Machine Learning

Classification

INTRODUCTION TO CLASSIFICATION PROBLEMS

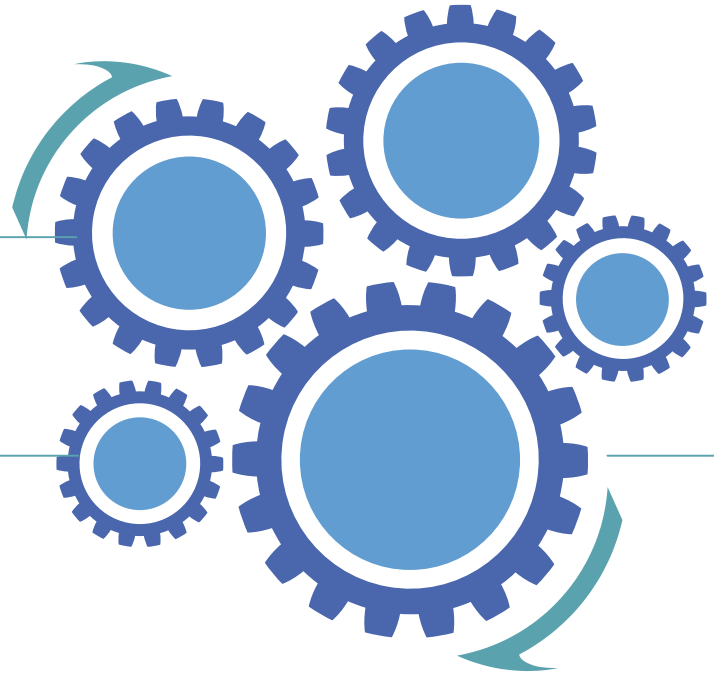
- Regression analysis to predict values that are numerical (discrete or continuous) BASED ON DATA
- In many other cases, we would like to predict a category BASED ON DATA instead of numerical values

  For example, given many details about employees like age, monthly income, department, years of experience etc., an HR would like to predict whether an employee will leave the organization or not.
- In this prediction, the target is a category, i.e. Yes or No type
- In such situations, where we would like to predict categories instead of numerical values, we have to use classification techniques instead of regression techniques

CLASSIFICATION EXAMPLE

Predict whether a customer will default his loan or not

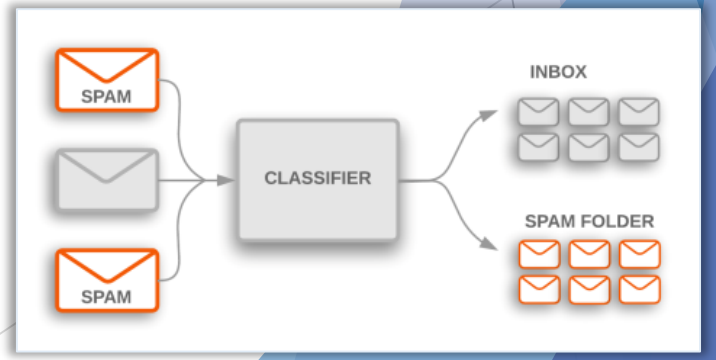
Diagnose a customer for the presence of cancer cells based on various test results



Predict whether a customer will leave the network next or not

Classify images as dog, human being, car, bike etc.

Classification not restricted to structured data alone. Even we can use classification to decide whether an email should be moved to inbox or spam folder



CLASSIFICATION TECHNIQUES

Decision Trees

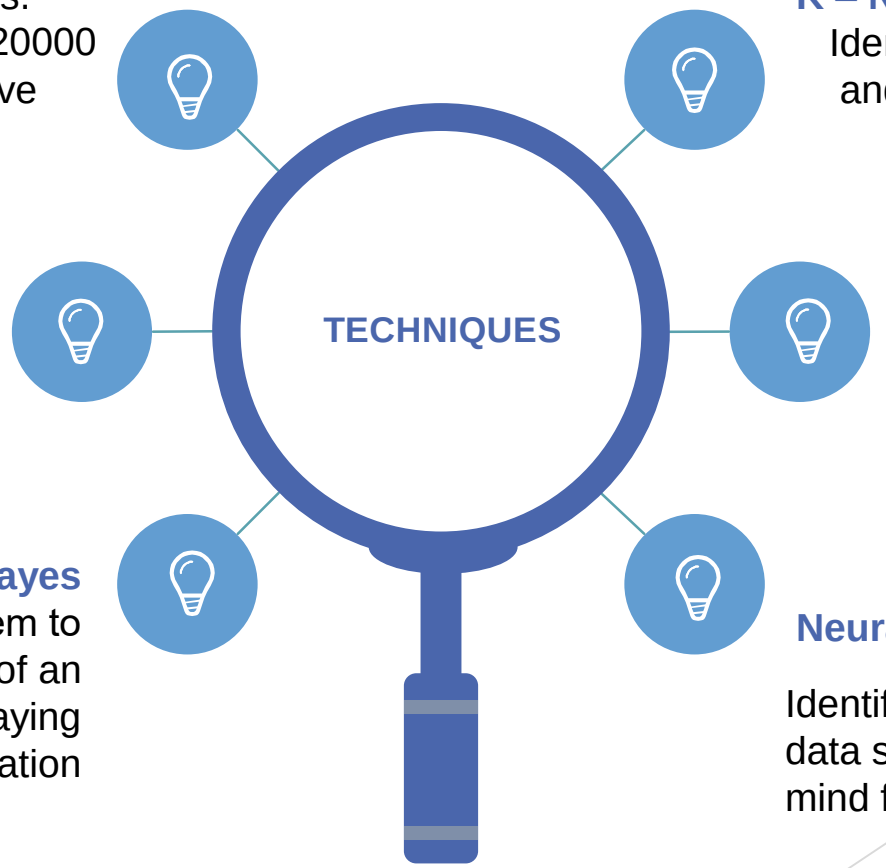
Create set of rules to classify targets.
Ex: People with Monthly Income < 20000 and Experience < 2 are likely to leave

