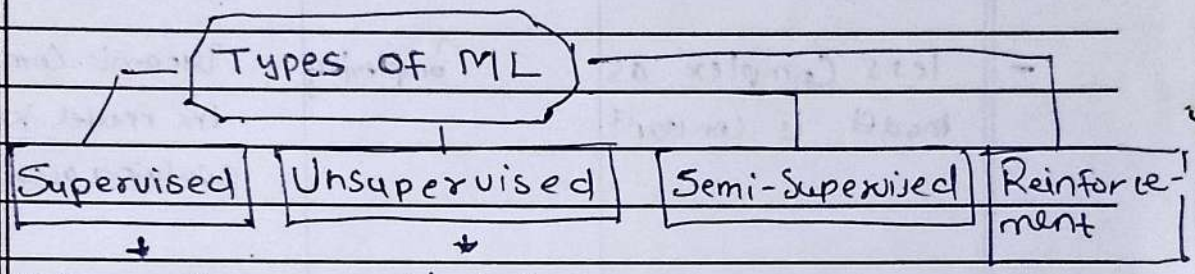
MACHINE LEARNING

Definition

Machine Learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" with data, without being explicitly programmed.



- Regression
- Classification
- Clustering
- Dimensionality Reduction
- Anamoly detection
- Assosication

Batch ML Vs Online ML

- **Batch ML:**

Also known as offline learning, involves training a machine learning model using entire dataset at once.

In Batch model is trained on a fixed dataset & Parameters are updated based on gradients computed from entire dataset.

- **Online ML:**

Also known as incremental learning or real-time learning, involves training a ML model continuously as new data become available over time.

In online learning, model is updated incrementally with each new data point or small batch of data.

Online learning libraries
River
vopal Rabbit

	Offline Learning	Features	Online Learning
-	less Complex as model is constant	Complexity	Dynamic Comp. as the model keep evolving over time
-	Fewer	Computational Powers	Continuous
-	Easier to implement	Use in Production	Difficult to implement.
	Image classification, & anthing in ML.	Applications	used in finance, health new data patterns.
	Industry proven Tools, e.g. Skikit, TensorFlow, Pytorch, Keras, Spark Mlib.	Tools	Active research, New pr tools. e.g. MOA, SAMOA, sckikit-multiflow

ML Types on the basis of Learning.

2 Types

Instance Based | Model Based

1. Instance Based Learning

- It is also known as Memory-based learning or lazy-learning
- here, the systems that learn the training examples by heart and then generalizes to new instances based on some similarity measures. It builds the hypotheses from the training instances.

Instance-based learning algorithms

1. K Nearest Neighbor (KNN)
2. Self-Organizing Map (SOM)
3. Learning Vector Quantization (LVQ)
4. Locally Weighted Learning (LWL)
5. Case-Based Reasoning ...etc.

: etc.

2. Model Based Learning

- It involves creating a model that generalizes from the training data to make predictions on new, unseen data. This model can capture patterns, relationships, and features within the data to provide a way to understand and predict outcomes.

Model-based learning algorithms are: decision trees, linear regression, neural networks, support vector machines etc.

* Challenges in Machine Learning *

1. Data Collection
2. Insufficient Data / Labelled Data
3. Non Representative Data
4. Poor Quality Data
5. Irrelevant Features
6. Overfitting
7. Underfitting
8. Software Integration
9. Offline Learning / Deployment
10. Cost Involved

* Application of Machine Learning (B2B)

Sectors of ML.

1. Retail - Amazon / Big Bazaar
   a. buying Pattern of user
   b. deciding Positioning of Product.
   c. targeted marketing
   d. seasonal Product etc.

2. Banking and Finance
   a. Profile matching with past defaulters of loan.
   b. Plans according to customers. etc.

3. Transport - OLA
   a. Search Pricing according to time
   b. Demand Forecasting
   c.

4. Manufacturing - Tesla
   a. Fault occurence sensors.
   b. Predictive Maintainance

5. Consumer Internet - Twitter
   a. ~~twits~~ Sentiment analysis tweets

★ Machine Learning Development Life Cycle (MLDLC)

1. Frame the Problem
2. Gathering Data
3. Data Preprocessing
4. Exploratory Data Analysis
5. Feature Engineering and Selection
6. Model Training, Evaluation and Selection
7. Model Deployment
8. Testing
9. Optimize

* Various Data Based Job Roles

1. Data Engineer
2. Data Analyst
3. Data Scientist
4. ML Engineer

1. Data Engineer Job Roles
- Scrape Data from the given sources.
- move / store the data in optimal servers / warehouses.
- Build data pipelines / APIs for easy access to the data
- Handle databases / data warehouses.

2. Data Analyst Job Roles
- Cleaning & organizing Raw data.
- Analyzing data to derive insights
- Creating data visualizations.
- Producing & maintaining reports
- Collaborating with teams / colleagues based on the insight gained.
- Optimizing data collection procedures

3. Data Scientist Job Roles
* Description: " A data scientist is someone who is better at statistics than any software engineer and better at software engineering than any statistician". - Josh Wills on Quora

- It is a fullstack developer who can do all things like data engineering, data Analytics & ML engineering

4. ML Engineer Job Role

- Deploying machine learning models to production ready environment
- Scaling and optimizing the model for production
- Monitoring and maintenance of deployed models.

* Tensor in Machine Learning

Tensors are mathematical objects that can be used to describe physical properties, like scalars & vectors, scalar is a zero rank tensor
vector is a first rank tensor

Types of Tensors

- 0D Tensor / vector
  Dimensions are zero
- 1D Tensor / vector
  [1, 2, 3, 4] → 1D Tensor → 1D Tensor
  ↳ vector

2D Tensor / Matrices

Collection of vectors.

$$\begin{bmatrix} [1,2,3] \\ [4,5,6] \\ [7,8,9] \end{bmatrix} \Rightarrow 2D.$$

ND Tensors

3D Tensor = 4 x 3 x 3.
4D tensor = vector of 3D tensor
5D Tensor = metrics of 4D tensor
⋮

* Main Python Tools *

- Anaconda
- Jupyter Notebook
- Virtual Environment
- Kaggle Notebook
- Google Collab

End to End Project

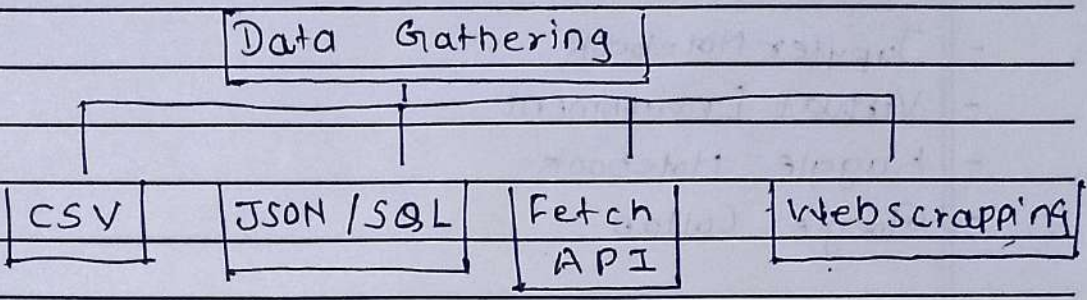
Steps

0. Preprocess + EDA + feature selection
1. Extract input and output cols
3. Train test split
4. Train the model
5. Evaluate the model / model selection
6. Deploy the model

* How to Frame a Machine Learning Problem

1. Business Problem to ML Problem Conversion
2. Type of Problem
3. Current Solutions Availability
4. Getting Data
5. Metrics to measure
6. Online Vs Batch ?
7. Check Assumptions

* Gathering Data



1] CSV - Comma Seperated Values
Some important Commands and parameters

1: Importing pandas
import pandas as pd

2. Opening a local csv file
df = pd.read_csv('file_name.csv')

3. Opening a ~~csv~~ Tsv file.
pd.read_csv('file-name.tsv', sep='\t')

4. giving column names with
pd.read_csv('.csv', name = ['col1', 'col2'])

5. Choosing perticular ~~column~~ index column
pd.read_csv('.csv', index_col = 'col-name')

6. Header parameter
for starting first row in n no. of column.

7. use_cols parameter
import only required parameters.

8. squeeze parameters.
- convert pandas df into series object.

9. Skiprows / nrows Parameter
- skiprows for skipping unwanted rows with give index of number.
- nrows for importing n no. of rows

10. Encoding parameter
- change the encoding of character

11. Skip bad lines : Parser Error

12. dtypes parameter: Change data type of column with this adding as dictionary.

13. Handling Dates
convert as object as "datetime 64"

14. Convertors
Applying transformation to any features.

15. na-values parameter
assigning / specifing any values to na values
e.g. na_values = ['-', '0']

16. Loading a huge dataset in chunks
chunksize = number
pd.read_csv('file.csv', chunksize = 5000)

> Working with JSON / SQL

- JSON - JavaScript Object Notation

For loading JSON data

pd.read_json('data.json')

- SQL - Structured Query Language

For loading SQL data

- run SQL file in any database (e.g. mysql)
- install connector library

!pip install mysql.connector

- import mysql.connector

run query:

Conn= mysql.connector.connect(host='localhost',
user='root',
Password='',
database='world')

it will return connection object

connection

* Now pass connection object on Function.

- pd.read_sql_query("SELECT * FROM city", conn)

(table-name) (object)

* Fetching Data from an API

API - Application Programming interface

These are the data pipelines for sharing data from one application to other

For loading data use requests module.

- import pandas as pd
import requests
response = requests.get('url')
response.json()

Fetching data using Web Scraping

For web scrapping use beautiful soup library.

import pandas as pd
import requests
~~import bs4~~
from bs4 import BeautifulSoup
web= request.get('url').text
now create object of library
soup = BeautifulSoup(web, 'lxml')

★ Understanding Your Data ★ D-19

1. Data size >> df.shape
2. Data head >> df.head()
3. Data sample >> df.sample(5) <Random>
4. Data type >> df.info()
5. Missing values >> df.isnull().sum()
6. Maths summary of data >> df.describe()
7. Duplicate values >> df.duplicated().sum()
   df.drop_duplicates()
8. corr betn cols >> df.corr()
   df.corr()['col-name']

★ EDA - Univariate Analysis ★ D-20

Individual analysis of every columns.

1. Categorical Data.
   a. Count plot.
   >> sns.countplot(df['col-name'])

   b. Pie Chart
   >> df['col-name'].value_counts().plot(kind='pie')

2. Numerical Data.
   a. Histogram >> plt.hist(df['col-name'])
   b. Distplot >> sns.distplot(df['col-name'])

   c. Boxplot >> sns.boxplot(df['col'])

   df['col'].min(), .max(), .mean(),
   df['col'].skew()

EDA - Bivariate and Multivariate D-21

Exploratory Data Analysis
Type (2) Bivariate
Type (3) Multivariate

The Bivariate & Multivariate analysis is easy with seaborn library.

(1) scatter plot (numerical - numerical)
sns.scatterplot (x, y, hue, style, size)
x = numerical-datavalue, y = numerical-datavalue

(2) bar plot (numerical - categorical)
sns.barplot (x = categorical, y = numerical, hue)

(3) box plot (numerical - categorical)
sns.boxplot (x = categorical, y = numerical, hue)

(4) Distplot (numerical - categorical)
sns.distplot (categorical_data, numerical, kde=)

(5) HeatMap (categorical - categorical)
sns.heatmap (pd.crosstab ('categorical', ~~'numerical'~~ 'categorical')
pd.crosstab ('categorical_data', 'categorical data')

(6) ClusterMap (categorical - categorical)
sns.clustermap (pd.crosstab ('categorical'), 'categorical')

⑦ Pair Plot
sns.pairplot (data_frame, hue =)

⑧ Line Plot (numerical - numerical)
x = time based number
y = numerical.

sns.lineplot(x= , y=)

★ Pandas Profiling ★

Pandas Profiling is a library which provides a solution to understanding data with the help of generating reports for datasets that have numerous features.

- Installing Pandas Profiling
  pip install pandas profiling

- Importing.
  from pandas-profiling import ProfileReport

- Make an object & add dataFrame.it. init
  report = ProfileReport(dataFrame)

- Make html file.
  report.to_file(output_file = 'output.html')

The the html file will be download in yours folder.

Feature Engineering

Feature Transformation	Feature Construction	Feature Selection	Feature Extraction

→ Missing Value Imputation
→ Handling Categorical Features
→ Outlier Detection
→ Feature Scaling

Feature Scaling (Standardization)

* When to use Standardization :-

	Algorithms
1.	K-Means
2.	K-NN
3.	PCA
4	ANN
5	Gradient Descent

Feature scaling is a technique to standardize the independent features present in the data in a fixed range

Standardization is also called as Z-score Normalization

$$x_i' = \frac{x_i - \bar{x}}{\sigma}$$

Feature Scaling - (Normalization)

Normalization

. Normalization is a technique often applied as a part of data prepration for ml. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting difference in the range of values or losing information

- MinMax Scaling
- Mean Normalization
- MaxAbs Scaling
- Robust Scaling

MinMax Scaling $x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$

range = [0, 1]

min = 0, max = 1.

Mean Normalization

range = [-1 to 1] $X'i = \frac{x_i - x_{mean}}{x_{max} - x_{min}}$

MaxAbs Scaling

range = [-1, 1] $x_i'' = \frac{x_i}{|x_{max}|}$

used when sparse data

Robust Scaling

range = $x_i' = \frac{x_i - x_{median}}{IQR}$

used when Outliers on data

Feature Transformation

2] Handling Categorical Features

data

- Numerica
- categorical
  - Nominal → one hot encoding
  - ordinal → Ordinal Encoding : for independent variables
    → Label Encoding for dependent (classification) variables.

Ordinal Data

Order matters, categories have weighted magnitude.

Ordinal Encoder => for categorical ordinal independent data (X)

Label Encoder => For categorical ordinal dependent variables (Y) output column is classification.

Nominal Categorical Data

To convert nominal Data we use One Hot Encoding

in pandas we have inbuild method called pd.get_dummies()

In sklearn.preprocessing we have a ~~catt.~~ model / class named OneHotEncoder for conversion

Dummy Variable Trap

The Dummy variable Trap also called as multicollinearity because it generates / causes multicollinearity among the dummy or onehotEncoded variables.

To reduce multicollinearity alway drop (n-1) column from n colums which are formed from encoding

Column Transformer

This is shortcut method for Encoding multiple features.

Import.

```
from sklearn.compose import ColumnTransformer
transformer = ColumnTransformer ( transformers
            = [ (), (), ()... ], remainder='')
```

Sk Learn · Pipelines

Pipelines chains together multiple steps so that the output of each step is used as input to the next step.

Pipelines makes it easy to apply the same preprocessing to train and test!

To use pipeline in ml we have library.

from : sklearn.pipeline import Pipeline, make-pipeline

Pipeline and make_pipeline these 2 classes Do same work we can use anyone among them

Syntax:

Pipeline(steps: list of tuples, ~~mem~~ memory: str or object with .joblib, verbose: bool)

e.g.

```
pipe = Pipeline([
            ('trf1', trf1),
            ('trf2', trf2)
        ], memory : None,
           verbose : False
        )
```

Attributes:

1] named_steps : Bunch
Access the steps by name.

2] classes : ndarray of shape(n-classes)
The classes labels

3] n_features_in_ : int
No. of features seen during first step fit method

4] Feature_names_in_ : ndarray of shape
(n_features_in_,)
Names of features seen during first step fit method

make_pipeline Syntax:

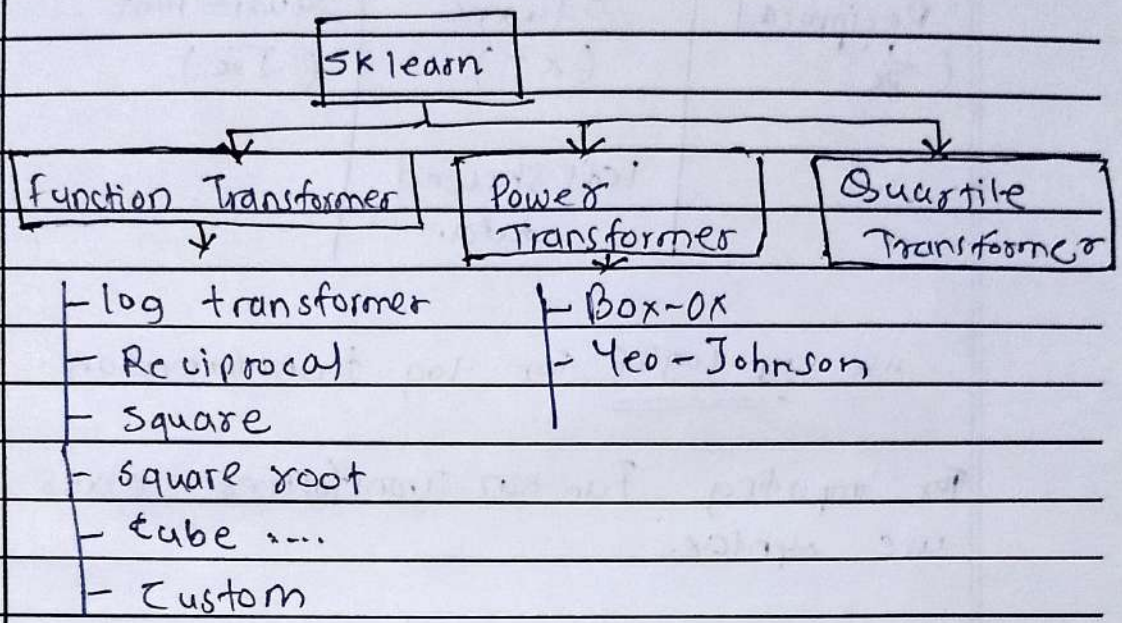
sklearn.pipeline.make_pipeline (*steps : list of estimator objects,
memory: default = None,
verbose: default = false)

Example:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline, Pipeline
pipe = make_pipeline ( StandardScalar(),
                       GaussianNB() )

pipe = Pipeline ( [ ('standardscalar', StandardScaler()),
                    ('gussiannb', GaussianNB())
                  ])
```

* FUNCTION TRANSFORMER *



How To find if data is normal?.

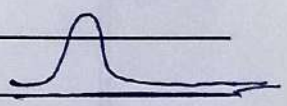
⇒ To find use
sns.distplot
Pd.skew()
. Plot the QQ plot

* Log Transform

Take the log of the data values

when to apply ?

⇒ on right skewed data.



when not to apply ?

⇒ on -ve values.

Why to use log transform.

⇒ Convert the right skewed data into normal distribution. which is helpful for linear & logistic model.

Other transforms

Reciprocal ($\frac{1}{x}$)	Square (x^2)	Square root ($\sqrt{x}$)
	left skewed data.	

use np.log1p for log transformation

For importing Function Transformer class use syntax.

from sklearn.preprocessing import FunctionTransformer

* Power Transformer *

- Box-Cox Transformer
- Yeo-Johnson Transform

Ex: Power Transformer is an another class of sklearn which is used for transformed the data into normal distribution.

Box Cox Transform

The exponent here is

$$x_i^{(\lambda)} = \begin{cases} \dfrac{x_i^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(x_i) & \text{if } \lambda = 0, \end{cases}$$

The exponent here is a variable called lambda(λ) that varies over the range of -5 to 5, and in the process of searching, we examine all values of λ.
Finally, we choose the optimal value (resulting in the best approximation to a normal distribution) for your variable.

Syntax for importing.

