

# Table of Content

## **1. Machine Learning (Supervised Algorithms)**

1.1 What is Machine Learning?	4
1.2 What is Data Preprocessing?	4
• Steps in Data Preprocessing	5
1. Import Libraries	5
2. Read Data	6
3. Checking for missing values	7
Let's look at the data	7
4. Checking for categorical data	8
Let's plot the distribution of features	9
5. Standardize the data	13
6. PCA transformation	14
7. Data splitting	18

## **2. Regression**

2.1 Linear Regression	21
2.2 Multiple Linear Regression	22
2.3 Logistic Regression	24
2.4 Hypothesis Testing	26

## **3. Decision Trees**

3.1 Introduction to Decision Tree	31
Decision Tree Visual Example	32
Assumptions we make while using Decision tree	32
Decision Tree Algorithm Pseudocode	33
Sklearn Library Installation	33
Balance Scale Data Set Description	33
Balance Scale Problem Statement	34
Decision Tree classifier implementation in Python with sklearn Library	34
Python packages used	34
Importing Python Machine Learning Libraries	35
Data Import	36
Data Slicing	38
Decision Tree Training	38
Decision Tree Classifier with criterion gini index	40
Decision Tree Classifier with criterion information gain	42
Prediction	42
Prediction for Decision Tree classifier with criterion as gini index	42
Prediction for Decision Tree classifier with criterion as information gain	42
Calculating Accuracy Score	43
Accuracy for Decision Tree classifier with criterion as gini index	44

## Module 2: Supervised Machine Learning Techniques

Accuracy for Decision Tree classifier with criterion as information gain	44
Conclusion	44
<b>4. Naïve Bayes</b>	465
4.1 Introduction to Naïve Bayes	46
4.2 What is the Bayes' Theorem	46
<b>5. Naïve Bayes</b>	46
5.1 Support Vector Machine	49
<b>6. K-Nearest Neighbour (KNN)</b>	51
6.1 K-Nearest neighbour Classification	52
K-nearest neighbor classifier implementation with scikit-learn	53
Libraries:	53
Data Import:	54
Output:	56
Data Imputation:	57
Train, Test data split:	58
KNN Implementation:	59
Accuracy Score:	61
Output:	62
K value Vs Accuracy Change Graph	63
<b>7. Random Forest Classification</b>	64
How it works:	65
Real Life Analogy:	67
Feature Importance:	68
Difference between Decision Trees and Random Forests:	69
Important Hyperparameters:	70
Advantages and Disadvantages:	71
Example¶	72
FAQs	75

# **MACHINE LEARNING (SUPERVISED LEARNING)**

## 1.1 What Is Machine Learning?

---

Machine Learning is an idea where the computers or system learns from examples and experience, without being explicitly programmed. Instead of writing code, you feed data to the generic algorithm, and it builds logic based on the data given.

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn.

### Examples of Machine Learning

There are many examples of machine learning. Here are a few examples of classification problems where the goal is to categorize objects into a fixed set of categories.

1. **Face detection:** Identify faces in images (or indicate if a face is present).
2. **Email filtering:** Classify emails into spam and not-spam.
3. **Medical diagnosis:** Diagnose a patient as a sufferer or non-sufferer of some disease.
4. **Weather prediction:** Predict, for instance, whether or not it will rain tomorrow.

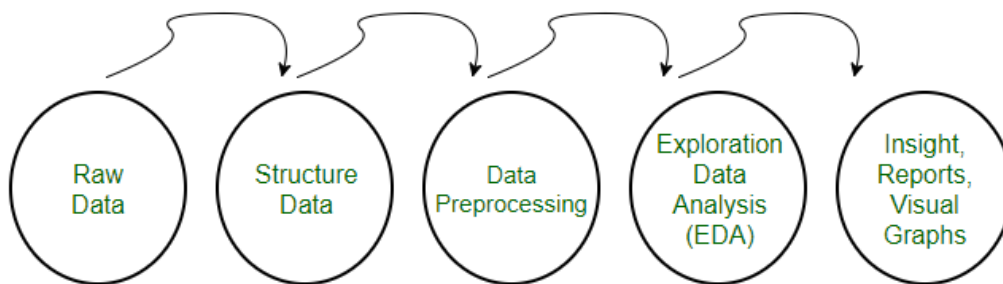
## 1.2 What is Data Preprocessing?

---

It is a data mining technique that transforms raw data into an understandable format. Raw data (real world data) is always incomplete and that data cannot be sent through a model. That would cause certain errors. That is why we need to preprocess data before sending through a model.

### Steps in Data Preprocessing

1. Import libraries
2. Read data
3. Checking for missing values
4. Checking for categorical data
5. Standardize the data
6. PCA transformation
7. Data splitting



**FIGURE 1.1:**

#### 1. Import Libraries

As main libraries, I am using Pandas, Numpy and time;

**Pandas:** Use for data manipulation and data analysis.

**Numpy:** A fundamental package for scientific computing with Python.

As for the visualization I am using **Matplotlib** and Seaborn.

For the data preprocessing techniques and algorithms, I used **Scikit-learn** libraries.

```
# importing libraries

import pandas as pd

import numpy as np

import time
# visual libraries
from matplotlib import pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
plt.style.use('ggplot')
# sklearn libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.metrics import
confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef, classification_report, roc_curve
from sklearn.externals import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

## 2. Read Data

You can find more details on dataset here:

<https://www.kaggle.com/mlg-ulb/creditcardfraud>

```
# Read the data in the CSV file using pandas
df = pd.read_csv('../input/creditcard.csv')
df.head()# gives first 5 rows of dataset as output
```

## Module 2: Supervised Machine Learning Techniques

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20

**TABLE 1.1: Dataset**

```
df.shape
# gives no of datapoints and no of features present
in dataset as output
> (284807, 31)
```

### 3. Checking for missing values

```
df.isnull().any().sum()
# gives the number of missing data values in the whole dataset
>0
```

Since there are no missing values found in the dataset, I didn't use any missing values handling techniques.

**Let's look at the data**

So the dataset is labeled as 0s and 1s.

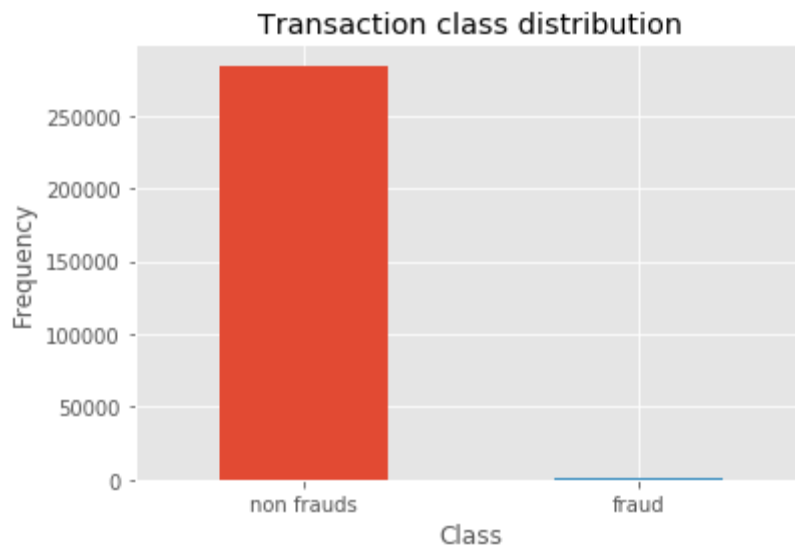
- 0 = non fraud
- 1 = fraud

```
All = df.shape[0]
fraud = df[df['Class'] == 1]
nonFraud = df[df['Class'] == 0]
x = len(fraud)/All
y = len(nonFraud)/All
print('frauds :',x*100,'%')
```

```
print('non frauds :',y*100,'%')
```

```
frauds : 0.1727485630620034 %  
non frauds : 99.82725143693798 %
```

```
# Let's plot the Transaction class against the Frequency  
labels = ['non frauds','fraud']  
classes = pd.value_counts(df['Class'], sort = True)  
classes.plot(kind = 'bar', rot=0)  
plt.title("Transaction class distribution")  
plt.xticks(range(2), labels)  
plt.xlabel("Class")  
plt.ylabel("Frequency")
```



**FIGURE 1.2: Class vs Frequency**

#### 4. Checking for categorical data

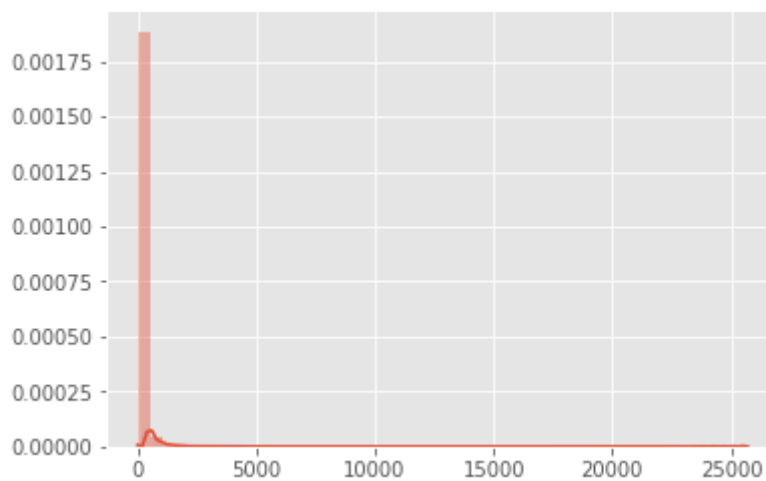
The only categorical variable we have in this data set is the target variable. Other features are already in numerical format, so no need of converting to categorical data.



### Let's plot the distribution of features

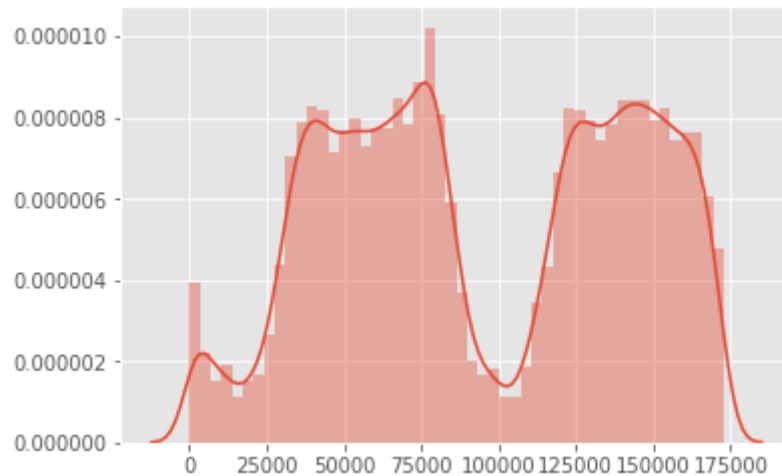
I used seaborn distplot() to visualize the distribution of features in the dataset. We have 30 features (describe where figure of 30 have come?) and target variable in the dataset.

```
# distribution of Amount  
amount = [df['Amount'].values]  
sns.distplot(amount)
```



**FIGURE 1.3: Distribution of Amount**

