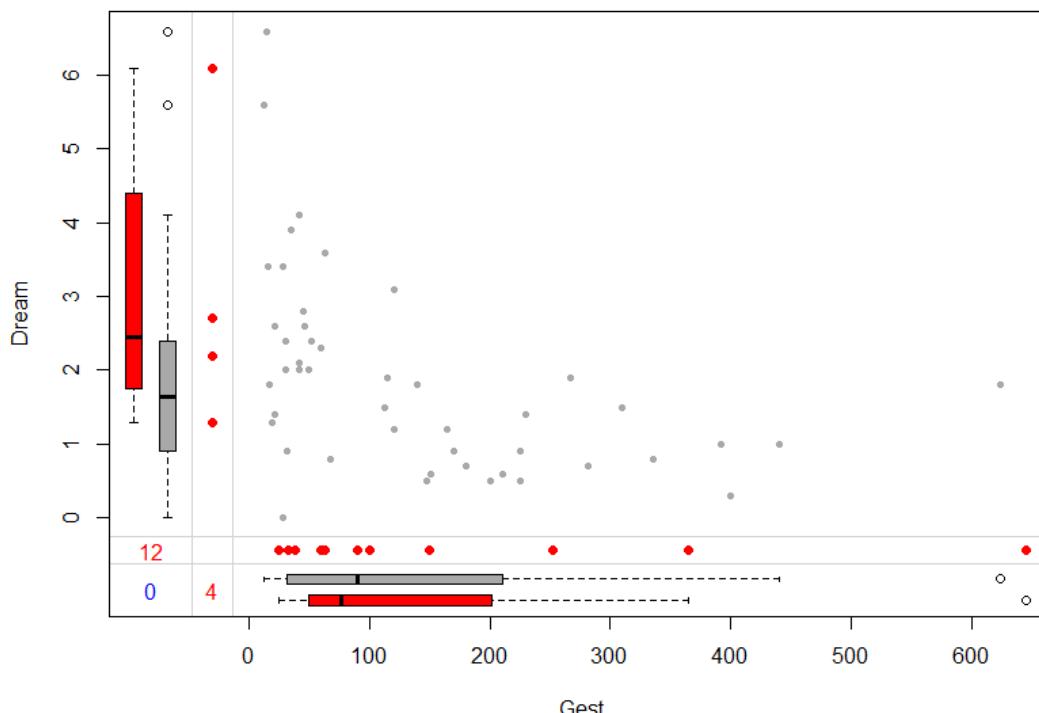
```
# call function aggr(), prop = FALSE convert percentage value into counts  
aggr(sleep, prop = FALSE, numbers = TRUE)
```



Exercise: Visualization - marginplot

Function `marginplot()` returns a scatter chart, where we can observe the relationship between two variables.

```
# call function marginplot (), pch indicates notation of obs, col tells R how you  
would like to see results in different color  
marginplot(sleep[c("Gest", "Dream")], pch=c(20),  
          col = c("darkgray","red","blue"))
```



Exercise: Visualization - boxplot

Load dataset mtcars into R and show relationship between mpg and # of cylinders

```
boxplot(mpg ~ cyl, # mpg is the target variable
```

```
# cyl is the explanatory variable
```