```
from sklearn.preprocessing import PowerTransformer
Now make an object of it.
Pt = PowerTransformer(method='box-cox')
```

★ Yeo – Johnson Transform ★

if..

$$x_i^{(\lambda)} = \begin{cases} [(x_i+1)^{\lambda}-1]/\lambda & \lambda \neq 0,\ x_i \geq 0 \\ \ln(x_i)+1 & \lambda = 0,\ x_i \geq 0 \\ -[(-x_i+1)^{2-\lambda}-1]/(2-\lambda) & \lambda \neq 2,\ x_i < 0 \\ -\ln(-x_i+1) & \lambda = 2,\ x_i < 0 \end{cases}$$

This transformation is somewhat of an adjustment to the Box-Cox transformation, by which we can apply it to negative numbers.

Syntax for import.

from sklearn.preprocessing import PowerTransformer

make an object of Yeo-Johnson Transform

pt = PowerTransformer()

(We don't need to specify "method" in Yeo-Johnson it is by default set on that)

Encoding Numerical Features

- Discretization (Binning)
- Binarization

* Discretization

Discretization is the process of transforming continuous variables into discrete variables by creating a set of contiguous intervals that span the range of the variable's values.
Discretization is also called binning, where bin is an alternative name for interval.

Why to use Discretization:
1. To handle Outliers
2. To improve the value spread

★ Types of Discretization ★

Binning

- Unsupervised Binning
  - Equal width (Uniform)
  - Equal frequency (quantile)
  - Kmeans
- Supervised Binning
  - Decision Tree
- Custom Binning

* Unsupervised Binning *

① Equal Width / Uniform Binning

formula = $\frac{Max - Min}{bins}$

· Characteristics:
1. Handle Outliers for some spread
2. No change in spread.
3. Bins are equally distributed.

② Equal Frequency / Quantile Binning

Intervals = 10
Each interval contains 10% of total observations

Characteristics:
1. Handle Outliers
2. Chang in value spread
3. Value is equally spread.

③ kMeans Binning
No of bins is decided by user K.
Use kmeans algorithm for Binning
&
Characteristics:
1. Adapts the distribution of the data
2. Cluster - Based Binning
3.

* ~~Custom Binning~~ *

Encoding discretized variable.

SKLearn
↓
KBinsDiscretizer()

Bins	Strategy	Encoding
	- Uniform	- Ordinal
	- quantile	- Onehot encoding
	- Kmean	

* Custom / Domain Based Binning

With the help of Pandas we can apply Custom Based Binning.
These Custom Binning can be made as per our requirement.

* Binarization

Here we convert the numerical values into Binary value. (0, 1)s

Handling date time data

For doing operations on data & time we required to import necessary modules.

```
import datetime as dt
import time
```

in data if date and time is in object format then convert it into datetime64[ns] format for performing operations. using pandas function "to_datetime()"

e.g.

```
date['column'] = pd.to_datetime(date['column'])
```

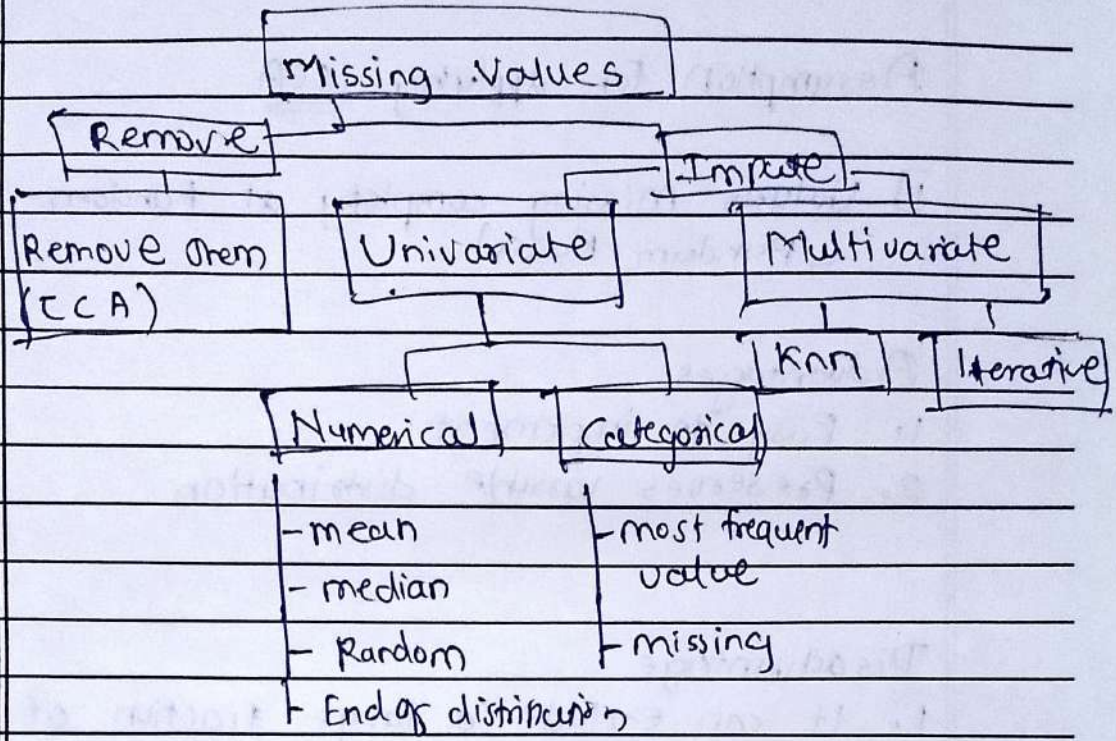
(here, date is a dataframe)

There are different - different datetime functions in a datetime module

```
import datetime as dt.
```

dt.time	dt.hour
dt.year	dt.minute
dt.month	dt.second
dt.day	dt.date
dt.month_name()	
dt.day_name()	
dt.dayofweek	
dt.isocalender().week	
dt.quarter	
dt.today()	

HANDLING Missing DATA



Complete Case Analysis

Complete-case analysis (CCA), also called "list-wise deletion" of cases, consists in discarding observations where values in any of the variables are missing.

Complete Case Analysis means literally analyzing only those observations for which there is information in all of the variables in the dataset.

Observation = Row
Variables = Column

Assumption for applying CCA

1) Values missing complety at Random locations. (Random Rows)

Advantages.
1. Easy to implement
2. Preserves variable distribution

Disadvantage:
1. It can exclude a large fraction of the original dataset.
2. Excluded observations could be informative for the analysis
3. When using our models in production, the model will not know how to handle missing data.

When to Apply CCA.
1. MCAR
2. Missing values = 5% of data

Handling Missing Numerical Data

Numerical

Univariate Imputation

Multivariate Imputation

(1) Mean / median Imputation

Benefit
1] Simple
2] (5% >) data values which are missing

Disadvantage
1] Distribution change.
2] Outliers
3] Covariance / correlation changes

When to use
1] MCAR
2] 5% <

(2) Arbitrary Value Imputation

Replace missing Numeric value with any unusual value {-1, 99, 999 ...etc}

Benefit : Easy to apply

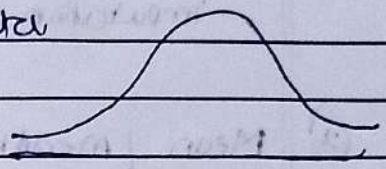
Disadvantage
1] PDF distribution
2) Variance / Covariance / Correlation changes

When to apply : When Data is not missing at Random position.

(4) End of Distribution Imputation.

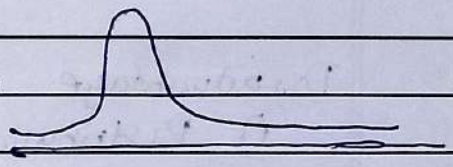
It is an extension of arbitrary value imputation

for Normally Distributed Data



it fill the missing values with (mean + 3σ)
(mean - 3σ)
where σ is a sigma. S.D.

for Skewed Distributed Data



it fill the missing values with (IQR proximity)

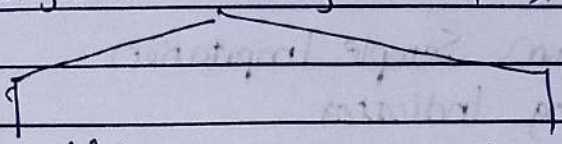
$Q_1 - 1.5 * IQR$ $Q_1 = 25^{th}$
$Q_3 + 1.5 * IQR$ $Q_3 = 75^{th}$

Benefit
1) easy to use.

Disadvantage: 1) PDF
2) Covarance
3) Variance

When to use
1) Data is Not Missing at Random
NMAR

* Handling Categorical Missing Data *



Most frequent Value Imputation

Missing Category Imputation

* Most Frequent Value Imputation

MFVI involves replacing missing values with most frequently occuring value (mode) in column.

To import syntax

```
from sklearn.impute import SimpleImputer
mak an object
imputer = SimpleImputer ( strategy = 'most_frequent')
imputer.fit_transform ( x_train)
```

* Missing Category Imputation

MCI is a technique used primarily for categorical variables. It involves creating a new category (label) specifically for missing values.

To implement with sklearn. (Syntax)

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer (strategy = 'constant',
                         fill_value = 'Missing')
imputer.fit_transform ( x_train)
imputer.transform ( x_test)
```

Random Sample Imputation
Missing Indicator
Automatic selection of parameter using Grid search CV.

1. Random Imputation
⇒ a. Preserves the variance of the variable
b. Memory heavy for deployment, as we need to store the original training set to extract values from and pro replace the NA in coming observations.

c. Well suited for linear models as it does not distort the distribution, regardless of the % of NA.

2- Missing Indicator
Replaces the Na with True & present value with false in new column. So model can improve accuracy.

```
import
from sklearn.impute import MissingIndicator
mi = MissingIndicator()
mi.fit(X_train)
```

Same method is inbuilt in SimpleImputer
si = SimpleImputer(add_indicator = True).
True the 'add_inticator parameter in SimpleImputer

3. Automatically - select - imputer - parameter using GridSearchCV.

GridSearchCV is a sklearn class in model-selection which it used for different value - parameter computation

import
from sklearn.model-selection import ~~train-test-split~~, GridSearchCV

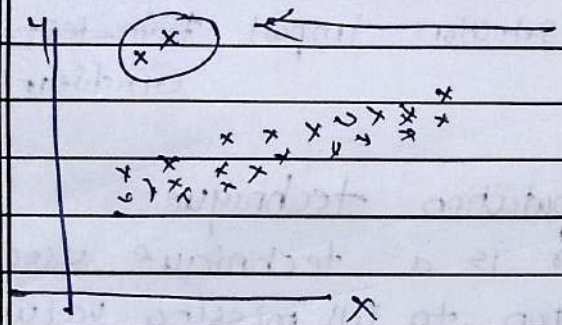
* Multivariate Imputation technique.
- The KNN imputer is a technique used in Multivariate imputation to fill missing values by considering their K-nearest neighbors.

import
class sklearn.impute.KNNImputer(*, missing-values = nan, n-neighbors = 5, weights = 'uniform', metric = 'nan-euclidean', add-indicator = False, keep-empty-features = False)

- Iterative Imputer / MICE
MICE stands for Multivariate Imputation by Chained Equations.
It is method for finding missing values by modeling each variable with missing data as a function of other variables. It is iterative process continues till convergence.

Outliers

Outliers are data points significantly different from the majority in a dataset. In ML outliers can impact model performance by skewing results or introducing noise. Identifying & handling outliers is crucial for building accurate & robust ML models.



Bad Impact of Outliers on ML algorithms.

- Linear
- Logistic
- Adaboost
- Deep Learning

Non-effective ML algorithms.

- Tree based

How to Treat Outliers.

1. Trimming (Removing directly)
2. Capping (setting boundries)
3. Treat accordingly like if it is NaN value.
4. Do Discretization

How To Detect Outliers ?

1. Normal Distribution
   if observation is $(\mu+3\sigma)$ & $(\mu-3\sigma)$ out from these 2 Range's. We treat that values as outliers.

2. Skewed Distribution
   Detect using plotting box plot.

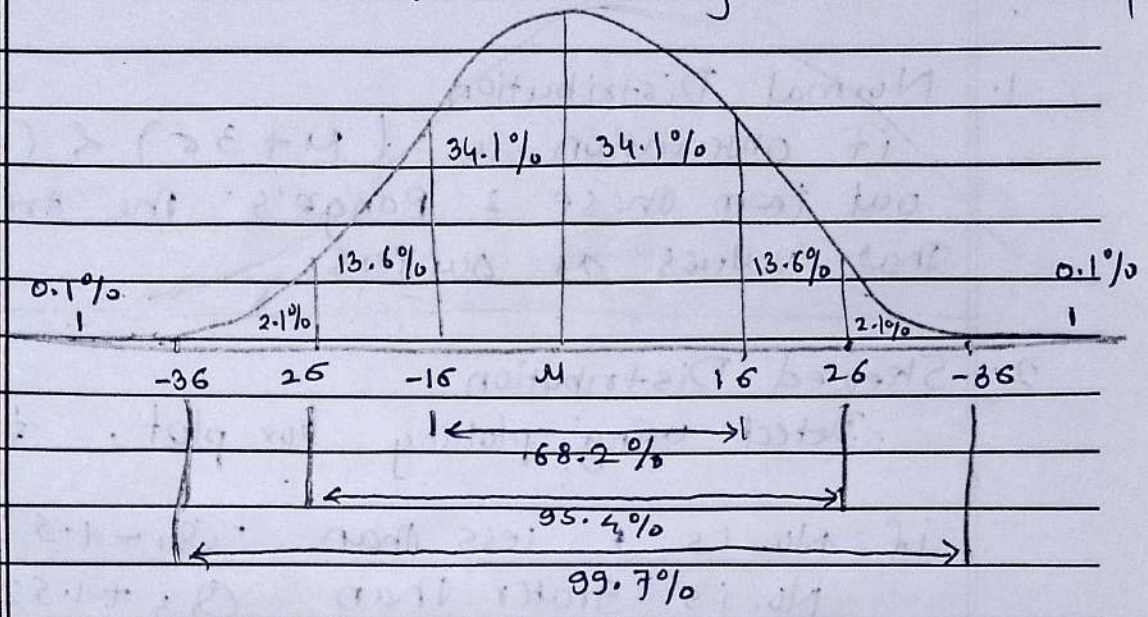
   if No. is less than $Q_1 - 1.5\,IQR$
   No. is grater than $Q_3 + 1.5\,IQR$
   We consider that value as outliers.

3. Percentile Based Approach
   if observation is less than 2.5%
   if observation is greater than 97.5%
   then consider that value as outliers.

* Techniques for Outlier Detection & Removal

1. Z-score
2. IQR based filtering
3. Percentile
4. Winsorization

Outlier Detection & Removal using Z-score method



Normal Curve (Bell curve)

Z score follows Normal curve distribution

Range of value distribution

$\mu + \sigma$, $\mu - \sigma$ } 68%

$\mu + 2\sigma$, $\mu - 2\sigma$ } 95%

$\mu + 3\sigma$, $\mu - 3\sigma$ } 99%

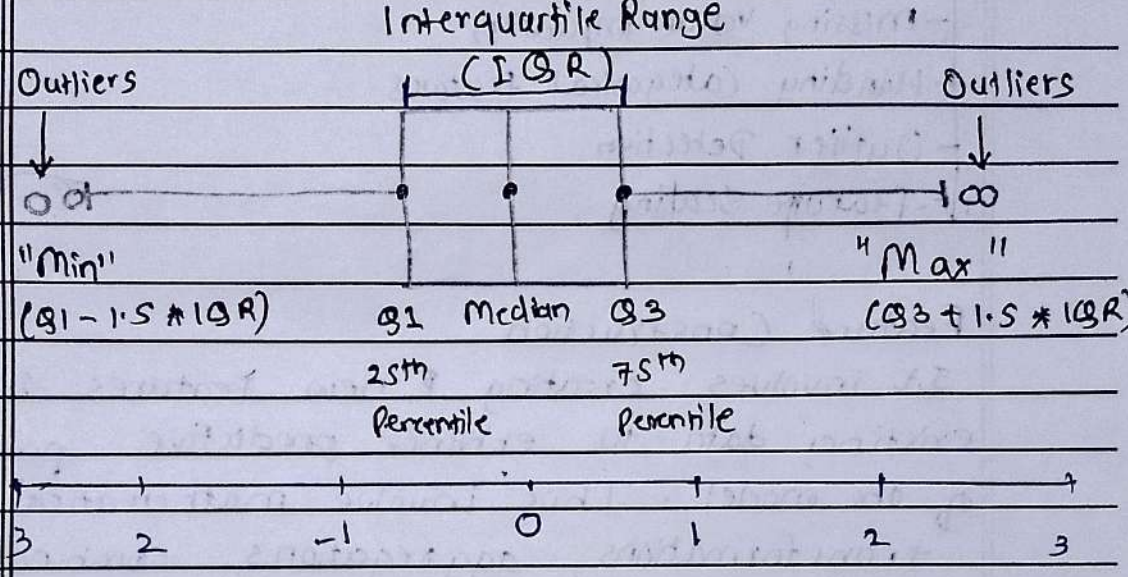
Outlier Treatment:

Trimming
directly remove the outliers

Capping
set an boundry to lower & upper limit for outliers

* Outlier Detection & Removal using IQR Method

The IQR Method is used at the time when our data is skewed on right or left side



IQR Proximity Rule

$$IQR = Q3 - Q1$$

$$IQR = 75\text{th percentile} - 25\text{th percentile}$$

• Outlier Detection Using Percentile Method

For applying Percentile method their is no condition is mention we can apply on any type of data distribution.

In this method we set the equal amount of boundary on upper & lower limit in percentile.

most common is 1 percentile & 5 percentile.

Feature Construction

Feature Engineering

Feature transformation · Feature Construction · Feature Selection · Feature Extraction

- Missing value imputation
- Handling Categorical features
- Outlier Detection
- feature Scaling

Feature Construction

It involves creating new features from existing data to enhance predictive power of the model. This involve mathematical transformations, aggregations, combinations of existing features & domain-specific knowledge to generate features.

Feature Splitting

Feature splitting involves breaking down a single feature into multiple components, which can provide more granular information and potentially improve model performance.

Example:

1. Splitting a timestamp into year, month, day, hour etc.
2. Spitting text features
   Spitting Categorical features into binary (dummy) variables.

Feature selection & Extraction in COD

Curse of Dimensionality
Curse of features

The curse of dimensionality is a phenomenon that occurs when analyzing & organizing data in high-dimensional spaces.

key aspects:

1. Volume Increase : As dimensions increases the volume of the spaces grows exponentially.
2. Sparsity : Data point become sparse. The distance between data points increases.
3. Increased Computational Complexity : requires more computational resources to process.
4. Overfitting in ML models :

To Reduce the Curse of Dimensionality we use

Dimensionality Reduction Techniques

Feature Selection	Feature Extraction
(1) Forward selection	1) PCA
(2) Backword Selection	2) LDA
(3) Bi-directional selection	3) t-SNE

Principle Component Analysis (PCA)

PCA converts the higher dimensional data into best possible lower dimensional data.

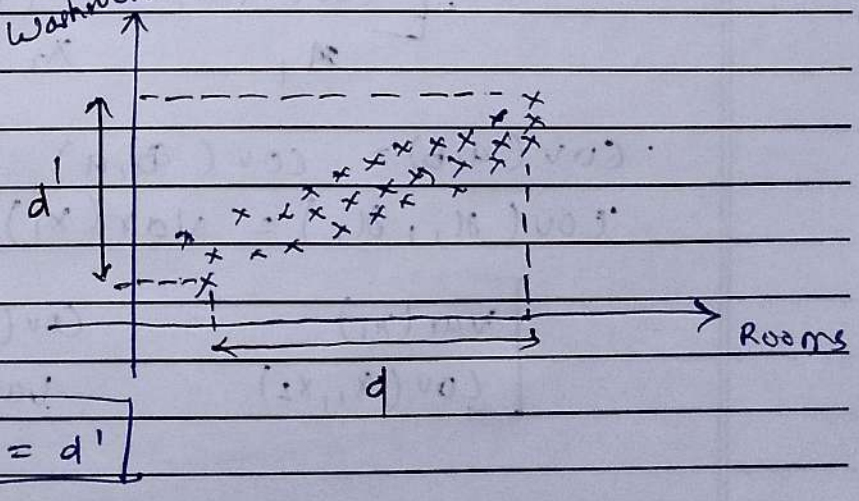
Feature Extraction is an updation over on feature selection's limitations.

