Decision Tree Algorithms - I

Contents

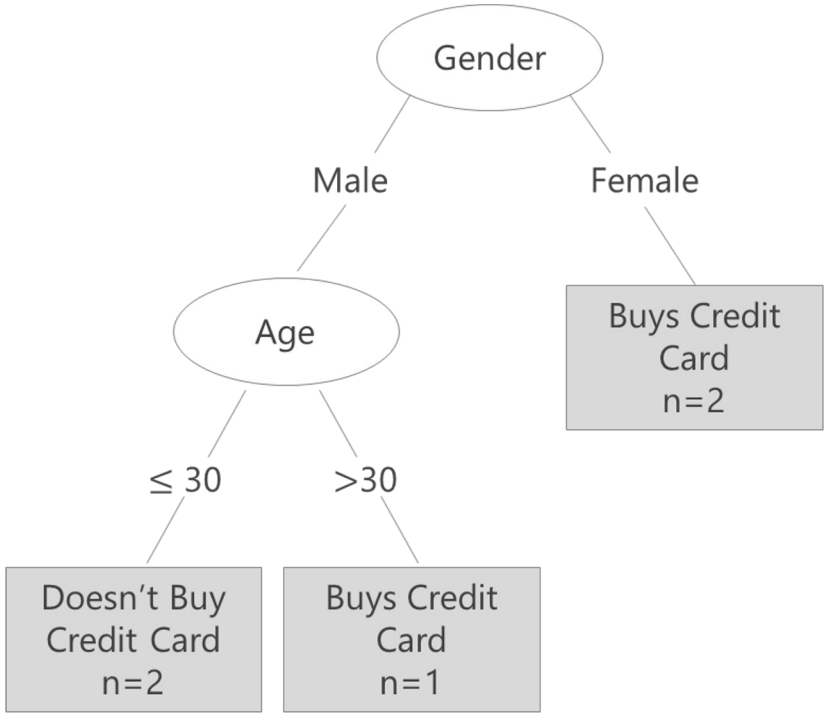
Introduction to Decision Tree

- One of the most robust predictive modeling techniques, **Decision Tree** uses data mining techniques for model building.
- Decision Tree **breaks down a data set into smaller subsets and presents association between target variable(dependent) and independent variables as a tree structure.**
- Final result is a **tree with Decision Nodes and Leaf Nodes.**
- A decision node has two or more branches and leaf node represents a classification or decision.

Decision Tree – Basic Framework

Suppose we have information about 5 customers and their decision about buying a credit card. This data can be represented as a Decision Tree:

Customer No.	Gender	Age	Occupation	Buys Credit Card
01	Male	≤30	Student	No
02	Male	≤30	Professional	No
03	Female	≤30	Business	Yes
04	Male	>30	Business	Yes
05	Female	>30	Professional	Yes



- First subset is based on Gender. All females opt for credit cards.
- Males, however, can be split further, based on Age. Male customers of age ≤30 years do not buy a credit card, whereas those >30 do buy cards.

What is CART ?

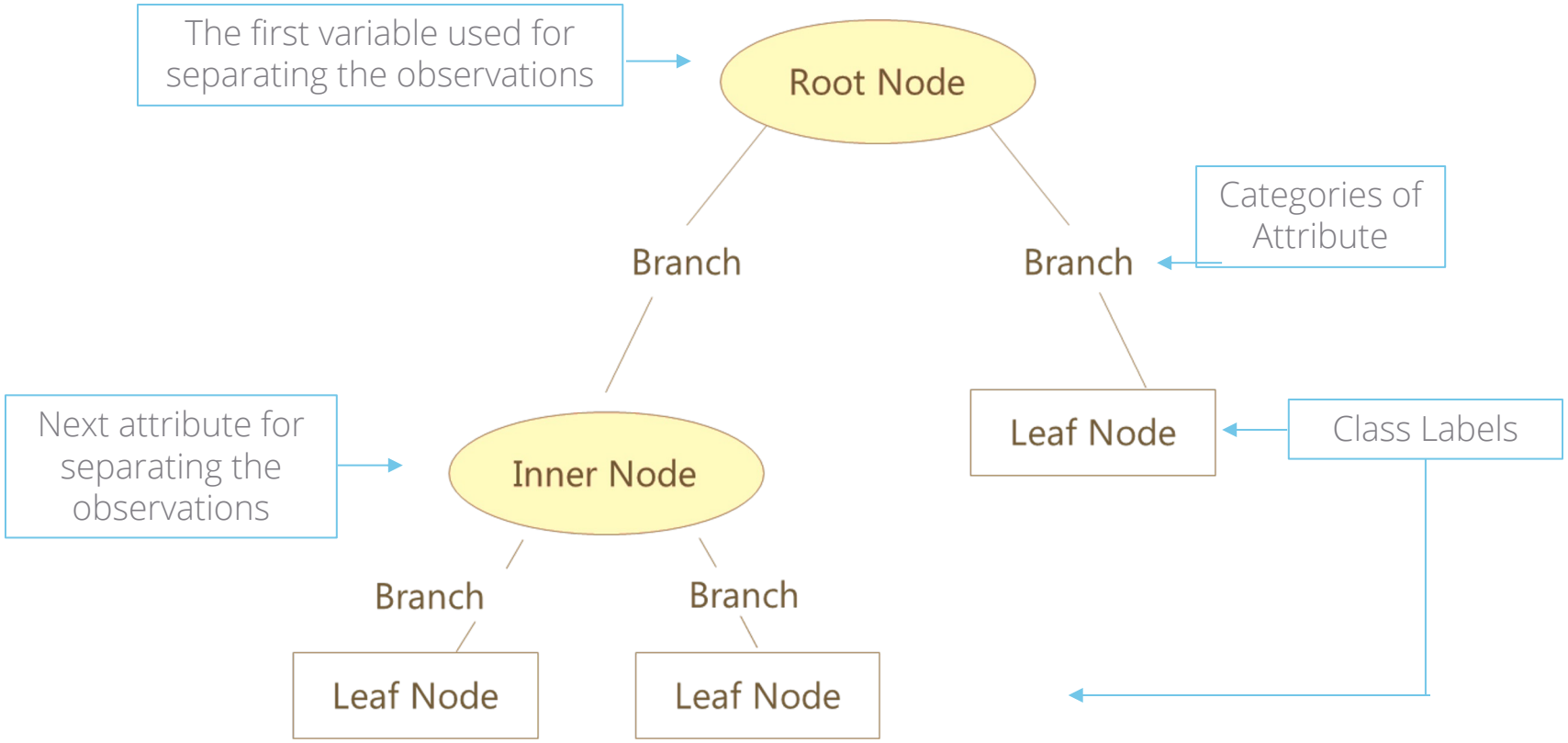
- The term **Classification And Regression Tree (CART)** analysis is an umbrella term used to refer to predictive decision tree procedures, first introduced by Breiman et al.

Classification Tree	Regression Tree
When the predicted outcome is the class to which the data belongs	When the predicted outcome can be considered a real number
In such cases, **dependent variable is categorical** Independent variables can be either continuous or categorical or both	In such cases, **dependent variable is continuous** Independent variables can be either continuous or categorical or both

Decision Tree – Basic Components

Component	Description	Alternate terms
Root node	Has no incoming edges and zero or more outgoing edges	Parent node
Internal nodes	Each has exactly one incoming edge and two or more outgoing edges	Decision nodes / Child nodes
Leaf node	Each has exactly one incoming edges and no outgoing edges	Terminal nodes
Branches	Categories of attributes	Edges

Decision Tree – Basic Components



* Class labels show observations belong to which class. The leaf node also shows Number of observations and Error rate (Actual classification vs classification given by the tree)

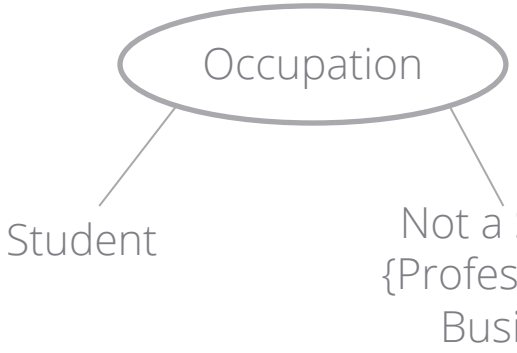
Binary and Non Binary Trees

Consider the same example illustrated earlier.

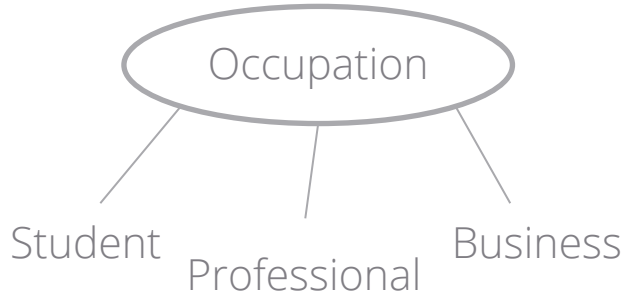
Occupation is the nominal attribute under consideration, with three distinct categories.

Customer No.	Occupation	Buys Credit Card
01	Student	No
02	Professional	No
03	Business	Yes
04	Business	Yes
05	Professional	Yes

There are two ways in which a decision node can be split into further branches:



Binary Tree



Non-Binary Tree

* Binary and non-binary branches can be generated for ordinal and continuous variables as well

Decision Tree Algorithms

- Trees used for regression and trees used for classification have some similarities - but also some differences, such as the procedure used to determine where to split.
- There are many specific decision-tree algorithms. Notable ones include:

1. ID3 (Iterative Dichotomiser 3)
2. C4.5 (Successor of ID3)
3. CART (Classification And Regression Tree)
4. CHAID (CHi-squared Automatic Interaction Detector)
    - Performs multi-level splits when computing classification trees
5. MARS (Multivariate Adaptive Regression Splines)
    - Extends decision trees to handle numerical data better
6. Conditional Inference Trees
    - Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning

ID3 Algorithm

- The **Iterative Dichotomiser 3 (ID3)** algorithm is invented by J. R. Quinlan

Uses a **top-down, greedy search method to build a classification decision tree**

Considers a fixed set of examples (training data) to build decision tree and the results are used to classify future observations

Precursor to C4.5 algorithm

ID3 Algorithm

ID3 searches through the attributes of the fixed set of observations and extracts the attribute that best separates the given examples

If the attribute perfectly classifies the training sets then ID3 stops; otherwise it recursively operates on the n (where n = number of possible values of an attribute) partitioned subsets to get their "best" attribute

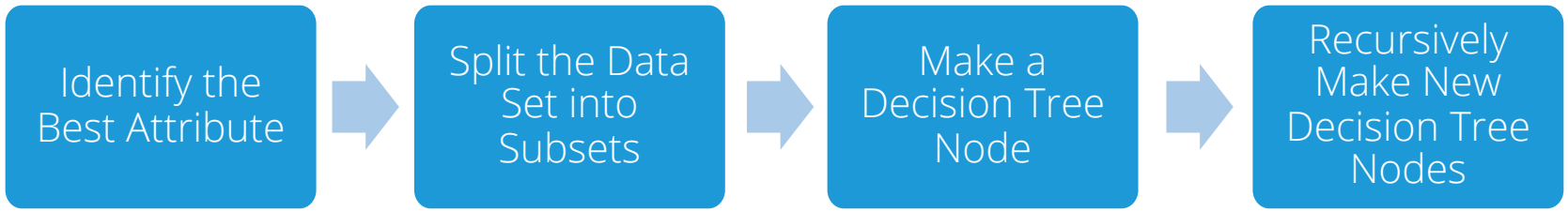
The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices

ID3 Algorithm

The training data set must have the following properties:

1. Attributes used to describe each observation must be uniform
2. Classification must be predefined for all observations
3. Classes should be discrete
4. The data set should have sufficient observations

Once such data set is obtained, the ID3 employs the following broad procedure



In order to decide which attribute should be included in the decision node, ID3 uses

Entropy and Information Gain

Entropy

- Entropy measures the homogeneity of a sample. It is used as a parameter for checking the amount of uncertainty associated with a set of probabilities.
- Entropy lies between 0 and 1

  If the sample is completely homogeneous the entropy is 0 and if the sample is equally divided it has entropy of 1
- Entropy can be of two types, for each category and at the variable level
- Entropy of a category is calculated as:

$$-\ P1 * \log_2(P1) -\ P2 * \log_2(P2)$$

where,

P1 is the proportion of class 1

P2 is the proportion of class 2

Entropy of a Category

Let us consider survey data from three cities depicting shopper's preferred brand

City	Brand A Voters	Brand B Voters	Number of Voters		
Manchester	90	310	400	22.5%	77.5%
Birmingham	10	90	100	10%	90%
London	100	100	200	50%	50%

Entropy for each city is calculated as:

Manchester: $- 0.225 * \log_2(0.225) - 0.775 * \log_2(0.775) = \mathbf{0.76919}$

Birmingham: $- 0.1 * \log_2(0.1) - 0.9 * \log_2(0.9) = \mathbf{0.46900}$

London: $- 0.5 * \log_2(0.5) - 0.5 * \log_2(0.5) = \mathbf{1}$

Entropy at the Variable Level

- Entropy at the variable level can be derived by adding weighted averages of all category level entropy values
- Weights are the proportion of respondents in each category (here in each city)

In the example under consideration,

Weights for the categories are

Manchester: $\mathbf{400/700 = 0.5714}$

Birmingham: $\mathbf{100/700 = 0.1428}$

Chennai: $\mathbf{200/700 = 0.2857}$

Entropy at the variable level is

$$\mathbf{0.57 * 0.76919 + 0.14 * 0.46900 + 0.29 * 1 = 0.79225}$$

Information Gain

- Information Gain is based on the decrease in entropy after a dataset is split on an attribute
- Constructing a decision tree is about finding attribute that returns the highest information gain

$$\text{Information Gain} = \text{Entropy of Sample (Dependent Variable)} - \text{Average Entropy of Any of the Independent Variable}$$

- Information gain can be interpreted as ability of reducing the uncertainty (Entropy) and hence increase predictability

Information Gain

City	Brand A Voters	Brand B Voters	Number of Voters		
Manchester	90	310	400	22.5%	77.5%
Birmingham	10	90	100	10%	90%
London	100	100	200	50%	50%

Entropy for complete sample is calculated as follows:

P1 = (Total Brand A Voters/Total Voters)