```
# distribution of Time  
time = df['Time'].values  
sns.distplot(time)
```

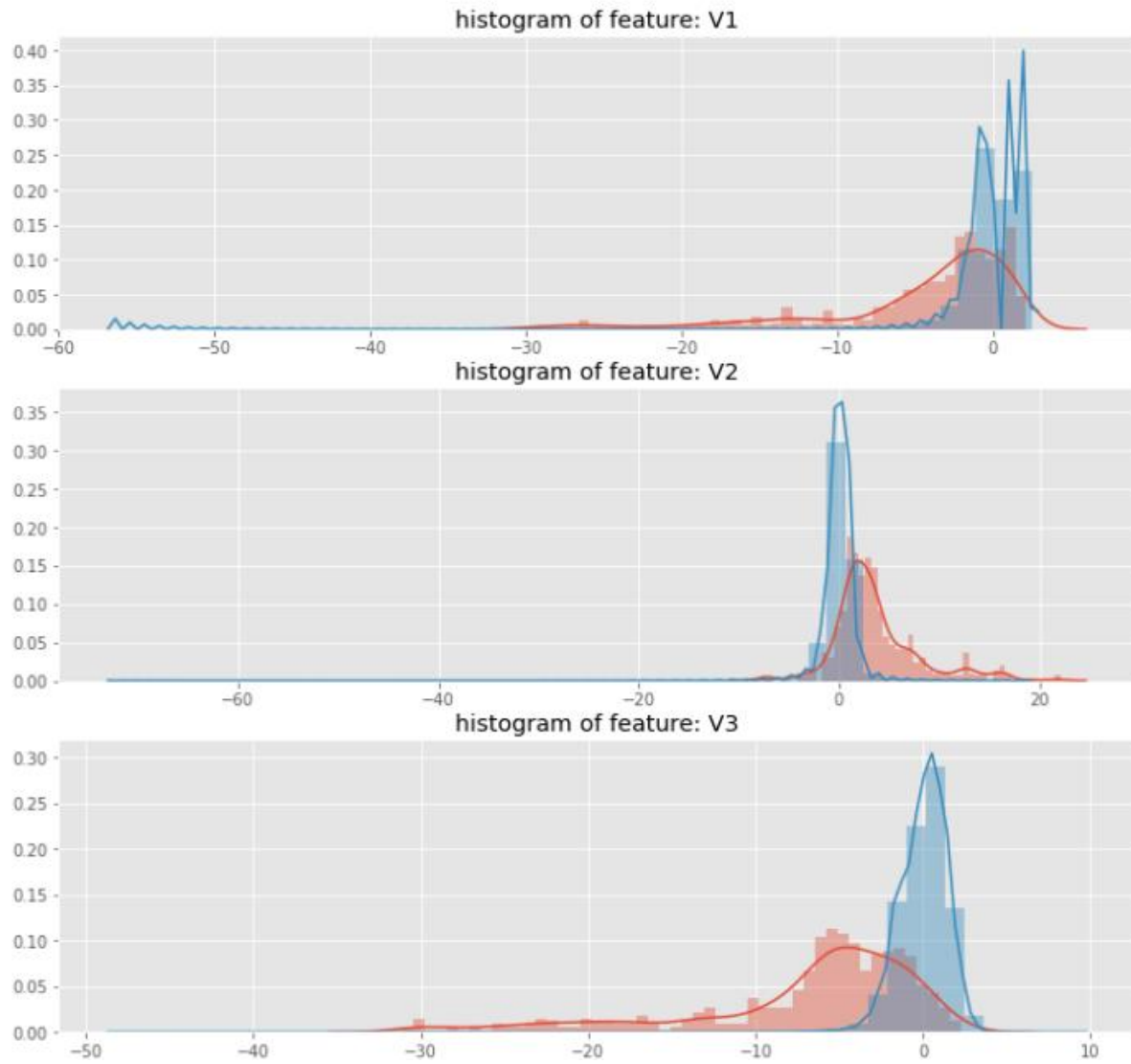


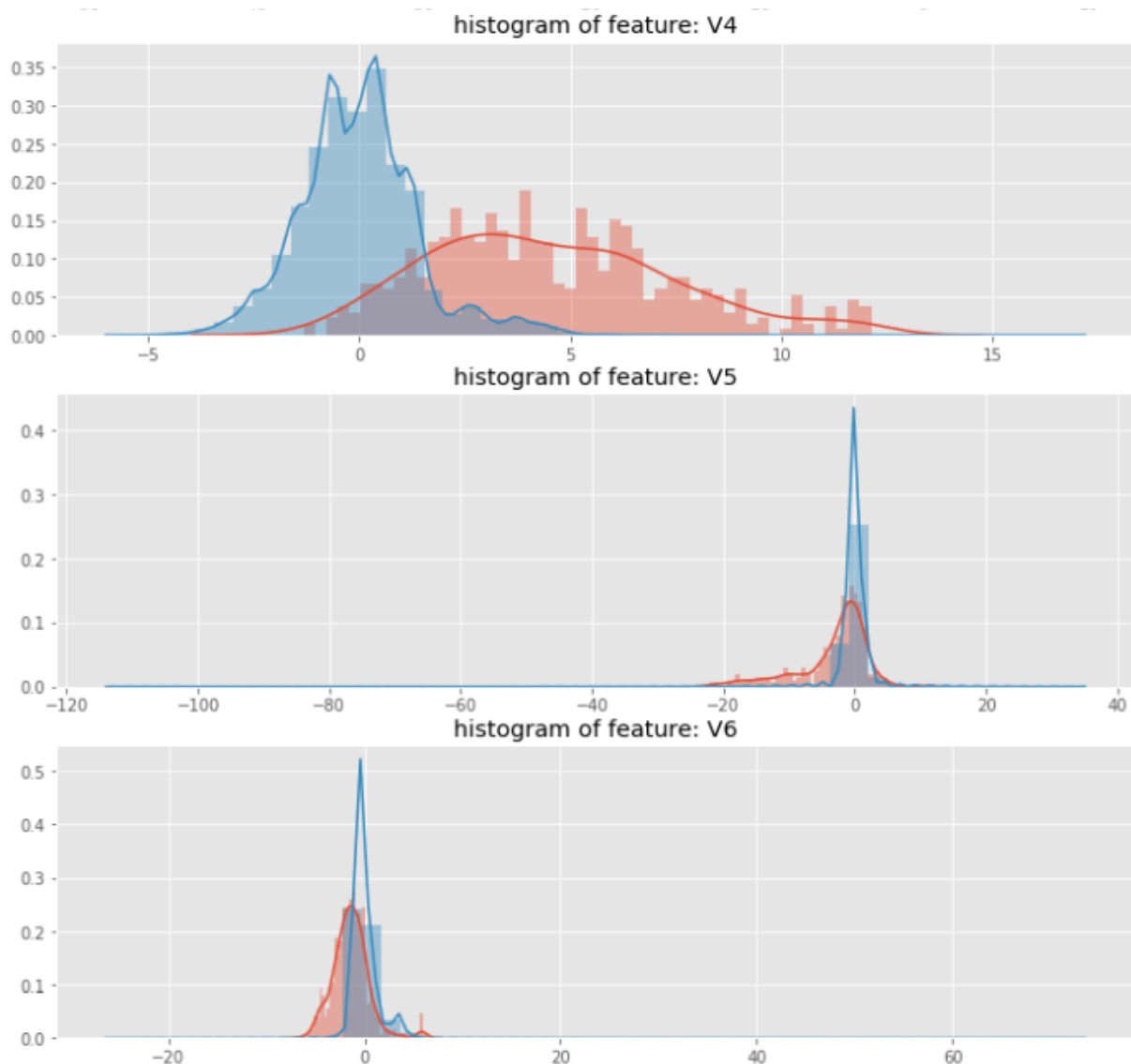
**FIGURE 1.4:** Distribution of Time (what is y and x axis?)

```
# distribution of anomalous features
anomalous_features = df.iloc[:,1:29].columns
#keep showing new data structure content using .head()

plt.figure(figsize=(12,28*4))
gs = gridspec.GridSpec(28, 1)
for i, cn in enumerate(df[anomalous_features]):
    ax = plt.subplot(gs[i])
    sns.distplot(df[cn][df.Class == 1], bins=50)
    sns.distplot(df[cn][df.Class == 0], bins=50)
    ax.set_xlabel('')
    ax.set_title('histogram of feature: ' + str(cn))
plt.show()
```

## Module 2: Supervised Machine Learning Techniques





Similarly, these histograms will go for all the 28 features (i.e, V28)

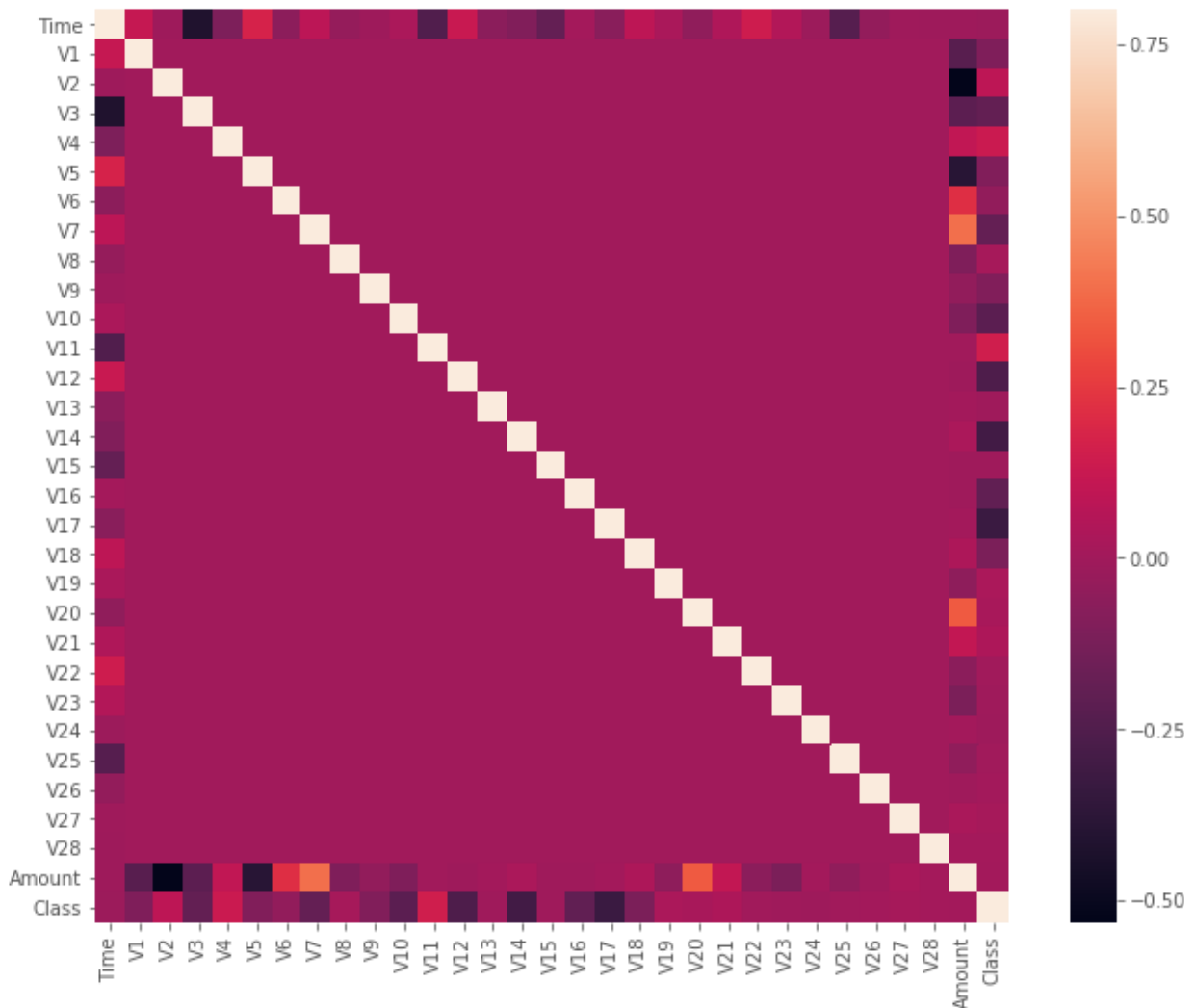
### FIGURE 1.5: Distribution of anomalous features

In this analysis I will not be dropping any features looking at the distribution of features, because I am still in the learning process of working with data preprocessing in numerous ways. So I would like to experiment step by step on data.

Instead all the features will be transformed to scaled variables.

```
# heatmap of correlation of features
correlation_matrix = df.corr()
```

```
fig = plt.figure(figsize=(12,9))
sns.heatmap(correlation_matrix, vmax=0.8, square = True)
plt.show()
```



**FIGURE 1.6:** Heatmap of features

High negative relation is dark colored and high positive relation is light colored

## 5. Standardize the data

The dataset contains only numeric input variables which is the result of a PCA transformation (Explained further below). Features V1, V2,

... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. So PCA is affected by scale so we need to scale the features in the data before applying PCA. For the scaling I am using Scikit-learn's StandardScaler(). In order to fit to the scaler the data should be reshaped within -1 and 1.

```
# Standardizing the features
df['Vamount'] =
StandardScaler().fit_transform(df['Amount'].values.reshape(-
1,1))
df['Vtime'] =
StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))

df = df.drop(['Time','Amount'], axis = 1)
df.head()
```

V22	V23	V24	V25	V26	V27	V28	Class	Vamount	Vtime
0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0	0.244964	-1.996583
-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0	-0.342475	-1.996583
0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0	1.160686	-1.996562
0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0	0.140534	-1.996562
0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0	-0.073403	-1.996541

**TABLE 1.2: Standardized dataset**

Now all the features are standardized into unit scale (mean = 0 and variance = 1).

## 6. PCA transformation

PCA (Principal Component Analysis) mainly using to reduce the size of the feature space while retaining as much of the information as possible. In here, all the features transformed into two features using PCA.

## Module 2: Supervised Machine Learning Techniques

```
X = df.drop(['Class'], axis = 1)
y = df['Class']

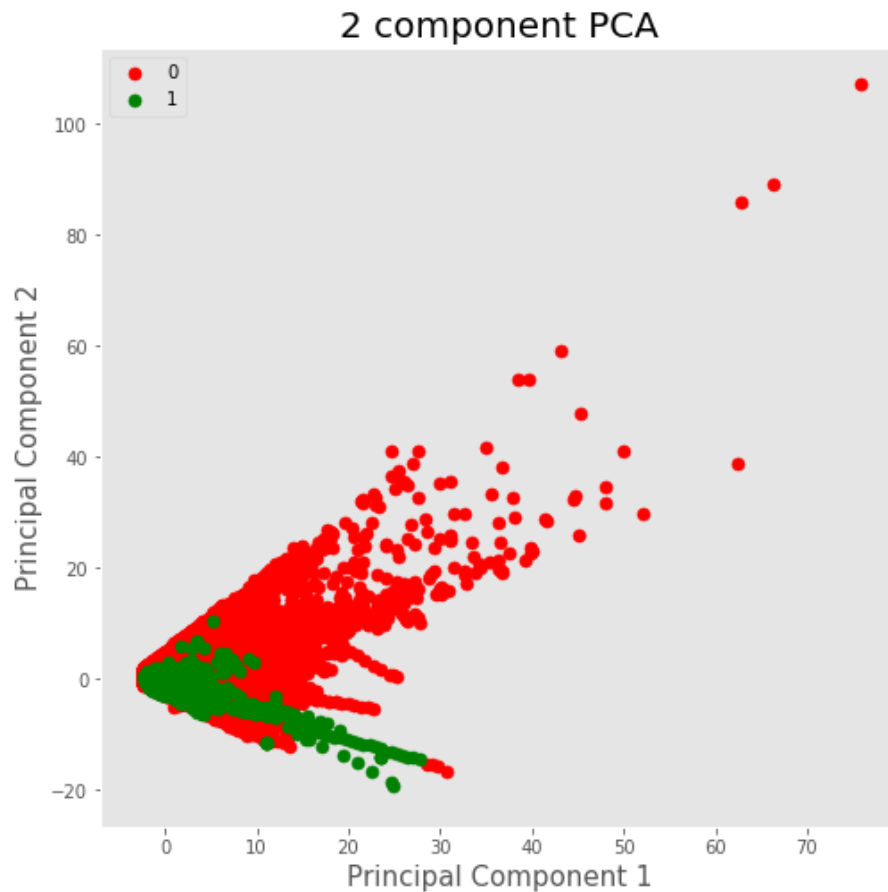
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X.values)
principalDf = pd.DataFrame(data = principalComponents
                           , columns = ['principal component 1', 'principal
component 2'])
finalDf = pd.concat([principalDf, y], axis = 1)
finalDf.head()
```

	principal component 1	principal component 2	Class
0	1.571633	-0.675537	0
1	-1.086136	-0.282819	0
2	2.053450	1.077546	0
3	1.150128	-0.427471	0
4	1.143864	-1.342195	0

**TABLE 1.3: Dimensional reduction**

