Data Science Certification Module 4 - Machine Learning

Veerasak Kritsanapraphan - IMC Institution

Introduction

Data Science Certification

Definitions of Machine Learning

Machine learning is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

A computer program that can learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, so that its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

-Mitchell

Why Machine Learning

- Big problems
  - ~100 million songs
  - ~130 million books
  - ~644 million websites

- Open ended problems
  - 8.6k tweets per second
  - 2.4 billion active Facebook users
  - ~66k new web pages per day

Time changing problems

Intrinsically hard problems

Human Level Perception

First speech recognition 1952

Complex Open-ended games

First chess program 1957

Deep Blue 1997 beat Kasparov

Successes of Machine Learning

Web Search

Finance

Marketing & E-commerce

Abuse / Security

Robotics

Digital Assistants

Games

Many Others

Types of Machine Learning Algorithms

Thousands of machine learning algorithms, hundreds new every year

Supervised (inductive) learning
Training data includes desired outputs

Unsupervised learning
Training data does not include desired outputs

Semi-supervised learning
Training data includes a few desired outputs

Reinforcement learning
Rewards from sequence of actions

Machine Learning

Tens of thousands of machine learning algorithms

Hundreds new every year

Decades of ML research oversimplified:

All of Machine Learning:

Learn a mapping from input to output f: X → Y

X: emails, Y: {spam, notspam}

Machine Learning

Input: x (images, text, emails…)

Output: y (spam or non-spam…)

(Unknown) Target Function

$f: X \rightarrow Y$ (the “true” mapping / reality)

Data

$(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)$

Machine Learning

- Every machine learning algorithm has three components:
  - Representation / Model Class
  - Evaluation / Objective Function
  - Optimization

Making Working Systems

Traditional Software

- Programming Languages
- Algorithms
- Data Structures
- Networking
- Etc.

- Statistics
- Data Science
- Machine Learning
- Computer Vision
- Natural Language Processing

Software Engineering

Machine Learning Engineering

Summary

Reasons for ML

Big problems

Open-ended problems

Time-changing problems

Hard problems

Types of Learning

Supervised, unsupervised, semi-supervised, reinforcement learning

- Supervised Learning
  - Training data $< x, \ f(x) >$
  - Goal to learn $f(x)$ that matches training data

- Components of ML algorithm
  - Model Structure
  - Loss Function
  - Optimization

Exciting time for ML – many challenges & many ways to get involved

Asking Question in Machine Learning

Fundamental Data Science for Data Scientist

What questions can data science answer?

1. Is this **A** or **B**?
2. Is this **weird**?
3. How **much** or how **many**?
4. How is this **organized**?
5. What should I **do next**?

Is this A or B?

Classification algorithm

Is this weird?

Anomaly detection algorithm

How **much** and How **many**?

Regression algorithm

How is this organized?

Clustering algorithm

What should I do next?

Reinforcement Learning algorithm

- Labeled data
- Direct feedback
- Predict outcome/future

- No labels
- No feedback
- “Find hidden structure”

- Decision process
- Reward system
- Learn series of actions

Unsupervised Learning

Supervised Learning

Clustering:
[DBSCAN on a toy dataset]

Regression:
[Soccer Fantasy Score prediction]

Classification:
[SVM on 2 classes of the Wine dataset]

Data Science Life Cycle

Fundamental Data Science for Data Scientist

Data Science Process

Data and Getting Data

Data Science Certification

Recap import data in R

```
# first row contains variable names, comma is separator
# assign the variable id to row names
# note the / instead of \ on mswindows systems

mydata <- read.table("c:/mydata.csv", header=TRUE,
  sep=",", row.names="id")
```

Data Frame

Function

Keep first line as a header

Field Separation

Filename to import

Getting Data

Iris Data Set from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris)

Attribute Information:

1. Sepal Length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal length in cm
5. Classes:
   - Iris Setosa
   - Iris Versicolour
   - Iris Virginica

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Nomenclature

IRIS

https://archive.ics.uci.edu/ml/datasets/Iris

Instances (samples, observations)

	sepal_length	sepal_width	petal_length	petal_width	class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
…	…	…	…	…	…
50	6.4	3.2	4.5	1.5	veriscolor
…	…	…	…	…	…
150	5.9	3.0	5.1	1.8	virginica

Features (attributes, dimensions, variables)

Classes (targets)

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$\mathbf{y} = [y_1, y_2, y_3, \cdots y_N]$$

Iris setosa

Iris versicolor

Iris virginica

Getting Data

```
> iris <- read.csv("iris.data.csv", header=TRUE)
> library(datasets)
> iris
> colnames(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length",
  "Petal.Width", "Species")
```

```
> 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
> nrow(iris)
[1] 150
> table(iris$Species)

    setosa versicolor  virginica
        50         50         50
>
```

Exploratory Data Analysis
Raw Data is Collected
Data is Processed
Clean Data
Real World
Machine Learning Algorithms Statistical Models
Build Data Product
Communicate
Visualizations
Report Findings
Make Decisions

Exploratory Data Analysis (EDA)

- The goal during EDA is to develop an understanding of your data.
- The easiest way to do this is to use questions as tools to guide your investigation.
- When you ask a question, the question focuses your attention on a specific part of your dataset and helps you decide which graphs, models, or transformations to make.
- Two types of questions will always be useful for making discoveries within your data.
    1. What type of variation occurs within my variables?
    2. What type of covariation occurs between my variables?

Variation

- The tendency of the values of a variable to change from measurement to measurement. You can see variation easily in real life.
- If you measure any continuous variable twice, you will get two different results. This is true even if you measure quantities that are constant, like the speed of light. Categorical variables can also vary if you measure across different subjects or different times.
- Every variable has its own pattern of variation, which can reveal interesting information. The best way to understand that pattern is to visualise the distribution of the variable's values.

Visualizing distributions

How you visualize the distribution of a variable will depend on whether the variable is categorical or continuous.

- A variable is **categorical** if it can only take one of a small set of values. In R, **categorical variables** are usually saved as factors or character vectors. To examine the distribution of a categorical variable, **use a bar chart.**
- A variable is **continuous** if it can take any of an infinite set of ordered values. Numbers and date-times are two examples of continuous variables. To examine the distribution of a **continuous** variable, **use a histogram**:

```
> categories <- table(iris$Species)
> barplot(categories, col=c('red', 'green', 'blue'))
>
```

```
> # Plot the histogram
> hist(iris$Sepal.Length, breaks=10, prob=T)
> # Plot the density curve
> lines(density(iris$Sepal.Length))
>
```

```
ggplot(iris, aes(x=iris$SepalLength, y=iris$SepalWidth,
group=iris$Species,shape=iris$Species)) +
geom_point(aes(colour=iris$Species)) +
scale_shape_manual(values=c(19,20,21))+
scale_colour_manual(values=c("blue", "red","gray"))
```

4.5
4.0
3.5
3.0
2.5
2.0
iris$Sepal.Width
5
6
7
8
iris$Sepal.Length
iris$Species
Iris-setosa
Iris-versicolor
Iris-virginica

Covariation

If variation describes the behavior within a variable, **covariation describes the behavior between variables**. **Covariation is the tendency for the values of two or more variables to vary together in a related way**. The best way to spot covariation is to visualize the relationship between two or more variables. How you do that should again depend on the type of variables involved.

1) **A categorical and continuous variable** : We want to explore the distribution of a continuous variable broken down by a categorical variable. Boxplot can help us

```
> boxplot(Sepal.Length~Species, data=iris)
>
```

2) Two continuous variables : One great way to visualize the covariation between two continuous variables: draw a **scatterplot**.

```
> # First plot the 2 variables
> plot(Petal.Width~Sepal.Length, data=iris)
> # Learn the regression model
> model <- lm(Petal.Width~Sepal.Length, data=iris)
> # Plot the regression line
> abline(model)
> # Now learn the local linear model
> model2 <- lowess(iris$Petal.Width~iris$Sepal.Length)
> lines(model2, col="red")
>
```

```
> # Scatter plot for all pairs
> pairs(iris[,c(1,2,3,4)])
> # Compute the correlation matrix
> cor(iris[,c(1,2,3,4)])
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1170695    0.8716902   0.8179410
Sepal.Width    -0.1170695   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8716902  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179410  -0.3661259    0.9628654   1.0000000
>
```

Workshop 4.1 - Getting Data

- Choose data from dataset below.
  - Sales Win or Loss Dataset
  - HR Employee Attrition Dataset
  - Telco Customer Churn Dataset
  - Titanic Survival Dataset
- Import data
- Practice data visualization
- Do step on page 42-51

Data Preparation

Data Science Certification

Preparing Training Data

At this step, the purpose is to transform the raw data in a form that can fit into the data mining model.

- Data sampling
- Data validation and handle missing data
- Normalize numeric value into a uniform range
- Compute aggregated value (a special case is to compute frequency counts)
- Expand categorical field to binary fields
- Discretize numeric value into categories
- Create derived fields from existing fields
- Reduce dimensionality
- Power and Log transformation

Data Sampling

```
> # select 10 records out from iris with replacement
> index <- sample(1:nrow(iris), 10, replace=T)
> index
 [1] 133  36 107 140  66  67  36   3  97  37
> irissample <- iris[index,]
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37            5.5         3.5          1.3         0.2     setosa
>
```

Impute missing data

- Discard the whole record
- Infer missing value based on the data of other record. Approach is to fill the missing data with the average or the median.

```
> # Create some missing data
> irissample[10, 1] <- NA
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37             NA         3.5          1.3         0.2     setosa
```

```
> library(e1071)
Loading required package: class
Warning message:
package ‘e1071’ was built under R version 2.14.2
> fixIris1 <- impute(irissample[,1:4], what='mean')
> fixIris1
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133      6.400000         2.8          5.6         2.2
36       5.000000         3.2          1.2         0.2
107      4.900000         2.5          4.5         1.7
140      6.900000         3.1          5.4         2.1
66       6.700000         3.1          4.4         1.4
67       5.600000         3.0          4.5         1.5
36.1     5.000000         3.2          1.2         0.2
3        4.700000         3.2          1.3         0.2
97       5.700000         2.9          4.2         1.3
37       5.655556         3.5          1.3         0.2
> fixIris2 <- impute(irissample[,1:4], what='median')
> fixIris2
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133           6.4         2.8          5.6         2.2
36            5.0         3.2          1.2         0.2
107           4.9         2.5          4.5         1.7
140           6.9         3.1          5.4         2.1
66            6.7         3.1          4.4         1.4
67            5.6         3.0          4.5         1.5
36.1          5.0         3.2          1.2         0.2
3             4.7         3.2          1.3         0.2
97            5.7         2.9          4.2         1.3
37            5.6         3.5          1.3         0.2
>
```

Normalize numeric value

```
> library(BBmisc)
> scaleiris <- normalize(iris[,1:4], method = "standardize", range = c(0, 1), margin = 1L, on.constant = "quiet")
> head(scaleiris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1   -0.8976739  1.01560199    -1.335752   -1.311052
2   -1.1392005 -0.13153881    -1.335752   -1.311052
3   -1.3807271  0.32731751    -1.392399   -1.311052
4   -1.5014904  0.09788935    -1.279104   -1.311052
5   -1.0184372  1.24503015    -1.335752   -1.311052
6   -0.5353840  1.93331463    -1.165809   -1.048667
>
```

Reduce dimensionality

There are many ways to reduce the number of input attributes.

1. Removing irrelevant input variables.
2. Removing redundant input variables.
3. PCA

Motivation

Clustering

One way to summarize a complex real-valued data point with a single categorical variable

Dimensionality reduction

Another way to simplify complex high-dimensional data

Summarize data with a lower dimensional real valued vector

- Given data points in d dimensions
- Convert them to data points in $r<d$ dimensions
- With minimal loss of information

Data Compression

Reduce data from 2D to 1D

Andrew N

Data Compression

Reduce data from 2D to 1D

$$x^{(1)} \rightarrow z^{(1)}$$
$$x^{(2)} \rightarrow z^{(2)}$$
$$\vdots$$
$$x^{(m)} \rightarrow z^{(m)}$$

Andrew N

Data Compression

Reduce data from 3D to 2D

Andrew N

Principal Component Analysis (PCA) problem formulation

$3D \rightarrow 2D$
$K = 2$

Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Principal Component Analysis

Goal: Find r-dim projection that best preserves variance

1. Compute mean vector μ and covariance matrix Σ of original points
2. Compute eigenvectors and eigenvalues of Σ
3. Select top r eigenvectors
4. Project points onto subspace spanned by them:

$$y = A(x - \mu)$$

where y is the new point, x is the old one,
and the rows of A are the eigenvectors

Covariance

Variance and Covariance:

Measure of the “spread” of a set of points around their center of mass(mean)

Variance:

Measure of the deviation from the mean for points in one dimension

Covariance:

Measure of how much each of the dimensions vary from the mean with **respect to each other**

- **Covariance is measured between two dimensions**
- **Covariance sees if there is a relation between two dimensions**
- **Covariance between one dimension is the variance**

Positive: Both dimensions increase or decrease together

Negative: While one increase the other decrease

Covariance

Used to find relationships between dimensions in high dimensional data sets

$$q_{jk} = \frac{1}{N} \sum_{i=1}^{N} \left(X_{ij} - E(X_j)\right)\left(X_{ik} - E(X_k)\right)$$

The Sample mean

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ: Eigenvector or characteristic vector

X: Eigenvalue or characteristic value

- ***The zero vector can not be an eigenvector***
- ***The value zero can be eigenvalue***

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ: Eigenvector or characteristic vector

X: Eigenvalue or characteristic value

Example

$$Show\ \ x = \begin{bmatrix} 2 \\ 1 \end{bmatrix} is\, an\, eigenvector\ for\ A = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix}$$

$$Solution: Ax = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$But\ \ for\ \ \lambda = 0,\ \ \lambda x = 0\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$Thus, x\, is\, an\, eigenvector\, of\ A, and\, \lambda = 0\ is\, an\, eigenvalue.$$

The space of all face images

- When viewed as vectors of pixel values, face images are extremely high-dimensional
  - 100x100 image = 10,000 dimensions
  - Slow and lots of storage
- But very few 10,000-dimensional vectors are valid face images
- We want to effectively model the subspace of face images

slide by Derek Hoiem

Eigenfaces example

Mean: μ

Top eigenvectors: $u_1,\ldots u_k$

slide by Derek Hoiem

Representation and reconstruction

- Face **x** in "face space" coordinates:

$$\mathbf{x} \rightarrow [\mathbf{u}_1^{\mathrm{T}}(\mathbf{x}-\mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x}-\mu)]$$
$$= \quad w_1, \ldots, w_k$$

- Reconstruction:

$\hat{x} = \mu + w_1u_1+w_2u_2+w_3u_3+w_4u_4+ \ldots$

slide by Derek Hoiem

Reconstruction

P = 4

P = 200

P = 400

After computing eigenfaces using 400 face images from ORL face database

slide by Derek Hoiem

SIFT feature visualization

- The top three principal components of SIFT descriptors from a set of images are computed
- Map these principal components to the principal components of the RGB space
- pixels with similar colors share similar structures

Application: Image compression

- Divide the original 372x492 image into patches:
  - Each patch is an instance that contains 12x12 pixels on a grid
- View each as a 144-D vector

PCA compression: 144D → 60D

PCA compression: 144D → 16D

16 most important eigenvectors

PCA compression: 144D) 6D

6 most important eigenvectors

PCA compression: 144D → 3D

3 most important eigenvectors

PCA compression: 144D → 1D

60 most important eigenvectors

Looks like the discrete cosine bases of JPG!...