P2 = (Total Brand B Voters/Total Voters)

$$\textbf{Entropy} = -(0.286) * \log_2(0.286) - (0.714) * \log_2(0.714) = \mathbf{0.86312}$$

Information Gain

Entropy at the variable level (Weighted average)

$$0.86312 - 0.79225 = \mathbf{0.070868}$$

Information Gain and ID3 Algorithm

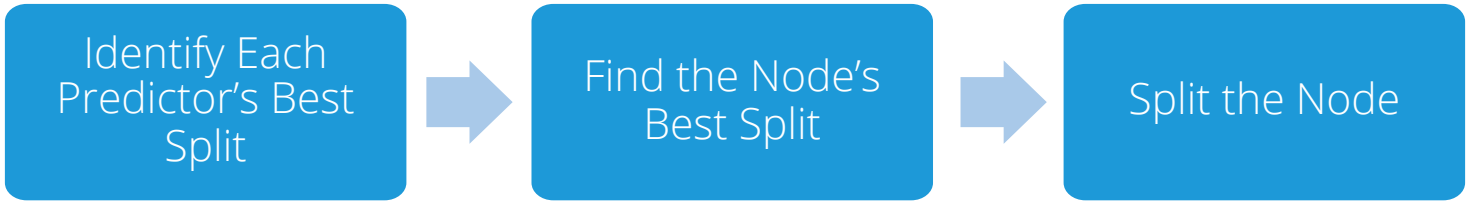
- Let us now go back to the basic ID3 algorithm; Step 1 of which is 'Identify the Best Attribute'

- Information Gain value is used to determine which attribute is the "best" – the attribute with most information gain is chosen
- Information gain for a variable is high when that variable has the low entropy at the variable level (Weighted average)
- Low entropy for a variable implies the classification based on that attribute is fairly homogenous , hence this attribute is selected as the first best attribute
- The same process is repeated till all attributes are used as split variables

CART Algorithm

- Classification and Regression Tree (CART) algorithm generates a binary decision tree by splitting a node into two branches
- Root node contains the complete sample (training data)
- The splits are univariate – each split depends on the value of only one predictor variable

The algorithm can be divided into three steps:



Gini impurity is used as the splitting criteria in classification problems

Gini Impurity

- Gini Impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset

$$\text{Gini}\,(t) = 1 - \sum_{i=1}^{c} [p(i|t)]^2$$

where

p(i|t) : Fraction of records belonging to class i at node t

- It reaches its minimum (zero) when all cases in the node fall into a single target category
- Attribute with the smaller Gini Impurity is considered for the split

Quick Recap

Decision Tree Algorithms	• ID3 uses top-down, greedy search method to build a classification decision tree • CART algorithm generates a binary decision tree, by splitting a node into two branches. Root node contains the complete sample.
Entropy, Information Gain and Gini Impurity	• Entropy measures the homogeneity of a sample • Information Gain is based on the decrease in entropy after a dataset is split on an attribute • Information Gain= Entropy of Sample − Average Entropy of Any of the Independent Variable • Gini Impurity measures how often a randomly chosen element from the set would be incorrectly labeled

Decision Tree Algorithms - II

Contents

Case Study – Predicting Loan Defaulters

Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

Objective

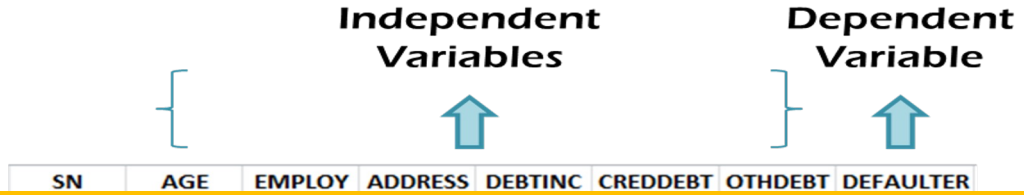
- To predict whether the customer applying for the loan will be a defaulter

Available Information

- Sample size is 700
- **Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts** are the independent variables
- **Defaulter** (=1 if defaulter, 0 otherwise) is the dependent variable

Data Snapshot

BANK LOAN



Column	Description	Type	Measurement	Possible Values
SN	Serial Number	Integer	-	-
AGE	Age Groups	Integer	1(<28 years), 2(28-40 years), 3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Integer	-	Positive value
ADDRESS	Number of years customer staying at current address	Integer	-	Positive value
DEBTINC	Debt to Income Ratio	Continuous	-	Positive value
CREDDEBT	Credit to Debit Ratio	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Integer	1(Defaulter), 0(Non-Defaulter)	2

Classification Tree in Python

```
# Importing the Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,
DecisionTreeRegressor, plot_tree

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, accuracy_score,roc_curve, roc_auc_score
```

- **sklearn.tree** module includes Decision Tree – based models for classification and regression

Classification Tree in Python

```
# Importing and Readying the Data for Modeling
bankloan = pd.read_csv("BANK LOAN.csv")


bankloan1 = bankloan.drop(['SN'], axis = 1)

bankloan1['AGE'] = bankloan1['AGE'].astype('category')

bankloan2 = pd.get_dummies(bankloan1)
bankloan2.head()
```

drop() is used to remove unwanted variables.

pd.get_dummies() converts categorical variables into dummy variables. Since AGE is a categorical variable, it is converted.

```
# Output
```

	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER	AGE_1	AGE_2	AGE_3
0	17	12	9.3	11.36	5.01	1	0	0	1
1	10	6	17.3	1.36	4.00	0	1	0	0
2	15	14	5.5	0.86	2.17	0	0	1	0
3	15	14	2.9	2.66	0.82	0	0	0	1
4	2	0	17.3	1.79	3.06	1	1	0	0

Classification Tree Using Information Gain

```
# Creating Data Partitions

X = bankloan2.loc[:,bankloan2.columns != 'DEFAULTER']
y = bankloan2.loc[:, 'DEFAULTER']

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.30,
                                                    random_state =
999)
```

- **train_test_split()** from sklearn.model_selection is used to split dataset into random train and test sets.
- **test_size** represents the proportion of dataset to be included in the test set.
- **random_state** sets the seed for the random number generator.

Classification Tree Using Information Gain

```
# Classification Tree Using Information Gain
dtcl = DecisionTreeClassifier(criterion='entropy',
                              min_samples_split=
int(len(X_train)*.10))
```

- **DecisionTreeClassifier()** from sklearn.tree fits a classification tree.
- **criterion=** 'entropy' specifies the function to measure the split. Default is 'gini' for Gini impurity. 'entropy' stands for information gain.
- **min_samples_split=** minimum number of samples required to split an internal node. This number is set to be 10% of the sample size.
- The output displays model specifications.

```
# Output
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=49,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Classification Tree in Python – Prediction

```
# Generating Predictions for the model

y_pred = dtcl.predict(X_test)
y_pred_probs = dtcl.predict_proba(X_test)

cutoff = 0.3
pred_test = np.where(y_pred_probs[:,1] > cutoff, 1, 0)
pred_test

# Output

array([0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1])
```

Classification Tree in Python – Confusion Matrix

```
# Confusion Matrix
confusion_matrix(y_test, pred_test, labels=[0, 1])
array([[107,  50],
       [ 14,  39]], dtype=int64)
accuracy_score(y_test, pred_test)
0.6952380952380952
precision_score(y_test, pred_test)
0.43820224719101125
recall_score(y_test, pred_test)
0.7358490566037735
```

- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as 'Sensitivity' = true positives / (true positives + false negatives)

```
# Area Under ROC Curve
auc = roc_auc_score(y_test, y_pred_probs[:,1])
print('AUC: %.3f' % auc)
AUC: 0.720
```

Classification Tree in Python – ROC Curve

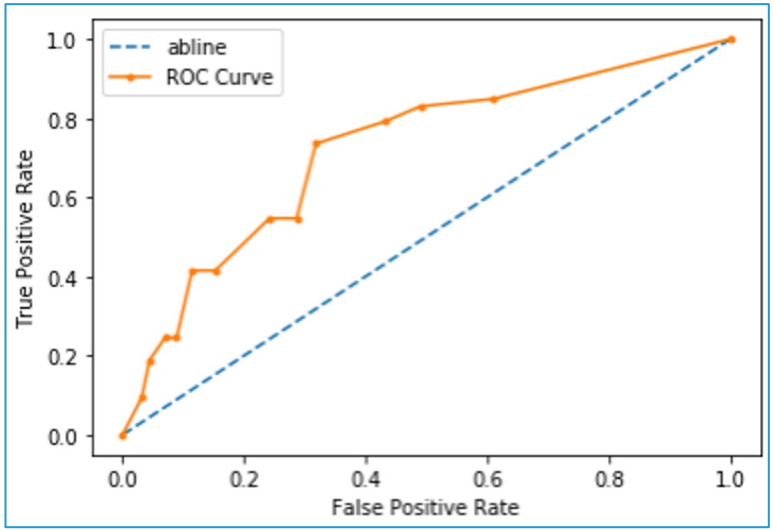
```
# Area Under ROC Curve
DTfpr, DTtpr, thresholds = roc_curve(y_test, y_pred_probs[:,1])

abline_probs = [0 for _ in range(len(y_test))]
abline_auc = roc_auc_score(y_test, abline_probs)
abline_fpr, abline_tpr, _ = roc_curve(y_test, abline_probs)

plt.plot(abline_fpr, abline_tpr, linestyle='--', label='abline')
plt.plot(DTfpr, DTtpr, marker='.', label='ROC Curve')
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.legend(); plt.show()
```

Classification Tree in Python – ROC Curve

```
# Output
```



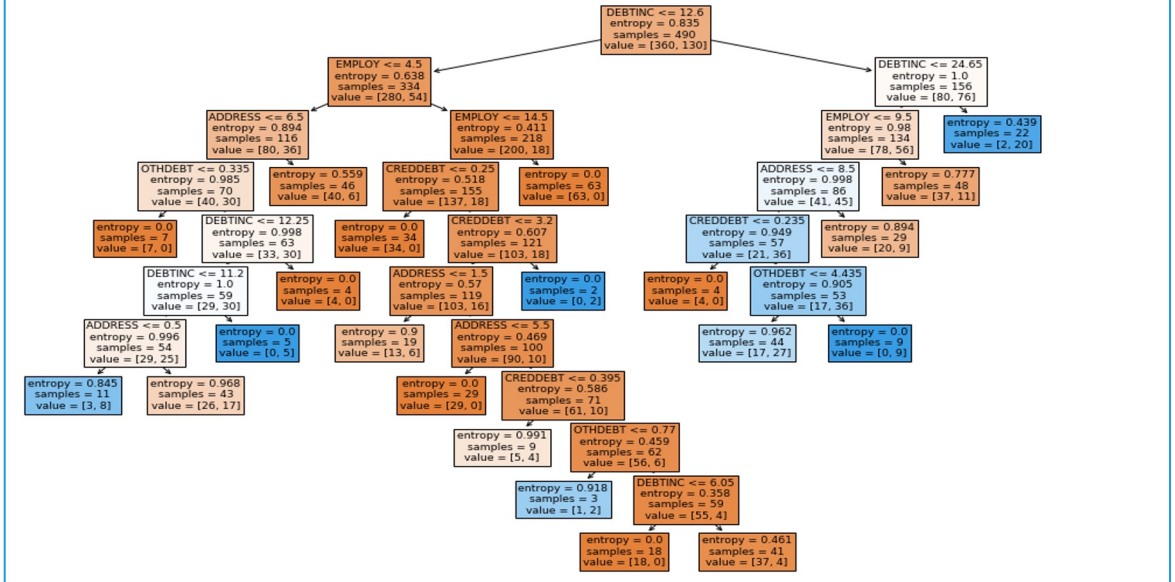

```
# Plotting The Tree
dtcl_infgain = DecisionTreeClassifier(criterion='entropy', min_samples_split
= int(len(X_train)*.10))
dtcl_infgain.fit(X_train, y_train)
```

Classification Tree Using Information Gain

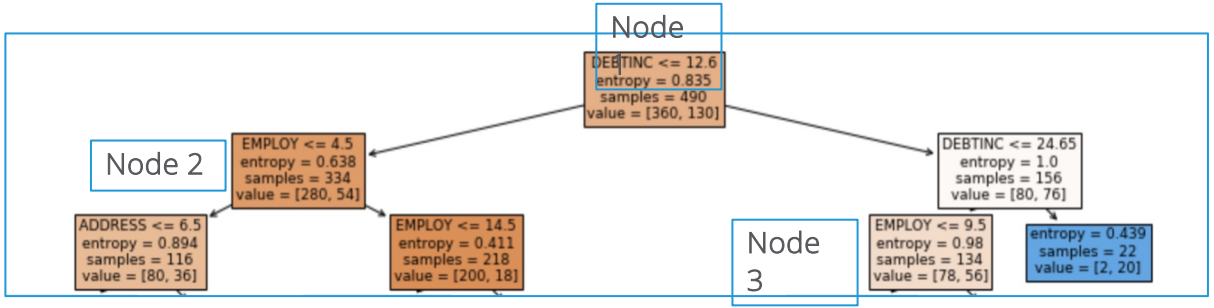
```
# Plotting The Tree
from sklearn.tree import plot_tree
plt.figure(figsize = (16,10))
plot_tree(dtcl_infgain, filled = True, feature_names = list(X.columns))
plt.show();
```

- **plot_tree** is used to plot the decision tree.
- **filled= True** paints nodes to indicate majority class for classification and **feature_names** is used to mention the feature names.