```
# 2D visualization
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = [0, 1]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Class'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component
1'],
               , finalDf.loc[indicesToKeep, 'principal component
2'],
               , c = color
               , s = 50)
```

```
ax.legend(targets)  
ax.grid()
```



**FIGURE 1.7: Scatter plot of PCA transformation**

Since the data is highly imbalanced, I am only taking 492 (tell why) rows from the non\_fraud transactions.

(As having more amount of such imbalanced data would decrease the performance)

```
# Lets shuffle the data before creating the subsamples  
df = df.sample(frac=1)
```

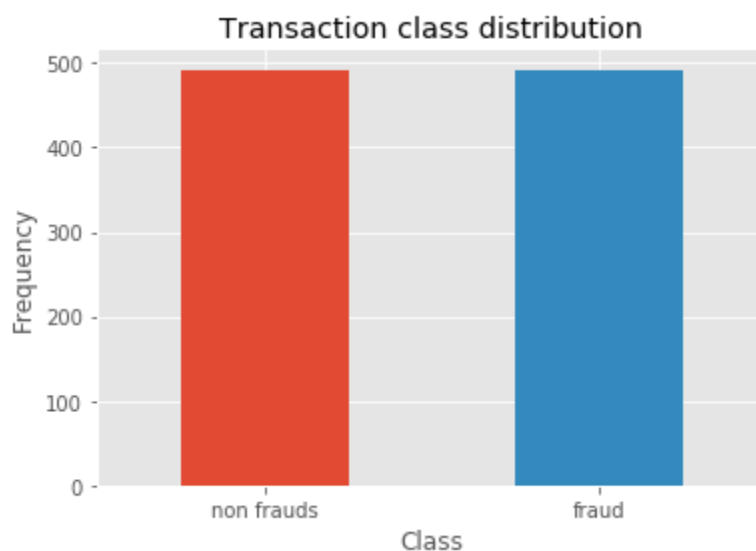


## Module 2: Supervised Machine Learning Techniques

```
frauds = df[df['Class'] == 1]
non_frauds = df[df['Class'] == 0][:492]

new_df = pd.concat([non_frauds, frauds])
# Shuffle dataframe rows
new_df = new_df.sample(frac=1, random_state=42)
```

```
# Let's plot the Transaction class against the Frequency
labels = ['non frauds', 'fraud']
classes = pd.value_counts(new_df['Class'], sort = True)
classes.plot(kind = 'bar', rot=0)
plt.title("Transaction class distribution")
plt.xticks(range(2), labels)
plt.xlabel("Class")
plt.ylabel("Frequency")
```



**FIGURE 1.8:** Distribution of classes

```
# prepare the data
features = new_df.drop(['Class'], axis = 1)
```

```
labels = pd.DataFrame(new_df['Class'])

feature_array = features.values
label_array = labels.values
```

### 7. Data splitting

```
# splitting the feature array and label array keeping 80% for
the training sets
X_train,X_test,y_train,y_test =
train_test_split(feature_array,label_array,test_size=0.20)

