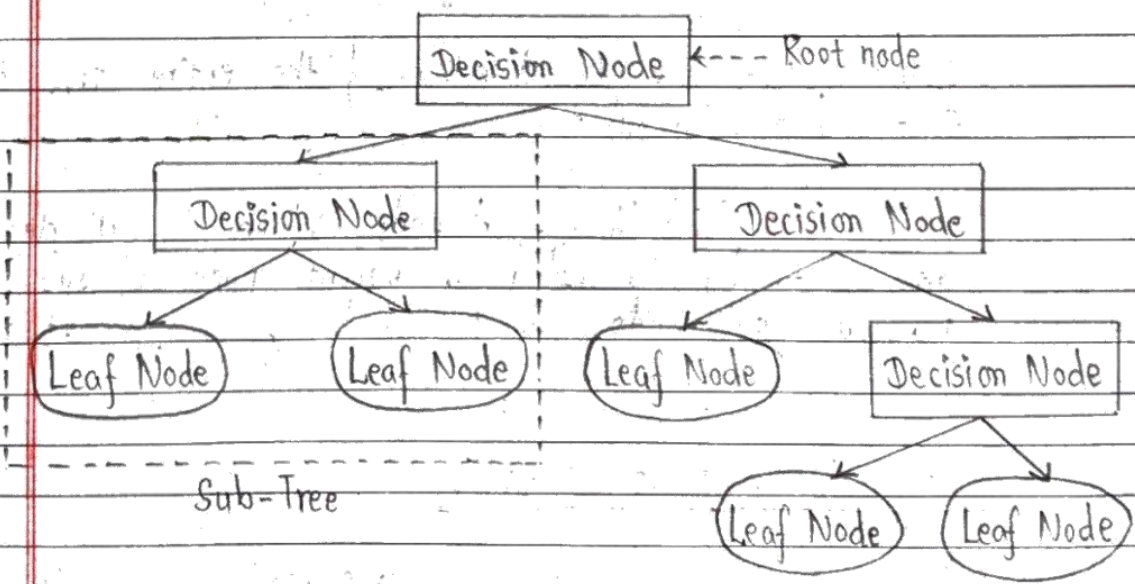
6. Non-Linear Supervised Algorithm in ML

1. Decision Tree (Classification)

- Decision Tree is a supervised learning technique that can be used for both classification and regression problems, but mostly it is preferred for solving classification problems.

- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.



Terminologies

Root Node : It represents the entire population or sample and this further gets divided into two or more homogenous sets.

Splitting: It is a process of dividing a node into two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.

Leaf / Terminal Node: Nodes do not split is called Leaf or Terminal node.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. It can be said the opposite process of splitting.

Branch / Sub-Tree: A subsection of the entire tree is called branch or sub tree.

Parent and Child Node: A node which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

Attribute Selection Measures

From attribute selection measures, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information gain
2. Entropy / Gini Index

1. Information Gain:

Information gain is a measurement of changes in

entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provide us about a class.
Mathematically,
Information gain = Entropy (S) - [(Weighted Avg) * Entropy (each feature)]

2. Entropy:
Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data.
Mathematically,

$$\text{Entropy}(S) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

where,
S = total number of samples
P(yes) = probability of yes
P(no) = probability of no

Gini Index:
Gini Index is measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.
An attribute with low Gini index should be preferred as compared to high Gini index.
Mathematically,

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} P_i^2$$

where,
n = total number of classes in dataset
P_i = probability of an object being classified into class i.

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values.
dataset.isnull().sum()  # No null values.

# Seperate dependent and independent variables.
x = dataset.iloc[:,:-1]
y = dataset["Purchase"]

# Scaling our data.
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
stdsc.fit(x)
x = pd.DataFrame(stdsc.transform(x), columns = x.columns)
x.head(3)

# Train test split.
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.2, random_state = 42)

# Build decision tree model.
```

```
from sklearn.tree import DecisionTreeClassifier # Classification
#model.

dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

dtc.score(x_test, y_test)

dtc.predict([[19, 19000]])

# Visualization of decision region.
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=dtc)
plt.show()

#Visualization of decision tree
plt.figure(figsize=(50, 50))
plot_tree(dtc)
plt.savefig("DecisionTree.jpg")
plt.show()
```

Handling Overfitting

"" Over fitting can be handled either by pre-pruning or post-pruning ""

```
# Check if the model is overfitted
dtc.score(x_train, y_train), dtc.score(x_test, y_test)
# Huge difference in score between training and testing data
# means it is overfitted.
```

Pre-Pruning

"In pre-pruning the max_depth is set while creating an object of DecisionTreeClassifier class."

