16/4/18

* Classification and Regression Trees :- (CART)

- Build a tree by splitting on variables
- To predict an outcome for an observation, follow the splits and at the end, predict the most frequent outcome.
- There are different ways to control the splits { how many ways

(1) By setting a lower bound for the number of points in each subset.

In R it is called minbucket

- The smaller it is, the more splits will be generated.
- If too small - overfitting will occur
- If too large - model will be simple with poor accuracy.

- stevens = read.csv("stevens.csv")
- library(caTools)

1 - reverse ✓
0 - firm

spl = sample.split(stevens$Reverse, SplitRatio = 0.7)

Train = subset(stevens, spl == TRUE)

Test = subset(stevens, spl == FALSE)

- install.packages("rpart")
install.packages("rpart.plot")

Creating CART

- StevensTree = rpart(Reverse ~ Circuit + ... + Unanst, data = Train, method = "class", minbucket = 25)

- prp (StevensTree) To plot the tree

[CART is a series of decision rules which can easily be explained]

Predictions with CART

- PredictCART = predict (StevensTree, newdata = Test, type = "class")

Creating confusion matrix

- table (Test$Reverse, PredictCART)

	0	1
0	41	36
1	22	71

Accuracy = (TN + TA)/N

(41 + 71)/(41 + 36 + 22 + 71) = 65.9 %

Generating ROC curve

- library (ROCR)

PredictRoc = predict (StevensTree, newdata = Test)

Plotting ROC curve

- Pred = prediction (PredictRoc [, 2], Test$Reverse)

Perf = performance (pred, "tpr", "fpr")

↑ true +ve rate

↑ false +ve rate

plot (perf)

plot (perf, colorize = TRUE)

Generating AUC

- performance (pred, "auc") @ y.values

* Random Forests :

- Designed to improve prediction accuracy of CART
- Builds large number of CART trees.
- Each tree can split on only a random subset of the variables.
- Each tree is built from a "bagged" / "bootstrapped" sample of the data.

Parameters :

(1) Minimum number of observations in a subset (minbucket parameter from cart)
- In R, controlled by nodesize parameter
- Smaller nodesize leads to bigger trees may take longer in R.

(2) Number of trees to build
- In R, called the ntree parameter

- install. packages ("randomForest")

Creating Random Forest

- StewartForest = randomForest (Reverse ~ Circuit + ... + Urban, data = Train, nodesize = 25, ntree = 200)

[Trees can be used for Classification problems & Regression problems
When we do a Classification problem the outcome should be a factor. So, convert Reverse to factor in Train & Test set]

- Train $ Reverse = as.factor (Train $ Reverse)
  Test $ Reverse = as.factor (Test $ Reverse)

Predictions with RandomForest

- Predict Forest = predict (StevensForest, newdata = Test)

Creating Confusion Matrix

- table (Test $ Reverse, PredictForest)

	0	1
0	40	37
1	19	74

Accuracy = (40 + 74) / (40 + 37 + 19 + 74)
= 0.67%

* CART

- Interpretable !!!
- White Box !
- Transparent !!

Random Forests

- Black Box !
- Unclear !!
- High Accuracy
- low Interpretability !!!

* Optimal Classification Trees : (OCT)

- Competetive with Black Box and is interpretable
- OCT - trees with parallel splits (one variable/split)
  OCT-H - trees with hyperplane splits (multiple variables/split)

* K-fold Cross-Validation :

- is method to predict minbucket parameter in CART method.
- We first split the given training set into K equal pieces.
- Use K-1 folds to estimate a model and test model on remaining one fold ("validation set")
- Repeat the above step for each K folds

- When we use cross-validation in R, we'll use a parameter called Complexity Parameter (cp).
  - Smaller cp leads to bigger tree (might overfit)

- install.packages("caret")
- install.packages("e1071")

defining how many folds

- numFolds = trainControl(method = "cv", number = 10)

Getting the possible value for cp parameter

- cpGrid = expand.grid(.cp = seq(0.01, 0.5, 0.01))
  (.cp = (0:10) * 0.001)

Performing cross validation

- train(Reverse ~ Circuit + ... Un cancel, data = Train,
  method = "rpart",
  trControl = numFolds,
  tune Grid = cpGrid)
  - the best cp will be the o/p (Type 'fit' to get o/p)