Benefits of PCA

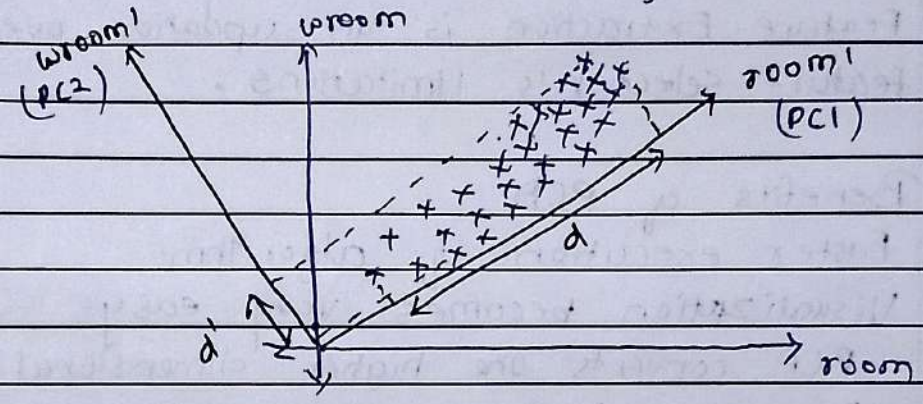
1) Faster execution of algorithm.
2) Visualization becomes very easy
   PCA converts the higher dimensional data into 3D or 2D. if
   eg. 10 D dim. data → PCA → 3D data

* If the data contains the columns in which variance is equally spreaded then the feature selection technique does not work at this time we use PCA. For better understanding see the below fig of equally variance spread & holding in 2 columns.

No. of rooms | No. of washrooms | Price.



According to the data spread the PCAlgo creates new axis at new plane which reduces the any one's dimension among 2 colums the new axis colums called as PC1 & PC2. see below fig.



The No. of PC ≤ n
where n is no of original features in data.

* Key Points:

• Covariance and Covariance matrix
Co-variance is a relation between 2 columns.

Covariance matrix

$$\begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{bmatrix} cov(x_1,x_1) & cov(x_2,x_1) \\ cov(x_1,x_2) & cov(x_2,x_2) \end{bmatrix}_{2\times 2}$$
$$x_1 \qquad x_2$$

• $cov(a,b) = cov(b,a)$
• $cov(x_1,x_1) = var(x_1)$

$$\begin{bmatrix} var(x_1) & cov(x_2,x_1) \\ cov(x_1,x_2) & var(x_2) \end{bmatrix}_{2\times 2}$$

Similarly for 3×3, x|y|z

	x	y	z
x	V_x	$C_{x,y}$	$C_{x,z}$
y	$C_{y,x}$	V_y	$C_{y,z}$
z	$C_{z,x}$	$C_{z,y}$	V_z

- Eigen vectors.

These are special vector. The Direction of eigen vectors is never changed while applying transformation. only magnitude changes direction remains same in eigen vector. $A\vec{v} = \lambda\vec{v}$, always true.

- Eigen values.

These are the measurement of how much the eigen vector is stretched or shrinked after transformation

measures the amount of stretching or shrinking of eigen vector.

Problem:

Find a unit vector which miximises the variance.

The largest eigenvector of covariance matrix always points into the direction of largest variance of data.

* STEPS TO SOLVE PCA

step1 : mean centring of the whole data

step2 : find covariance matrix

step3 : Find the eigen value & eigen vectors for the covariance matrix. ($\lambda_1, \lambda_2, \lambda_3$ are eigen values)

step4 : After getting Principle Components transform all the points from higher dim space to lower w.r.t. Principle Component.

for transforming points
firstly do the (dot)(.) product of $U^T \cdot X$ with all points one by on on principle Component and take transpose of the data shape.

(1000, 3) (2, 3)
↓ ↓ ↕ ↘
Row Col vectors 3D space

(1000, 3) · (3, 2) (Transpose of vectors)

(1000, 2)

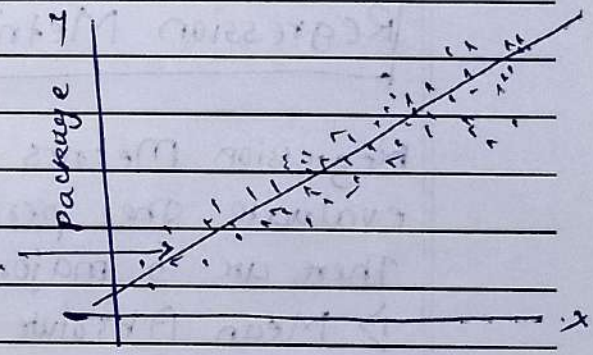
SARASWATI Education Society's
SARASWATI College of Engineering

Simple Linear Regression

The Simple LR is a supervised learning method. The LR algorithm draws a line of best fit on a linear sort of data.

formula : $\boxed{y = m \cdot x + b}$

where y is dependently variable,
x is independent var.
m is weightage
b is offset



import.

~~from sklearn.model_sele~~

from sklearn.linear_model import LinearRegression

make object

lr = LinearRegression()

$$\boxed{b = \bar{y} - m\bar{x}}$$

$$\boxed{m = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}}$$

Equation of Line of best fit:

$$y = m\bar{x} + b$$

where, $\bar{y}$ = mean of y_i
$\bar{x}$ = mean of x_i
y_i = current value of y
x_i = current value of x

Regression Metrics

Regression Metrics are measures used to evaluate the performance of a regression model. There are 5 majorly used Metrics.

1) Mean Absolute Error (MAE)
2) Mean Squared Error (MSE)
3) (RMSE) Root Mean Squared Error
4) R-squared (R-2)
5) Adjusted (R-2)

* Loss Functions : It quantifies the difference between the predicted values and the actual values for a single data point. It is also called as Error function.
MAE, MSE & RMSE are comes under Loss functions.

1. Mean Absolute Error (MAE)

Measures the average magnitude of errors treating all errors equally regardless of their direction.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

or

$$= \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

Advantages,

1) It has always a unit of dependent variable.
2) Robust to outliers.

Disadvantages

1) Modulus graph is not differentiable

2. Mean Squared Error (MSE)

Measures average of squares of the errors

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

or

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

Advantages

Differentiable
We as loss function

Disadvantage

Not Robust to Outliers.

3) Root Mean Square Error:

RMSE = $\sqrt{MSE}$ give highe weight to large errors.

$$= \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}} \quad \text{or} \quad \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Advantage

Output is in same unit of dependent var.

disadvantage:

Robust to outliers

* Absolute

4) R2 Score

It is also coined as coefficient of determination or goodness of fit.

$$R2 = 1 - \frac{SSR}{SSM}$$

where, SSR = Sum of squared error Regression

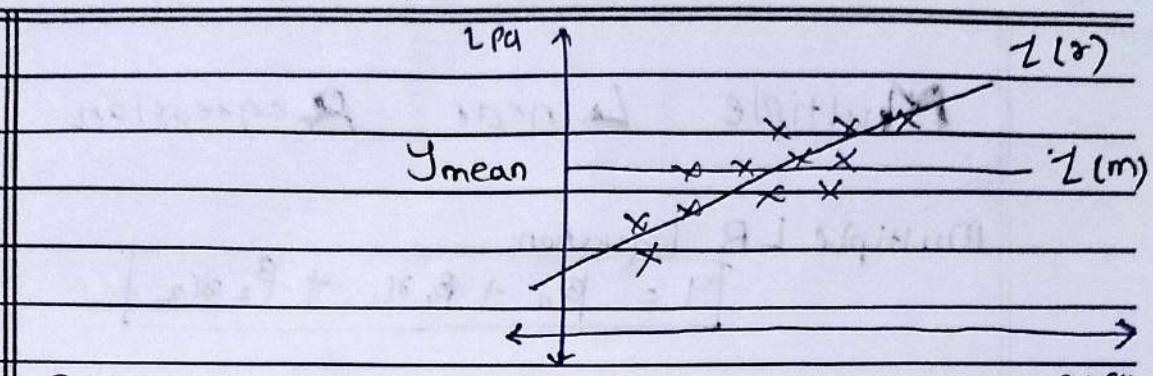
SSM = Sum of squared error Mean

$$R2 = 1 - \frac{\left[\sum_{i=1}^{n} (y_i - \hat{y}_i)^2\right]_R}{\left[\sum_{i=1}^{n} (y_i - \hat{y}_i)^2\right]_M}$$

if $R_2 \to 0$ then bad

$R_2 \to 1$ then good

$R_2 \to$ -ve then SSR > SSm very bad.



Problem with R2 score.
R2 score increases with irrelavent features also R2 score fluctuat to high or in increment ~~when~~ also when any un important feature also added.

4) Adjusted R2 score
Adjust the R2 value based on the no of predictor in the model.

$$\text{Adjusted } R2 = 1 - \left[\frac{(1-R_2)(n-1)}{(n-1-k)}\right]$$

where, n → no. of rows
k → total no of independent columns.

Multiple Linear Regression

Multiple LR Equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

The Multiple LR is an extension of simple LR that models the relationship between two or more independent variables & a single dependent variable by fitting linear eqn to observed data.

Equation in genral form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

where y is independent var.

β_0 is intercept / offset

$\beta_1, \beta_2, \beta_3, \ldots, \beta_n$ are coefficient for independent variables

$x_1, x_2, \ldots, x_p$ are independent variables

ϵ is a error term.

For understanding this algorithms in detail lets take an example:

We have data of 100 students containing 4 columns:

	cgpa	iq	graduate	LPA
(100,4)	x_1	x_2	x_3	y

is shape of data.

eqn of hyperplane is -

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

where $\hat{y}$ is y predicted

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{100} \end{bmatrix} = \begin{bmatrix} \beta_0 & \beta_1 x_{11} & \beta_2 x_{12} & \beta_3 x_{13} \\ \beta_0 & \beta_1 x_{21} & \beta_2 x_{22} & \beta_3 x_{23} \\ \vdots & & & \\ \beta_0 & \beta_1 x_{1001} & \beta_2 x_{1002} & \beta_3 x_{1003} \end{bmatrix}$$

for n rows and m columns.

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} \beta_0 & \beta_1 x_{11} & \beta_2 x_{12} & \beta_3 x_{13} \cdots \beta_m x_{1m} \\ \beta_0 & \beta_1 x_{21} & \beta_2 x_{22} & \beta_3 x_{23} \cdots \beta_m x_{2m} \\ \vdots & & & \\ \beta_0 & \beta_1 x_{n1} & \beta_2 x_{n2} & \beta_3 x_{n3} \cdots \beta_m x_{nm} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \cdots x_{1m} \\ 1 & x_{21} & x_{22} & x_{23} \cdots x_{2m} \\ \vdots & & & \\ 1 & x_{n1} & x_{n2} & x_{n3} \cdots x_{nm} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix}$$

A or X $\qquad$ β

$$\hat{Y} = X\beta \quad \text{— (1)}$$

where X is matrixes & β is coefficient matrics

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad e = y - \hat{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}$$

$$e = \begin{bmatrix} y_1 - \hat{y_1} \\ y_2 - \hat{y_2} \\ \vdots \\ y_n - \hat{y_n} \end{bmatrix}$$

$$E = e^T e$$

$$\begin{bmatrix} (y_1 - \hat{y_1}) & (y_2 - \hat{y_2}) \cdots (y_n - \hat{y_n}) \end{bmatrix} \begin{matrix} (y_1 - \hat{y_1}) \\ (y_2 - \hat{y_2}) \\ \vdots \\ (y_n - \hat{y_n}) \end{matrix}$$

$(1 \times n)$ $\quad$ $(n \times 1)$

$$(y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2 + (y_3 - \hat{y_3})^2 + \cdots + (y_n - \hat{y_n})^2$$

$$\boxed{\sum_{i=1}^{n} (y_i - \hat{y_i})^2}$$

$\because (A+B)^T = A^T + B^T$

$\because (A-B)^T = A^T - B^T$

$$E = e^T e = (y - \hat{y})^T (y - \hat{y})$$

$$= (y^T - \hat{y}^T)(y - \hat{y})$$

$$= \left[y^T - (X\beta)^T\right](y - X\beta)$$

... from (1) $\boxed{\hat{y} = X\beta}$

$$E = y^T y - y^T X\beta - (X\beta)^T y + (X\beta)^T X\beta$$

$$\boxed{E = y^T y - 2y^T X\beta + \beta^T X^T X\beta}$$

From ... $y^T X\beta = X\beta^T y$

$A^T B = B^T A$

differentiate the eqn

$$\frac{dE}{d\beta} = \frac{d}{d\beta}\left[y^T y - 2y^T x\beta + \beta^T x^T x\beta \right] = 0$$

$$= 0 - 2y^T x + \frac{d}{d\beta}\left[\beta^T x^T x\beta \right] = 0$$

$$= -2y^T x + 2x^T x\beta^T = 0$$

$$= 2x^T x\beta^T = 2y^T x$$

$$\beta^T = y^T x \cdot \frac{1}{[x^T x]}$$

$$\beta^T = y^T x (x^T x)^{-1}$$

$$(\beta^T)^T = \left[y^T x (x^T x)^{-1} \right]^T$$

$$\beta = \left[(x^T x)^{-1} \right]^T (y^T x)^T$$

$$\beta = \left[(x^T x)^{-1} \right]^T x^T y$$

$$\beta = (x^T x)^{-1} x^T y$$

$$\boxed{\beta = (x^T x)^{-1} x^T y}$$

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix}$$

$[(m+1)*(m+1)]\ [(m+1)*n]\ [n*1]$

$[(m+1)*n]\ [n*1]$

$(m+1) * 1$

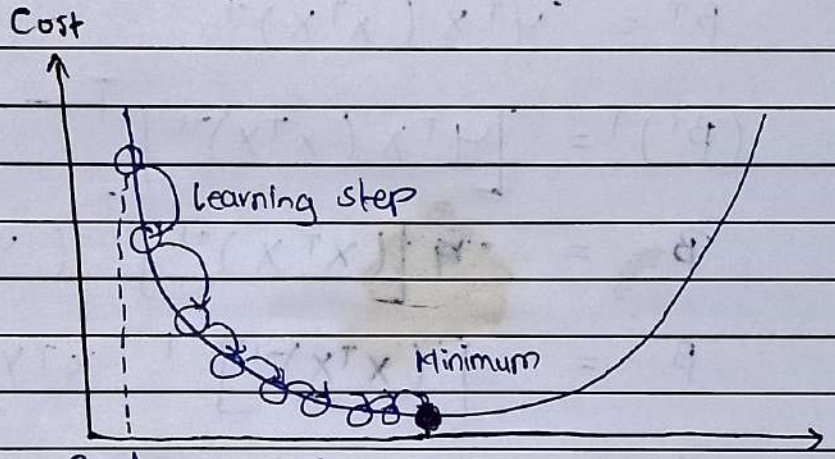
$(m+1)*1 = (m+1) * 1$

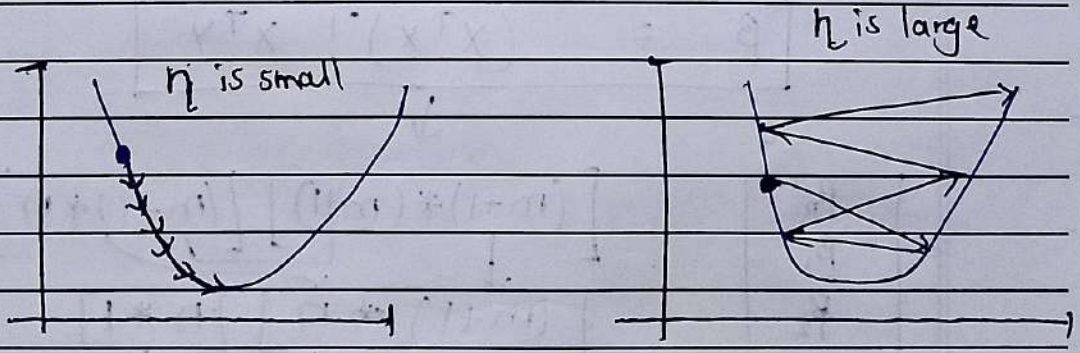
$\beta = (x^T x)^{-1} x^T y$ //

GRADIENT DESCENT

Gradient : It is a derivative that defines the effects on outputs of the function with a little bit of variation in inputs

Gradient descent : is a method for unconstrained mathematical optimization. It is a first order iterative algorithm for finding a local minimum of a differentiable multivariate function.



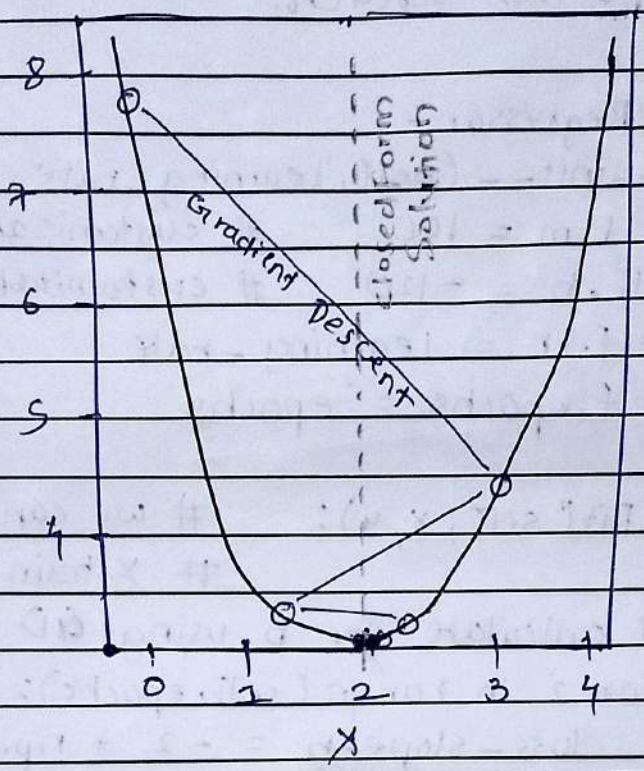


(η) is learning rate.

$$b_{new} = b_{old} - slope$$

$$\downarrow$$

$$b_{new} = b_{old} - \eta\, slope$$



Gradient descent = $-2 \sum_{i=1}^{n} (y_i - mx_i - b)$

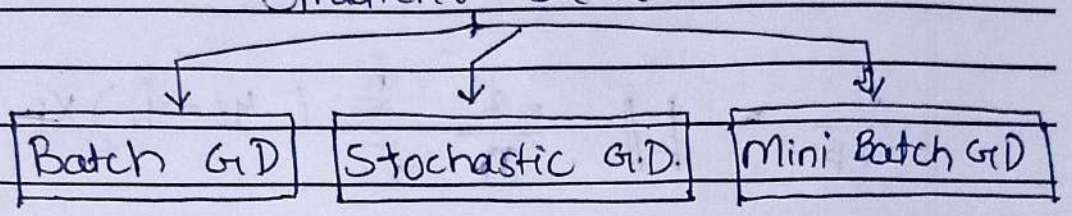
Formula in code

```
for i in range (self.epochs):
    loss_slope_b = -2 * np.sum(y - self.m * X.ravel()
                                self.b)
    loss_slope_m = -2 * np.sum(y - self.m *
                   X.ravel()-self.b). *
                                X.ravel())
    self.b = self.b - (loss_slope_b * self.lr)
    self.m = self.m - (loss_slope_m * self.lr)
```

Own Algo from scratch.

```
line.
1   class GDRegressor:
2       def __init__(self, learning_rate, epochs):
3           self.m = 100          # customizable
4           self.b = -120         # customisable
5           self.lr = learning_rate
6           self.epochs = epochs
7
8       def fit(self, X, y):      # we can replace with.
                                  # X_train, y_train.
9           # calculate the b using GD
10          for i in range(self.epochs):
11              loss_slope_b = -2 * np.sum(y -
                    self.m * X.ravel() - self.b)
12              loss_slope_m = -2 * np.sum((y -
                self.m * X.ravel() - self.b) * X.ravel())
13
14              self.b = self.b - (loss_slope_b * self.lr)
15              self.m = self.m - (loss_slope_m * self.lr)
16          print(self.m, self.b)
17
18      def predict(self, X):
19          return self.m * X + self.b
    # make object.
    gd = GDRegressor(lr = 0.001, epochs = 100)
    gd.fit(X, y)   # replace with X_train, y_train
    gd..
    y_pred = gd.predict(X_test)
```

* Types of Gradient Descent *



• Batch Gradient Descent

Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred as a training epoch. Batch GD usually produces a stable error gradient & convergence.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

$$\beta_0 = \beta_0 - \eta \cdot \text{slope} = \frac{\partial L}{\partial \beta_0}$$

$$\beta_1 = \beta_1 - \eta \, \text{slope} = \frac{\partial L}{\partial \beta_1}$$

$$\beta_2 = \beta_2 - \eta \, \text{slope} = \frac{\partial L}{\partial \beta_2}$$

$$\beta_n = \beta_n - \eta \cdot \text{slop} = \frac{\partial L}{\partial \beta_n}$$

Loss function & its derivatives on each epochs -

Loss function for MSE

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

$$\frac{\partial L}{\partial \beta_1} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})x_{i1}$$

$$\frac{\partial L}{\partial \beta_2} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})x_{i2}$$

$$\frac{\partial L}{\partial \beta_3} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})x_{i3}$$

General formula of loss function.

$$\boxed{\frac{\partial L}{\partial \beta_m} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})x_{im}}$$

Formula for intercept (β_0) derivative.

$$\boxed{\frac{\partial L}{\partial \beta_0} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})}$$ //

Stochastic Gradient Descent (SGD)

Stochastica gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. These make SGD faster. Its Frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum & finding global one.

