COMPLETE ML, NLP BOOTCAMP MLOPS & DEPLOYMENT

PYTHON

• Numpy:

- np.arange(0,10,2).reshape(5,1) → op: [[0] [2] [8]]

- np.ones((3,4)) → [[1,1,1,1] [1,1,1,1] [1,1,1,1]]

- np.eye(3) → forms identity matrix → [[1,0,0] [0,1,0] [0,0,1]]

- arr = np.array([1,2,3,4],[5,6,7,8],[9,10,11,12])
  print(arr[0:2,2:]) → [[3,4] [7,8]]

- Numpy array of shape (6,6) with values = 1 to 36. → np.arange(1,37).reshape(6,6) → array → print(array)

- Numpy array of shape (5,5) filled with random integers. Use fancy indexing to extract the elements at the corners of the array.

```
array = np.random.randint(1,21, size=(5,5))
print(array)
corners = array[[0,0,-1,-1],[0,-1,0,-1]]
print(corners)
```

arr. → circled ele.s = output.

• Pandas:

- Create a series (1D array) from dictionary: data = {'a':1, 'b':2} → pd.Series(data) → op: a 1, b 2

For 2D array → Dataframe

- Data aggregating & grouping:

```
grouped_mean = df.groupby('Product')['Value'].sum()
```

- Merging & joining dataframe

```
df1 = pd.DataFrame({'key':['A','B','C'], 'value1':[1,2,3]})
df2 = pd.DataFrame({'key':['A','B','D'], 'value2':[4,5,6]})
pd.merge(df1,df2, on="key", how="inner")
pd.merge(df1,df2, on="key", how="outer")
```

• Reading Data from Different Sources

→ from io import StringIO

```
!pip install lxml
!pip install html5lib
!pip install beautifulsoup4
```

- df = pd.read_html(url)
- df_excel = pd.read_excel('data.xlsx')

• Seaborn:

```
import seaborn as sns
tips = sns.load_dataset('tips')
sns.scatterplot(x='total_bill', y='tip', data=tips)        : scatterplot
sns.lineplot(x='size', y='total_bill', data=tips)          : line plot
sns.barplot(x='day', y='total_bill', data=tips)            : categorical / Bar plots
sns.boxplot(  "  "  )                                      : box plot
sns.violinplot(  "  "  )                                   : violin plot
sns.histplot(tips['total_bill'], bins=10, kde=True)        : histograms
sns.kdeplot(tips['total_bill'], fill=True)                 : KDE plot
sns.pairplot(tips)                                         : pairplot → shows relation btw all variables
corr = tips[['total_bill', 'tip', 'size']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')             : Heatmap
```

• SQLite.

It's a lightweight, self-contained, serverless & zero-configuration SQL database engine.

- Connect to sqlite DB: `import sqlite3` → `connection = sqlite3.connect('example.db')`
  - After this, example.db will be created in current directory

Now to execute queries on the DB:

- `cursor = connection.cursor()`

- Create a table

```
cursor.execute('''
    Create Table If Not Exists employees (
        id Integer primary key,
        name Text not null,
        age Integer,
        department text
    )
''')
connection.commit()
```

- Inserting the data

```
cursor.execute('''
    insert into employees(
    name, age, department)
    values ('Ankit', 32, 'Swf')
''')
connection.commit()
```

- Query the data

```
cursor = execute('select * from employees')
rows = cursor.fetchall()
for row in rows:
    print(row)
```

- Insert multiple rows

```
sales_data = [('2023-01-01', 'Product1', 100), ('2023-01-02', 'Product2', 200)]
cursor.executemany('''
    insert into sales (date, product, sales, region)
    values (?,?,?,?)
''', sales_data)
```

• File Operations - Read and Write Files

- Read a whole file:

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content.strip())
```

- Write a file with overwriting:

```
with open('example.txt', 'w') as file:
    file.write('I am kind \n')
```

- Write a file without overwriting:

```
with open('example.txt', 'a') as file:
    file.write('Appending this line \n')
```

- Writing a list of lined to a file: → `file.writelines(lines)` (lines: list)

→ The 'w+' mode in Python is used to open a file for both reading and writing. If the file does not exist, it will be created. If the file exists, the content in file is overwritten.

```
with open('example.txt', 'w+') as file:
    file.write("Hello World \n")
    file.seek(0)  → moving cursor to the beginning
    content = file.read()
    print(content)
```

• Working with File Paths:

```
- import os
  cwd = os.getcwd()  → gets current working directory
- items = os.listdir('.')  → Listing files & directories
  print(items)
- os.path.isfile(path)  → checks if 'path' is a file   (let path = 'example.txt')
  os.path.isdir(path) → checks if 'path' is a directory
```

• Exception Handling: try, catch & finally

```
def divide(a,b):
    try:
        result = a/b
    except ZeroDivisionError as e:
        print(f"Error {e}")
        result = None
    finally:
        print('Execution Complete')   } This block always runs independent of try or catch.
    return result

print(divide(10,0))
```

#OOPS

Inheritance

» Single Inheritance:

```
class Car:
    def __init__(self, doors, enginetype):  # Constructor
        self.doors = doors
        self.enginetype = enginetype

    def drive(self):
        print(f"{self.enginetype} car.")

car1 = Car(5, "petrol")
car1.drive()

class Tesla(Car):
    def __init__(self, doors, enginetype, selfdriving):
        super().__init__(doors, enginetype)
        self.is_selfdriving = selfdriving

    def selfdriving(self):
        print(f"Supports {self.is_selfdriving}")

t1 = Tesla(5, "electric", True)
t1.selfdriving()
t1.drive()
```

Output:
petrol car
Supports : True
electric car

» Multiple Inheritance:

ex:
```
class Animal:
    # constructor -> (self, name)
    def speak(self)

class Pet:
    # constructor -> (self, owner)

class Dog(Animal, Pet):
    def __init__(self, name, owner):
        Animal.__init__(self, name)
        Pet.__init__(self, owner)
```

Polymorphism

ex: Method Overriding:

```
class Veh:
    def start(self):
        print("V")

class Car(Veh):
    def start(self):
        print("C")
        super().start()

car = Car()
car.start()
```

op:
C
V

ex: polymor. with func.'s & methods:

```
class Shape:
    def area(self):
        return "Area is"

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, rad):
        self.rad = rad

    def area(self):
        return 3.14 * rad * rad

def print_area(shape):
    print(f"Area = {shape.area()}")

rectangle = Rectangle(4,3)
circle = Circle(3)

print_area(rectangle)
print_area(circle)
```

(next line)

ex: Operator Overloading: allows to define the behaviour of operators for custom objects.

- common operator overloading magic methods:
  - `__add__(self, other)` : for +
  - `__sub__(self, other)` : for -
  - `__mul__` : for *
  - `__truediv__` : for division
  - `__eq__` : for checking '=='

ex:
```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x,
                      self.y + other.y)

v1 = Vector(2,3)
v2 = Vector(4,5)
print(v1 + v2)
```

- Abstract Class: It is a class that is used as a base class for defining abstract classes, which cannot be instantiated directly.

(It's blueprint for other classes. It can't be directly used to create objects)

ex:
```
from abc import ABC, abstractmethod
class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass

class Car(Vehicle):
    def start(self):
        return "CarE"

class Bike(Vehicle):
    def start(self):
        return "BikeE"

def Veh(vehicles):
    print(vehicles.start())

car = Car()
Veh(car)
```

(indentation present)

op: CarE

The 'Vehicle' class can't be used on its own to create objects. It only defines that any 'Vehicle' must have 'start' method but doesn't specify what it does.

Encapsulation:

ex:
```
class Person:
    def __init__(self, name, age, gender):
        self.__name = name      → private access Modifier
        self._age = age         → protected access
        self.gender = gender    → public access

def get_name(person):
    return person.gender
person = Person("Ankit", 19, "M")
get_name(person)  → op: M
```

dir(person):
lists all attributes & methods available for the 'person' object

Abstraction → hides complex implementation details

- **Magic Methods**: predefined methods in Python that one can override to change behaviour of the objects.

Ex: →
- `__init__` : initializes a new instance of a class
- `__str__` : returns a string representation of an object
- `__repr__` : returns an official string representation of an object
- `__len__` : returns the length of an object

ex:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name}, {self.age} years".

    def __repr__(self):
        return f"Person(name={self.name})"
```

```
person=Person("Ankit",19)
print(person)
print(repr(person))
```
↓ op:
Ankit, 19 years
Person(name=Ankit)

- **Custom Exception**:

(In java/c++ → try-catch
In python → try-except)

ex:
```
class Error(Exception):
    pass
class dobException(Error):
    pass
year = int(input("Enter your dob"))
age = year - 2024
try:
    if age<=30 and age>=20:
        print("Yes")
    else
        raise dobException()
except dobException:
    print("No")
```

(when we are raising an exception, we also need to catch it, so writing 'except' block.

#Iterators: Allows efficient looping & memory management.

ex:
```
m = [1,2,3,4]
iterator = iter(m)
next(iterator)
```
→ op: outputs next ele. each time we run this code block
(, gives "StopIteration" error at end
, uses Lazy loading technique.

#Generators: are a simpler way to create iterators

ex:
```
def square(n):
    for i in range(3)
        yield i**2      #'yield' → creates a local variable & it saves the value abc

a = square(3)
next(a)
```

#Decorators: Allows us to modify the behaviour of a func. or class method

↳ 3 features: function copy, closures, decorators

- **Function copy**: (we can copy 1 func. to other)

```
def welcome():
    return "Hi"

wel = welcome
print(wel())
delete welcome
print(wel())
```
op: Hi
Hi

- **Closures** function: a function inside a function

ex:
```
def main_wel(func, lst):
    def sub_wel():
        print("Hi")
        print(func(lst))
    return sub_wel()

main_wel(len, [1,2,3])
```
op: Hi
3

- **Decorator**:

```
def my_decorator(func):
    def wrapper():
        print("Before")
        func()
        print("After")
    return wrapper

@my_decorator
def say_hello():
    print("Hi")

say_hello()  → op: Before
                   Hi
                   After
```

(This goes inside)

*args: allows the wrapper to accept any no. of positional arguments

**kwargs: allows the wrapper to accept any no. of keyword argument)

↓ ex:

```
def example(*args, **kwargs):
    print("Args:", args)
    print("kwargs:", kwargs)

example(1,2,3, name="AV", age=19)
```
op: Args: (1,2,3)
kwargs: {'name': 'AV', 'age': 19}

Python Logging

built-in logging module offers a flexible framework for emitting log messages from Python programs.

Several log levels:

- DEBUG: detailed info., typically of interest only when diagnosing problems.
- INFO: confirmation that things are working as expected.
- WARNING: something unexpected happened.
- ERROR: A more serious issue, the software has not been able to perform some func.
- CRITICAL: Very serious error.

ex:
```
import logging
logging.basicConfig(level = logging.DEBUG)
logging.debug("debug message")
```
→ Logs: DEBUG: debug message

→ Program: a sequence of instructions written in programming language
→ Process: an instance of a program that is being executed.
→ Thread: unit of execution within a process

Multiprocessing

→ used when we do "parallel execution"

```
import multiprocessing
import time

if __name__ == "__main__":
    p1 = multiprocessing.Process(target = square_numbers)   # (func. defined by us)
    p2 = ""  # another process
    t = time.time()
    p1.start()
    p2.start()      → start the process
    p1.join()
    p2.join()       → wait for process to complete
    print(time.time() - t)   → (prints time to complete)
```

- We can also use "ProcessPoolExecutor"

Multithreading

```
import threading
import time

t1 = threading.Thread(target = print_numbers)
t2 = ""  # another thread
t = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
print(time.time() - t)
```

- We can also use "ThreadPoolExecutor"

Memory Management

- Reference Counting: is the primary method Python uses to manage memory.

Each object in python maintains a count of references pointing to it. When the reference count = 0, the memory occupied by the object is deallocated.

ex:
```
import sys
a = []
print(sys.getrefcount(a))
```
(one reference from a, one from getrefcount())

- Garbage Collection: Python includes a cyclic garbage collector to handle reference cycles. Reference cycles occur when objects reference each other, preventing their reference counts from reaching zero.

- Best practices:

1) Use local variables
2) Avoid circular reference
3) Use Generators
4) Explicitly delete objects
5) Profile Memory Usage

FLASK → web framework

(Web Server) ⇄ (Web App)

WSGI Protocol → (Web Server Gateway Interface)

```
from flask import Flask, render_template, request
app = Flask(__name__)   # creates an instance of Flask class which will be my WSGI application.
@app.route("/")   # By def. GET method
def welcome():
    return "<html><H1> Welcome </H1></html>"
```

render_template: responsible in redirecting to another HTML page. It looks for a folder 'templates' & it will redirect to the html pages present in it.

```
@app.route("/index", methods = ["GET"])
def index():  # function name must be unique everytime
    return render_template('about.html')

@app.route("/form", methods = ['GET', 'POST'])
def form():
    if request.method == 'POST':
        name = request.form['name']
        return f'Hello {name}!'
    return render_template('form.html')

if __name__ == "__main__":   # Entry point of any .py file. → what main() in C++
    app.run(debug = True)   # Results in running entire Flask app.
```

'debug = True' → results in dynamic web pages if any changes in code is done. It restarts the server automatically.

- To run above app → in terminal
  "python <filename>"

We can also define : (POST, PUT, DELETE & dynamic url) operations in Flask

Ex:
```
@app.route('/successres/<int:score>')
def successres(score):
    res = "";
    if score >= 50:
        res = "PASSED"
    else:
        res = "FAILED"

    exp = {'score': score, 'res': res}
    return render_template('result.html', results = exp)
```

Now in result.html: ⇒ Jinja2 template: default template engine used by flask
(allows Python like expressions to embed into HTML for creating dynamic web content.)

```
<html>
<h2> Final Results </h2>
<body>
<table border = 2>
    {% for key, value in results.items() %}
    <tr>
        <th> {{key}} </th>
    </tr>
    {% endfor %}
</table>
</body>
</html>
```

- {{ y }} - expression to print output in html
- {% %} - condition → for loop
- {# #} - for comment

STREAMLIT

It is an open-source app framework for ML & DS projects. It allows us to create beautiful web applications with simple python scripts.