Creating the cart model with cp

- StevensTreeCV = rpart(Reverse ~ Circuit + ... + Unsanct,
  data = Train,
  method = "class",
  cp = 0.18)

Accuracy = 72.35 %

→ Practicing CART Models :

Finding Accuracy for Baseline Model :

- Claims = read.csv("ClaimData.csv")

  Now split the data in Train and Test set using sample.split

- ClaimsTrain, ClaimsTest

- table(ClaimsTest$bucket2009, ClaimsTest$bucket2008)
  ↓ Table1 (list)

  Accuracy : 0.6838

  To find PenaltyAir

- PenaltyMatrix = matrix(c(0,1,2,3,4, 2,0,1,2,3, 4,2,0,1,2, 6,4,2,0,1, 8,6,4,2,0), byrow = TRUE, nrow = 5)

- sum(as.Matrix(Table1) * PenaltyMatrix) / nrow(ClaimsTest)

  Penalty air = 0.74

Finding Accuracy with CART model :

- library (rpart)
  library (rpart.plot)
- ClaimsTree = rpart (bucket2009 ~ age + ... + bucket2008 + reimbursement2008, data = ClaimsTra, method = "class", cp = 0.00005)

  ↑ all diseases as independent variables

  prp (ClaimsTree)
- PredictTest = predict (ClaimsTree, newdata = ClaimsTest, type = "class")

- table (ClaimsTest$bucket2009, PredictTest)

  Accuracy = 0.71
- sum (as.matrix (Table1) * PenaltyMatrix) / nrow (ClaimsTest)

  Penaltyair = 0.757

- ClaimsTree = rpart (bucket2009 ~ age + ----- , data = ClaimsTrain, method = "class", cp = 0.00005, parms = list (loss = PenaltyMatrix))

  ↑ To minimize the worst type of errors to decrease the penalty air.

* Regression Trees :

- Trees can also be used for Regression the output at each leaf of the tree is no longer a category, but a number.
- Just like classification trees, regression trees can capture non-linearities that linear regression can't.

boston - dataset

- plot (boston$LON, boston$LAT)
  points (boston$LON[boston$CHAS == 1], boston$LAT[boston$CHAS == 1], col = "blue", pch = 19)

- latlontree = rpart (MEDV ~ LAT + LON, data = boston)
  prp (latlontree)
  plot (boston$LON, boston$LAT)
  points boston$LON
  fittedvalues = predict (latlontree)
  points (boston$LON[fittedvalues >= 21.2], boston$LAT[fittedvalues >= 21.2], col = "blue", pch = 19)

* RSS :

- Residual sum of squares / Sum of square differ

$$RSS = \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

$$\sum_{\text{leaves}} (RSS \text{ at each leaf}) + \lambda S$$

$$cp = \frac{\lambda}{RSS(\text{no splits})}$$

* Text Analytics :-

- To build text Analytics model we use Bag of words method
- Bag of words - Counts the number of times each word occurred in a String.

Cleaning up Irregularities - Apple, APPLE, @apple

Stemming - Algorithmic process of performing reduction

Eg: argue argued argues arguing

↓

argu

↳(stem)

- tweets = read.csv("tweets.csv", stringsAsFactors = FALSE)
- tweets$Negative = as.factor(tweets$Avg <= -1)

↳ creates a new variable with
TRUE if tweets Avg is <= -1
FALSE if tweets Avg is > -1

Preprocessing Text Data (Bag of Words Approach)

- install.packages("tm")
- install.packages("SnowballC")
- corpus = Corpus(VectorSource(tweets$Tweet))

↳ A collection of documents

corpus[[1]] - Shows the first Tweet

stopwords("english")[1:10]

- corpus = tm_map(corpus, tolower)

corpus = tm_map(corpus, removePunctuation)

corpus = tm_map(corpus, removeWords, c("apple", stopwords("english")))

corpus = tm_map(corpus, stemDocument)

Extracting the word frequencies

- frequencies = DocumentTermMatrix(corpus) → o/p : 3289 words

inspect(frequencies[1000:1005, 505:515])

findFreqTerms(frequencies, lowfreq = 20)