```
dtc = DecisionTreeClassifier(max_depth = 3)
dtc.fit(x_train, y_train)

dtc.score(x_train, y_train), dtc.score(x_test, y_test)
# Less difference is seen now.

# Visualization of decision region.
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf = dtc)
plt.show()

# Visualization of decision tree
plt.figure(figsize = (50, 50))
plot_tree(dtc)
plt.show()
```

Post-Pruning

"It first finds the best max_depth by training the model to different depths and then train the model to selected max depth."

```
# Find the best max_depth
for i in range(1, 20):
    dtc = DecisionTreeClassifier(max_depth = i)
    dtc.fit(x_train, y_train)
    print(dtc.score(x_train, y_train), dtc.score(x_test,
          y_test), i)
```

```
# Now from above output decide the value of max_
# depth.
dtc = DecisionTreeClassifier(max_depth = 4)
dtc.fit(x_train, y_train)

dtc.score(x_train, y_train), dtc.score(x_test, y_test)
# Less difference is seen now.

# Visualization of decision region.
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=dtc)
plt.show()

# Visualization of decision tree.
plot.figure(figsize=(50, 50))
plot_tree(dtc)
plt.show()
```

2. Decision Tree (Regression)

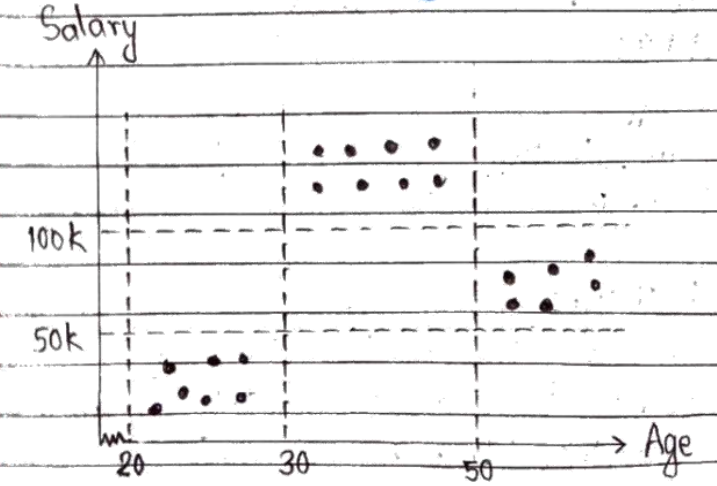


Fig: Graph

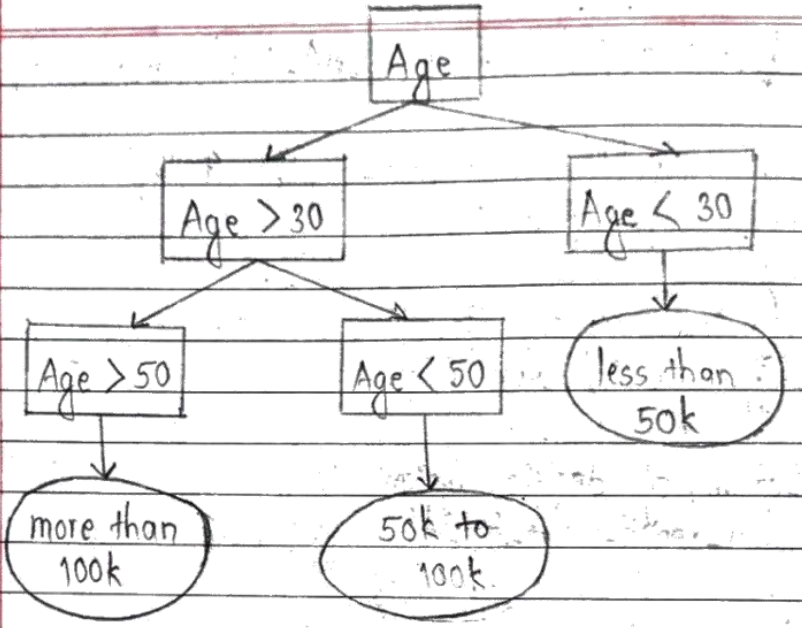


Fig: Decision Tree

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum

# Fill the null values
for column in dataset.columns:
    dataset[column].fillna(dataset[column].mean(),
                           inplace=True)
```