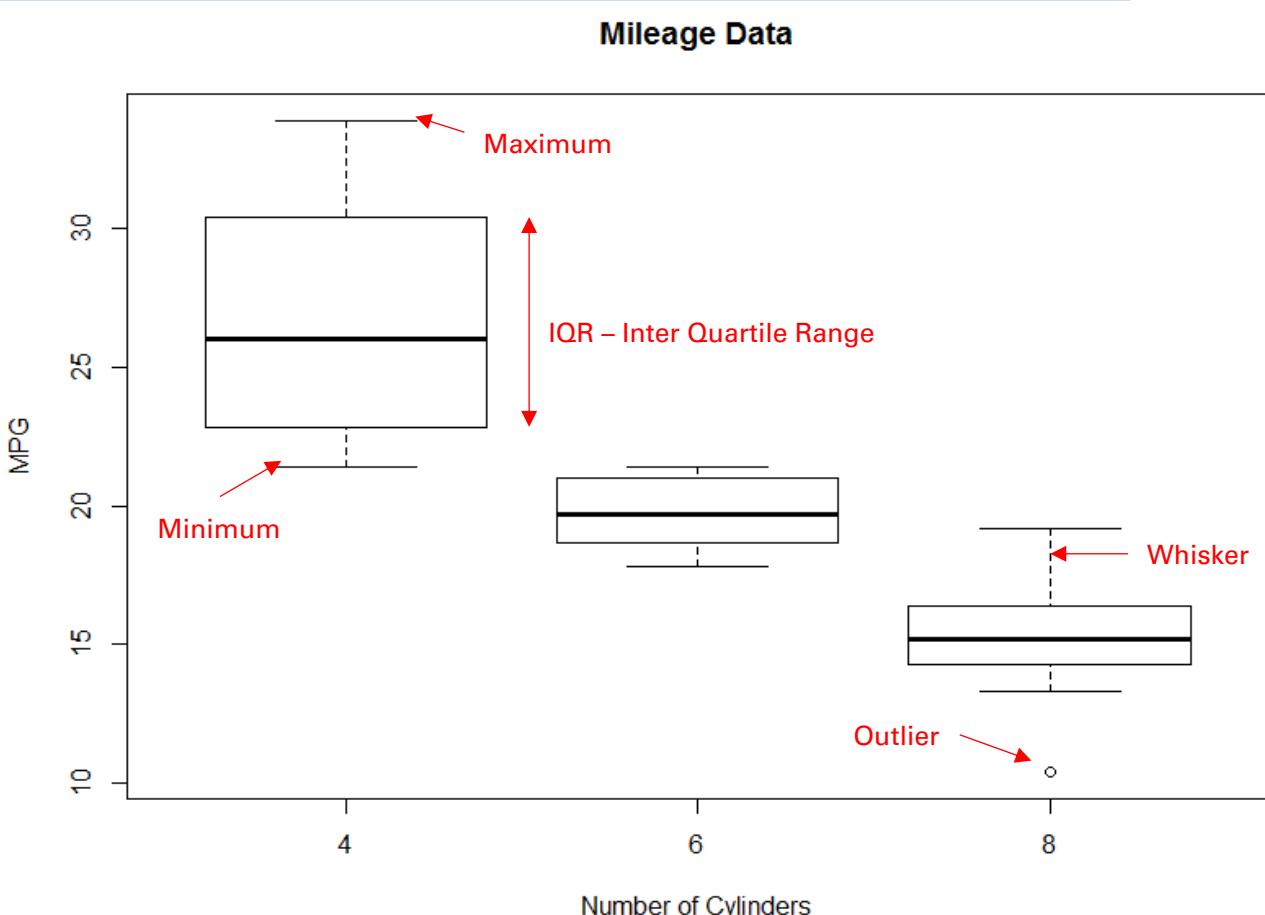
```
data = mtcars,
```

```
col = "grey",
```

```
main = "Mileage Data",
```

```
ylab = "MPG",
```

```
xlab = "Number of Cylinders" )
```