↑ to find the most popular terms

here, the words that appear atleast 20 times

- sparse = removeSparseTerms(frequencies, 0.995)

↑ Sparsity Threshold

sparse → o/p : 309 words

Converting sparse matrix to dataframe

- tweetsSparse = as.data.frame(as.matrix(sparse))

- *colnames(tweetsSparse) = make.names(colnames(tweetsSparse))

↑ convert variable names to appropriate names

Adding the dependent variable to Dataset

- tweetsSparse$Negative = tweets$Negative

Splitting the data into Train and Test set

```
- set.seed(123)
  split = sample.split(tweets_Sparse$Negative, SplitRatio = 0.7)
  trainSparse = subset(tweets_Sparse, split == TRUE)
  testSparse = subset(tweets_Sparse, split == FALSE)
```

Use CART to build predictive model

```
- tweetCART = rpart(Negative ~ ., data = trainSparse, method = "class")

  prp(tweetCART)

- predictCART = predict(tweetCART, newdata = testSparse, type = "class")
```

Accuracy

```
- table(testSparse$Negative, predictCART)   -> 0.88
```

Using baseline model for Accuracy

```
- table(testSparse$Negative)
```

FALSE 300 TRUE 55 -> 300/355 = 0.84

Using Random forests to build Accuracy

```
- tweetRF = randomForest(Negative ~ ., data = trainSparse)
  predictRF = predict(tweetRF, newdata = testSparse)
  table(testSparse$Negative, predictRF)   -> 0.885
```

- grepl("cat", "dogs and cats", fixed = TRUE) → TRUE

  grepl("cat", "dogs and rats", fixed = TRUE) → FALSE

Eg:

wikiWords2 $ HTTP = ifelse(grepl("http", wiki$Abbrd, fixed = TRUE), 1, 0)

* Clustering :-

- Netflix Algorithms - Cinematch
- Collaborative Filtering - making predictions for a user by collecting information from other users.

Content Filtering - making predictions by ~~of~~ number of ~~applications~~ comparision b/w content of the item and a user profile.

- Clustering - unsupervised learning methods
  - ↳ methods - Hierarchical clustering
    K-means clustering

=> Distance between points :

Euclidean Distance (ED)

$$d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ik} - x_{jk})^2}$$

k = number of independent variables

Distance between clusters :

Minimum distance - distance (ED) between the points that are closest in that cluster.

Centroid distance - distance between centroid of clusters
↑
avg of all data points in that cluster

Normalize Data = $\frac{\text{Subtracting the mean of data}}{\text{Standard deviation}}$

* Hierarchial Clustering :

Dendrogram - the display of hierarchial cluster process

Read data from text file

- movies = read.table ("movieLens.txt", header = FALSE, sep = "|", quote = "\"")

Adding col names to movies

- colnames (movies) = c ("ID", "Title", "Release Date", ... "westerns")

To remove variables from data-set

- movies $ ID = NULL

movies $ Release Date = NULL

To remove duplicates in dataset

- movies = unique (movies)

Hierarchial Clustering

To find distance b/w data points

- distances = dist (movies [1:20], method = "euclidean")

Cluster the data points

- clusterMovies = hclust (distances, method = "ward")

Plotting dendrogram

- plot (clusterMovies)

Plotting Datapoints to its cluster

- clusterGroups = cutree(clusterMovies, k = 10) ← Dividing into 10 clusters

To know the percentage of movies in each genre

- tapply(movies $ Action, clusterGroups, mean)

- subset(movies, Title = "Man in Black (~~1997~~)") -> 257

clusterGroups[257] -> 2 (Cluster 2)

- cluster2 = subset(movies, clusterGroup == 2)

cluster2 $ Title [1:10] -> will show the Top 10 movies in cluster 2

* K-means Clustering :

(1) Specify desired number of clusters

(2) Randomly assign each datapoint to a cluster

(3) Compute cluster centroids

(4) Reassign each point to the closest cluster centroid

(5) Re-compute cluster centroids.

(6) Repeat 4 & 5 until no improvements are made.