2D Discrete Cosine Basis

http://en.wikipedia.org/wiki/Discrete_cosine_transform

Dimensionality reduction

PCA (Principal Component Analysis):

Find projection that maximize the variance

ICA (Independent Component Analysis):

Very similar to PCA except that it assumes non-Guassian features

Multidimensional Scaling:

Find projection that best preserves inter-point distances

LDA(Linear Discriminant Analysis):

Maximizing the component axes for class-separation

...

Principle Component Analysis

```r
# Load data
data(iris)
head(iris, 3)

  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

# log transform
log.ir <- log(iris[, 1:4])
ir.species <- iris[, 5]

# apply PCA - scale. = TRUE is highly
# advisable, but default is FALSE.
ir.pca <- prcomp(log.ir,
                 center = TRUE,
                 scale. = TRUE)

# print method
print(ir.pca)

Standard deviations:
[1] 1.7124583 0.9523797 0.3647029 0.1656840

Rotation:
                    PC1         PC2        PC3         PC4
Sepal.Length  0.5038236 -0.45499872  0.7088547  0.19147575
Sepal.Width  -0.3023682 -0.88914419 -0.3311628 -0.09125405
Petal.Length  0.5767881 -0.03378802 -0.2192793 -0.78618732
Petal.Width   0.5674952 -0.03545628 -0.5829003  0.58044745
```

```r
# summary method
summary(ir.pca)

Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7125 0.9524 0.36470 0.16568
Proportion of Variance 0.7331 0.2268 0.03325 0.00686
Cumulative Proportion  0.7331 0.9599 0.99314 1.00000
```

```r
# Predict PCs
predict(ir.pca,
        newdata=tail(log.ir, 2))

           PC1          PC2         PC3          PC4
149 1.0809930 -1.01155751 -0.7082289 -0.06811063
150 0.9712116 -0.06158655 -0.5008674 -0.12411524
```

```r
library(devtools)
install_github("ggbiplot", "vqv")

library(ggbiplot)
g <- ggbiplot(ir.pca, obs.scale = 1, var.scale = 1,
              groups = ir.species, ellipse = TRUE,
              circle = TRUE)
g <- g + scale_color_discrete(name = '')
g <- g + theme(legend.direction = 'horizontal',
               legend.position = 'top')
print(g)
```

Add derived attributes

```
> iris2 <- transform(iris, ratio=round(Sepal.Length/Sepal.Width, 2))
> head(iris2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio
1          5.1         3.5          1.4         0.2  setosa  1.46
2          4.9         3.0          1.4         0.2  setosa  1.63
3          4.7         3.2          1.3         0.2  setosa  1.47
4          4.6         3.1          1.5         0.2  setosa  1.48
5          5.0         3.6          1.4         0.2  setosa  1.39
6          5.4         3.9          1.7         0.4  setosa  1.38
```

Discretize numeric value into categories

library(arules)

```
data(iris)
x <- iris[,1]

### look at the distribution before discretizing
hist(x, breaks = 20, main = "Data")

def.par <- par(no.readonly = TRUE) # save default
layout(mat = rbind(1:2,3:4))

### convert continuous variables into categories (there are 3 types of flowers)
### the default method is equal frequency
table(discretize(x, breaks = 3))
hist(x, breaks = 20, main = "Equal Frequency")
abline(v = discretize(x, breaks = 3,
  onlycuts = TRUE), col = "red")
# Note: the frequencies are not exactly equal because of ties in the data

### equal interval width
table(discretize(x, method = "interval", breaks = 3))
hist(x, breaks = 20, main = "Equal Interval length")
abline(v = discretize(x, method = "interval", breaks = 3,
  onlycuts = TRUE), col = "red")

### k-means clustering
table(discretize(x, method = "cluster", breaks = 3))
hist(x, breaks = 20, main = "K-Means")
abline(v = discretize(x, method = "cluster", breaks = 3,
  onlycuts = TRUE), col = "red")

### user-specified (with labels)
table(discretize(x, method = "fixed", breaks = c(-Inf, 6, Inf),
    labels = c("small", "large")))
hist(x, breaks = 20, main = "Fixed")
abline(v = discretize(x, method = "fixed", breaks = c(-Inf, 6, Inf),
    onlycuts = TRUE), col = "red")

par(def.par)  # reset to default
```

Equal Frequency
Equal Interval length
K-Means
Fixed
Frequency
x
0
5
10
15
5
6
7
8

Binarize categorical attributes

```
> library(fastDummies)
> results <- dummy_cols(iris)
> results[45:55,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species Species_setosa Species_versicolor Species_virginica
45          5.1         3.8          1.9         0.4     setosa               1                  0                 0
46          4.8         3.0          1.4         0.3     setosa               1                  0                 0
47          5.1         3.8          1.6         0.2     setosa               1                  0                 0
48          4.6         3.2          1.4         0.2     setosa               1                  0                 0
49          5.3         3.7          1.5         0.2     setosa               1                  0                 0
50          5.0         3.3          1.4         0.2     setosa               1                  0                 0
51          7.0         3.2          4.7         1.4 versicolor               0                  1                 0
52          6.4         3.2          4.5         1.5 versicolor               0                  1                 0
53          6.9         3.1          4.9         1.5 versicolor               0                  1                 0
54          5.5         2.3          4.0         1.3 versicolor               0                  1                 0
55          6.5         2.8          4.6         1.5 versicolor               0                  1                 0
>
```

Workshop 4.2 - Data Preparation

- Use data from Workshop 4.1
- Practice data preparation using step from Page 53-78

Machine Learning Techniques

Fundamental Data Science for Data Scientist

Topic

- **Basic concepts**
- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

An example application

An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.

A decision is needed: whether to put a new patient in an intensive-care unit.

Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.

Problem: to predict **high-risk patients** and discriminate them from **low-risk patients**.

Another application

A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,

- age
- Marital status
- annual salary
- outstanding debts
- credit rating

etc.

Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved.

Machine learning and our focus

- Like human learning from past experiences.
- A computer does not have “experiences”.
- **A computer system learns from data**, which represent some “past experiences” of an application domain.
- **Our focus**: learn **a target function** that can be used to predict the values of a discrete class attribute, e.g., **approve** or **not-approved**, and **high-risk** or **low risk**.
- The task is commonly called: **Supervised learning**, **classification**, or **inductive learning**.

The data and the goal

Data: A set of data records (also called examples, instances or cases) described by

k attributes: $A_1, A_2, \ldots A_k$.

a class: Each example is labelled with a pre-defined class.

Goal: To learn a **classification model** from the data that can be used to predict the classes of new (future, or test) cases/instances.

An example: data (loan application)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

An example: the learning task

- Learn a classification model from the data
- Use the model to classify future loan applications into
  - Yes (approved) and
  - No (not approved)
- What is the class for following case/instance?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

Supervised vs. unsupervised Learning

Supervised learning: classification is seen as supervised learning from examples.

- Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).
- Test data are classified into these classes too.

Unsupervised learning (clustering)

- Class labels of the data are unknown
- Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

- Learning (training): Learn a model using the training data
- Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$

What do we mean by learning?

Given

a data set *D*,

a task *T*, and

a performance measure *M*,

a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.

In other words, the learned model helps the system to perform *T* better as compared to no learning.

An example

Data: Loan application data

Task: Predict whether a loan should be approved or not.

Performance measure: accuracy.

No learning: classify all future applications (test data) to the majority class (i.e., Yes):

Accuracy = 9/15 = 60%.

We can do better than 60% with learning.

Fundamental assumption of learning

Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

Supervised Learning

Sebastian Raschka 2014
This work is licensed under a Creative Commons Attribution 4.0 International License.

Topic

- Basic concepts
- **Decision tree induction**
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Introduction

- Decision tree learning is one of the most widely used techniques for classification.
  - Its classification accuracy is competitive with other methods, and
  - it is very efficient.
- The classification model is a tree, called decision tree.
- C4.5 by Ross Quinlan is perhaps the best known system. It can be downloaded from the Web.

The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

A decision tree from the loan data

- Decision nodes and leaf nodes (classes)

Use the decision tree

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	No

Is the decision tree unique?

- No. Here is a simpler tree.
- We want smaller tree and accurate tree.
  - Easy to understand and perform better.

- Finding the best tree is NP-hard.
- All current tree building algorithms are heuristic algorithms

NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP"

From a decision tree to a set of rules

- A decision tree can be converted to a set of rules
- Each path from the root to a leaf is a rule.

Own_house = true → Class =Yes [sup=6/15, conf=6/6]
Own_house = false, Has_job = true → Class = Yes [sup=5/15, conf=5/5]
Own_house = false, Has_job = false → Class = No [sup=4/15, conf=4/4]

Algorithm for decision tree learning

Basic algorithm (a greedy **divide-and-conquer** algorithm)

- Assume attributes are categorical now (continuous attributes can be handled too)
- Tree is constructed in a top-down recursive manner
- At start, all the training examples are at the root
- Examples are partitioned recursively based on selected attributes
- Attributes are selected on the basis of an impurity function (e.g., information gain)

Conditions for stopping partitioning

- All examples for a given node belong to the same class
- There are no remaining attributes for further partitioning – majority class is the leaf
- There are no examples left

Decision tree learning algorithm

```
. Algorithm decisionTree(D, A, T)
    if D contains only training examples of the same class c_j ∈ C then
        make T a leaf node labeled with class c_j;
    elseif A = ∅ then
        make T a leaf node labeled with c_j, which is the most frequent class in D
    else  // D contains examples belonging to a mixture of classes. We select a single
          // attribute to partition D into subsets so that each subset is purer
        p_0 = impurityEval-1(D);
        for each attribute A_i ∈ {A_1, A_2, …, A_k} do
            p_i = impurityEval-2(A_i, D)
        end
        Select A_g ∈ {A_1, A_2, …, A_k} that gives the biggest impurity reduction,
            computed using p_0 − p_i;
        if p_0 − p_g < threshold then     // A_g does not significantly reduce impurity p_0
          make T a leaf node labeled with c_j, the most frequent class in D.
        else                              // A_g is able to reduce impurity p_0
            Make T a decision node on A_g;
            Let the possible values of A_g be v_1, v_2, …, v_m. Partition D into m
                disjoint subsets D_1, D_2, …, D_m based on the m values of A_g.
            for each D_j in {D_1, D_2, …, D_m} do
                if D_j ≠ ∅ then
                    create a branch (edge) node T_j for v_j as a child node of T;
                    decisionTree(D_j, A-{A_g}, T_j)// A_g is removed
                end
            end
        end
    end
```

Choose an attribute to partition data

- The *key* to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.

  A subset of data is pure if all instances belong to the same class.
- The *heuristic* in C4.5 is to choose the attribute with the maximum Information Gain or Gain Ratio based on information theory.

The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Two possible roots, which is better?

- Fig. (B) seems to be better.

Handling continuous attributes

- Handle continuous attribute by splitting into two intervals (can be more) at each node.
- How to find the best threshold to divide?
  - Use information gain or gain ratio again
  - Sort all the values of an continuous attribute in increasing order $\{v_1, v_2, \ldots, v_r\}$,
  - One possible threshold between two adjacent values v_i and v_{i+1}. Try all possible thresholds and find the one that maximizes the gain (or gain ratio).

An example in a continuous space

(A) A partition of the data space

(B). The decision tree

Avoid overfitting in classification

Overfitting: A tree may overfit the training data

- Good accuracy on training data but poor on test data
- Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers

Two approaches to avoid overfitting

- Pre-pruning: Halt tree construction early
  - Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.
- Post-pruning: Remove branches or sub-trees from a "fully grown" tree.
  - This method is commonly used. C4.5 uses a statistical method to estimates the errors at each node for pruning.
  - A validation set may be used for pruning as well.

An example

Likely to overfit the data

(A) A partition of the data space

(B). The decision tree

Other issues in decision tree learning

- From tree to rules, and rule pruning
- Handling of miss values
- Handing skewed distributions
- Handling attributes and classes with different costs.
- Attribute construction
- Etc.