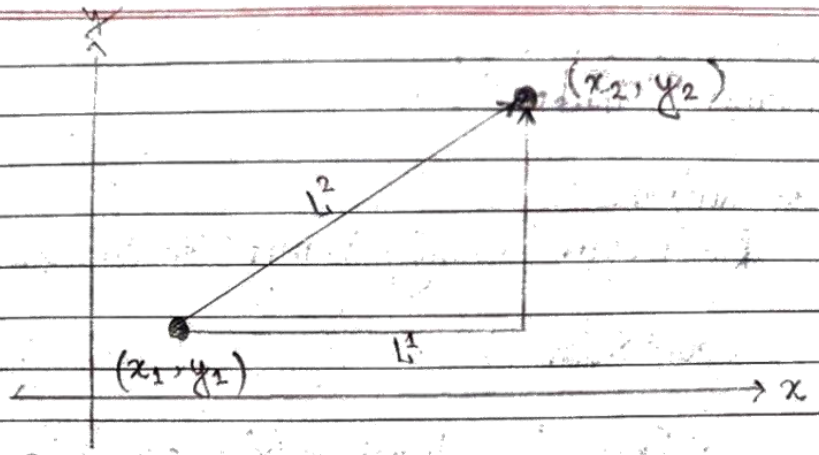
```
# Visualize data.
sns.pairplot(data = dataset)
plt.show()

# Seperate dependent and independent variables.
x = dataset.iloc[:, :-1]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.2, random_state = 42)

# Build a model
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()
dtr.fit(x_train, y_train)

dtr.score(x_test, y_test)

# Check if the model is overfitted.
dtr.score(x_train, y_train), dtr.score(x_test, y_test)

# We find huge difference between score of train and
# test data so the model is overfitted.
# Find the best value of max_depth.
for i in range(1, 20):
    dtr = DecisionTreeRegressor(max_depth = i)
    dtr.fit(x_train, y_train)
```

```
print (dtr.score (x_train , y_train), dtr.score (
        x_test , y_test ), i)
```

```
# Select the best value of max_depth and train the
# model.
dtr = DecisionTreeRegressor (max_depth = 4)
dtr. fit (x_train , y_train)

# For visualization of decision tree
from sklearn.tree import plot_tree

plt.figure (figsize =(50, 50))
plot_tree (dtr)
plt.show ()
```

3. K-Nearest Neighbours (Classification)

- K-NN algorithm can be used for regression as well as for classification but mostly it is used for classification problems.
- K-NN algorithm is a non-parametric algorithm, which means it doesn't make any assumption on underlying data.
- It is also called a lazy learner algorithm.



Manhattan distance, $L^1 = |x_2 - x_1| + |y_2 - y_1|$
Euclidean distance, $L^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

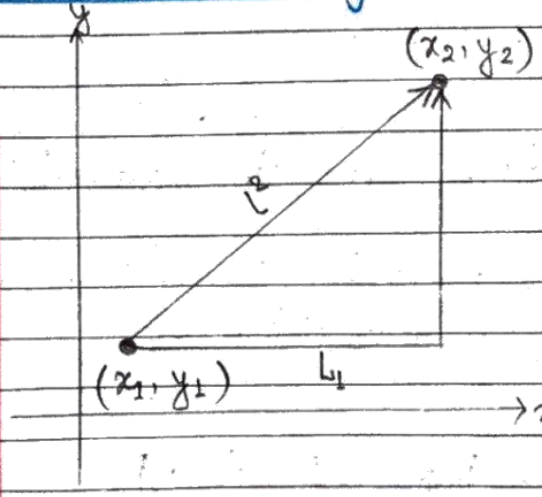
Working practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

# Visualize the data
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.show()

#Seperate dependent and Independent variables.
```

```
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Data scaling
from sklearn.preprocessing import StandardScalar

ssc = StandardScalar()
ssc.fit(x)
x = pd.DataFrame(ssc.transform(x), columns = x.columns)

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
    y, test_size = 0.2, random_state = 42)

# Train model from dataset
from sklearn.neighbours import KNeighboursClassifier

knn = KNeighboursClassifier()
knn.fit(x_train, y_train)

knn.score(x_test, y_test)

# Check if the model is overfitted
knn.score(x_train, y_train), knn.score(x_test, y_test)

# Find the best value of n_neighbours
for i in range(1, 30):
    knn = KNeighboursClassifier(n_neighbours = i)
    knn.fit(x_train, y_train)
```

```
print (i, knn.score (x_train, y_train), knn.score
                    (x_test, y_test))

# Train the model by using best value of n_neighbours.
knn = KNeighboursClassifier (n_neighbours = 4)
knn.fit (x_train, y_train)