Logistic Regression

Use regression like analysis to predict categories

Naïve Bayes

Use Bayes' theorem to calculate probability of an employee leaving or staying in an organization



K – Nearest Neighbors (KNN)

Identify similar data points and take a polling

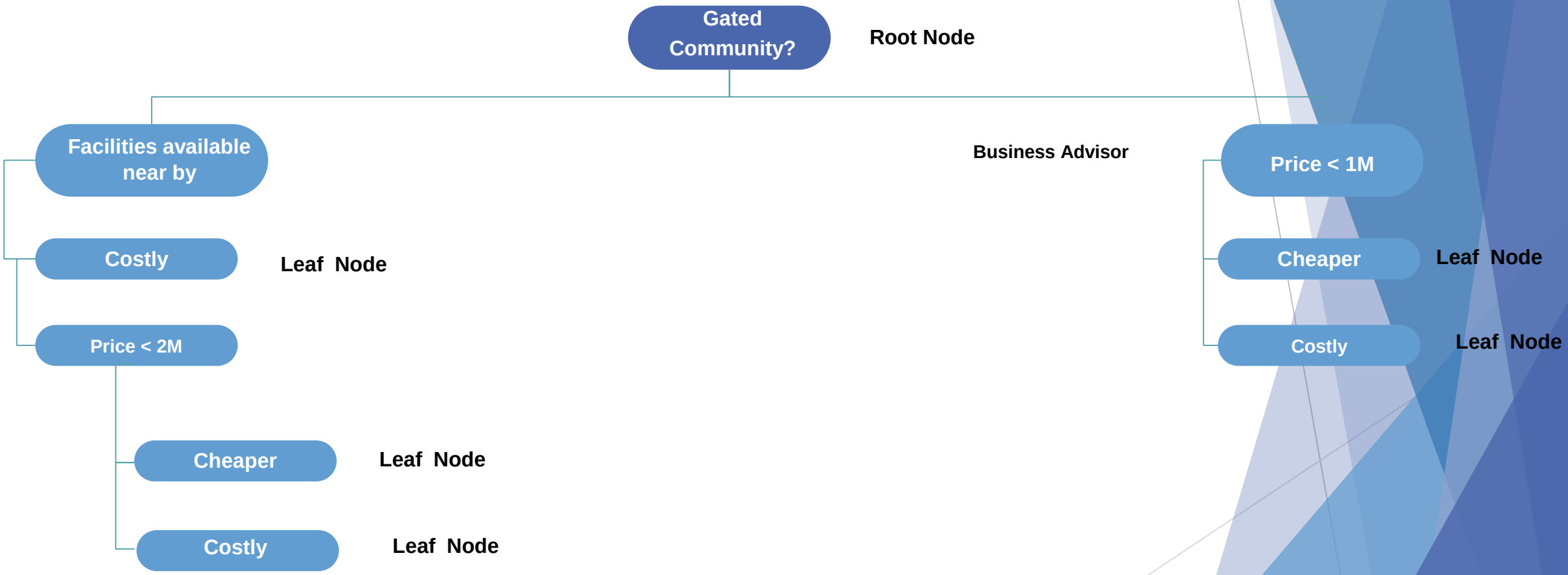
Support Vector Machines

Used for both classification or regression challenges. It is care very good when you have a huge number of features.

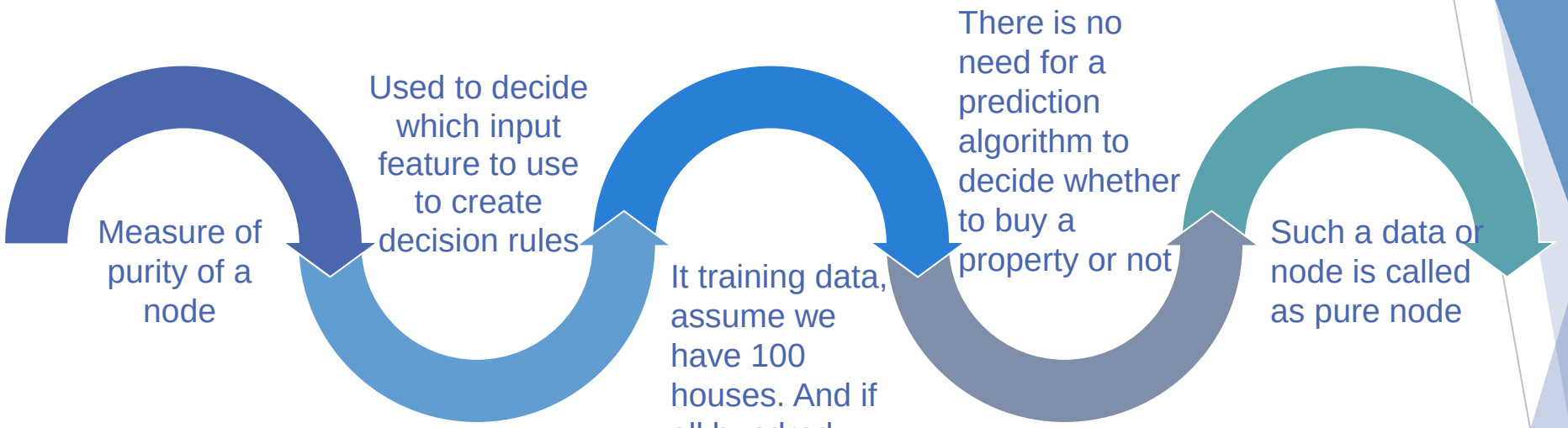
Neural Networks

Identify underlying relationships within a data set by mimicking the way a human mind functions

EXAMPLE OF DECISION TREE



GINI IMPURITY



Gini impurity for a pure node will be zero

$$\textbf{Gini impurity} = 1 - \sum_{i=0}^{K} p(i)^2$$

Where K = No. of classes in target variable

$$GI = 1 - p(\text{costly } homes)2 - p(\text{cheaper } homes)2$$
$$GI = 1 - (0/400)2 - (400/400)2$$
$$GI = 0$$

No. of houses = 400
Costly = 0
Cheaper= 400

GINI IMPURITY

In reality, every node will be impure

No. of homes = 400
Costly homes = 100
Cheaper homes= 300

$$Gini\ impurity = 1 - \sum_{i=0}^{K} p(i)^2$$

Where K = No. of classes in target variable

$$GI = 1 - p(costly\ \ homes)^2 - p(cheaper\ homes)^2$$
$$GI = 1 - (100/400)^2 - (300/400)^2$$
$$\boldsymbol{GI = 0.375}$$

INPUT FEATURES

Use input features to reduce gini impurity.