Decision Tree in R

Fundamental Data Science for Data Scientist

Iris Data Preparation

```
> set.seed(1234)
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Decision Tree

Conditional Inference Trees

Description

Recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework.

Usage

```
ctree(formula, data, subset = NULL, weights = NULL,
      controls = ctree_control(), xtrafo = ptrafo, ytrafo = ptrafo,
      scores = NULL)
```

Arguments

`formula`	a symbolic description of the model to be fit. Note that symbols like `:` and `-` will not work and the tree will make use of all variables listed on the rhs of `formula`.
`data`	a data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Only non-negative integer valued weights are allowed.
`controls`	an object of class `TreeControl`, which can be obtained using `ctree_control`.
`xtrafo`	a function to be applied to all input variables. By default, the `ptrafo` function is applied.
`ytrafo`	a function to be applied to all response variables. By default, the `ptrafo` function is applied.
`scores`	an optional named list of scores to be attached to ordered factors.

Decision Tree - Create Model

```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # check the prediction
> table(predict(iris_ctree), trainData$Species)
```

	setosa	versicolor	virginica
setosa	40	0	0
versicolor	0	37	3
virginica	0	1	31

```
> print(iris_ctree)

         Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  112

1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
  2)*  weights = 40
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
    4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
      5)*  weights = 21
    4) Petal.Length > 4.4
      6)*  weights = 19
  3) Petal.Width > 1.7
    7)*  weights = 32
```

```
> plot(iris_ctree)
```

```
> plot(iris_ctree, type="simple")
```

Decision Tree - Prediction

```
> # predict on test data
> testPred <- predict(iris_ctree, newdata = testData)
> table(testPred, testData$Species)

testPred     setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         12         2
  virginica       0          0        14
```

Workshop 4.3 - Decision Tree

1. Use data from Workshop 2
2. Perform Decision Tree Algorithm to predict outcomes

Topic

- Basic concepts
- Decision Tree
- **Logistic Regression**
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Overview of Logistic Regression

A linear model for classification and probability estimation.

Can be very effective when:

- The problem is linearly separable
- Or there are a lot of relevant features (10s - 100s of thousands can work)
- You need something simple and efficient as a baseline
- Efficient runtime

Logistic regression will generally not be the most accurate option.

Components of Learning Algorithm:

Model Structure –

Linear model with sigmoid activation

Loss Function –

Log Loss

Optimization Method –

Gradient Descent

Structure of Logistic Regression

Bias Weight

Weight per Feature

$$\text{Linear Model: } w_0 + \sum_{i}^{n} w_i * x_i$$

$$\hat{y} = sigmoid(w_0 + \sum_{i}^{n} w_i * x_i)$$

Example:

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

.25	-1	1

1	1
1	0
0	1
0	0

Intuition about additional dimensions

3 Dimensions
Decision surface is plane

N-Dimensions
Decision surface is n-dimensional hyper-plane

High-dimensions are weird
High-dimensional hyper-planes can represent quite a lot

.25	-1	1

Loss Function: Log Loss

y^ -- The predicted y (pre-threshold)

Log Loss:

Examples If y is 1: $-\log(y\hat{})$

If y is 0: $-\log(1 - y\hat{})$

		Loss
.9	1	.105
.5	1	.693
.9	0	2.3
.95	0	2.99

Use Natural Log (base e)

Logistic Regression Optimization

Initialize model weights to 0

Do ‘numIterations’ steps of gradient descent (thousands of steps)

- Find the gradient for each weight by averaging across the training set
- Update each weight by taking a step of size $\propto$ opposite the gradient

Parameters

- $\propto$ – size of the step to take in each iteration
- numIterations – number of iterations of gradient descent to perform
  - Or use a convergence criteria…
- Threshold – value between 0-1 to convert $\hat{y}$ into a classification

Logistic using R

```r
library(nnet)
model_multi = multinom(Species ~ ., data = iris_trn, trace = FALSE)
summary(model_multi)$coefficients
```

```
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    26.81602    -6.983313   -16.24574     20.35750    3.218787
## virginica    -34.24228    -8.398869   -17.03985     31.94659   11.594518
```

```r
head(predict(model_multi, newdata = iris_trn))
```

```
## [1] setosa    virginica setosa    setosa    virginica setosa
## Levels: setosa versicolor virginica
```

```r
head(predict(model_multi, newdata = iris_trn, type = "prob"))
```

```
##           setosa   versicolor    virginica
## 23  1.000000e+00 2.607782e-19 3.891079e-44
## 106 4.461651e-38 2.328295e-09 1.000000e+00
```

Workshop 4.4 - Logistic Regression

1. Use data from Workshop 4.2
2. Perform Logistic Regression Algorithm to predict outcomes

Evaluating classification methods

Predictive accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

Efficiency

time to construct the model

time to use the model

Robustness: handling noise and missing values

Scalability: efficiency in disk-resident databases

Interpretability:

understandable and insight provided by the model

Compactness of the model: size of the tree, or the number of rules.

Evaluation methods

- **Holdout set**: The available data set D is divided into two disjoint subsets,
  - the *training set* D_{train} (for learning a model)
  - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
  - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the holdout set. (the examples in the original data set D are all labeled with classes.)
- This method is mainly used when the data set D is large.

Evaluation methods (cont...)

- **n-fold cross-validation**: The available data is partitioned into *n* equal-size disjoint subsets.
- Use each subset as the test set and combine the rest *n*-1 subsets as the training set to learn a classifier.
- The procedure is run *n* times, which give *n* accuracies.
- The final estimated accuracy of learning is the average of the *n* accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Evaluation methods (cont...)

- **Leave-one-out cross-validation**: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has *m* examples, this is *m*-fold cross-validation

Evaluation methods (cont...)

- Validation set: the available data is divided into three subsets,
  - a training set,
  - a validation set and
  - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Classification measures

- Accuracy is only one measure (error = 1-accuracy).
- **Accuracy is not suitable in some applications**.
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, we are interested only in the minority class.
  - High accuracy does not mean any intrusion is detected.
  - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the **positive class**, and the rest **negative classes**.

Precision and recall measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),
FN: the number of incorrect classifications of positive examples (**false negative**),
FP: the number of incorrect classifications of negative examples (**false positive**), and
TN: the number of correct classifications of negative examples (**true negative**).

Precision and recall measures (cont…)

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP}. \qquad r = \frac{TP}{TP + FN}.$$

- Precision p is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.
- Recall r is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

An example

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- This confusion matrix gives

  precision p = 100% and

  recall r = 1%

- because we only classified one positive example correctly and no negative examples wrongly.
- Note: precision and recall only measure classification on the positive class.

F_1-value (also called F_1-score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p+r}$$

F_1-score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F_1-value to be large, both p and r much be large.

Receive operating characteristics curve

- It is commonly called the ROC curve.
- It is a plot of the true positive rate (TPR) against the false positive rate (FPR).
- True positive rate:

$$TPR = \frac{TP}{TP + FN}$$

- False positive rate:

$$FPR = \frac{FP}{TN + FP}$$

Sensitivity and Specificity

- In statistics, there are two other evaluation measures:
  - Sensitivity: Same as TPR
  - Specificity: Also called True Negative Rate (TNR)

- Then we have

$$TNR = \frac{TN}{TN + FP}$$

$$FPR = 1 - specificity$$

Example ROC curves

Fig. 3.8. ROC curves for two classifiers (C_1 and C_2) on the same data

Area under the curve (AUC)

- Which classifier is better, C_1 or C_2?
  - It depends on which region you talk about.
- Can we have one measure?
  - Yes, we compute the area under the curve (AUC)
- If AUC for C_i is greater than that of C_j, it is said that C_i is better than C_j.
  - If a classifier is perfect, its AUC value is 1
  - If a classifier makes all random guesses, its AUC value is 0.5.

Drawing an ROC curve

Rank		1	2	3	4	5	6	7	8	9	10
Actual class		+	+	–	–	+	–	–	+	–	–
TP	0	1	2	2	2	3	3	3	4	4	4
FP	0	0	0	1	2	2	3	4	4	5	6
TN	6	6	6	5	4	4	3	2	2	1	0
FN	4	3	2	2	2	1	1	1	0	0	0
TPR	0	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1	1	1
FPR	0	0	0	0.17	0.33	0.33	0.50	0.67	0.67	0.83	1

Another evaluation method: Scoring and ranking

- Scoring is related to classification.
- We are interested in a single class (positive class), e.g., buyers class in a marketing database.
- Instead of assigning each test instance a definite class, scoring assigns a probability estimate (PE) to indicate the likelihood that the example belongs to the positive class.

Ranking and lift analysis

- After each example is given a PE score, we can rank all examples according to their PEs.
- We then divide the data into n (say 10) bins. A lift curve can be drawn according how many positive examples are in each bin. This is called lift analysis.
- Classification systems can be used for scoring. Need to produce a probability estimate.
  - E.g., in decision trees, we can use the confidence value at each leaf node as the score.

An example

- We want to send promotion materials to potential customers to sell a watch.
- Each package cost $0.50 to send (material and postage).
- If a watch is sold, we make $5 profit.
- Suppose we have a large amount of past data for building a predictive/classification model. We also have a large list of potential customers.
- How many packages should we send and who should we send to?

An example

Assume that the test set has 10000 instances. Out of this, 500 are positive cases.
After the classifier is built, we score each test instance. We then rank the test set, and divide the ranked test set into 10 bins.

Each bin has 1000 test instances.

Bin 1 has 210 actual positive instances
Bin 2 has 120 actual positive instances
Bin 3 has 60 actual positive instances
...
Bin 10 has 5 actual positive instances

Lift curve

Bin	1	2	3	4	5	6	7	8	9	10
210	120	60	40	22	18	12	7	6	5
42%	24%	12%	8%	4.40%	3.60%	2.40%	1.40%	1.20%	1%
42%	66%	78%	86%	90.40%	94%	96.40%	97.80%	99%	100%

Evaluation of Classifier in R

Fundamental Data Science for Data Scientist

Confusion Matrix

install.packages("caret")

library(caret)

confusionMatrix(testPred, testdata$Species)

```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              13              2
  Iris-virginica            0               0             13

Overall Statistics

               Accuracy : 0.9535
                 95% CI : (0.8419, 0.9943)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9303
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                       1.0000                 1.0000                0.8667
Specificity                       1.0000                 0.9333                1.0000
Pos Pred Value                    1.0000                 0.8667                1.0000
Neg Pred Value                    1.0000                 1.0000                0.9333
Prevalence                        0.3488                 0.3023                0.3488
Detection Rate                    0.3488                 0.3023                0.3023
Detection Prevalence              0.3488                 0.3488                0.3023
Balanced Accuracy                 1.0000                 0.9667                0.9333
```

Creating ROC

```
library(ROCR)
df <- trainData[,1:4]
df$virgi <- as.numeric(trainData$Species=="Iris-virginica")
svmMod <- svm(virgi ~ ., data = df)

svmPred <- predict(svmMod, df[,-5])

svmPredictiction <- prediction(svmPred, df$virgi)
svmPerf <- performance(svmPredictiction, "tpr", "fpr")

plot(svmPerf)
abline(a=0, b= 1)

library(pROC)
predictions <- as.numeric(predict(svmfit, testData, type = "response"))
multiclass.roc(testData$Species,predictions)
```

```
> multiclass.roc(testData$Species,predictions)

Call:
multiclass.roc.default(response = testData$Species, predictor = predictions)

Data: predictions with 3 levels of testData$Species: Iris-setosa, Iris-versicolor, Iris-virg
inica.
Multi-class area under the curve: 1
```

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- **Naïve Bayesian classification**
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Bayesian classification

Probabilistic view: Supervised learning can naturally be studied from a probabilistic point of view.

Let A_1 through A_k be attributes with discrete values. The class is C.

Given a test example *d* with observed attribute values a_1 through a_k.

Classification is basically to compute the following posteriori probability.

The prediction is the class c_j such that

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

is maximal

Apply Bayes' Rule

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

$$= \frac{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_j)\Pr(C = c_j)}{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|})}$$

$$= \frac{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_j)\Pr(C = c_j)}{\sum_{r=1}^{|C|} \Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_r)\Pr(C = c_r)}$$

- $\Pr(C=c_j)$ is the class prior probability: easy to estimate from the training data.

Computing probabilities

The denominator $P(A_1=a_1,\ldots,A_k=a_k)$ is irrelevant for decision making since it is the same for every class.

We only need $P(A_1=a_1,\ldots,A_k=a_k \mid C=c_i)$, which can be written as

$$\Pr(A_1=a_1|A_2=a_2,\ldots,A_k=a_k, C=c_j)* \Pr(A_2=a_2,\ldots,A_k=a_k \mid C=c_j)$$

Recursively, the second factor above can be written in the same way, and so on.

Now an assumption is needed.

Conditional independence assumption

All attributes are conditionally independent given the class $C = c_j$.

Formally, we assume,

$$\Pr(A_1=a_1 \mid A_2=a_2, \ldots, A_{|A|}=a_{|A|}, C=c_j) = \Pr(A_1=a_1 \mid C=c_j)$$

and so on for A_2 through $A_{|A|}$. I.e.,

$$\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_i) = \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

Final naïve Bayesian classifier

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

$$= \frac{\Pr(C = c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)}{\sum_{r=1}^{|C|} \Pr(C = c_r) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_r)}$$

Classify a test instance

If we only need a decision on the most probable class for the test instance, we only need the numerator as its denominator is the same for every class.

Thus, given a test example, we compute the following to decide the most probable class for the test instance

$$c = \arg\max_{c_j} \Pr(c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

An example

- Compute all probabilities required for classification

A	B	C
m	b	t
m	s	t
g	q	t
h	s	t
g	q	t
g	q	f
g	s	f
h	b	f
h	q	f
m	b	f

$Pr(C = t) = 1/2$, $Pr(C = f) = 1/2$

$Pr(A=m \mid C=t) = 2/5$ $Pr(A=g \mid C=t) = 2/5$ $Pr(A=h \mid C=t) = 1/5$

$Pr(A=m \mid C=f) = 1/5$ $Pr(A=g \mid C=f) = 2/5$ $Pr(A=h \mid C=n) = 2/5$

$Pr(B=b \mid C=t) = 1/5$ $Pr(B=s \mid C=t) = 2/5$ $Pr(B=q \mid C=t) = 2/5$

$Pr(B=b \mid C=f) = 2/5$ $Pr(B=s \mid C=f) = 1/5$ $Pr(B=q \mid C=f) = 2/5$

Now we have a test example:

A = m B = q C = ?

An Example (cont ...)

For C = t, we have

$$\Pr(C=t)\prod_{j=1}^{2}\Pr(A_j=a_j \mid C=t)=\frac{1}{2}\times\frac{2}{5}\times\frac{2}{5}=\frac{2}{25}$$

For class C = f, we have

$$\Pr(C=f)\prod_{j=1}^{2}\Pr(A_j=a_j \mid C=f)=\frac{1}{2}\times\frac{1}{5}\times\frac{2}{5}=\frac{1}{25}$$

C = t is more probable. t is the final class.

Additional issues

Numeric attributes: Naïve Bayesian learning assumes that all attributes are categorical. Numeric attributes need to be discretized.

Zero counts: An particular attribute value never occurs together with a class in the training set. We need smoothing.

Missing values: Ignored

$$\Pr(A_i = a_i \mid C = c_j) = \frac{n_{ij} + \lambda}{n_j + \lambda n_i}$$

On naïve Bayesian classifier

Advantages:

Easy to implement

Very efficient

Good results obtained in many applications

Disadvantages

Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated (those highly correlated data sets)

naïve Bayesian classifier in R

Fundamental Data Science for Data Scientist

Naive Bayes

Naive Bayes Classifier

Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'formula'
naiveBayes(formula, data, laplace = 0, ..., subset, na.action = na.pass)
## Default S3 method:
naiveBayes(x, y, laplace = 0, ...)


## S3 method for class 'naiveBayes'
predict(object, newdata,
  type = c("class", "raw"), threshold = 0.001, eps = 0, ...)
```

Naive Bayes

Arguments

`x`	A numeric matrix, or a data frame of categorical and/or numeric variables.
`y`	Class vector.
`formula`	A formula of the form `class ~ x1 + x2 + ...`. Interactions are not allowed.
`data`	Either a data frame of predictors (categorical and/or numeric) or a contingency table.
`laplace`	positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
`...`	Currently not used.
`subset`	For data given in a data frame, an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is not to count them for the computation of the probability factors. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
`object`	An object of class `"naiveBayes"`.
`newdata`	A dataframe with new predictors (with possibly fewer columns than the training data). Note that the column names of `newdata` are matched against the training data ones.
`type`	If `"raw"`, the conditional a-posterior probabilities for each class are returned, and the class with maximal probability else.
`threshold`	Value replacing cells with probabilities within `eps` range.
`eps`	double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by `theshold`.)