# Make the new predictions
knn.predict (ssc.fit_transform ([[32, 32000]])) # Use
# scaled data.

# Visualize the decision region
from mlxtend.plotting import plot_decision_regions

plot_decision_regions (x.to_numpy(), y.to_numpy(),
                       clf = knn)
plt.show ()
```

4. K-Nearest Neighbours (Regression)



$$L^1 = |x_2 - x_1| + |y_2 - y_1|$$

$$L^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

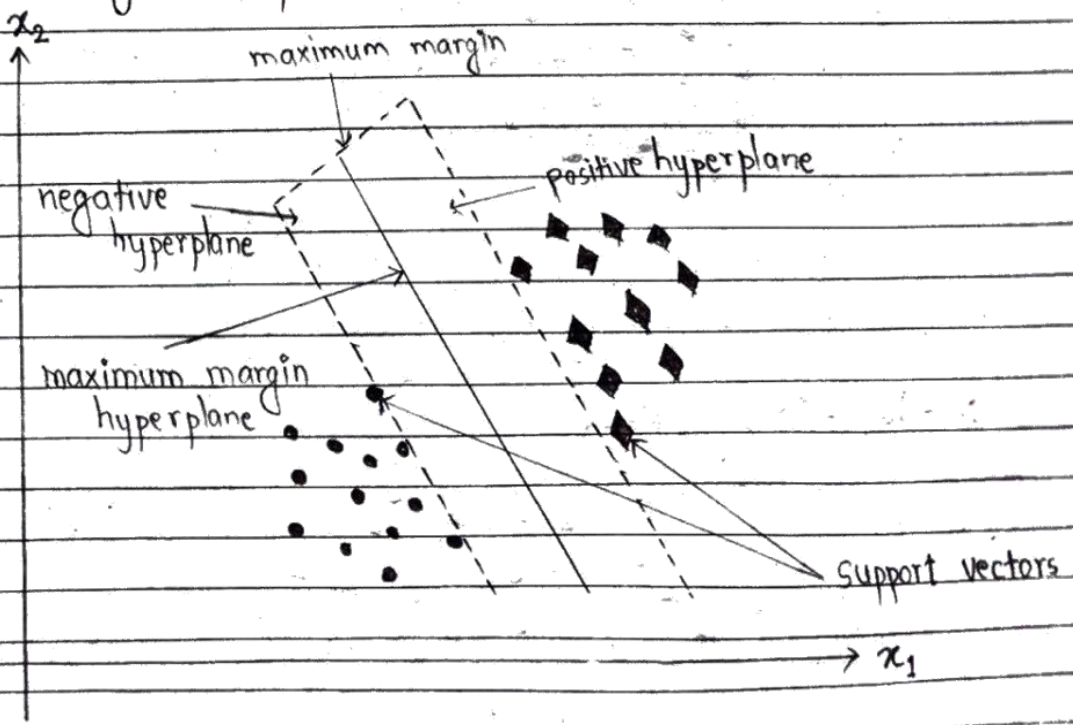
Working practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Drop all the null values
dataset.dropna(inplace=True)

# Visualize data
sns.pairplot(data=dataset)
plt.show()

# Seperate dependent and independent data
x = dataset.iloc[:, :-1]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size=0.2, random_state=42)

# Train your model
from sklearn.neighbours import KNeighboursRegressor
```

```
knn = KNeighbourRegressor()
knn.fit (x_train , y_train)

knn.score (x_test , y_test)

# Check if the model is overfitted.
knn.score (x_train , y_train) , knn.score (x_test , y_test)
```

5. Support Vector Machines (Classification)

↳ SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.

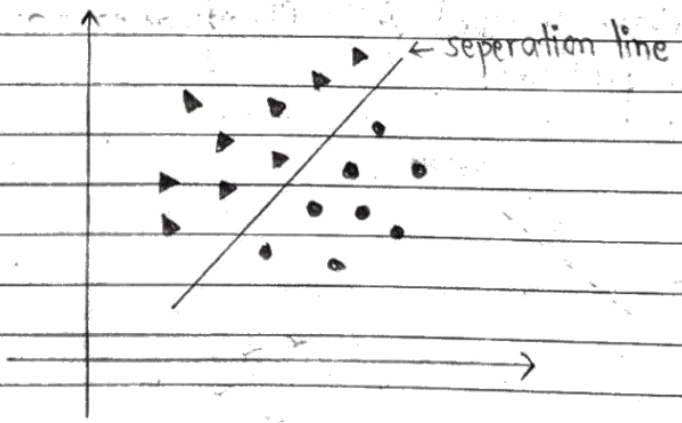


↳ **Hard Margin :** The algorithm aims to find a hyperplane that perfectly seperates the data into two classes without any misclassifications.