```
# Output
```



Classification Tree Interpretation



Interpretation :

- Due to a large number of continuous predictors, a tree with several nodes and branches is generated.
- Tree starts with all 490 observations (Train set). 360 are non-defaulters (0) and the remaining 130 are defaulters (1).
- DEBTINC is the first split variable, left branch is <=12.6 and right branch is >12.6. 334/490 have DEBTINC<=12.6.
- EMPLOY is the second split on left branch, which further divides 334 obs. into 280 non-defaulters (0) and the remaining 54 as defaulters (1).
- The algorithm progresses till no further variable split is left.

Case Study – Modeling Motor Insurance Claims

Background

- A car insurance company collects range of information from their customers at the time of buying and claiming insurance. The company wishes to check if there any of it can be used to model and predict claim amount

Objective

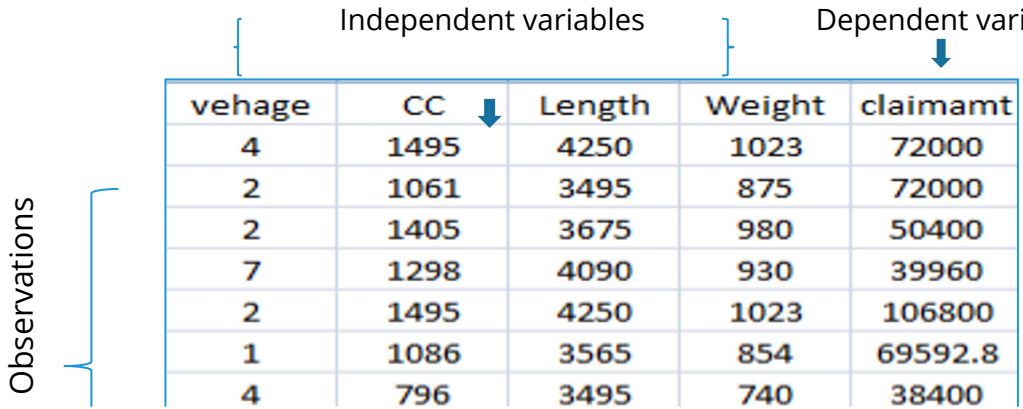
- To model motor insurance claim amount based on vehicle related information collected at the time of registering and claiming insurance

Available Information

- Sample size is 1000
- Independent Variables: Vehicle Information – **Vehicle Age, Engine Capacity, Length and Weight of the Vehicle**
- Dependent Variable: **Claim Amount**

Data Snapshot

Motor_Claims



vehage	CC	Length	Weight	claimamt
4	1495	4250	1023	72000
2	1061	3495	875	72000
2	1405	3675	980	50400
7	1298	4090	930	39960
2	1495	4250	1023	106800
1	1086	3565	854	69592.8
4	796	3495	740	38400

Columns	Description	Type	Measurement	Possible values
vehage	Age of the vehicle at the time of claim	Integer	Years	positive values
CC	Engine capacity	Integer	cc	positive values
Length	Length of the vehicle	Integer	mm	positive values
Weight	Weight of the vehicle	Integer	kg	positive values
claimamt	Claim amount	Continuous	INR	positive values

Regression Tree in Python

```
# Importing and Readying the Data for Modeling
motor = pd.read_csv("Motor Claims.csv")
motor.head()

# Output
```

	vehage	CC	Length	Weight	claimamt
0	4	1495	4250	1023	72000.0
1	2	1061	3495	875	72000.0
2	2	1405	3675	980	50400.0
3	7	1298	4090	930	39960.0
4	2	1495	4250	1023	106800.0

Since all variables are continuous, no further processing is needed.

```
# Creating Data Partitions
X = motor.loc[:,motor.columns != 'claimamt']
y = motor.loc[:, 'claimamt']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.30,
                                                    random_state =
999)
```

Regression Tree in Python

```
# Regression Tree Using MSE

dtreg = DecisionTreeRegressor(min_samples_split =
                              int(len(X_train)*.10))
dtreg.fit(X_train, y_train)
```

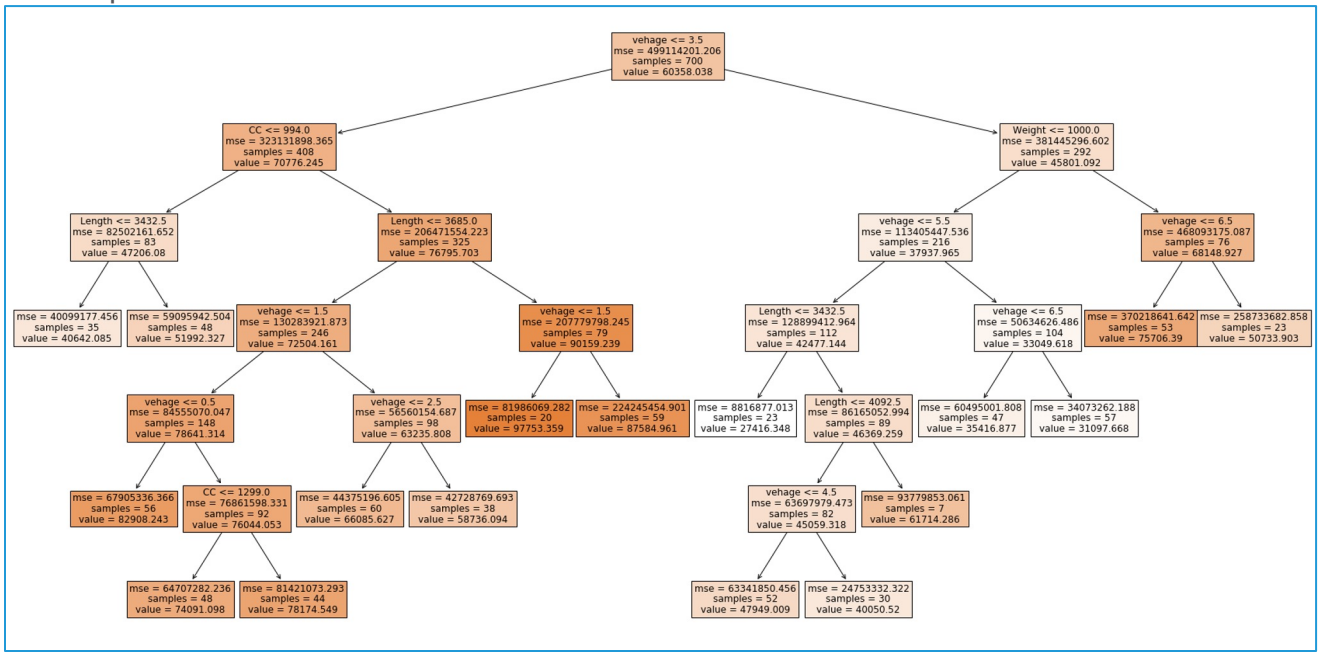
- **DecisionTreeRegressor()** from sklearn.tree fits a regression tree.
- **min_samples_split=** minimum number of samples required to split an internal node. This number is set to be 10% of the sample size.
- The output displays model specifications.

```
# Output
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=70,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

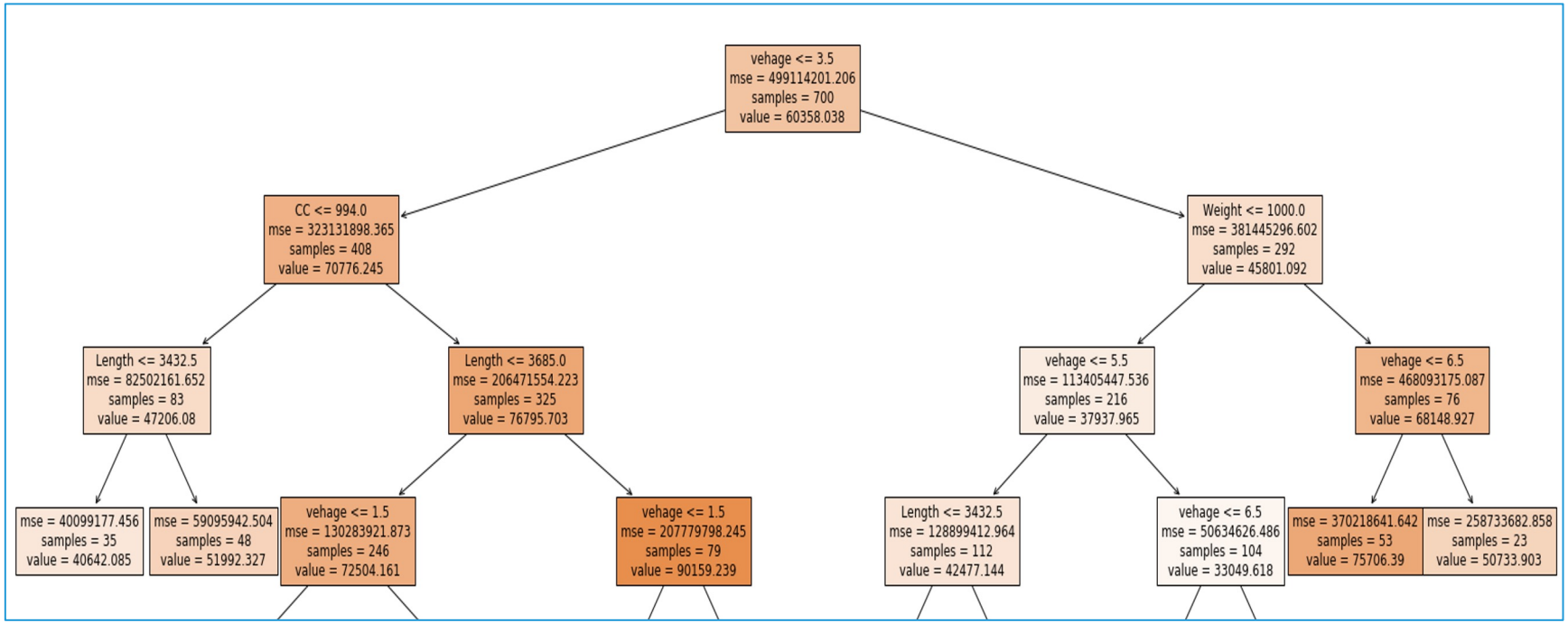
Regression Tree in Python

```
# Plotting The Tree
plt.figure(figsize = (30,15))
plot_tree(dtreg, filled = True, feature_names = list(X.columns))
plt.show();
```

```
# Output
```



Regression Tree in Python



Interpretation :

- Tree starts with all 700 training observations, 60358.038 is the average claim amount of these observations.
- vehage is the first split variable, left branch is <=3.5 and right branch is >3.5.
- 408 have vehage <=3.5 which has 70776.246 average claim amount .
- The process continues till there is no variable left for splitting.

Regression Tree in Python

```
# Predictions
y_pred_reg = dtreg.predict(X_test)
y_pred_reg
```

- **predict()** returns predicted regression value for X. Output is an array.

```
# Output
array([47949.00923077, 74091.09775   , 50733.9026087 , 58736.09431579,
       50733.9026087 , 35416.87659574, 74091.09775   , 31097.668     ,
       58736.09431579, 82908.24257143, 87584.96054237, 87584.96054237,
       51992.32725   , 82908.24257143, 66085.6268    , 75706.38973585,
       74091.09775   , 58736.09431579, 75706.38973585, 31097.668     ,
       31097.668     , 97753.359     , 82908.24257143, 74091.09775   ,
       47949.00923077, 40050.52      , 35416.87659574, 58736.09431579,
       35416.87659574, 87584.96054237, 40050.52      , 35416.87659574,
       75706.38973585, 40642.0848    , 35416.87659574, 31097.668     ,
       31097.668     , 35416.87659574, 40642.0848    , 97753.359     ,
```

Quick Recap

CART in Python	• **`DecisionTreeClassifier`** and **`DecisionTreeRegressor`** from **`sklearn.tree`** library are used for classification and regression respectively. • **`plot_tree`** from **`sklearn.tree`** library is used for plotting decision tree.

Random Forest Method I

Learn How Ensemble Learning Can be Used for Predictive Modeling

Contents

Bootstrapping

Bootstrapping is a method for estimating the sampling distribution of an estimator by resampling with replacement from the original sample

- The method is especially useful in situations where sampling distribution of estimator is not standard distribution.
- The method can be used in any statistical inference problem.
- The use of the term 'bootstrap' comes from the phrase “To pull oneself up by one's bootstraps ” - generally interpreted as succeeding in spite of limited resources.

Bootstrapping

- Many conventional statistical methods of analysis make assumptions about normality, including correlation, regression, t tests, and analysis of variance. When these assumptions are violated, such methods may fail.

- Bootstrapping, a data-based simulation method, is steadily becoming more popular as a statistical methodology. It is intended to simplify the calculation of statistical inferences, sometimes in situations where no analytical answer can be obtained.

- As computer processors become faster and more powerful, the time and effort required for bootstrapping decreases to levels where it becomes a viable alternative to standard parametric techniques.

Bootstrapping

Suppose the original sample is 12, 23, 11, 29, 34, 38, 41, 45, 6

Median = 29.00

We now draw multiple random samples using Bootstrapping

	Sample 1	Sample 2	Sample 3			Sample B
	23	11	6			41
	23	29	45			45
	29	11	11			11
	29	29	29			34
	34	34	11			34
	38	34	38			38
	41	41	41			41
	45	45	41			45
	41	11	6			6
Median	34.00	29.00	29.00			38.00

Sampling distribution of sample median is generated

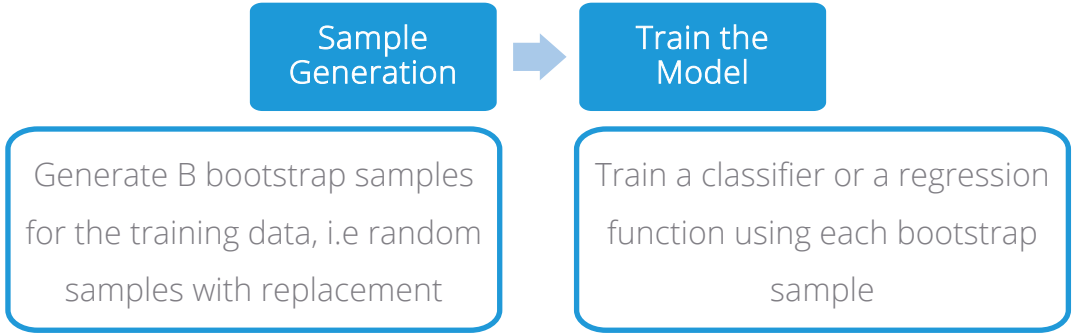
Assuming B=1000, **25th** value and **975th** value will provide 95% confidence interval for median

Bagging

- The term “Bagging” was Introduced by Breiman (1996).
- “Bagging” stands for “Bootstrap Aggregating”.
- It is an ensemble method: a method of combining results from multiple resamples.
- Ensemble method can also be applied by using different classifiers for a given sample.