Naive Bayes

```
> library(e1071)
> # Can handle both categorical and numeric input,
> # but output must be categorical
> model <- naiveBayes(Species~., data=iristrain)
> prediction <- predict(model, iristest[,-5])
> table(prediction, iristest[,5])

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8
```

```
> confusionMatrix(prediction, testdata$Species)
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15              0             0
  Iris-versicolor           0             13             2
  Iris-virginica            0              0            13

Overall Statistics

               Accuracy : 0.9535
                 95% CI : (0.8419, 0.9943)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9303
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                      1.0000                 1.0000                0.8667
Specificity                      1.0000                 0.9333                1.0000
Pos Pred Value                   1.0000                 0.8667                1.0000
Neg Pred Value                   1.0000                 1.0000                0.9333
Prevalence                       0.3488                 0.3023                0.3488
Detection Rate                   0.3488                 0.3023                0.3023
Detection Prevalence             0.3488                 0.3488                0.3023
Balanced Accuracy                1.0000                 0.9667                0.9333
```

Workshop 4.5 - Naive Bayes Classifier

1. Use data from Workshop 4.2
2. Perform Naive Bayes Classifier Algorithm to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- **K-nearest neighbor**
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting

k-Nearest Neighbor Classification (kNN)

- Unlike all the previous learning methods, kNN does not build model from the training data.
- To classify a test instance d, define k-neighborhood P as k nearest neighbors of d
- Count number n of training instances in P that belong to class c_j
- Estimate $\Pr(c_j|d)$ as n/k
- No training is needed. Classification time is linear in training set size for each test case.

kNNAlgorithm

Algorithm kNN(D, d, k)

1 Compute the distance between d and every example in D;

2 Choose the k examples in D that are nearest to d, denote the set by P ($\subseteq D$);

3 Assign d the class that is the most frequent class in P (or the majority class);

- *k is usually chosen empirically via a validation set or cross-validation by trying a range of k values.*
- *Distance function is crucial, but depends on applications.*

Example: k=6 (6NN)

Government

Science

Arts

A new point

Pr(science|)?

Discussions

- kNN can deal with complex and arbitrary decision boundaries.
- Despite its simplicity, researchers have shown that the classification accuracy of kNN can be quite strong and in many cases as accurate as those elaborated methods.
- kNN is slow at the classification time
- kNN does not produce an understandable model

k-NN in R

Fundamental Data Science for Data Scientist

K-NN

k-Nearest Neighbour Classification

Description

k-nearest neighbour classification for test set from training set. For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

Usage

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Arguments

`train`	matrix or data frame of training set cases.
`test`	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
`cl`	factor of true classifications of training set
`k`	number of neighbours considered.
`l`	minimum vote for definite decision, otherwise `doubt`. (More precisely, less than `k-l` dissenting votes are allowed, even if `k` is increased by ties.)
`prob`	If this is true, the proportion of the votes for the winning class are returned as attribute `prob`.
`use.all`	controls handling of ties. If true, all distances equal to the `k`th largest are included. If false, a random selection of distances equal to the `k`th is chosen to use exactly `k` neighbours.

K-NN

```
> library(class)
> train_input <- as.matrix(iristrain[,-5])
> train_output <- as.vector(iristrain[,5])
> test_input <- as.matrix(iristest[,-5])
> prediction <- knn(train_input, test_input,
                    train_output, k=5)
> table(prediction, iristest$Species)

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         1
  virginica       0          0         9
>
```

```
> confusionMatrix(prediction, testdata$Species)
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              12              0
  Iris-virginica            0               1             15

Overall Statistics

               Accuracy : 0.9767
                 95% CI : (0.8771, 0.9994)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.965
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor
Sensitivity                      1.0000                 0.9231
Specificity                      1.0000                 1.0000
Pos Pred Value                   1.0000                 1.0000
Neg Pred Value                   1.0000                 0.9677
Prevalence                       0.3488                 0.3023
Detection Rate                   0.3488                 0.2791
Detection Prevalence             0.3488                 0.2791
Balanced Accuracy                1.0000                 0.9615
                     Class: Iris-virginica
Sensitivity                         1.0000
Specificity                         0.9643
Pos Pred Value                      0.9375
Neg Pred Value                      1.0000
Prevalence                          0.3488
Detection Rate                      0.3488
Detection Prevalence                0.3721
Balanced Accuracy                   0.9821
```

myFormula

Workshop 4.6 - k-NN

1. Use data from Workshop 4.2
2. Perform k-NN Algorithm to predict outcomes
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- **Support Vector Machine**
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Support Vector Machine (SVM)

- SVMs maximize the *margin* around the separating hyperplane.
  - A.k.a. large margin classifiers
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- Solving SVMs is a *quadratic programming* problem
- Seen by many as the most successful current text classification method*

Maximum Margin: Formalization

w: decision hyperplane normal vector

$\mathbf{x}_i$: data point *i*

y_i: class of data point *i* (+1 or -1) NB: Not 1/0

Classifier is: $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T\mathbf{x}_i + b)$

Functional margin of $\mathbf{x}_i$ is: $y_i(\mathbf{w}^T\mathbf{x}_i + b)$

But note that we can increase this margin simply by scaling **w**, **b**….

Functional margin of dataset is twice the minimum functional margin for any point

The factor of 2 comes from measuring the whole width of the margin

Geometric Margin

Distance from example to the separator is $r = y\dfrac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$

Examples closest to the hyperplane are ***support vectors***.

Margin ρ of the separator is the width of separation between support vectors of classes.

Derivation of finding *r:*

Dotted line **x'–x** is perpendicular to decision boundary so parallel to **w**.

Unit vector is **w**/|**w**|, so line is r**w**/|**w**|.

x' = **x** – yr**w**/|**w**|.

x' satisfies $\mathbf{w}^T\mathbf{x}'$+b = 0.

So $\mathbf{w}^T$(**x** –yr**w**/|**w**|) + b = 0

Recall that |**w**| = sqrt($\mathbf{w}^T\mathbf{w}$).

So $\mathbf{w}^T\mathbf{x}$ –yr|**w**| + b = 0

So, solving for r gives:

r = y($\mathbf{w}^T\mathbf{x}$ + b)/|**w**|

Linear SVM Mathematically
The linearly separable case

Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set $\{(\mathbf{x_i}, y_i)\}$

$$\mathbf{w}^T\mathbf{x_i} + b \geq 1 \quad \text{if } y_i = 1$$

$$\mathbf{w}^T\mathbf{x_i} + b \leq -1 \quad \text{if } y_i = -1$$

For support vectors, the inequality becomes an equality

Then, since each example's distance from the hyperplane is

$$r = y\frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$$

The margin is:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

Linear Support Vector Machine (SVM)

Hyperplane

$\mathbf{w}^T \mathbf{x} + b = 0$

Extra scale constraint:

$\min_{i=1,\ldots,n} |\mathbf{w}^T\mathbf{x}_i + b| = 1$

This implies:

$w^T(x_a - x_b) = 2$

$\rho = ||x_a - x_b||_2 = 2/||\mathbf{w}||_2$

Support Vector Machine in R

Fundamental Data Science for Data Scientist

Support Vector Machine

svm {e1071} R Documentation

Support Vector Machines

Description

`svm` is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Arguments

`formula`
: a symbolic description of the model to be fit.

`data`
: an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.

`x`
: a data matrix, a vector, or a sparse matrix (object of class `Matrix` provided by the **Matrix** package, or of class `matrix.csr` provided by the **SparseM** package, or of class `simple_triplet_matrix` provided by the **slam** package).

`y`
: a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).

```r

library(tidyverse)     # data manipulation and visualization
library(e1071)         # SVM methodology

# Fit Support Vector Machine model to data set
svmfit <- svm(Species~., data = trainData, method="C-classification",
              kernel = "radial", scale = FALSE, probability=T)
# Plot Results
plot(svmfit, data=trainData, formula=Sepal.Length~Sepal.Width)
prediction <- predict(svmfit, testData, probability=T)
table(testData$Species, prediction)

```

Support Vector Machine

```
> library(caret)
> confusionMatrix(testData$Species, prediction)
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              13              0
  Iris-virginica            0               0             15

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9178, 1)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                       1.0000                 1.0000                1.0000
Specificity                       1.0000                 1.0000                1.0000
Pos Pred Value                    1.0000                 1.0000                1.0000
Neg Pred Value                    1.0000                 1.0000                1.0000
Prevalence                        0.3488                 0.3023                0.3488
Detection Rate                    0.3488                 0.3023                0.3488
Detection Prevalence              0.3488                 0.3023                0.3488
Balanced Accuracy                 1.0000                 1.0000                1.0000
```

Workshop 4.7 - SVM

1. Use data from Workshop 4.2
2. Perform SVM Algorithm to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- **Neural Net**
- Ensemble methods: Bagging and Boosting
- Summary

Neural Network

- Neural networks learning methods provide a robust approach to approximating real-valued, discrete valued and vector valued target functions.
- Often used in problems such as handwriting, voice, or image recognition
- **Feed-Forward Neural Networks**, **Convolutional Neural Networks**, **Deep Learning**

Biological Motivation

- Inspired by biological learning systems that are built from very complex webs of interconnected neurons

Facts About Neuro-biology

- Human brain is estimated to contain a densely interconnected network of approximately 10^{11} neurons.
- Each connected, on average, to 10^4 other neurons.
- Neuron activity is typically exited through connections to other neurons.
- The fastest neuron switching times are in the order of 10^{-3} seconds.
- It requires 10^{-1} seconds to visibly recognize your mother.

Analogy of Biological Learning Systems

- Artificial neural networks are built out of a interconnected set of simple units, where each unit takes a number of real-valued inputs (can be the outputs of other units) and produces a single real-valued output (which may become the input of other units).

Example

A neural network is like a black box that know how to process input to create an output. The calculation is quite complex and difficult to understand.

Neural Networks Process

1. Identify the input and output features
2. Transform the input and output so that they are in a small range (-1 to 1)
3. Set up a network with an appropriate topology
4. Train the network on a representative set of training examples
5. Use the validation set to choose the set of weights that minimizes the error
6. Evaluate the network using the test set to see how well it performs
7. Apply the model generated by the network to predict outcomes for unknown input

What is a Neural Network? (1)

This simple neural network takes four inputs and produces an output. This result of training this network is equivalent to the statistical technique called logistic regression.

This network has a middle layer called the *hidden layer*, which makes the network more powerful by enabling it to recognize more patterns.

What is a Neural Network? (2)

Increasing the size of the hidden layer makes the network more powerful but introduces the risk of overfitting. Usually, only one hidden layer is needed.

A neural network can produce multiple output values.

Three Main Components of a Neural Network

- **Activation function**
- **Network topology**
- **How the network is trained?**
  - Feed forward
  - Back propagation

Activation Function (1)

- **Combination function**
  - The function that merges all the input into a single value.
  - The most common combination function is the weighted sum.
- **Transfer function**
  - The function that transfers the value of the combination function to the output of the unit.

Feed-Forward Neural Networks (1)

Num_Apartments	1	0.0000
Year_Built	1923	0.5328
Plumbing_Fixtures	9	0.3333
Heating_Type	B	1.0000
Basement_Garage	0	0.0000
Attached_Garage	120	0.5263
Living_Area	1614	0.2593
Deck_Area	0	0.0000
Porch_Area	210	0.4646
Recroom_Area	0	0.0000
Basement_Area	175	0.2160

Feed-Forward Neural Networks: Input Layer

- **Input layer**
  - Each unit in the input layer is connected to exactly one source field.

Num_Apartments	1	0.0000
Year_Built	1923	0.5328
Plumbing_Fixtures	9	0.3333
Heating_Type	B	1.0000
Basement_Garage	0	0.0000
Attached_Garage	120	0.5263
Living_Area	1614	0.2593
Deck_Area	0	0.0000
Porch_Area	210	0.4646
Recroom_Area	0	0.0000
Basement_Area	175	0.2160

Feed-Forward Neural Networks : Hidden layer (1)

- **Hidden layer**
  - Each unit in the hidden layer is typically connected to all the units in the input layer.
  - The units in the hidden layer calculate their output by multiplying the value of each input by its corresponding weight, adding them up, and applying the transfer function.

Feed-Forward Neural Networks: Hidden layer (2)

- **Hidden layer**
  - A neural network can have any number of hidden layers, but in general one hidden layer is sufficient.
  - The wider the layer (the more units it contains), the greater the capacity of the network to recognize patterns.

Feed-Forward Neural Networks : Output Layer (1)

- **Output layer**
  - It is fully connected to all the units in the hidden layer.
  - Most of the time, the neural network is being used to calculate a single value, so there is only one unit in the output layer and the value.
  - **We must map this value back to understand the output.**

Feed-Forward Neural Networks: Output Layer (2)

- **Output layer**
  - It is possible for the output layer to have more than one unit.
  - Eg. A department store chain wants to predict the likelihood that customers will be purchasing products from various departments, such as women's apparel, furniture, and entertainments.

Feed-Forward Neural Networks: Output Layer (3)

- After feeding the inputs for a customer into the network, the network calculates three values.

Back-Propagation Algorithm

- Assume the network is a fixed structure that corresponds to a directed graph.
- Learning corresponds to choosing a weight value for each edge in the graph.
- It attempts to minimize the squared error between the network output values and the target values for these outputs.
- It is the ANN learning technique that is most commonly used.

Back Propagation Neural Networks: Steps (1)

1. The network gets a training example and, using the existing weights in the network, it calculates the output.
2. Back propagation calculates the error by taking the difference between the calculated result and the expected (actual result).

Back Propagation Neural Networks: Steps (2)

3. The error is fed back through the network and the weights are adjusted to minimize the error.
4. After being shown enough training examples, the weights on the network no longer change significantly and the error no longer decreases. This is the point where training stops.