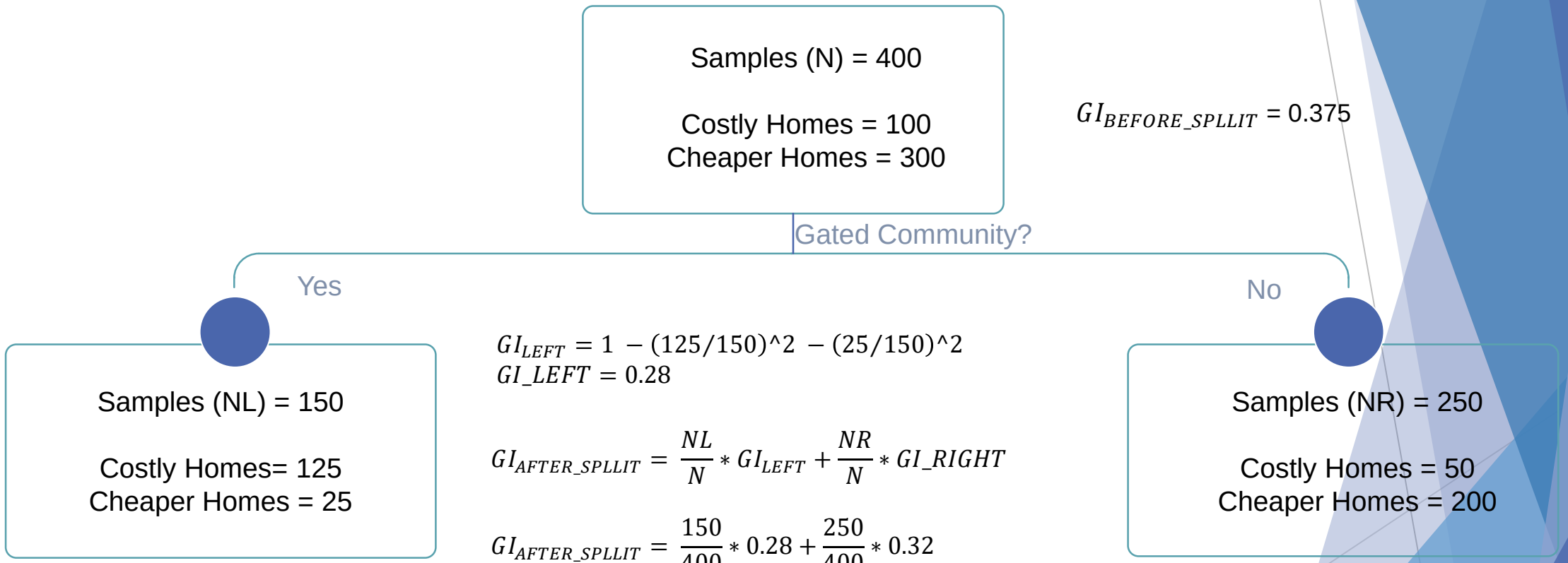
Features like:

- Property type whether it is gated community or not
- Price of the property
- Nearby facilities available

WHICH ONE TO USE ?

The input feature which maximum helps to reduce the **gini impurity** is the one which will be used in root node.

GINI IMPURITY AFTER SPLIT



GI_{BEFORE_SPLLIT} = 0.375

$GI_{LEFT} = 1 - (125/150)^2 - (25/150)^2$

$GI\_LEFT = 0.28$

$$GI_{AFTER\_SPLLIT} = \frac{NL}{N} * GI_{LEFT} + \frac{NR}{N} * GI\_RIGHT$$

$$GI_{AFTER\_SPLLIT} = \frac{150}{400} * 0.28 + \frac{250}{400} * 0.32$$

GI_{AFTER_SPLLIT} = 0.3

INFORMATION GAIN

- Information Gain = Gini Impurity before split – Gini Impurity after split
- Gives a measure of information gained on the target variable, by using input feature
- The input feature which gives maximum information gain will be used in the root node for splitting the data in two parts
- In the following table, the input feature whether the property is gated community or not has the maximum information gain, hence it will be used in root node

Input Feature	Gini Impurity at root node	Gini impurity after split	Information Gain
Gated community or not	0.375	0.3	0.075
Property Price	0.375	0.35	0.025
Facilities available or not	0.375	0.37	0.005

RECURSIVE PARTITIONING

01 The first decision rule is decided based on the input feature which helps us to gain maximum information

02 Many a times, having a single decision rule is not quite helpful

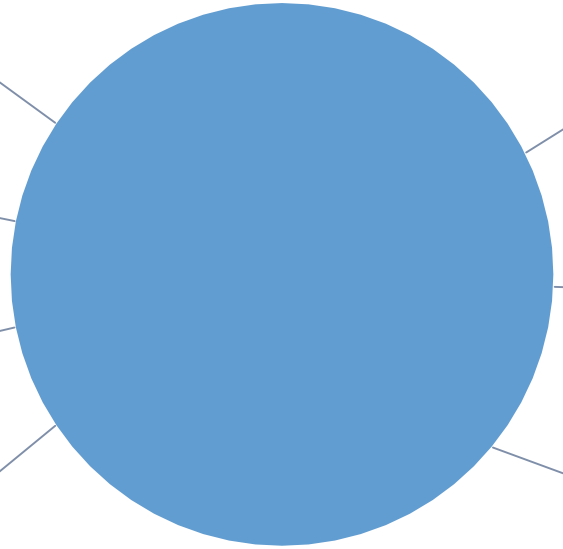
03 We would like to use other input features as well to gain more information