- To run it → open terminal → "streamlit run <filename>"

Ex:
```
import streamlit as st
import pandas as pd
import numpy as np

st.title("Hello")
df = pd.DataFrame(
    np.random.randn(20,3), columns = ['a','b','c']
)
st.line_chart(df)
```

Q) Merge 2 lists into dictionary.

A)
```
def func(keys, values):
    if len(keys) != len(values):
        return False
    d = dict(zip(keys, values))
                  └ makes into key-val pairs
    return d
```

Q) Merge multiple dictionaries into 1

A)
```
def func(d1, d2, d3):
    d = d1 | d2 | d3
    return d
```

* If same key present in other dictionaries, the value of rightmost dictionary in above will overwrite the prev. value.

Q) Count frequency of words in sentence.

A)
```
def func(sentence):
    sentence = sentence.lower()
    word = sentence.split()
    word_count = {}
    for word in words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    return word_count
```

Q) Check if tuple is palindromic

A)
```
def func(tup):
    return tup == tup[::-1]
```

Q) Merge dictionaries with common keys → (add their value if key → common)

A)
```
def func(dicts):
    ans = {}
    for d in dicts:
        for key, value in d.items():
            if key in ans:
                ans[key] += value
            else:
                ans[key] = value
    return ans
```

FEATURE ENGINEERING

MISSING VALUES:

1) Missing Completely at Random (MCAR): Missing values are completely random & not related to any observed or unobserved data. The probability of a value being missing is the same for all observations.
2) Missing at Random (MAR): Missing values are related to observed data but not to the missing value themselves. Given the observed data, the probability of a value being missing can be estimated.
3) Missing Not at Random (MNAR): The missing data is directly related to the value that is missing.

⇒ Handling Missing Values:

1) Mean Value Imputation: filling with 'mean'
   ex: df['Age_mean'] = df['age'].fillna(df['age'].mean())
2) Median Value Imputation: {If we have outliers in the dataset}
   ex: df['Age_median'] = df['age'].fillna(df['age'].median())
3) Mode Imputation: {Categorical values}
   ↳ filling with 'mode' of that column.

⇒ Handling Imbalanced Dataset:

1) UpSampling: Increasing the no. of instances in the minority class to balance the dataset.

// refer example in "notebook" → used → from sklearn.utils import resample

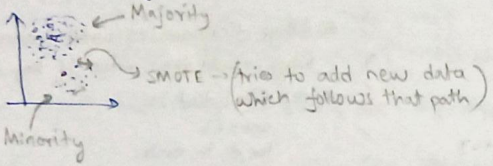
- Basically, it duplicates the same data to match the 'no.' of majority class

2) DownSampling: Reducing the no. of instances in the majority class to balance the dataset.

(Generally not preffered as there's loss of data.)

3) SMOTE: {Synthetic Minority Oversampling Technique}

It generates synthetic instances of minority classes by interpolating between existing instances.

ex: 

from imblearn.over_sampling import SMOTE

refer example in "notebook"

* lst = [45, 32, ..., 74] # a list

import seaborn as sns

sns.boxplot(lst) → 

→ My boxplot will automatically generate "5 number summary" and draw a box plot.

(HANDLING OUTLIERS)

DATA ENCODING:

↳ means that we will be converting the categorical features into some numerical values, so that the model will be able to understand for a specific purpose. We have 3 types of encoding: → Nominal/OHE Encoding
→ Label & Ordinal Encoding
→ Target Guided Ordinal Encoding

1) Nominal / One-Hot Encoding:

- Here each category is represented as binary vector where each bit corresponds to a unique category. Ex: If we have a cat. var. "color" with 3 possible values (R, G, B), we can represent it using one hot encoding as follows:
  - Red: [1,0,0]
  - Green: [0,1,0]
  - Blue: [0,0,1]
- Disadvantage: If we have categorical variable with many features (like 100), then it will create 100 new columns for this, which is not recommended.

ex:
```
from sklearn.preprocessing import OneHotEncoder
df = pd.DataFrame({'color': ['red', 'blue', 'green', 'green', 'red']})   # creating a simple DF
encoder = OneHotEncoder()   # creating an instance of OHE
encoded = encoder.fit_transform(df[['color']]).toarray()   # 2D → converts into a sparse matrix
encoder_df = pd.DataFrame(encoded, columns = encoder.get_feature_names_out())
```

encoder_df →

	color_blue	color_green	color_red
0	0	0	1
1	1	0	0
2	0	1	0
3	0	1	0

# we can also concat.
pd.concat([df, encoder_df], axis=1)

2) Label Encoding:

- It involves assigning a unique numerical label to each category in the variable. The labels are usually assigned in alphabetical order or based on the frequency of the categories.

ex: Color → Red: 1, Green: 2, Blue: 3

ex:
```
from sklearn.preprocessing import LabelEncoder
lbl = LabelEncoder()
lbl.fit_transform(df[['color']])
```

- Disadvantage: Here red=2, blue=0, green=1. The model might start thinking that red have higher value/priority than blue. This should not happen because these are nominal values.

But yes, if there's a situation where we need to assign ranks to these particular values to this category, we can definetely do it with the help of ordinal encoding.

3) Ordinal Encoding:

- In this, each category is assigned a numerical value based on its position in the order.

ex: education level → High School: 1, College: 2, Graduate: 3, Post-Graduate: 4 ← represented via ordinal encoding

ex:
```
from sklearn.preprocessing import OrdinalEncoder
df = pd.DataFrame({'size': ['small', 'medium', 'large', 'medium', 'small', 'large']})
encoder = OrdinalEncoder(categories = [['small', 'medium', 'large']])
encoder.fit_transform(df[['size']])

# New data
encoder.transform([['small']])
```

4) Target Guided Ordinal Encoding:

- It's used to encode categorical variables based on their relationship with the target variable.
- This technique is useful when we have a categorical variable with a large no. of unique categories, & we want to use this variable as a feature in our machine learning model.
- Here, we replace each category in the categorical variable with a numerical value based on the mean or median of the target variable for that category.

ex:
```
df = pd.DataFrame({'city': ['A', 'B', 'C', 'D', 'A', 'C'], 'price': [10, 20, 30, 20, 25, 40]})
mean_price = df.groupby('city')['price'].mean().to_dict()
mean_price → {'A': 17.5, 'B': 20.0, 'C': 35.0, 'D': 20.0}
df['city_encoded'] = df['city'].map(mean_price)
```

df[['price', 'city_encoded']] ⟶ copy

	price	city_encoded
0	10	19.5
1	20	20.0
2	30	35.0

Practicing EDA:

1) Red Wine Data: → .head()
.info() → summary of dataset
.describe() → descriptive summary of dataset

→ checking unique & missing values. finding duplicate records & dropping them

→ Using 'sns' visualizations to bring out conclusions → like: pairplot, histplot, scatterplot, etc.

2) Flight Price Prediction: → .head(), .info(), .describe()

→ splitting columns ⟶ ex: Date of Journey (24/03/2019) ⟶ into: Day (24) | Month (03) | Year (2019)

→ Also checking the dtype of all columns. Ensure numeric columns have dtype as 'int' or 'float' & not 'obj'

→ Replacing 'nan', empty blocks with 'mode' of that column

→ Using OHE to change categorical values into numerical values

3) Google Play Store Dataset:

→ .shape, .info(), .describe(), .isnull().sum()

In EDA, we just need observation

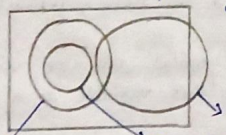
→ Converting col. values into meaningful data ⟶ ex: In reviews → 3M+ (changing to int)

→ Removing duplicate records

→ Plotting different parameters & noting observations (like: top 10 apps used)

INTRODUCTION TO MACHINE LEARNING

AI = to create an application which can perform it's own task without any human intervention
ex: Netflix recommendation system



Data Science

Deep Learning (Multi-Layered NN) = Mimic the human brain

ML
"
(It provides stats tools to analyze, visualize, predict & forecast the data sometimes)

Machine Learning Technique
- Supervised ML
- Unsupervised ML
- Reinforcement Learning

1) Supervised ML → Regression: predicts continuous values. {continuous}
→ Classification: predicts discrete class labels. {categorical}
↳ Dependent on output feature
- binary classification = only 2 categories
- Multiclass classification } > 2 categories

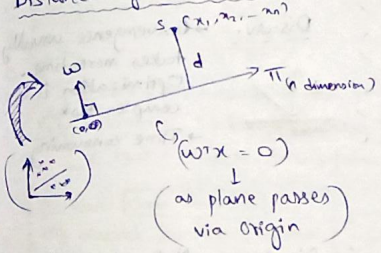
- Linear Regression
- Ridge & Lasso
- ElasticNet
- Logistic Regression (Classification)
- Decision Tree
- Random Forest
- Ada Boost
- XgBoost

(both classification & regression)

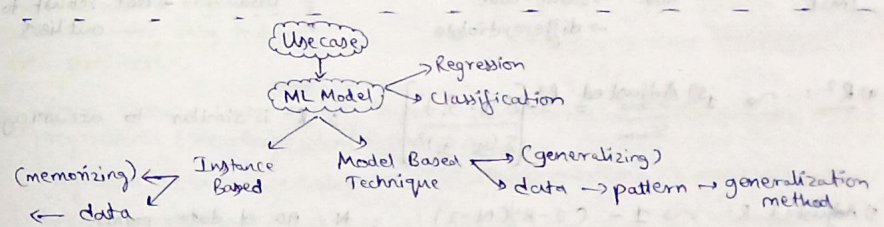
2) Unsupervised ML (we don't have o/p feature label)
we create clusters to group similar kind of data
- K Means
- Hierarchical Mean
- DBScan Clustering

// Refer "Eqn. of a Line, Plane, Hyperplane" → PREV. CHAPTER

Distance of a point from plane:



$d = \frac{w^T \cdot s}{\|w\|}$; $w^T \cdot s = \|w\| \cdot \|s\| \cos\theta$



Usual ML	Instance Based Learning
• Prepare the data for model training	• Prepare the data for model training. No diff. here
• Train model from training data to estimate model parameters i.e. discover patterns	• Do not train model. Pattern discovery postponed until scoring query received.
• Store the model in suitable form	• There is no model to store.
• Generalize the rules in form of model, even before scoring instance is seen	• No generalization before scoring. Only generalize for each scoring instance individually as and when seen.
• Predict for unseen scoring instance using model	• Predict for unseen scoring instance using training data directly
• Can throw away training data after model training.	• Training data must be kept since each query uses part/full set of training observations
• Requires a known model form	• May not have explicit model form.
• Storing models generally requires less storage	• Storing training data generally requires more storage.
• Scoring for new instance is generally faster.	• Storing for new instance may be slow.

UNDERSTANDING LINEAR REGRESSION INDEPTH INTUITION AND PRACTICALS

Refer "Application of Linear Algebra, Stats & Differential Calculus" → PREV. CHAPTER

Convergence Algorithm:

Repeat until convergence ⇒ $\theta_j = \theta_j - \alpha\left(\frac{\partial}{\partial\theta_j} J(\theta_0,\theta_1)\right)$; $J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^{(i)} - y^{(i)})^2$ (COST FUNCTION)

- $\theta_0 = \theta_0 - \alpha\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x)^{(i)} - y^{(i)})\right)$
- $\theta_1 = \theta_1 - \alpha\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x)^{(i)} - y^{(i)})x^{(i)}\right)$

{// Refer Prev. Chapter}

Cost Function → Performance Metrics: → I/W Que.: Check usage of all performance metrics

1) **MSE:** ⇓ $\frac{\sum_{i=1}^{n}(y-\hat{y})^2}{n}$ (Mean Squared Error)
   - Adv.: → Differentiable; → It has 1 local & 1 global minima.; → Converges faster
   - Disadv.: → Not robust to outliers; → It is not in same unit.

2) **MAE:** (Mean Absolute Error) ⇓ $\frac{1}{n}\left(\sum_{i=1}^{n}|y-\hat{y}|\right)$
   - Adv.: → Robust to outliers; → It will be in the same unit.
   - Disadv.: → Convergence usually takes more time. Optimization is a complex task.; → Time consuming.

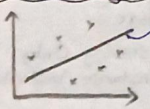
3) **RMSE:** (Root Mean Squared Error) ⇓ $\sqrt{MSE}$
   - Adv.: → Same unit; → differentiable
   - Disadv.: → Not robust to outliers

4) **R^2:** ⇝ $1 - \frac{SS_{res}}{SS_{total}} = 1 - \left[\frac{\sum(y_i-\hat{y}_i)^2}{\sum(y_i-\bar{y}_i)^2}\right]$; R^2 is similar to accuracy

5) **Adjusted R^2:** ⇝ $1 - \frac{(1-R^2)(N-1)}{N-p-1}$; N = no. of data points, p = no. of independent features

{// Refer Prev. Chapter}

Linear Regression Using OLS (Ordinary Least Square):



$h_\theta(x) = \beta_0 + \beta_1 x_1$ → OLS: calculates β_0 & β_1 (main aim: reduce the error)

- $\beta_0 = \bar{y} - \beta_1\bar{x}$ } Intercept
- $\beta_1 = \frac{\sum_{i=1}^{n}(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})}$ } Coefficient

(for derivation → refer pdf)

Basically, cost func. is used & differentiated wrt β_0 & β_1

Ex: (Refer Notebook for output)

```
df = pd.read_csv('height-weight.csv')
df.head()  → exploring data set
plt.scatter(df['Weight'], df['Height'])  → scatterplot for visualization
df.corr()  → Checking correlation btw variables
```

No. closer to 1 = highly correlated; No. closer to -1 = least correlated

SIMPLE LINEAR REGRESSION

```
import seaborn as sns
sns.pairplot(df)
# Independent & Dependent features
X = df[['Weight']]  → To create independent features, ensure it's in the form of a DataFrame or 2D array.
y = df['Height']  → Dependent/Target variable can be series or 1D array.
# Train_test_split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# standardization
from sklearn.preprocessing import StandardScaler
```

We apply this because the ip values might be higher & we need to draw a gradient descent, so higher the value, more the time it will take. So we apply Z-score & convert it into $\mu=0$ & $\sigma=1$, so that it takes less time.