Back-Propagation Algorithm: Input

- Training example is a pair of the form <x, t>, where x is the vector of network input values and t is the vector of target network output values
  - n is the learning rate
  - n_{in} is the number of network inputs
  - n_{hidden} is the number of units in the hidden layer
  - n_{out} is the number of output units
- The input from unit i into unit j is denoted x_{ji}
- The weight from unit i to unit j is denoted w_{ji}

Back-Propagation Algorithm: process (1)

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units
- Initialize all network weight to small random numbers
- Until the termination condition is met, Do repeat the steps

Back-Propagation Algorithm: process (2)

Propagate the input forward through the network:

1. Input the instance x to the network and compute the output o_u of every unit u in the network

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term e_k

$$e_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

Back-Propagation Algorithm: process

3. For each hidden unit h, calculate its error term e_h

$$e_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{kh} e_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = n e_j x_{ji}$$

Termination conditions

- Stop after a fixed number of iterations
- Stop when the error on the training examples falls below some threshold
- Stop when the error on a separate validation set of examples meets some criterion

Hidden Layer Representations

- Training examples constrain the network inputs and outputs.
- The weight tuning procedure is free to set weights that define whatever hidden unit representation is most effective in minimizing the squared error.
- The ability of multilayer networks to automatically discover useful representation at the hidden layers is a key feature of ANN learning.

Heuristics for Using Feed-Forward & Back Propagation Networks

- One important decision is the number of units in the hidden layer. The more unit, the more patterns the network can recognize.
- We generally do not use hidden layers larger than the number of inputs.
- A good place to start for many problems is to experiment with one, two, and three nodes in the hidden layer.

Choosing the Training Set: Coverage of values for all Features

- In general, it is good to have several examples in the training set for each value of a categorical feature and for values throughout the ranges of ordered discrete and continuous features.

Choosing the Training Set: Number of Features

- The more features used as inputs into the networks, the larger the networks needs to be, increasing the risk of overfitting and increasing the size of the training set.
- The more features, the longer it takes the network to converge to a set of weights.
- In many cases, it is useful to calculate new variables that represent particular aspects of the business problems.

Choosing the Training Set: Size of Training Set

- The more features there are, the more training examples that are needed to get a good coverage of patterns in the data.
- Overfitting is guaranteed to happen when there are fewer training examples than there are weights in the network.

Choosing the Training Set: Number of Output

- The number of training examples for each possible output should be about the same.

Preparing the Data: Features with Continuous Values

- The value should be scaled to be in a reasonable range.
- Plan for a larger range
- Reject out-of-range values
- Peg values lower than the minimum to the minimum and higher than the maximum to the maximum
- Map the minimum value to –0.9 and the maximum value to 0.9 instead of –1 and 1

Preparing the Data: Features with Ordered Discrete Values (1)

- First, count the number of different values and assign each a proportional fraction into some range.
- eg. If there are five distinct values, these get mapped to 0, 0.25, 0.50, 0.75, and 1.0

$$0 \rightarrow 0000 = 0/16 = 0.0000$$
$$1 \rightarrow 1000 = 8/16 = 0.5000$$
$$2 \rightarrow 1100 = 12/16 = 0.7500$$
$$3 \rightarrow 1110 = 14/16 = 0.8750$$

Preparing the Data: Features with Ordered Discrete Values (2)

- It is also possible to break a range into unequal parts.
- One example is called thermometer codes

  - 0 → 0 0 0 0 = 0 / 16 = 0.0000
  - 1 → 1 0 0 0 = 8 / 16 = 0.5000
  - 2 → 1 1 0 0 = 12 / 16 = 0.7500
  - 3 → 1 1 1 0 = 14 / 16 = 0.8750

  It shows that the difference on one end of the scale is less significant than differences on the other end.

Preparing the Data: Features with Categorical Values (1)

- When working with categorical variables in neural networks, the mapping of variables into numbers introduces an ordering of the variables, which the neural network takes into account.
- The second way of handling categorical features is to break the categories into flags, one for each value.

Preparing the Data: Features with Categorical Values (2)

Gender	N coding			N-1 coding	
	Gender Male Flag	Gender Female Flag	Gender Unknown Flag	Gender Male Flag	Gender Female Flag
Male	+1.0	-1.0	-1.0	+1.0	-1.0
Female	-1.0	+1.0	-1.0	-1.0	+1.0
Unknown	-1.0	-1.0	+1.0	-1.0	-1.0

Interpreting the Results (1)

- When estimating a continuous value, often the output needs to be scaled back to the correct range.
- For binary or categorical output variables, the approach is still to take the inverse of the translation used for training the network.
- eg. If “churn” is given the value of 1 and “no churn” a value of –1, the values near 1 represent churn and those near –1 represent no churn.

Interpreting the Results (2)

- When there are two outcomes, the meaning of the output depends on the training set used to train the network.
- The average value produced by the network during training is usually going to be close to the average value in the training set.

Interpreting the Results (3)

- One way to handle is, eg.
- If the training set had 50% churn and 50% no churn, the average value the network will produce form the training examples is going to be close to 0.0.
- Values higher than 0.0 are more like churn and those less than 0.0, less like churn.
- If the original training set had 10% churn, then the cutoff would more reasonable be –
- 0.8 rather than 0.0 (0.8 is 10% of the way from –1 to 1).

Interpreting the Results (4)

- For binary values, it is also possible to create a network that produces two output, one for each value.
- In this case, each output represents the strength of evidence that that category is the correct one. The chosen category would then be the one with the higher value.

How to Know What is Going on Inside a Neural Network

Sensitivity analysis uses the test set to determine how sensitive the output of the network is to each input :

1. Find the average value for each input.
2. Measure the output of the network when all inputs are at their average value.
3. Measure the output of the network when each input is modified, one at a time, to be at its minimum and maximum values.

Summary

- Neural networks can be used for both categorical and continuous inputs.
- Neural networks learn best when input fields have been mapped to the range between -1 and +1.
- Neural networks work best when there are only a few variables.
- When training a network, there is no guarantee that the resulting set of weights is optimal.
- A neural network cannot explain what it is doing.

GoogLeNet (Inception)

9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other

16

Neural Networks in R

Fundamental Data Science for Data Scientist

Neural Network

Training of neural networks

Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

Neural Network

Arguments

`formula`	a symbolic description of the model to be fitted.
`data`	a data frame containing the variables specified in `formula`.
`hidden`	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
`threshold`	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
`stepmax`	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
`rep`	the number of repetitions for the neural network's training.
`startweights`	a vector containing starting values for the weights. The weights will not be randomly initialized.
`learningrate.limit`	a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
`learningrate.factor`	a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
`learningrate`	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
`lifesign`	a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.

Neural Network

`lifesign.step`	an integer specifying the stepsize to print the minimal threshold in full lifesign mode.
`algorithm`	a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.
`err.fct`	a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.
`act.fct`	a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.
`linear.output`	logical. If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.
`exclude`	a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.
`constant.weights`	a vector specifying the values of the weights that are excluded from the training process and treated as fix.
`likelihood`	logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of confidence.interval is meaningfull.

Neural Network

```r
library(neuralnet)
iris <- read.csv("iris.data.csv", header=TRUE)
# Prepare iris
set.seed(567)
ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
trainData <- iris[ind==1,]
testData <- iris[ind==2,]
nnet_iristrain <- trainData

#Binarize the categorical output
nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-setosa')
nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-versicolor')
nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-virginica')
names(nnet_iristrain)[6] <- 'Setosa'
names(nnet_iristrain)[7] <- 'Versicolor'
names(nnet_iristrain)[8] <- 'Virginica'

nn <- neuralnet(Setosa+Versicolor+Virginica ~ Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,
                data=nnet_iristrain, hidden=c(3))

plot(nn)
mypredict <- compute(nn, testData[-5])$net.result
# Put multiple binary output to categorical output
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}
idx <- apply(mypredict, c(1), maxidx)
prediction <- c('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')[idx]
table(prediction, testData$Species)
```

```
> table(prediction, testData$Species)

prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              13              0
  Iris-virginica            0               0             15
>
```

Accuracy

```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              12              0
  Iris-virginica            0               1             15

Overall Statistics

               Accuracy : 0.9767442
                 95% CI : (0.8771095, 0.9994114)
    No Information Rate : 0.3488372
    P-Value [Acc > NIR] : < 0.00000000000000022204

                  Kappa : 0.9649837
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                   1.0000000             0.9230769            1.0000000
Specificity                   1.0000000             1.0000000            0.9642857
Pos Pred Value                1.0000000             1.0000000            0.9375000
Neg Pred Value                1.0000000             0.9677419            1.0000000
Prevalence                    0.3488372             0.3023256            0.3488372
Detection Rate                0.3488372             0.2790698            0.3488372
Detection Prevalence          0.3488372             0.2790698            0.3720930
Balanced Accuracy             1.0000000             0.9615385            0.9821429
```

Workshop 4.8 - Neural Networks

1. Use data from Workshop 4.2
2. Build Neural Networks Model to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- **Ensemble methods: Bagging and Boosting**
- Summary

Combining classifiers

- So far, we have only discussed individual classifiers, i.e., how to build them and use them.
- Can we combine multiple classifiers to produce a better classifier?
- Yes, sometimes
- We discuss two main algorithms:
  - **Bagging**
  - **Boosting**

Bagging

- Breiman, 1996
- Bootstrap Aggregating = Bagging
  - Application of bootstrap sampling
    - Given: set D containing m training examples
    - Create a sample S[i] of D by drawing m examples at random with replacement from D
    - S[i] of size m: expected to leave out 0.37 of examples from D

Bagging (cont…)

- Training
  - Create k bootstrap samples S[1], S[2], …, S[k]
  - Build a distinct classifier on each S[i] to produce k classifiers, using the same learning algorithm.
- Testing
  - Classify each new instance by voting of the k classifiers (equal weights)

Bagging Example

Original	1	2	3	4	5	6	7	8
Training set 1	2	7	8	3	7	6	3	1
Training set 2	7	8	5	6	4	2	7	1
Training set 3	3	6	2	7	5	6	2	2
Training set 4	4	5	1	4	6	4	3	8

Bagging (cont …)

When does it help?

- When learner is unstable
  - Small change to training set causes large change in the output classifier
  - True for decision trees, neural networks; not true for *k*-nearest neighbor, naïve Bayesian, class association rules
- Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners

Bagging Predictors, Leo Breiman, 1996

Boosting

- A family of methods:
  - We only study **AdaBoost** (Freund & Schapire, 1996)
- Training
  - Produce a sequence of classifiers (the same base learner)
  - Each classifier is dependent on the previous one, and focuses on the previous one's errors
  - Examples that are incorrectly predicted in previous classifiers are given higher weights
- Testing
  - For a test case, the results of the series of classifiers are combined to determine the final class of the test case.

AdaBoost

Weighted training set

called a weaker classifier

(x_1, y_1, w_1)
(x_2, y_2, w_2)
...
(x_n, y_n, w_n)

Non-negative weights sum to 1

- Build a classifier $\mathbf{h_t}$ whose accuracy on training set > ½ (better than random)

Change weights

AdaBoost algorithm

Algorithm AdaBoost.M1

Input: sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$
with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, \ldots, T$:

1. Call **WeakLearn**, providing it with the distribution D_t.
2. Get back a hypothesis $h_t : X \to Y$.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$.

   If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t:

   $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}.$$

Does AdaBoost always work?

- The actual performance of boosting depends on the data and the base learner.
  - It requires the base learner to be unstable as bagging.
- Boosting seems to be susceptible to noise.
  - When the number of outliners is very large, the emphasis placed on the hard examples can hurt the performance.

Ensemble in R

Fundamental Data Science for Data Scientist

Ensemble : Bagging

Random Forest

-

At each node:
choose some small subset of variables at random
find a variable (and a value for that variable) which optimizes the split

Random Forest

Here is how such a system is trained; for some number of trees T:

1) Sample N cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.

2) At each node:

a) For some number m (see below), m predictor variables are selected at random from all the predictor variables.

b) The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.

c) At the next node, choose another m variables at random from all predictor variables and do the same.

Bagging

```
> library(randomForest)
#Train 100 trees, random selected attributes
> model <- randomForest(Species~., data=iristrain, nTree=500)
#Predict using the forest
> prediction <- predict(model, newdata=iristest, type='class')
> table(prediction, iristest$Species)
> importance(model)
             MeanDecreaseGini
Sepal.Length         7.807602
Sepal.Width          1.677239
Petal.Length        31.145822
Petal.Width         38.617223
```

```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              13              2
  Iris-virginica            0               0             13

Overall Statistics

               Accuracy : 0.9535
                 95% CI : (0.8419, 0.9943)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9303
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor
Sensitivity                      1.0000                 1.0000
Specificity                      1.0000                 0.9333
Pos Pred Value                   1.0000                 0.8667
Neg Pred Value                   1.0000                 1.0000
Prevalence                       0.3488                 0.3023
Detection Rate                   0.3488                 0.3023
Detection Prevalence             0.3488                 0.3488
Balanced Accuracy                1.0000                 0.9667
                     Class: Iris-virginica
Sensitivity                         0.8667
Specificity                         1.0000
Pos Pred Value                      1.0000
Neg Pred Value                      0.9333
Prevalence                          0.3488
Detection Rate                      0.3023
Detection Prevalence                0.3023
Balanced Accuracy                   0.9333
```

Boosting

```
> library(adabag)
> iris.adaboost <- boosting(Species~., data=trainData, boost=TRUE, mfinal=5)
> iris.adaboost$class
> table(iris.adaboost$class, trainData$Species)
> prediction <- predict(iris.adaboost, newdata=testData)
> table(prediction$class, testData$Species)
> confusionMatrix(as.factor(prediction$class), testData$Species)
```

```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              13              1
  Iris-virginica            0               0             14

Overall Statistics

               Accuracy : 0.9767
                 95% CI : (0.8771, 0.9994)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9651
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor
Sensitivity                      1.0000                 1.0000
Specificity                      1.0000                 0.9667
Pos Pred Value                   1.0000                 0.9286
Neg Pred Value                   1.0000                 1.0000
Prevalence                       0.3488                 0.3023
Detection Rate                   0.3488                 0.3023
Detection Prevalence             0.3488                 0.3256
Balanced Accuracy                1.0000                 0.9833
                     Class: Iris-virginica
Sensitivity                         0.9333
Specificity                         1.0000
Pos Pred Value                      1.0000
Neg Pred Value                      0.9655
Prevalence                          0.3488
Detection Rate                      0.3256
Detection Prevalence                0.3256
Balanced Accuracy                   0.9667
```

Workshop 4.9 - Ensemble Method

1. Use data from Workshop 4.2
2. Train your model using either Bagging (Random Forest) or Boosting (AdaBoost) to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Machine Learning Summary

- Applications of supervised learning are in almost any field or domain.
- We studied 8 classification techniques.
- There are still many other methods, e.g.,
    - Bayesian networks
    - Neural networks
    - Genetic algorithms
    - Fuzzy classification
    - This large number of methods also show the importance of classification and its wide applicability.
- It remains to be an active research area.