Intercept (β_0)

$$\frac{\partial L}{\partial \beta_0} = -2 (y_i - \hat{y_i})$$

where, $i = idx$

Coefficien. (β_1)

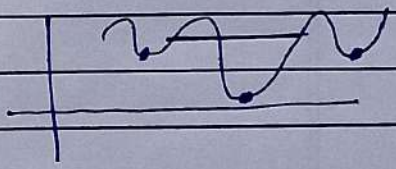
$$\frac{\partial L}{\partial \beta_1} = -2 \{ y[i] - \hat{y}), X[i] \}$$

$$-2 [(y_i - \hat{y}), x_i)]$$

When no of epochs are fixed in same size for batch & stochastic comparing with the speed batch computes faster than stochastic

When to use Stochastic GD

1) Big data
2) Non convex function



• Mini-Batch Gradient Descent.

Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batches and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent & the speed of stochastic gradient descent.

Min batch = Batch GD + Stochastic GD

* Polynomial Regression

Polynomial regression is a form of regression analysis in which the relationship between the independent variable X and the dependent variable y is modeled as an nth degree polynomial in X.
Polynomial Regression is a special case of Linear Regression where we fit the polynomial equation on the data with a curvilinear relationship between the dependent and independent variables.

Simple LR $\Rightarrow y = \beta_0 + \beta_1 x_1$

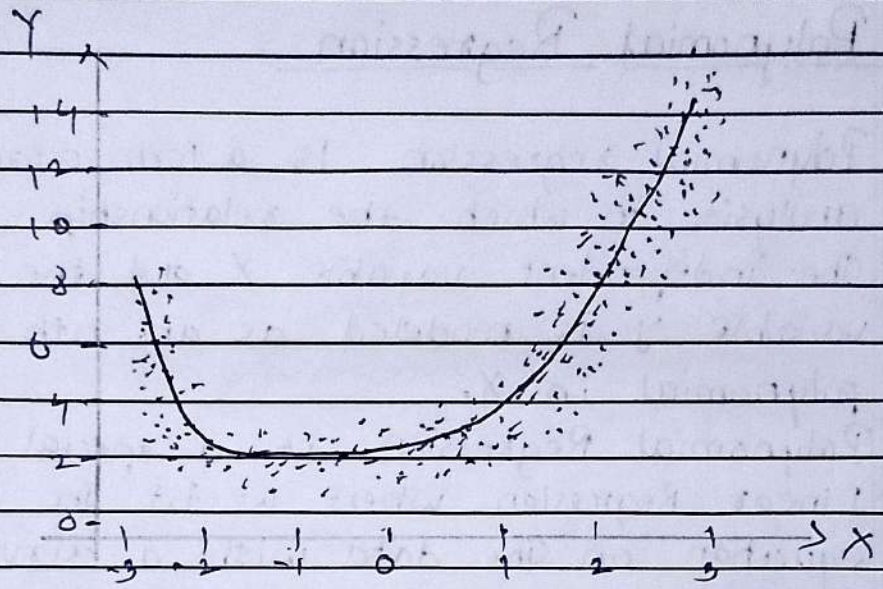
Multiple LR $\Rightarrow y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$

Polynomial LR $\Rightarrow y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \dots + \beta_n x_1^n$

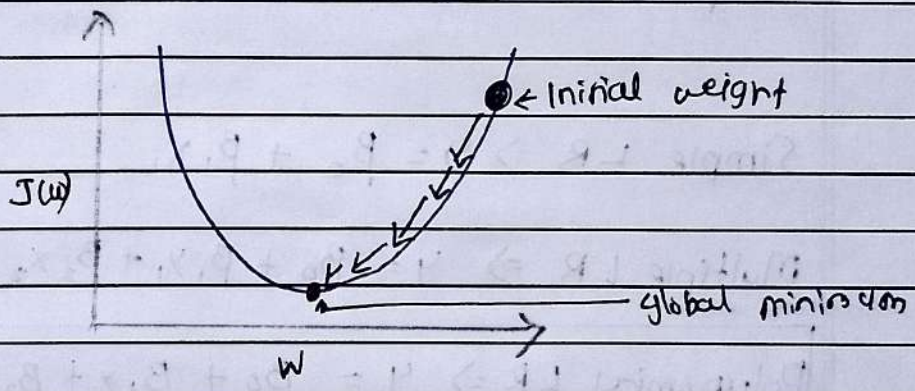
Cost Function : CF of Polynomial is Mean Square Error,

$$J = \frac{1}{n} \sum_{i=1}^{n} (Pred_i - y_i)^2$$

# cost Function of Polynomial Reg.
J = 1/n * sum(square(pred - y))
and one form,
J = 1/n * sum(square(pred - ($\beta_0 + \beta_1 x + \beta_2 x^2 + b3x^3 \dots$)))



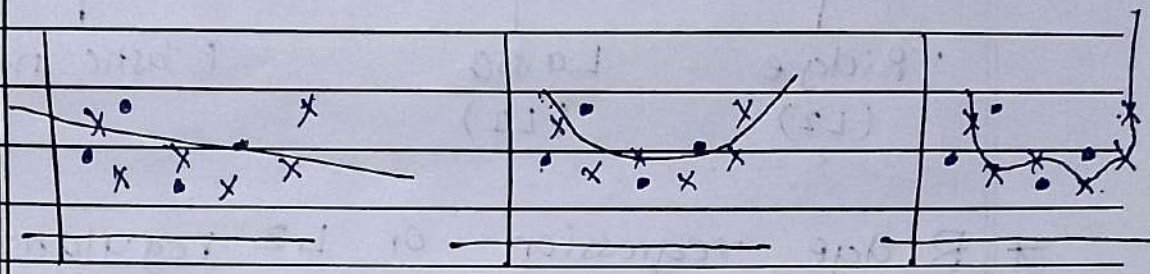
Polynomial regression.



Gradient Descent for Polynomial Reg.

BIAS VARIANCE TRADE-OFF

Bias-variance tradeoff describes the relationship betn a mode's complexity. it trying to minimize the 2 sources of error which are bias & variance error



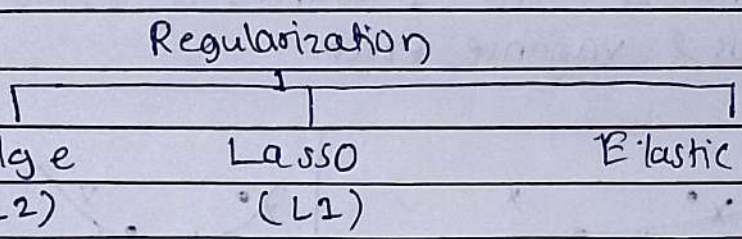
	Model	Model 2	Model 3
Bias	High	Mid Low	Low
Variance	Low	Low	High
fitting	Underfitting	best fit	Overfitting
Model	Simple	Sweet Spot	Complex

- High variance means model makes strong assumptions about the data leads to underfitting.
- High variance means model is very sensitive to small fluctuations in the training data, leads to overfitting.

For Handing Bias variance Trade-Off: 3 3 important techniques. 1) Bagging
2) Boosting
3) Regularization

Regularization

Regularization is a set of methods for reducing overfitting in ML models.



* Ridge regression - or L2 regularization

It penalizes high-value coefficients by introducing a penalty term in SSE Loss fn.

- Using Ridge in code using sklearn. class.

```
import.
from sklearn.linear_model import Ridge
make obj
R = Ridge (alpha = 0.01)  # alpha is hyper-parameter.
train the model.
R.fit (X_train, y_train)
Predict.
R.predict (X_test).
```

Ridge Regression Mathematical Formulation.

Loss function. (L) $= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda m^2$

$$L = \sum_{i=1}^{n} (y_i - mx_i - b)^2 + \lambda m^2$$

$\frac{\partial L}{\partial b}$ & $\frac{\partial L}{\partial m}$.

$\frac{\partial L}{\partial b} = 0$, from this.

$b = \bar{y} - m\bar{x}$ where $\bar{y} \rightarrow$ y-mean
$\bar{x} \rightarrow$ x-mean.

Now.

$$L = \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})^2 + \lambda m^2$$

(replace b by $\bar{y} - m\bar{x}$)

$$\frac{\partial L}{\partial m} = 2\sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})(-x_i + \bar{x}) + 2\lambda m = 0$$

$$= -2\sum_{i=1}^{n} (y_i - \bar{y} - mx_i + m\bar{x})(x_i - \bar{x}) + 2\lambda m = 0$$

$$= \lambda m - \sum_{i=1}^{n} [(y_i - \bar{y}) - m(x_i - \bar{x})](x_i - \bar{x}) = 0$$

$$= \lambda m - \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2 = 0$$

$$= \lambda m - \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) + m\sum_{i=1}^{n} (x_i - \bar{x})^2 = 0$$

$$= \lambda m + m\sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})$$

$$= m = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2 + \lambda}$$

$$m = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2 + \lambda}$$

$$b = \bar{y} - m\bar{x}$$

loss function of N-Dim. Data.

$$L = (xw - y)^T (xw - y) + \lambda \|w\|^2$$

$$W = (X^T X)^{-1} X^T Y$$

~~$W = (X^T X$~~

$$W = (X^T X + \lambda I)^{-1} X^T Y$$

code for Ridge Reg. for N-dim. dataset.

```
class MyRidge:
    def __init__(self, alpha = 0.1):
        self.alpha = alpha
        Self.coef_ = None
        self.intercept_ = None

    def fit(self, X_train, y_train):
        X_train = np.insert(X_train, 0, 1, axis=1)
        I = np.identity(X_train.shape[1])
        I[0][0] = 0
        result = np.linalg.inv(np.dot(X_train.T,
            X_train) + np.dot(self.alpha, I)).
            dot(X_train.T).dot(y_train)
```

```
        self.coef_ = result[1:]
        self.intercept_ = result[0]

    def predict(self, X_test):
        return np.dot(self.coef_, X_test)
                + self.intercept_

# Our class is created Now Let, test it.

reg = MyRidge()
reg.fit(X_train, Y_train)
Y_pred = reg.predict(X_test)
r2_score(Y_test, Y_pred)
reg.coef_
reg.intercept_
```

Ridge Regression using Gradient Descent

Gradient Descent...

Loss function of linear reg.

$$L = \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

Loss function in vector form. for linear. reg.

$$L = (xw - y)^T (xw - y)$$

Ridge Reg. Loss fun.

$$L = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 + \lambda m^2$$

LF in Vector form.

$$L = (xw - y)^T (xw - y) + \lambda \|w\|^2$$

$$L = (xw - y)^T (xw - y) + \lambda \cdot w^T w$$

where,

$$x = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ 1 & x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \rightarrow (n+1)$$

$$L = (xw - y)^T (xw - y) + \lambda w^T w$$

mul by $\frac{1}{2}$

$$L = \frac{1}{2} (xw - y)^T (xw - y) + \frac{1}{2} \lambda w^T w$$

$$L = \frac{1}{2} [w^T x^T x w - w^T x^T y - y^T w x + y^T y] + \frac{1}{2} \lambda w^T w$$

$$L = \frac{1}{2} [w^T x^T x w - 2 w^T x^T y + y^T y] + \frac{1}{2} \lambda w^T w$$

$$\frac{\partial L}{\partial w} = \frac{1}{2} [2 x^T x w - 2 x^T y] + \frac{1}{2} 2 \lambda w$$

$$\frac{\partial L}{\partial w} = \boxed{x^T x w - x^T y + \lambda w} = \left(\frac{\Delta L}{\Delta w}\right)_{11}$$

Key Points about Ridge Regression

1) How the coefficients get affected (β)

if λ value keep 0 then act as linear reg
If λ value is increase from 0 to grater number all the coefficients will get shrinked get decreased towards zero but never turn zero.

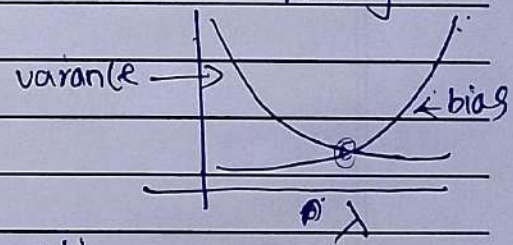
2) Higher values are impacted more.

While increasing λ towards grater value (∞) the higher values will decreased faster than ~~lower~~ lesser values. but not closed to zero
higher value = higher impact.

3) Bias Variance Trade off.

Bias & variance both affect by λ value.
if λ ⇒ 0 then bias ↓ overfitting.
λ ⇒ ∞ grater then bias ↑ underfiting.

λ value should be
= bias low &
varian low.



4) Impact on the Loss Function

as λ value increase the Loss function moves towards zero. but not become zero.

5) Why called Ridge

- Hard Constraint Ridge regression ~~is~~ or
- λ soln come on Ridge of circle / perimeter of circle for best results. best soln for MSE & λ value.

• LASSO REGRESSION (L1 Regularization)

Lasso regression is a regularization technique that penalizes high value, correlated coefficients. It introduces a regularization term (also called penalty term) into the (SSE) loss function. This penalty term is absolute value of the sum of coefficients. Controlled in turn by hyperparameter lambda(λ), it reduces select feature weights to zero. Lasso regression removes multicollinear feature from the model altogether. (auto feature selection is applied)

Lasso Reg. Loss fun.

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \boxed{\lambda \cdot ||W||} \rightarrow \text{Penalty term}$$

$$L = MSE + \lambda ||W||$$

where, $||W|| \Rightarrow [|w_1| + |w_2| + |w_3| + \dots + |w_n|]$

>> import lasso for code from sklearn.

```
from sklearn.linear_model import Lasso
Lass_model = Lasso(alpha= 0.5)
Lass_model.fit(X_train, y_train)
Lass_model.predict(X_test)
```

Key point in Lasso Regression

1) How are coefficient affected ?
λ value ↑ then coefficient get shrinked.
unimportant values become zero
when λ value ↑
higher impact for decrementation of higher important values (coefficients).

2) Higher coefficient are affected more
as λ value ↑ Higher coefficient decreased fast & at last become zero.

3) Impact on Bias & Variance.
λ value ↑.
overfiting ↓ & Bias ↑ & variance ↓
~~variance ↑ & variance ↓~~

4) Effect of Regularization on loss function
λ value ↑ increases
loss function ~~move~~ (curve) moves towards zero.
At the end it will reach zero.

5) Sparsity is created by lasso reg.

Lasso reg. & loss function.

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda |m|$$

$(b = \bar{y} - m\bar{x})$

$$L = \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})^2 + 2\lambda|m|$$

$$\frac{\partial}{\partial m} = \sum_{i=1}^{n} (y_i - mx_i - \bar{y} + m\bar{x})^2 + 2\lambda m$$

$$= 2\Sigma (y_i - mx_i - \bar{y} + m\bar{x})(-x_i + \bar{x}) + 2\lambda = 0$$

$$= -2\Sigma [(y_i - \bar{y}) - m(x_i - \bar{x})](x_i - \bar{x}) + 2\lambda = 0$$

$$= -\Sigma [(y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2] + \lambda = 0$$

$$= -\Sigma [(y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2] + \lambda = 0$$

$$= -\Sigma (y_i - \bar{y})(x_i - \bar{x}) + m\Sigma (x_i - \bar{x})^2 + \lambda = 0$$

$$\therefore m\Sigma (x_i - \bar{x})^2 = \Sigma (y_i - \bar{y})(x_i - \bar{x}) - \lambda$$

$$m = \frac{\Sigma (y_i - \bar{y})(x_i - \bar{x}) - \lambda}{\Sigma (x_i - \bar{x})^2} \quad //$$

for $m > 0$

$$m = \frac{\Sigma (y_i - \bar{y})(x_i - \bar{x}) - \lambda}{\Sigma (x_i - \bar{x})^2}$$

for $m = 0$

$$m = \frac{\Sigma (y_i - \bar{y})(x_i - \bar{x})}{\Sigma (x_i - \bar{x})^2}$$

for $m < 0$

$$m = \frac{\Sigma (y_i - \bar{y})(x_i - \bar{x}) + \lambda}{\Sigma (x_i - \bar{x})^2}$$

Now for sparsity lets take an example.

assume - $(y_i - \bar{y}) = Y$

$(x_i - \bar{x}) = X$

for $m > 0$.

$$m = \frac{YX - \lambda}{X^2}$$

suppose $YX = 100$

$X^2 = 50$

$$m = \frac{100 - \lambda}{50}$$

for $\lambda = 0$ $\quad m = \frac{100 - 0}{50} = \frac{100}{50} = 2.$

$\boxed{m = 2}$

for $\lambda = 10$ $\quad m = \frac{100 - 10}{50} = \frac{90}{50} = \boxed{\frac{9}{5}}$

for $\lambda = 50$ $\quad m = \frac{100 - 50}{50} = \frac{50}{50} = \boxed{1}$

for $\lambda = 100$ $\quad m = \frac{100 - 100}{50} = \frac{0}{50} = \boxed{0}$

when $\boxed{YX = \lambda}$ then $\boxed{m = 0}$

when λ values goes in -ve then we foll $m < 0$ eqn. which is ~~YX~~.

$$m = \frac{YX + \lambda}{X^2}$$ which again

increse the m value to +ve greater so algo stops at ~~Dact~~ p. 0

Now, for
$m < 0$.

$$m = \frac{\Sigma(y_i - \bar{y})(x_i - \bar{x}) + \lambda}{\Sigma(x_i - \bar{x})^2}$$

$$m = \frac{xy + \lambda}{x^2} \quad \left\{ \begin{array}{l} xy = -100 \\ x^2 = 50. \end{array} \right.$$

when $\lambda = 0$, $\frac{-100+0}{50} = -2$ $\boxed{m = -2}$

$\lambda = 50$, $\frac{-100+50}{50} = -1$ $\boxed{m = -1}$

$\lambda = 100$, $\frac{-100+100}{50} = 0$ $\boxed{m = 0}$

$\lambda = 150$, $\frac{-100+150}{50} = \frac{50}{50} = \boxed{m = 1}$

for $\lambda = 150$, m become 1 so, $m > 0$.
or $m \neq 0$.

So, we will use $m > 0$ formula.

$$m = \frac{xy - \lambda}{x^2} = \frac{-100-150}{50} = \frac{-250}{50}$$

$$= \frac{-250}{50} = \boxed{-5} //$$

$\boxed{m = -5}$, so we can't go agin for larg value than -2 so algo stops at zero 0

2) Why Ridge Reg not create sparsity.

→ Ridge Reg. formula.

$$= \frac{\Sigma(y_i - \bar{y})(x_i - \bar{x})}{(x_i - \bar{x})^2 + \lambda}$$

in ridge reg. λ is at denominator so it can't be zero but in lasso λ is in numerator so it can be zero. cause denominator can't be zero //.

* ElasticNet Regression

- Elastic net regression essentially combines both ridge and lasso regression but inserting both the L1 & L2 penalty terms into the SSE loss function. Elastic net inset both penalty values into the cost function equation. Elastic net a. handles ~~rest~~ addresses multicollinearity while also enabling feature selection.

Loss function

$$L = \sum (y_i - \hat{y}_i)^2 + a||w||^2 + b||w||$$

where a & b are independent.