Exercise: Visualization – violin plot

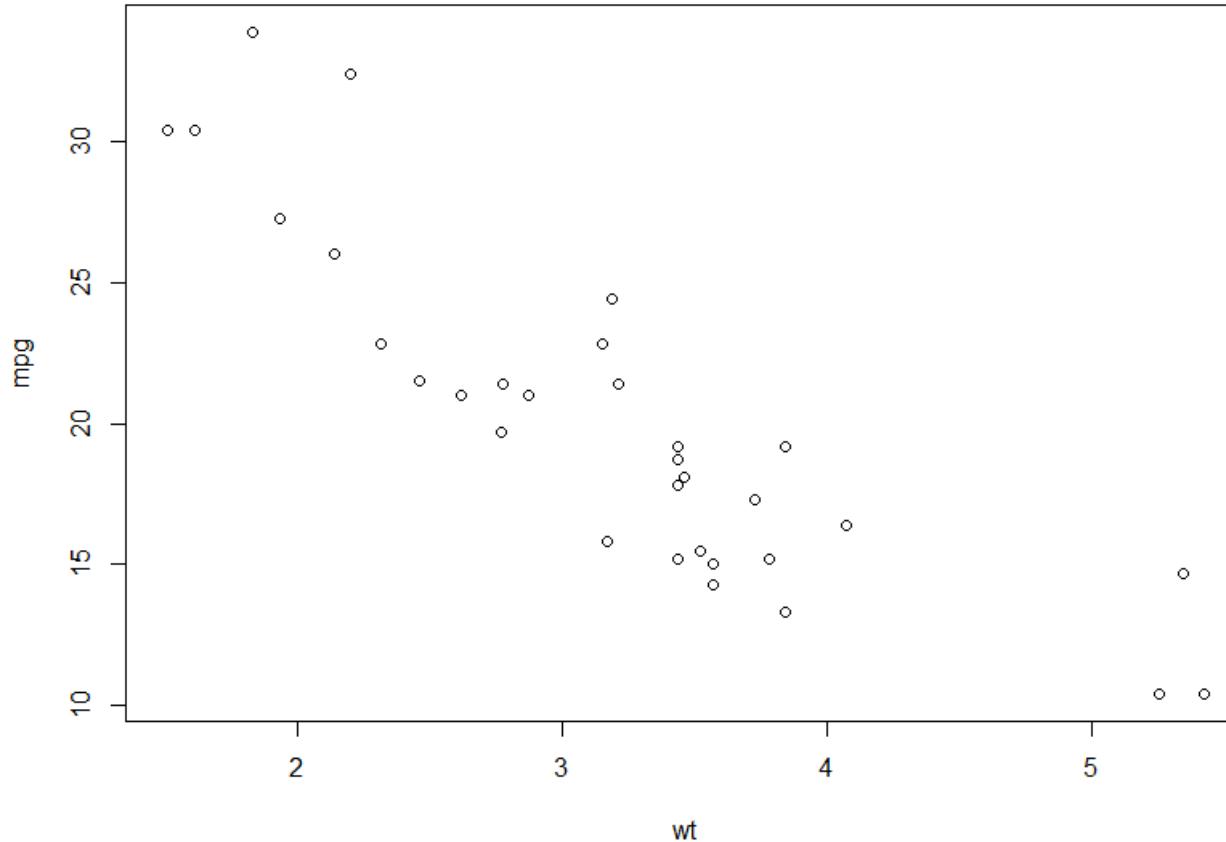
A **violin plot** provides a comprehensive chart that combines information in box plot and density distribution.

```
install.packages("vioplot")
library(vioplot)
v1 <- mtcars$mpg[mtcars$cyl == 4]
v2 <- mtcars$mpg[mtcars$cyl == 6]
v3 <- mtcars$mpg[mtcars$cyl == 8]
# draw violin plots for vectors
vioplot(v1,v2,v3,
        names=c("4 cylinders", "6 cylinders", "8 cylinders"),
        col="gold")
```