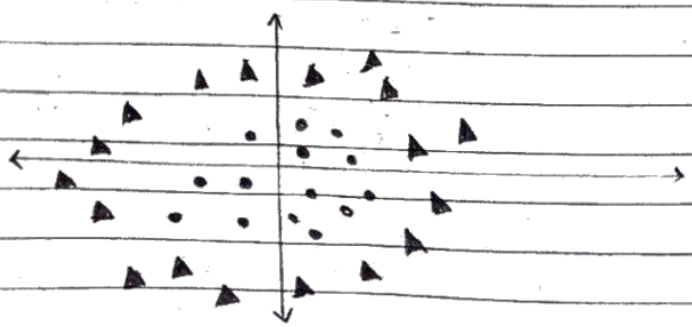
↳ **Soft Margin:** The algorithm allows for some mis-classifications to find a hyperplane that generalizes better to unseen data and is more robust to outliers.

Types of SVM.

1. Linear SVM:

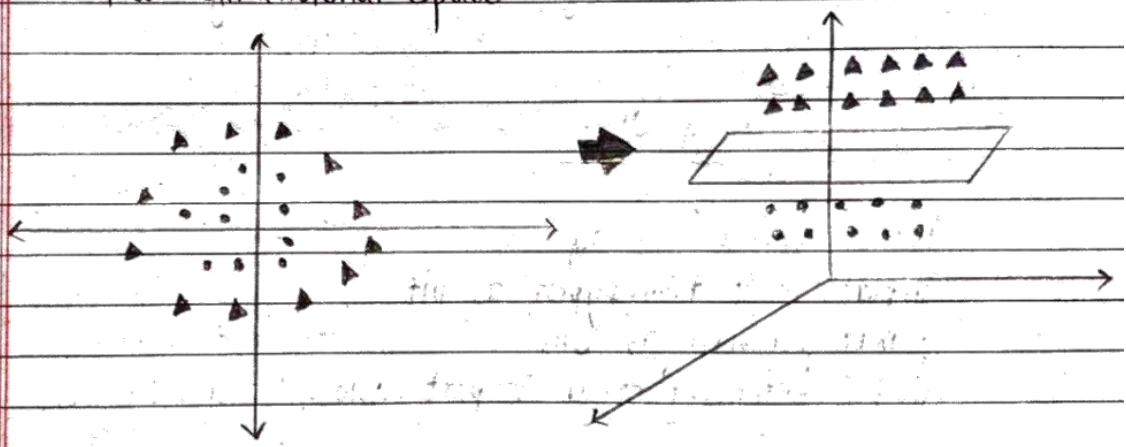


2. Non-linear SVM.:



3. Simple SVM:
Typically used for linear regression and classification problems.

4. Kernel SVM:
Has more flexibility for non-linear data because you can add more features to fit a hyperplane instead of a two dimensional space.



Kernel Functions

- Kernal functions play a crucial role in transforming input data into a higher-dimensional space.
- The primary purpose of kernel functions is to allow SVMs to handle non-linearly seperable data by implicitly mapping the input data into a higher dimensional feature space where linear seperation may be more feasible.
- This transformation is done without explicitly calculating

the coordinates of the data points in that higher dimensional space.

↳ Some kernel functions are:-
Linear : $K(w, b) = w^T x + b$
Polynomial : $K(w, x) = (\gamma w^T x + b)^N$
Gaussian RBF : $K(x, x') = \exp(-|x - x'|^2) / 2\sigma^2$
Sigmoid : $K(x, x') = \tanh(\alpha x \cdot x' + \beta)$