# normalize: Scale input vectors individually to unit norm
(vector length).
X_train = normalize(X_train)
X_test=normalize(X_test)
```

## 1.3

## Supervised Machine Learning Problems and Solutions

The most straightforward tasks fall under the umbrella of supervised learning. In supervised learning, we have access to examples of correct input-output pairs that we can show to the machine during the training phase. The common example of handwriting recognition is typically approached as a supervised learning task. We show the computer a number of images of handwritten digits along with the correct labels for those digits, and the computer learns the patterns that relate images to their labels.

Learning how to perform tasks in this way, by explicit example, is relatively easy to understand and straightforward to implement, but there is a crucial task: we can only do it if we have access to a dataset of correct input-output pairs. In the handwriting example, this means that at some point we need to send a human in to classify the images in the training set. This is laborious work and often infeasible, but where the data does

exist, supervised learning algorithms can be extremely effective at a broad range of tasks.

Supervised machine learning tasks can be broadly classified into two subgroups: regression and classification. Regression is the problem of estimating or predicting a continuous quantity. What will be the value of the S&P 500 one month from today? How tall will a child be as an adult? How many of our customers will leave for a competitor this year? These are examples of questions that would fall under the umbrella of regression. To solve these problems in a supervised machine learning framework, we would gather past examples of “right answer” input/output pairs that deal with the same problem. For the inputs, we would identify features that we believe would be predictive of the outcomes that we wish to predict.

For the first problem, we might try to gather as features the historical prices of stocks under the S&P 500 on given dates along with the value of the S&P 500 one month later. This would form our training set, from which the machine would try to determine some functional relationship between the features and eventual S&P 500 values.

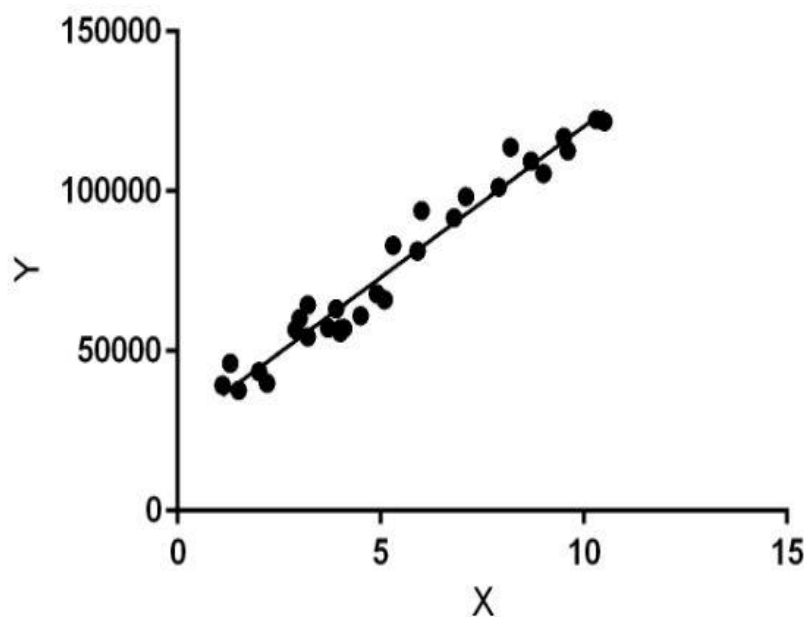
Classification deals with assigning observations into discrete categories, rather than estimating continuous quantities. In the simplest case, there are two possible categories; this case is known as binary classification. Many important questions can be framed in terms of binary classification. Will a given customer leave us for a competitor? Does a given patient have cancer? Does a given image contain a hot dog? Algorithms for performing binary classification are particularly important because many of the algorithms for performing the more general kind of classification where there are arbitrary labels are simply a bunch of binary classifiers working together. For instance, a simple solution to the handwriting recognition problem is to simply train a bunch of binary classifiers: a 0-detector, a 1-detector, a 2-detector, and so on, which output their certainty that the image is of their respective digit. The classifier just outputs the digit whose classifier has the highest certainty

---

## **REGRESSION**

## 2.1 ML | Linear Regression

**Linear Regression** is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.



**FIGURE 2.1**

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is **Linear Regression**.

## 2.2 What Is Multiple Linear Regression?

---

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

**Equation:**

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_p x_{ip} + \epsilon$$

**Example:**

An analyst may want to know how the movement of the market affects the price of Exxon Mobil (XOM). In this case, his linear equation will have the value of the S&P 500 index as the independent variable, or predictor, and the price of XOM as the dependent variable.

In reality, there are multiple factors that predict the outcome of an event. The price movement of Exxon Mobil, for example, depends on more than just the performance of the overall market. Other predictors such as the price of oil, interest rates, and the price movement of oil futures can affect the price of XOM and stock prices of other oil companies. To understand a relationship in which more than two variables are present, a multiple linear regression is used.

Multiple linear regression (MLR) is used to determine a mathematical relationship among a number of random variables. In other terms, MLR examines how multiple independent variables are related to one

## Module 2: Supervised Machine Learning Techniques

dependent variable. Once each of the independent factors has been determined to predict the dependent variable, the information on the multiple variables can be used to create an accurate prediction on the level of effect they have on the outcome variable. The model creates a relationship in the form of a straight line (linear) that best approximates all the individual data points.

Referring to the MLR equation above, in our example:

$y_i$  = dependent variable: price of XOM

$x_{i1}$  = interest rates

$x_{i2}$  = oil price

$x_{i3}$  = value of S&P 500 index

$x_{i4}$  = price of oil futures

$B_0$  = y-intercept at time zero

$B_1$  = regression coefficient that measures a unit change in the dependent variable when  $x_{i1}$  changes - the change in XOM price when interest rates change

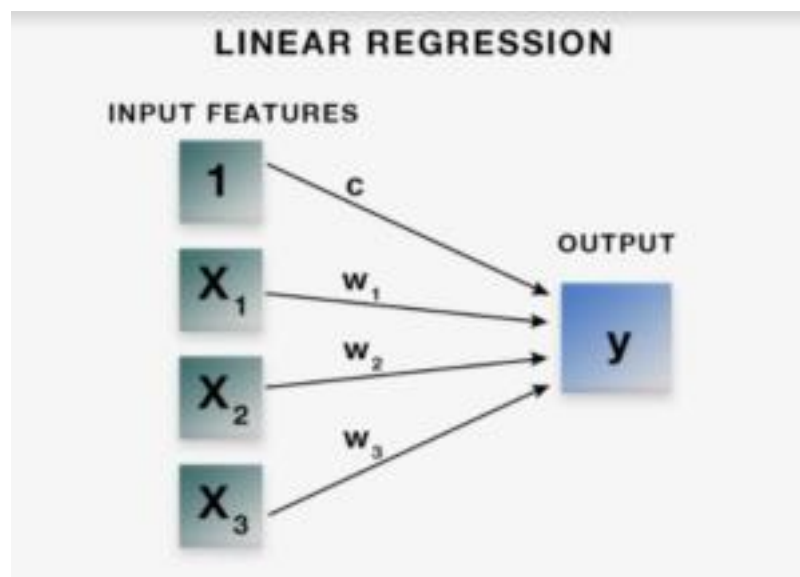
$B_2$  = coefficient value that measures a unit change in the dependent variable when  $x_{i2}$  changes - the change in XOM price when oil prices change

And so on for  $B_3$  and  $B_4$

## 2.3 Logistic Regression

Logistic regression falls under the category of supervised learning; it measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function. In spite of the name 'logistic regression', this is not used for machine learning regression problem where the task is to predict the real-valued output. It is used for classification problems which is used to predict a binary outcome (1/0, -1/1, True/False) given a set of independent variables.

Logistic regression is a bit similar to the linear regression or we can say it as a generalized linear model. In linear regression, we predict a real-valued output 'y' based on a weighted sum of input variables.



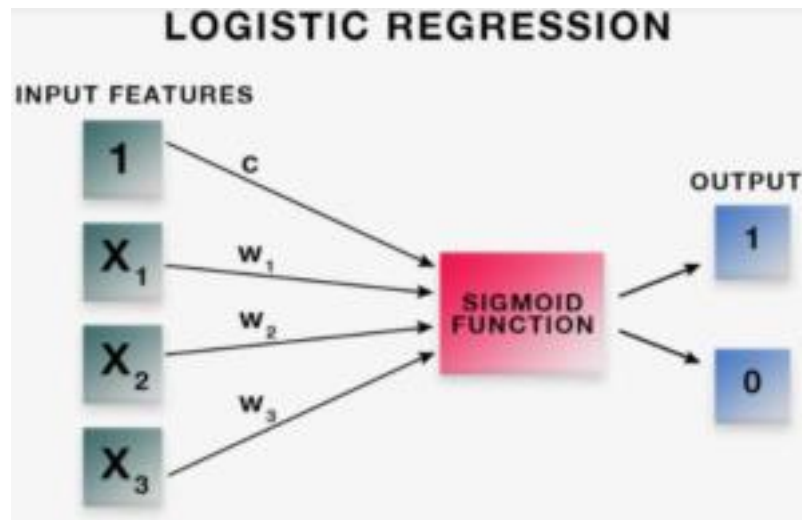
$$y = c + x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \dots + x_n * w_n$$

The aim of linear regression is to estimate values for the model coefficients  $c$ ,  $w_1$ ,  $w_2$ ,  $w_3$  ....  $w_n$  and fit the training data with minimal squared error and predict the output  $y$ .

Logistic regression does the same thing, but with one addition. The logistic regression model computes a weighted sum of the input variables similar to the linear regression, but it runs the result through a special



non-linear function, the logistic function or sigmoid function to produce the output  $y$ . Here, the output is binary or in the form of 0/1 or -1/1.



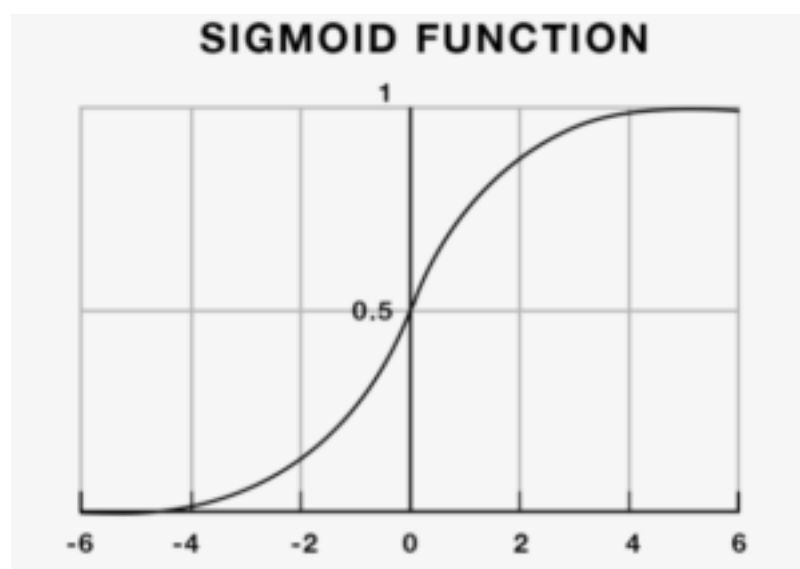
$$y = \text{logistic} (c + x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \dots + x_n * w_n)$$

$$y = 1 / 1 + e^{-(c + x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \dots + x_n * w_n)}$$

The sigmoid/logistic function is given by the following equation.

$$y = 1 / 1 + e^{-x}$$

As you can see in the graph, it is an S-shaped curve that gets closer to 1 as the value of input variable increases above 0 and gets closer to 0 as the input variable decreases below 0. The output of the sigmoid function is 0.5 when the input variable is 0.



**FIGURE 2.2**

Thus, if the output is more than 0.5, we can classify the outcome as 1 (or positive) and if it is less than 0.5, we can classify it as 0 (or negative).

Now, let us consider the task of predicting the stock price movement. If tomorrow's closing price is higher than today's closing price, then we will buy the stock (1), else we will sell it (-1). If the output of analysis using logistic regression is 0.7, then we can say that there is a 70% chance that tomorrow's closing price is higher than today's closing price and classify it as 1.

## 2.4 Hypothesis Testing

---

The process of hypothesis testing is to draw inferences or some conclusion about the overall population or data by conducting some statistical tests on a sample. The same inferences are drawn for different machine learning models through T-test which I will discuss in this tutorial.

For drawing some inferences, we have to make some assumptions that lead to two terms that are used in the hypothesis testing.

- Null hypothesis: It is regarding the assumption that there is no anomaly pattern or believing according to the assumption made.
- Alternate hypothesis: Contrary to the null hypothesis, it shows that observation is the result of real effect.

### **P value**

It can also be said as evidence or level of significance for the null hypothesis or in machine learning algorithms. It's the significance of the predictors towards the target.

## Module 2: Supervised Machine Learning Techniques

Generally, we select the level of significance by 5 %, but it is also a topic of discussion for some cases. If you have a strong prior knowledge about your data functionality, you can decide the level of significance.

On the contrary if the p-value is less than 0.05 in a machine learning model against an independent variable, then the variable is considered as valuable predictor which means there is heterogeneous behavior with the target which is useful and can be learned by the machine learning algorithms.

The steps involved in the hypothesis testing are as follow: Assume a null hypothesis, usually in machine learning algorithms we consider that there is no dependence between the target and independent variable.

- Collect a sample
- Calculate test statistics
- Decide either to accept or reject the null hypothesis

### **Calculating test or T statistics**

For Calculating T statistics, we create a scenario.

Suppose there is a shipping container making company which claims that each container is 1000 kg in weight not less, not more. Well, such claims look shady, so we proceed with gathering data and creating a sample.

## Module 2: Supervised Machine Learning Techniques

After gathering a sample of 30 containers, we found that the average weight of the container is 990 kg and showing a standard deviation of 12.5 kg.

So calculating test statistics:

$$T = (\text{Mean} - \text{Claim}) / (\text{Standard deviation} / \text{Sample Size}^{(1/2)})$$

Which is -4.3818 after putting all the numbers.

Now we calculate t value for 0.05 significance and degree of freedom.

Note: Degree of Freedom = Sample Size - 1

From T table the value will be -1.699.

<b>t Table</b>						
cum. prob	$t_{.50}$	$t_{.75}$	$t_{.80}$	$t_{.85}$	$t_{.90}$	$t_{.95}$
one-tail	0.50	0.25	0.20	0.15	0.10	0.05
two-tails	1.00	0.50	0.40	0.30	0.20	0.10
df						
1	0.000	1.000	1.376	1.963	3.078	6.314
2	0.000	0.816	1.061	1.386	1.886	2.920
3	0.000	0.765	0.978	1.250	1.638	2.353
4	0.000	0.741	0.941	1.190	1.533	2.132
5	0.000	0.727	0.920	1.156	1.476	2.015
6	0.000	0.718	0.906	1.134	1.440	1.943
7	0.000	0.711	0.896	1.119	1.415	1.895
8	0.000	0.706	0.889	1.108	1.397	1.860
9	0.000	0.703	0.883	1.100	1.383	1.833
10	0.000	0.700	0.879	1.093	1.372	1.812

**TABLE 2.1: Standard T table**

**Standard T table** is shown above. After comparison, we can see that the generated statistics are less than the statistics of the desired level of significance. So we can reject the claim made.

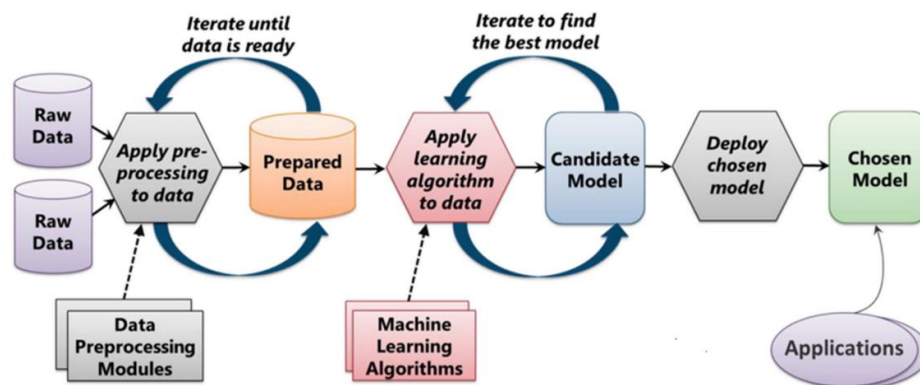
You can calculate the t value using stats.t.ppf() function of stats class of scipy library.

### Errors

As hypothesis testing is done on a sample of data rather than the entire population due to the unavailability of the resources in terms of data. Due to inferences are drawn on sample data the hypothesis testing can lead to errors, which can be classified into two parts:

- Type I Error: In this error, we reject the null hypothesis when it is true.
- Type II Error: In this error, we accept the null hypothesis when it is false.

### The Machine Learning Process



From "Introduction to Microsoft Azure" by David Chappell

## **DECISION TREES**

## 3.1 Introduction to Decision Tree

---

- ❖ Decision Tree learning is one of the predictive modeling techniques. Decision trees used in data mining are of two main types:
  - **Classification tree** - when the predicted outcome is the class to which the data belongs.
  - **Regression tree** - when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).
- ❖ Decision Tree breaks down a data set into smaller subsets and presents association between target variable (dependent) and independent variables as a tree structure. A 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.
- ❖ The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both of the above procedures, first introduced by Breiman et al.

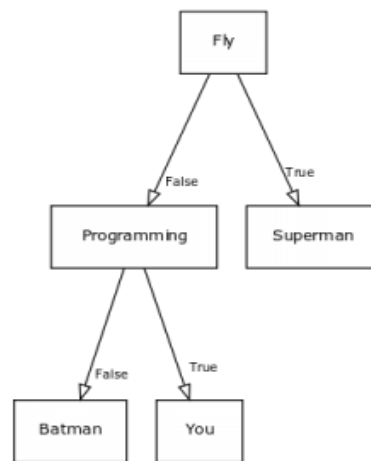
### Types of Decision Tree:

- ❖ 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:
  - ID3 (Iterative Dichotomiser 3)
  - C4.5 (successor of ID3)
  - CART (Classification And Regression Tree)
  - CHAID (CHi-squared Automatic Interaction Detector). Performs multi- level splits when computing classification trees.
  - MARS: Extends decision trees to handle numerical data better.

- 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.

### ❖ Decision Tree Visual Example

- At every node of the tree, we can turn left or right. Based on numbers we walk the branches. At the end of branches are outcomes. Once the classifier is trained based on this data. We can then use the classifier to make predictions.



**FIGURE 3.1: A graphical example of a decision tree**

Based on numeric input, a computer can decide the output. If its input would be [False, True], it would predict 'You'.

### Assumptions we make while using Decision tree

- ❖ In the beginning, the whole training set is considered at the root.
- ❖ Feature values are preferred to be categorical. If values are continuous then they are discretized prior to building the model.
- ❖ Records are distributed recursively on the basis of attribute values.



- ❖ Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

### **Decision Tree Algorithm Pseudocode**

1. Place the best attribute of our dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

While building our decision tree classifier, we can improve its accuracy by tuning it with different parameters. But this tuning should be done carefully since by doing this our algorithm can overfit on our training data & ultimately it will build bad generalization model.

### **Sklearn Library Installation**

Python's sklearn library holds tons of modules that help to build predictive models. It contains tools for data splitting, pre-processing, feature selection, tuning and supervised – unsupervised learning algorithms, etc. It is similar to Caret library in R programming.

For using it, we first need to install it. The best way to install data science libraries and its dependencies is by installing Anaconda package. You can also install only the most popular machine learning Python libraries.

Sklearn library provides us direct access to a different module for training our model with different machine learning algorithms like K-nearest neighbor classifier, Support vector machine classifier, decision tree, linear regression, etc.

### **Balance Scale Data Set Description**

Balance Scale data set consists of 5 attributes, 4 as feature attributes and 1 as the target attribute. We will try to build a classifier for predicting the Class attribute. The index of target attribute is 1st.

- 1.: 3 (L, B, R)
2. Left-Weight: 5 (1, 2, 3, 4, 5)
3. Left-Distance: 5 (1, 2, 3, 4, 5)

4. Right-Weight: 5 (1, 2, 3, 4, 5)

5. Right-Distance: 5 (1, 2, 3, 4, 5)

Index	Variable Name	Variable Values
1.	Class Name( Target Variable)	“R” : balance scale tip to the right “L” : balance scale tip to the left “B” : balance scale be balanced
2.	Left-Weight	1, 2, 3, 4, 5
3.	Left-Distance	1, 2, 3, 4, 5
4.	Right-Weight	1, 2, 3, 4, 5
5.	Right-Distance	1, 2, 3, 4, 5

**TABLE3.1**

The above table shows all the details of data.

### **Balance Scale Problem Statement**

The problem we are going to address is To model a classifier for evaluating balance tip’s direction.

### **Decision Tree classifier implementation in Python with sklearn Library**

The modeled Decision Tree will compare the new records metrics with the prior records (training data) that correctly classified the balance scale’s tip direction.

### **Python packages used**

#### **❖ NumPy**

- NumPy is a Numeric Python module. It provides fast mathematical functions.

## Module 2: Supervised Machine Learning Techniques

- Numpy provides robust data structures for efficient computation of multi-dimensional arrays & matrices.
- We used numpy to read data files into numpy arrays and data manipulation.

### ❖ Pandas

- Provides DataFrame Object for data manipulation
- Provides reading & writing data b/w different files.
- DataFrames can hold different types data of multidimensional arrays.

### ❖ Scikit-Learn

- It's a machine learning library. It includes various machine learning algorithms.
- We are using its
  - train\_test\_split,
  - DecisionTreeClassifier,
  - accuracy\_score algorithms.

*If you haven't setup the machine learning setup in your system the below posts will helpful.*

[Python Machine learning setup in ubuntu  
\(https://dataaspirant.com/2014/11/01/python-packages-for-datamining/\)](https://dataaspirant.com/2014/11/01/python-packages-for-datamining/)

[Python machine learning virtual environment setup  
\(https://dataaspirant.com/2016/03/22/python-datamining-packages-virtual-environment-setup-in-ubuntu/\)](https://dataaspirant.com/2016/03/22/python-datamining-packages-virtual-environment-setup-in-ubuntu/)

## Importing Python Machine Learning Libraries

This section involves importing all the libraries we are going to use. We are importing numpy and sklearn train\_test\_split, DecisionTreeClassifier & accuracy\_score modules.

```
1 import numpy as np
2 import pandas as pd
```

## Module 2: Supervised Machine Learning Techniques

```
3 from sklearn.cross_validation import train_test_split
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.metrics import accuracy_score
6 from sklearn import tree
```

Numpy arrays and pandas dataframes will help us in manipulating data. As discussed above, sklearn is a machine learning library. The cross\_validation's **train\_test\_split()** method will help us by splitting data into train & test set.

The tree module will be used to build a Decision Tree Classifier. Accuracy\_score module will be used to calculate accuracy metrics from the predicted class variables

### Data Import

For importing the data and manipulating it, we are going to use pandas dataframes. First of all, we need to download the dataset. You can download the dataset from [here](#). All the data values are separated by commas.

After downloading the data file, we will use Pandas **read\_csv()** method to import data into pandas dataframe. Since our data is separated by commas “,” and there is no header in our data, so we will put header parameter value “None” and sep parameter's value as “,”.

```
1 balance_data = pd.read_csv(
2 'https://archive.ics.uci.edu/ml/machine-learning-
3 databases/balance-scale/balance-scale.data',
                                     sep= ',', header= None)
```

We are saving our data into “balance\_data” dataframe.

For checking the length & dimensions of our dataframe, we can use len() method & “.shape”.

```
1 print "Dataset Length:: ", len(balance_data)
2 print "Dataset Shape:: ", balance_data.shape
```

### Output:

```
1 Dataset Length:: 625
2 Dataset Shape:: (625, 5)
```

We can print head .e, top 5 lines of our dataframe using **head()** method.