$\left(\frac{x_i-\mu}{\sigma}\right)$

```
scaler = StandardScaler()  → Z-score transformation occurs in all values of x_train
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

'transform' represents we will use same μ & σ as the one used in training dataset.

To avoid data leakage, as my model shouldn't know anything about test data, so we are using only training data parameters. ← WHY?

```
from sklearn.linear_model import LinearRegression  → Applying ML Model
regression = LinearRegression(n_jobs=-1)  → used for computation. '-1' = use all processor available in the system.
regression.fit(x_train, y_train)
print(regression.coef_)  → β1 → means 1 unit movement in x equals 'β1' movement in y
print(regression.intercept_)  → β0
plt.scatter(x_train, y_train)
plt.plot(x_train, regression.predict(x_train))  → Visualizing →
# Prediction of test data
y_pred = regression.predict(x_test)  → Predicted height o/p = intercept + coef(weight)
from sklearn.metrics import mean_absolute_error, mean_squared_error  } Performance metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(mse, mae, rmse)
from sklearn.metrics import r2_score
score = r2_score(y_test, y_pred)
print(score)
1 - (1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)  (adjusted r2 score)
```

0: from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)

1: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
1.1
from sklearn.linear_model import LinearRegression
regression = LinearRegression(n_jobs = -1)
regression.fit(x_train, y_train)

SHORTCUTS

2: from sklearn.metrics import mean_absolute_error, mean_squared_error
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(mse, mae, rmse)
from sklearn.metrics import r2_score
score = r2_score(y_test, y_pred)
print(score)
1 - (1-score)*(len(y_test) - 1) / (len(y_test) - x_test.shape[1] - 1)
(adjusted r2_score)

3: import statsmodels.api as sm
model = sm.OLS(y_train, x_train).fit()
prediction = model.predict(x_test)
print(prediction)
model.summary
regression.predict(scaler.transform([[72]]))

4: from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2, include_bias = True)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)
regression = LinearRegression()
regression.fit(x_train_poly, y_train)
y_pred = regression.predict(x_test_poly)
score = r2_score(y_test, y_pred)
print(score)

5: linreg = LinearRegression()
linreg.fit(x_train_scaled, y_train)
y_pred = linreg.predict(x_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
score = r2_score(y_test, y_pred)
plt.scatter(y_test, y_pred)

6: from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

7: from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
logistic.fit(x_train, y_train)
y_pred = logistic.predict(x_test)
print(y_pred)
logistic.predict_proba(x_test)

8: model = LogisticRegression()
params = {'penalty' : ['l1', 'l2', ...],
'c' : [100, ...],
'solver' : ['sag', ...]}

9: grid = GridSearchCV(estimator = model, param_grid = params, scoring = 'accuracy', cv = cv, n_jobs = -1)
grid.fit(x_train, y_train)
grid.best_params_ , grid.best_score_

y_pred = grid.predict(x_test)

10: param_grid = {'c' : [0.1, ...],
'gamma' : [1, ...],
'kernel' : [rbf, ...]}
grid = GridSearchCV(SVC(), param_grid = param_grid(), refit = True, cv = 5, verbose = 3)
grid.fit(x_train, y_train)
grid.best_params_
y_pred2 = grid.predict(x_test)

11: Refer ⇒ "Random Forest Machine Learning"
12: ↗ ↖ 13: ; 14: Refer ⇒ "NLP For Machine Learning" → (BOW + N-grams)

OLS Linear Regression

```
import statsmodels.api as sm
model = sm.OLS(y_train, x_train).fit()
prediction = model.predict(x_test)
print(prediction)        → prints predicted values of x_test
model.summary()          → Gives full summary of model (like coef, std. err., etc)
# Prediction for new data
regression.predict(scalar.transform([[72]]))   → we need to standardize & then predict
```

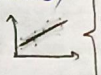
③

(* Here all 'plt' diagrams styling written in brief → refer notebook for detailing)

Ex: (Multiple Linear Regression) ; Refer Notebook for output

```
df = pd.read_csv('economic_index.csv')
df.head()
df.drop(columns=["Unnamed: 0", "year", "month"], axis=1, inplace=True)   → drop unnecessary columns
df.isnull().sum()   → check null values
# Visualization
import seaborn as sns
sns.pairplot(df)
df.corr()
# Get independent & dependent features
X = df.iloc[:, :-1]   # X = df[['interest_rate', 'unemployment_rate']]
                      → means take all rows ':', except last row ':-1'
y = df.iloc[:, -1]    → means take the last column = 'index_price'
X.head()
y
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=
import seaborn as sns
sns.regplot(x=df['interest_rate'], y=df['index_price'])
```

→ Plots data & regression linear model fit. → Works only for 2 features



→ Shaded region: we can hyperparameter tune that line within that range

```
sns.regplot(x=df['index_price'], y=df['unemployment_rate'])
```

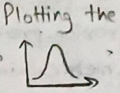
①

```
from sklearn.model_selection import cross_val_score
validation_score = cross_val_score(regression, X_train, y_train,
                                   scoring='neg_mean_squared_error', cv=3)
np.mean(validation_score)   → (avg. of '3' tests) = (cv=3)
# Prediction
y_pred = regression.predict(X_test)
y_pred
```

②

```
residuals = y_test - y_pred
sns.displot(residuals, kind='kde')
plt.scatter(y_pred, residuals)
```

→ Plotting the residuals



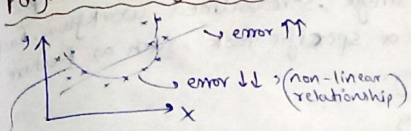
→ Normal distribution is coming means the model we have created is good

↓

(Point are uniformly distributed, so we can conclude that there is no issue in our process)

③

Polynomial Regression:



$h_\theta(x) = \beta_0 + \beta_1 x$ → Simple Linear Regression (1 i/p → 1 o/p feature)

$h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$ → Multiple LR

Simple LR (only 1 feature):

- → Polynomial degree = 0 : $h_\theta(x) = \beta_0 x^0$ ⇒ constant value
- → " " = 1 : $h_\theta(x) = \beta_0 x^0 + \beta_1 x^1$ ⇒ Simple LR
- → " " = 2 : $h_\theta(x) = \beta_0 x^0 + \beta_1 x + \beta_2 x^2$
- → " " = n : $h_\theta(x) = \beta_0 x^0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n$

(If 2 independent features)

- degree = 1 → $h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
- degree = 2 → $h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2$

(As degree increases, the curve gets more twisted & the best fit line will come & eventually overfit. We should select a degree value which should not overfit/underfit but generalize the model)

Ex: (Polynomial Regression) ; Refer Notebook for output

```
# creating dataset
X = 6 * np.random.rand(100, 1) - 3
y = 0.5 * X ** 2 + 1.5 * X + 2 + np.random.randn(100, 1)
plt.scatter(X, y)
```

◎

```
#We can scale the values, but they're already small, so it's not necessary
```

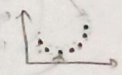
①

```
from sklearn.metrics import r2_score
score = r2_score(y_test, regression.predict(X_test))
print(score)
plt.plot(X_train, regression.predict(X_train))
```

```
# Applying Polynomial transformation
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=True)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
X_train_poly, X_test_poly
regression = LinearRegression()
regression.fit(X_train_poly, y_train)
y_pred = regression.predict(X_test_poly)
score = r2_score(y_test, y_pred)
print(score)
print(regression.coef_, regression.intercept_)
plt.scatter(X_train, regression.predict(X_train_poly))
plt.scatter(X_train, y_train)
```

④

$h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$ (sets $\beta_0 = 1$)



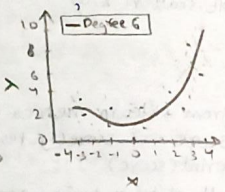
④ → with "degree = 3"

```
# Prediction of new data set
X_new = np.linspace(-3, 3, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = regression.predict(X_new_poly)
plt.plot(X_new, y_new, label="New Predictions")
plt.plot(X_train, y_train, label="Training Points")
plt.plot(X_test, y_test, label="Testing Points")
```

Refer diagram in notebook

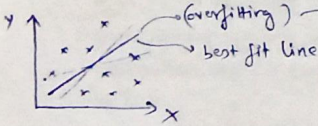
Pipelining: It's a process in DSML where multiple sequential steps are assembled into a single, streamlined workflow. Each step in the pipeline typically performs a specific task, such as feature extraction, preprocessing, or modeling.

Ex:
```
from sklearn.pipeline import Pipeline
def poly_regression(degree):
    x_new = np.linspace(-3,3,200).reshape(200,1)
    poly_features = PolynomialFeatures(degree=degree, include_bias=True)
    lin_reg = LinearRegression()
    poly_regression = Pipeline([
        ("poly_features", poly_features),
        ("lin_reg", lin_reg)
    ])
    poly_regression.fit(x_train, y_train)# Polynomial & fit of LR
    y_pred_new = poly_regression.predict(x_new)
    # Plotting prediction line
    plt.plot(x_new, y_pred_new, 'r', label="Degree "+ str(degree), linewidth=2)
    plt.plot(x_train, y_train, 'b', linewidth=3)
    plt.plot(x_test, y_test, 'g', linewidth=3)
    plt.legend(loc="upper left")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.axis([-4, 4, 0, 10])
    plt.show()
poly_regression(6)
```
(x_lower_limit, y_lower_limit, x_upper_limit, y_upper_limit)



RIDGE, LASSO, & ELASTICNET ML ALGORITHMS

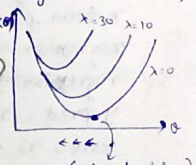
Ridge Regression: (or L2 regularization) → used to reduce overfitting



(to reduce this) — we can also say it's like hyper-parameter tuning the linear regression

$$\text{Cost func.} = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^i - y^i)^2 + \lambda\sum_{i=1}^{n}(\text{slope})^2$$

(no. of slopes/co-eff.)



(This should never be equal to 0 as it would mean that it's overfitting) (λ ∝ 1/slope)

ex: $h_\theta(x) = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_3$

(let) $= 0.34 + 0.52x_1 + 0.48x_2 + 0.24x_3$ → means: if x1 moves by 1, then 'y' will move by 0.52. Same for x2, x3

↓ (after ridge)

$= 0.34 + 0.40x_1 + 0.38x_2 + 0.14x_3$ → after applying ridge regression, value will decrease but it will never be equal to 0.

Lasso Regression: (or L1 regularization) → used for feature selection

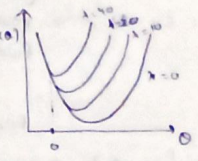
$$\text{(cost func.)} = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^i - y^i) + \lambda\sum_{i=1}^{n}|\text{slope}|$$

ex: $h_\theta(x) = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \theta_4x_4$ (4 independent features)

(let) $= 0.52 + 0.65x_1 + 0.72x_2 + 0.34x_3 + 0.12x_4$

↓ (after lasso)

$0.52 + 0.51x_1 + 0.60x_2 + 0.14x_3 + \boxed{0x_4}$ → 0



(x_4 is only 0.12 times related to output, so we can say that this feature is not that imp.)

- Here also 'θ' value decreases but after one point of time, it will become 0 resulting in removal of less-correlated feature(s).

ElasticNet Regression: → Reduce Overfitting, Feature Selection

$$\text{Cost func.} = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^i - y^i)^2 + \lambda_1\sum_{i=1}^{m}(\text{slope})^2 + \lambda_2\sum_{i=1}^{m}|\text{slope}|$$

(λ_1: reduce overfitting; λ_2: feature selection)

Types of Cross Validation:

Dataset → training → train, validation; test

(cross_validation)

1) Leave One Out CV:

ex: training → 500 records

exp 1: Val. | Train → Acc 1

exp 2: | Val. | Train → Acc 2

⋮

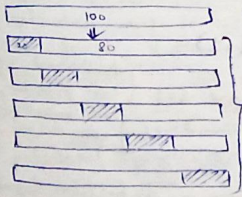
exp 500: Train | Val. → Acc 500

Disadvantage: → Complexity of training model increases

→ Overfitting → train acc. ↑, val. acc. ↓ } new test data → acc. ↓

2) K-Fold CV:

ex: k=5, n=100



Model trained on 5 different versions of training data & evaluated on 5 diff. versions of the test data.

→ avg. accuracies; test size = 100/5 = 20

3) Stratified K-Fold CV:

It's similar to K-Fold CV, but with an imp twist: it maintains the same proportion of different classes (labels) in each fold as in the entire dataset.

- It's useful when working with imbalanced datasets

4) Time Series CV:

Here order of observations is crucial.

ex: CV = 4 → Fold 1: train on [T1], test on [T2]
Fold 2: " " [T1, T2], test on [T3]
Fold 5: " " [T1, T2, T3, T4], test on [T5]

ex: financial forecasting, weather forecasting

Ex: (Ridge & Lasso Regression)

```
#importing
dataset = pd.read_csv('Algerian_forest_fires_dataset_UPDATE.csv', header=1)   (removes 1st row, refer dataset to know why?)
# .head() , .info()                                                            → We have 2 regions
# Data Cleaning
dataset.isnull().sum()                                                         (check missing vals.)
dataset[dataset.isnull().any(axis=1)]                                          (shows rows that have any missing value)
dataset.loc[:122, "Region"] = 0
dataset.loc[122:, "Region"] = 1                                                } creating new col.
df = dataset
df[['Region']] = df[['Region']].astype(int)
df.isnull().sum()
df = df.dropna().reset_index(drop=True)
df = df.drop(122).reset_index(drop=True)                                       → refer (dataset (or) notebook) to know why?
df.columns → Notice there are whitespaces in col. names.
df.columns = df.columns.str.strip()                                            (removing them)
#Change req col. to int dtype                                                  → used "df.info" to see dtypes of all
df[['month','day','year','Temperature','RH','Ws']] = df[[ ]].astype(int)

#Change other col. to float dtype
objects = [features for features in df.columns if df[features].dtypes=='O']
for i in objects:
    if i!='Classes':
        df[i] = df[i].astype(float)

df.describe()
#Save Cleaned dataset
df.to_csv('Aff_cleaned_dataset.csv', index=False)
#EDA
df_copy = df.drop(['day','month','year'], axis=1)
df_copy['Classes'].value_counts()  → o/p:  fire 131
                                           not fire 107
                                           fire  4
                                           :
                                   (Additional spaces are present, so we have to remove that)
df_copy['Classes'] = np.where(df_copy['Classes'].str.contains('not fire'),0,1)
# Visualizing
# Density Plot = shows distribution of dataset over a continuous interval
plt.style.use('seaborn-v0_8')
df_copy.hist(bins=50, figsize=(20,15))
plt.show()
(Correlation) { df_copy.corr()
               sns.heatmap(df.corr(numeric_only=True))
```

```
# Pie chart
percentage = df_copy['Classes'].value_counts(normalize=True)*100
classlabels = ["Fire", "Not Fire"]
plt.pie(percentage, labels=classlabels, autopct='%1.1f%%')
plt.show()
# Box Plot
sns.boxplot(df['FWI'], color='green')
df['Classes'] = np.where(df['Classes'].str.contains('not fire'), 'not fire', 'fire')
# Monthly Fire Analysis
dftemp = df.loc[df['Region']==1]
sns.countplot(x='month', hue='Classes', data=df)
plt.title('Fire Analysis of Sidi-Bel Regions', weight='bold')
-Similarly for df['Region']==0
```

#Observations

Ex: (Model Training) → considering this as regression prob. → predicting 'FWI' col.