Bagging Method Framework

Bagging has two basic steps:

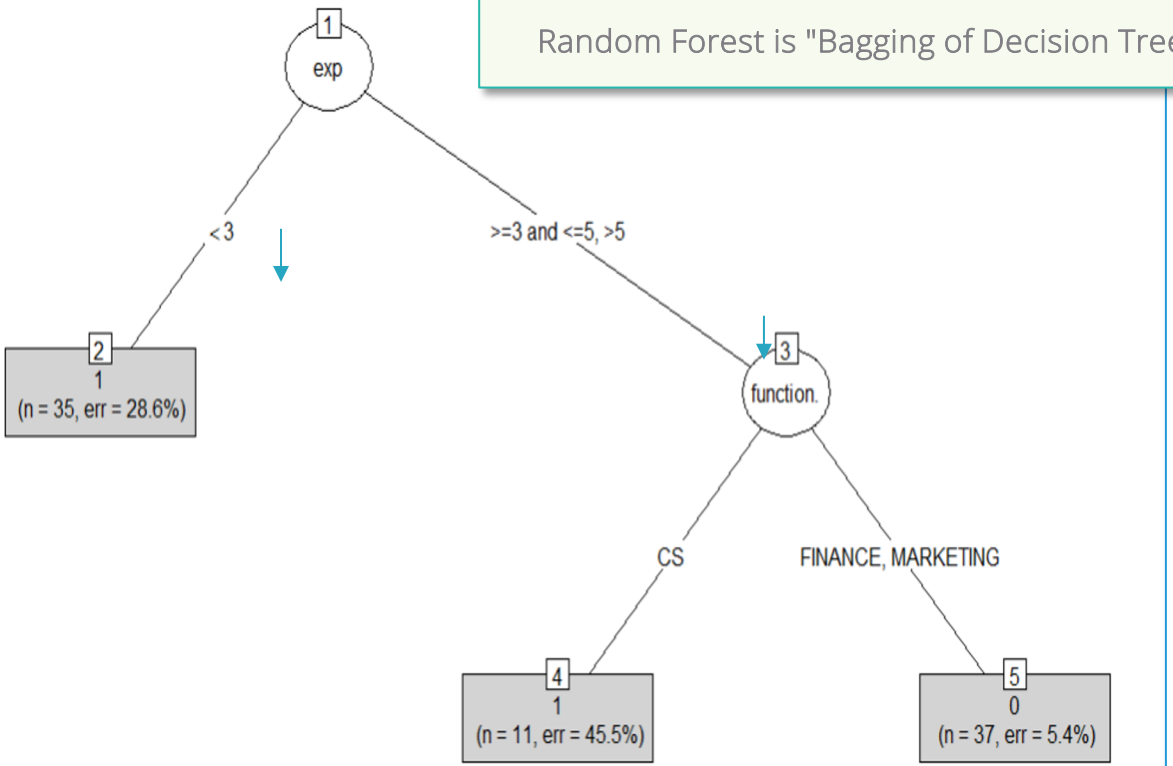


Model for **classification** Majority vote on the classification

Model for **regression** Average of the predicted value

Bagging improves performance for unstable classifiers which vary significantly with small changes in the data set

Re-look at the CHAID Decision Tree



Random Forest Classifier

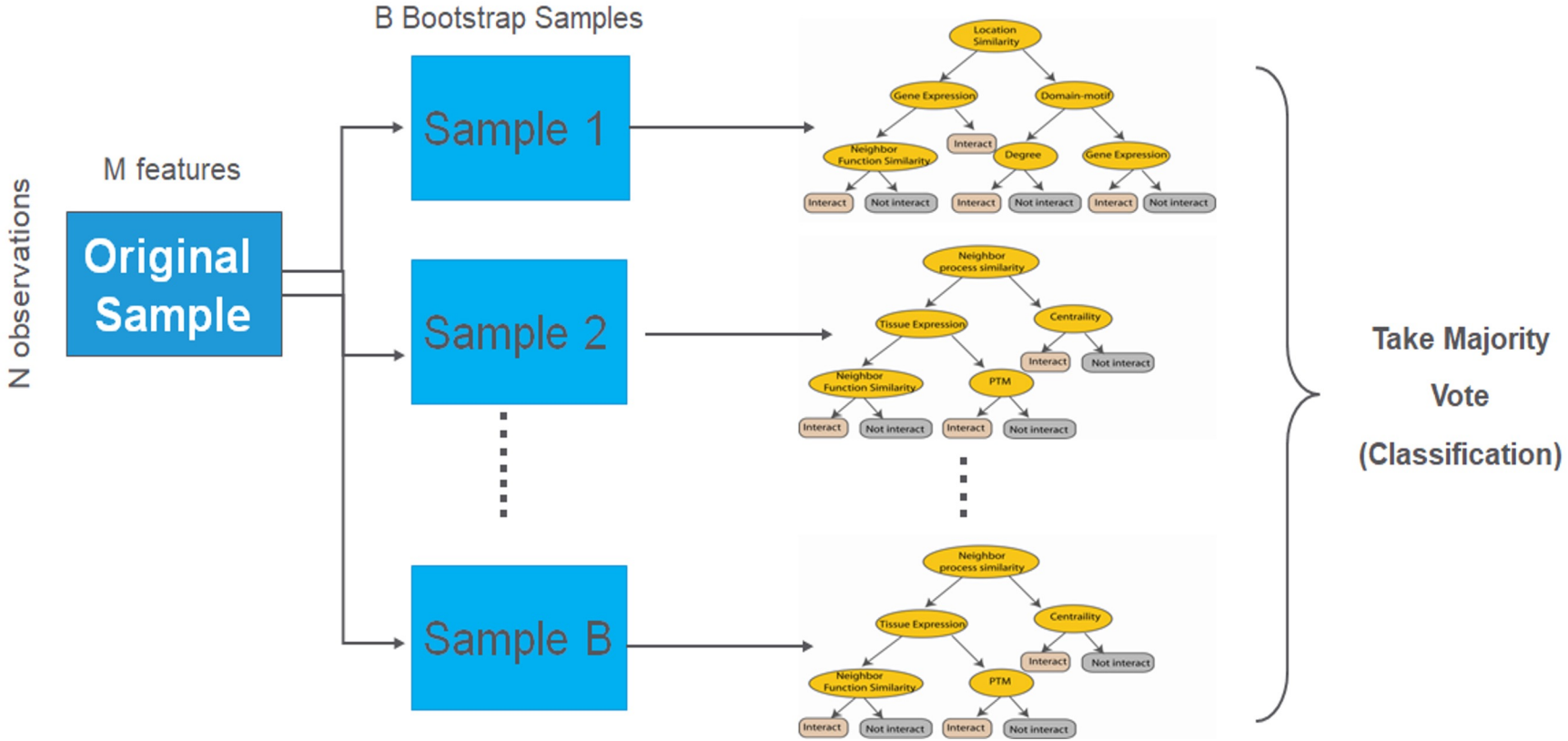
Random Forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.

- The term came from random decision forests that was first proposed by Tin Kam Ho of Bell Labs in 1995.
- The method combines Breiman's “Bagging" idea and the random selection of features.

Random Forest Algorithm

1	Grow a forest of many trees (R default is 500)
2	Grow each tree on an independent **bootstrap sample*** from the training data
3	At each node: • Select *m* variables **at random** out of all *M* possible variables (independently for each node) • Find the best split on the selected *m* variables
4	Grow the trees to maximum depth
5	Vote/average the trees to get predictions for new data

Random Forest Algorithm



Random Forest Algorithm

- In bootstrap samples of data, approx. 2/3 of original values get included in each sample.
- Fit a tree to its greatest depth determining the split at each node through minimizing the loss function considering a random sample of covariates (size is user specified).
- For each tree,
  - Predict classification of the leftover 1/3 using the tree, and calculate the misclassification rate = *Out of Bag (OOB) Error Rate*
  - For each variable in the tree, permute the variables values and compute the OOB error, compare to the original OOB error, the increase is a indication of the variable's importance .

Random Forest Algorithm

- Aggregate OOB error and importance measures from all trees to determine overall OOB error rate and Variable Importance measure

OOB Error Rate

Calculate the overall percentage of misclassification

Variable Importance

Average increase in OOB error over all trees

Quick Recap

Bootstrapping	• Method for estimating the sampling distribution of an estimator by resampling with replacement from the original sample
Bagging	• "Bagging" stands for "Bootstrap Aggregating" • It is an ensemble method: a method of combining results from multiple resamples
Random Forest Method	• Its an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees • Random forests also work for regression problems • The method combines Breiman's "Bagging" idea and the random selection of features

Random Forest Method II

Learn How Ensemble Learning Can be Used for Predictive Modeling

Contents

Case Study – Predicting Loan Defaulters

Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

Objective

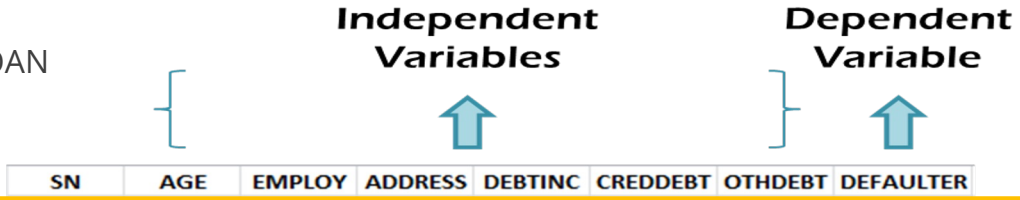
- To predict whether the customer applying for the loan will be a defaulter

Available Information

- Sample size is 700
- **Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts** are the independent variables
- **Defaulter** (=1 if defaulter, 0 otherwise) is the dependent variable

Data Snapshot

BANK LOAN



Column	Description	Type	Measurement	Possible Values
SN	Serial Number	Integer	-	-
AGE	Age Groups	Integer	1(<28 years), 2(28-40 years), 3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Integer	-	Positive value
ADDRESS	Number of years customer staying at current address	Integer	-	Positive value
DEBTINC	Debt to Income Ratio	Continuous	-	Positive value
CREDDEBT	Credit to Debit Ratio	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Integer	1(Defaulter), 0(Non-Defaulter)	2

Random Forest in Python

```
# Import required Libraries
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,precision_score,
recall_score, accuracy_score,roc_curve, roc_auc_score
```

```
# Importing and Readying the Data
```

```
bankloan = pd.read_csv("BANK LOAN.csv")
bankloan1 = bankloan.drop(['SN'], axis = 1)
```

Random Forest in Python

```
# Importing and Readying the Data
bankloan1['AGE'] = bankloan1['AGE'].astype('category')
bankloan1.dtypes
```

- Since it's a classification problem, dependent variable is assigned classes by converting to categorical using **as.type('category').**

```
# Output:
```

AGE	category
EMPLOY	int64
ADDRESS	int64
DEBTINC	float64
CREDDEBT	float64
OTHDEBT	float64
DEFAULTER	int64

- Create dummies **using pd.get_dummies** to convert categorical variable into dummy/indicator variables.

```
bankloan2 = pd.get_dummies(bankloan1)
bankloan2.head()
```

```
# Output:
```

	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER	AGE_1	AGE_2	AGE_3
0	17	12	9.3	11.36	5.01	1	0	0	1
1	10	6	17.3	1.36	4.00	0	1	0	0
2	15	14	5.5	0.86	2.17	0	0	1	0
3	15	14	2.9	2.66	0.82	0	0	0	1
4	2	0	17.3	1.79	3.06	1	1	0	0

Random Forest in Python

```
# Creating Train and Test Data Sets

X = bankloan2.loc[:,bankloan2.columns != 'DEFAULTER']
y = bankloan2.loc[:, 'DEFAULTER']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state = 999)

# Build Random Forest model

rf = RandomForestClassifier(random_state=999, n_estimators=100,
oob_score=True, max_features='sqrt')
rf.fit(X_train, y_train)
```

- **RandomForestClassifier()** performs Random Forest Algorithm
- **random_state=** sets the seed for random sampling
- **n_estimators=** defines the number of trees in the forest.
- **oob_score=** defines whether to use out-of-bag samples to estimate the generalization accuracy.
- **max_features=** defines the number of features to consider when looking for the best split: If "auto", then max_features=sqrt(n_features). If "sqrt", then max_features=sqrt(n_features) (same as "auto"). If "log2", then max_features=log2(n_features). If None, then max_features=n_features.

Note : Since the samples are generated randomly, the outputs will vary slightly for different devices.

Random Forest in Python – Prediction

```
# Calculating Predictions for the model
```

```
y_pred = rf.predict(X_test)
y_pred_probs = rf.predict_proba(X_test)

cutoff = 0.3
pred_test = np.where(y_pred_probs[:,1] > cutoff, 1, 0)
pred_test
```

```
# Output
```

```
array([0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1])
```

Random Forest in Python – Confusion Matrix

```
# Confusion Matrix
confusion_matrix(y_test, pred_test, labels=[0, 1])

array([[127,  30],
       [ 17,  36]], dtype=int64)
accuracy_score(y_test, pred_test)
0.7761904761904762
precision_score(y_test, pred_test)
0.5454545454545454
recall_score(y_test, pred_test)
0.6792452830188679
```

- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as 'Sensitivity' = true positives / (true positives + false negatives)

```
# Area Under ROC Curve
auc = roc_auc_score(y_test, y_pred_probs[:,1])
print('AUC: %.3f' % auc)
AUC: 0.852
```

Random Forest in Python – ROC Curve

- **oob_score_** gives out of bag accuracy
- **feature_importances_** gives the feature importances

```
# OOB Score
rf.oob_score_
0.753061224489796
```