$a = 0.5$ & $b = 0.5$. both have equal weightage.

$$\lambda = a+b \;;\; l1\_ratio = \frac{a}{a+b}$$

when λ & l1_ratio are hyperparameters.

~~l1_ratio = 0.9. => 90% ridge & 10% lasso~~

90% ridge & 10% lasso

~~l1_ratio = 0.4. => 40% ridge & 60% lasso~~

l1_ratio = 0.9 ⇒ 90% lasso & 10% rigid.

l1_ratio = 0.4 ⇒ 40% lasso & 60% rigid.

implement elasticnet using sklearn.

```
from sklearn.linear_model import ElasticNet
reg = ElasticNet(alpha=0.005, l1_ratio=0.9)
reg.fit(X_train, Y_train)
reg.predict(X_test)
```

Prerequsist for applying logistic regression.
- Data must be linearly separable

Perceptron Trick

LOG Reg eqn $Ax + By + C = 0$

$Ax_1 + Bx_2 + C = 0$

$Ax_1 + Bx_2 + Cx_3 + d = 0$ → $[A, B, C]$

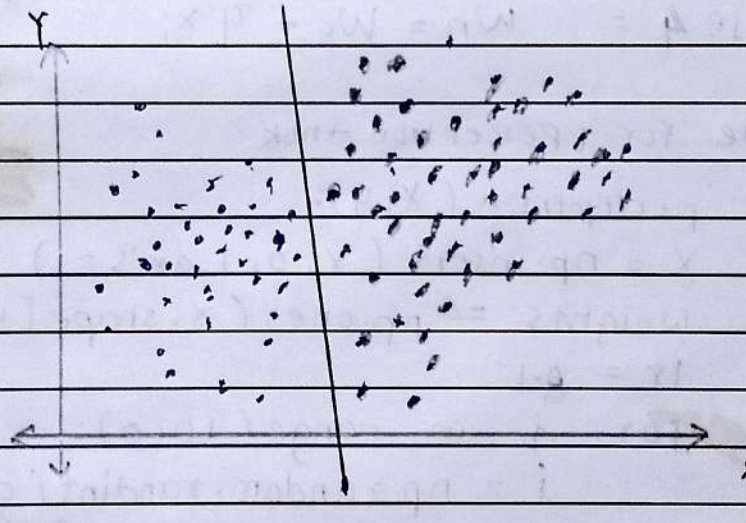
How to labe regions.

if eqn of line is to point is on

$Ax_1 + By_1 + C > 0$ = +v region

$Ax_1 + By_1 + C < 0$ = −v region

$Ax_1 + By_1 + C = 0$ = on the line



Algo:

$Ax + By + C = 0$

$W_0 + W_1 x_1 + W_2 x_2 = 0$, $W_0 = C, W_1 = A, W_2 = B$

$W_0 x_0 + W_1 x_1 + W_2 x_2 = 0$

$\sum_{i=0}^{2} W_i x_i = 0$ $\quad [W_0\ W_1\ W_2] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$

Algorithm. (1)

if $x_i \in N$ and $\sum w_i x_i \geq 0$

$w_n = w_o - \eta x_i$

if $x_i \in P$ and $\sum w_i x_i < 0$

$w_n = w_o + \eta x_i$

where, N = Negative Point

P = Positive Point

Algorithm (2)

$w_n = w_o + \eta (y_i - \hat{y_i}) x_i$

		$y_i - \hat{y_i}$	y_i	$\hat{y_i}$
case 1 =	$w_n = w_o$	0	1	1
case 2 =	$w_n = w_o$	0	0	0
case 3 =	$w_n = w_o + \eta x_i$	1	1	0
case 4 =	$w_n = w_o - \eta x_i$	-1	0	1

code for perceptron trick.

```
def perceptron(X, y):
    X = np.insert(X, 0, 1 axis=1)
    weights = np.ones(X.shape[1])
    lr = 0.1
    for i in range(1000):
        j = np.random.randint(0, 100)
        y_hat = step(np.dot(X[j], weights))
        weights = weights + lr * (y[j] - y_hat)
                  * X[j]
    return weights[0], weights[1:]
```

Logistic Regression.

For logistic reg. we have to make changes in our Algo. We can't make $(y_i - \hat{y}) = 0$

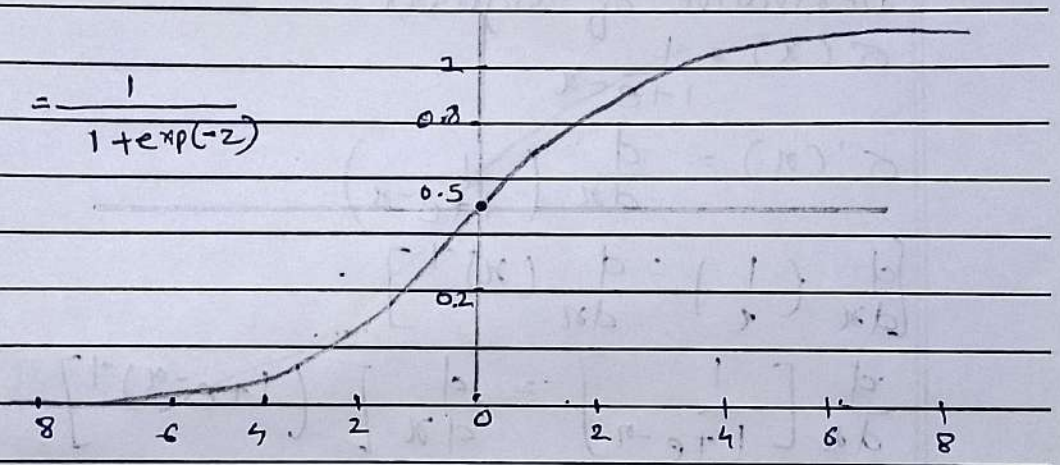
if $(y_i - \hat{y}_i) = 0$ then $w_n = w_o$ so eqn line will never chang.

$$w_n = w_o + \eta (y_i - \hat{y}_i) x_i$$

$$(y_i - \hat{y}_i) \neq 0$$

The Sigmoid Function

$$= \frac{1}{1 + \exp(-z)}$$



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

if $\sigma(z)$ is +ve then $0.5 >, \Rightarrow 1$

if $\sigma(z)$ is -ve then $0.5 < z) 0$

$$m = -\frac{A}{B} \quad \text{coef}[0]$$

$$\text{coef}[1]$$

$$c = \frac{-c}{B} \quad -\text{intercept}$$

$$\text{coef}[1]$$

$$L = \sum_{i=1}^{n} -y_i \log(\hat{y_i}) - (1-y_i)\log(1-\hat{y_i})$$

log loss error function / binary cross entropy

$$\boxed{L = \frac{-1}{n}\sum_{i=1}^{n} y_i \cdot \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})}$$

Derivative of sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

$$\left[\frac{d}{dx}\left(\frac{1}{x}\right) = \frac{d}{dx}(x)^{-1}\right] \ldots$$

$$\frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}\left[(1+e^{-x})^{-1}\right]$$

$$= \frac{-1}{(1+e^{-x})^2}\frac{d}{dx}(1+e^{-x}) \quad \ldots\ldots \text{(chain rule)}$$

$$= \frac{-1}{(1+e^{-x})^2}\frac{d}{dx}(e^{-x})$$

$$= \frac{-e^{-x}}{(1+e^{-x})^2}\frac{d}{dx}(-x) \Rightarrow \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1 \cdot e^{-x}}{(1+e^{-x})(1+e^{-x})} = \frac{1}{1+e^{-x}} \times \frac{e^{-x}}{1+e^{-x}}$$

$$= \sigma(x)\left[\frac{e^{-x}}{1+e^{-x}}\right] \quad \ldots\ldots \left[\frac{1}{1+e^{-x}} = \sigma(x)\right]$$

$$= \sigma(x)\left[\frac{1+e^{-x}-1}{1+e^{-x}}\right] = \sigma(x)\left[\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right]$$

$$= \sigma(x)[1-\sigma(x)] \Rightarrow$$

$$= \boxed{\sigma'(x) = \sigma(x)[1-\sigma(x)]}$$

Gradient Descent for Logistic Reg.

Rows = m, Columns = n

1	2	3	.....	n	y
x_{11}	x_{12}	x_{13}	.....	x_{1n}	y_1
x_{21}	x_{22}	x_{23}	.....	x_{2n}	y_2
x_{31}	x_{32}	x_{33}	.....	x_{3n}	y_3
⋮					⋮
x_{m1}	x_{m2}	x_{m3}	.....	x_{mn}	y_m

Coeff = $\{w_1\ w_2\ w_3 \dots w_n\} \Rightarrow (n+1)$

$\sigma(w_1x_{11} + w_2x_{12} + w_3x_{13} \dots + w_nx_{1n} + w_0) = \hat{y}_1$

$\sigma(w_1x_{21} + w_2x_{22} + w_3x_{23} + \dots + w_nx_{2n} + w_0) = \hat{y}_2$

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \begin{bmatrix} \sigma(w_0 + w_1x_{11} + w_2x_{12} + \dots + w_nx_{1n}) \\ \sigma(w_0 + w_1x_{21} + w_2x_{22} + \dots + w_nx_{2n}) \\ \vdots \\ \sigma(w_0 + w_1x_{m1} + w_2x_{m2} + \dots + w_nx_{mn}) \end{bmatrix}$$

$$\hat{Y} = \sigma\left(\begin{bmatrix} w_0 + w_1x_{11} + w_2x_{12} + \dots + w_nx_{1n} \\ w_0 + w_1x_{21} + w_2x_{22} + \dots + w_nx_{2n} \\ \vdots \\ w_0 + w_1x_{m1} + w_2x_{m2} + \dots + w_nx_{mn} \end{bmatrix}\right)$$

$$\hat{Y} = \sigma\left(\begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & & & & \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}\right)$$

$$\boxed{\hat{Y} = \sigma(XW)}$$

Loss fun.

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})$$

$$L = -\frac{1}{m}\left[\underbrace{\sum_{i=1}^{m} y_i \log(\hat{y_i})}_{A} + \underbrace{\sum_{i=1}^{m}(1-y_i)\log(1-\hat{y_i})}_{B}\right]$$

take, A.

$$\sum_{i=1}^{m} y_i \log(\hat{y_i}) = y_1 \log \hat{y_1} + y_2 \log \hat{y_2} + y_3 \log \hat{y_3} + \ldots + y_m \log \hat{y_m}$$

we also can write it as.

$$\begin{bmatrix} y_1 & y_2 & y_3 & \ldots & y_m \end{bmatrix} \begin{bmatrix} \log \hat{y_1} \\ \log \hat{y_2} \\ \vdots \\ \log \hat{y_m} \end{bmatrix}$$

$$\begin{bmatrix} y_1 & y_2 & y_3 & \ldots & y_m \end{bmatrix} \log \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_m} \end{bmatrix} = \hat{y}$$

$$y \log \hat{y} = y \log(\sigma(xw))$$

$$\hat{y} = \sigma(xw)$$

$$L = -\frac{1}{m}\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]$$

where $\hat{y} = \sigma(xw)$ //

Loss function in matrix form.

$$L = \frac{-1}{m}\left[y \log(\sigma(wx)) + (1-y)\log(1-\sigma(wx)) \right]$$

Applying Gradient descent

$$\frac{dL}{dw} = -\frac{1}{m}\left[y \log \hat{y} + (1-y)(\log(1-\hat{y})) \right]$$

$$\frac{d}{dw} y \log \hat{y} \Rightarrow y \frac{d}{dw} \log \hat{y} \Rightarrow \frac{y}{\hat{y}} \frac{d}{dL}(\hat{y})$$

$$\Rightarrow \frac{y}{\hat{y}} \frac{d}{dL} \sigma(wx) \Rightarrow \frac{y}{\hat{y}} \sigma(wx)[1-\sigma(wx)] \frac{d}{dw}(wx)$$

$$= \frac{y}{\hat{y}} \cdot \hat{y}(1-\hat{y}) x = \boxed{y(1-\hat{y}) x}$$

$\hat{y} = \sigma(wx)$

$$\frac{d}{dw}(1-y)\log(1-\hat{y}) \Rightarrow (1-y)\frac{d}{dw}\log(1-\hat{y})$$

$$\Rightarrow \frac{(1-y)}{(1-\hat{y})} \cdot \frac{d}{dw}[1-\hat{y}]$$

$$\Rightarrow -\frac{(1-y)}{(1-\hat{y})} \frac{d}{dw} \sigma(wx)$$

$$\Rightarrow -\frac{(1-y)}{(1-\hat{y})} [\sigma(wx)[1-\sigma(wx)]] \frac{d}{dL}(wx)$$

$$\Rightarrow -\frac{(1-y)}{(1-y)} \hat{y}(1-\hat{y}) x$$

$$\Rightarrow \boxed{-\hat{y}(1-y) x}$$

$$\frac{dL}{dw} = -\frac{1}{m}\left[y(1-\hat{y})x - \hat{y}(1-y)x \right]$$

$$= -\frac{1}{m}\left[y(1-\hat{y}) - \hat{y}(1-y)\right]x$$

$$= -\frac{1}{m}\left[y - y\hat{y} - \hat{y} + y\hat{y}\right]x$$

$$\frac{\Delta L}{\Delta W} = -\frac{1}{m}(y-\hat{y})x$$

according to gradient descent-

$$W = W + \eta\frac{1}{m}(y-\hat{y})X \quad //$$

where,

$$W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}_{(n+1),1} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & & & & \\ 1 & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}_{m,n+1} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{(m,1)}$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix}_{(m,1)}$$

$$W = W + \frac{\eta}{m}(y-\hat{y})X$$

$(n+1,1)$ $(n+1,1)$ $(m,1)$ $m,(n+1)$

$(1, m+1)$

Accuracy and Confusion Matrix

CLASSIFICATION METRICS

Accuracy

$$\text{Accuracy} = \frac{\text{No. of Correct Prediction}}{\text{Total No. of Predictions}}$$

implement

```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_Pred)
```

- Problem with Accuracy.
  Accuracy neve tells the : Precision & Recall Trade-off.
- Imbalanced datasets - accuracy is misleading with imbalance dataset.
- Equal weightage for Errors. - accuracy treats all errors equall like False negative & False Positive are not differentiable in accuracy score.

Confusion Matrix

Actual \ Prediction	1	0
1	True Positive	False Negative
0	False Positive	True Negative

A \ P	1	0
1	TP	FN
0	FP	TN

1 = Yes
0 = No.

TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative

- accuracy from confusion matrix

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Type I error

FP is also know as Type I error.

Type II error

FN is know as Type II error.

implement:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_pred)
```

- **Precision**

Measures the proportion of true positive prediction among all positive

- Proportion of predicted positive
- Not More Harmful than Recall

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

- **Recall**

Measures the proportion of correctly predicted positive observation to all observations in actual class.

Cost of FN is high

$$\text{Recall} = \frac{TP}{(TP + FN)}$$

F1 score

- It is used to penalize the lower value. It calculates the Harmonic mean for predictions.

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

* Multiclass Classification *

Here we calculate all classes pricision.

• Precision

(1) Macro precision : for balanced classes take avg of all classes precision

(2) Weighted Precision : for unbalanced / weighted baised classes.

all precision values are multiplied by their class weight

$$Class\ w.t. = \frac{each\ class\ actual\ value}{total\ items}$$

for 3 precisions.

$$macro = \frac{P_1 + P_2 + P_3}{3}$$

$$Weighted = Class\ w.t_1 \times P_1 + Class\ wt_2 \times P_2 + Class\ wt_3 \times P_3$$

• Recall

$$= \frac{Total.\ Predicted\ class\ value}{Total\ actual\ class\ value}$$

- macro
- weighted

F1-Score

F1-score is similar for all like we have to calculate F1-score individually for each class.

Eg. Precision = P_1, P_2, P_3

Recall = R_1, R_2, R_3.

$$F1\text{-}score_1 = \frac{2 P_1 \times R_1}{P_1 + R_1}$$

$$F1\text{-}score_2 = \frac{2 P_2 \times R_2}{P_2 + R_2}$$

$$F1\text{-}score_3 = \frac{2 P_3 \times R_3}{P_3 + R_3}$$

implementation using sklearn-

- Macro.

```
from sklearn.metrics import precision_score,
                            recall_score,
                            f1_score.
precision_score(y_test, y_pred, average='macro')
```

same for Recall & f1-score.

- Weighted

```
Recall_score(y_test, y_Pred), average='weighted')
```

same for Precision_score & f1-score.

- All in one

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

Softmax Regression

Softmax regression (or multinominal logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes.
Softmax regression allows us to handle $y(i) \in \{1, \ldots, K\}$ where K is the number of classes.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where, K is class $K = \{1, \ldots, K\}$

e.g. softmax fun apply for 1st class of 3 classes

$$\sigma(z)_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$0 < \sigma < 1$

(classes = 0, 1, 2 etc.)
Yes, No, opt etc.

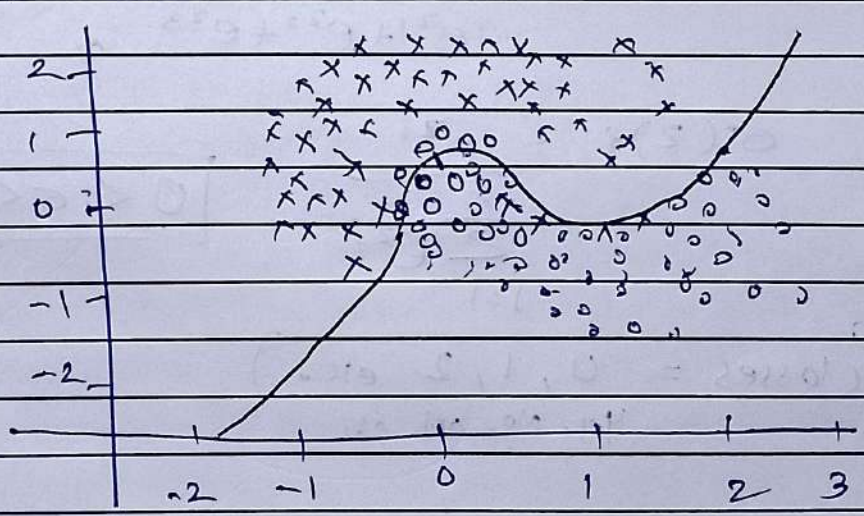
implementation

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(multi_class='multinomial')
clf.fit(X_train, y_train)
clf.predict(X_test)
```

Polynomial Logistic Regression

Polynomial features are created by transforming the original input features into a new set of feature that include original features & their polynomial combination up to specific degree. The ~~allows~~ transformation allows logistic regression to capture non-linear relationships between one input variables & the target variable

Logistic regression with polynomial features is technique used for handling non-linear relationships in data. The idea is to transform input features into higher-degree polynomial, to capture more complex relationship between variables.



DECISION TREE

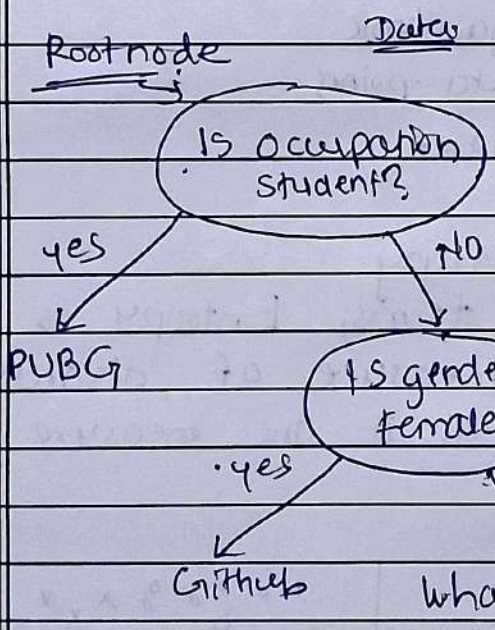
SARASWATI Education Society's
SARASWATI College of Engineering

PAGE NO.: 93
DATE: 11/7/2024

A decision tree is a decision support hierarchical model that uses a tree-like model of decision and their possible consequences, chance event outcomes, resource costs & utility. It is one way to display an algorithm that only contains conditional control statements.