04 Left node can be further partitioned using any other input feature or the same input feature with different cut point

05 To decide which input feature to use, the same process in used to calculate the Gini Impurity before split and after split with new set of observations in the left node

06 This process is repeated for right node as well and for any further splits

07 Hence the name recursive partitioning



PREDICT DEFAULTERS

01 A leading bank contains many customers who use their credit card services

02 Every month, lot of customers fail to pay the credit card bill on time. Such customers are called as defaulters

03 The organization would like to predict the names of those customers who are likely to default next month, so that they can plan their finances accordingly

01 For this exercise, they have collected various customer's past data who paid bill on time as well as those who defaulted

02 They have collected information like age, credit history, balance, etc.

03 The default column is the target column, which we would like to predict

```
In [22]: credit = pd.read_csv('/datasets/credit-default.csv')
         credit.columns

Out[22]: Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
                'amount', 'savings_balance', 'employment_length', 'installment_rate',
                'personal_status', 'other_debtors', 'residence_history', 'property',
                'age', 'installment_plan', 'housing', 'existing_credits', 'default',
                'dependents', 'telephone', 'foreign_worker', 'job'],
               dtype='object')
```

DATA PREPROCESSING

Check for the presence of missing values. In this dataset we do not have missing values.

```
In [24]: pd.isnull(credit).sum().sum()

Out[24]: 0
```

Convert all categorical columns to numerical columns, except for target column

```
In [18]: credit_dummy = pd.get_dummies(credit.drop('default', axis=1))
         credit_dummy.shape

Out[18]: (1000, 61)
```

Ideally we should perform exploratory data analysis to identify the presence of outliers and treat them if necessary

PREPARE TRAIN AND TEST DATA

- Split data to training and testing data
- While splitting the data, if we additionally pass target column as second parameter, we can directly get the input and output features for training and testing
- Out of 1000 observations, 800 are randomly sampled for training and remaining 200 will be considered for testing the model

```
In [31]: from sklearn.model_selection import train_test_split
         train_x, test_x, train_y, test_y = train_test_split(credit_dummy,
                                                             credit['default'],
                                                             test_size=0.2,
                                                             random_state=100)

         print(train_x.shape, train_y.shape)
         print(test_x.shape, test_y.shape)

         (800, 61) (800,)
         (200, 61) (200,)
```

FIT DECISION TREE

- Initiate a DecisionTreeClassifier with max_depth=3
- Max_depth is the depth of the tree. i.e. number of levels to recursively partition the data
- One should use hyper parameter tuning to choose optimal depth

```
In [32]: from sklearn.tree import DecisionTreeClassifier
         model = DecisionTreeClassifier(max_depth=3, random_state = 100)
```

- Pass input features and target variable from training data to fit the model to create decision rules

```
In [33]: model.fit(train_x, train_y)

Out[33]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3
         ,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=1
         00,
                     splitter='best')
```

FUNCTION TO PLOT DECISION TREE

Following is the function to plot a decision tree in jupyter notebook:

```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```

- Install GraphViz software and configure the correct path inside the function

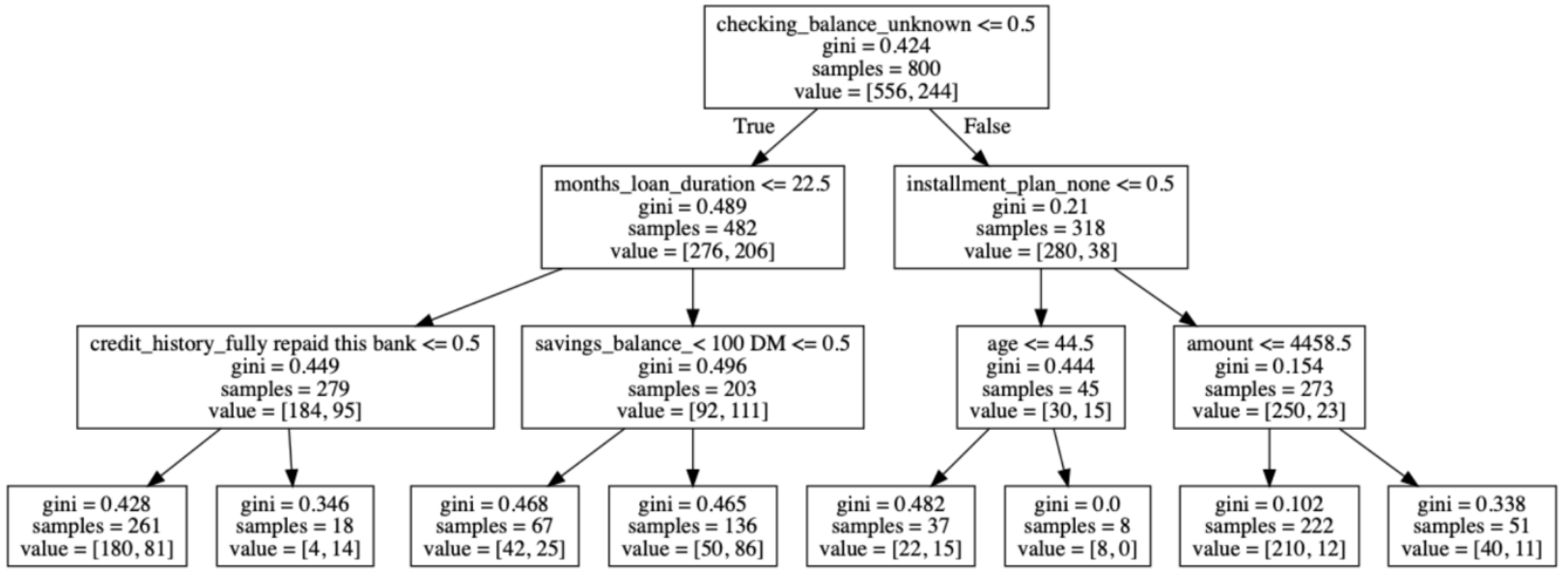
```
In [34]: def draw_tree(model, columns):
             import pydotplus
             from sklearn.externals.six import StringIO
             from IPython.display import Image
             import os
             from sklearn import tree

             graphviz_path = 'C:\Program Files (x86)\Graphviz2.38/bin/'
             os.environ["PATH"] += os.pathsep + graphviz_path

             dot_data = StringIO()
             tree.export_graphviz(model,
                                  out_file=dot_data,
                                  feature_names=columns)
             graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
             return Image(graph.create_png())
```

PLOT DECISION TREE

- Pass the decision tree model and the input features names to draw_tree function to visualize the decision rules identified by the model
- Since max_depth=3, we have three levels of decision rules