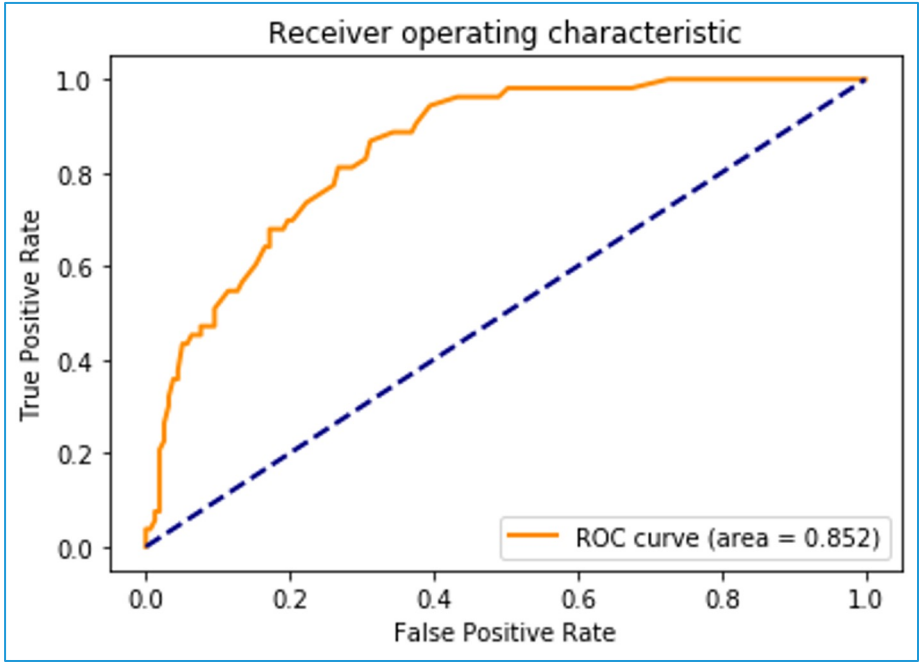
```
rf.feature_importances_
array([0.18827389, 0.14472019, 0.23581877, 0.20153387, 0.18166248,
       0.0233408 , 0.01439653, 0.01025348])
```

```
# ROC Curve
RFfpr, RFtpr, thresholds = roc_curve(y_test, y_pred_probs[:,1])

# plot the roc curve for the model
plt.figure()
lw = 2
plt.plot(RFfpr, RFtpr, color='darkorange',lw=lw, label='ROC curve
(area = %0.3f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.axis('tight')
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

Random Forest in Python – ROC Curve

```
# Output:
```



Random Forest in Python – Variable Importance

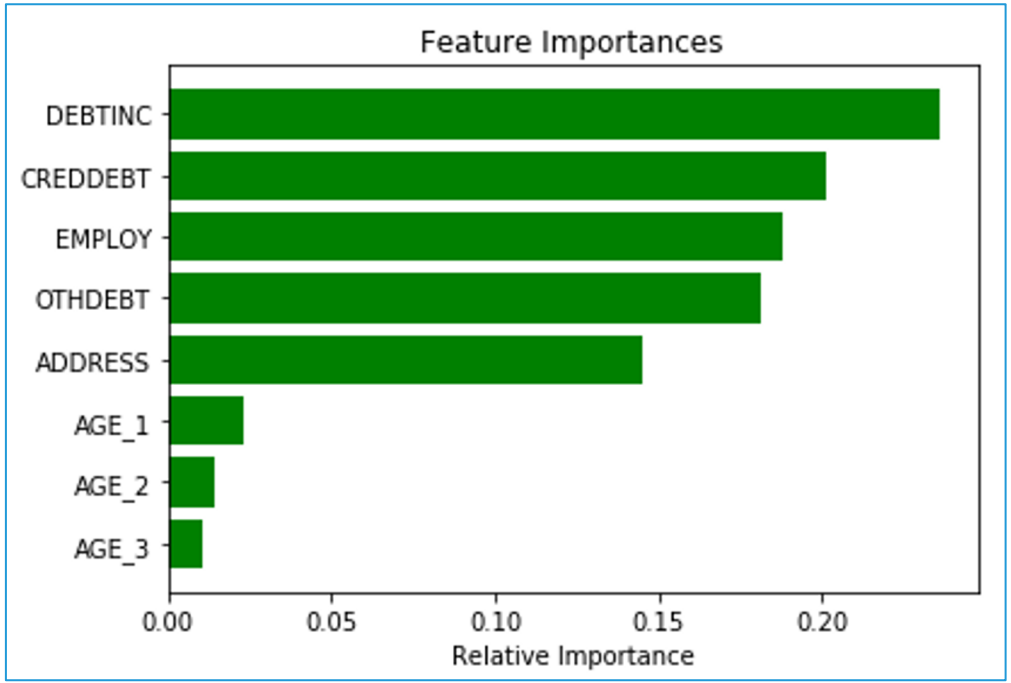
```
# Importance Matrix
features = list(X.columns)
importances = rf.feature_importances_
indices = np.argsort(importances)


plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='g',
align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show();
```

- **argsort()** is used to sort along the given axis, here it being '**importances**'

Random Forest in Python – Variable Importance

```
# Output
```



Quick Recap

Bootstrapping	• Method for estimating the sampling distribution of an estimator by resampling with replacement from the original sample
Bagging	• "Bagging" stands for "Bootstrap Aggregating" • It is an ensemble method: a method of combining results from multiple resamples
Random Forest Method	• Its an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees • Random forests also work for regression problems • The method combines Breiman's "Bagging" idea and the random selection of features
Random Forest in Python	• **`RandomForestClassifier()`** in library **"`sklearn`"** runs random forest analysis • The output can even generate variable importance and can be used for predictions.

Market Basket Analysis

Contents

About Association Rules

Associati on Rule Learning

Method for discovering interesting relations between variables in large databases

- Based on the **concept of strong rules**, Rakesh Agrawal introduced association rules for **discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets**
- For example, the rule found in the sales data of a supermarket would indicate that **if a customer buys onions and potatoes together, they are also likely to buy burger**
- Association rule learning method can be applied in many areas such as **web usage mining, fraud detection, continuous production and bioinformatics**

Introduction to Market Basket Analysis

- The most widely used area of application for association rules is **Market Basket Analysis**

> Market Basket Analysis (Association Analysis) is a **mathematical modeling technique based upon the theory that if you buy a certain group of items, you are likely to buy another group of items**

- It is used to **analyze the customer purchasing behavior and helps in increasing the sales and maintain inventory** by focusing on the point of sale transaction data

Market Basket Analysis – Uses

Product Building	• Develop combo offers based on products bought together
Optimisation	• Organise and place associated products/categories nearby inside a store
Advertising and Marketing	• Determine the layout of the catalog of an ecommerce site
Inventory Management	• Control inventory based on product demands and what products sell together

Definitions and Terminology

Term	Definition
Transactions	A set of items (Item set)
Support	Ratio of **number of times two or more items occur together** to **the total number of transactions** Support can be thought of as P(A and B)
Confidence	Conditional probability that **a randomly selected transaction will include Item B given Item A** **P(B | A)** (written as A => B)
Lift	Ratio of the **probability of Items A and B occurring together (Joint probability)** to **the product of P(A) and P(B)**

Get an Edge!

The Famous Story

An article in The Financial Times of London (Feb. 7, 1996) stated,

"The example of what data mining can achieve is the case of a large US supermarket chain which discovered a strong association for many customers between a brand of babies nappies (diapers) and a brand of beer. Most customers who bought the nappies also bought the beer. The best hypothesisers in the world would find it difficult to propose this combination but data mining showed it existed, and the retail outlet was able to exploit it by moving the products closer together on the shelves."

Rule Evaluation – Support

Transaction No.	Item 1	Item 2	Item 3	…
100	Beer	Diaper	Chocolate	
101	Milk	Chocolate	Shampoo	
102	Beer	Wine	Vodka	
103	Beer	Cheese	Diaper	
104	Ice Cream	Diaper	Beer	

A B

Support of {Diaper, Beer}

$$\text{Support} = \frac{\text{No.of transactions containing both A and B}}{\text{Total no.of transactions}} = \frac{3}{5} = 60\%$$

Support of {Diaper, Beer} is 3/5

Rule Evaluation – Confidence

Transaction No.	Item 1	Item 2	Item 3	…
100	Beer	Diaper	Chocolate	
101	Milk	Chocolate	Shampoo	
102	Beer	Wine	Vodka	
103	Beer	Cheese	Diaper	
104	Ice Cream	Diaper	Beer	

$$\text{Confidence for } \{A\} \Rightarrow \{B\} = \frac{\text{No. of transactions containing both A and B}}{\text{No. of transactions containing A}}$$

Confidence for {Diaper} ⇒ {Beer} is 3/3

When Diaper is purchased, the likelihood of Beer purchase is 100%

Confidence for {Beer} ⇒ {Diaper} is 3/4

When Beer is purchased, the likelihood of Diaper purchase is 75%

{Diaper} ⇒ {Beer} is a more important rule according to Confidence

Rule Evaluation – Lift

Transaction No.	Item 1	Item 2	Item 3	Item 4
100	Beer	Diaper	Chocolate	
101	Milk	Chocolate	Shampoo	
102	Beer	Milk	Vodka	Chocolate
103	Beer	Milk	Diaper	Chocolate
104	Milk	Diaper	Beer	

A B

Consider {Chocolate} ⇒ {Milk}

$$\text{Lift} = \frac{P(A \cap B)}{P(A)P(B)} = \frac{3/5}{(4/5)(4/5)} = 0.9375$$

Lift < 1 indicates Chocolate is decreasing the chance of Milk purchase
Support and confidence are high but lift is low

Case Study – Online Retail Data

Background

- A typical retail transactional data from a UK retailer from 2010-11

Objective

- To mine association rules and information about item sets

Available Information

- Total number of transactions is 541909
- Items are aggregated to 392 categories
- Data is collected for 1 year (365 days)

Data Snapshot

ONLINE RETAIL

Variables

Observations

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	01-12-2010 08:26	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 08:26	2.75	17850	United Kingdom

Column	Description	Type	Measurement	Possible Values
InvoiceNo	Invoice Number	Numeric	-	-
StockCode	Stock Code	Categorical	-	-
Description	Product Description	Character	WHITE HANGING HEART T-LIGHT HOLDER, etc	-
Quantity	Quantity	Continuous	-	Positive and Negative value
InvoiceDate	Date of Invoice	Date	dd-mm-yyyy hh:mm	01/12/2010 8:26 to 09/12/2011 12:50
UnitPrice	Price per unit of product	Continuous	-	Positive and Negative value
CustomerID	Customer ID	Continuous	-	-
Country	Country name	Categorical	United Kingdom, France, etc	

Market Basket Analysis in Python

```
#Market Basket Analysis Using Apriori Recommendation
```

```
pip install mlxtend
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
df = pd.read_excel('Online Retail.xlsx')
df.head()
```

- *We will be using library "**mlxtend**" for performing Market Basket Analysis in Python.*
- *Library **"mlxtend"** is used for extracting frequent itemsets with applications in association rule learning*

*

Data Source : Dr Daqing Chen, Director: Public Analytics group. chend '@' lsbu.ac.uk, School of Engineering, London South Bank University, London SE1 0AA, UK, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science

Market Basket Analysis in Python

```
# Output:
```

Index	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 8.26	2.55	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 8.26	3.39	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 8.26	2.75	17850	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 8.26	3.39	17850	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 8.26	3.39	17850	United Kingdom

Visualise Item Frequency

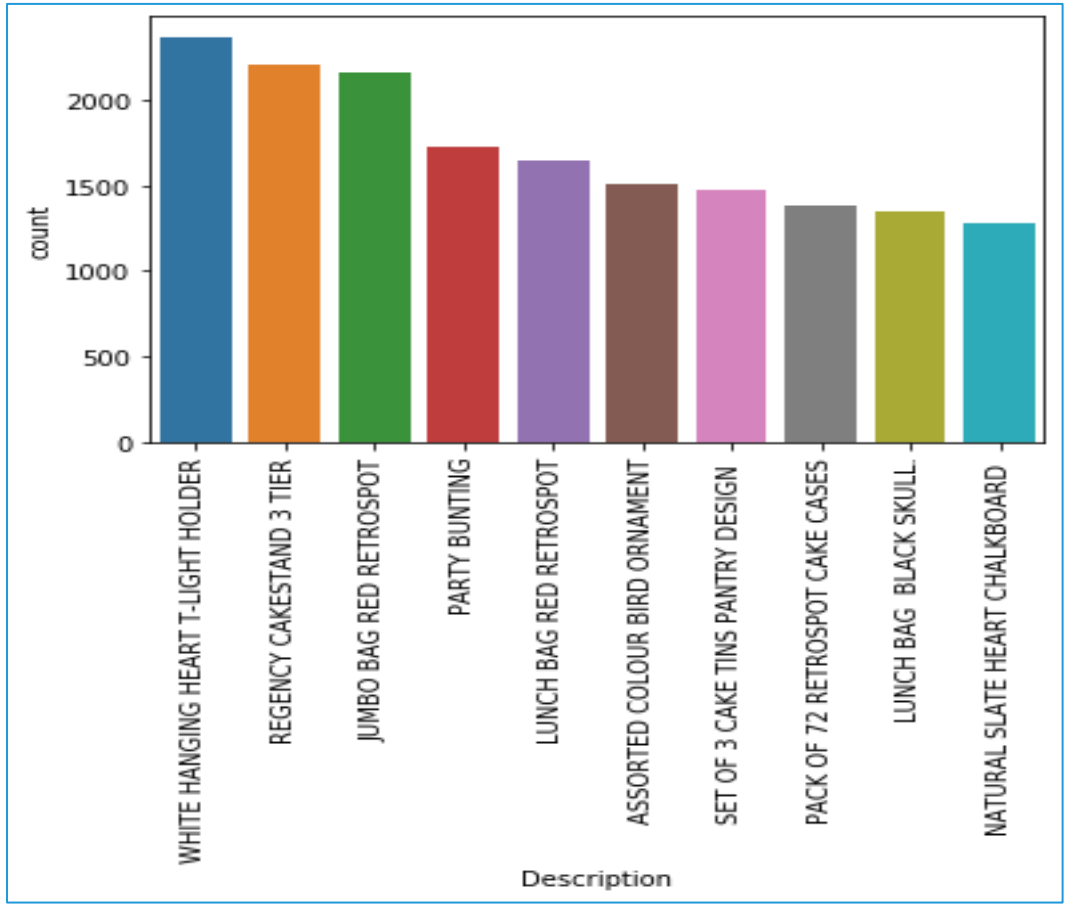
```
#Item Frequency Plot
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x = 'Description', data = df, order =
df['Description'].value_counts().iloc[:10].index)
plt.xticks(rotation=90)
```

- ***sns.countplot()*** *calculates item frequency and returns a barplot.*
- ***order*** *= Order to plot the categorical levels in, otherwise the levels are inferred from the data objects*

Item Frequency Plot

```
# Output
```



Interpretation:

The plot shows items by frequency in a descending order.

Basic Data Cleanup

```
# Data Cleaning and Consolidation
```

```
df['Description'] = df['Description'].str.strip()
df.dropna(axis=0, subset=['InvoiceNo'], inplace=True)
df['InvoiceNo'] = df['InvoiceNo'].astype('str')
df = df[~df['InvoiceNo'].str.contains('C')]
```

- ***strip()*** *returns a copy of the string with both leading and trailing characters removed .*
- ***dropna()*** *removes all the missing values and a new object is returned which does not have any NaN values present in it.*
- ***contains()*** *function is used to test if pattern or regex is contained within a string of a Series or Index. Here it is used to remove 'C' from 'InvoiceNo.*