Example

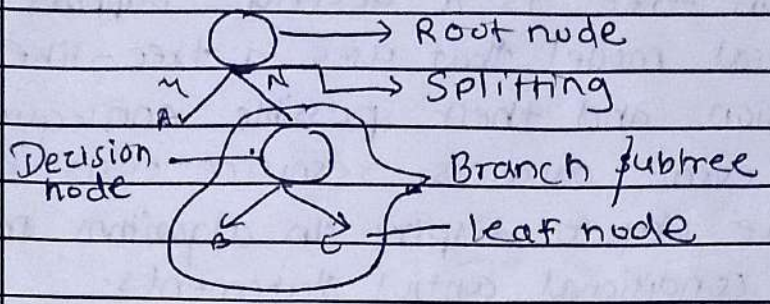
Gender	Occupation	Suggestion
F	Student	PUBG
F	Programmer	Github
M	Programmer	Whatsapp
F	Programmer	Github
M	Student	PUBG
M	Student	PUBG




```
if occupation == student
    print(PUBG)
else
    if gender == female
        print(Github)
    else
        print(Whatsapp)
```

Algo.

Terminology



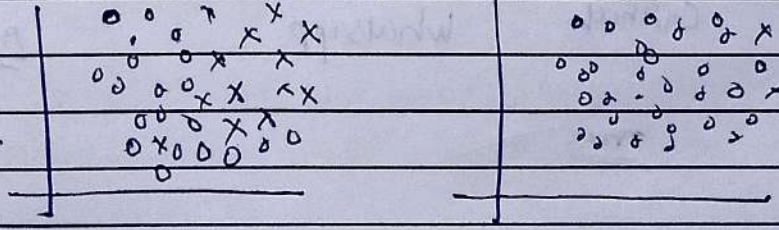
Advantages	Disadvantages
- Intuitive & easy to understand	Overfitting
- minimal dat preperation is required	prone to errors for imbalanced datasets

The cost of using the tree for inference is logarithmic in the number of data points used to train the tree

Decision Trees Entropy

- In the most easy terms, Entropy is nothing but the measure of disorder. Or you can also call it the measure of purity / impurity.

e.g

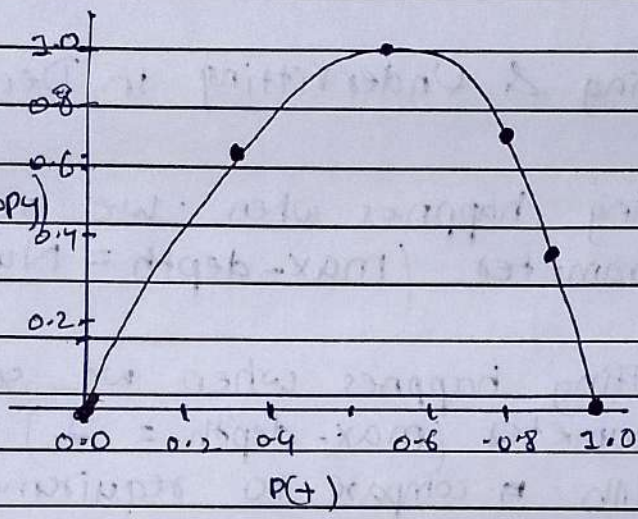


High entropy | Low entropy

$$E(S) = \sum_{i=1} -P_i \log_2 P_i$$

where, Pi frequentist probability of an element (class 'i' in our data

$$E(D) = -P_{yes} \log_2 (P_{yes}) - P_{no} \log_2 (P_{no})$$



* Information Gain *

Information Gain, is a metric used to train Decision Trees. Specifically; this metric measures the quality of a split.

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain.

Info Gain = E(Parent) - {Weighted Average} * E{Children}

Weighted Entropy = P * E

* Gini Impurity *

$$G.I = 1 - (P_Y^2 + P_N^2)$$

* Overfitting & Underfitting in Decision Trees.

Overfitting happnes when we set the hyperparameter (max-depth = None) or high

Underfitting happnes when we set the hyperparameter (max-depth = 1) or very less with r. compare to requirments.

* Regression Trees *

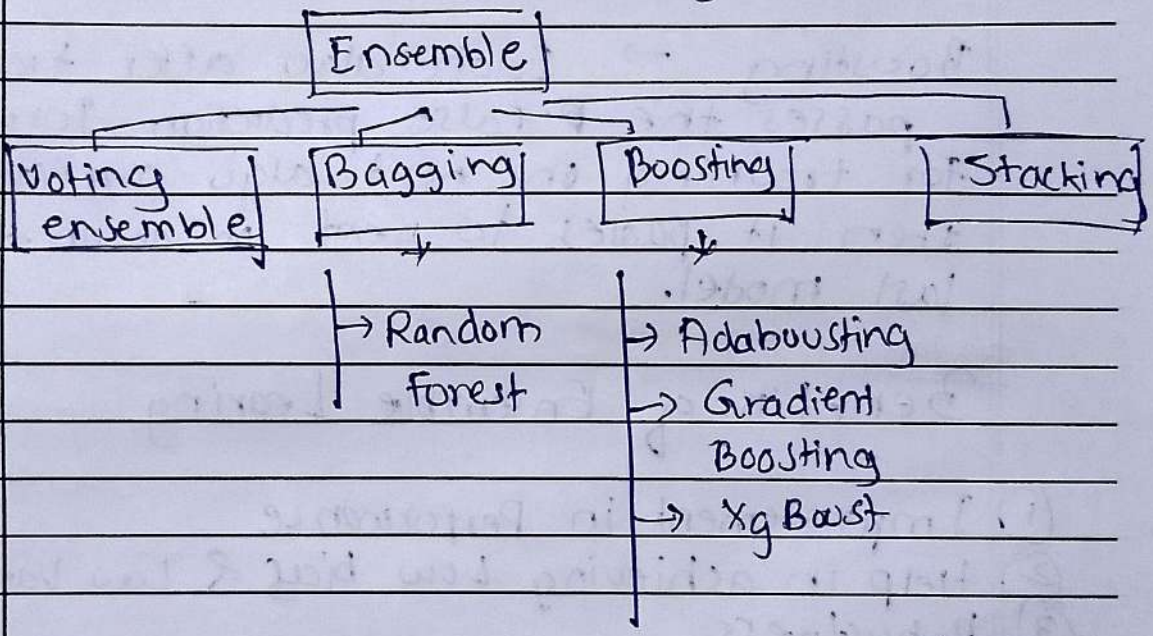
Used when non-linear Relationship in dataset is present

For Decision Tree visualization
Use : library called dtreeviz.
install
pip install dtreeviz
import
from dtreeviz.trees import *

Ensemble methods use mutiple learnning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algo.

Ensemble learning trains two or more ml algorithms to a specific classification or regression task.

Type of Ensemble Learning



1. Voting Ensemble
   - Base models are different Algo's and same Data.

2. Stacking
   - Same as Voting but additionally we give weights to every models output. and another last model is trained with results + additional weights of

remaining models.

Bagging → Bootstrapped Aggregation.
- Based models are same but the data is different for each model.

Random Forest → it is special case of bagging.

Boosting → Each algo after training passes the false prediction forward for training the next algo correctly then it pasies to next on an on.... till last model.

Benefits of Ensemble Learing

1. Improvement in Performance
2. Help in achieving Low bias & Low Variance.
3. Robustness

1] Voting Ensemble

Ensemble voting is a popular technique in ensemble learning where the final prediction is made by combining the predictions of multiple individual models.
It can be used for classification & regression problems.

Voting Classifier

The concept is to build a single model that learns from various models and predicts output based on their aggregate majority ~~and predicts output based on their~~ of votes for each output class, rather than building separate specialized models & determining the accuracy for each of them.

2 differen types of voting classifier

- Hard voting: The predicted output class is a class with the highest majority of votes.

- Soft voting: The average probabilities of the classes determine which one will be the final prediction.

To import voting.

```
>> from sklearn.ensemble import VotingClassifier
# Hard voting-
>> vc = VotingClassifier(estimators = [('lr', LogisticRegression()), ('rf', RandomForestClassifier()),
                                       ('knn', KNeighborsClassifier())],
                         voting = "hard")
# Soft voting
vc = VotingClassifier(estimators = estimators,
                      voting = "soft")
```

Voting regressor

It combines or 'ensembles' multiple regression models & overperforms the individual models present as its estimators. ML models as $m_1, m_2, m_3 \ldots m_n$ each model will produce a prediction P_n for given data I. If we pass throug Voting Regressor then final prediction will be P_{voting} is Average of all predictions.

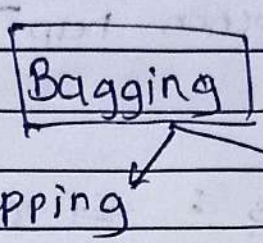
$P_{voting} = \frac{1}{x} \sum_{n=1}^{x} P_n$ ⇒ Simple avg

$P_{voting} = \sum_{n=1}^{x} (wt_n \cdot P_n)$ ⇒ weighted avg where wt_n is custom w.t.

implementation

```
>> from sklearn.ensemble import VotingRegressor
>> vr = VotingRegressor(estimators)
```

Bagging



Bagging also known as bootstrap aggregation, is commony used to reduce variance within a nosiy data set.

In bagging, a random sample of data in a training set is selected with replacement. After generating several data samples, these weak models are then trained independently. Depending on the type of task - regression or classification,

There are3 different ways for performing Bagging

1. Pasting
2. Random Subspace
3. Random Patches

1. **Pasting** : It is sampling without replacement. each subset is created by randomly sampling the training data without duplicating instances.

Use case: Whe dataset is large enough, & risk of overfitting due to duplicated instances in bootstrapped samples is a concern.

2. **Random Subspace** : It involves training each base model on a random subset of the features rather than the instances.

Use case - dealing with high dimensional data where feature selection help in improving model performance.

3. Random Patches : It combine both the random subspace & pasting methods. It involves training each base model on a random subset of both instance & features.
Use case - beneficial when dataset is large & high-dimensional.

Bagging Classifier

Is an ensemble meta-estimator that fits base classifier each on random subsets of original dataset & then aggregate their individual predictions to form final prediction.

class sklearn.ensemble.BaggingClassifier(estimator=None, n_estimator=10, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)

Implementation of code

```
>> from sklearn.ensemble import BaggingClassifier
>> bag = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimator=500, max_samples=0.5, bootstrap=True, random_state=42)
>> bag.fit(X_train, y_train)
>> bag.predict(X_test)
```

Bagging Regressor

A bagging regressor is an ensemble estimator that fits base regressors each on random subset of the original dataset & then aggregate their individual predictions (either by voting or averaging) to form a final prediction.

```
class sklearn.ensemble.BaggingRegressor(estimator=None,
    n_estimator=10, max_samples=1.0,
    max_features=1.0, bootstrap=True,
    bootstrap_features=False, oob_score=False,
    warm_start=False, n_jobs=None,
    random_state=None, verbos=0)
```

```
# implementation of code
>> from sklearn.ensemble import BaggingRegressor
>> bag = BaggingRegressor(random_state=1)
>> bag.fit(X_train, y_train)
>> bag.predict(X_test)
```

RANDOM FOREST

Random forests or random decision forests is an ensemble method for classification, regression and other task that operates by constructing a multitude of decision trees at training time.
For classification tasks, the ~~mean~~ output of the random forest is the class selected by most trees.
For regression tasks, the mean or average prediction of the individual trees is returned

Random forests converts the ~~High bias - High variance HBHV model~~ LowBias High Variance. algorithm into LowBias Low variance. algorithm LBHV converts LBLV

Difference Between: Bagging & Random Forest

Bagging -

① In Bagging we can use multiple algorithms (DecisionTree, SVM, KNN, etc.) it is generalize method.

② In Bagging Feature / column sampling the No. of columns choosed in starting are always fixed for all trees creation ③ Column Sampling is at tree level

Columns not shuffle they are fixed.

④ eg if 2 colums are selected Col 3, & Col 5 So in all model it will be fixed.

Random Forest

(1) In Random Forest we only use Decision Tree model / Algorithm It is focused method.
(2) Column Sampling at node level
(3) Column / feature Sampling the no. of columns choose in starting is fixed and the internal columns are shuffle in each node.
eg. if 2 colums are selected col2, & col4 after 1 node again randomly colums will shuffle now it will choose col3, col1 for next not it will go on & on..

- Random Forest Classifier.

```
class sklearn.ensemble.RandomForestClassifier (n_estimators=10
criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, max_features='sqrt', max_leaf_
nodes=None, min_impurity_decrease=0.0, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None,
verbose=0, warm_start=False, ccp_alpha=0.0,
class_weight=None, max_samples=None,
monotonic_cst=None)
```

implementation in code

```
>> from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier (n_estimator=500,
        random_state=42, max_features=2)
rf.fit (X_train, y_train)
rf.predict (X_test)
```

RandomForestRegressor

is a meta estimator that fits a no. of decision tree regressors on various sub-samples of the dataset & uses averaging to improve the predictive accuracy

syntax.

class sklearn.ensemble.RandomForestRegressor (n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)

implementation using sklearn..

```
>> from sklearn.ensemble import RandomForestRegressor
>> rfr = RandomForestRegressor(n_estimators=1000)
rfr.fit(x_train, y_train)
rfr.predict(x_test)
```

* Out of Bag Evaluation OOB_Score

Random forests do not require validation dataset. Most random forests use a technique called out-of-bag-evaluation (OOB evaluation) to evaluate the quality of the model. OOB evaluate treats the training set as if it were on the test set of a cross-validation.
over ~33% of training examples are missed by decision tree while choosing data with replacement.

implementation:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(oob_score=True)
rfc.fit(x_train, y_train)
~~rfc.predic~~
rfc.oob_score_
```

* Feature Importance using Random Forest & Decision Trees

It also known as variables or attributes, are individual measurable properties or characteristics of the phenomena being observed.
Formula for Decision trees feature importance.

$$f_{i_k} = \frac{\sum_{j \in \text{node split on feature } k} n_i}{\sum_{j \in \text{all nodes}} n_i}$$

$$n_i = \frac{N\_t}{N}\left[\text{impurity} - \left(\frac{N\_t\_r}{N\_t} * \text{right-impurity}\right) - \left(\frac{N\_t\_l}{N\_t} * \text{left-impurity}\right)\right]$$

where, n_i = node importance
N_t = No. of rows to Node
N = Total no. of rows present in data
impurity = 'gini'
N_t_r = Node to right
N_t_l = Node to left.
right_impurity = impurity on right node
left_impurity = impurity on left node

$$f_{ik} = \frac{\sum_{j \in \text{nod split on feature } k} n_i}{\sum_{j \in \text{all nodes}} n_i}$$

implementation of Random forest feature importance

```
>> from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X,y)
rf.feature_importances_
```

```
>> from sklearn.ensemple import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X,y)
clf.feature_importance
```

Implementation of Decision Tree importance

ADABOOST

AdaBoost Adaptive Boosting is used in conjunction with many other algos to improve performance The output of the other learning algorithms ('Weak learners') is combines into a weighted sum that represents the final output of the boosted classifier.

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1 - error}{error}\right)$$

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1 - E_m}{E_m}\right)$$

where E_m = weighted error rate of the weak classifier.

α_m = weight of model m.

$$E_m = \frac{\sum_{y_i \neq k_m} (x_i) w_i^{(m)}}{\sum_{i=1}^{N} w_i^{(m)}}$$

For misclassified
new_wt = curr_wt * e^{2}

For correctly classified
new_wt = curr_wt * e^{-2}

AdaBoost Classifier

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original data set, ~~& then fits~~.

Syntax.

```
class sklearn.ensemble.AdaBoostClassifier(estimator=
        None, *, n_estimators = 50, learning_rate = 1.0,
        algorithm = 'SAMME.R', random_state = None)
```

Implementation in sklearn.

```
>> from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier()
abc.fit(X, y)
abc.predict(x_test)
```

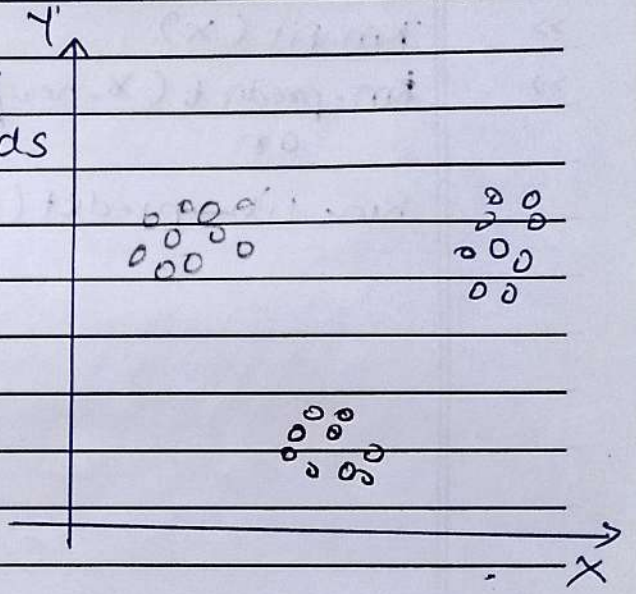
	Bagging	Vs	Boosting
(1)	Types of model used LBHV model used		HBLV model used
(2)	full grown decision tree		shallow grown decision treee
(3)	Parallel training		Sequential (one-by-one)
(4)	equally weightage of base learners		biased or unequal weightage of base learners.

K-means clustering

Is a unsupervised learning method of vector quantization, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or centroid), serving as a prototype of the cluster.

Steps for achieving K-means cluster.

1) Decide n clusters
2) Initialize n centroids
3) Assign cluster
4) Move Centroid
5) Finish



To find ϕ or Decide number of clusters we follow (Elbow Method)

= WCSS => within clusters Sum of squared distance

$WCSS = \sum_{i=1}^{n} d_i^2$,

A the point whe WCSS got stable we choose no. of cluster in fig a. 4 clusters is choosen

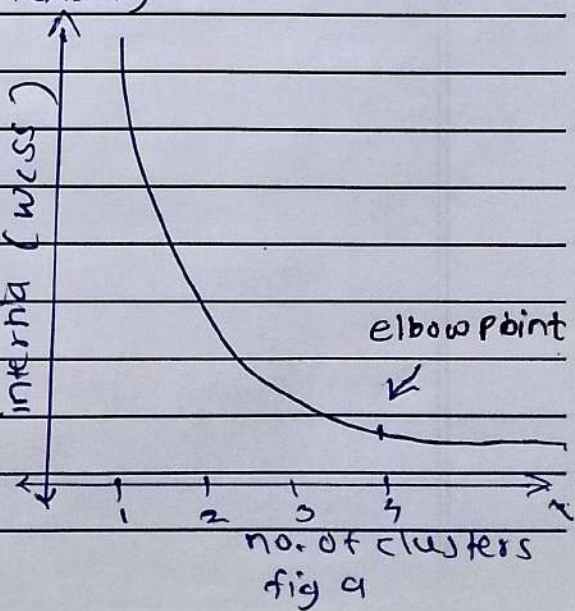


fig a

Syntax:

```
class sklearn.cluster.KMeans(n_clusters = 8, *,
    init = 'k-means++', n_init = 'auto', max_iter = 300,
    tol = 0.0001, verbose = 0, random_state = None,
    copy_x = True, algorithm = 'lloyd')
```

implementation in sklearn:

```
>> from sklearn.cluster import KMeans
>> km = KMeans(n_clusters = 4)
>> km.fit(X)
>> km.predict(X_new/X)
```

or

```
km.fit_predict(X)
```

Gradient Boosting

Gradient boosting is a ML technique based on boosting in a functional space, where the target is pseudo-residuals rather than the typical residuals used in traditional boosting. It gives a prediction model in the form of an ensemble of weak prediction models.
Gradient boosting is used with decision trees of a fixed size as base learners.

Algorithm

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a diff. loss function $L(y, F(x))$, no of iterations M.

1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$.

2. For $m = 1$ to M:

(a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

(b) Fit a regression tree to the targets r_{im} giving terminial regions R_{jm}, $j = 1, 2, \ldots J_m$.

(c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

(d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

3. Output $\hat{f}(x) = f_M(x)$.

The Loss function that we have used in one abov algo is Least square.

$$L = \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

(MSE)

Syntax. for Regressor.

class sklearn.ensemble.GradientBoostingRegressor(
*, loss='squared_error', learning_rate=0.1,
n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3,
min_impurity_decrease=0.0, init=None, random_state=None,
max_features=None, alpha=0.9, verbose=0,
max_leaf_nodes=None, warm_start=False,
validation_fraction=0.1, n_iter_no_change=None,
tol=0.0001, ccp_alpha=0.0)

implementation in sklearn.