Unsupervised Learning

Fundamental Data Science for Data Scientist

Supervised learning vs. unsupervised learning

Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.

These patterns are then utilized to predict the values of the target attribute in future data instances.

Unsupervised learning: The data have no target attribute.

We want to explore the data to find some intrinsic structures in them.

Clustering

- Clustering is a technique for finding similarity groups in data, called **clusters**. I.e.,
  - it groups data instances that are similar to (near) each other in one cluster and data instances that are very different (far away) from each other into different clusters.
- Clustering is often called an **unsupervised learning** task as no class values denoting an *a priori* grouping of the data instances are given, which is the case in supervised learning.
- Due to historical reasons, clustering is often considered synonymous with unsupervised learning.
  - In fact, association rule mining is also unsupervised
- This chapter focuses on clustering.

An illustration

The data set has three natural groups of data points, i.e., 3 natural clusters.

What is clustering for?

- Let us see some real-life examples
- Example 1: groups people of similar sizes together to make “small”, “medium” and “large” T-Shirts.
    - Tailor-made for each person: too expensive
    - One-size-fits-all: does not fit all.
- Example 2: In marketing, segment customers according to their similarities
    - To do targeted marketing.

What is clustering for? (cont…)

Example 3: Given a collection of text documents, we want to organize them according to their content similarities,

To produce a topic hierarchy

In fact, clustering is one of the most utilized data mining techniques.

It has a long history, and used in almost every field, e.g., medicine, psychology, botany, sociology, biology, archeology, marketing, insurance, libraries, etc.

In recent years, due to the rapid increase of online documents, text clustering becomes important.

Aspects of clustering

- A clustering algorithm
  - Partitional clustering
  - Hierarchical clustering
  - …
- A distance (similarity, or dissimilarity) function
- Clustering quality
  - Inter-clusters distance ⇒ maximized
  - Intra-clusters distance ⇒ minimized
- The quality of a clustering result depends on the algorithm, the

K-means clustering

K-means is a partitional clustering algorithm

Let the set of data points (or instances) D be

$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$,

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ir})$ is a vector in a real-valued space $X \subseteq R^r$, and r

is the number of attributes (dimensions) in the data.

The k-means algorithm partitions the given data into k clusters.

Each cluster has a cluster **center**, called **centroid**.

k is specified by the user

K-means algorithm

Given k, the k-means algorithm works as follows:

1) Randomly choose k data points (seeds) to be the initial centroids, cluster centers

2) Assign each data point to the closest centroid

3) Re-compute the centroids using the current cluster memberships.

4) If a convergence criterion is not met, go to 2).

K-means algorithm – (cont …)

```
Algorithm k-means(k, D)
  Choose k data points as the initial centroids (cluster centers)
  repeat
      for each data point x ∈ D do
          compute the distance from x to each centroid;
          assign x to the closest centroid          // a centroid represents a cluster
      endfor
      re-compute the centroids using the current cluster memberships
  until the stopping criterion is met
```

Stopping/convergence criterion

1. no (or minimum) re-assignments of data points to different clusters,
2. no (or minimum) change of centroids, or
3. minimum decrease in the **sum of squared error** (SSE),

C_i is the jth cluster, $\mathbf{m}_j$ is the centroid of cluster C_j (the mean vector of all the data points in C_j), and dist($\mathbf{x}$, $\mathbf{m}_j$) is the distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$.

$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \qquad (1)$$

An example

(A). Random selection of k centers

Iteration 1: (B). Cluster assignment

(C). Re-compute centroids

An example (cont …)

Iteration 2: (D). Cluster assignment

(E). Re-compute centroids

Iteration 3: (F). Cluster assignment

(G). Re-compute centroids

An example distance function

The k-means algorithm can be used for any application data set where the **mean** can be defined and computed. In the **Euclidean space**, the mean of a cluster is computed with:

$$\mathbf{m}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \tag{2}$$

where $|C_j|$ is the number of data points in cluster C_j. The distance from one data point $\mathbf{x}_i$ to a mean (centroid) $\mathbf{m}_j$ is computed with

$$\begin{aligned} dist(\mathbf{x}_i, \mathbf{m}_j) &= \| \mathbf{x}_i - \mathbf{m}_j \| \\ &= \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \ldots + (x_{ir} - m_{jr})^2} \end{aligned} \tag{3}$$

A disk version of k-means

- K-means can be implemented with data on disk
  - In each iteration, it scans the data once.
  - as the centroids can be computed incrementally
- It can be used to cluster large datasets that do not fit in main memory
- We need to control the number of iterations
  - In practice, a limited is set (< 50).
- Not the best method. There are other scale-up algorithms, e.g., BIRCH (balanced iterative reducing and clustering using hierarchies).

A disk version of k-means (cont...)

```
Algorithm disk-k-means(k, D)
  Choose k data points as the initial centriods m_j, j = 1, ..., k;
  repeat
      initialize s_j = 0, j = 1, ..., k;          // 0 is a vector with all 0's
      initialize n_j = 0, j = 1, ..., k;          // n_j is the number points in cluster j
      for each data point x ∈ D do
          j = arg min_j dist(x, m_j);
          assign x to the cluster j;
          s_j = s_j + x;
          n_j = n_j + 1;
      endfor
      m_i = s_j/n_j, i = 1, ..., k;
  until the stopping criterion is met
```

Strengths of k-means

Strengths:

Simple: easy to understand and to implement

Efficient: Time complexity: $O(tkn)$,

where n is the number of data points,

k is the number of clusters, and

t is the number of iterations.

Since both k and t are small. k-means is considered a linear algorithm.

K-means is the most popular clustering algorithm.

Note that: it terminates at a local optimum if SSE is used. The global optimum is hard to find due to complexity.

Weaknesses of k-means

The algorithm is only applicable if the mean is defined.

For categorical data, ***k*-mode** - the centroid is represented by most frequent values.

The user needs to specify *k*.

The algorithm is sensitive to **outliers**

- Outliers are data points that are very far away from other data points.
- Outliers could be errors in the data recording or some special data points with very different values.

Weaknesses of k-means: Problems with outliers

(A): Undesirable clusters

(B): Ideal clusters

Weaknesses of k-means: To deal with outliers

One method is to **remove some data points** in the clustering process that are much further away from the centroids than other data points.

To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.

Another method is to **perform random sampling**. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.

- Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

Weaknesses of k-means (cont …)

(A). Random selection of seeds (centroids)

(B). Iteration 1

(C). Iteration 2

Weaknesses of k-means (cont …)

(A). Random selection of k seeds (centroids)

(B). Iteration 1

(C). Iteration 2

- There are some methods to help choose good seeds

Weaknesses of k-means (cont …)

The *k*-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).

(A): Two natural clusters

(B): *k*-means clusters

K-means summary

- Despite weaknesses, k-means is still the most popular algorithm due to its simplicity, efficiency and
  - other clustering algorithms have their own lists of weaknesses.
- No clear evidence that any other clustering algorithm performs better in general
  - although they may be more suitable for some specific types of data or applications.
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!

Common ways to represent clusters

Use the centroid of each cluster to represent the cluster.

- compute the radius and
- standard deviation of the cluster to determine its spread in each dimension

- The centroid representation alone works well if the clusters are of the hyper-spherical shape.
- If clusters are elongated or are of other shapes, centroids are not sufficient

Using classification model

All the data points in a cluster are regarded to have the same class label, e.g., the cluster ID.

run a supervised learning algorithm on the data to find a classification model.

$x \le 2 \rightarrow$ cluster 1

$x > 2,\ y > 1.5 \rightarrow$ cluster 2

$x > 2,\ y \le 1.5 \rightarrow$ cluster 3

Use frequent values to represent cluster

- This method is mainly for clustering of categorical data (e.g., *k*-modes clustering).
- Main method used in text clustering, where a small set of frequent words in each cluster is selected to represent the cluster.

Clusters of arbitrary shapes

Hyper-elliptical and hyper-spherical clusters are usually easy to represent, using their centroid together with spreads.

Irregular shape clusters are hard to represent. They may not be useful in some applications.

- Using centroids are not suitable (upper figure) in general
- K-means clusters may be more useful (lower figure), e.g., for making 2 size T-shirts.

k-Mean in R

Fundamental Data Science for Data Scientist

K-Means Clustering

1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

```
library(stats)
set.seed(101)
km <- kmeans(iris[,1:4], 3)
plot(iris[,1], iris[,2], col=km$cluster)
points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)
```

table(km$cluster, iris$Species)

```
   setosa versicolor virginica
1       0         48        14
2      50          0         0
3       0          2        36
>
```

Another round

```
set.seed(900)
km <- kmeans(iris[,1:4], 3)
plot(iris[,1], iris[,2], col=km$cluster)
points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)
```

table(km$cluster, iris$Species)

```
   setosa versicolor virginica
1       0         46        50
2      17          4         0
3      33          0         0
>
```

Workshop 4.10 - k-Mean Clustering

1. Use data from Workshop 4.2
2. Perform k-Mean clustering to divide data into k groups

Hierarchical Clustering

Produce a nested sequence of clusters, a tree, also called Dendrogram.

Types of hierarchical clustering

Agglomerative (bottom up) clustering: It builds the dendrogram (tree) from the bottom level, and

- merges the most similar (or nearest) pair of clusters
- stops when all the data points are merged into a single cluster (i.e., the root cluster).

Divisive (top down) clustering: It starts with all data points in one cluster, the root.

- Splits the root into a set of child clusters. Each child cluster is recursively divided further
- stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point

Agglomerative clustering

- It is more popular then divisive methods.
- At the beginning, each data point forms a cluster (also called a node).
- Merge nodes/clusters that have the least distance.
- Go on merging
- Eventually all nodes belong to one cluster

Agglomerative clustering algorithm

Algorithm Agglomerative(D)

Make each data point in the data set D a cluster,
Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in D$;
repeat
    find two clusters that are nearest to each other;
    merge the two clusters form a new cluster c;
    compute the distance from c to all other clusters;
until there is only one cluster left

An example: working of the algorithm

(A). Nested clusters (B) Dendrogram

Measuring the distance of two clusters

- A few ways to measure distances of two clusters.
- Results in different variations of the algorithm.
  - Single link
  - Complete link
  - Average link
  - Centroids
  - …

Single link method

The distance between two clusters is the distance between two closest data points in the two clusters, one data point from each cluster.

It can find arbitrarily shaped clusters, but

- It may cause the undesirable "chain effect" by noisy points

Two natural clusters are split into two

Complete link method

- The distance between two clusters is the distance of two furthest data points in the two clusters.
- It is sensitive to outliers because they are far away

Average link and centroid methods

Average link: A compromise between

- the sensitivity of complete-link clustering to outliers and
- the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects.

  In this method, the distance between two clusters is the average distance of all pair-wise distances between the data points in two clusters.

Centroid method: In this method, the distance between two clusters is the distance between their centroids

The complexity

- All the algorithms are at least $O(n^2)$. n is the number of data points.
- Single link can be done in $O(n^2)$.
- Complete and average links can be done in $O(n^2 \log n)$.
- Due the complexity, hard to use for large data sets.
  - Sampling
  - Scale-up methods (e.g., BIRCH).

Distance functions

- Key to clustering. “similarity” and “dissimilarity” can also commonly used terms.
- There are numerous distance functions for
  - Different types of data
    - Numeric data
    - Nominal data
  - Different specific applications

Distance functions for numeric attributes

Most commonly used functions are

Euclidean distance and

Manhattan (city block) distance

We denote distance with: $dist(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are data points (vectors)

They are special cases of Minkowski distance. h is positive integer.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \left((x_{i1} - x_{j1})^h + (x_{i2} - x_{j2})^h + \ldots + (x_{ir} - x_{jr})^h\right)^{\frac{1}{h}}$$

Euclidean distance and Manhattan distance

If h = 2, it is the **Euclidean distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ir} - x_{jr})^2}$$

If h = 1, it is the **Manhattan distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = | x_{i1} - x_{j1} | + | x_{i2} - x_{j2} | + \ldots + | x_{ir} - x_{jr} |$$

Weighted Euclidean distance

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \ldots + w_r(x_{ir} - x_{jr})^2}$$

Squared distance and Chebychev distance

Squared Euclidean distance: to place progressively greater weight on data points that are further apart.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ir} - x_{jr})^2$$

Chebychev distance: one wants to define two data points as "different" if they are different on any one of the attributes.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \ldots, |x_{ir} - x_{jr}|)$$

Distance functions for binary and nominal attributes

Binary attribute: has two values or states but no ordering relationships, e.g.,

Gender: male and female.

We use a confusion matrix to introduce the distance functions/measures.

Let the ith and j^{th} data points be $\mathbf{x}_i$ and $\mathbf{x}_j$ (vectors)

Confusion matrix

$$
\begin{array}{cc|cc|c}
 & & \multicolumn{2}{c}{\text{Data point } j} & \\
 & & 1 & 0 & \\
\hline
\text{Data point } i & 1 & a & b & a+b \\
 & 0 & c & d & c+d \\
\hline
 & & a+c & b+d & a+b+c+d
\end{array}
\tag{10}
$$

a: the number of attributes with the value of 1 for both data points.

b: the number of attributes for which $x_{if} = 1$ and $x_{jf} = 0$, where x_{if} (x_{jf}) is the value of the fth attribute of the data point $\mathbf{x}_i$ ($\mathbf{x}_j$).

c: the number of attributes for which $x_{if} = 0$ and $x_{jf} = 1$.

d: the number of attributes with the value of 0 for both data points.

Symmetric binary attributes

A binary attribute is **symmetric** if both of its states (0 and 1) have equal importance, and carry the same weights, e.g., male and female of the attribute Gender

Distance function: Simple Matching Coefficient, proportion of mismatches of their values

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b + c}{a + b + c + d}$$

Symmetric binary attributes: example

$\mathbf{x}_1$	1	1	1	0	1	0	0
$\mathbf{x}_2$	0	1	1	0	0	1	0

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{2+1}{2+2+1+2} = \frac{3}{7} = 0.429$$

Asymmetric binary attributes

Asymmetric: if one of the states is more important or more valuable than the other.

By convention, state 1 represents the more important state, which is typically the rare or infrequent state.

Jaccard coefficient is a popular measure

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b + c}{a + b + c}$$

We can have some variations, adding weights

Nominal attributes

Nominal attributes: with more than two states or values.

the commonly used distance measure is also based on the simple matching method.

Given two data points $\mathbf{x}_i$ and $\mathbf{x}_j$, let the number of attributes be r, and the number of values that match in $\mathbf{x}_i$ and $\mathbf{x}_j$ be q.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{r - q}{r}$$

Distance function for text documents

- A text document consists of a sequence of sentences and each sentence consists of a sequence of words.
- To simplify: a document is usually considered a “bag” of words in document clustering.
    - Sequence and position of words are ignored.
- A document is represented with a vector just like a normal data point.
- It is common to use similarity to compare two documents rather than distance.
    - The most commonly used similarity function is the **cosine similarity**.