```
1 print "Dataset:: "
2 balance_data.head()
```

```
1 Dataset::
```

### Output:

0	1	2	3	4	
0					
1					
2					
3					
4					

### Data Slicing

Data slicing is a step to split data into train and test set. Training data set can be used specifically for our model building. Test dataset should not be mixed up while building model. Even during standardization, we should not standardize our test set.

```
1 X = balance_data.values[:, 1:5]
2 Y = balance_data.values[:, 1]
```

The above snippet divides data into feature set & target set. The “X ” set consists of predictor variables. It consists of data from 2nd column to 5th column. The “Y” set consists of the outcome variable. It consists of data in the 1st column. We are using “.values” of numpy converting our dataframes into numpy arrays.

Let’s split our data into training and test set. We will use sklearn’s train\_test\_split() method.

```
1 X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3,
    random_state = 100)
```

The above snippet will split data into training and test set. **X\_train**, **y\_train** are training data & **X\_test**, **y\_test** belongs to the test dataset.

The parameter test\_size is given value **0.3**; it means test sets will be **30%** of whole dataset & training dataset’s size will be **70%** of the entire dataset. random\_state variable is a pseudo-random number generator state used for random sampling. If you want to replicate our results, then use the same value of random\_state.

### Decision Tree Training

Now we fit Decision tree algorithm on training data, predicting labels for validation dataset and printing the accuracy of the model using various parameters.

**DecisionTreeClassifier()**: This is the classifier function for DecisionTree. It is the main function for implementing the algorithms. Some important parameters are:

- ❖ **criterion**: It defines the function to measure the quality of a split. Sklearn supports “gini” criteria for Gini Index & “entropy” for Information Gain. By default, it takes “gini” value.

- ❖ **splitter:** It defines the strategy to choose the split at each node. Supports “best” value to choose the best split & “random” to choose the best random split. By default, it takes “best” value.
- ❖ **max\_features:** It defines the no. of features to consider when looking for the best split. We can input integer, float, string & None value.
  - If an integer is inputted then it considers that value as max features at each split.
  - If float value is taken then it shows the percentage of features at each split.
  - If “auto” or “sqrt” is taken then  $\text{max\_features} = \sqrt{n\_features}$ .
  - If “log2” is taken then  $\text{max\_features} = \log_2(n\_features)$ .
  - If None, then  $\text{max\_features} = n\_features$ . By default, it takes “None” value.
- ❖ **max\_depth:** The max\_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples. By default, it takes “None” value.
- ❖ **min\_samples\_split:** This tells above the minimum no. of samples reqd. to split an internal node. If an integer value is taken then consider min\_samples\_split as the minimum no. If float, then it shows percentage. By default, it takes “2” value.
- ❖ **min\_samples\_leaf:** The minimum number of samples required to be at a leaf node. If an integer value is taken then consider min\_samples\_leaf as the minimum no. If float, then it shows percentage. By default, it takes “1” value.
- ❖ **max\_leaf\_nodes:** It defines the maximum number of possible leaf nodes. If None then it takes an unlimited number of leaf nodes. By default, it takes “None” value.
- ❖ **min\_impurity\_split:** It defines the threshold for early stopping tree growth. A node will split if its impurity is above the threshold otherwise it is a leaf.

Let's build classifiers using criterion as gini index & information gain. We need to fit our classifier using fit(). We will plot our decision tree classifier's visualization too.

### Decision Tree Classifier with criterion gini index

```
1 clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,  
2                                 max_depth=3, min_samples_leaf=5)  
3 clf_gini.fit(X_train, y_train)
```

#### Output:

```
1 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,  
2                         max_features=None, max_leaf_nodes=None, min_samples_leaf=5,  
3                         min_samples_split=2, min_weight_fraction_leaf=0.0,  
4                         presort=False, random_state=100, splitter='best')
```

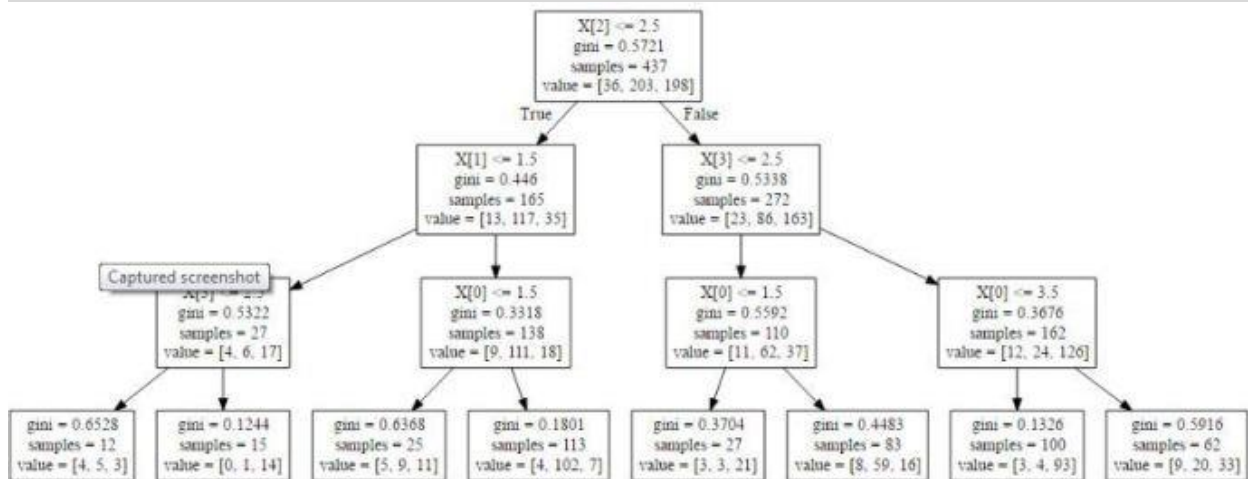


FIGURE 3.2

### Decision Tree Classifier with criterion information gain

```
1 clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,  
2                                     max_depth=3, min_samples_leaf=5)  
3 clf_entropy.fit(X_train, y_train)
```

```
1 DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
```



```
2 max_features=None, max_leaf_nodes=None, min_samples_leaf=5,
3 min_samples_split=2, min_weight_fraction_leaf=0.0,
4 presort=False, random_state=100, splitter='best')
```

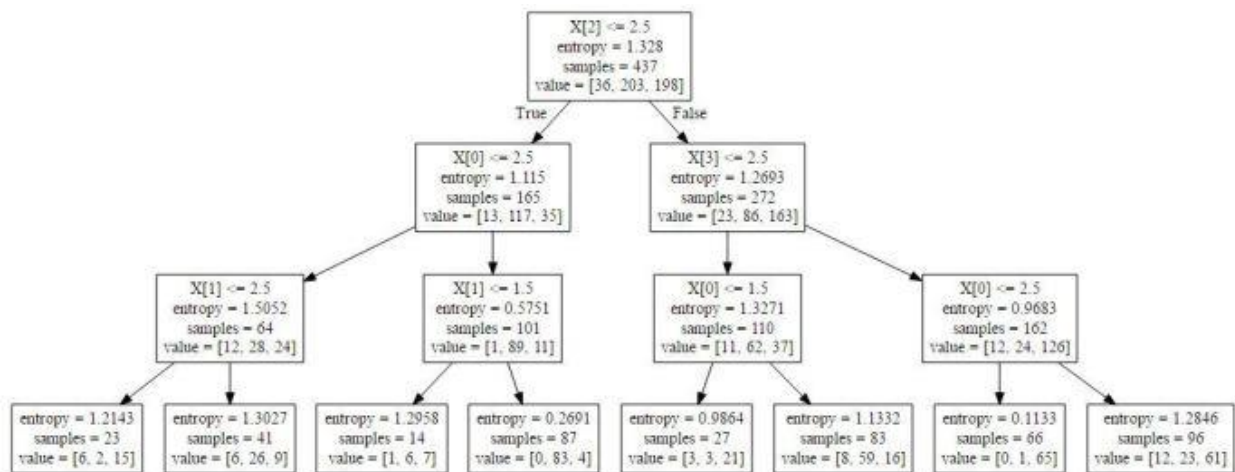


FIGURE 3.3

## Prediction

Now, we have modeled 2 classifiers. One classifier with gini index & another one with information gain as the criterion. We are ready to predict classes for our test set. We can use `predict()` method. Let's try to predict target variable for test set's 1st record.

```
1 clf_gini.predict([[4, 4, 3, 3]])
```

## Output

```
1 array(['R'], dtype=object)
```

This way we can predict class for a single record. It's time to predict target variable for the whole test dataset.

### Prediction for Decision Tree classifier with criterion as gini index

```
1 y_pred = clf_gini.predict(X_test)
2 y_pred
```

#### Output

```
1 array(['R', 'L', 'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'R', 'L', 'L',
2        'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L',
3        'L', 'L', 'R', 'L', 'L', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'L',
4        'L', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'R',
5        'L', 'R', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'R', 'L', 'L', 'L',
6        'L', 'L', 'R', 'R', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R', 'R',
7        'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R',
8        'R', 'L', 'L', 'L', 'R', 'R', 'L', 'L', 'L', 'R', 'L', 'R', 'R',
9        'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R',
10       'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'R',
11       'R', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'L',
12       'R', 'L', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'R', 'R', 'R',
13       'L', 'R', 'R', 'R', 'R', 'R', 'L', 'L', 'R', 'R', 'R', 'R', 'L',
14       'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R',
15       'R', 'L', 'L', 'R', 'R', 'R'], dtype=object)
```

### Prediction for Decision Tree classifier with criterion as information gain

```
1 y_pred_en = clf_entropy.predict(X_test)
2 y_pred_en
```

### Output

```
1  array(['R', 'L', 'R', 'L', 'R', 'L', 'R', 'L', 'R', 'R', 'R', 'R', 'L',
2      'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L',
3      'R', 'L', 'R', 'L', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'R',
4      'L', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'L', 'R',
5      'L', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R',
6      'L', 'L', 'R', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R', 'R',
7      'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R',
8      'R', 'L', 'L', 'L', 'R', 'R', 'L', 'L', 'L', 'R', 'L', 'L', 'R',
9      'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R',
10     'L', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'L', 'L', 'L', 'R',
11     'R', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'L',
12     'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'R', 'R', 'R', 'R',
13     'L', 'R', 'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L',
14     'R', 'L', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'R', 'R', 'R',
15     'L', 'L', 'L', 'R', 'R', 'R'], dtype=object)
```

### Calculating Accuracy Score

The function `accuracy_score()` will be used to print accuracy of Decision Tree algorithm. By accuracy, we mean the ratio of the correctly predicted data points to all the predicted data points. Accuracy as a metric helps to understand the effectiveness of our algorithm. It takes 4 parameters.

- ❖ `y_true`,
- ❖ `y_pred`,
- ❖ `normalize`,
- ❖ `sample_weight`.

Out of these 4, `normalize` & `sample_weight` are optional parameters. The parameter `y_true` accepts an array of correct labels and `y_pred` takes an array of predicted labels that are returned by the classifier. It returns accuracy as a float value.

### **Accuracy for Decision Tree classifier with criterion as gini index**

```
1 print "Accuracy is ", accuracy_score(y_test,y_pred)*100
```

#### **Output**

```
1 Accuracy is 73.4042553191
```

### **Accuracy for Decision Tree classifier with criterion as information gain**

```
1 print "Accuracy is ", accuracy_score(y_test,y_pred_en)*100
```

#### **Output**

```
1 Accuracy is 70.7446808511
```

### **Conclusion**

In this article, we have learned how to model the decision tree algorithm in Python using the Python machine learning library `scikit-learn`. In the process, we learned how to split the data into train and test dataset. To model decision tree classifier we used the information gain, and gini index split criteria. In the end, we calculate the accuracy of these two decision tree models.

---

## **NAÏVE BAYES**

## 4.1 Introduction to Naive Bayes

---

A Naive Bayes Classifier is a supervised machine-learning algorithm that uses the Bayes' Theorem, which assumes that features are statistically independent. The theorem relies on the *naïve* assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results.

## 4.2 What Is the Bayes' Theorem?

---

Naive Bayes Classifiers rely on the Bayes' Theorem, which is based on conditional probability or in simple terms, the likelihood that an event (A) will happen *given that* another event (B) has already happened. Essentially, the theorem allows a hypothesis to be updated each time new evidence is introduced. The equation below expresses Bayes' Theorem in the language of probability:

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

Let's explain what each of these terms means.

- “P” is the symbol to denote probability.
- $P(A | B)$  = The probability of event A (hypothesis) occurring given that B (evidence) has occurred.
- $P(B | A)$  = The probability of the event B (evidence) occurring given that A (hypothesis) has occurred.
- $P(A)$  = The probability of event B (hypothesis) occurring.
- $P(B)$  = The probability of event A (evidence) occurring.

## Module 2: Supervised Machine Learning Techniques