* Image Segmentation :-

Applications :

- Medical Imaging - Locate & Measure Tumor
- Object Detection - photos, facial recognition
- Recognition Tasks - fingerprint, iris

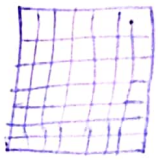
Methods :

- Clustering methods
- Edge detection
- Region-growing methods

* Clustering method :

Intensity vector of size $n = 7 \times 7$

Pairwise distances $\frac{n(n-1)}{2}$



7 × 7 pixels

Example :

Reading a matrix data

- flower = read.csv("flower.csv", header = FALSE)
- flowerMatrix = as.matrix(flower) ← Converting to matrix
- flowerVector = as.vector(flowerMatrix) ← Converting to vector

Hierarchial Clustering

To find distance

- distance = dist(flowerVector, method = "euclidean")

- clusterIntensity = hclust (distance, method = "ward")

Plotting clusters in dendrogram

- plot (clusterIntensity)
- rect.hclust (clusterIntensity, k = 3, border = "red")

Getting clusters

- flowerCluster = cutree (clusterIntensity, k = 3)

To get mean intensities of 3 clusters

- tapply (flowerVector, flowerClusters, mean)

Output the image

- dim (flowerCluster) = c (50, 50) ← Converting vectors to matrix
- image (flowerCluster, ~~axes~~ = FALSE) ← O/P image

~~image (flowerCluster, axes = FALSE, col =~~ (Matrix)

- image (flowerMatrix, axes = FALSE, ← I/P image

col = gray (seq (0, 1, length = 256)))

Example 2 :

- healthy = read.csv("healthy.csv", header = FALSE)
  healthyMatrix = as.matrix(healthy)
  healthyVector = as.vector(healthyMatrix)

Specifying number of clusters

- K = 5
  set.seed(1)
  Kmeans clustering
- KHC = Kmeans(healthyVector, centers = K, iter.max = 1000)

Extracting the image from clusters

- healthyClusters = KHC$cluster

[From KHC we can get the means (centers) of each cluster and size of each cluster
↑ str(KHC)]

Output Image

- dim(healthyClusters) = c(nrow(healthyMatrix), ncol(healthyMatrix))
  image(healthyClusters, axes = FALSE, col = rainbow(k))

Detecting Tumors in MRI from Example 2

- tumor = read.csv("tumor.csv", header = FALSE)

tumorMatrix = as.matrix(tumor)

tumorVector = as.vector(tumorMatrix)

Train Set Test Set

- install.packages("flexclust")

Conversion of data for prediction

- KMC.kcca = as.kcca(KMC, healthyVector)

tumorClusters = predict(KMC.kcca, newdata = tumorVector)

- dim(tumorClusters) = c(nrow(tumorMatrix), ncol(tumorMatrix))

image(tumorClusters, axes = FALSE, col = rainbow(k))

* Visualizations :-

Data visualizations - mapping of data properties to visual properties

ggplot -> Data - dataframe

Aesthetic mapping - color, shape, scale, x-y axes, subsets

Geometric objects - points, lines, boxplots, bars, polygons

- WHO = read.csv("WHO.csv")
- install.packages("ggplot2")
- scatterplot = ggplot(WHO, aes(x = GNI, y = FertilityRate))
- scatterplot + geom_point() ← point graph
- scatterplot + geom_line() ← line graph

geom_point(color = "blue", size = 3, shape = 17)

↑ Triangle = 17

- scatterplot + geom_point() + ggtitle("GNI vs FR")

Saving the plot into pdf

- pdf("MyPlot.pdf")
  print(fertilityGNIplot)
  dev.off()

Three variable plot

- ggplot (WHO , aes (x = GNI , y = Fertility Rate , col = Regions))
  + geom_point ()

Scatterplot1 =

- ggplot (WHO , aes (x = log (Fertility Rate) , y = Under 15))
  + geom_point ()

- model = lm (Under 15 ~ log (Fertility Rate) , data = WHO)

Scatterplot1 + stat_smooth (method = "lm"
, level = 0.99
, se = FALSE
, colr = "orange"

→ Heat Maps :

3 variables - x-axis (1) , y-axis (2) , shades of color (3)

- mvt = read.csv ("mvt.csv" , stringsAsFactors = FALSE)