Exercise: Visualization – scatter plot

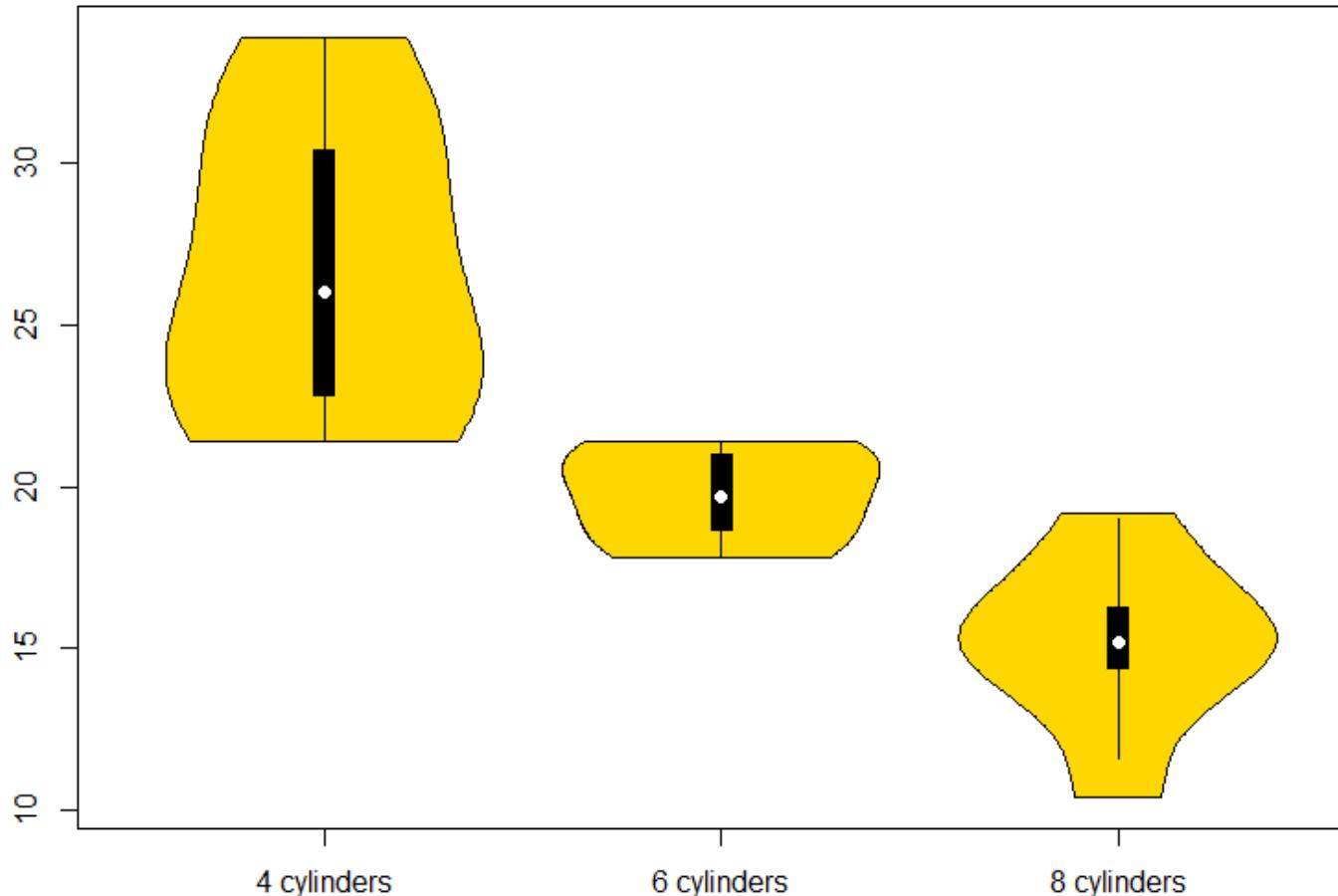
Using mtcars again we apply dot plot to check how each observation is distributed in a given sample space.

```
plot(mpg ~ wt, data = mtcars)
```



Exercise: Visualization – violin plot

The outline of a violin chart indicates the density of distribution. The black bar at center tells the IQR of distribution.



Exercise: Naive Bayes Classifier

Installing Packages

In the Mushroom dataset, there are 8123 observations belonging to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. There are two types of mushroom in terms of edibility. If `classes=e`, the mushroom is `edible`, if `classes=p`, the mushroom is `poisonous`. We want to tell which mushrooms are `edible` from those `poisonous` by looking at some of their characteristics. Source: University of California Irvine

View working directory then set working directory

```
getwd()  
setwd('/xdisk/chrisreidy/workshops')
```

- `e1071` – This is the dataset we are going to use.

```
install.packages('e1071')  
library(e1071)
```