```
#Importing - reading dataset : 'Aff_cleaned_dataset.csv' - drop 'month','year','day'
#Encoding
df['Classes'] = np.where(df['Classes'].str.contains('not fire'),0,1)
df['Classes'].value_counts()
x = df.drop('FWI', axis=1)        → Independent & Dependent Features
y = df['FWI']                     → If we consider this as classification prob., we must remove 'Classes' col.
```

ⓞ

```
x_train.corr() → Note: Features which are highly (+)ve correlated can be removed
#check for multicollinearity
sns.heatmap(x_train.corr(), annot=True)
#Func. to identify & return a set of col. names from the dataset that have corr. above a specific threshold
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (abs(corr_matrix.iloc[i,j]) > threshold:
                col_name = corr_matrix.columns[i]
                col_corr.add(col_name)
    return col_corr
corr_features = correlation(x_train, 0.85)
x_train.drop(corr_features, axis=1, inplace=True)
x_test.drop(corr_features, axis=1, inplace=True)
x_train.shape, x_test.shape
#Standardization
① → x_train_scaled, x_test_scaled
# Box plots to understand understand effect of StandardScaler
plt.subplots(figsize=(15,5))
plt.subplot(1,2,1)
sns.boxplot(data=x_train)
plt.subplot(1,2,2)
sns.boxplot(data=x_train_scaled)
```

Linear Regression Model

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet, LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
linreg = LinearRegression()
linreg.fit(x_train_scaled, y_train)
y_pred = linreg.predict(x_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
score = r2_score(y_test, y_pred)
plt.scatter(y_test, y_pred)
```

(5)

Lasso Regression

```
lasso = Lasso()
```

(5) → use 'lasso' instead of 'linreg'

} r2_score is low doesn't mean the model is not performing well, we can also conclude that prev. model is overfitting.

Ridge Regression

```
ridge = Ridge()
```

(5) → use 'ridge' instead of 'linreg'

ElasticNet

```
elastic = ElasticNet()
```

(5) → use 'elastic' instead of 'linreg'

HyperParameter Tuning

```
from sklearn.linear_model import LassoCV, RidgeCV, ElasticNetCV
lassocv = LassoCV(cv=10)
lassocv.fit(x_train_scaled, y_train)
lassocv.alpha_ , lassocv.alphas_ , lassocv.mse_path_
```

(5)

(Explore all it's features)

```
ridgecv = RidgeCV(cv=10)
```

(5)

```
elasticcv = ElasticNetCV(cv=10)
```

(5)

STEP BY STEP PROJ. IMPLEMENTATION WITH LIFECYCLE OF ML PROJ.

Ex: Refer Linear Regression example (height-weight) in previous-previous section. (SEE NEXT PAGE)

```
# New data
scaled_weight = scaler.transform([[80]])  → scaled_weight = array([[0.3235...]])
print("Height prediction for weight 80 kg is: ", regression.predict([scaled_weight[0]]))
```

Assumptions

```
plt.scatter(y_test, y_pred_test)
```

⇒ If we are getting our points linearly, then we can assume we have done very good predictions

```
residuals = y_test - y_pred_test
sns.distplot(residuals, kde=True)
```

⇒ If residuals have a normal distribution, we can say we have created a good model.

```
plt.scatter(y_pred_test, residuals)
```

→ If we get uniform distributions, it confirms we have created a good model

Ex: (Multiple Linear Regression) : Refer Notebook for o/p.

```
# Importing
from sklearn.datasets import fetch_california_housing
california = fetch_california_housing()
california, california.keys(), print(california.DESCR), california.target_names
california.feature_names, california.data.shape
# Prepare dataset
df = pd.DataFrame(california.data, columns = california.feature_names)
df.head()
df['Price'] = california.target
# df.info(), .describe(), .corr()   ; sns.heatmap(df.corr(), annot=True)
# Independent & Dependent Features
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

o/p feature

(0)

(1)

```
regression.coef_ , regression.intercept_
```

(slope/co-eff.)

```
# Prediction for test data
y_pred = regression.predict(x_test)
```

(2)

```
# Assumptions
plt.scatter(y_test, y_pred)
residuals = y_test - y_pred
sns.distplot(residuals, kind='kde')
plt.scatter(y_pred, residuals)
```

Refer prev. page to know what they represent about model

Pickling : used for saving & loading a model.

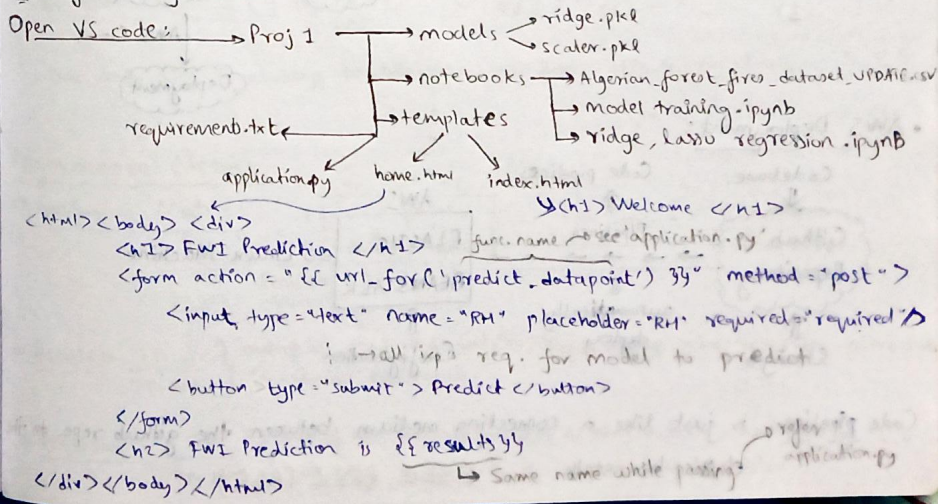
```
import pickle
pickle.dump(regression, open('regression.pkl', 'wb'))
```

Extension must be '.pkl'

```
model = pickle.load(open('regression.pkl','rb'))
# Testing the saved model
model.predict(x_test)
```

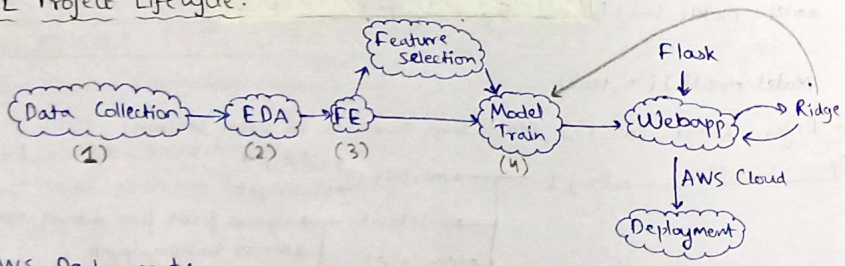
Ex: Refer 'Ridge' Lasso & ElasticNet examples in prev. section

Open VS code:

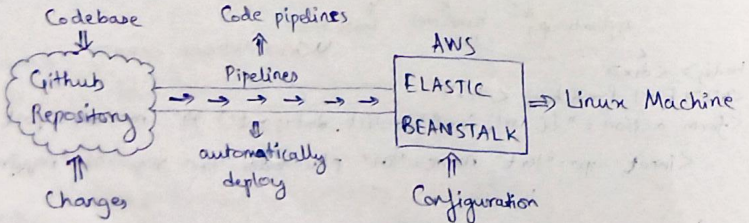


```
# application.py
# To install below libraries, run "pip install -r requirements.txt" in terminal
from flask import Flask, request, jsonify, render_template
import pickle

application = Flask(__name__) # name → "application" → so that it is easy for us to deploy it on AWS
app = application
ridge_model = pickle.load(open('models/ridge.pkl','rb'))
standard_scaler = pickle.load(open('models/scaler.pkl','rb'))

@app.route('/')
def index:
    return render_template('index.html')

@app.route('/predictdata', methods=['GET','POST'])
def predict_datapoint():
    if request.method == 'POST':
        RH = float(request.form.get('RH'))
        ⋮   → get all i/p's
        new_data_scaled = standard_scaler.transform([[RH, ..... ]])   (all i/p's)
        result = ridge_model.predict(new_data_scaled) → 'result' is in the form of list
        return render_template('home.html', results = result[0])
    else:
        return render_template('home.html')

if __name__ == '__main__':
    app.run(host = "0.0.0.0")
    # Deployed on AWS Elastic Beanstalk
```

ML Project Lifecycle:



• AWS Deployment:

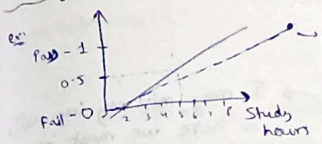


Code pipeline: is just like a connecting medium between the github repo. to the elastic beanstalk. (SEE PREV. PAGE)

LOGISTIC REGRESSION

Logistic Regression is used for solving "Binary Classification" problem.

Solving via Linear Regression?



(adding outlier) → (best fit line changes)

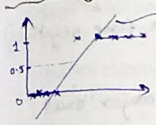
if study hours ≤ 5 → fail, else pass

$\begin{cases} \leq 0.5 \Rightarrow 0 \\ > 0.5 \Rightarrow 1 \end{cases}$

Suppose we add SH = 12 in dataset, now best fit line will incline more towards x-axis, & now SH = 5 → ≤ 0.5, so it's not a good model

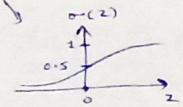
Also, the val. is going < 0 & > 1, so we need to squash it i.e. once the line is out of bounds, make it parallel to x-axis → only possible with the help of Logistic Regression

How Logistic Regression solves Classification Prob.:

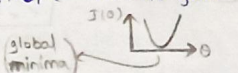


$h_\theta(x) = \theta_0 + \theta_1 x_1$ ⇒ best fit line

Sigmoid Activation: $\frac{1}{1+e^{-z}}$

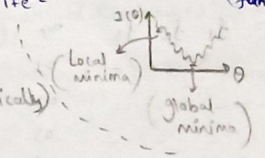


$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^i - y^i)^2$ — Linear Reg: $h_\theta(x) = \theta_0 + \theta_1 x$ → Convex func



Let's denote: $cost(h_\theta(x)^i, y^i)$

Linear Reg. → $h_\theta(x) = \frac{1}{1+e^{-z}}$, $z = \theta_0 + \theta_1 x_1$ → (non convex func.)



Log loss ⇒ $cost(h_\theta(x)^i, y^i) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$

(Proved mathematically)

$\therefore cost(h_\theta(x)^i, y^i) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$

$\therefore J(\theta_0,\theta_1) = -\frac{1}{2m}\sum_{i=1}^{m}\left(y^i \log(h_\theta(x)^i) - (1-y^i)\log(1-h_\theta(x)^i)\right)$

(Our main aim is always to minimize cost func. $J(\theta_0,\theta_1)$ by changing θ_0 & θ_1. (we can use convergence algo.))

Performance Metrics:

1) Confusion Matrix:

Actual values \ Predicted values	1	0
1	True(+) TP	False(+) FN
0	False(−) FP	True(−) TN

→ Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$

2) Recall = $\frac{TP}{TP+FN}$ } out of all predicted values, how many are correctly predicted is shown by 'recall.'

3) Precision = $\frac{TP}{TP+FP}$ } out of all correct results/actual values, how many are correctly predicted is shown by 'precision'

4) F-Beta Score = $(1+\beta^2)\left(\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}\right)$

- If FP & FN are both imp. → β = 1
- If FP is more imp. than FN → β = 0.5
- If FN >> FP → β = 2

ex1: {Spam Classification}

	Spam 1	Not spam 0
Spam 1	TP	FP
Not spam 0	FN	TN

Mail → Spam, Model → Spam } good

Mail → Not spam, Model → Spam } Blunder → I may miss my imp. mail, so here we must increase our precision. (Reduce False Positives)

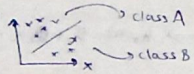
ex2: {Diabetes prediction}

	D	No D
D	TP	FP
No D	FN	TN

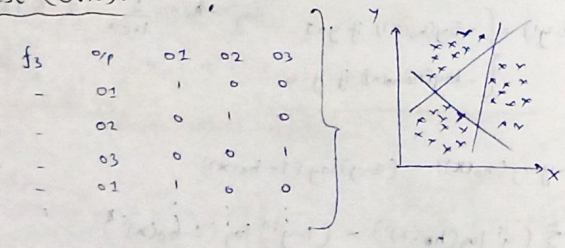
Truth → D, Model → D } good

Truth → D, Model → No D } Blunder → Imp. for person → so we must increase our recall (reduce false negatives)

Truth → No D, Model → D } Person can check. Not a major issue!

One vs One:  } Binary Classification

One vs Rest (OVR):



ex: f_1	f_2	f_3	o/p	o1	o2	o3
-	-	-	01	1	0	0
-	-	-	02	0	1	0
-	-	-	03	0	0	1
-	-	-	01	1	0	0

MultiClass Classification
→ M1
→ M2 — binary classification
→ M3
We have 3 models because we have 3 outputs
all this is done internally.

New test data → M1 → 0.25, M2 → 0.20, M3 → 0.55 ∴ o/p = M3

- OVR breaks down multi-class problem into several binary classification problems.
- In above ex., OVR creates 3 separate classifiers:
  - class F1 vs Not F1 (i.e. F2 & F3)
  - class F2 vs Not F2 (i.e. F1 & F3)
  - class F3 vs Not F3 (i.e. F1 & F2)

Ex:
```
# importing
from sklearn.datasets import make_classification
# create dataset
x,y = make_classification(n_samples=1000, n_features=10,
                          n_classes=2, random_state=15)
```
make_classification: classification approach. It creates independent & dependent features. Creates standardized data so we need not do it.