```
1
2 training_set=pd.read_csv('./data/labeledTrainData.tsv',sep='\t') # reading the training data-set
3
4 #getting training set examples labels
5 y_train=training_set['sentiment'].values
6 x_train=training_set['review'].values
7
8 """
9     Again - it's not a problem at all if you didnt understand this block of code - You should just
10     train & test data is being loaded and saved in their corresponding variables
11 """
12
13
14 from sklearn.model_selection import train_test_split
15 train_data,test_data,train_labels,test_labels=train_test_split(x_train,y_train,shuffle=True,test_si
16 classes=np.unique(train_labels)
17
18 # Training phase...
19
20 nb=NaiveBayes(classes)
21 nb.train(train_data,train_labels)
22
23 # Testing phase
24
25 pclasses=nb.test(test_data)
26 test_acc=np.sum(pclasses==test_labels)/float(test_labels.shape[0])
27
28 print ("Test Set Accuracy: ",test_acc) # Output : Test Set Accuracy:  0.84224 :)
```

**FIGURE 4.1**

## **SUPPORT VECTOR MACHINE (SVM)**



## 5.1 Support Vector Machine

Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiate the two classes very well.

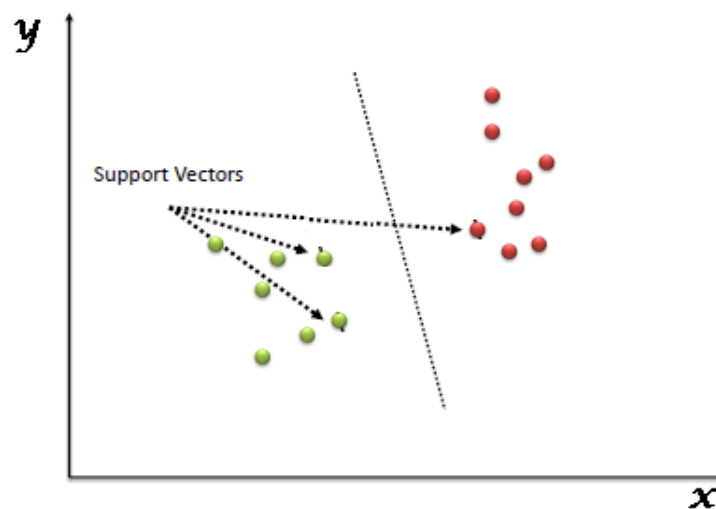


FIGURE 5.1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We
could
# avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not
scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1, gamma=0).fit(X, y)
```

```
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```

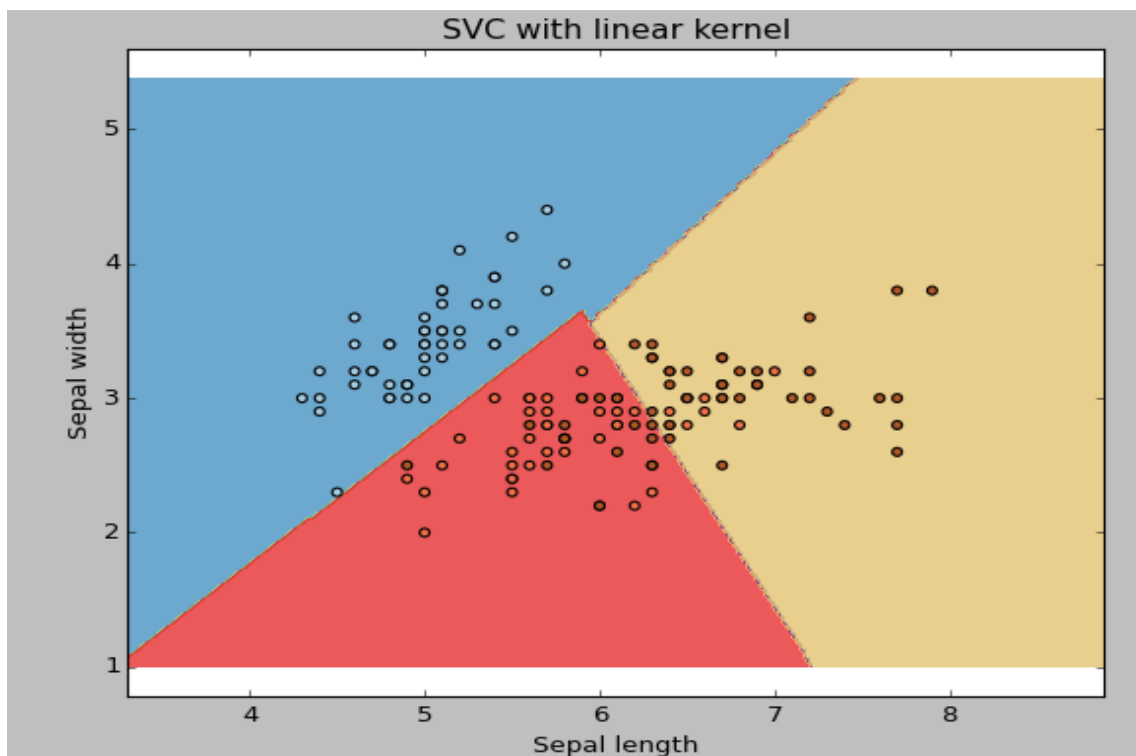


FIGURE 5.2

## **K-NEAREST NEIGHBOUR (KNN)**

## 6.1 K- Nearest Neighbor Classification

**KNN** is a **non-parametric, lazy** learning algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point. A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its **K** nearest neighbors measured by a distance function. If **K** = 1, then the case is simply assigned to the class of its nearest neighbor.

### Distance functions

Euclidean

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^k |x_i - y_i|$$

Minkowski

$$\left( \sum_{i=1}^k (|x_i - y_i|)^q \right)^{1/q}$$

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

**Hamming Distance**

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

**TABLE 6.1**

Choosing the optimal value for **K** is best done by first inspecting the data. In general, a large **K** value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good **K** value by using an independent dataset to validate the **K** value. Historically, the optimal **K** for most datasets has been between 3-10. That produces much better results than 1NN.

**K-nearest neighbor classifier implementation with scikit-learn**

All the code snippets can be typed directly to jupyter Ipython notebook.

### **Libraries:**

This section involves importing all the libraries. We are importing numpy and sklearn imputer, train\_test\_split, KNeighborsClassifier & accuracy\_score modules.

```
1 import numpy as np

2 from sklearn.preprocessing import Imputer

3 from sklearn.cross_validation import train_test_split

4 from sklearn.neighbors import KNeighborsClassifier

5 from sklearn.metrics import accuracy_score
```

### **Data Import:**

We are using breast cancer data. You can download it from [archive.ics.uci.edu](http://archive.ics.uci.edu) website. For importing the data and manipulating it, we are going to use numpy arrays.

Using genfromtxt() method, we are importing our dataset into the 2d numpy array.

You can import text files using this function. We are passing 3 parameters:

#### ❖ **fname**

- It handles the filename with extension.

### ❖ delimiter

- The string used to separate values. In our dataset “,”(comma) is the separator.

### ❖ dtype

- It handles data type of variables.

All the values are numeric in our database. But some values are missing and are replaced by “?”. So, we will have to perform data imputation. Due to this reason, we are using float dtype.

```
1 cancer_data = np.genfromtxt(  
2 fname='breast-cancer-wisconsin.data', delimiter= ',', dtype=  
float)
```

Using the above code we have imported our data into a 2d numpy array.

❖ **len()**: Function to find out the no. of records in our data.

❖ **str()**: Function to get an idea about the basic structure of data.

❖ **shape**: To get array dimensions.

```
1 print "Dataset Length:: ", len(cancer_data)  
2 print "Dataset:: ", str(cancer_data)  
3 print "Dataset Shape:: ", cancer_data.shape
```

## Module 2: Supervised Machine Learning Techniques

### Output:

```
1 Dataset Length:: 699
2 Dataset::
3 [[ 1.00002500e+06  5.00000000e+00  1.00000000e+00 ...,  1.00000000e+00
4     1.00000000e+00  2.00000000e+00]
5 [ 1.00294500e+06  5.00000000e+00  4.00000000e+00 ...,  2.00000000e+00
6     1.00000000e+00  2.00000000e+00]
7 [ 1.01542500e+06  3.00000000e+00  1.00000000e+00 ...,  1.00000000e+00
8     1.00000000e+00  2.00000000e+00]
9 ...,
10 [ 8.88820000e+05  5.00000000e+00  1.00000000e+01 ...,  1.00000000e+01
11     2.00000000e+00  4.00000000e+00]
12 [ 8.97471000e+05  4.00000000e+00  8.00000000e+00 ...,  6.00000000e+00
13     1.00000000e+00  4.00000000e+00]
14 [ 8.97471000e+05  4.00000000e+00  8.00000000e+00 ...,  4.00000000e+00
15     1.00000000e+00  4.00000000e+00]]
16
17 Dataset Shape:: (699L, 11L)
```

The cancer dataset first column consists of patient's id. To make this prediction process unbiased, we should remove this patient id. We can use `numpy delete()` method for this operation.

**`delete()`:** It returns a new transformed array. Three parameters should to passed.

❖ **`arr`:** It holds the array name.



- ❖ **obj:** It indicates which sub-arrays to remove.
- ❖ **axis:** The axis along which to delete. axis = 1 is used for columns & axis = 0 for rows.

```
1 cancer_data = np.delete(arr = cancer_data, obj= 0, axis = 1)
```

Now, we wish to divide the dataset into feature & label dataset. i.e., feature data is predictor variables they will help us to predict labels(criterion variable). Here, first 9 columns include continuous variables that will help us to predict whether a patient is having the benign tumor or malignant tumor.

```
1 X = cancer_data[:,range(0,9)]  
  
2 Y = cancer_data[:,9]
```

### **Data Imputation:**

Imputation is a process of replacing missing values with substituted values. In our dataset, some columns have missing values. We can replace missing values with mean, median, mode or any particular value.

Sklearn provides `Imputer()` method to perform imputation in 1 line of code. We just need to define `missing_values`, `axis`, and `strategy`. We are using “median” value of the column to substitute with the missing value.

## Module 2: Supervised Machine Learning Techniques

```
1 imp = Imputer(missing_values="NaN", strategy='median', axis=0)

2 X = imp.fit_transform(X)
```

### **Train, Test data split:**

For dividing data into train data & test data, we are using `train_test_split()` method by sklearn.

**`train_test_split()`:** We are using 4 parameters X, Y, test\_size, random\_state

- ❖ **X, Y:** X is a numpy array consisting of feature dataset & Y contains labels for each record.
- ❖ **test\_size:** It represents the size of test data needs to split. If we use 0.4, it indicates 40% of data should be separated and saved as testing data.
- ❖ **random\_state:** It's pseudo-random number generator state used for random sampling. If you want to replicate our results, then use the same value of random\_state.

Now, X\_train & y\_train are training datasets. X\_test & y\_test are testing datasets.

y\_train & y\_test are 2d numpy arrays with 1 column. To convert it into a 1d array, we are using `ravel()`.

```
1 X_train, X_test, y_train, y_test = train_test_split(
```

```
2 X, Y, test_size = 0.3, random_state = 100)

3 y_train = y_train.ravel()

4 y_test = y_test.ravel()
```

### **KNN Implementation:**

Now we are fitting KNN algorithm on training data, predicting labels for dataset and printing the accuracy of the model for different values of K(ranging from 1 to 25).

**KNeighborsClassifier()**: This is the classifier function for KNN. It is the main function for implementing the algorithms. Some important parameters are:

- ❖ **n\_neighbors**: It holds the value of K, we need to pass and it must be an integer. If we don't give the value of n\_neighbors then by default, it takes the value as 5.
- ❖ **Weights**: It holds a string value i.e., name of the weight function. The Weight function used in prediction. It can hold values like 'uniform' or 'distance' or any user defined function.
  - **'uniform'** weight used when all points in the neighborhood are weighted equally. Default value for weights taken as 'uniform'

## Module 2: Supervised Machine Learning Techniques

- **'distance'** weight used for giving closer neighbors- higher weight and far neighbors-less weight, i.e., weight points by the inverse of their distance.
- **user defined function** we can call the user defined functions. The user defined function can be used when we want to produce custom weight values. It accepts distance values and returns an array of weights.
- ❖ **algorithm:** It specifies algorithm which should be used to compute the nearest neighbors. It can have values like 'auto', 'ball\_tree', 'kd\_tree', 'brute'. It is an optional parameter.
  - a) 'ball\_tree' , 'kd\_tree' are used to implement ball tree algorithm. These are special kind of data structures for space partitioning.
  - b) 'brute' is used to implement brute-force search algorithm.
  - c) 'auto' is used to give control to the system. By using 'auto', it automatically decides the best algorithm according to values of training data.`fit()`
- ❖ **data.fit():** A fit method is used to fit the model. It is passed with two parameters: X and Y. For training data fitting on KNN algorithm, this needs to be called.
  - **X:** It consists of training data with features.
  - **Y:** It consists of training data with labels.`predict()`: It predicts class labels for the data provided as its parameters.

- ❖ If an array of features data is entered as parameters, then an array of labels is given as output.

### **Accuracy Score:**

**accuracy\_score()**: This function is used to print accuracy of KNN algorithm. By accuracy, we mean the ratio of the correctly predicted data points to all the predicted data points. Accuracy as a metric helps to understand the effectiveness of our algorithm. It takes 4 parameters.

- ❖ y\_true,
- ❖ y\_pred,
- ❖ normalize,
- ❖ sample\_weight.

Out of these 4, normalize & sample\_weight are optional parameters. The parameter y\_true accepts an array of correct labels and y\_pred takes an array of predicted labels that are returned by the classifier. It returns accuracy as a float value.