Exercise: Naive Bayes Classifier

Preprocessing

Save Mushroom.csv under your working directory. The null value in mushroom dataset is denoted by question mark.

```
# read in csv file mushroom.csv. Note the question mark  
# represents null value  
mushroom <- read.csv("Mushroom.csv", na.strings = "?")  
summary(mushroom)
```

```
# check completion  
nrow(mushroom[ !complete.cases(mushroom) , ] )  
## [1] 2480
```

Exercise: Naive Bayes Classifier

Preprocessing

Naive Bayes is an algorithm that depends on probability. To predict a conditional probability, we have to figure out the prior probability of each predictive variables. Therefore, a dataset with null value will raise risk for our prediction

```
# we can retain observations that do not contain NA(null) value  
mushroom = mushroom[complete.cases(mushroom), ]
```

Exercise: Naive Bayes Classifier

Training and testing sets

Next, we will create train and test sets of the data. We will fit the model with the training set, and use the test set to evaluate the model. We will do a 70/30 split (70% will be training data).

```
# 70% of original data will be used for training
sample_size <- floor(0.7 * nrow(mushroom) )
# randomly select index of observations for training
training_index <- sample(nrow(mushroom), size = sample_size,
replace = FALSE)
train <- mushroom[training_index, ]
test <- mushroom[-training_index, ]
```

Exercise: Naive Bayes Classifier

Fitting and Model Performance

There is a Naive Bayes classifier in the `e1071` package, loaded into our current session already via function library(`e1071`). Fit the model to the training data.

```
# note the period coming after tilde. It means all the other  
variables in that dataset will be predictive variable  
mushroom.model <- naiveBayes(classes ~ . , data = train)  
# We can explore the detail conditional probabilities for each  
variables by calling the object mushroom.model itself.  
mushroom.model
```

Exercise: Naive Bayes Classifier

Fitting and Model Performance

After fitting, run the test data through the model to get the predicted class for each observation.

```
# The result of prediction, a vector, will be attached to test  
set labelled as "class". The return of prediction is a vector  
including predicted type of mushroom  
mushroom.predict <- predict(mushroom.model, test, type = "class")
```

Exercise: Naive Bayes Classifier

Fitting and Model Performance

Show the performance of the model.

```
# pick actual value and predicted value together in a dataframe  
# called results  
results <- data.frame(actual = test[, 'classes'], predicted =  
mushroom.predict)  
  
# We can get a popular matrix called confusion matrix  
table(results)  
  
# columns indicate the number of mushrooms in actual type;  
# likewise, rows indicate the number those in predicted type.  
  
            predicted  
actual      edible    poisonous  
          edible TN=1067        FP=  2  
          poisonous FN=   46        TP=580
```

Exercise: Neural Network

Installing Packages

For this example, we need to install package ISLR. The data we will use is a built-in dataset of ISLR called College Data Set. It has several features of a college and a categorical column indicating whether or not the School is Public or Private.

- ISLR – This is the dataset we are going to use.
- caTools – We will use the caTools package to randomly split the data into a training set and test set.
- neuralnet – This package contains the function about neural network neuralnetwork().

```
install.packages('ISLR')
install.packages('caTools')
install.packages('neuralnet')
```

Exercise: Neural Network

Loading Data

Install package ISLR, which is a dataset package and contains data we will use - College.

```
library(ISLR)
```

```
print(head(College, 2))
```

		Private	Apps	Accept	Enroll	Top10perc
Abilene Christian University	Yes	1660	1232	721	23	
Adelphi University	Yes	2186	1924	512	16	
		Top25perc	F.Undergrad	P.Undergrad	Outstate	
Abilene Christian University	52	2885	537		7440	
Adelphi University	29	2683	1227		12280	
		Room.Board	Books	Personal	PhD	Terminal
Abilene Christian University	3300	450	2200	70	78	
Adelphi University	6450	750	1500	29	30	
		S.F.Ratio	perc.alumni	Expend	Grad.Rate	

Exercise: Neural Network

Preprocessing

It is important to normalize data before training a neural network on it. The neural network may have difficulty converging before the maximum number of iterations allowed if the data is not normalized. There are a lot of different methods for normalization of data.