```
In [36]: draw_tree(model, train_x.columns)
```

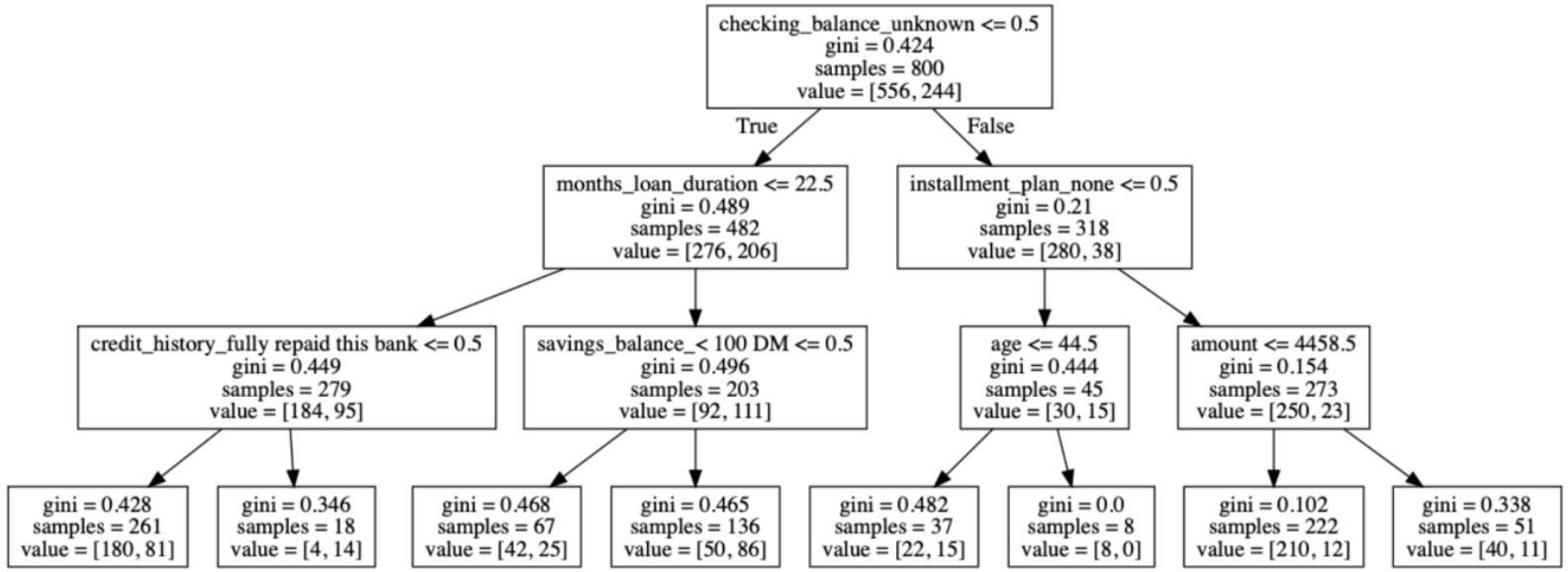
Out[36]:



PLOT DECISION TREE

```
In [36]: draw_tree(model, train_x.columns)
```

Out[36]:

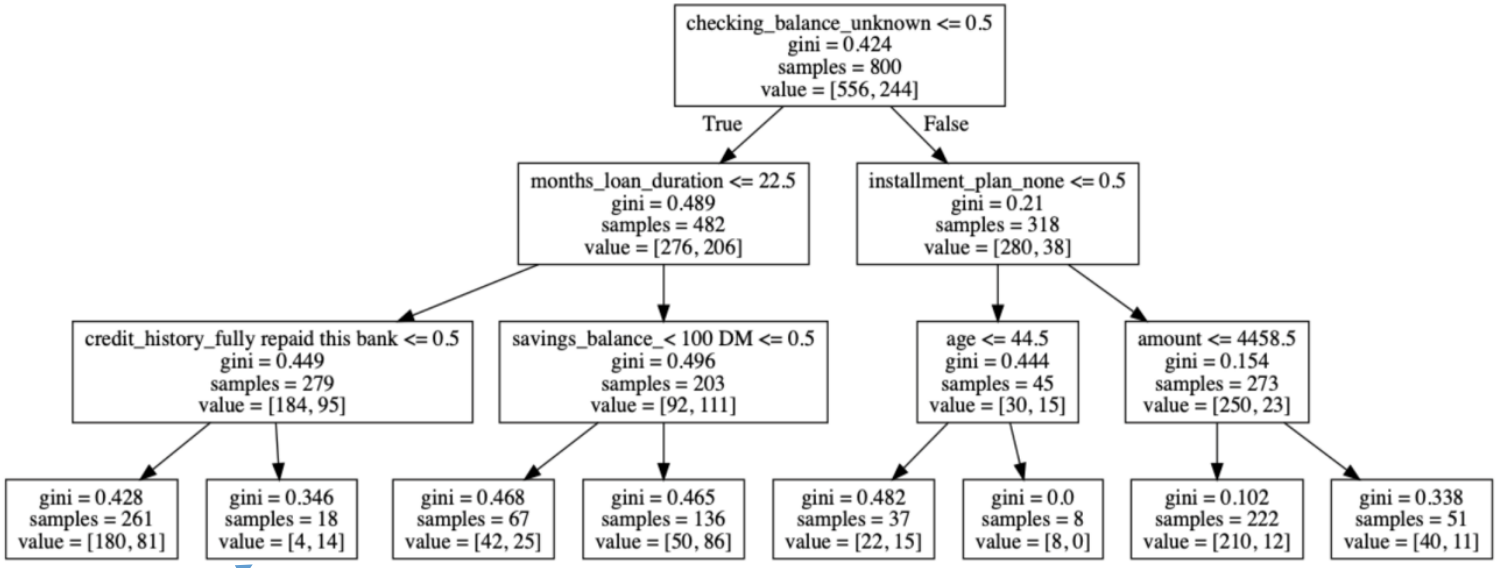


Out of 261 customers who fall under this node, 180 have paid the credit card bill on time, 81 customers have defaulted. In test data, whoever falls under this node, have 69% probability (180/261) of paying the bill on time. Hence for those customers model prediction will be “non-defaulters” i.e. Class 1 in our example.

PLOT DECISION TREE