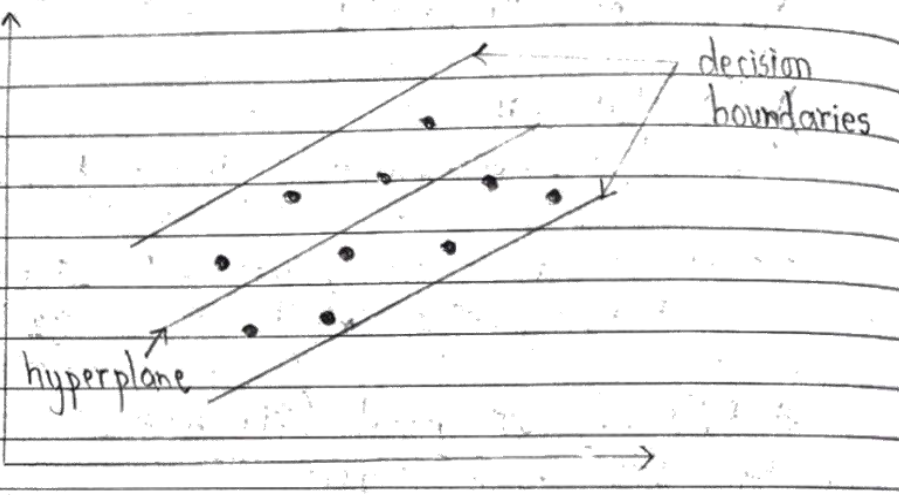
Working Practically:-

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

# Visualize the data
plt.figure(figsize=(4,3))
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchased")
plt.show()

# Seperate Input and output
x = dataset.iloc[:, :-1]
```

```
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 82)

# Build the SVM model (Classification)
from sklearn.svm import SVC

svc = SVC(kernel = "linear")
svc.fit(x_train, y_train)

svc.score(x_test, y_test)

# Check if the data is overfitted.
svc.score(x_train, y_train), svc.score(x_test, y_test)

# Visualize decision region
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf = svc)
plt.show()
```

6. Support Vector Machines (Regression)

↳ Support Vector Regression (SVR) is a regression technique that uses Support Vector Machines (SVM) for modeling and predicting continous outcomes.



Working Practically

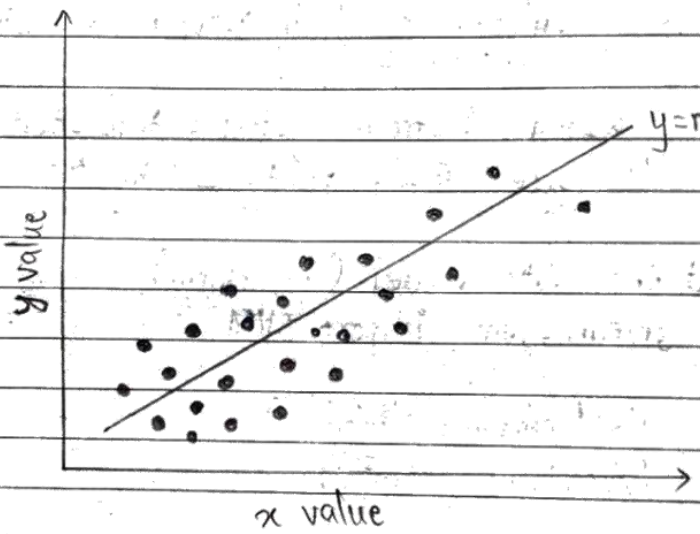
```
import pandas as pd.
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Drop all the null values
dataset.dropna(inplace = True)

# Visualize data
plt.figure(figsize = (5,5))
sns.scatterplot(x: "Number 3", y = "Total Sum", data =
                dataset)
plt.show()
```

```
# Seperate required dependent and independent data.
x = dataset[["Number3"]]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 42)

# Build the SVM model (Regression)
from sklearn.svm import SVR

svr = SVR(kernel = "linear")
svr.fit(x_train, y_train)

svr.score(x_test, y_test)

# Check if the model is overfitted
svr.score(x_train, y_train), svr.score(x_test, y_test)

# Visualize the prediction line.
sns.scatterplot(x = "Number3", y = "Total Sum", data =
                dataset)
plt.plot(dataset["Number3"], svr.predict(x), color = "red")
plt.show()
```

7. Hyperparameter Tuning

Model Parameters

- Model parameters are configuration variables that are internal to the model, and a model learns them on its own.



- From diagram, m and c are model parameters.

Model Hyperparameters

- Hyperparameters are these parameters that are explicitly defined by the user to control the learning process.
- The best value can be determined either by the rule of thumb or by trial and error.
- Example: DecisionTreeClassifier (max-depth = 3)

Hyperparameter Tuning

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem.

The two best strategies for hyperparameter tuning are:

1. GridSearchCV
2. RandomizedSearchCV

1. **GridSearchCV :** GridSearchCV is a technique to search through the best parameter values from the given set of the grid of parameters.
It is slow when you have a large number of hyperparameters or huge dataset.

2. **RandomizedSearchCV :** It goes through only a fixed number of hyperparameter settings.
It moves within the grid in random fashion to find the best set of hyperparameters.
It can miss the best hyperparameters.

Working Practically:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
```

```
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Drop all null values
dataset.dropna(inplace = True)

# Visualize data
sns.scatterplot(x= dataset["Number 2", y= dataset["
                Number 3"], data = dataset)
plt.show()

# Select the required dependent and independent data
x = dataset[["Number 2"]]
y = dataset ["Number 3"]

# Train test split.
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size = 0.2, random_state = 42)

# Build model
from sklearn.tree Import DecisionTreeRegressor

dtr = DecisionTreeRegressor()
dtr.fit(x_train, y_train)

# Check if the model is overfitted
dtr.score(x_train, y_train), dtr.score(x_test, y_test)
```

```
# Yes it is overfitted so we need to do hyperparameter
# tuning to make it more accurate.

# Create a dictionary with the hyperparameters you
# want to tune
hyperparameters_to_tune = {
    "criterion": ["squared_error", "friedman_mse",
                  "absolute_error", "poisson"],
    "splitter": ["best", "random"],
    "max_depth": [i for i in range(2, 20)],
}
```