Normally it is better to scale the data from 0 to 1, or -1 to 1. We can specify the center and scale as additional arguments in the `scale()` function. For example:

```
# Create Vector of Column Max and Min Values
```

```
maxs <- apply(College[,2:18], 2, max)  
mins <- apply(College[,2:18], 2, min)
```

```
# Use scale() and convert the resulting matrix to a data frame
```

```
scaled.data <- as.data.frame(scale(College[,2:18], center = mins,  
                                scale = maxs - mins))
```

Exercise: Neural Network

Preprocessing - continued

```
# Check out results  
print(head(scaled.data,2))
```

	Apps	Accept	Enroll
Abilene Christian University	0.03288692646	0.04417701272	0.10791253736
Adelphi University	0.04384229271	0.07053088583	0.07503539405
	Top10perc	Top25perc	F.Undergrad
Abilene Christian University	0.2315789474	0.4725274725	0.08716353479
Adelphi University	0.1578947368	0.2197802198	0.08075165058
	P.Undergrad	Outstate	Room.Board
Abilene Christian University	0.02454774445	0.2634297521	0.2395964691
Adelphi University	0.05614838562	0.5134297521	0.7361286255
	Books	Personal	PhD
Abilene Christian University	0.1577540107	0.2977099237	0.6526315789
Adelphi University	0.2914438503	0.1908396947	0.2210526316
	Terminal	S.F.Ratio	perc.alumni
Abilene Christian University	0.71052631579	0.4182305630	0.1875
Adelphi University	0.07894736842	0.2600536193	0.2500

Exercise: Neural Network

Train and Test Split

Let us now split our data into a training set and a test set. We will run our neural network on the training set and then see how well it performed on the test set.

We will use the `caTools` package to randomly split the data into a training set and test set.

```
# Convert Private column from Yes/No to 1/0
Private = as.numeric(College$Private)-1
data = cbind(Private, scaled.data)

library(caTools)
set.seed(101) # sets random numbers

# Create Split (any column is fine)
split = sample.split(data$Private, SplitRatio = 0.70)

# Split based off split Boolean Vector
train = subset(data, split == TRUE)
test = subset(data, split == FALSE)
```

Exercise: Neural Network

Neural Network Function

Before we actually call the function, we need to create a formula to insert into the machine learning model. The neuralnetwork() function won't accept the typical decimal R format for a formula involving all features (e.g. $y \sim .$). However, we can use a simple script to create the expanded formula and save us some typing:

```
feats <- names(scaled.data)
# Concatenate strings
f <- paste(feats,collapse=' + ')
f <- paste('Private ~', f)
# Convert to formula
f <- as.formula(f)
f
## Private ~ Apps + Accept + Enroll + Top10perc + Top25perc + F.Undergrad +
##   P.Undergrad + Outstate + Room.Board + Books + Personal +
##   PhD + Terminal + S.F.Ratio + perc.alumni + Expend + Grad.Rate

library(neuralnet)
nn <- neuralnet(f, train, hidden = c(10,10,10), linear.output = FALSE)
```

Exercise: Neural Network

Predictions and Evaluations

Now let's see how well we performed! We use the compute() function with the test data (just the features) to create predicted values. This returns a list from which we can call net.result

```
# Compute Predictions off Test Set  
predicted.nn.values <- compute(nn, test[2:18])  
  
# Check out net.result  
print(head(predicted.nn.values$net.result))  
[,1]  
Adrian College      1  
Alfred University    1  
Allegheny College    1  
Allentown Coll. of St. Francis de Sales 1  
Alma College        1  
Amherst College     1
```

Exercise: Neural Network

Predictions and Evaluations

Notice we still have results between 0 and 1 that are more like probabilities of belonging to each class. We'll use `sapply()` to round these off to either 0 or 1 class so we can evaluate them against the test labels.

```
predicted.nn.values$net.result <- sapply(predicted.nn.values$net.result,  
                                         round, digits = 0)
```