```
basket = (df[df['Country'] =="France"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))
basket.head()
```

- *After the cleanup, consolidation of the items into 1 transaction per row with each product is done.*

Basic Data Cleanup

Output:

Description	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 EGG HOUSE PAINTED WOOD	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETRO POT	12 PENCILS SMALL TUBE SKULL	12 PENCILS TALL TUBE POSY
InvoiceNo								
536370	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
536852	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
536974	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
537065	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
537463	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Consolidation of items

```
# Data consolidation
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket.applymap(encode_units)
basket_sets.drop('POSTAGE', inplace=True, axis=1)
```

- ***applymap()*** *method applies a function that accepts and returns a scalar to every element of a DataFrame.*
- *This way, we generated a data frame that shows us whether a particular items is bought or not.*

```
frequent_itemsets = apriori(basket_sets, min_support=0.07,
use_colnames=True)
```

- *Once data is structured properly, frequent item sets that have a support of at least 7% is generated.*

Get and Display the Rules

```
#Get the Rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

- ***association_rules()*** *generate the rules with their corresponding support, confidence and lift.*

```
#Show Top 5 Rules
rules.head()
```

```
# Output:
```

Index	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	frozenset({'ALARM CLOCK BAKELIKE PINK'})	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	0.102040816	0.096938776	0.073979592	0.725	7.478947368	0.06408788	3.283858998
1	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	frozenset({'ALARM CLOCK BAKELIKE PINK'})	0.096938776	0.102040816	0.073979592	0.763157895	7.478947368	0.06408788	3.79138322
2	frozenset({'ALARM CLOCK BAKELIKE RED'})	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	0.094387755	0.096938776	0.079081633	0.837837838	8.642958748	0.069931799	5.568877551
3	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	frozenset({'ALARM CLOCK BAKELIKE RED'})	0.096938776	0.094387755	0.079081633	0.815789474	8.642958748	0.069931799	4.916180758
4	frozenset({'POSTAGE'})	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	0.765306122	0.096938776	0.084183673	0.11	1.134736842	0.009995835	1.014675533

Manage How the Rules are Displayed

```
#Sort the Rules
rules[ (rules['lift'] >= 6) &
       (rules['confidence'] >= 0.8) ]
```

- *Dataframe can be filtered using standard pandas code. In this case, rules with high lift (>6) and high confidence (>8) are displayed.*

```
# Output:
```

Index	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	frozenset({'ALARM CLOCK BAKELIKE RED'})	0.0969388	0.0943878	0.0790816	0.8157895	8.6429587	0.0699318	4.9161808
3	frozenset({'ALARM CLOCK BAKELIKE RED'})	frozenset({'ALARM CLOCK BAKELIKE GREEN'})	0.0943878	0.0969388	0.0790816	0.8378378	8.6429587	0.0699318	5.5688776
17	frozenset({'SET/6 RED SPOTTY PAPER PLATES'})	frozenset({'SET/2 0 RED RETROSPOT PAPER NAPKINS'})	0.127551	0.1326531	0.1020408	0.8	6.0307692	0.0851208	4.3367347

Interpretation:

- *Green and red alarm clocks are purchased together and the red paper cups, napkins and plates are purchased together in a manner that is higher than the overall probability would suggest*

Manage How the Rules are Displayed

```
basket['ALARM CLOCK BAKELIKE GREEN'].sum()
340.0
basket['ALARM CLOCK BAKELIKE RED'].sum()
316.0
```

- *In order to check how much opportunity is there to use the popularity of one product to drive sales of another, their sum is calculated.*
- *For example, it can be seen that 340 Green Alarm clocks are sold but only 316 Red Alarm clocks are sold, hence maybe selling of Red Alarm Clock can be increased through recommendations*

Combinations by country

```
basket2 = (df[df['Country'] =="Germany"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))
basket_sets2 = basket2.applymap(encode_units)
basket_sets2.drop('POSTAGE', inplace=True, axis=1)
frequent_itemsets2 = apriori(basket_sets2, min_support=0.05,
use_colnames=True)
rules2 = association_rules(frequent_itemsets2, metric="lift",
min_threshold=1)

rules2[ (rules2['lift'] >= 4) &
        (rules2['confidence'] >= 0.5)]
```

- *It is interesting to see how the combinations vary by country of purchase.*
- *Here, some popular combinations in Germany are displayed*

Combinations by country

```
# Output:
```

Index	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1	frozenset({'PLASTERS IN TIN CIRCUS PARADE'})	frozenset({'PLASTERS IN TIN WOODLAND ANIMALS'})	0.1159737	0.1378556	0.067833698	0.58490566	4.242887092	0.051846071	2.076984285
7	frozenset({'PLASTERS IN TIN SPACEBOY'})	frozenset({'PLASTERS IN TIN WOODLAND ANIMALS'})	0.107221	0.1378556	0.061269147	0.571428571	4.145124717	0.046488133	2.011670314
11	frozenset({'RED RETROSPOT CHARLOTTE BAG'})	frozenset({'WOODLAND CHARLOTTE BAG'})	0.0700219	0.1269147	0.059080963	0.84375	6.648168103	0.050194159	5.587746171

Interpretation :

- *It can be inferred that Germans like Plasters in Tin Spaceboy and Woodland Animals.*

Quick Recap

In this session, we learnt **Market Basket Analysis:**

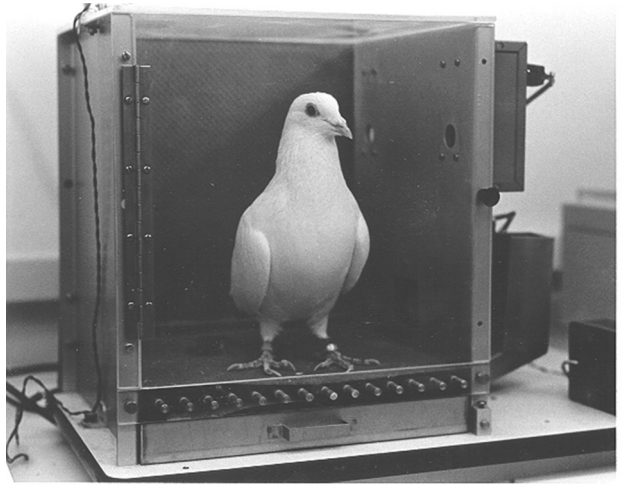
Market Basket Analysis	• Mathematical modeling technique based upon the theory that if you buy a certain group of items, you are likely to buy another group of items • Transactions, Support, Confidence and Lift are the key concepts used in this analysis • The analysis is performed by creating and studying rules based on different itemsets
Market Basket Analysis in Python	• Library **`mlxtend`** is used for undertaking MBA in Python • **`sns.countplot()`** plots frequency • **`apriori()`** builds frequent items • **`association_rules()`** builds the rules

Artificial Neural Networks Introduction (Python)

Contents

Pigeons as Art Experts

- Experiment:
- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist (e.g. Van Gogh)



- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (When presented with pictures they had been trained on)
- Discrimination still 85% successful for previously unseen paintings of the artists.

Pigeons as Art Experts

Pigeons do not simply memorize the pictures

They can extract and recognise patterns (the ‘style’)

They generalize from the already seen to make predictions

This is what neural networks (biological and artificial) are good at (unlike conventional computer).

Neural Networks Introduction

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.

- Each neuron in ANN receives a number of inputs.

- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).

How do our brains work?

The Brain is a massively parallel information processing system. Our brains are a huge network of processing elements. A typical brain contains a network of 10 billion neurons.

Features of Neural Networks

- Massive connectivity
- Nonlinear, Parallel, Robust and Fault Tolerant
- Capability to adapt to surroundings
- Ability to learn and generalize from known examples
- Collective behaviour is different from individual behaviour

- Artificial Neural Networks mimics some of the properties of the biological neural networks

Why Neural Networks?

- There are two basic reasons why we are interested in building artificial neural networks (ANNs):

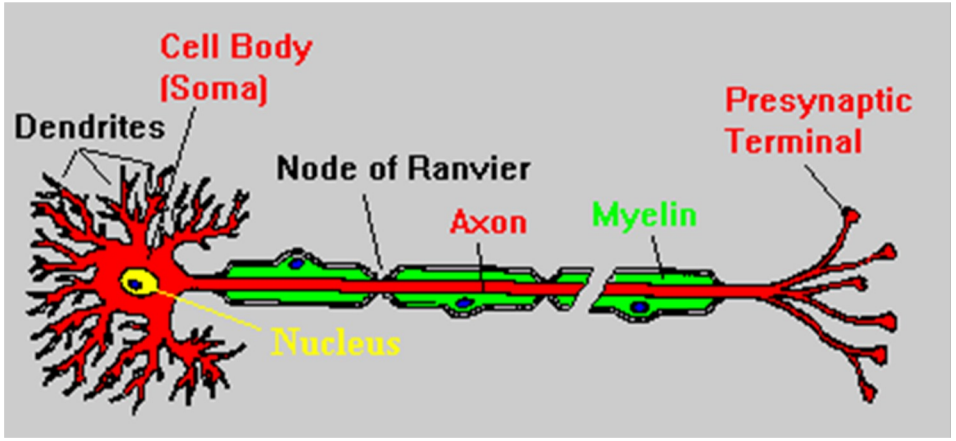
- Technical viewpoint:
- Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing.

- Biological viewpoint:
- ANNs can be used to replicate and simulate components of the human (or animal) brain, thereby giving us insight into natural information processing.

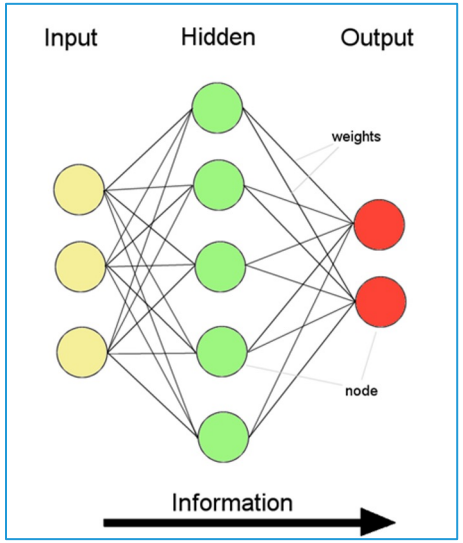
Biological Neuron vs Artificial Neuron (Perceptron)

- Information flow is unidirectional
- Data is presented to Input layer
- Passed on to Hidden Layer
- Passed on to Output layer
- Information is distributed
- Information processing is parallel

Biological Neuron



Network function f: R3 {0, 1}2



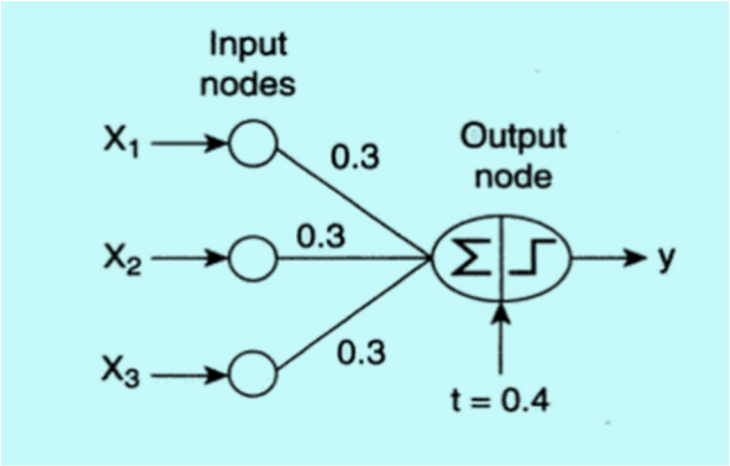
Perceptron Model

- Consider data with 3 Boolean variables X1, X2, X3 and an output variable y

Data

X1	X2	X3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	0	-1
0	0	1	-1
0	1	0	-1
0	1	1	1