Using GridSearchCV:

```
from sklearn.model_selection import GridSearchCV

gdcv = GridSearchCV(DecisionTreeRegressor(), param_grid =
                    hyperparameters_to_tune)
gdcv.fit(x_train, y_train)

# Check best score
gdcv.best_score_

# Check the best hyperparameters value
gdcv.best_params_

# Train the model using best values of hyperparameters
dtr = DecisionTreeRegressor(criterion = "friedman_mse",
                            max_depth = 5, splitter = "random")
dtr.fit(x_train, y_train)
```

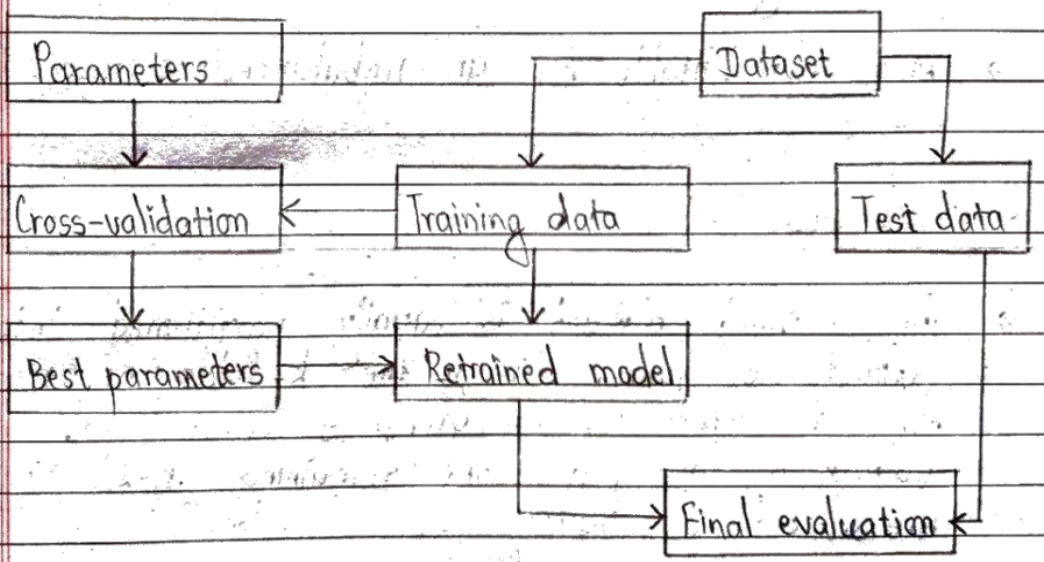
```
# Check if the model is overfitted.
dtr.score (x_train, y_train) , dtr.score (x_test , y_test)
# No this is not overfitted.
```

Using RandomizedSearchCV:

```
from sklearn.model_selection import RandomizedSearchCV

rdcv = RandomizedSearchCV ( DecisionTreeRegressor (),
                param_distributions = hyperparameters_to_tune,
                n_iter = 20 )
rdcv.fit (x_train, y_train)

# Check best score.
rdcv.best_score_

# Check the best hyperparameters value
rdcv.best_params_

# Train the model using best values of hyperparameters.
dtr = DecisionTreeRegressor ( criteria = "friedman_mse",
        max_depth = 5, splitter = "random" )
dtr.fit (x_train, y_train)

# Check if the model is overfitted
dtr.score (x_train, y_train) , dtr.score (x_test , y_test)
# No it is not overfitted.
```

8. Cross Validation

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of input data.



Methods Used for Cross-Validation

- Leave p out cross-validation
- Leave one out cross-validation
- Holdout cross-validation
- Repeated random subsampling validation
- k-fold cross-validation
- Stratified k-fold cross validation
- Time series cross-validation
- Nested cross-validation.

• K-Fold Cross-Validation.