```
1 for K in range(25):  
2     K_value = K+1  
3     neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform',  
4                                 algorithm='auto')
```

## Module 2: Supervised Machine Learning Techniques

```
5 neigh.fit(X_train, y_train)

6 y_pred = neigh.predict(X_test)

print "Accuracy is ", accuracy_score(y_test,y_pred)*100,"% for K-
Value:",K_value
```

### **Output:**

```
1 Accuracy is 95.2380952381 % for K-Value: 1

2 Accuracy is 93.3333333333 % for K-Value: 2

3 Accuracy is 95.7142857143 % for K-Value: 3

4 Accuracy is 95.2380952381 % for K-Value: 4

5 Accuracy is 95.7142857143 % for K-Value: 5

6 Accuracy is 94.7619047619 % for K-Value: 6

7 Accuracy is 94.7619047619 % for K-Value: 7

8 Accuracy is 94.2857142857 % for K-Value: 8

9 Accuracy is 94.7619047619 % for K-Value: 9

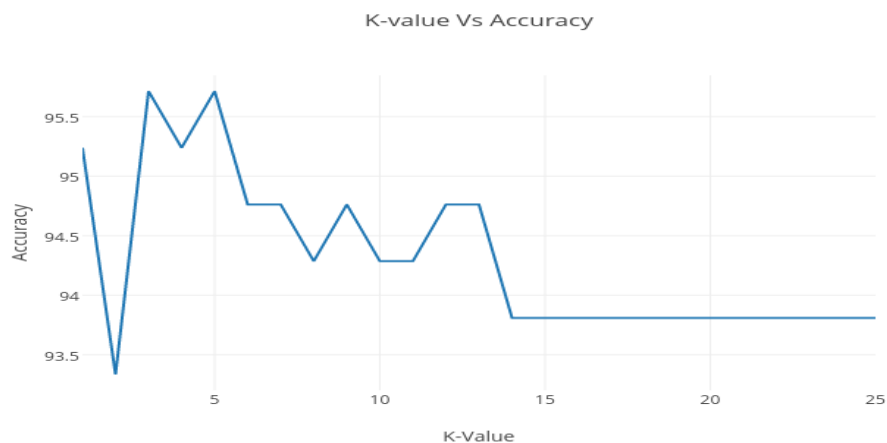
1 Accuracy is 94.2857142857 % for K-Value: 10
0 Accuracy is 94.2857142857 % for K-Value: 11

1 Accuracy is 94.7619047619 % for K-Value: 12
1 Accuracy is 94.7619047619 % for K-Value: 13
1 Accuracy is 93.8095238095 % for K-Value: 14
2
```

## Module 2: Supervised Machine Learning Techniques

```
1 Accuracy is 93.8095238095 % for K-Value: 15
3 Accuracy is 93.8095238095 % for K-Value: 16
1 Accuracy is 93.8095238095 % for K-Value: 17
4 Accuracy is 93.8095238095 % for K-Value: 18
1 Accuracy is 93.8095238095 % for K-Value: 19
5 Accuracy is 93.8095238095 % for K-Value: 20
1 Accuracy is 93.8095238095 % for K-Value: 21
6 Accuracy is 93.8095238095 % for K-Value: 22
1 Accuracy is 93.8095238095 % for K-Value: 23
7 Accuracy is 93.8095238095 % for K-Value: 24
1 Accuracy is 93.8095238095 % for K-Value: 25
8 Accuracy is 93.8095238095 % for K-Value: 25
```

**K value vs Accuracy Change Graph**



**FIGURE 6.1**

## **RANDOM FOREST**



## 7.1 Random Forest Classification

---

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it.

### Table of Contents:

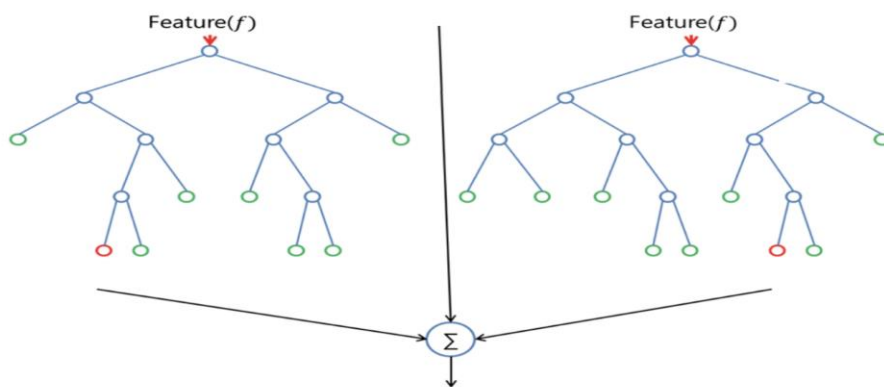
- How it works
- Real Life Analogy
- Feature Importance
- Difference between Decision Trees and Random Forests
- Important Hyperparameters (predictive power, speed)
- Advantages and Disadvantages
- Use Cases
- Summary

### How it works:

Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

*To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.*

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



**FIGURE 7.1**

Random Forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, you don't have to combine a decision tree with a bagging classifier and can just easily use the classifier-class of Random Forest. Like I already said, with Random Forest, you can also deal with Regression tasks by using the Random Forest regressor.

Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

### **Real Life Analogy:**

Imagine a guy named Andrew, that wants to decide, to which places he should travel during a one-year vacation trip. He asks people who know him for advice. First, he goes to a friend, that asks Andrew where he traveled to in the past and if he liked it or not. Based on the answers, he will give Andrew some advice.

This is a typical decision tree algorithm approach. Andrews friend created rules to guide his decision about what he should recommend, by using the answers of Andrew.

Afterwards, Andrew starts asking more and more of his friends to advise him and they again ask him different questions, where they can derive some recommendations from. Then he chooses the places that where recommend the most to him, which is the typical Random Forest algorithm approach.

### **Feature Importance:**

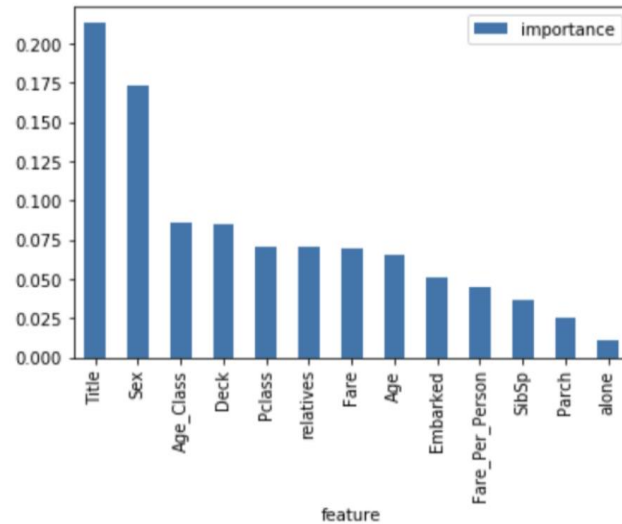
Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the sum of all importance is equal to 1.

If you don't know how a decision tree works and if you don't know what a leaf or node is, here is a good description from Wikipedia: In a decision tree each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). A node that has no children is a leaf.

Through looking at the feature importance, you can decide which features you may want to drop, because they don't contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa.

Below you can see a table and a visualization that show the importance of 13 features, which I used during a supervised classification project with the famous Titanic dataset on kaggle. You can find the whole project [here](https://www.kaggle.com/niklasdonges/end-to-end-project-with-python). (<https://www.kaggle.com/niklasdonges/end-to-end-project-with-python>)

feature	importance
Title	0.213
Sex	0.173
Age_Class	0.086
Deck	0.085
Pclass	0.071
relatives	0.070
Fare	0.069
Age	0.065
Embarked	0.051
Fare_Per_Person	0.045
SibSp	0.037
Parch	0.025
alone	0.011



**FIGURE 7.2**

### Difference between Decision Trees and Random Forests:

Like I already mentioned, Random Forest is a collection of Decision Trees, but there are some differences.

If you input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions.

For example, if you want to predict whether a person will click on an online advertisement, you could collect the ad's the person clicked in the past and some features that describe his decision. If you put the features and labels into a decision tree, it will generate some rules. Then you can predict whether the advertisement will be clicked or not. In comparison, the Random Forest algorithm randomly

selects observations and features to build several decision trees and then averages the results.

Another difference is that “deep” decision trees might suffer from overfitting. Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Afterwards, it combines the subtrees. Note that this doesn’t work every time and that it also makes the computation slower, depending on how many trees your random forest builds.

### **Important Hyperparameters:**

The Hyperparameters in random forest are either used to increase the predictive power of the model or to make the model faster. I will here talk about the hyperparameters of sklearn’s built-in random forest function.

#### **1. Increasing the Predictive Power**

Firstly, there is the “**n\_estimators**” hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.

Another important hyperparameter is “**max\_features**”, which is the maximum number of features Random Forest considers to split a node. Sklearn provides several options, described in their [documentation](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html). (<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>)

The last important hyper-parameter we will talk about in terms of speed, is “**min\_sample\_leaf**”. This determines, like its name already says, the minimum number of leafs that are required to split an internal node.

### **2. Increasing the Models Speed**

The “**n\_jobs**” hyperparameter tells the engine how many processors it is allowed to use. If it has a value of 1, it can only use one processor. A value of “-1” means that there is no limit.

“**random\_state**” makes the model’s output replicable. The model will always produce the same results when it has a definite value of random\_state and if it has been given the same hyperparameters and the same training data.

Lastly, there is the “**oob\_score**” (also called oob sampling), which is a random forest cross validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out of bag samples. It is very similar to the leave-one-out cross-validation method, but almost no additional computational burden goes along with it.

### **Advantages and Disadvantages:**

Like I already mentioned, an advantage of random forest is that it can be used for both regression and classification tasks and that it's easy to view the relative importance it assigns to the input features.

Random Forest is also considered as a very handy and easy to use algorithm, because its default hyperparameters often produce a good prediction result. The number of hyperparameters is also not that high and they are straightforward to understand.

One of the big problems in machine learning is overfitting, but most of the time this won't happen that easy to a random forest classifier. That's because if there are enough trees in the forest, the classifier won't overfit the model.

The main limitation of Random Forest is that a large number of trees can make the algorithm to slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.

And of course, Random Forest is a predictive modeling tool and not a descriptive tool. That means, if you are looking for a description of the relationships in your data, other approaches would be preferred.



### Example

You've already seen the code to load the data a few times. At the end of data-loading, we have the following variables:

- train\_X
- val\_X
- train\_y
- val\_y

We build a random forest model similarly to how we built a decision tree in scikit-learn - this time using the `RandomForestRegressor` class instead of `DecisionTreeRegressor`.