```
from sklearn.ensemble import GradientBoostingRegressor
reg = GradientBoostingRegressor(random_state=0)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

Syntax for classifier.

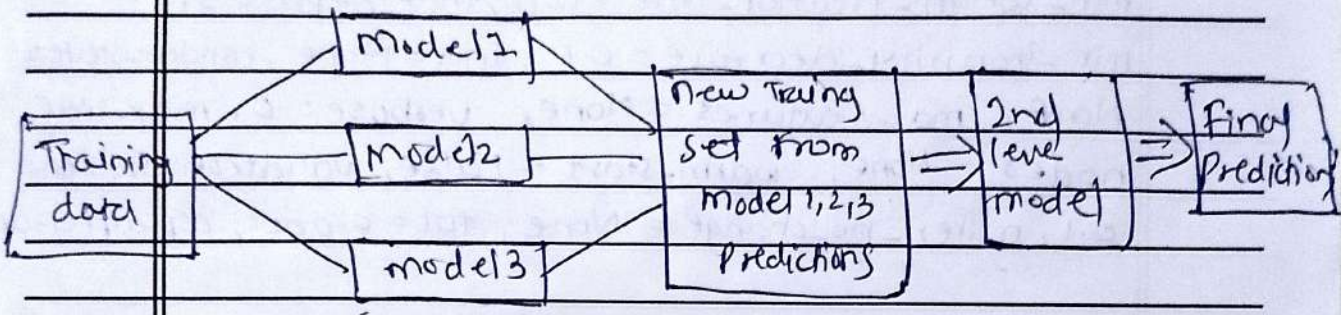
class sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_sample_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)

implementation

```
From sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimator=100, learning_rate=1.0, max_depth=1, random_state=0)
clf.fit(x_train, y_train)
clf.predict(x_test)
```

STACKING

Stacking is a way to ensemble multiple classifications or regression model. There The point of stacking is to explore a space of different models for the same problem.

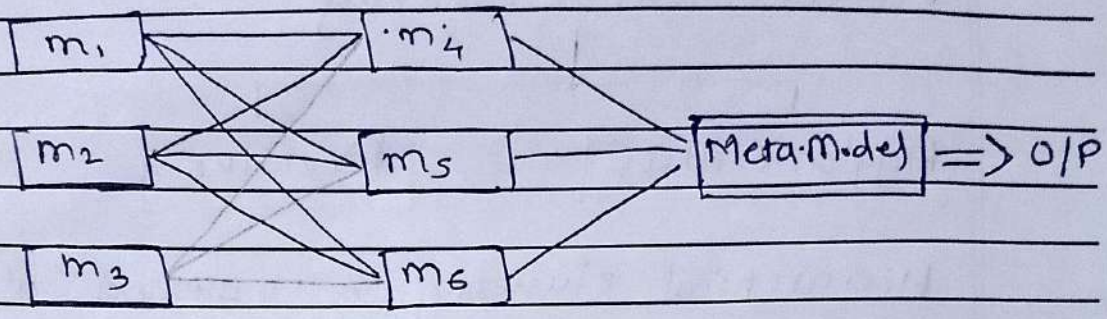


There are two ~~or~~ three methods for stacking.

(1) Stacked Model : An ensemble of models in which the meta-model is trained on out-of-fold predictions made by the base models during k-fold cross validation.

(2) Blended Model : An ensemble of models in which the meta-model is trained on predictions made by the base models on a holdout dataset (e.g. the validation dataset).

(3) Multi Layer Stacking : This create a hierarchical structure of models where each layer's models are trained on predictions of the previous layer.



Syntax

class sklearn.ensemble.StackingClassifier (estimators, final_estimator = None, *, cv = None, stack_method = 'auto', n_jobs = None, passthrough = False, verbose = 0)

implementation using sklearn

```
from sklearn.ensemble import StackingClassifier
estimators = [ ('rf', RandomForestClassifier (n_estimators = 10, random_sate = 42)),
               ('knn', kNeighborsClassifier (n_neighbors = 10)),
               ('gbdt', GradientBoostingClassifier())]
From sklean.linear_model import LogisticRegression
clf = StackingClassifier( estimators = estimators,
      final_estimator = LogisticRegression(),
      cv = 10)

clf.fit ( X_train , y_train)
clf.predict ( X_test)
```

Hierarchical Clustering

- Agglomerative
- Divisive

Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. Strategies are of 2 ways:

- Agglomerative Clustering: This is a "bottom-up" approach: Each obs. starts in its own cluster, & pairs of clusters are merged as one moves up the hierarchy.

- Divisive Clustering: This is "top-down" approach: All observations start in one cluster, & splits are performed recursively as one moves down the hierarchy.

The merges & splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a dendrogram dendogram.

Algorithm:

1. Initialize the Proximity Matrix
2. Make each point a cluster
3. Inside a loop
   a. Merge the 2 closest clusters
   b. Update the Proximity Matrix
4. Until only one cluster is left.

Types of Agglomerative Clustering

1. Min (single-link) => 'single'
'single' uses the minimum of distances bet^n all observations of the two sets.

2. Max (complete link) => 'complete' or 'maximum'
uses the maximum distance bet^n all observations of the two sets.

3. Average => 'average'
uses the average of the distance of each obs of the two sets.

4. 'ward'
minimizes the variance of clusters being merge

Syntax

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2, *, metric='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False)
```

implementation

```
from sklearn.cluster import AgglomerativeClustering
agc = AgglomerativeClustering()
agc.fit(X)
agc.labels_
```

K-nearest neighbors algorithm (K-NN) is a non-parametric supervised learning method. it is used for classification & regression

K-nn algorithm assumes the similarity between the new case/data. & available cases & put the new case into the category that is most-similar to the available categories.

K-nn alg. stores all the available data & classifies a new data point based on the similarity means when new data appears then it can be easily classified into a well suite category by using K-NN algorithm.

* How to select ~~data~~ K.

K

heuristic — $\sqrt{n}$

experimental — loop for all cases.

n = drow_count.

* Overfiting & Underfitting.

if K = 1 or very low then => Overfitting

if K = n or near to n then => Underfitting

n = no of rows.

* Limitations

1) Large dataset => Lazy learning technique
2) High dim data => Curse of dimensionality.
3) Outliers

4) Non-homogeneous Scales. (Feature scales)
5) Imbalanced dataset
6) Inference and not for prediction uses.

Syntax

```
class sklearn.neighbors.KNeighborsClassifier(
n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
```

Implementation

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
neigh.predict(x_test)
```

Assumptions of Linear Regression

1. Linear Relationship between input & output
2. No Multicollinearity
3. Normality of Residual
4. Homoscedasticity
5. No Autocorrelation of Errors

1 → Find linearity betn each feature with target column.

2 → All input features should independent in nature.
- calculate variance inflation factor for checking multicollinearity
- Calculate one co-relation betn columns. for checking multicollinearity

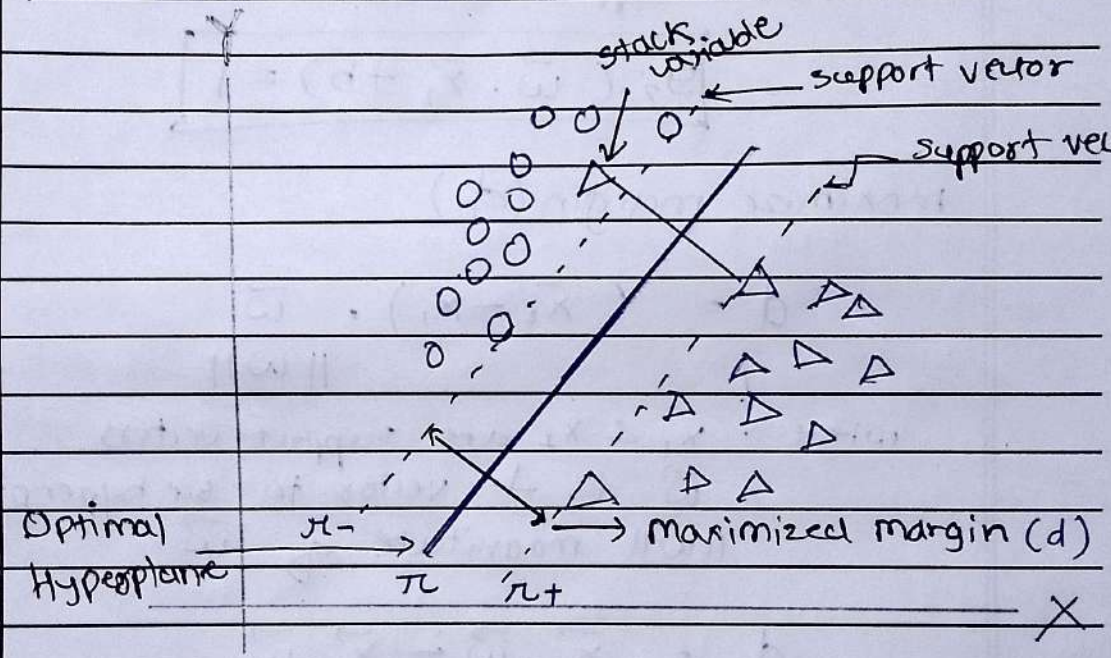
3 → Error should be normally distributed

4 → Having the same scatter the ~~feature~~ spread should uniform on residul values.

5 → Should NO positive auto-correlation of Residuals.

Support Vector machines (SVM) are supervised max-margin models with associated learning algorithms that analyze data for classification & regression analysis

SVM distinguish between two classes by finding the optimal hyperplane that maximizes the margin between the closest data points of opposite classes. SVM algorithm can handle both linear & nonlinear classification tasks.



Let's derive decision rule of SVM.

we know.

$$\vec{w} \cdot \vec{u} \geq c$$

$$\vec{w} \cdot \vec{u} - c \geq 0$$

$$\vec{w} \cdot \vec{u} + b \geq 0$$

for any x_i

$$\hat{y} = \begin{cases} +1 & \text{if } \vec{w} \cdot \vec{x_i} + b \geq 0 \\ -1 & \text{if } \vec{w} \cdot \vec{x_i} + b < 0 \end{cases}$$

$\hat{y}$ = Predicted value.

assume $w^T x + b = 1$ for +ve

$w^T x + b = -1$ for -ve

Optimization function

$$\begin{cases} \vec{w}\cdot\vec{x} + b \geq 1 \\ \vec{w}\cdot\vec{x} + b \leq -1 \end{cases} \quad \begin{cases} y_i(\vec{w}\cdot\vec{x}_2 + b) \geq 1 \\ y_i(\vec{w}\cdot\vec{x}_2 + b) \geq 1 \end{cases}$$

$$y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1$$

for support vectors.

$$\boxed{y_i(\vec{w}\cdot\vec{x}_i + b) = 1}$$

maximize margin (d)

$$d = (\bar{x}_2 - \bar{x}_1)\cdot\frac{\vec{w}}{\|w\|}$$

where · x_1 & x_2 are support vectors

$\vec{w}$ is ⊥ vector to hyperplane.

$\|w\|$ magnitude of $\vec{w}$.

$$d = \frac{\vec{x}_2\cdot\vec{w} - \vec{x}_1\cdot\vec{w}}{\|w\|} \quad \text{---- (3)}$$

$$y_i(\vec{w}\cdot\vec{x}_i + b) = 1$$

$$1(\vec{w}\cdot\vec{x}_2 + b) = 1$$

$$\vec{w}\cdot\vec{x}_2 = (1-b) \quad \text{---- (1)}$$

Similarly of x_1

$$-1(\vec{w}\cdot\vec{x}_1 + b) = 1$$

$$\vec{w}\cdot\vec{x}_2 = -b-1 \quad \text{---- (2)}$$

replace the value in eqn (3)

$$\frac{\vec{x}_2 \vec{w} - \vec{x}_1 \vec{w}}{\|w\|} = \frac{(-b) - (-b-1)}{\|w\|}$$

$$= \frac{1 - b + b + 1}{\|w\|}$$

$$\boxed{d = \frac{2}{\|w\|}} \quad \ldots (3)$$

to maximize ore eqn (3).

$$\boxed{\underset{(w^*, b^*)}{\text{argmax}} \frac{2}{\|w\|} \quad \text{such that } y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1}$$

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$$

Hard Margin optimization function.

Hard Margin SVM: is a type of SVM that tries to find a linear separator (hyperplane) that perfectly separates the data points of different classes without any errors

Constraint: requires data to be perfectly separable.

Soft Margin SVM: It allows some data points to be misclassified or to be lie within the margin. it introduces a penalty for misclassified points to handle cases where data is not perfectly ~~sp~~ separable.

Constraint: It uses a regularization parameter (C) to control the trade-off between

achieving a larger margin and having a lower error rate. A smaller C allows more misclassifications, while a larger C tries to classify all training examples correctly.

The optimize function for soft margin is

$$\max f(x) \Leftrightarrow \min \frac{1}{f(x)}$$

$$\underset{(w^*, b^*)}{\operatorname{argmin}} \quad \frac{\|w\|}{2} + C \sum_{i=1}^{n} \xi_i$$

subject to:

$$y_i (w \cdot x_i + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots N$$

$$\xi_i \geq 0 \quad \text{for all } i = 1, 2, \ldots N.$$

where,

w is weight vector

b is bias term

(zeta) $\xi_i (x_i)$ are slack variables that allow some misclassification

C is regularization parameter controls trade of bet^n max one. margin & min one misclassification error.

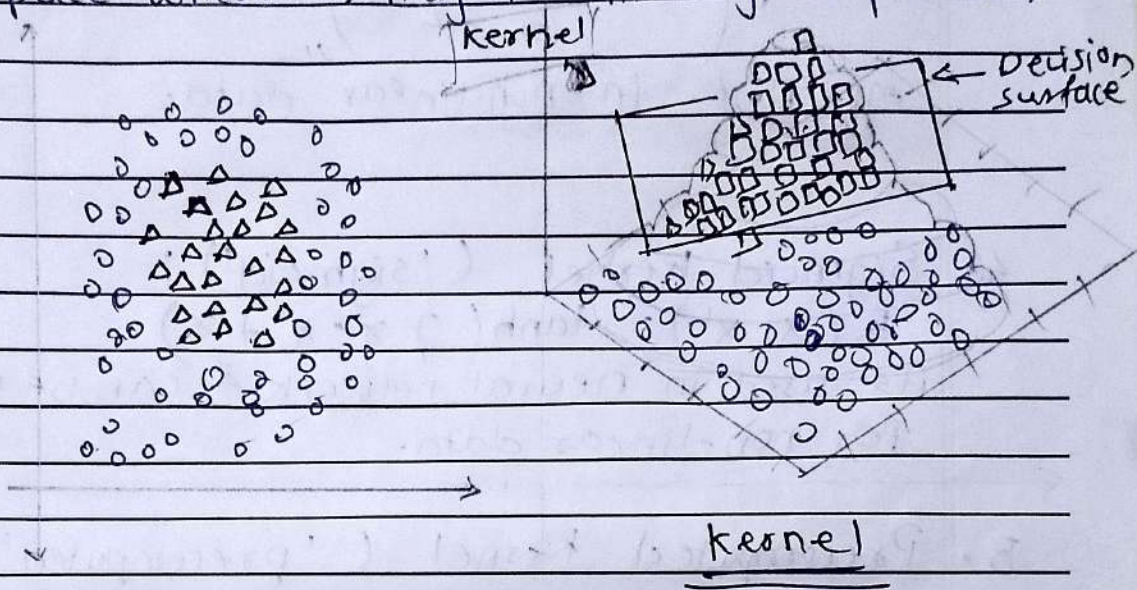
y_i is class label of i^{th} e.g.

x_i is feature vector of i^{th} traing eg.

N is no of training. e.g.

Kernel Trick in SVM

The "Kernel Trick" is a method used in SVMs to convert data (that is no linearly separable) into a higher-dimensional feature space where it may be linearly separated.



kernel

different types of kernels used in SVM.

1. Linear kernel ('linear')
   - It is dot product of two vectors used for linearly separable data. $K(X, X') = X \cdot X'$

2. Polynomial kernel ('poly'):
   - is a non-linear kernel that represents the similarity of vectors in feature space over polynomial of original variables $K(X, X') = (\gamma X \cdot X' + r)^d$
   γ is scaling parameter, r is constant, d is degree of polynomial.
   γ is scaling parameter
   it is useful for scenarios where the relationship bet^n class labels & attributes is non-linear.

3. Radial Basis Function (RBF) kernel ('rbf'):
- formula $K(x, x') = \exp(-\gamma \|x - x'\|^2)$
where $\gamma = \frac{1}{2\sigma^2}$

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

effective in non-linear data.

4. Sigmoid kernel ('sigmoid'):
$K(x, x') = \tanh(\gamma x \cdot x' + r)$
its used in neural network & can be used for non-linear data.

5. Precomputed kernel ('precomputed'):
- requires manully compute the kernel matrix before training svm.
- kernel matrix should be an $n \times n$ matrix where n is no. of samples.

6. Callable kernel
custom kernel function as a callable fun should take arguments X & Y. & return kernel matrix.
- eg. def custom_kernel(X, Y):
return np.dot(X, Y.T)

SVM syntax & parameters.

class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbos=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)

implementation.

```
>> from sklearn.svm import SVC
>> model = SVC(kernel='linear', C=1)
>> model.fit(X,y)  # x_train, y_train
>> model.predict(X_test)
```

NAIVE BAYES CLASSIFIER

Condition Probability

A, B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \{P(B) \neq 0\}$$

Independent Events	Mutually Exclusive Event
A, B	A, B
$P(A \cap B) = P(A) * P(B)$	$P(A \cap B) = 0$
$P(A|B) = \frac{n(A \cap B)}{n(B)}$	$P(A|B) = \frac{P(A \cap B)}{P(B)}$
$P(A) = \frac{n(A \cap B)}{n(B)}$	$P(A|B) = 0$
$P(A|B) = P(A)$	

Bayes Theorem

we know, Conditional probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad \text{---- (i)}$$

$A \cap B = B \cap A$

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

$$P(A \cap B) = P(B|A) * P(A) \quad \text{---- (ii)}$$

likelihood

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

$P(A)$ => prior

$P(B)$ => Evidence

Postierior

given $\{P(B) \neq 0\}$

Naive Bayes Classifier

naive Bayes classifiers are a family of linear "probabilistic classifiers" which assumes that the features are conditionall independent, given the target class.

Naive bayes is a conditional probability model it assigns probabilities $p(C_k | x_1, \ldots, x_n)$ for each of the K possible outcomes or classes C_k given a problem instance to be classified, represented by a vector $x = (x_1, \ldots, x_n)$ encoding n features.

$$P(C_k|x) = \frac{P(C_k)\, P(x|C_k)}{P(x)}$$

$$\text{Posterior} = \frac{\text{Prior} \times \text{likelihood}}{\text{evidence}}$$ //

Types of Naive Bayes classifiers

- Gaussian Naive Bayes (GaussianNB): This used with Gaussian distributions - i.e normal distributions - & continuous variables. This model is fitted by finding the mean and standard deviation of each class

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$ //

- Multinominal Naive Bayes (MultinominalNB): This type assumes that features are from multinomial distributions. This variant is useful when using discrete data, such as frequency

counts, & it is typically applied within NLP.