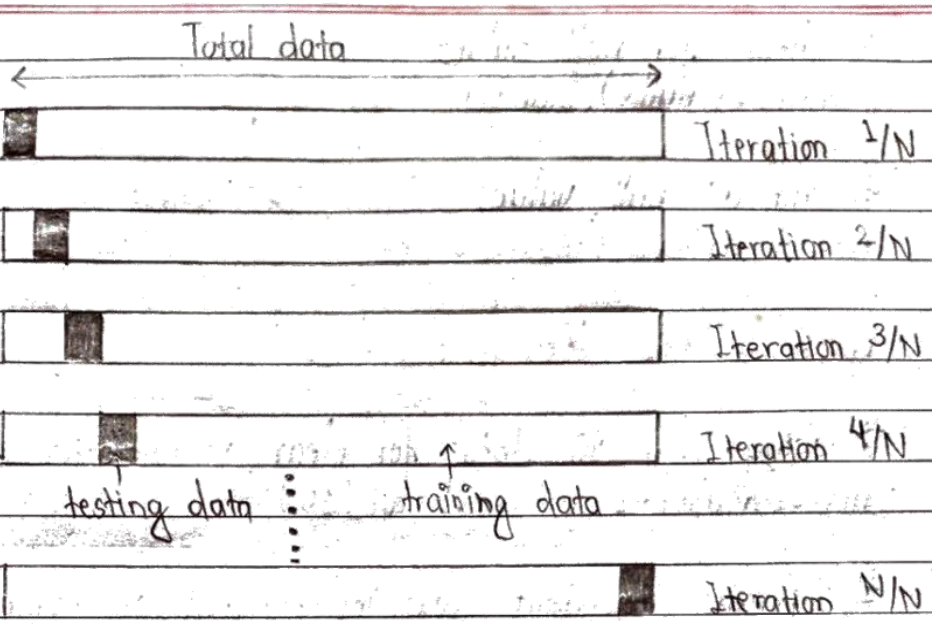
- The original dataset is equally partitioned into k subparts or folds. Out of the k-folds or groups, for each iteration, one group is selected as validation data, and the remaining (k-1) groups are selected as training data.
- It is not suitable for an imbalanced dataset.

• Stratified K-fold Cross-Validation

- The original dataset is equally partitioned into k subparts or folds. Out of the k-folds or groups, for each iteration, one group is selected as validation data, and the remaining (k-1) groups are selected as training data.
- Stratified k-fold cross-validation solve the problem of an imbalanced dataset.

• Leave-One-Out Cross-Validation

- Leave-one-out cross-validation (LOOCV) is an exhaustive cross-validation technique. It is a category of Lp OCV with the case of $p=1$.
- If the dataset is huge, it takes a lot time to train the model.



Leave P-Out Cross Validation

Leave p-out cross validation (LpOCV) is an exhaustive cross validation technique, that involves using p-observation as validation data and remaining data is used to train the model. This is repeated in all ways to cut the original sample on a validation set of p observations and a training set.

Working Practically

```
import pandas as pd

dataset = pd.read_csv("Maths.csv")
dataset.head(3)
```

```
# Check for null values
dataset.isnull().sum()

# Drop all null values
dataset.dropna(inplace=True)
```

Check Validation Methods :

```
# Taking only 10 datas for clear visualization
new_dataset = dataset.head(10)

# Seperate dependent and independent variables
new_x = new_dataset.iloc[:, :-1]
new_y = new_dataset["Total Sum"]

# Check the validation methods in new_dataset
from sklearn.model_selection import LeaveOneOut,
                LeavePOut, KFold, StratifiedKFold

# LeaveOneOut
loo = LeaveOneOut()
for train, test in loo.split(new_x, new_y):
    print(train, test)

# LeavePOut
lpo = LeavePOut(p=2)
for train, test in lpo.split(new_x, new_y):
    print(train, test)

# KFold
kf = KFold(n_splits=5)
```

```
for train, test in kf.split(new_x, new_y):
    print(train, test)

# Stratified KFold
# skf = StratifiedKFold(n_splits=5)
# for train, test in skf.split(new_x, new_y):
#     print(train, test)
''' StratifiedKFold works only on classification analysis '''
```

Check Range of Accuracy :

```
# Seperate dependent and independent variables
x = dataset[["Number 1"]]
y = dataset["Total Sum"]

# Check how much accuracy can our dataset give.
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

cross_val_score(LinearRegression(), x, y, cv=5)

# Sort the cross_val_score
cvs = cross_val_score(LinearRegression(), x, y, cv=5)
cvs.sort()
cvs * 100  # Display in percentage.
```