Now let's create a simple confusion matrix:

```
table(test$Private, predicted.nn.values$net.result)
```

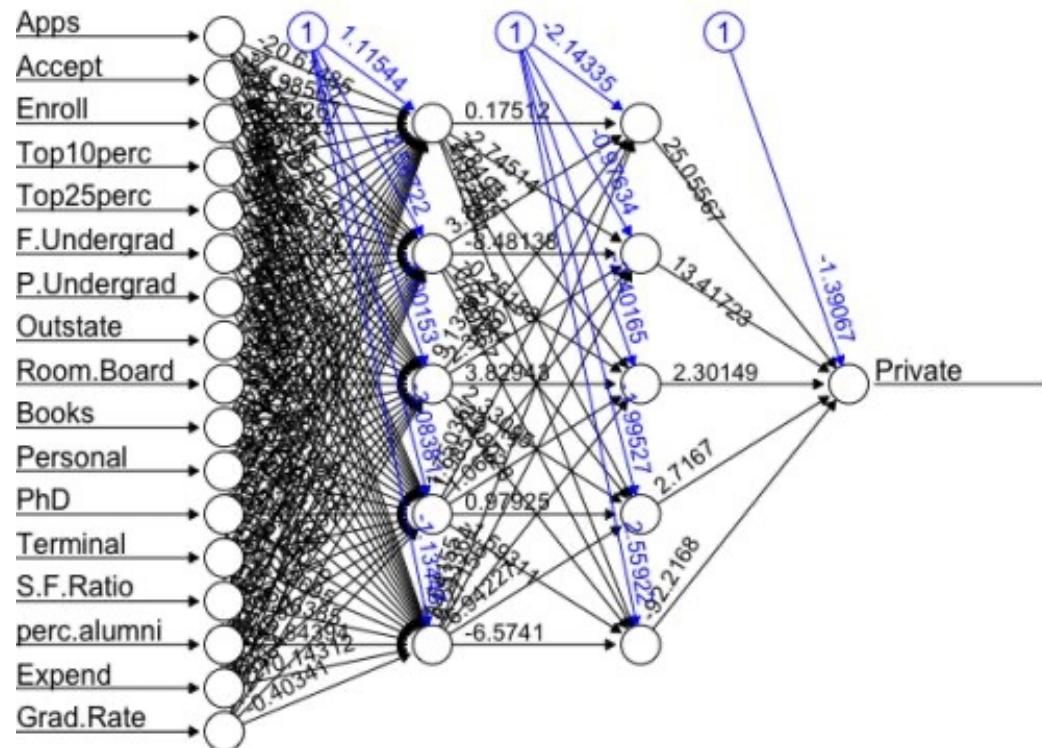
	Predicted No	Predicted Yes
Actual No	TN = 55	FP = 9
Actual Yes	FN = 6	TP = 163

Exercise: Neural Network

Visualize

We can visualize the Neural Network by using the `plot(nn)` command. The black lines represent the weighted vectors between the neurons. The blue line represents the bias added. Unfortunately, even though the model is clearly a very powerful predictor, it is not easy to directly interpret the weights. This means that we usually have to treat Neural Network models more like black boxes.

`plot(nn)`



Exercise: Accuracy Terminology

	Predicted No	Predicted Yes
Actual No	TN = 55	FP = 9
Actual Yes	FN = 6	TP = 163

Starting with the confusion matrix:

- True Positive is bottom right cell divided by sum of bottom row
- TP is also called Sensitivity or Recall or “did I correctly predict positive?”
- True Negative is top left divided by sum of top row; also called Specificity
- $FP <- 1 - TN$
- $FN <- 1 - TP$: Percent of observations in negative class
- Precision is True Positive / Total Positive
- $F <- 2 * \text{precision} * \text{recall} / (\text{precision} + \text{recall})$ - aka balanced mean
- Receiver Operating Characteristic Curve (ROC) – measures Sensitivity (TP) to Specificity (TN). Visually, closer to right angle is better than diagonal.