- Bernoulli Naive Bayes (BernoulliNB) : This classifier is used with Boolean variables. That is, variables with two values, i.e. True & False or 1 & 0.

implementation.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
gnb.predict(X_test)
```

XGBOOST

SARASWATI Education Society's
SARASWATI College of Engineering

PAGE NO.: 133
DATE: 22/7/2024

XGBoost (eXtreme Gradient Boosting) is an open-source software library which provide a regularizing gradient boosting framework for C++, Java, Python, R, Julia, Perl, Scala. It works on Linux, Microsoft Windows & macOS. It runs on a single machine, as well as the distributed processing frameworks Apache Hadoop, Apache Spark, Apache Flink & Dask.

XGBoost initially started by Tiangi Chen as a part of Distributed ML community group (DMLC).
Developed by Tiangi Chen from University of Washington.

Features.

Performance | Speed | Flexibility

1. Flexibility

- Cross Platform support
- Multiple Language Support
- Integration with other libraries & tools
- Support all kinds of ML problems.

2. Speed

- Uses parallel Processing in tree building stage
- Uses Optimized Data Structures
- Cache Awareness
- Out of Core Computing - sequential
- Distributed Computing - parallel
- GPU Support

3. Performance

- Regularized Learning Objective
- Handling Missing Values
- Sparsity Aware Split Finding
- Efficient Split Finding
  (Weighted Quantile Sketch + Approximate Tree Learning)
  (Convert Continous var into descret) uses histogram based training using weighted quantile sketch.

- Tree Pruning

Comparing With Other Boosting Algorithms

XGBoost vs. AdaBoost
AdaBoost Emphasizes incorrect predictions using weights. Correct predictions reduce weight; incorrect prediction increase weight.
less effective than XGBoost in accuracy & speed.

XGBoost Vs. CatBoost

CatBoost is another gradient boosting framework. It specializes in handling categorical features without preprocessing. perform well out-of-the-box with less hyperparameter tuning. Like XGBoost, CatBoost has built in support for handing missing data. CatBoost especially useful for datasets with many categorical features.

XGBoost vs. LightGBM

LightGBM (Light Gradient Boosting Machine) LightGBM was devdoped by microsoft in 2016 Most decison tree learning algorithm grow trees depth-wise. it uses leaf-wise tree growth strategy. Like XGBoost, LightGBM exhibits fast model training speed & accuracy & performance well with large datasets.

XGBoost for Regression

For split finding in Regression 2 algorithms are used.

1. Exact Greedy Algorithm for split finding
2. ~~Split finding~~
3. Approximate Algorithm for split finding

XGBoost Objective function contains loss function & regularization term. it tells diff. bet^n actual value & predicted values.
XGBoost expects to have base learners which are uniformly bad at the remainder so that when all the predictions are combined, bad predictions cancels out & better one sums up to form final good predictions.

For building XGBoost tree for Regression.

step 1: Calculate the similarity scores. it helps in growing the tree.

$$\text{Similarity Score} = \frac{(\text{sum of residuals})^2}{\text{No. of residuals} + \lambda}$$

step 2: Calculate the gain to determine how to split the data.

$$\text{Gain} = \text{Left tree (similarity score)} + \text{Right tree (similarity score)} - \text{Root (similarity score)}$$

$$\boxed{L_t + R_t - \text{Root}}$$

step 3 : Prune the tree by calculating the difference between Gain & gamma

Gain − gamma

if result is positive no. the do not prune if resut is negative then prune & again subtract gramma from the next Gain value way up the tree.

step 4 : Calculate output value for the remaining leaves

$$\text{Output value} = \frac{\text{Sum of residuals}}{\text{No. of residuals} + \lambda}$$

1st model is avg of all x_i (feature values) (base model)

$$\text{Similarity Score} = \frac{(\sum \text{Residuals})^2}{N + \lambda}$$ //

$$\text{Output Value} = \frac{\sum \text{Residuals}}{N + \lambda}$$ //

where N is no of residuals.

Loss Function / objective function.
Loss fun regularization para

$$L = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \Omega(f_2(x_i))$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

where, & T ⇒ no. of leaf node.
w ⇒ weight of leaf
γ, λ ⇒ regularization Parameter.

Derivation

Stage I

m_1 mean → $L = \sum_{i=1}^{n} L(y_i, \hat{y_i})$

$f_1(x_i)$

Stage II

m_1 mean | m_2 dtree

$L = \sum_{i=1}^{n} L(y_i, \hat{y_i}) + \Omega(f_2(x_i))$

$f_1(x_i)$ $f_2(x_i)$

$\hat{y_i} = f_1(x_i) + f_2(x_i)$

$L = \sum_{i=1}^{n} L(y_i, f_1(x_i) + f_2(x_i)) + \Omega(f_2(x_i))$

Stage III

m_1 mean | m_2 dt₁ | m_3 dt₂

$f_1(x_i)$ $f_2(x_i)$ $f_3(x_i)$

$$L = \sum_{i=1}^{n} (y_i, f_1(x_i) + f_2(x_i) + f_3(x_i)) + \Omega(f_3(x_i))$$

Stage t

m_1 mean | m_2 dt₁ | ..... | m_t dt_t

$f_1(x_i)$ $f_2(x_i)$ $f_t(x_i)$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} L(y_i, f_1(x_i) + f_2(x_i) + \dots + f_t(x_i)) + \Omega(f_t(x_i))$$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{<t-1>} + f_t(x_i)) + \Omega(f_t)$$

At any stage t

$$\mathcal{L}^{<t>} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{<t-1>} + f_t(x_i)) + \Omega(f_t(x_i))$$

we have to find ~~the~~ such output for $f_t(x_i)$ which will minimize the whole objective function

But our $\Omega(f_t(x_i))$ faces one ~~Taylor Series~~ (zig-zag curve) ∴ we need to approximate the $\sum_{i=1}^{n} L(y_i, \hat{y}_i^{<t-1>} + f_t(x_i))$

The solution is apply Taylor series is technique which do the approximation using polynomial.

Taylor series Equation

$$f(x) \simeq f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} \dots$$

$x \to \hat{y}_i^{<t-1>} + f_t(x_i)$, $a \to \hat{y}_i^{<t-1>}$

from derivating our simplified objective fuction is

$$\mathcal{L}^{<t>} \simeq \sum_{i=1}^{n} \left[L(y_i, \hat{y}_i^{<t-1>}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t(x_i))$$

Where

$$g_i = \frac{\partial L(y_i, \hat{y_i}^{<t-1>})}{\partial \hat{y_i}^{<t-1>}}$$

(gradient)

$$h_i = \frac{\partial^2 L(y_i, \hat{y_i}^{<t-1>})}{\partial \hat{y_i}^{<t-1>2}}$$

(hession)

We can remove the constant term.

$$L(y_i, \hat{y_i}^{<t-1>})$$

now. objective function is

$$\mathcal{L}^{<t>} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t(x_i))$$

Now, expand $\Omega(f_t(x_i))$ to $\Rightarrow$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

Now, replace.

$f_t(x_i)$ with $\rightarrow w_j$

w_j is weight.

$$\sum_{i=1}^{n}$$

and replace

$$\sum_{i=1}^{n} \rightarrow \sum_{j=1}^{T} \sum_{i \in I_j}$$

$a_1 + a_2 + \ldots + a_n =$

$a_3 + a_1 + a_2 + a_4 \ldots a_n$

So our objective function becomes.

$$I^{(t)} = \sum_{j=1}^{T} \left[\sum_{i \in I_j} g_i w_j + \frac{1}{2} \sum_{i \in I_j} h_i w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1} w_j^2$$

$$I^{(t)} = \sum_{j=1}^{T} \left[\sum g_i w_j + \frac{1}{2} \sum_{i \in I_j} h_i w_j + \frac{1}{2} \lambda w_j^2 \right] + \gamma T$$

$$L^{(t)} = \sum_{j=1}^{T} \left[\sum_{i \in I_j} g_i w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

differentiat. w.r.t. w_j

$$\frac{d I^{(t)}}{\partial w_j} = \sum_{i \in I_j} g_i + \left(\sum_{i \in I_j} h_i + \lambda \right) w_j = 0$$

$$w_j = - \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

→ gradient

↓

Heisian

we know

$$g_i = \frac{\partial L \left(y_i, \hat{y_i}^{(t-1)} \right)}{\partial \hat{y_i}^{(t-1)}}$$

diff. w.r.t. $\hat{y_i}$

$$\frac{\partial L}{\partial \hat{y_i}} = \sum_{i=1} \left(y_i - \hat{y_i} \right)$$

$$= \sum_{i=1}^{n} \left(\hat{y_i}^{(t-1)} - y_i \right)$$

and

$$h_i = \frac{\partial^2 L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)2}}$$

diff. w.r.t $\hat{y}_i^{(t)}$

$$\frac{\partial L}{\partial \hat{y}_i^{(t)}} = \sum_{i=1}^{n} (y_i - \hat{y}_i^{(t)}) \Rightarrow \sum_{i=1}^{n} (y_i^{(t)} - y_i^{(t)})$$

$$\Rightarrow \sum_{i=1}^{n} 1 \quad //$$

Now put the differentiated values in w_j. It becomes.

$$w_j = \frac{-\sum_{i \in I_j} g_i (y_i - \hat{y}_i^{(t-1)})}{\sum_{i \in I_j} 1 + \lambda}$$

$$\boxed{w_j = \frac{\sum_{i \in I_j} (y_i - \hat{y}_i^{(t-1)})}{\sum_{i \in I_j} 1 + \lambda}}$$

$$\boxed{w_j = \frac{\text{Sum of residuals}}{\text{No. of residuals} + \lambda}}$$

* XGBoost Output value of for Regression.

Output Value for Classification

$$w_j = \frac{\text{Sum of residual}}{\sum_{i \in J_j} P_i (1-P_i) + \lambda}$$

where P_i → Predicted probability of previous timestamp.

$$\boxed{w_j = \frac{\text{Sum of residual}}{\sum_{i \in I_j} P_i (1-P_i) + \lambda}}$$

where P_i → Predicted Probability of previous time-stamp.

* Similarity Score

Derivation

we have to find the minimum loss

$$L^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

$$L^{(t)}(q) = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) \frac{\left(-\sum_{i \in I_j} g_i \right)}{\sum_{i \in I_j} h_i + \lambda} + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\left(\sum_{i \in I_j} h_i + \lambda \right)^2} \right] + \gamma T$$

$$L^{(t)}(q) = \sum_{j=1}^{T} \left[-\frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \frac{1}{2} \cdot \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} \right] + \gamma T$$

$$= \sum_{j=1}^{T} \left[-\frac{1}{2} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} \right] + \gamma T$$

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

Similarity Score

Tree building using Similarity Score

Similarity score of Parent.

$$\mathcal{L}^{(t)}(q_p) = -\frac{1}{2} \frac{\left(\sum_{i \in I_p} g_i\right)^2}{\sum_{i \in I_p} h_i + \lambda} + \gamma$$

Similarity score of Child

$$\mathcal{L}^{(t)}(q_c) = -\frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} \right] + 2\gamma$$

When I_L for left node I_R for right node

$$\mathcal{L}^{(t)}(q_p) - \mathcal{L}^{(t)}(q_c)$$

$$-\frac{1}{2} \frac{\left(\sum_{i \in I_p} g_i\right)^2}{\sum_{i \in I_p} h_i + \lambda} + \gamma - \left(-\frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} \right] + 2\gamma \right)$$

take $\frac{1}{2}$ common & $\gamma - 2\gamma = -\gamma$

$$L_{split} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I_p} g_i\right)^2}{\sum_{i \in I_p} h_i + \lambda} \right] - \gamma$$

Loss reduction after the fit

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN)

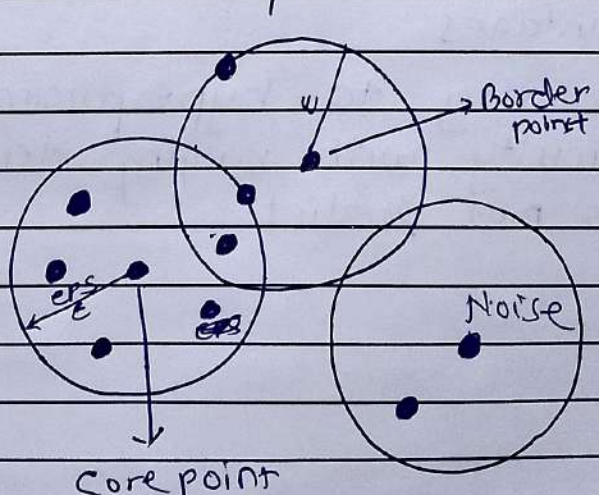
It is a density-based clustering non-parametric algorithm. given a set of points in some space, it groups together points that are closely packed & marks as outliers points that lie alone in low-density regions

In this algorithm we have 3 types of data points.

Core Point: A point is a core point if it has more than MinPts points within eps.

Border Point: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.

Noise or outlier: A point which is not a core or border point.



Steps in Algorithm.

Step 1 : Identify all points as either core point, border point or noise point.

Step 2 : for all of the unclustered core points
a : Create new cluster
b : add all the points that are unclustered & density connected to the current point into this cluster.

Step 3 : for each unclustered border point assign it to the cluster of nearest core point

step 4 : Leave all the noise points as it is.

* Advantages
1. Robust to outliers
2. No need to specify clusters
3. Can find arbitrary shaped clusters
4. Only 2 hyperparameter to tune

* Disadvantages
1. Sensitivity to hyperparameters
2. Difficulty with varying density clusters
3. Does not predict

Syntax Parameters

class sklearn.cluster.DBSCAN (eps=0.5, *, min_samples = 5, metric = 'euclidean', metric_params = None, algorithm = 'auto', leaf_size = 30, p = None, n_jobs = None)

metric : str, or callable, default = 'euclidean'

algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'},

* Implementation

```
>> from sklearn.cluster import DBSCAN
>> cluster = DBSCAN ( eps=3, min_samples=2)
>> cluster.fit(X)
>> cluster.labels_
```

Imbalanced data

Imbalanced data is a common problem in ML, which brings challenges to feature correlation class separation & evaluation & result in poor model performance.

It can be found in countless of applications in finance, healthcare, and public sectors.

Example

bank employee responsible for detecting the validity of credit card transaction.

a] Normal b] Fraudulent

most transaction are normal - but 0-1% are likely to fraudulent

Normal (0) = 99%

Fraudulent (1) = 0.1%

This is imbalanced data.

* Challenges with Imbalanced data.

① The majority class - class with highest no of samples.

② The minority class - class with the lowest no of samples.

③ The class ratio - ratio bet^n minority class & majority class

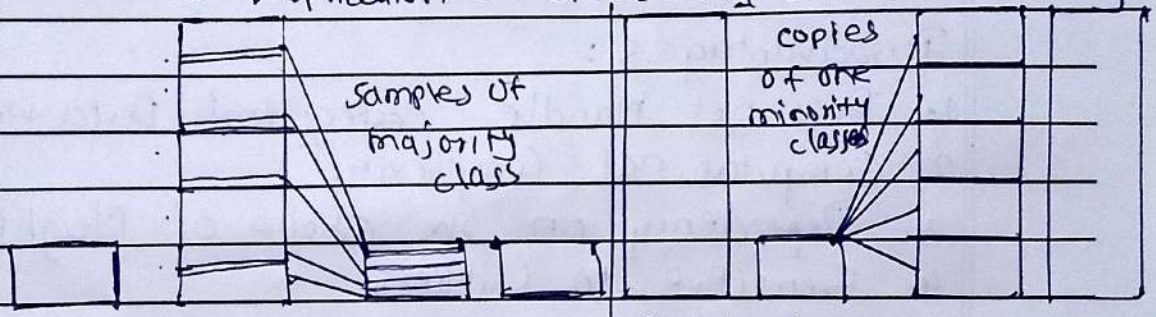
* Sampling techniques to handle imbalanced data.

1. Random Under-Sampler (RU)

Down samples the majority class by randomly selecting instances to match the minority class.

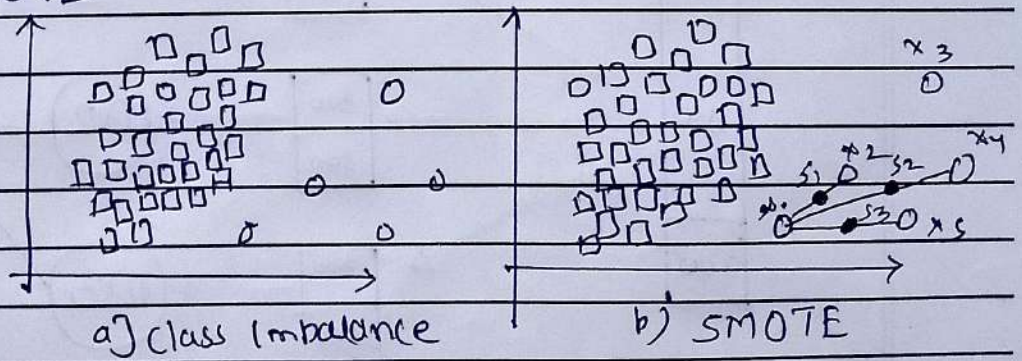
Pros :→ Reduction in bias
→ Faster training
Cons : = Info loss leading to underfitting
= Sampling Bias

2. Random Over-Sampler (RO)
Upsamples the ~~no~~ minority class by randomly duplicating instances until class size are balanced.
Pros :→ Avoids data deletion / Reduce bias
cons :→ Increased size
→ Duplication of data may cause overfitting.



Undersampling Oversampling

3. SMOTE



a] Class Imbalance b) SMOTE

STEPS TO apply SMOTE
Approach: Generates new instances for the minority class to avoid exact duplication.

Algorithm:

Step: Train KNN on minority class obs. And each obs 5 closest neighbours.

Step 2: Select examples from minority class at rand

Step 3: Select a neighbour of each ex. at random

Step 4: Extract a random no. between 0 & 1

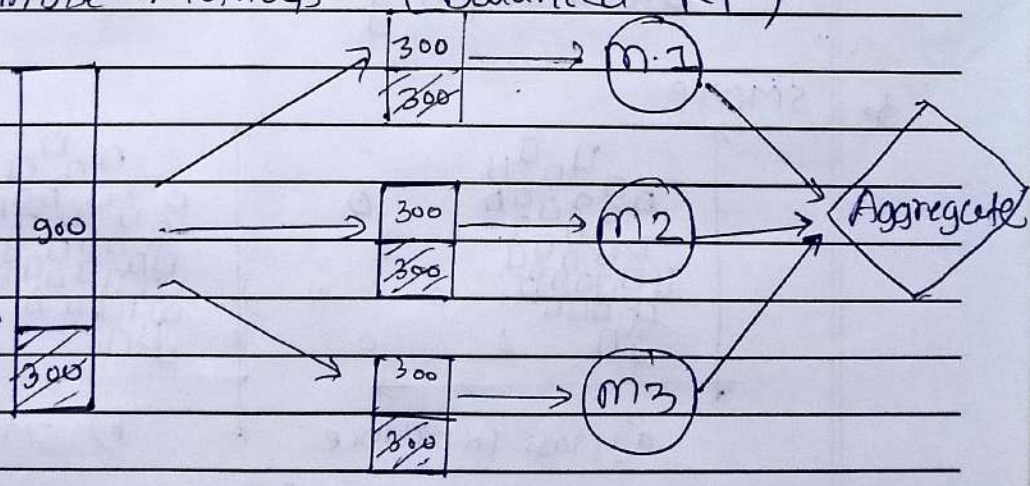
Step 5: Cal the new e.x. as = original sample - Factor * (original sample - neighbour)

Step 6: The final dataset consist of the original dataset + newly created examples.

Disadvantages:

1. Does not Handle Categorical Data well
2. Computational Complexity
3. Dependency on the Choice of Neighbors
4. Sensitive to Outliers
5. Balance Achieved May Not Reflect True Nature

4. Ensemble Methods (Balanced RF)



1200

Approach: Modifies the traditional RF algo to handle class imbalance by using balanced bootstrapping.

Algorithm.

Step 1: Each tree is trained on a bootstrap sample that contains an equal no of instances from each other.

Step 2: This done by randomly under-sampling the majority class &/or over-sampling the minority class.

5. Cost Sensitive Learning

Is an approach within ML that considers varying costs associated with diff types of errors. This method diverges from traditional approaches by introducing a cost matrix, explicitly specifying the penalties like class weight.

Class weights in Cost-Sensitive Learning
class weights adjust the importance of different classes in imbalanced datasets:
It increases model sensitivity to minority class by assigning higher weights to them.

Implementation
1. Manual: set weights manually based on class importance.
2. Automatic: use library features. like sklearn. there is a set parameter for every differentiable Loss function models. "class_weight" for automatic weight calculation.