```
In [36]: draw_tree(model, train_x.columns)
```

Out[36]:



Out of 18 customers who fall under this node, 4 have paid the credit card bill on time, 14 customers have defaulted. In test data, whoever falls under this node, have only 28% probability (4/14) of paying the bill on time. Hence for those customers model prediction will be “defaulters” i.e. Class 2 in our example.

PREDICT PROBABILITIES

- Pass the input features from test data to **model.predict_proba()** function to predict the probability values for defaulting and not defaulting for each customer in the test data
- The result can be converted to a data frame format for ease of operations

```
In [50]: pred_probs = model.predict_proba(test_x)
         pred_probs = pd.DataFrame(pred_probs, columns=['Class 1', 'Class 2'])
         pred_probs.head(2)
```

Out[50]:

	Class 1	Class 2
0	0.945946	0.054054
1	0.689655	0.310345

- The probability of the first customer in the test data to default is very less (5%). Hence prediction will be that the customer will not default
- The probability of the second customer to default is 31%. But probability of non defaulting is 69%. Hence again the prediction will be that the customer will not default

PREDICT PROBABILITIES

- For the following customers, we can observe that the probability of default is very high (63%)
- Hence for these two customers, we will predict that they will default the credit card payment

In [53]:
```
pred_probs = model.predict_proba(test_x)
pred_probs = pd.DataFrame(pred_probs, columns=['Class 1', 'Class 2'])
pred_probs.iloc[[16, 18]]
```

Out[53]:

	Class 1	Class 2
16	0.367647	0.632353
18	0.367647	0.632353

PREDICT CLASSES

- We can directly predict the target classes using **model.predict()** function, instead of probability values
- Out of first twenty results, we can see that model has predicted 3 people as defaulters

```
In [42]: test_predict = model.predict(test_x)
         test_predict[:20]

Out[42]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1])
```

- Out of 200 test samples, model has identified that 33 customers will default
- Based on their credit card usage, the company can upfront plan the financials accordingly

```
In [44]: pd.Series(test_predict).value_counts()

Out[44]: 1    167
         2     33
         dtype: int64
```

COMPARE ACTUAL VS. PREDICTIONS

Combine the actual and predicted values in a data frame to compare the results

```
In [55]: pd.DataFrame({'actual': test_y, 'predicted': test_predict})
```

Out[55]:

	actual	predicted	
249	2	1	→ Wrong prediction. Customer has actually defaulted.
353	2	1	
537	1	1	→ Correct prediction
424	2	1	
330	1	2	→ Wrong prediction. Customer hasn't defaulted
923	1	1	
315	2	2	→ Correct prediction. Model identified defaulter correctly

CALCULATE ACCURACY

Count all correct predictions. Out of 200 predictions, 147 predictions match the actuals

```
In [58]: df_comp = pd.DataFrame({'actual': test_y, 'predicted': test_predict})
         (df_comp['actual'] == df_comp['predicted']).sum()

Out[58]: 147
```

Accuracy of the model is percentage of correct predictions. Model has 73.5% accuracy

```
In [59]: (df_comp['actual'] == df_comp['predicted']).sum() / df_comp.shape[0]

Out[59]: 0.735
```

We can also use accuracy score from sklearn.metrics to calculate accuracy. Always pass actual values as first parameter

```
In [61]: from sklearn.metrics import accuracy_score
         accuracy_score(df_comp['actual'], df_comp['predicted'])

Out[61]: 0.735
```

TYPE OF PREDICTIONS

- There are four type of predictions, when compared with the actual values
- In target variable, consider the class which we are interested in predicting correctly. In our example, we would like to identify the defaulters accurately. Hence defaulters (Class 2) as positive class and non-defaulters (Class 1) as negative class

Actual	Predicted	Type of prediction	Description
1 (non-defaulter)	1 (non-defaulter)	True Negative	Predicted as negative class and the prediction is true
1 (non-defaulter)	2(defaulter)	False Positive	Predicted as positive class, but the prediction is false
2 (defaulter)	2(defaulter)	True Positive	Predicted as positive class and the prediction is true
2 (defaulter)	1 (non-defaulter)	False Negative	Predicted as negative class, but the prediction is false

IDENTIFY THE TYPE OF PREDICTIONS