```
In [2]:
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
202888.18157951365
```

### Conclusion

There is likely room for further improvement, but this is a big improvement over the best decision tree error of 250,000. There are parameters which allow you to change the performance of the Random Forest much as we changed the maximum

depth of the single decision tree. But one of the best features of Random Forest models is that they generally work reasonably even without this tuning.

You'll soon learn the XGBoost model, which provides better performance when tuned well with the right parameters (but which requires some skill to get the right model parameters).

---

## **Frequently Asked Questions**

**Q1) Mention the difference between Data Mining and Machine learning?**

**Ans.** Machine learning relates with the study, design and development of the algorithms that give computers the capability to learn without being explicitly programmed. While, data mining can be defined as the process in which we try to extract knowledge or unknown interesting patterns from the given unstructured data. During this process machine learning algorithms are used.

**Q2) What is 'Overfitting' in Machine learning?**

**Ans.** In machine learning, when a statistical model maps random error or noise instead of underlying relationship 'overfitting' occurs. When a model is excessively complex, overfitting is normally observed, because of having too many parameters with respect to the number of training data types. Such a model exhibits poor performance and not fit for business use.

**Q3) Why overfitting happens?**

**Ans.** The possibility of overfitting exists as the criteria used for training the model is not the same as the criteria used to judge the efficacy of a model.

**Q4) How can you avoid overfitting?**

**Ans.** By using a lot of data, overfitting can be avoided, overfitting happens relatively as you have a small dataset, and you try to learn from it. But if you have a small database and you are forced to come with a model based on that. In such situation, you can use a technique known as cross validation. In this method the dataset splits into two section, testing and training datasets, the testing dataset will only test the model while, in training dataset, the datapoints will come up with the model.

In this technique, a model is usually given a dataset of a known data on which training (training data set) is run and a dataset of unknown data against which the model is tested. The idea of cross validation is to define a dataset to “test” the model in the training phase.

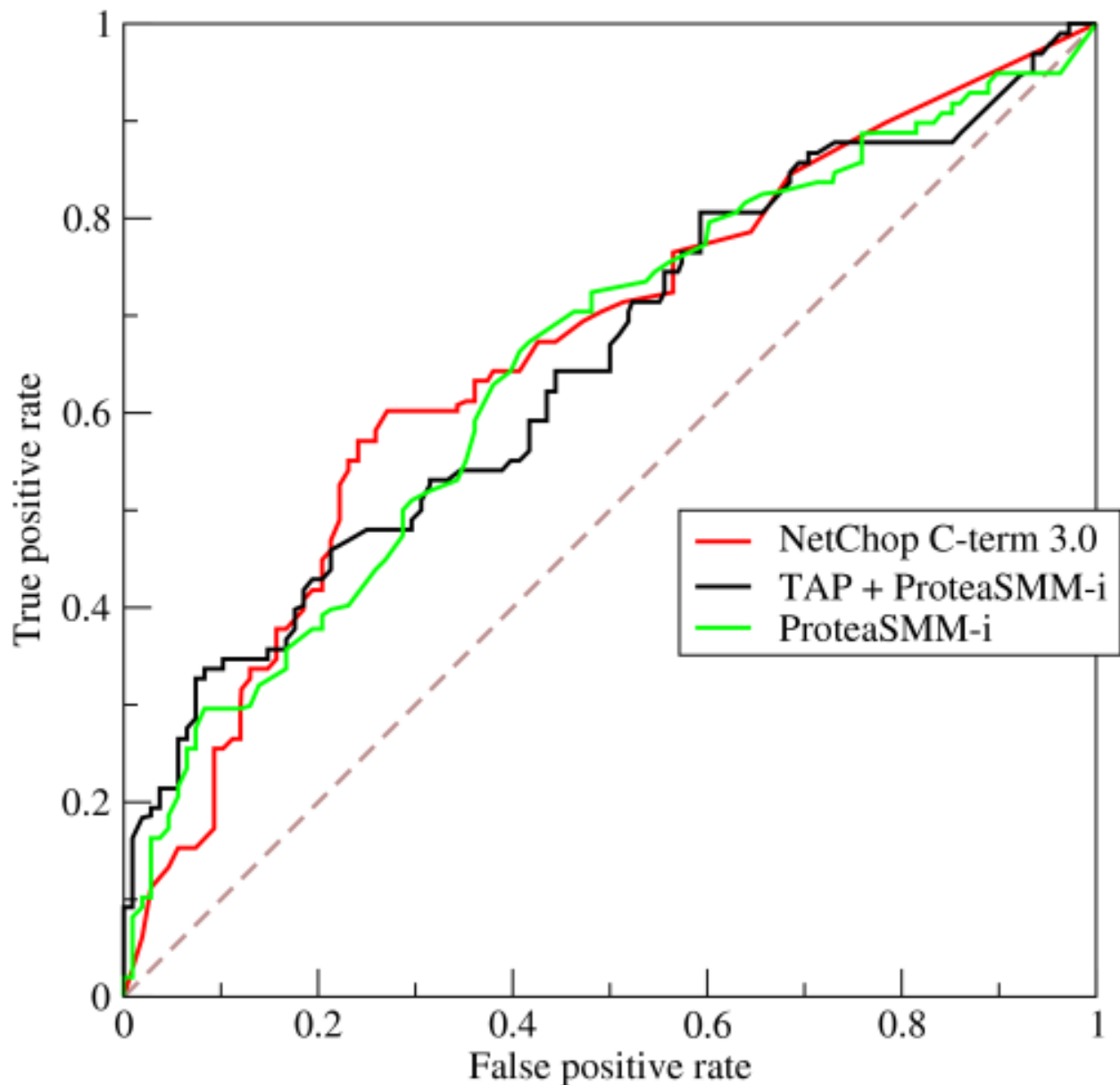
### **Q5) How is KNN different from k-means clustering?**

**Ans.** K-Nearest Neighbors is a supervised classification algorithm, while k-means clustering is an unsupervised clustering algorithm. While the mechanisms may seem similar at first, what this really means is that in order for K-Nearest Neighbors to work, you need labeled data you want to classify an unlabeled point into (thus the nearest neighbor part). K-means clustering requires only a set of unlabeled points and a threshold: the algorithm will take unlabeled points and gradually learn how to cluster them into groups by computing the mean of the distance between different points.

The critical difference here is that KNN needs labeled points and is thus supervised learning, while k-means doesn't — and is thus unsupervised learning.

### **Q6) Explain how a ROC curve works**

**Ans.** The ROC curve is a graphical representation of the contrast between true positive rates and the false positive rate at various thresholds. It's often used as a proxy for the trade-off between the sensitivity of the model (true positives) vs the fall-out or the probability it will trigger a false alarm (false positives).



#### Q7) Define precision and recall

**Ans.** Recall is also known as the true positive rate: the amount of positives your model claims compared to the actual number of positives there are throughout the data. Precision is also known as the positive predictive value, and it is a measure of the amount of accurate positives your model claims compared to the number of positives it actually claims. It can be easier to think of recall and precision in the context of a case where you've predicted that there were 10 apples and 5 oranges in

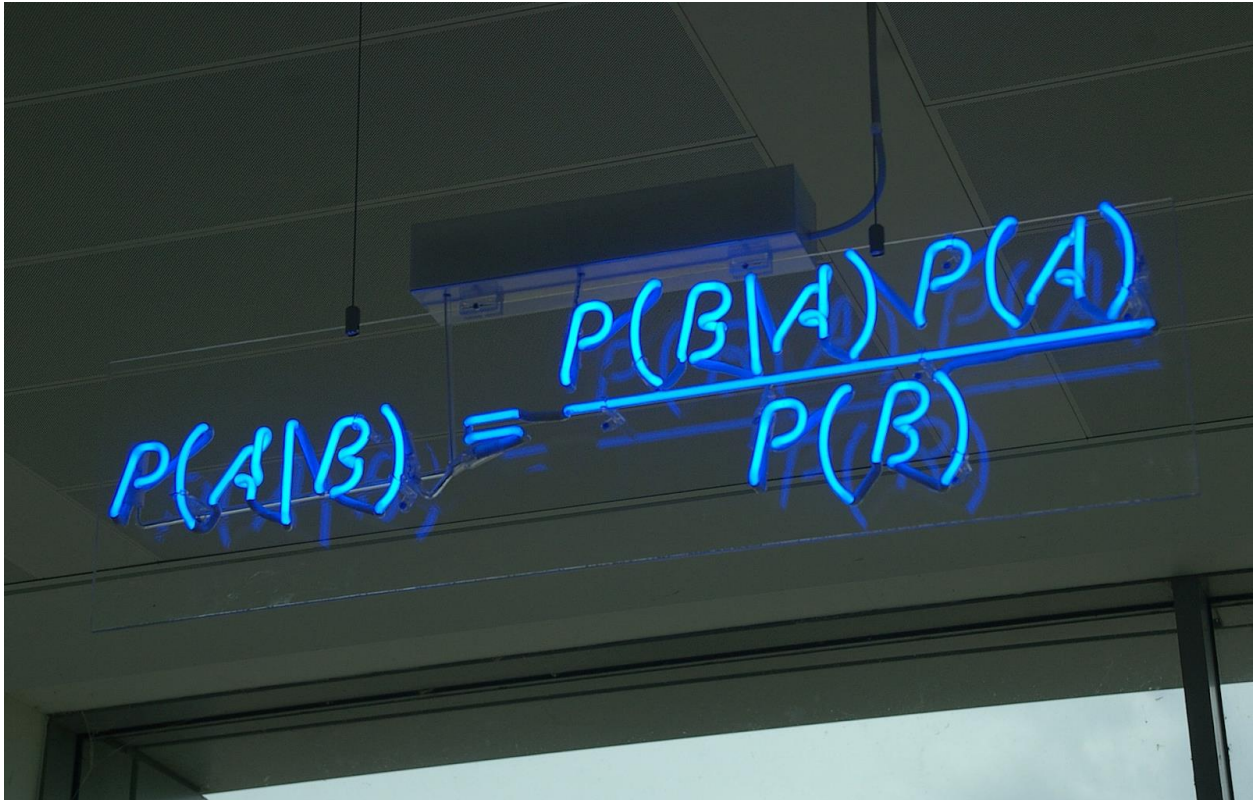
a case of 10 apples. You'd have perfect recall (there are actually 10 apples, and you predicted there would be 10) but 66.7% precision because out of the 15 events you predicted, only 10 (the apples) are correct.

### **Q8) What is Bayes' Theorem? How is it useful in a machine learning context?**

**Ans.** Bayes' Theorem gives you the posterior probability of an event given what is known as prior knowledge.

Mathematically, it's expressed as the true positive rate of a condition sample divided by the sum of the false positive rate of the population and the true positive rate of a condition. Say you had a 60% chance of actually having the flu after a flu test, but out of people who had the flu, the test will be false 50% of the time, and the overall population only has a 5% chance of having the flu. Would you actually have a 60% chance of having the flu after having a positive test?

Bayes' Theorem says no. It says that you have a  $(.6 * 0.05)$  (True Positive Rate of a Condition Sample) /  $(.6*0.05)(\text{True Positive Rate of a Condition Sample}) + (.5*0.95)$  (False Positive Rate of a Population) = 0.0594 or 5.94% chance of getting a flu.

A blue neon sign is mounted on a dark ceiling, displaying the formula for Bayes' Theorem. The formula is written in a stylized, glowing blue font. It reads: 
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$
 The sign is illuminated, and the background is dark, making the neon text stand out. The sign is slightly tilted and has some visible wiring and mounting hardware.

Bayes' Theorem is the basis behind a branch of machine learning that most notably includes the Naive Bayes classifier. That's something important to consider when you're faced with machine learning interview questions.

**Q9) How is a decision tree pruned?**

**Ans.** Pruning is what happens in decision trees when branches that have weak predictive power are removed in order to reduce the complexity of the model and increase the predictive accuracy of a decision tree model. Pruning can happen bottom-up and top-down, with approaches such as reduced error pruning and cost complexity pruning.

Reduced error pruning is perhaps the simplest version: replace each node. If it doesn't decrease predictive accuracy, keep it pruned. While simple, this heuristic actually comes pretty close to an approach that would optimize for maximum accuracy



**Q10) What cross-validation technique would you use on a time series dataset?**

**Ans.** Instead of using standard k-folds cross-validation, you have to pay attention to the fact that a time series is not randomly distributed data — it is inherently ordered by chronological order. If a pattern emerges in later time periods for example, your model may still pick up on it even if that effect doesn't hold in earlier years!

You'll want to do something like forward chaining where you'll be able to model on past data then look at forward-facing data.

- fold 1 : training [1], test [2]
- fold 2 : training [1 2], test [3]
- fold 3 : training [1 2 3], test [4]
- fold 4 : training [1 2 3 4], test [5]
- fold 5 : training [1 2 3 4 5], test [6]

**Q11) When should you use classification over regression?**

**Ans.** Classification produces discrete values and dataset to strict categories, while regression gives you continuous results that allow you to better distinguish differences between individual points. You would use classification over regression if you wanted your results to reflect the belongingness of data points in your dataset to certain explicit categories (ex: If you wanted to know whether a name was male or female rather than just how correlated they were with male and female names.)

## Module 2: Supervised Machine Learning Techniques

Classification	Regression
<ul style="list-style-type: none"><li>• Classification is the task of predicting a discrete class label</li><li>• In a classification problem data is labelled into one of two or more classes</li><li>• A classification problem with two classes is called binary, more than two classes is called a multi-class classification</li><li>• Classifying an email as spam or non-spam is an example of a classification problem</li></ul>	<ul style="list-style-type: none"><li>• Regression is the task of predicting a continuous quantity</li><li>• A regression problem requires the prediction of a quantity</li><li>• A regression problem with multiple input variables is called a multivariate regression problem</li><li>• Predicting the price of a stock over a period of time is a regression problem</li></ul>

### **Q12) What is the difference between Entropy and Information Gain?**

**Ans.**

- Entropy is an indicator of how messy your data is. It decreases as you reach closer to the leaf node.
- The Information Gain is based on the decrease in entropy after a dataset is split on an attribute. It keeps on increasing as you reach closer to the leaf node.

### **Q13) What are features?**

**Ans.** Most simply, features are how we represent the data. There are many different ways to do this; for example, if we were comparing plays, and trying to predict when they were written, we could make the entire text a single feature. Much more useful, however, would be to use a bag of words model. More generally, features are the parameters on which predictive variable is dependent on.; this helps to remove noise in the data, which we don't necessarily want our classifier to learn, and it also makes classification problems tractable.

### **Q14) What is bagging and boosting in Machine Learning?**

**Ans.**

Similarities	Difference
<ul style="list-style-type: none"><li>Both are ensemble methods to get N learners from 1 learner</li></ul>	<ul style="list-style-type: none"><li>While they are built independently for Bagging, Boosting tries to add new models that do well where previous models fall.</li></ul>
<ul style="list-style-type: none"><li>Both generate several training data sets by random sampling</li></ul>	<ul style="list-style-type: none"><li>Only Boosting determines weight for the data to tip the scales in favour of the most difficult cases</li></ul>
<ul style="list-style-type: none"><li>Both make the final decision by taking the average of N learners</li></ul>	<ul style="list-style-type: none"><li>Is an equally average for Bagging and a weighted average for Boosting more weight in those with better performance on training data</li></ul>
<ul style="list-style-type: none"><li>Both are good at reducing variance and proving higher scalability</li></ul>	<ul style="list-style-type: none"><li>Only Boosting tries to reduce bias. On the other hand, Bagging may solve the problem of over-fitting, while boosting can increase it</li></ul>

### Q15) What are labels?

**Ans.** Labels are what we are ultimately interested in. If we are trying to predict which sports team is going to win a game, our labels might be “home team wins” and “home team loses”. If we are trying to predict the weather, our labels could be “sunny”, “rainy”, and “cloudy”. For most machine learning models, we require that the training data is labelled; this is called supervised learning. It is possible to do machine learning with unlabelled training data (this is called unsupervised learning) but it is much harder.

### Q16) What is training?

**Ans.** When you train the classifier, you are using an algorithm to make the models learn which combinations of features are correlated with which labels. The algorithm varies according to the classifier; for example, logistic regression uses gradient descent. As a result, different algorithms may make different predictions. Often, a comprehensive understanding of the inner workings of classifiers along with expert domain knowledge about how the objects you are classifying tend to behave is required to choose an appropriate classification method. This is less true for your final projects (though you are free to delve as deeply into the literature as you like),

and more true if you were trying to decide if a particular set of symptoms and test results (features) imply that a patient has cancer.

### **Q17) What is testing?**

**Ans.** Typically, you will get higher training accuracy scores than testing accuracy scores. This is due to the fact that the classifier inevitably learns some trends which are specific to just the training data and not to general data for that problem; this is known as overfitting. Looking at the testing score can show the discrepancy. Alternatively, using cross validation can show the same thing. In fact, depending on your data set, cross validation is what you should do to test your data. Cross Validation is a method to prevent overfitting. It splits your training data into equal chunks - if you do 10-fold cross validation, it will run 10 times, using a different set of 9 chunks as the training and a different last chunk as test each time. Note that cross validation is not a silver bullet - there may be some cases where the test and training sets are fundamentally different in some way, so high cross validation scores may not be predictive of how well your classifier will perform. For example, if you are performing sports related analysis and your training data is taken from the 2000-2013 baseball seasons, it makes more sense to use the 2014 season as the test data than to do cross validation

### **Q18) Why don't we train on the test set?**

**Ans.** Your classifier learns everything it knows about the world from the training data. When it is training, it tries to make decisions that make its accuracy as good as possible on the training set. This means that the classifier will perform extremely well on the training data; it also means that the classifier has seen the training data

before, so it isn't a real test of its capabilities. It's basically the same as showing a student the answers to the exam, and then trying to infer their mastery of the course from how well they do. One of the big mistakes a classifier can make is to overfit on the training set. We can try to negate this effect using cross validation, but the performance on the test set is a very clear way to see how much the classifier overfit. Let's say we have a classifier that memorizes the input and always maps that exact input to the correct label. This sort of classifier will have 100% accuracy on the training set, but won't have learned anything else. If the test set is included in training, this classifier will have memorized the correct answers, and will have 100% accuracy on the test set as well. When using this classifier into the wild, it may not perform any better than chance.

---