Data standardization

- In the Euclidean space, standardization of attributes is recommended so that all attributes can have equal impact on the computation of distances.
- Consider the following pair of data points
  - $\mathbf{x}_i$: (0.1, 20) and $\mathbf{x}_j$: (0.9, 720).

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(0.9 - 0.1)^2 + (720 - 20)^2} = 700.000457,$$

- The distance is almost completely dominated by (720-20) = 700.
- Standardize attributes: to force the attributes to have a common value range

Interval-scaled attributes

- Their values are real numbers following a linear scale.
  - The difference in Age between 10 and 20 is the same as that between 40 and 50.
  - The key idea is that intervals keep the same importance through out the scale
- Two main approaches to standardize interval scaled attributes, **range** and **z-score**. *f* is an attribute

$$range(x_{if}) = \frac{x_{if} - \min(f)}{\max(f) - \min(f)},$$

Interval-scaled attributes (cont …)

Z-score: transforms the attribute values so that they have a mean of zero and a **mean absolute deviation** of 1. The mean absolute deviation of attribute *f*, denoted by s_f, is computed as follows

$$s_f = \frac{1}{n}\left(|x_{1f} - m_f| + |x_{2f} - m_f| + \ldots + |x_{nf} - m_f|\right)$$

$$m_f = \frac{1}{n}\left(x_{1f} + x_{2f} + \ldots + x_{nf}\right)$$

Z-score: $$z(x_{if}) = \frac{x_{if} - m_f}{s_f}.$$

Ratio-scaled attributes

Numeric attributes, but unlike interval-scaled attributes, their scales are exponential,

For example, the total amount of microorganisms that evolve in a time *t* is approximately given by

$$Ae^{Bt},$$

where *A* and *B* are some positive constants.

Do log transform:

$$\log(x_{if})$$

Then treat it as an interval-scaled attribuete

Nominal attributes

- Sometime, we need to transform nominal attributes to numeric attributes.
- Transform nominal attributes to binary attributes.
  - The number of values of a nominal attribute is *v*.
  - Create *v* binary attributes to represent them.
  - If a data instance for the nominal attribute takes a particular value, the value of its binary attribute is set to 1, otherwise it is set to 0.
- The resulting binary attributes can be used as numeric attributes, with two values, 0 and 1.

Nominal attributes: an example

- Nominal attribute *fruit*: has three values,
  - Apple, Orange, and Pear
- We create three binary attributes called, Apple, Orange, and Pear in the new data.
- If a particular data instance in the original data has Apple as the value for *fruit*,
  - then in the transformed data, we set the value of the attribute Apple to 1, and
  - the values of attributes Orange and Pear to 0

Ordinal attributes

- Ordinal attribute: an ordinal attribute is like a nominal attribute, but its values have a numerical ordering. E.g.,
  - Age attribute with values: Young, MiddleAge and Old. They are ordered.
  - Common approach to standardization: treat as an interval-scaled attribute.

Mixed attributes

- Our distance functions given are for data with all numeric attributes, or all nominal attributes, etc.
- Practical data has different types:
  - Any subset of the 6 types of attributes,
  - **interval-scaled**,
  - **symmetric binary**,
  - **asymmetric binary**,
  - **ratio-scaled**,
  - **ordinal** and
  - **nominal**

Convert to a single type

- One common way of dealing with mixed attributes is to
  - Decide the dominant attribute type, and
  - Convert the other types to this type.
- E.g, if most attributes in a data set are interval-scaled,
  - we convert ordinal attributes and ratio-scaled attributes to interval-scaled attributes.
  - It is also appropriate to treat symmetric binary attributes as interval-scaled attributes.

Convert to a single type (cont …)

- It does not make much sense to convert a nominal attribute or an asymmetric binary attribute to an interval-scaled attribute,
  - but it is still frequently done in practice by assigning some numbers to them according to some hidden ordering, e.g., prices of the fruits
- Alternatively, a nominal attribute can be converted to a set of (symmetric) binary attributes, which are then treated as numeric attributes.

Combining individual distances

This approach computes individual attribute distances and then combine them.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{f=1}^{r} \delta_{ij}^{f} d_{ij}^{f}}{\sum_{f=1}^{r} \delta_{ij}^{f}}$$

This distance value is between 0 and 1. r is the number of attributes in the data set. The indicator δ_{ij}^{f} is 1 when both values x_{if} and x_{jf} for attribute f are non-missing, and it is set to 0 otherwise. It is also set to 0 if attribute f is asymmetric and the match is 0-0. Equation (25) cannot be computed if all δ_{ij}^{f} 's are 0. In such a case, some default value may be used or one of the data points is removed.

d_{ij}^{f} is the distance contributed by attribute f, and it is in the 0-1 range.

Hierarchical Clustering in R

Fundamental Data Science for Data Scientist

Hierarchical Clustering

Compute distance between every pairs of point/cluster.

(a) Distance between point is just using the distance function.

(b) Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB).

(c) Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.

Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains

```
set.seed(101)
sampleiris <- iris[sample(1:150, 40),] # get samples from iris dataset
# each observation has 4 variables, ie, they are interpreted as 4-D points
distance   <- dist(sampleiris[,-5], method="euclidean")
cluster    <- hclust(distance, method="average")
plot(cluster, hang=-1, label=sampleiris$Species)
```

Cluster Dendrogram
Height
0
1
2
3
4
setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
virginica virginica virginica virginica virginica virginica virginica
versicolor versicolor versicolor virginica virginica virginica versicolor virginica virginica
versicolor versicolor versicolor

It's possible to prune the result tree.

```
par(mfrow=c(1,2))
group.3 <- cutree(cluster, k = 3)  # prune the tree by 3 clusters
table(group.3, sampleiris$Species) # compare with known classes
```

```
group.3 setosa versicolor virginica
      1      0         15         9
      2     13          0         0
      3      0          0         3
>
```

```
plot(sampleiris[,c(1,2)], col=group.3, pch=19, cex=2.5, main="3 clusters")
points(sampleiris[,c(1,2)], col=sampleiris$Species, pch=19, cex=1)
```

Workshop 4.11 - Hierarchical Clustering

1. Use data from Workshop 4.2
2. Perform Hierachical clustering to divide data into 4 groups

How to choose a clustering algorithm

- Clustering research has a long history. A vast collection of algorithms are available.
  - We only introduced several main algorithms.
- Choosing the “best” algorithm is a challenge.
  - Every algorithm has limitations and works well with certain data distributions.
  - It is very hard, if not impossible, to know what distribution the application data follow. The data may not fully follow any “ideal” structure or distribution required by the algorithms.
  - One also needs to decide how to standardize the data, to choose a suitable distance function and to select other parameter values.

Choose a clustering algorithm (cont ...)

- Due to these complexities, the common practice is to
  - run several algorithms using different distance functions and parameter settings, and
  - then carefully analyze and compare the results.
- The interpretation of the results must be based on insight into the meaning of the original data together with knowledge of the algorithms used.
- Clustering is highly application dependent and to certain extent subjective (personal preferences).

Cluster Evaluation: hard problem

The quality of a clustering is very hard to evaluate because

We do not know the correct clusters

Some methods are used:

User inspection

- Study centroids, and spreads
- Rules from a decision tree.
- For text documents, one can read some documents in clusters.

Cluster evaluation: ground truth

We use some labeled data (for classification)

Assumption: Each class is a cluster.

After clustering, a confusion matrix is constructed. From the matrix, we compute various measurements, **entropy, purity, precision, recall and F-score.**

Let the classes in the data D be $C = (c_1, c_2, ..., c_k)$. The clustering method produces k clusters, which divides D into k disjoint subsets, $D_1, D_2, ..., D_k$.

Evaluation measures: Entropy

Entropy: For each cluster, we can measure its entropy as follows:

$$entropy(D_i) = -\sum_{j=1}^{k} \Pr_i(c_j) \log_2 \Pr_i(c_j), \tag{29}$$

where $\Pr_i(c_j)$ is the proportion of class c_j data points in cluster i or D_i. The total entropy of the whole clustering (which considers all clusters) is

$$entropy_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times entropy(D_i) \tag{30}$$

Evaluation measures: purity

Purity: This again measures the extent that a cluster contains only one class of data. The purity of each cluster is computed with

$$purity(D_i) = \max_j(\Pr_i(c_j)) \tag{31}$$

The total purity of the whole clustering (considering all clusters) is

$$purity_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times purity(D_i) \tag{32}$$

An example

Example 14: Assume we have a text collection D of 900 documents from three topics (or three classes), Science, Sports, and Politics. Each class has 300 documents. Each document in D is labeled with one of the topics (classes). We use this collection to perform clustering to find three clusters. Note that class/topic labels are not used in clustering. After clustering, we want to measure the effectiveness of the clustering algorithm.

Cluster	Science	Sports	Politics		Entropy	Purity
1	250	20	10		0.589	0.893
2	20	180	80		1.198	0.643
3	30	100	210		1.257	0.617
Total	300	300	300		1.031	0.711

A remark about ground truth evaluation

Commonly used to compare different clustering algorithms.

A real-life data set for clustering has no class labels.

- Thus although an algorithm may perform very well on some labeled data sets, no guarantee that it will perform well on the actual application data at hand.

The fact that it performs well on some label data sets does give us some confidence of the quality of the algorithm.

This evaluation method is said to be based on **external data** or information.

Evaluation based on internal information

Intra-cluster cohesion (compactness):

Cohesion measures how near the data points in a cluster are to the cluster centroid.

Sum of squared error (SSE) is a commonly used measure.

I**nter-cluster separation** (isolation):

Separation means that different cluster centroids should be far away from one another.

In most applications, expert judgments are still the key.

Indirect evaluation

- In some applications, clustering is not the primary task, but used to help perform another task.
- We can use the performance on the primary task to compare clustering methods.
- For instance, in an application, the primary task is to provide recommendations on book purchasing to online shoppers.
  - If we can cluster books according to their features, we might be able to provide better recommendations.
  - We can evaluate different clustering algorithms based on how well they help with the recommendation task.
  - Here, we assume that the recommendation can be reliably evaluated.

Supervised learning for unsupervised learning

- Decision tree algorithm is not directly applicable.
  - it needs at least two classes of data.
  - A clustering data set has no class label for each data point.
- The problem can be dealt with by a simple idea.
  - Regard each point in the data set to have a class label *Y*.
  - Assume that the data space is uniformly distributed with another type of points, called **non-existing points**. We give them the class, *N*.
- With the *N* points added, the problem of partitioning the data space into data and empty regions becomes a supervised classification problem.

An example

(A): The original data space

(B). Partitioning with added N points

- A decision tree method is used for partitioning in (B).

Can it done without adding N points?

Yes.

Physically adding N points increases the size of the data and thus the running time.

More importantly: it is unlikely that we can have points truly uniformly distributed in a high dimensional space as we would need an exponential number of points.

Fortunately, no need to physically add any N points.

We can compute them when needed

Characteristics of the approach

It provides representations of the resulting data and empty regions in terms of hyper-rectangles, or rules.

It detects outliers automatically. Outliers are data points in an empty region.

It may not use all attributes in the data just as in a normal decision tree for supervised learning.

It can automatically determine what attributes are useful. Subspace clustering ...

Drawback: data regions of irregular shapes are hard to handle since decision tree learning only generates hyper-rectangles (formed by axis-parallel hyper-planes), which are rules.

Building the Tree

The main computation in decision tree building is to evaluate entropy (for information gain):

$$entropy(D) = -\sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

Can it be evaluated without adding *N* points? Yes.

Pr(c_j) is the probability of class c_j in data set *D*, and |*C*| is the number of classes, *Y* and *N* (2 classes).

To compute Pr(c_j), we only need the number of *Y* (data) points and the number of *N* (non-existing) points.

We already have *Y* (or data) points, and we can compute the number of *N* points on the fly.

Simple: as we assume that the *N* points are uniformly distributed in the space.

An example

The space has 25 data (*Y*) points and 25 *N* points. Assume the system is evaluating a possible cut *S*.

N points on the left of *S* is 25 * 4/10 = 10. The number of *Y* points is 3.

Likewise, # *N* points on the right of *S* is 15 (= 25 - 10).The number of *Y* points is 22.

With these numbers, entropy can be computed.

How many N points to add?

We add a different number of *N* points at each different node.

The number of *N* points for the current node *E* is determined by the following rule (note that at the root node, the number of inherited *N* points is 0):

If the number of N points inherited from the parent node of E is less than the number of Y points in E **then**
    the number of N points for E is increased to the number of Y points in E
else the number of inherited N points is used for E

An example

Example 17: Fig. 20 gives an example. The (parent) node P has two children nodes L and R. Assume P has 1000 Y points and thus 1000 N points, stored in $P.Y$ and $P.N$ respectively. Assume after splitting, L has 20 Y points and 500 N points, and R has 980 Y points and 500 N points. According to the above rule, for subsequent partitioning, we increase the number of N points at R to 980. The number of N points at L is unchanged.

How many N points to add? (cont...)

Basically, for a *Y* node (which has more data points), we increase *N* points so that

#Y = #N

The number of *N* points is not reduced if the current node is an *N* node (an *N* node has more *N* points than *Y* points).

A reduction may cause outlier *Y* points to form *Y* nodes (a *Y* node has an equal number of *Y* points as *N* points or more).

Then data regions and empty regions may not be separated well.

Building the decision tree

Using the above ideas, a decision tree can be built to separate data regions and empty regions.

The actual method is more sophisticated as a few other tricky issues need to be handled in

tree building and

tree pruning.

Summary

- Clustering is has along history and still active
  - There are a huge number of clustering algorithms
  - More are still coming every year.
- We only introduced several main algorithms. There are many others, e.g.,
  - density based algorithm, sub-space clustering, scale-up methods, neural networks based methods, fuzzy clustering, co-clustering, etc.
- Clustering is hard to evaluate, but very useful in practice. This partially explains why there are still a large number of clustering algorithms being devised every year.
- Clustering is highly application dependent and to some extent subjective.

Recommendation System

- Concept
- Building Recommendation System

Recommendation System Topics

- Basic concepts
- Apriori algorithm
- Summary

Association rule mining

- Proposed by **Agrawal et al in 1993**.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for **Market Basket Analysis** to find how items purchased by customers are related.

Bread → Milk [sup = 5%, conf = 100%]

The model: data

$I = \{i_1, i_2, \ldots, i_m\}$: a set of *items*.