```
pd.DataFrame(X)
y
```

⑦
```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
logistic.fit(x_train, y_train)
y_pred = logistic.predict(x_test)
print(y_pred)
logistic.predict_proba(x_test)
```
Shift+tab → few: 'penalty' (l1 = lasso, l2 = ridge } we can apply these as well); 'C' → inversely prop. to 'λ' in L1 & L2; 'class weight': useful when we are having imbalanced dataset, we can specify on which dataset we should mostly focus on.
(probability of outputs)

⑥
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
score = accuracy_score(y_test, y_pred)
print(score)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
# Hyperparameter Tuning
```
⑧
```
model = LogisticRegression()
params = {'penalty': ['l1','l2', ...],
          'C': [100, ...],
          'solver': ['saga', ...]}
```
```
from sklearn.model_selection import GridSearchCV, StratifiedKFold, RandomizedSearchCV
cv = StratifiedKFold()  → we can use k fold cv as well
```
⑨
```
grid = GridSearchCV(estimator=model, param_grid=params, scoring='accuracy', cv=cv,
                    n_jobs=-1)
grid.fit(x_train, y_train)
grid.best_params_ , grid.best_score_
y_pred = grid.predict(x_test)
```
⑥

```
# RandomSearch
```
⑨ → change: 1) grid → random cv 2) GridSearchCV → RandomizedSearchCV 3) cv=5

```
# For multiclass classification problem
x,y = make_classification(n_samples=1000, n_features=10, n_informative=3,
                          n_classes=3, random_state=1)
```
⑦ → change to: logistic = LogisticRegression(multi_class='ovr')

⑥

```
# For Imbalanced dataset
from collections import Counter
x,y = make_classification(n_samples=1000, n_features=2, n_clusters_per_class=1,
                          n_redundant=0, weights=[0.99], random_state=10)
```
(n_classes) → by def. 2 o/p features
n_clusters_per_class → controls how many groups are formed within each class.
No extra redundant features are added. All features are informative, no duplicates
99% of samples belong to class 0, making it imbalanced

#n_clusters_per_class = 1 → (fig) → (fig 2) → 4 clusters formed & data is not linearly seperable now!

>1

Counter(y)

sns.scatterplot(x=pd.DataFrame(x)[0], y=pd.DataFrame(x)[1], hue=y)

⓪

⑧ → added few more parameters

⑨

⑥

With ROC Curve and ROC AUC Score → Plot of TPR (y) vs FPR (x) at various threshold levels, showing a model's performance across diff. values

#Now "model.predict(x_test)" gives o/p in 0's & 1's. By definition, threshold val. = 0.5 means o/p = {1 → th > 0.5, 0 → th ≤ 0.5}. Now depending on different use cases, we need to change threshold value.

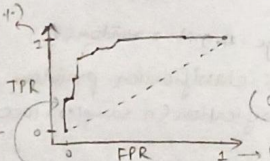
```
from sklearn.metrics import roc_curve, roc_auc_score
from matplotlib import pyplot
x,y = make_classification(n_samples=1000, n_classes=2, random_state=1)
⓪
dummy_model_prob = [0 for _ in range(len(y_test))]   → dummy model with default output as 0
model = LogisticRegression()
model.fit(x_train, y_train)
model_prob = model.predict_proba(x_test)   → gives probabilities of output feature
model_prob
model_prob = model_prob[:, 1]
dummy_model_auc = roc_auc_score(y_test, dummy_model_prob)
model_auc = roc_auc_score(y_test, model_prob)
print(...)
dummy_fpr, dummy_tpr, _ = roc_curve(y_test, dummy_model_prob)
model_fpr, model_tpr, thresholds = roc_curve(y_test, model_prob)
model_fpr, model_tpr, thresholds
pyplot.plt(dummy_fpr, dummy_tpr, linestyle='--', label='Dummy Model')
pyplot.plt(model_fpr, model_tpr, marker='.', label='Logistic')
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
pyplot.legend()
pyplot.show()
```

o/p: array([[0.8..., 0.2], ...]) → prob. for class 0; prob. for class 1 ← we are selecting the 2nd column

o/p: 0.5, 0.907674 → obvio. → since dummy, so half of them must be true

returns FPR, TPR, Threshold (for ignoring "auc")

(denote ↑)



AUC(Logistic) > AUC(dummy) → (area under curve)

(denote ↑)

There all points are based on some threshold values. Based on this value, we calculate FPR & TPR & that is plotted.

{We also have FNR & TNR}

After we plot this curve, we have to select a threshold which results in high TPR & low FPR. Once that point is detected, we will select that threshold for our Logistic Regression

```
# Display threshold values
fig = pyplot.figure(figsize=(20,50))
ax = fig.subplot(111)
for xyz in zip(model_fpr, model_tpr, thresholds):
    ax.annotate('%s' % np.round(xyz[2],2), xy=(xyz[0], xyz[1]))
```

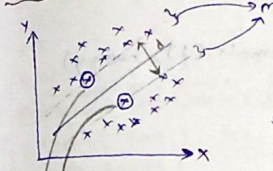
sample	actual class	predicted prob.	classification (Threshold = 0.8)
1	1	0.95	+
2	1	0.85	+
3	1	0.6	−
4	1	0.45	−
5	1	0.8	+
6	0	0.9	+
7	0	0.3	−

See above table for better understanding of ROC curve.
Also, threshold value is told by "domain expert".

SUPPORT VECTOR MACHINES

SVM is ML Algo. helps us solve both classification & regression problems.

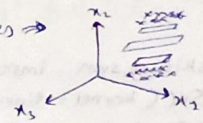
1) Support Vector Classifier (SVC):



→ created in such a way that distance (d) btw them is maximum

Support vectors } nearest points to marginal lines

if we had 3 categories ⇒



Hard Margin: → requires a strict seperation with no misclassifications.
→ suitable for linearly seperable data.

Soft Margin:

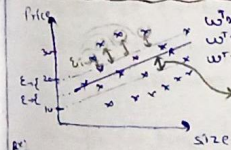
- Allows some misclassification, making it more flexible for non-linearly seperable data. In reality, there will always be misclassification.

Cost func. = $\min_{(w,b)} \frac{|w|}{2} + \underbrace{C \sum_{i=1}^{n} \xi_i}_{\text{HINGE LOSS}}$ (hyperparameter) — spelled as 'leita'

(Soft Margin)

(how many points we want to avoid misclassification) (summation of the distance of the incorrect data points from the marginal plane.)

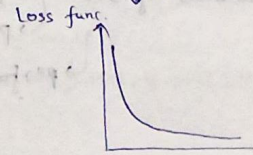
2) Support Vector Regression (SVR):



Cost func. = $\min_{(w,b)} \frac{|w|}{2} + C \sum_{i=1}^{n} \xi_i$ (hyperparameter) → HINGE LOSS

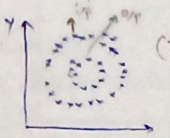
Constraint: $|y_i - w_i x_i| \leq \epsilon + \xi$ → Loss func. ≤ (margin error) + (error above the margin)

Note: $C \propto \frac{1}{\text{Loss func.}}$ ∴

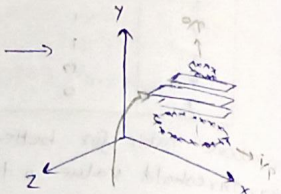


SVM Kernels

When datapoints are not linearly seperable, we use SVM kernels

ex: 

⟶ Transformations ⟶ (We apply some mathematical formulas)



Linear SVC ⇒ (Acc.↑)

We have different types of kernels:

- Polynomial kernel
- RBF kernel
- Sigmoid Kernel

Ex: (SVC Implementation)

```
# importing
X,y = make_classification(n_samples=1000, n_features=2, n_classes=2,
                          n_cluster_per_class=2, n_redundant=0)
```
→ Class 0 & Class 1

```
X, y
sns.scatterplot(x=pd.DataFrame(X)[0], y=pd.DataFrame(X)[1], hue=y)
# Points at 'x' & 'y' → point color based on class of 'hue'
```
→ class 0 → Blue colour
→ class 1 → Orange colour

⑨

```
from sklearn.svm import SVC
svc = SVC(kernel='linear')
svc.fit(X_train, y_train)
svc.coef_
y_pred = svc.predict(X_test)
```

⑤

Similarly trying for other kernels
- → rbf = SVC(kernel='rbf')
- → polynomial = SVC(kernel='poly')
- → sigmoid = SVC(kernel='sigmoid')

Hyperparameter Tuning

⑩

```
param_grid = {'C': [0.1, ...],
              'gamma': [1, ...],
              'kernel': ['rbf', ...]}
```
- controls the amount of info. to be displayed, in console during execution of grid search process.
- Higher values = more detailed logs

```
grid = GridSearchCV(SVC(), param_grid=param_grid, refit=True, cv=5, verbose=3)
grid.fit(X_train, y_train)
grid.best_params_
y_pred2 = grid.predict(X_test)
print(classification_report(y_test, y_pred2))
print(confusion_matrix(y_test, y_pred2))
```
- refit: means after finding the best combo. of hyperparams based on cross-validation, the model is retrained (refit) on the entire training dataset using those best hyperparams.

Ex: (SVM Kernels Indepth Explanation) → Refer Notebook

Basically → initial dataset ⇒ ↑(0)↑

→ Creating few new features & visualizing using "scatter_3d"

→ Using different kernels of SVC → fit → predict → performance metrics

Ex: (SVR Implementation)

```
# importing
df = sns.load_dataset('tips')
```
.head() , .info() ; df['col.name'].value_counts() → (checking for diff. columns)

```
df.columns
# independent & dependent features
X = df[['tip', 'sex', 'smoker', 'day', 'time', 'size']]
y = df[['total_bill']]
```

⑥

Feature Encoding

- Label encoding: If we have 2 values, we can substitute by 0 and 1.
- OHE: If >2 values, we can create those many features & wherever that specific value is, it will be =1 & rest all ='0'

(here 'days' col. has 4 values)

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le1 = LabelEncoder()
le2 = ''
le3 = ''
```
} applying Label Encoding to binary features

```
X_train['sex'] = le1.fit_transform(X_train['sex'])
X_train['smoker'] = le2.fit_transform(X_train['smoker'])
X_train['time'] = le3.fit_transform(X_train['time'])
X_train.head()
(...) ; replace 'X_train' with 'X_test'
from sklearn.compose import ColumnTransformer
```
→ (OHE applied with the help of this)

```
ct = ColumnTransformer(transformers=[('onehot', OneHotEncoder(drop='first'), [3])],
                       remainder='passthrough')
```
- (takes value in the form of tuples)
- ex: RGB. We drop 'R' col. which is first col. & now if both G=0 & B=0, we can say that it represents 'R' col.
- 'days' = 3rd index in table (starting from 0)
- (We do this for efficiency & it also avoids multicollinearity & avoids redundant columns without losing any info.)

```
import sys
import numpy as np
np.set_printoptions(threshold=sys.maxsize)
```
→ (Just a setting to see the entire dataset)

```
ct.fit_transform(X_train)
X_train = ct.fit_transform(X_train)
X_test = ct.transform(X_test)
from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train, y_train)
y_pred = svr.predict(X_test)
```
(# See 'Data Leakage' in Linear Regression Part)

⑦

Hyperparameter Tuning

⑩ → change 'SVC' to 'SVR()'

```
print(r2_score(y_test, y_pred2))
print(mean_absolute_error(y_test, y_pred2))
```

NAIVE BAYES THEOREM

Naive Bayes is used for "Classification" problem.

Bayes Theorem:

$\{P(A \text{ and } B) = P(A) * P(B_A)\}$ → $P(A/B) = \frac{P(A) * P(B/A)}{P(B)}$

P(A/B) = prob. of event A, given B has occured

P(A) = prob. of event A, P(B) = prob. of event B

P(B/A) = prob. of event B, given A has occured.

Variants of Naive Bayes:

1) **Bernoulli Naive Bayes:** Whenever your features are following a Bernoulli distribution, then we need to use Bernoulli Naive Bayes Algorithm.

→ Refer "Mathematics - Basics to Advanced for Data Science & GenAI" chapter

2) **Multinomial Naive Bayes:** when 'ip' is in text format

↓ converted to numerical values ↓ vectors ⇒ with the help of NLP
- ① Bow
- ② Tf-Idf
- ③ word2vec

3) **Gaussian Naive Bayes:** If the features are following gaussian distribution i.e. we have continuous values, then we use this algo.

↳ 'IRIS' dataset ~ is a continuous dataset

Ex:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
x, y = load_iris(return_X_y=True)
                 (X capital)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
print(confusion_matrix(y_pred, y_test))
print(accuracy_score(y_pred, y_test))
print(classification_report(y_pred, y_test))
```

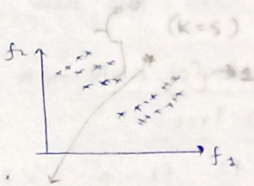
// Refer how to do feature engineering in a classification problem.

K NEAREST NEIGHBOUR ML ALGORITHM

KNN ML Algo. helps us solve both classification & regression problems.

1) Classification:

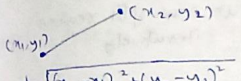
Steps:
1) Initialize 'K' value; (K>0)
   K = 1, 2, 3, ... ⇒ hyperparameter
2) Find the K Nearest Neighbours for the test data
3) From those, {K=5} how many neighbour belong to 0 & 1 category



↳ we find its 5 nearest neighbours & conclude on basis of that.