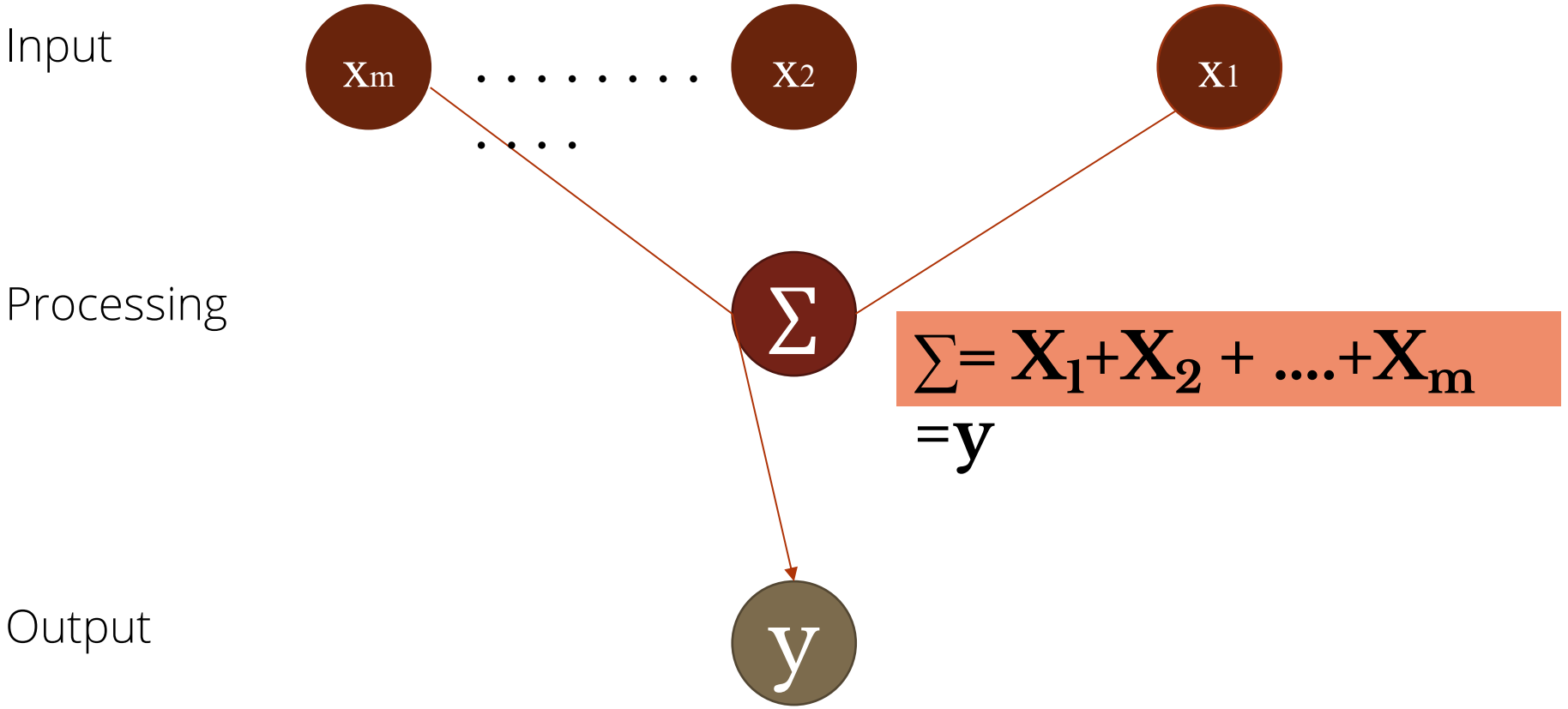
Perceptron



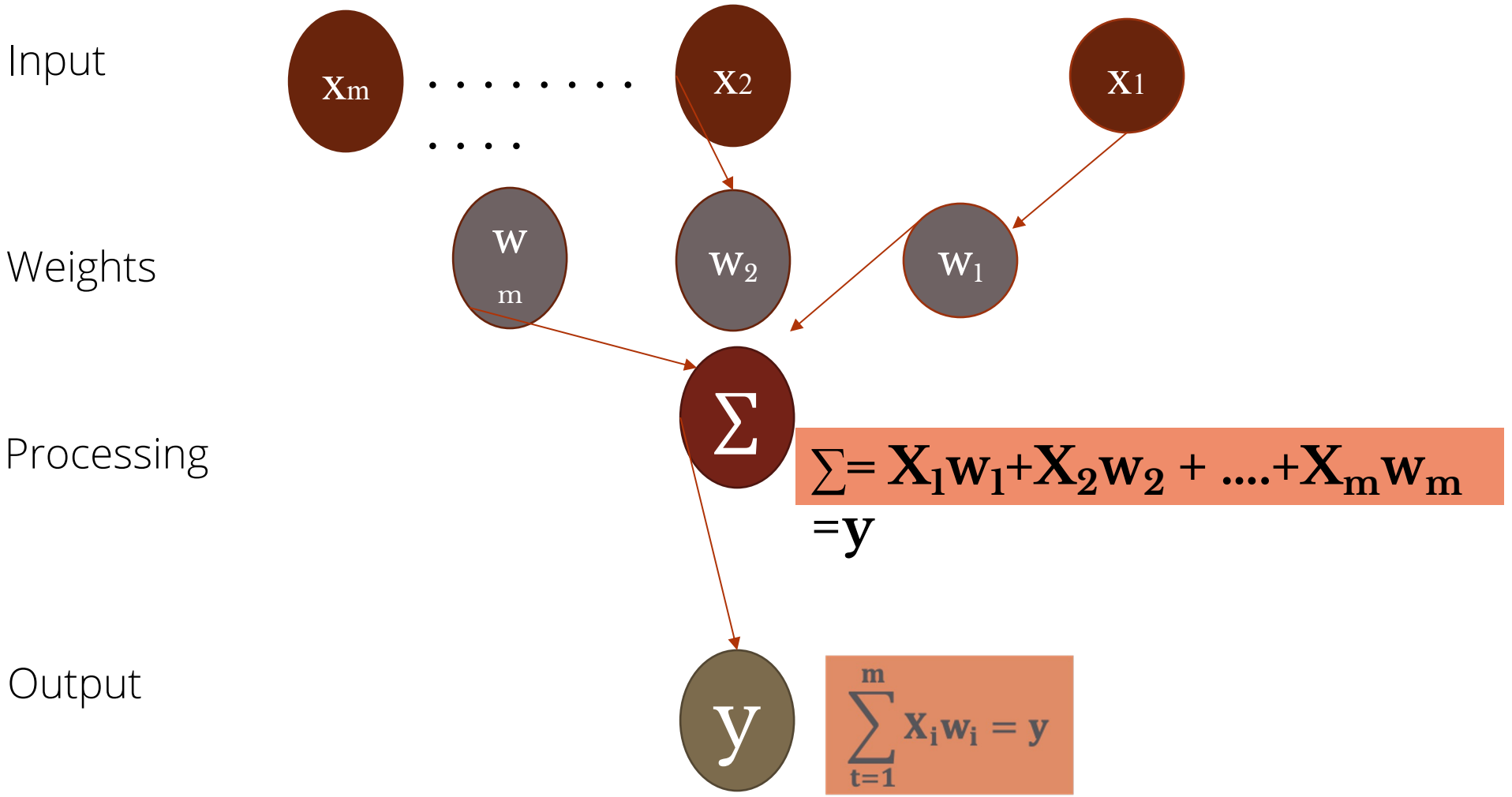
$$y \begin{cases} -1, & \text{if } 0.3X1 + 0.3X2 + 0.3X3 - 0.4 > 0; \\ 1, & \text{if } 0.3X1 + 0.3X2 + 0.3X3 - 0.4 < 0. \end{cases}$$

- Perceptron consists of two types of nodes: input nodes which is used to represent the attributes, and output node which is used to represent the model output.
- Perceptron computes its output value, by performing a weighted sum on its inputs, subtracting a bias factor t from the sum, and then examining the sign of the result.

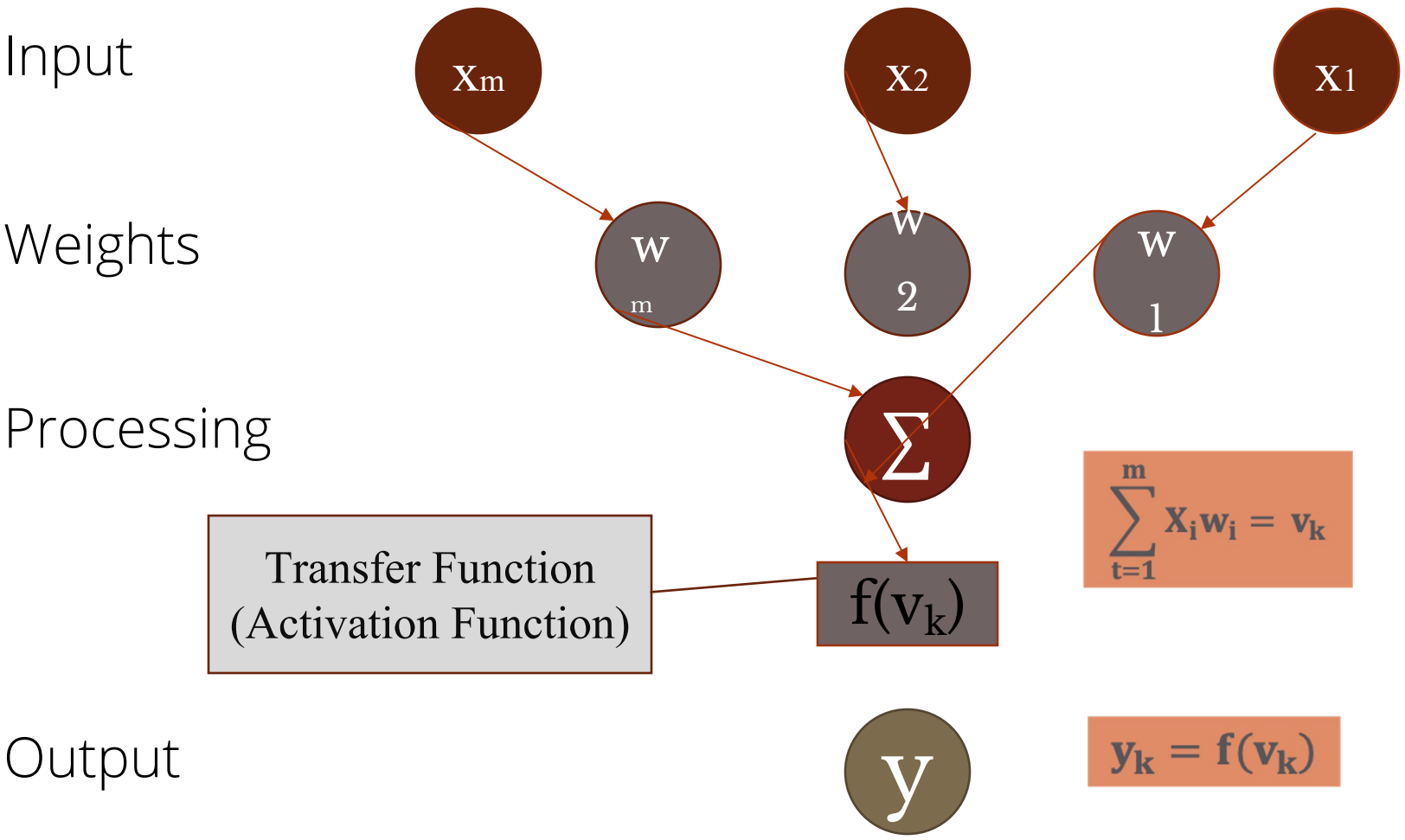
A Single Artificial Neuron…



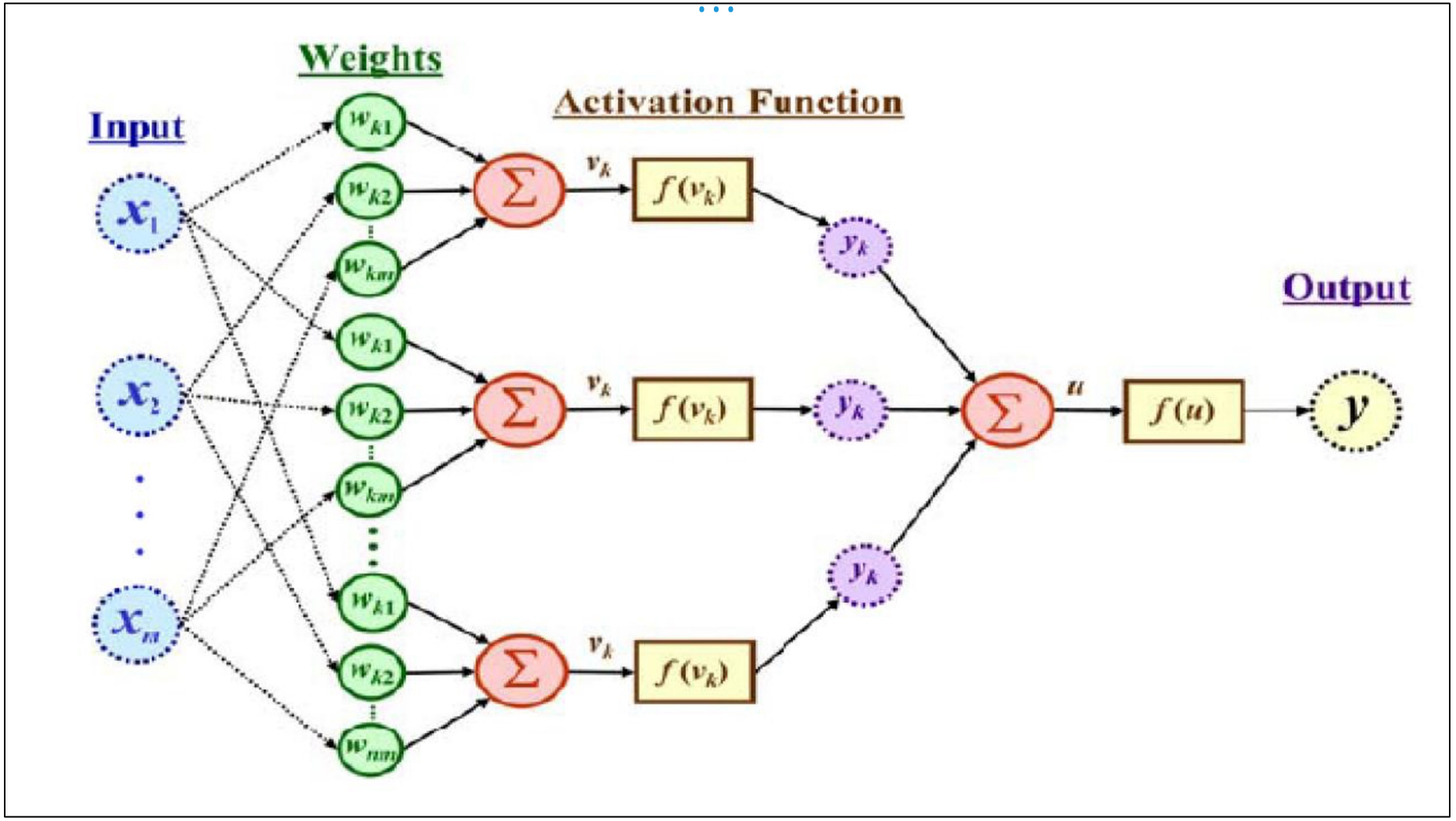
A Single Artificial Neuron…



A Single Artificial Neuron…

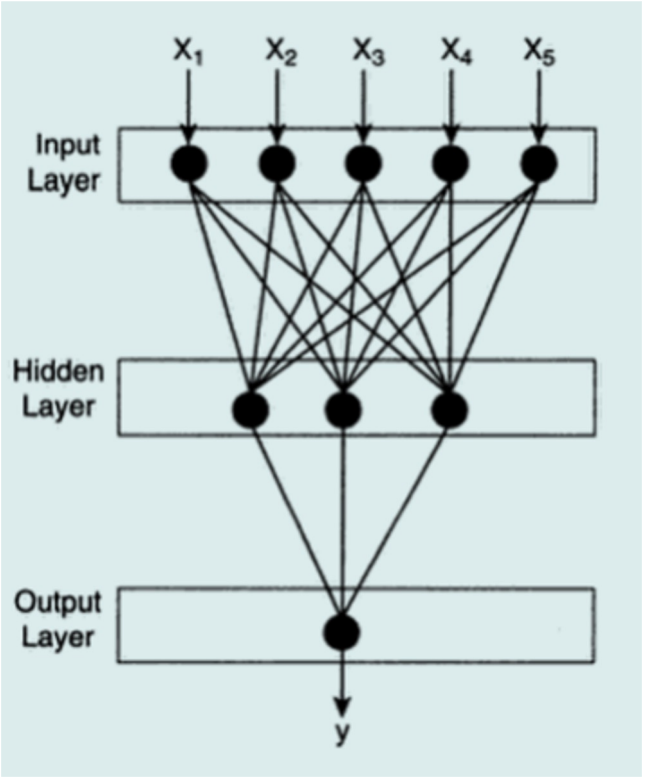


The output is a function of the input, that is affected by the weights, and the transfer functions