Transaction t :

t a set of items, and $t \subseteq I$.

Transaction Database T: a set of transactions $T = \{t_1, t_2, \ldots, t_n\}$.

Transaction data: supermarket data

Market basket transactions:

t1: {bread, cheese, milk}
t2: {apple, eggs, salt, yogurt}
… …
tn: {biscuit, eggs, milk}

Concepts:

An *item*: an item/article in a basket
I: the set of all items sold in the store
A *transaction*: items purchased in a basket; it may have TID (transaction ID)
A *transactional dataset*: A set of transactions

Transaction data: a set of documents

A text document data set. Each document is treated as a "bag" of keywords

doc1: Student, Teach, School

doc2: Student, School

doc3: Teach, School, City, Game

doc4: Baseball, Basketball

doc5: Basketball, Player, Spectator

doc6: Baseball, Coach, Game, Team

doc7: Basketball, Team, City, Game

The model: rules

A transaction *t* contains *X*, a set of items (itemset) in *I*, if $X \subseteq t$.

An association rule is an implication of the form:

$$X \rightarrow Y, \text{ where } X, Y \subset I, \text{ and } X \cap Y = \varnothing$$

An itemset is a set of items.

E.g., X = {milk, bread, cereal} is an itemset.

A *k*-itemset is an itemset with *k* items.

E.g., {milk, bread, cereal} is a 3-itemset

Rule strength measures

Support: The rule holds with support *sup* in *T* (the transaction data set) if sup% of transactions contain $X \cup Y$.

$$sup = \Pr(X \cup Y).$$

Confidence: The rule holds in *T* with confidence *conf* if *conf*% of tranactions that contain *X* also contain *Y*.

$$conf = \Pr(Y \mid X)$$

An association rule is a pattern that states when *X* occurs, *Y* occurs with certain probability.

Support and Confidence

Support count: The support count of an itemset X, denoted by $X.count$, in a data set T is the number of transactions in T that contain X. Assume T has n transactions.

Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal and key features

Goal: Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).

Key Features

Completeness: find all rules.

No target item(s) on the right-hand-side

Mining with data on hard disk (not in memory)

An example

t1:	Beef, Chicken, Milk
t2:	Beef, Cheese
t3:	Cheese, Boots
t4:	Beef, Chicken, Cheese
t5:	Beef, Chicken, Clothes, Cheese, Milk
t6:	Chicken, Clothes, Milk
t7:	Chicken, Milk, Clothes

Transaction data

Assume:

minsup = 30%

minconf = 80%

An example frequent *itemset*:

{Chicken, Clothes, Milk} [sup = 3/7]

Association rules from the itemset:

Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

… …

Clothes, Chicken → Milk, [sup = 3/7, conf = 3/3]

Transaction data representation

A simplistic view of shopping baskets,

Some important information not considered. E.g,

the quantity of each item purchased and

the price paid.

Many mining algorithms

There are a large number of them!!

They use different strategies and data structures.

Their resulting sets of rules are all the same.

> Given a transaction data set T, and a minimum support and a minimum confident, the set of association rules existing in T is uniquely determined.

Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.

We study only one: **the Apriori Algorithm**

Recommendation System Topics

- Basic concepts
- Apriori algorithm
- Summary

The Apriori algorithm

Probably the best known algorithm

Two steps:

Find all itemsets that have minimum support (*frequent itemsets*, also called large itemsets).

Use frequent itemsets to generate rules.

E.g., a frequent itemset

{Chicken, Clothes, Milk} [sup = 3/7]

and one rule from the frequent itemset

Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

Step 1: Mining all frequent itemsets

A **frequent *itemset*** is an itemset whose support is ≥ minsup.

Key idea: **The apriori property (downward closure property)**: any subsets of a frequent itemset are also frequent itemsets

The Algorithm

Iterative algo. (also called level-wise search): Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on.

In each iteration k, only consider itemsets that contain some k-1 frequent itemset.

Find frequent itemsets of size 1: F_1

From $k = 2$

C_k = candidates of size k: those itemsets of size k that could be frequent, given F_{k-1}

F_k = those itemsets that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once).

Example – Finding frequent itemsets

Dataset T

minsup=0.5

TID	Items
T100	1, 3, 4
T200	2, 3, 5
T300	1, 2, 3, 5
T400	2, 5

itemset:count

1. scan T → C_1: {1}:2, {2}:3, {3}:3, {4}:1, {5}:3

→ F_1: {1}:2, {2}:3, {3}:3, {5}:3

→ C_2: {1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

2. scan T → C_2: {1,2}:1, {1,3}:2, {1,5}:1, {2,3}:2, {2,5}:3, {3,5}:2

→ F_2: **{1,3}**:2, **{2,3}**:2, **{2,5}:**3, **{3,5}**:2

→ C_3: {2, 3,5}

3. scan T → C_3: **{2, 3, 5}**:2 → F_3: **{2, 3, 5}**

Details: ordering of items

The items in I are sorted in lexicographic order (which is a total order).

The order is used throughout the algorithm in each itemset.

$\{w[1], w[2], \ldots, w[k]\}$ represents a k-itemset w consisting of items $w[1], w[2], \ldots, w[k]$, where $w[1] < w[2] < \ldots < w[k]$ according to the total order.

Details: the algorithm

Algorithm Apriori(*T*)

$C_1 \leftarrow$ init-pass(T);

$F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq minsup\}$; // n: no. of transactions in T

for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**

  $C_k \leftarrow$ candidate-gen(F_{k-1});

  for each transaction $t \in T$ **do**

    for each candidate $c \in C_k$ **do**

      if c is contained in t **then**

        $c.count$++;

    end

  end

  $F_k \leftarrow \{c \in C_k \mid c.count/n \geq minsup\}$

end

return $F \leftarrow \boxed{?}_k F_k$;

Apriori candidate generation

The candidate-gen function takes F_{k-1} and returns a superset (called the candidates) of the set of all frequent k-itemsets. It has two steps

join step: Generate all possible candidate itemsets C_k of length k

prune step: Remove those candidates in C_k that cannot be frequent.

Candidate-gen function

Function candidate-gen($F_{k\text{-}1}$)

$C_k \leftarrow \varnothing$;

forall $f_1, f_2 \in F_{k\text{-}1}$

with $f_1 = \{i_1, \ldots, i_{k\text{-}2}, i_{k\text{-}1}\}$

and $f_2 = \{i_1, \ldots, i_{k\text{-}2}, i'_{k\text{-}1}\}$

and $i_{k\text{-}1} < i'_{k\text{-}1}$ **do**

$c \leftarrow \{i_1, \ldots, i_{k\text{-}1}, i'_{k\text{-}1}\}$; // join f_1 and f_2

$C_k \leftarrow C_k \cup \{c\}$;

for each (k-1)-subset s of c **do**

if ($s \notin F_{k\text{-}1}$) **then**

delete c from C_k; // prune

end

end

return C_k;

An example

F_3 = {{1, 2, 3}, {1, 2, 4}, {1, 3, 4},

{1, 3, 5}, {2, 3, 4}}

After join

C_4 = {{1, 2, 3, 4}, {1, 3, 4, 5}}

After pruning:

C_4 = {{1, 2, 3, 4}}

because {1, 4, 5} is not in F_3 ({1, 3, 4, 5} is removed)

Step 2: Generating rules from frequent itemsets

Frequent itemsets ≠ association rules

One more step is needed to generate association rules

For each frequent itemset *X*,

For each proper nonempty subset *A* of *X*,

Let *B* = X - *A*

A → B is an association rule if

Confidence(A → B) ≥ minconf,

support(A → B) = support(A∪B) = support(X)

confidence(A → B) = support(A ∪ B) / support(A)

Generating rules: an example

Suppose {2,3,4} is frequent, with sup=50%

Proper nonempty subsets: {2,3}, {2,4}, {3,4}, {2}, {3}, {4}, with sup=50%, 50%, 75%, 75%, 75%, 75% respectively

These generate these association rules:

2,3 → 4, confidence=100%

2,4 → 3, confidence=100%

3,4 → 2, confidence=67%

2 → 3,4, confidence=67%

3 → 2,4, confidence=67%

4 → 2,3, confidence=67%

All rules have support = 50%

Generating rules: summary

To recap, in order to obtain $A \rightarrow B$, we need to have support($A \cup B$) and support(A)

All the required information for confidence computation has already been recorded in itemset generation. No need to see the data *T* any more.

This step is not as time-consuming as frequent itemsets generation.

On Apriori Algorithm

Seems to be very expensive

- Level-wise search
- K = the size of the largest itemset
- It makes at most K passes over data
- In practice, K is bounded (10).
- The algorithm is very fast. Under some conditions, all rules can be found in linear time.
- Scale up to large data sets

More on association rule mining

- Clearly the space of all association rules is exponential, $O(2^m)$, where m is the number of items in *I*.
- The mining exploits sparseness of data, and high minimum support and high minimum confidence values.
- Still, it always produces a huge number of rules, thousands, tens of thousands, millions, ...

Summary

- Association rule mining has been extensively studied in the data mining community.
- There are many efficient algorithms and model variations.
- Other related work includes
  - Multi-level or generalized rule mining
  - Constrained rule mining
  - Incremental rule mining
  - Maximal frequent itemset mining
  - Numeric association rule mining
  - Rule interestingness and visualization
  - Parallel algorithms
  - …

Building Recommendation System in R

Fundamental Data Science for Data Scientist

Support, Confidence and Lift

There are several measures used to understand various aspects of associated products.

Let's understand the measures with the help of an example.

- In a store, there are 1000 transactions overall.
- Item A appears in 80 transactions and
- Item B occurs in 100 transactions.
- Items A and B appear in 20 transactions together.

Support is the ratio of number of times two or more items occur together to the total number of transactions.

- **Support of A** = Pr(A) = 80/1000 = 8% and
- **Support of B** = Pr(B) = 100/1000 = 10%.

Confidence is a conditional probability that a randomly selected transaction will include Item A given Item B.

- **Confidence of A** = Pr(A/B) = 20/100 = 20%.

Lift can be expressed as the ratio of the probability of Items A and B occurring together to the multiple of the two individual probabilities for Item A and Item B.

- **Lift** = Pr(A,B) / Pr(A).Pr(B) = (20/1000)/((80/1000)x(100/1000)) = 2.5.

How would you use Support, Confidence and Lift?

Support of a product or product bundle indicates the popularity of the product or product bundle in the transaction set. Higher the support, more popular is the product or product bundle. This measure can help in identifying driver of traffic to the store. Hence, if Barbie dolls have a higher support then they can be attractively priced to attract traffic to a store.

Confidence can be used for product placement strategy and increasing profitability. Place high-margin items with associated high selling (driver) items. If Market Basket Analysis indicates that customers who bought high selling Barbie dolls also bought high-margin candies, then candies should be placed near Barbie dolls.

Lift indicates the strength of an association rule over the random co-occurrence of Item A and Item B, given their individual support. Lift provides information about the change in probability of Item A in presence of Item B. Lift values greater than 1.0 indicate that transactions containing Item B tend to contain Item A more often than transactions that do not contain Item B.

Apriori Algorithm

?apriori

Usage

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

Arguments

`data`
: object of class `transactions` or any data structure which can be coerced into `transactions` (e.g., a binary matrix or data.frame).

`parameter`
: object of class `APparameter` or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 10.

`appearance`
: object of class `APappearance` or named list. With this argument item appearance can be restricted. By default all items can appear unrestricted.

`control`
: object of class `APcontrol` or named list. Controls the performance of the mining algorithm (item sorting, etc.)

Apriori Algorithm

So lets get started by loading up our libraries and data set.

```
# Load the libraries

library(arules)

library(arulesViz)

library(datasets)


# Load the data set

data(Groceries)
```

Explore Data

```
# Create an item frequency plot for the top 20 items
itemFrequencyPlot(Groceries, topN=20,type="absolute")
```

rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

Show the top 5 rules, but only 2 digits

options(digits=2)

inspect(rules[1:5])

```
  lhs                      rhs            support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
>
```

summary(rules)

```
set of 410 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 229 140  12

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.3     5.0     6.0

summary of quality measures:
    support           confidence        lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.83   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00125   Mean   :0.87   Mean   : 4.0
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

Sort Rules

rules<-sort(rules, by="confidence", decreasing=TRUE)

```
  lhs                          rhs          support confidence lift
1 {rice,
   sugar}                   => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}        => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                    => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                   => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}           => {whole milk}  0.0010          1  3.9
>
```

Change to have limit association in one

```
# change to have maximum of 3
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf =
0.8,maxlen=3))
inspect(rules[1:5])
```

```
  lhs                      rhs            support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
```

Rules pruned

```
subset.matrix <- is.subset(rules, rules)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1
rules.pruned <- rules[!redundant]
rules<-rules.pruned
summary(rules)
```

```
set of 330 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 216  84   1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.2     5.0     6.0

summary of quality measures:
    support           confidence        lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.82   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00127   Mean   :0.86   Mean   : 3.8
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

Targeting Items

What are customers likely to buy before buying whole milk?

What are customers likely to buy if they purchase whole milk?

This essentially means we want to set either the Left Hand Side and Right Hand Side. This is not difficult to do with R!

Find whole milk's antecedents

```
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf =
0.08), appearance = list(default="lhs",rhs="whole milk"), control =
list(verbose=F))
rules<-sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

```
  lhs                          rhs          support confidence lift
1 {rice,
   sugar}                  => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}       => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                   => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                  => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}          => {whole milk}  0.0010          1  3.9
>
```

Likely to buy after buy whole milk

```
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2),
appearance = list(default="rhs",lhs="whole milk”), control = list(verbose=F))
rules<-sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

```
  lhs              rhs                  support confidence lift
1 {whole milk} => {other vegetables}    0.075       0.29  1.5
2 {whole milk} => {rolls/buns}          0.057       0.22  1.2
3 {whole milk} => {yogurt}              0.056       0.22  1.6
4 {whole milk} => {root vegetables}     0.049       0.19  1.8
5 {whole milk} => {tropical fruit}      0.042       0.17  1.6
>
```

Workshop 4.12 - Build Recommendation System

1. Use Titanic Data download from http://www.rdatamining.com/data/titanic.raw.rdata?attredirects=0&d=1
2. Generate Association Rules
3. set rhs=c("Survived=No", "Survived=Yes") in appearance to make sure that only "Survived=No" and "Survived=Yes" will appear in the rhs of rules

Conclusion

[1] Rachel Schutt and Cathy O'Neil, *Doing Data Science*, O'Reilly Media, Inc., 2014

Q.
A.

Contact

Contact :

Email: veerasak.kritsanapraphan@gmail.com

Twitter: @veerasakk