Converting Date

- mvt$Date = strptime (mvt$Date , format = "%m/%d/%y %H:%M")

mvt$Weekday = weekdays (mvt$Date)

mvt$Hour = mvt$Date$hour

Converting table to dataframe

- WeekdayCounts = as.data.frame (table (mvt$Weekday))

- ggplot (WeekdayCounts , aes (x = Var1 , y = Freq))
  + geom_line (aes (group = 1))

↑ Scatterplot2

To get the weekdays in chronological order

```
- WeekdayCounts$Var1 = factor(WeekdayCounts$Var1,
                              ordered = TRUE,
                              levels = c("Sunday",
                                         "Monday",
                                         :
                                         "Saturday"))
```

```
- scatterplot2 + xlab("Day of week")
               + ylab("Total motor thefts")
```

Converting Data from table to data-frame

```
- DayHourCounts = as.data.frame(table(mvt$Weekday,
                                      mvt$Hour))
- DayHourCounts$Hour = as.numeric(as.character(DayHourCounts
                                               $Var2))
```

```
- ggplot(DayHourCounts, aes(x = Hour, y = Freq)) +
      geom_line(aes(group = Var1), col = Var1)
```

Plotting Heatmap

```
- ggplot(DayHourCounts, aes(x = Hour, y = Var1))
      + geom_tile(aes(fill = Freq))
      + scale_fill_gradient(name = "Total MV Thefts", low = "white",
                            high = "red")
      + theme(axis.title.y = element_blank())
```

↳ to get rid of the axis name

* MAP Plotting :

- install.packages("maps")
- install.packages("ggmap")

Loading a map

```
-chicago = get_map(location = "chicago", zoom = 11)
ggmap(chicago)
```

Plotting on map

```
- ggmap(chicago) + geom_point(data = mvt[1:100,],
                              aes(x = Longitude, y = Latitude))
```

```
- LatLongCounts = as.data.frame(table(round(mvt$Longitude,2),
                                       round(mvt$Latitude,2)))
```

↑ creating a dataframe with latitude & longitude and rounding off with 2 decimals.

Converting a factor variable to numeric variable

```
LatLongCounts$Long = as.numeric(as.character(LatLongCounts$Long))
LatLongCounts$Lat = as.numeric(as.character(LatLongCounts$Lat))

ggmap(chicago) + geom_point(data = LatLongCounts,
                            aes(x = Long, y = Lat, color = Freq,
                                size = Freq))
  + scale_color_gradient(low = "yellow", high = "red")
```

(OR)

```
ggmap(chicago) + geom_tile(data = LatLongCounts,
                           aes(x = Long, y = Lat, alpha = Freq),
                           fill = "red")
```

→ HeatMap on the united states : (m-10, a-12)

- murders = read.csv("murders.csv")

Loading a map (United States)

- statesMap = map_data("state")

Plotting the US map

- ggplot(statesMap, aes(x = long, y = lat, group = group))
  + geom_polygon(fill = "white", color = "black")

Making the state variable same in murders & statesMap df

- murders $ regions = tolower(murders $ State)

Merging the 2 df

- murderMap = merge(statesMap, murders, by = "regions")

- ggplot(murderMap, aes(x = long, y = lat, group = group, fill = MurderRate))
  + geom_polygon(color = "black")
  + scale_fill_gradient(low = "black", high = "Red", guide = "legend")

→ Bar graph :

- intl = read.csv("intl.csv")
- ggplot(intl, aes(x = Region, y = Percentage of Intl))
  - x var
  - y var
  + geom_bar(stat = "identity")
  + geom_text(aes(label = Percentage of Intl))

- intl = transform(intl, Region = reorder(Region, -Percent of Intl))
  - ordering based on the percentage of Intl values
  - decreasing order

intl $ Percent of Intl = intl $ Percent of Intl * 100

↳ changing y-axis

Updated ggplot :

- ggplot(intl, aes(x = Region, y = Percent of Intl.))
  + geom_bar(stat = "identity", fill = "dark blue")
  + geom_text(aes(label = Percent of Intl), vjust = -0.4)
  + ylab("Percent of International students")
  + theme(axis.title.x = element_blank(),
    axis.text.x = element_text(angle = 45, hjust = 1))