```
In [55]: pd.DataFrame({'actual': test_y, 'predicted': test_predict})
```

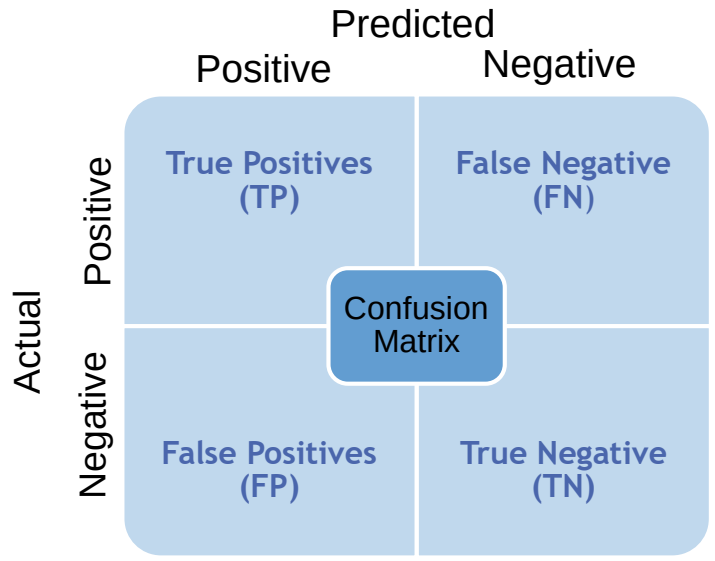
Out[55]:

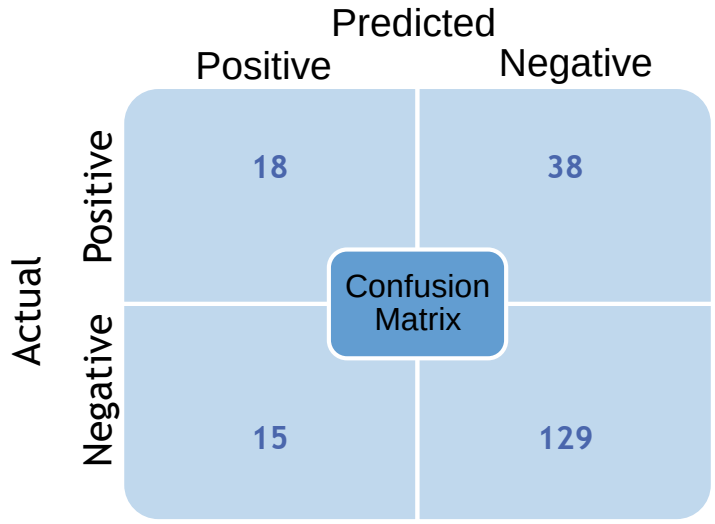
	actual	predicted	
249	2	1	→ False Negative
353	2	1	
537	1	1	→ True Negative
424	2	1	
330	1	2	→ False Positive
923	1	1	
315	2	2	→ True Positive

CONFUSION MATRIX

- Usually the types of predictions are visualized using a matrix
- Rows are for actual classes. Column represents for predicted classes

```
In [71]: tn, fp, fn, tp = confusion_matrix(df_comp['actual'],
                                           df_comp['predicted'],
                                           labels=[1,2]).ravel()

         tn, fp, fn, tp

Out[71]: (129, 15, 38, 18)
```





SENSITIVITY & SPECIFICITY

ACCURACY

Accuracy indicates the overall performance of the model

Accuracy = (TP+TN) / (TP+FN+FP+TN)

= (18+129) / (18+38+15+129)

= 0.735

SENSITIVITY

It is a measure of how accurately model is identifying the positive class i.e. defaulters. In our model, out of 56 defaulters in test data, we were able to identify only 18 correctly, which is just 32%

Sensitivity = TP / (TP + FN) = 18 / (18 + 38) = 0.32

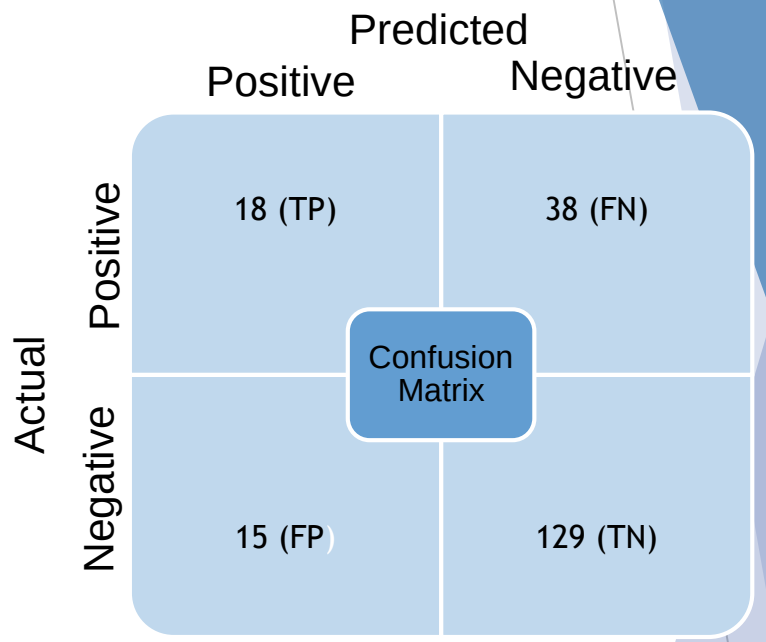
SPECIFICITY

It is a measure of how accurately model is identifying the negative class i.e. non-defaulters. In our model, out of 144 non-defaulters in test data, we were able to identify 129 correctly. Which is 90%

Specificity = TN / (TN + FP)

= 129 / (129 + 15)

= 0.90



Note: Our model is able to identify **non-defaulters correctly than identifying defaulters.** Because, in our data, percentage of non-defaulters is very high (72%) compared to defaulters (28%). Hence accuracy alone is not a good measure to evaluate the model for imbalanced target labels

HANDS-ON PRACTICE

Lets Try

- Decision trees
- Decision Trees Examples

INTRODUCTION TO LOGISTIC REGRESSION

The **Regression problems** have continuous and usually unbounded outputs whereas, **Classification problems** have discrete and finite outputs called classes or categories.

There are two main types of classification problems:

- **Binary** or **binomial classification:** for the binary-classification task . Exactly two classes of the output variables to choose between. Usually 0 and 1, true and false, or positive and negative.
- **Multiclass** or **multinomial classification:** for the multi-classification task . Tthree or more classes of the outputs to choose from

Independent variables (Inputs):
They are also called input variables or predictors and they don't depend on other features of interest (or assume so for the purpose of the analysis)

Dependent variables (Outputs):
They are also called output variables or responses and they depend on the independent variables

LOGISTIC REGRESSION

It is a supervised classification model to estimate discrete values - dependent variable - (1/0, yes/no, true/false) to define a boundary based on given set of independent variable(s)

Like all regression analyses, the logistic regression is a predictive analysis

Logistic Regression Equation:

$$y = e^{(b0 + b1*x)} / (1 + e^{(b0 + b1*x)})$$

Logistic regression algorithm can also be used to solve the multi-classification problems

LOGIT FUNCTION

Logit function is used to fit the data of an event, so it is also called as logit regression

Maximum likelyhood is used as an estimate for training a logistic regression

This estimate is used to minimize the empirical loss

It predicts the probability ‘p’ of occurrence, who's values lies between 0 and 1

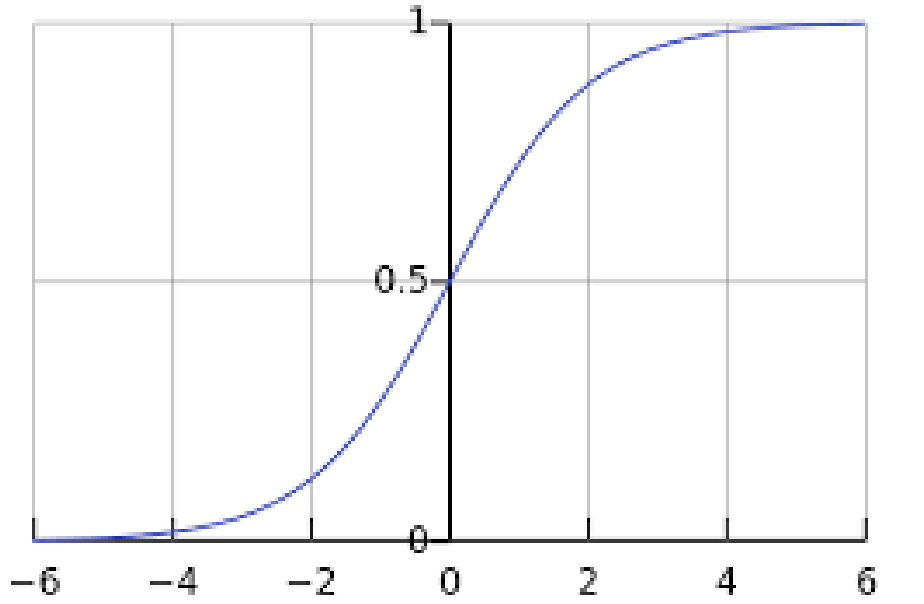
A logistic function is an S-shaped curve with equation :

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

x_0 = x value of midpoint
L = maximum value
k = growth rate

OR 1 / (1 + e^-value)

e = the natural logarithm base (Euler's number)



Standard logistic sigmoid function
$L = 1, k = 1, x_0 = 0$

- p is the probability of success
- q is the probability of failure, q=1- p
- odds = p/q
- ln(odd) = ln(p/q)
- logit(p) = ln(p/q) = $\theta_0+\theta_1X_1+\theta_2X_2+...+\theta_nX_n$
- Log is used for simplicity to repeat the step function

LOGISTIC REGRESSION PROS AND CONS

Advantages:

- Makes no assumptions about distributions of classes in feature space
- Easily extended to multiple classes (multinomial regression)
- Natural probabilistic view of class predictions
- Quick to train
- Very fast at classifying unknown records
- Good accuracy for many simple data sets
- Can interpret model coefficients as indicators of feature

Disadvantage:

- Linear decision boundary

LOGISTIC REGRESSION WITH SCIKIT-LEARN

General steps for classification:

1. Import necessary packages, functions and classes

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import matplotlib.pyplot as plt
import pandas as pd
```