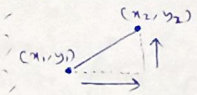
2 ways to calculate distance:

1) Euclidian Distance:



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

2) Manhattan Distance:



Ex:

```
# importing                                          3 features
x, y = make_classification(n_samples=1000, n_features=3, n_redundant=1,
                           n_classes=2, random_state=1)
                           binary target
```

①

```
from sklearn.neighbors import KNeighborsClassifier → refer doc. for more info.
classifier = KNeighborsClassifier(n_neighbors=5, algorithm='auto')
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
```

⑥

{Try Hyperparameter Tuning using GridSearchCV for diff. 'K' values}

2) Regression:

For test data, we calc. → avg. of all these 'K' nearest points to find the o/p.

↳ If we have a lot of outliers, we can go with median.

Ex:

```
from sklearn.datasets import make_regression
x, y = make_regression(n_samples=1000, n_features=2, noise=10,
                       random_state=4)
```

①

```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors=6, algorithm='auto')
regressor.fit(x_train, y_train)
y_pred = regressor.predict(x_test)
```

②

Issue

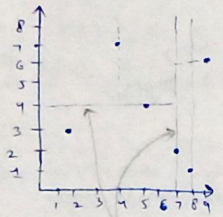
To find distance → $T_c = O(n)$ → if $n \sim$ million → High time complexity

To solve this issue, we have 2 optimization → KD Tree, Ball Tree

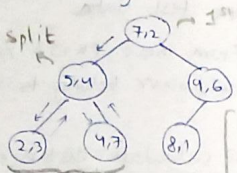
Variants of KNN → (to ↓ T_c)

1) KD Tree:

ex:	f1	f2
	7	2
	5	4
	9	6
	2	3
	4	7
	8	1
Median =	6·5	3·5
	↓	↓
(upper value which is there in dataset)	7	4



Binary Tree ~ KD Tree



(Points seperated by splits)

Here, we also backtrack while traversing

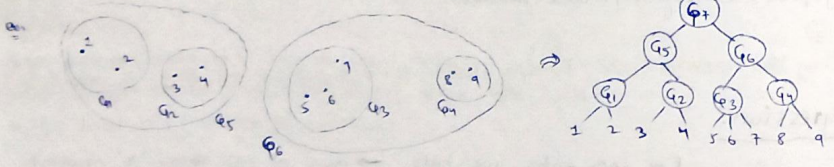
(we now split regions iteratively)

→ Now we split regions → this reduces search area for nearest neighbors.

(Points left of median go into one region ; points to the right go into another)

2) Ball Tree:

Recursively divide dataset into "balls", where each ball consists a subset of the data points.



DECISION TREE CLASSIFIER & REGRESSOR

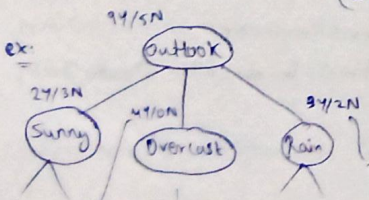
1) Classification:

Decision Tree Classifier → ID3 (Iterative Dichotomiser) → makes 'generic tree' →

Decision Tree Classifier → CART → makes 'binary tree' →

(sklearn uses CART)

ex:



→ Decision keeps happening until we get a leaf node.

(impure split)

(pure split) → (when only 'Y' or 'N' is present)

(leaf node)

- If we want to check its purity → (pure/impure split)

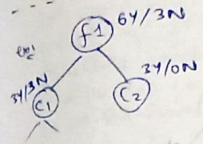
Entropy, Gini Impurity

- What feature we need to select for splitting → Info. gain

- Entropy: $H(S) = -P_+ \log_2(P_+) - P_- \log_2(P_-)$ } ex: → binary classification

P_+ : prob. of being positive

P_- : prob. of being negative



$H(C_1) = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) = 1 \Rightarrow$ Impure split

(Prob. of Y, Prob. of N)

$H(C_2) = -\frac{3}{3}\log_2\left(\frac{3}{3}\right) - 0 \cdot \log_2(0) = -1\log_2 1 - 0 \Rightarrow$ Pure split

- Gini Impurity: $G.I = 1 - \sum_{i=1}^{c} (p)^2$

$= 1 - (p_+)^2 + (p_-)^2)$ → For C1

$= 1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right)$

$= 0.5 \Rightarrow$ Impure split

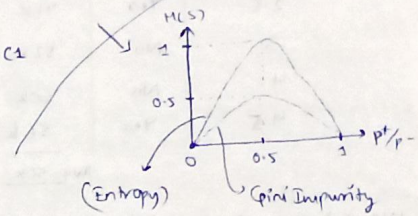
For C2:

$G.I = 1 - \left(\frac{3}{3}\right)^2$

$= 0$

↓

Pure split



(Value always btw 0 & 1)

*If 3 categories → o/p → (entropy) $H(S) = -P_{c_1}\log_2(P_{c_1}) - P_{c_2}\log_2(P_{c_2}) - P_{c_3}\log_2(P_{c_3})$

- Whenever dataset is small → use "Entropy" → (determine quality of split)

" " " large → use "Gini Impurity" (default by decision tree)

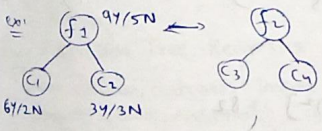
- Information Gain: (helps us understand which feature one should select & do the decision tree split

(feature)

$Gain(S, f_1) = H(S) - \sum_{v \in val} \frac{|S_v|}{|S|} H(S_v)$

(entropy of the root node)

ex:



→ $H(S) = -P_+\log_2(P_+) - P_-\log_2(P_-)$

$= -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log\left(\frac{5}{14}\right)$

≈ 0.94

$H(C_1) = -\frac{6}{8}\log\left(\frac{6}{8}\right) - \frac{2}{8}\log\left(\frac{2}{8}\right) \approx 0.81$

$H(C_2) = 1$

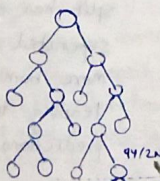
$\therefore Gain(S, f_1) = 0.94 - \left[\frac{8}{14} \times 0.81 + \frac{6}{14} \times 1\right]$

$= 0.049$

$Gain(S, f_2) = 0.051 > Gain(S, f_1) = 0.049$ → This indicates that I have to start splitting from feature f2.

- Post & Pre Pruning:

Pruning: prevents overfitting, where the model fits the training data too closely, leading to 'high variance' in test data.



→ (overfitting)

- Post-pruning: → Builds complete decision tree first, then prune necessary branches.

→ Effectively reduces overfitting

→ Best for smaller datasets

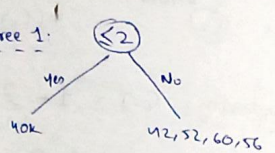
- **Pre-pruning:** → Limits the tree's growth during construction by setting hyperparameters like 'max_depth', etc.
  → Ideal for Large datasets

2) Regression:

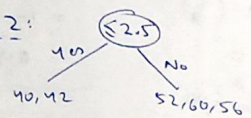
ex:

experience	gap	Salary
2	Yes	40k
2.5	Yes	42k
3	No	52k
4	No	60k
4.5	Yes	56k
		Avg = 50k

Decision Tree 1: (DT1)



Decision Tree 2:



We solve using "Variance Reduction" = $Var(parent) - \sum w_i \cdot var(child_i)$

- Variance = $\frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2$ → (MSE) (avg.)

$\therefore Var(root) = \frac{1}{5}\left[(40-50)^2 + (42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2\right] = 60.8$

For DT1: → $Var(c_1) = \frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2 = \frac{1}{1}(40-50)^2 = 100$ (child 1)

→ $Var(c_2) = \frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2 = \frac{1}{4}\left[(42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2\right]$

$= 51$

$\therefore$ Variance Reduction = $Var(root) - \sum w_i\, Var.(child)$

$= 60.8 - \left[\frac{1}{5}\times 100 + \frac{4}{5}\times 51\right] = 60.8 - 20 - 40.8 = 0$

$\therefore (VR)_1 = 0$

For DT2: → $Var(c_1) = \frac{1}{2}\left[(40-50)^2 + (42-50)^2\right] = 82$

→ $Var(c_2) = \frac{1}{3}\left[4 + 100 + 36\right] = 46.66$

$\therefore (VR)_2 = 60.8 - \left(\frac{2}{5}\times 82 + \frac{3}{5}\times 46.66\right) = +0.004$

$\therefore (VR)_1 < (VR)_2$ → $\therefore$ We select 2nd split → as it indicates that the split has effectively separated the data into more homogenous groups, leading to more accurate predictions.

So now → if test data → ≤2.5 = true → we will take avg.

i.e. o/p = $\frac{40+42}{2} = 41$

Ex: (Decision Tree Classifier)

```
# importing
from sklearn.datasets import load_iris
iris = load_iris()
print(iris['DESCR'])   → description
print(iris['target'])
# independent features
x = pd.Dataframe(iris['data'], columns = ['sepal length', 'sepal width', 'petal length', 'petal width'])
x
y = iris['target']
```

①

```
from sklearn.tree import DecisionTreeClassifier
treeclassifier = DecisionTreeClassifier()
treeclassifier.fit(x_train, y_train)
```

Post-pruning: we can set diff. parameters here. Ex: 'max_depth = 2'

```
# visualize
from sklearn import tree
```

(not affected by scale of features so no need to apply standardization)

```
plt.figure(figsize = (15,10))
tree.plot_tree(treeclassifier, filled = True)
```

ensures nodes in decision tree are coloured

```
y_pred = treeclassifier.predict(x_test)
```

because we have 3 distinct classes of iris flowers: Setosa, Versicolour & Virginica

⑥ → Conf. matrix = 3×3 = $\begin{bmatrix} TP_0 & FP_1 & FP_2 \\ FN_0 & TP_1 & FP_2 \\ FN_0 & FN_1 & TP_2 \end{bmatrix}$

```
# Pre pruning & Hyperparameter Tuning
param = {'criterion': ['gini', ...], 'splitter': ['best', ...], 'max_depth': [1,2, ...],
         'max_features': ['auto', ...]}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(treeclassifier, param_grid = param, cv = 5, scoring = 'accuracy')
import warnings
warnings.filterwarnings('ignore')
grid.fit(x_train, y_train)
grid.best_params_ , grid.best_score_
y_pred = grid.predict(x_test)
```

⑤

Ex: (Decision Tree Regressor → Diabetes Prediction)

```
from sklearn.datasets import load_diabetes
dataset = load_diabetes()
print(dataset['DESCR'])
dataset
df_diabetes = pd.Dataframe(dataset.data, columns = ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6'])
df_diabetes.head()
x = df_diabetes , y = dataset['target']
```

⑤

```
x_train.corr()
plt.figure(figsize = (15,10))
sns.heatmap(x_train.corr(), annot = True)
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
# Hyperparameter tuning
param → same as above one
grid = GridSearchCV(regressor, param_grid = param, cv = 5, scoring = 'neg_mean_squared_error')
[...]
```

// Plotting tree

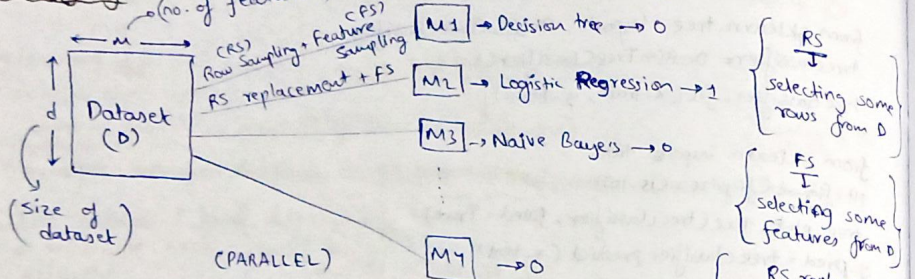
```
selected_model = DecisionTreeRegressor(criterion='squared_error', max_depth=4, max_features='log2', splitter='random')
selected_model.fit(x_train, y_train)
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(selected_model, filled=True)
```

(got from 'grid.best_params_')

RANDOM FOREST MACHINE LEARNING

Bagging: (IDEA)

(Base Learners)



RS / I — selecting some rows from D

FS / I — selecting some features from D

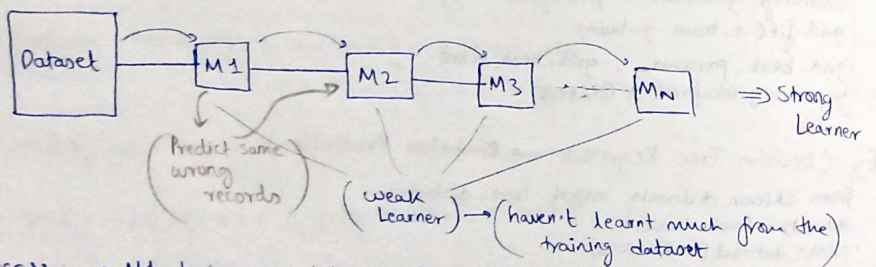
RS replacement → RS → removing some rows & adding new rows

If → classification ⇒ Majority voting algorithm

→ regression ⇒ average

1) AdaBoost 2) Gradient Boosting 3) Xgboost (Extreme Gradient Boost)

Boosting: (IDEA)



Process:
- → M1 trains on dataset. It predicts correct & wrong op's.
- → Now M1 passes those wrong records along with some training dataset to M2.
- → Above process repeats. (SEQUENTIALLY)

* Now in Random forest → we use all models (M1, M2, ... MN) = Decision Trees

Q) Why should we use Random forest instead of DT?