Artificial Neural Network (ANN)

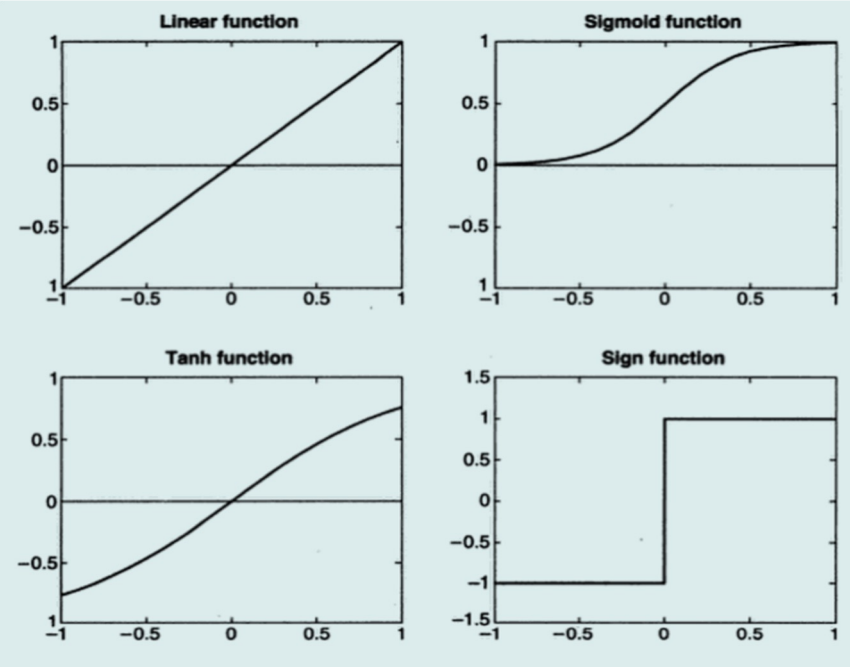
- A neural network is more complex than a perceptron model in multiple ways.
- Network may contain several intermediary layers between its input and output layers which are called hidden layers and the nodes embedded in these layers are called hidden nodes. This restructuring is known as a multilayer neural network.
- In a feed forward network, nodes in one layer are connected to nodes in the next layer.
- In a recurrent network, the links may connect nodes within the same layer or from one layer to the previous layers.



Types of ANN

- The artificial neural network may use types of activation functions other than sign function. Examples of other activation function include linear, sigmoid, and hyperbolic tangent functions.

- These additional complexities allow multilayer neural networks to model more complex relationships between the input and output variables.

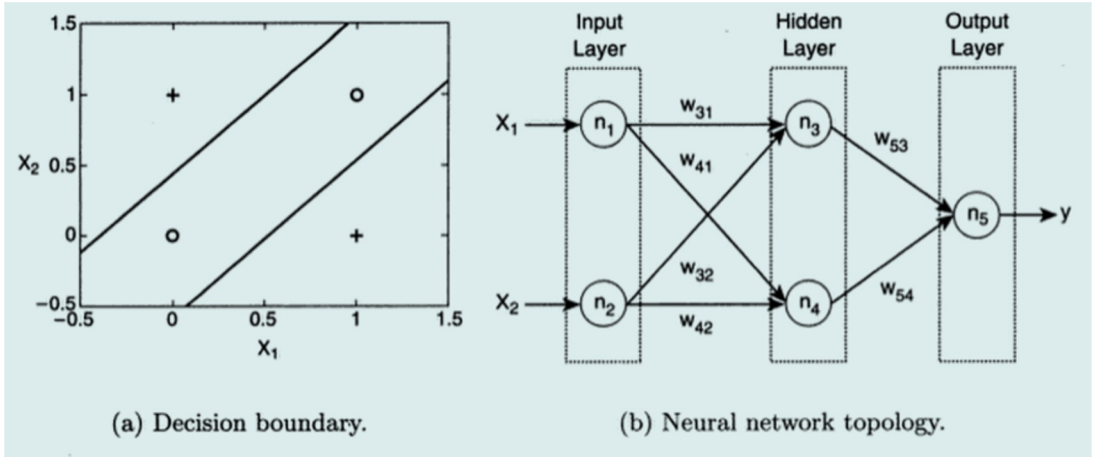


Learning ANN

- In ANN, we determine set of weights w that minimize total sum of squared errors.

$$E(w) = \frac{1}{2}\sum_{i-1}^{N}(y_i - \hat{y}_i)^2$$

- Minimum solution can be obtained by replacing $\hat{y}=w.x$ in the above equation.
- In most cases, output of ANN is non linear function of its parameters, because of the choice of its activation.
- To arrive at a global optimal solution in non linear models, we use gradient descent method to efficiently solve optimization problem.



(a) Decision boundary. (b) Neural network topology.

Gradient Descent Method

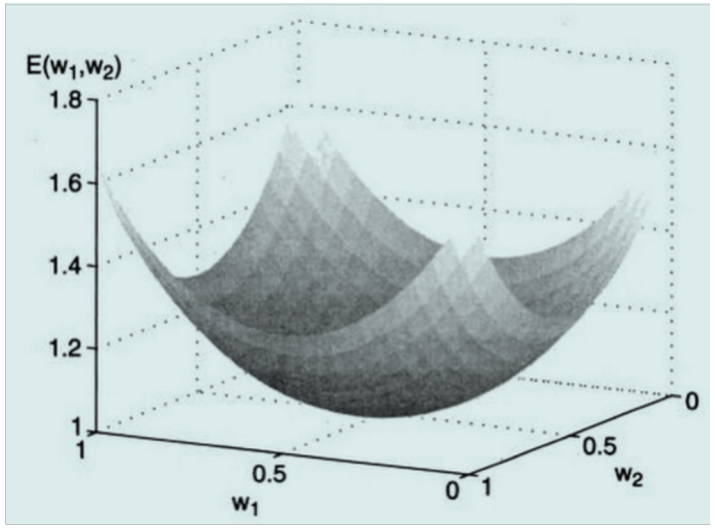
- The gradient descent algorithm can be written as:

$$w_j \longleftarrow w_j - \lambda \frac{\partial E(w)}{\partial w_j}$$

- λ is the learning rate, while the second term states that the weight should be increased in a direction that reduces the overall error term

- Each w is initialized to random small value and the iterations are performed until local minima is obtained

- Converging to a local minimum can be very slow or fast depending on the size of learning rate. The weights of the output and hidden



Backpropagation

- In case of hidden nodes, computation becomes complex and error term cannot be assessed without prior information of weights. Back propagation method can be used which works in 2 phases – forward and backward.
- During forward phase, we compute output value of each neuron in the network using previous iteration weight. The computation progresses in the forward direction, outputs at the neuron k are computed prior to computing the outputs at level k+1.
- In backward phase, weight update formula is applied in the reverse direction, i.e. weights at level k+1 are updated before weights at level k.
- This backward propagation approach allows us to use errors for neurons at layer k+1 to estimate the errors for neurons at layer k.

Resilient Backpropagation

- Compared to the traditional back propagation algorithm, the Resilient propagation (rprop) algorithm offers faster convergence and is usually more capable of escaping from local minima.
- Rprop is a first-order algorithm and its time and memory requirement scales linearly with the number of parameters.
- Resilient propagation does not take into account the value of the partial derivative (error gradient), but rather considers only the sign of the error gradient to indicate the direction of the weight update.
- In practice, Rprop is easier to implement than BPNN.

Quick Recap

Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- In a feed forward network, nodes in one layer are connected to nodes in the next layer.
- In a recurrent network, the links may connect nodes within the same layer or from one layer to the previous layers.
- In ANN, we determine set of weights w that minimize total sum of squared errors.
- To arrive at a global optimal solution in non linear models, we use gradient descent method to efficiently solve optimization problem.
- In case of hidden nodes, we use backpropagation. The Resilient propagation (rprop) algorithm offers faster convergence than traditional backpropagation.

Building a Neural Network Model Using Python

Contents

Case Study – Predicting Loan Defaulters

Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

Objective

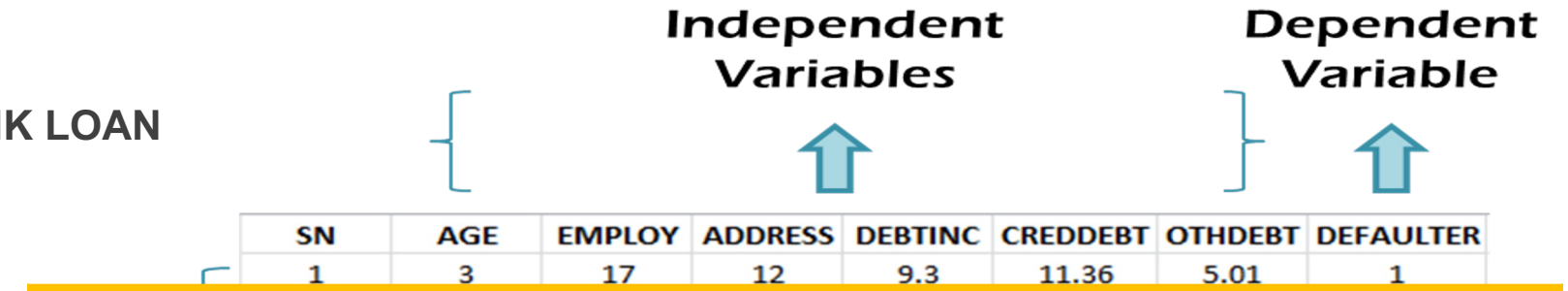
- To predict whether the customer applying for the loan will be a defaulter

Available Information

- Sample size is 700
- Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts are the independent variables
- Defaulter (=1 if defaulter, 0 otherwise) is the dependent variable

Neural Network in Python...Data Snapshot

BANK LOAN



Column	Description	Type	Measurement	Possible Values
SN	Serial Number	-	-	-
AGE	Age Groups	Integer	1(<28 years),2(28-40 years),3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Integer	-	Positive value
ADDRESS	Number of years customer staying at current address	Integer	-	Positive value
DEBTINC	Debt to Income Ratio	Continuous	-	Positive value
CREDDEBT	Credit to Debit Ratio	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Integer	1(Defaulter), 0(Non-Defaulter)	2

Data Pre-Processing

- Since AGE is categorical variable, we create dummy variables before proceeding to neural network model.
- To set up a neural network to a dataset it is very important that we ensure a proper scaling of data. The scaling of data is essential because otherwise, a variable may have a large impact on the prediction variable only because of its scale.
- The common techniques to scale data are min-max normalization and Z-score normalization
- The min-max normalization transforms the data into a common range, thus removing the scaling effect from all the variables. Here we are using min-max normalization for scaling data.

Neural Network in Python - Classifying Loan Defaulters...

```
import os
os.chdir("D:/SANKHYA/Neural Networks")
import pandas as pd
bankloan = pd.read_csv("BANK LOAN.csv")
dummies = pd.get_dummies(bankloan['AGE'],prefix='AGE',drop_first=True)
bankloan = pd.concat([bankloan, dummies], axis=1)

from sklearn.preprocessing import MinMaxScaler
import numpy as np

scaler = MinMaxScaler()
bankloan['EMPLOY']=scaler.fit_transform(np.array(bankloan['EMPLOY']).r
eshape(-1, 1))
bankloan['ADDRESS']=scaler.fit_transform(np.array(bankloan['ADDRESS'])
.reshape(-1, 1))
bankloan['DEBTINC']=scaler.fit_transform(np.array(bankloan['DEBTINC'])
.reshape(-1, 1))
bankloan['CREDDEBT']=scaler.fit_transform(np.array(bankloan['CREDDEBT'
]).reshape(-1, 1))
bankloan['OTHDEBT']=scaler.fit_transform(np.array(bankloan['OTHDEBT'])
.reshape(-1, 1))
```

Neural Network in Python- Classifying Loan Defaulters...

```
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(hidden_layer_sizes=(3,), max_iter=300,
                           activation =
'relu',solver='adam',random_state=1)

from sklearn.model_selection import train_test_split
X = bankloan.loc[:,bankloan.columns != 'DEFAULTER']
y = bankloan.loc[:, 'DEFAULTER']

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30,
                                                    random_state = 999)



classifier.fit(X_train, y_train)
```

Neural Network in Python- Classifying Loan Defaulters…

```
y_pred = classifier.predict_proba(X_test)[0:210,1]

predicted_class=np.zeros(y_pred.shape)
predicted_class[y_pred>0.3]=1
from sklearn.metrics import classification_report
print(classification_report(y_test,predicted_class))
```

Output

```
              precision    recall  f1-score   support

           0       0.77      0.39      0.52       157
           1       0.27      0.66      0.38        53

    accuracy                           0.46       210
   macro avg       0.52      0.52      0.45       210
weighted avg       0.64      0.46      0.48       210
```

More About ANN Work

- Applications in Artificial Intelligence - Handwriting or Face Recognition, Voice Analysis
- The “building blocks” of neural networks are the neurons. In technical systems, we also refer to them as units or nodes.
- Basically, each neuron receives input from many other neurons, changes its internal state (activation) based on the current input, sends one output signal to many other neurons, possibly including its input neurons (recurrent network)
- Information is transmitted as a series of electric impulses, so-called spikes.
- The frequency and phase of these spikes encodes the information.
- In biological systems, one neuron can be connected to as many as 10,000 other neurons.
- Neurons of similar functionality are usually organized in separate areas (or layers).
- Often, there is a hierarchy of interconnected layers with the lowest layer receiving sensory input and neurons in higher layers computing more complex functions.

Quick Recap

Neural Networks in Python	• Library **“sklearn"** has **MLPClassifier ()** that trains a neural network model