2. Get data to work with

```
data = pd.read_csv('/Dataset/bank-full.csv', sep=';')
```

3. Do necessary data cleaning process

4. Perform exploratory data analysis

LOGISTIC REGRESSION WITH SCIKIT-LEARN

5. Create dummies if there is categorical columns and drop the target variable

```
data_dummy = pd.get_dummies(data.drop('y',axis=1))
```

```
#Remove unknown columns if necessary
data_dummy.drop(data_dummy[['job_unknown', 'education_unknown', 'poutcome_unknown']],
                axis=1, inplace=True)
```

6. Check for independence between features. If any feature is not independent of each other, drop those columns else go to the next step

```
sns.heatmap(data_dummy.corr())
plt.show()
```

7. Split the data into train data and test data

```
#Splitting data to train and test data
x_train,x_test,y_train,y_test=train_test_split(data_dummy,data['y'],test_size=0.3,random_state=0)
```

LOGISTIC REGRESSION WITH SCIKIT-LEARN

8. Create the classification model and train (or fit) it with existing data

```
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

9. Evaluate the model

```
y_predict = model.predict(x_test)
train_score=model.score(x_train,y_train)
#Accuracy of the test model
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(test_score))
```

```
Accuracy of logistic regression classifier on test set: 0.90
```

```
model.intercept_
model.coef_
```

LOGISTIC REGRESSION – CONFUSION MATRIX

```
#Check the model performance with .predict_proba().
#This returns the matrix of probabilities that the predicted output is equal to zero or one:
model.predict_proba(x_test)
```

```
array([[0.99347096, 0.00652904],
       [0.20415306, 0.79584694],
       [0.98054658, 0.01945342],
       ...,
       [0.99024004, 0.00975996],
       [0.95049157, 0.04950843],
       [0.94023742, 0.05976258]])
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,y_predict)
confusion_matrix
```

```
array([[11638,   331],
       [ 1038,   557]], dtype=int64)
```

Assumption: The result is telling us that we have 11638+557= 12195 correct predictions and 1038+331= 1369 incorrect predictions.

True positives in the upper-left position
False negatives in the lower-left position
False positives in the upper-right position
True negatives in the lower-right position

CLASSIFICATION REPORT

Logistic Regression Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.91      0.99      0.95     10981
          1       0.70      0.18      0.29      1376

avg / total       0.88      0.90      0.87     12357
```

Interpretation:

Of the entire test set, 88% of the promoted term deposit were the term deposit that the customers liked. Of the entire test set, 90% of the customer's preferred term deposit were promoted.

HANDS-ON PRACTICE

Lets Try

- Logistic and SVM