A)

Decision Tree → Overfitting → train acc. ↑ = Low bias; test acc. ↓ = High variance → Low bias, Low variance

Random Forest → Generalized model

Ex: (Random Forest Classification)

```
#importing
df = pd.read_csv('Travel.csv')
df.head()
# Feature Engineering → Data Cleaning → Handling Missing Values
df.isnull().sum()
df['Gender'].value_counts() → similarly check for other categories
features_with_na = [features for features in df.columns if df[features].isnull().sum() >= 1]
for feature in features_with_na:
    print(feature, np.round(df[feature].isnull().mean()*100,5), '% missing values')
# Statistics on numerical col
df[features_with_na].select_dtypes(exclude='object').describe()
# Imputing null values
df.Age.fillna(df.Age.median(), inplace=True)        → Age = median
df.TypeofContact.fillna(df.TypeofContact.mode()[0])  → TypeofContact = mode
                                                     → DurationOfPitch = median
                                                     → similar for others
df.isnull().sum()
df.drop('CustomerID', inplace=True, axis=1) → Not an imp. prop., so removing it.
#Feature Extraction
df['TotalVisiting'] = df['NumberOfPersonVisiting'] + df['NumberOfChildrenVisiting']
df.drop(columns=['(col. name)'], axis=1, inplace=True)
df.shape
# Get all numeric features
num_features = [feature for feature in df.columns if df[feature].dtype != 'O']
print(num_features)
# Categorical features
cat_features = [feature for feature in df.columns if df[feature].dtype == 'O']
print(cat_features)
# Discrete features
discrete_features = [feature for feature in num_features if len(df[feature].unique())<=25]
print(discrete_features)
# Continuous features
continuous_features = [feature for feature in num_features if feature not in discrete_features]
print(continuous_features)
# Independent & dependent features
x = df.drop(['ProdTaken'], axis=1)
y = df['ProdTaken']
x.head(), y.value_counts()
```

(used to access the 1st/most frequent val.)

(refer notebook)

```
x.info()
cat_features = x.select_dtypes(include="object").columns
num_features = x.select_dtypes(include="object").columns
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder(drop='first')
preprocessor = ColumnTransformer([
    ("OneHotEncoder", oh_transformer, cat_features),
    ("StandardScaler", numeric_transformer, num_features)
])
preprocessor
```

(used to apply different preprocessing steps to specified columns of a dataset in a single step)

→ [Refer "Support Vector Machines"]

```
x_train = preprocessor.fit_transform(x_train)
pd.DataFrame(x_train)
x_test = preprocessor.transform(x_test)
# Importing: RandomForestClassifier, GradientBoostingClassifier, DecisionTreeClassifier, LogisticRegression
#            sklearn.ensemble                                    sklearn.tree            sklearn.linear_model

models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boost": GradientBoostingClassifier()
}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(x_train, y_train)
    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)
    print(list(models.keys())[i])
    print("Model performance for training set")
    # printing -> Accuracy, F1 score, Precision, Recall, Roc Auc Score
    print("Model performance for test set")
    # printing -> (")
    print('\n')
# Hyperparameter training
rf_params = {"max_depth": [5, ...],
             "max_features": [5, ...],
             "min_samples_split": [2, ...],
             "n_estimators": [100, ...]}
randomcv_models = [("RF", RandomForestClassifier(), rf_params)]
model_param = {}
for name, model, params in randomcv_models:
    random = RandomizedSearchCV(estimator=model, param_distributions=params,
                                n_iter=100, cv=3, verbose=2, n_jobs=-1)
    random.fit(x_train, y_train)
    model_param[name] = random.best_params_
for model_name in model_param:
    print(f"Best Params for {model_name}")
    print(model_param[model_name])
models = {"Random Forest": RandomForestClassifier(...)}   # Best Params from above op
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(x_train, y_train)
    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)
    print(list(models.keys())[i])
    print("Model performance for training set")
    # printing -> Accuracy, F1 score, Precision, Recall, Roc Auc Score
    print("Model performance for test set")
```

(11) Refer Notebook

(12)

(13)

```
# plot ROC AUC Curve
from sklearn.metrics import roc_auc_score, roc_curve
plt.figure()
auc_models = [{                    # add models to the list that you want to view on ROC plot
    "label": "Random Forest Classifier",
    "model": RandomForestClassifier(...),   -> best params
    "auc": 0.8424
}]
for algo in auc_models:
    model = algo['model']
    model.fit(x_train, y_train)
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,1])
    plt.plot(fpr, tpr, label='%s ROC Curve = %0.2f' % (algo['label'], algo['auc'])
# custom settings for plt -> REFER NOTEBOOK
```

Ex: (Random Forest Regressor)

```
# importing
df = pd.read_csv('cardekho.csv', index_col=[0])
df.head()
# Data Cleaning
df.isnull().sum()
df.drop('car_name', axis=1, inplace=True)
df.drop('brand', axis=1, inplace=True)
# Getting all different types of features
num_features = [feature for feature in df.columns if df[feature].dtype != 'O']
                (numerical)
cat_features = [feature for feature in df.columns if df[feature].dtype == 'O']
               (categorical)
discrete_features = [feature for feature in num_features if len(df[feature].unique())
                                                            <= 25]
continuous_features = [feature for feature in num_features if feature not in
                                                            discrete_features]
# Independent & dependent features
x = df.drop(['selling_price'], axis=1)
y = df['selling_price']
# Feature Encoding & Scaling
df['model'].value_counts()   (len = 120)
                             -> we perform Label Encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x['model'] = le.fit_transform(x['model'])
x.head()
len(df['seller_type'].unique()), len(df['fuel_type'].unique()), len(df['transmission_type']
                                                                    .unique())
# 3, 5, 2 -> we perform One Hot Encoding
# ColumnTransformer
num_features = x.select_dtypes(exclude="object").columns
onehot_columns = ['seller_type', 'fuel_type', 'transmission_type']
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
```

We use Label Encoding to assign numerical labels to categorical data when we want the model to capture potential relationships btw labels & the target variable. (Ex: selling price). This allows the model to identify patterns where specific labels correlate with higher/lower target values

```
numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder(drop='first')
preprocessor = ColumnTransformer([
                ("OneHotEncoder", oh_transformer, onehot_columns),
                ("StandardScaler", numeric_transformer, num_features)
            ], remainder='passthrough')
X = preprocessor.fit_transform(X)
pd.DataFrame(X)
```

⑩

Model Training & Selection

Importing: RandomForestRegressor (sklearn.ensemble), KNeighborsRegressor (sklearn.neighbors), Ridge, Lasso (sklearn.linear_model), DecisionTreeRegressor (sklearn.tree)

⑪
- initialize above models
- for performance → use: "RMSE", "MAE", "R2 score"
- rest all same.

Hyperparameter Tuning

```
knn_params = {"n_neighbors": [2, 3, 10, 20, 40, 50]}
rf_params → Same as prev. example
random_models = [("KNN", KNeighborsRegressor(), knn_params),
                 ("RF", RandomForestRegressor(), rf_params)]
```

⑫
⑬
- add KNN in 'models' as well. Use best_params_ from above cell ⑩.
- for performance → use: "RMSE", "MAE", "R2 score"
- rest all same

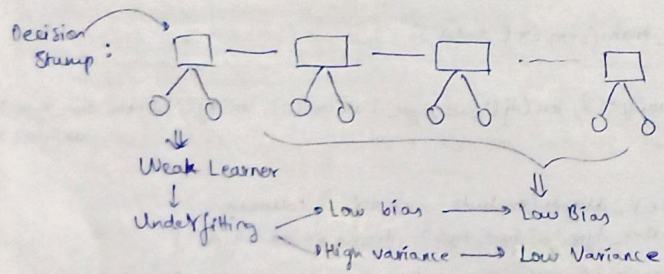
ADABOOST MACHINE LEARNING ALGORITHM

Ensemble Techniques

Bagging
1) Random Forest Classifier
2) Random Forest Regressor

Boosting
1) Adaboost
2) Gradient Boosting
3) Xgboost

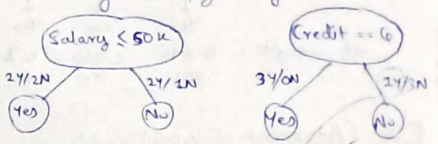
- **Adaboost**: We assign "weights" to weak learners.
  - func.: $f = \alpha_1(M_1) + \alpha_2(M_2) + \dots + \alpha_n(M_n)$; $\{\alpha_1, \alpha_2, \dots \alpha_n\}$: weights, $\{M_1, M_2, \dots M_n\}$: Decision Tree Stumps



- **Maths Intuition**

Salary	Credit	Approval	Sample wt
<=50K	B	No	1/7
<=50K	G	Yes	1/7
<=50K	G	Yes	1/7
<=50K	B	No	1/7
>50K	G	Yes	1/7
>50K	N	Yes	1/7
>50K	N	No	1/7
<=50K			

① We create decision tree stump & we select the best DT stump using entropy (or) gini index.



When credit == G, we have all 'Y'. Here we are getting 'Y' when credit != G, ∴ we can say this is an error.

② Sum of total errors & performance of stump ⟷

(2.1) Sum of all the total error = $\frac{1}{7}$ = (TE)

(2.2) Performance of Stump = $\frac{1}{2} \ln\left[\frac{1 - TE}{TE}\right]$

$= \frac{1}{2} \ln\left(\frac{1 - 1/7}{1/7}\right) \approx 0.896$

$\therefore f = \alpha_1(M_1) + \alpha_2(M_2) + \dots + \alpha_n(M_n)$

$(0.896) \rightarrow$ Weight

③ Update the weights for correctly & incorrectly classified points

(-) Whichever is correctly classified, we reduce their weights & whichever is incorrectly classified, we increase their weights. We do this because there will be a high probability of selecting the wrong records by the next decision tree stump.

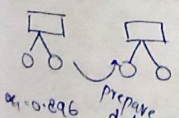
- For correct classified points:
  - = weight * $e^{-(\text{performance of stump})}$
  - = $\frac{1}{7} * e^{-(0.896)}$ = 0.058
- For incorrect classified points:
  - = weight * $e^{(\text{performance of stump})}$
  - = $\frac{1}{7} * e^{(0.896)}$ = 0.349

④ Normalized Weights Computation & Assigning Bins

Salary	Credit	Approval	Update Wts	Normalized Wts	Bins Assignment
<=50K	B	N	0.058	0.08	0 - 0.08
<=50K	G	Y	0.058	0.08	0.08 - 0.16
<=50K	G	Y	0.058	0.08	0.16 - 0.24
>50K	B	N	0.058	0.08	0.24 - 0.32
>50K	G	Y	0.058	0.08	0.32 - 0.40
→>50K	N	Y	0.349	0.50	0.40 - 0.90
<=50K	N	N	0.058	0.08	0.90 - 0.98
			sum = 0.697	≈1	

(But sum must be 1, so we normalize it) → (divide by 0.697)

⑤ Select data points to send to next stump



(5.1) Iterative process

(selecting random values btw 0 & 1)

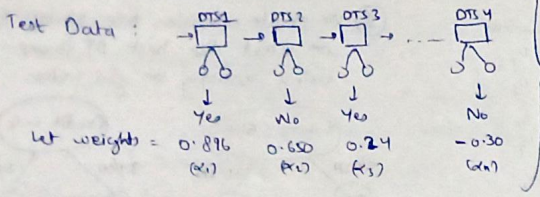
We get a random no., see check in which 'bin' it lies, then we select that record.

Note: Rows can be repeated.

⑥ These records will be sent to next DT Stump.

We will get 'α_2'

④ Final Prediction



Ex: (Adaboost Classification)

↓

Refer "Random Forest Classification" example. Keep everything same. Just add "from sklearn.ensemble import AdaBoostClassifier" while training & checkout the results. You may also hyperparameter tune it. {Refer docs}

Ex: (AdaBoost Regression)

↓

Refer "Random Forest Regressor" example. Keep everything same. Just add "from sklearn.ensemble import AdaBoostRegressor" while training & checkout the results. You may also hyperparameter tune it. {Refer docs}

GRADIENT BOOSTING

Steps: 1) Create a base model
2) Compute residuals, error → (r_i) (Ex: compute avg. & then subtract y_i) (org; op)
3) Construct a decision tree considering $ip = x_i$ & $op = r_i$

$\therefore F(x) = \alpha_0(h_0(x)) + \alpha_1(h_1(x)) + \dots + \alpha_n(h_n(x))$; $\alpha_i \rightarrow$ learning rate

$F(x) = \sum_{i=0}^{\hat{}} \alpha_i(h_i(x))$ → final func. of gradient boosting

Ex: (GradientBoost Classification)

↓

Refer "Random forest Classification" example. Keep everything same. Just add "from sklearn.ensemble import GradientBoostingClassifier" while training & checkout the results. You may also hyperparameter tune it. {Refer docs}

Ex: (GradientBoost Regressor)

↓

Refer "Random Forest Regressor" example. Keep everything same. Just add "from sklearn.ensemble import GradientBoostingRegressor" while training & checkout the results. You may also hyperparameter tune it. {Refer docs}

XGBOOST MACHINE LEARNING ALGORITHM

1) Classification:

Steps: 1) Construct a base model such that it is not biased towards anything. {log(odds) = $\log\left(\frac{p}{1-p}\right)$}
2) Construct a decision tree with root.
3) Calculate similarity weight = $\frac{\sum(\text{residual})^2}{\sum p_i(1-p_i) + \lambda}$ (cover value) → Hyperparameter
4) Calculate Gain
5) Wrt finding final op, we apply sigmoid activation func.

Ex: Refer "Random Forest Classification" example. Keep everything same. Just add "from xgboost import XGBClassifier" while training & checkout the results. {If not present → !pip install xgboost} You may also hyperparameter tune it. {Refer docs}

2) Regressor:

Steps: 1) Create a base model.
2) Residual Computation.
3) Construct Decision Tree 1 using $\{x_i, R_i\}$

Ex: Refer "Random Forest Regressor" example. Keep everything same. Just add "from xgboost import XGBRegressor" while training & checkout the results. You may also hyperparameter tune it. {Refer docs}

UNSUPERVISED MACHINE LEARNING

Unsupervised ML → Clustering : group your data into similar clusters.

1) K Means Algo. 2) Hierarchical Clustering 3) DBScan Clustering 4) Silhouette Clustering

In supervised ML → we have o/p features / dependent features, but in unsupervised ML → we have to cluster our data.

PCA

Refer "Application of Linear Algebra in Dimensionality Reduction" → REV. CHAPTER

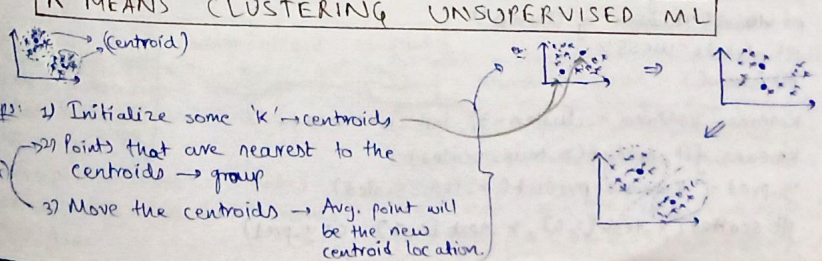
Ex:

```
#importing
from sklearn.datasets import load_breast_cancer
cancer_dataset = load_breast_cancer()
print(cancer_dataset.DESCR), cancer_dataset.keys()
df = pd.DataFrame(cancer_dataset['data'], columns = cancer_dataset['feature_names'])
df.head()
from sklearn.preprocessing import StandardScaler #Standardization
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)
#Applying PCA Algorithms
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
data_pca = pca.fit_transform(scaled_data)
data_pca → 2 features
pca.explained_variance_
plt.scatter(data_pca[:,0], data_pca[:,1], c=cancer_dataset['target'])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

(helps in giving diff color to points based on 'target' col. features) ↓ (ex: if Y/N 2 colors)

K MEANS CLUSTERING UNSUPERVISED ML

Ex:



Steps: 1) Initialize some 'k' → centroids
2) Points that are nearest to the centroids → group
3) Move the centroids → Avg. point will be the new centroid location.
(repeat)

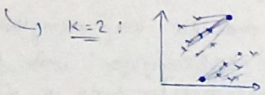
- How do we select K value?

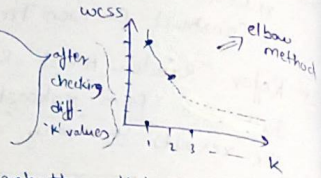
⇒ WCSS = within cluster sum of squares

⇒ Initialize K=1 to 20

(,

$$wcss = \sum_{i=1}^{n} (\text{distance btw points to nearest centroid})^2$$



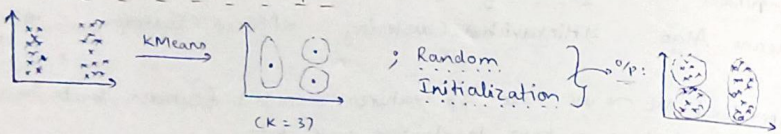


⇒ Euclidian dist. = $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ (default)

; Manhattan dist. = $|x_2-x_1| + |y_2-y_1|$

- Random Initialization Trap :

ex:



To resolve this mismatch, we use "KMeans ++ Initialization Technique"

↓

{ it initializes the centroid in such a way that it's completely at a farther distance between each other.

⇒ For new 'test' data point, we find the closest distance btw point & centroid, and then group with closest centroid.

Ex: # importing

```
from sklearn.datasets import make_blobs
x, y = make_blobs(n_samples=1000, centers=3, n_features=2)
x
y
plt.scatter(x[:,0], x[:,1], c=y)
```

(3)

(1.1)

```
from sklearn.cluster import KMeans
# Elbow method to select K value
wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(x_train_scaled)
    wcss.append(kmeans.inertia_)
wcss
# Plot elbow curve
plt.plot(range(1,11), wcss)
plt.xticks(range(1,11))
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()
```

{ decide cluster value from the graph → { the point where there is not much change in 'y' is the ideal point.

```
Kmeans = KMeans(n_clusters=3, init="k-means++")
Kmeans.fit_predict(x_train_scaled)
y_pred = Kmeans.predict(x_test_scaled)
plt.scatter(x_test[:,0], x_test[:,1], c=y_pred)
```

To validate 'K' value, we have 2 more methods : kneelocator, silhouette scoring

kneelocator

```
!pip install kneed
from kneed import KneeLocator
kl = KneeLocator(range(1,11), wcss, curve="convex", direction="decreasing")
kl.elbow
```

→ We can validate our elbow point from this

(From prev. graph → convex: ∪, concave: ∩ (since it's "kind of convex, so curve=convex"))

(prev. curve is ∪, so we write, dir="↓")

Silhouette Scoring

```
from sklearn.metrics import silhouette_score
silhouette_coefficients = []
for k in range(2,11):  # We start at 2 clusters for silhouette coefficients (for meaningful evaluation)
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(x_train_scaled)
    score = silhouette_score(x_train_scaled, kmeans.labels_)
    silhouette_coefficients.append(score)
```

(We need to give 2 parameters)

[wrt kmeans what label we have got i.e to which cluster that specific datapoint belongs to.]

silhouette_coefficients

Plotting silhouette score (highest peak = no. of clusters)

```
plt.plot(range(2,11), silhouette_coefficients)
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficients")
plt.show()
```

HIERARICHAL CLUSTERING

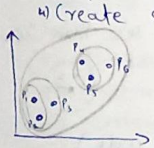
(Here we don't consider centroids)

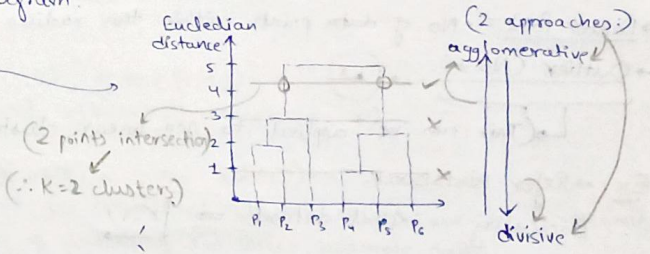
Steps: 1) Each point will be considered as a seperate cluster.

2) Find the nearest point & create a new cluster.

3) Keep on doing the same process until we get a single cluster.

4) Create a dendrogram.





- K-Means vs Hierarichal

1) Dataset size → Huge = K-Means; Small = Hierarichal Clustering

(Select the longest vertical line such that no horizontal line passes through it.)

2) KMean → Numerical Data
HC → variety of data (can use 'cosine similarity')

3) KMeans = Elbow Method = no. of centroids
HC = no. of clusters

Ex: # importing

```
from sklearn import datasets
iris = datasets.load_iris()
iris.keys()
iris_data = pd.DataFrame(iris.data)
iris_data.columns = iris.feature_names
iris_data
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(iris_data)
# Apply PCA = reducing the dimensions
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_scaled = pca.fit_transform(x_scaled)
plt.scatter(pca_scaled[:,0], pca_scaled[:,1], c=iris.target)
```

Agglomerative Clustering → constructing a dendrogram

```
import scipy.cluster.hierarchy as sc
plt.title("Dendrograms")
sc.dendrogram(sc.linkage(pca_scaled, method='ward'))
plt.xlabel('Sample Index')
plt.ylabel('Euclidean Distance')
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, metric='euclidean', linkage='ward')
cluster.fit(pca_scaled)
cluster.labels_
plt.scatter(pca_scaled[:,0], pca_scaled[:,1], c=cluster.labels_)
```

(Refer docs to see diff. methods)

Silhouette Scoring

```
from sklearn.metrics import silhouette_score
silhouette_coefficients = []
for k in range(2,11):
    agglo = AgglomerativeClustering(n_clusters=k, metric='euclidean', linkage='ward')
    agglo.fit(x_scaled)
    score = silhouette_score(x_scaled, agglo.labels_)
    silhouette_coefficients.append(score)
plt.plot(range(2,11), silhouette_coefficients)
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.show()
```

(We start at 2 clusters for silhouette coeff.)

DBSCAN CLUSTERING

→ **Core Point**: • No. of points within the 'ε', should be greater ≥ minpts.

(Let minpts = 4, ε = radius; other datapoints)

→ **Border Point**: • No. of data points within this radius will be less than minpts.

→ **Outlier (Noise)**:

(This can be applied to non-linear clustering as well.)

Ex: → Refer Notebook → (similar way written in SVM kernels)

Basically → initial dataset ⇒ → standardizing & applying DBSCAN. (from sklearn.cluster import DBSCAN)

SILHOUETTE CLUSTERING

Steps: 1) Create a specific cluster & then wrt a particular point calculate avg. distance.

$$a_i = \frac{1}{|C_i|-1} \sum_{j \in C_i, i \neq j} d(i,j)$$

(avg. distance a_i; $|C_i|$ = No. of points belonging to cluster 'i')

2) We take another cluster & then from that same point, we try to calculate the avg. distance to the other points in the 2nd cluster.

$$b_i = \min_{j \neq i} \frac{1}{|C_j|} \sum_{j \in C_j} d(i,j)$$

3) $$s(i) = \begin{cases} 1 - a(i)/b(i) & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1 & \text{if } a(i) > b(i) \end{cases}$$

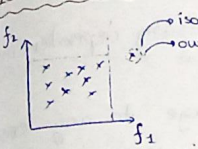
$\therefore -1 \leq s(i) \leq 1$

(More near to +1, better clustering model we have created.)

ANOMALY DETECTION MACHINE LEARNING ALGORITHMS

1) Isolation Forest:

(Internally it creates many decision trees which are known as "isolated trees" wrt each datapoint.)



Anomaly Score: $S(x,n) = 2^{\left(\frac{-E(h(x))}{c(n)}\right)}$

- n = no. of data points
- x = data point
- E(h(x)) = average search depth for x from the isolated tree
- c(n) = average depth of h(x)

→ $E(h(x)) << c(n) \Rightarrow S(x,n) \approx 1 \Rightarrow$ Anomaly Score = OUTLIERS

→ $E(h(x)) >> c(n) \Rightarrow S(x,n) \approx 0.5 \Rightarrow$ Normal data point

Ex:

```
df = pd.read_csv("healthcare.csv")
plt.scatter(df.iloc[:,0], df.iloc[:,1])
from sklearn.ensemble import IsolationForest
clf = IsolationForest(contamination = 0.2)
clf.fit(df)
predictions = clf.predict(df)
index = np.where(predictions < 0)   → Outliers
index
x = df.values
plt.scatter(df.iloc[:,0], df.iloc[:,1])
plt.scatter(x[index,0], x[index,1], edgecolors='r')
```

	0	1
0	1.617	1.944
1	1.256	1.609
2	-2.349	4.392

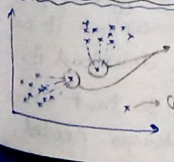
2) DBSCAN Clustering:

Advantages:
- No need for predefined cluster count (unlike K-means).
- Capable of clustering non-linear separable data.
- Robust to outliers: efficiently detects & excludes outliers.
- Few parameters needed (like 'ε' & min. pts)
- Fast region queries

Disadvantages:
- Border points can belong to multiple clusters, causing ambiguity.
- Sensitive to parameter selection.
- Dependency on distance measures (i.e. Euclidean/Manhattan).

3) Local Outlier Factor Anomaly Detection:

(LOF)



(as they're near to a specific cluster)

- LOF → uses KNN internally
- We will take first 'k' values from a point, then we take its k neighbours of those k points & we check the density of points

- Locality is given by KNN, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers.

DOCKER

Containers:

- A way to package application with all the necessary dependencies & configuration.
- Portable artifact; easily share & move this package to any env..
- Makes development & deployment more easy & efficient.

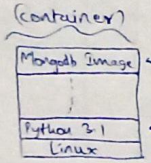
⇒ IMAGE — CONTAINER

- Package/Artifact
- Move/Share this artifact

- Docker image → run → starts the application by creating a container (installs all dependencies & makes an env.)

(container)

or.



(Refer: Docker vs Virtual Machine)

⇒ Docker - Compose: a tool that helps us define & share multi-container applications.

(docker-compose.yml)

⇒ Volumes: preferred mechanism for persisting data generated by & used by docker containers.

PROJECTS

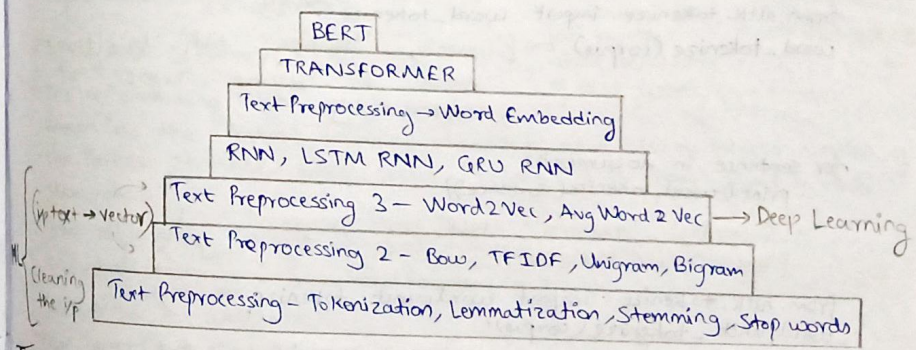
Project 1:

1) Created a repository on Github.
   - In VS code terminal; → conda create -p venv python==3.8 -y (Creates venv folder)
     → conda activate venv/
2) Create .gitignore file.
3) Create setup.py, requirements.txt (pip install -r requirements.txt)
   - setup.py = responsible for creating my ML application as a package. Now we can also install this package in different projects. After making it as a package, we can upload it on Python Pypi and from there anybody can download it.
   - Create src folder & add this file "__init__.py", so that when find_packages() from setup.py searches all the packages needed, it will consider the src folder as a package.
   - Write setup.py
   - In requirements.txt, adding '-e', so that it automatically triggers setup.py
4) Create components folder & create __init__.py file.
   (modules that we are going to specifically use in the project)
5) Creating these files inside components folder: → data_ingestion.py, → data_transformation.py, → model_trainer.py
6) Create pipeline folder. Create these files in this folder: → __init__.py, → train_pipeline.py, → predict_pipeline.py
7) Create logger.py, exception.py, utils.py in src folder. Write own custom exception.py. Write logger.py to log the exceptions which has occurred.
8) Create notebook folder & refer the dataset & both the .ipynb files.
9) Write → data_ingestion.py → data_transformation.py → model_trainer.py
10) Write app.py & predict_pipeline.py
11) Deploying on AWS, Azure.

* Refer: MLFlow, BentoML, DagsHub !

NLP FOR MACHINE LEARNING

Overview:



Tokenization:

- CORPUS = Paragraph
- DOCUMENTS = Sentences
- VOCABULARY = Unique Words
- WORDS = words

Tokenization: process wherein we convert paragraphs into sentences (or sentences into words). (aka Tokens)