Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200182

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5													
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5													
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5													
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4													
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–													
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4													
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–													
Total	21	22	22	23													
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

PLOT NO. U-15, JVPD SCHEME, BHAKTIVEDANTA SWAMI MARG, VILE PARLE (WEST), MUMBAI - 400056
Tel.: 42335000/42335001 Email : info@djsce.ac.in / admin@djsce.ac.in Website : www.djsce.ac.in

Machine Learning

Experiment 5

Ayush Jain 60004200132 B3

Aim : To implement KNN in python without using any inbuilt function and without inbuilt function.

Theory:

A **classification problem** has a discrete value as its output. For example, “likes pineapple on pizza” and “does not like pineapple on pizza” are discrete. There is no middle ground. The analogy above of teaching a child to identify a pig is another example of a classification problem.

Age	Likes Pinapple on Pizza
42	1
65	1
50	1
76	1
96	1
50	1
91	0
58	1
25	1
23	1
75	1
46	0
87	0
96	0
45	0
32	1
63	0
21	1
26	1
93	0
68	1
96	0

This image shows a basic example of what classification data might look like. We have a predictor (or set of predictors) and a label. In the image, we might be trying to predict whether someone likes pineapple (1) on their pizza or not (0) based on their age (the predictor).

It is standard practice to represent the output (label) of a classification algorithm as an integer number such as 1, -1, or 0. In this instance, these numbers are purely representational. Mathematical operations should not be performed on them because doing so would be meaningless. Think for a moment. What is "likes pineapple" + "does not like pineapple"? Exactly. We cannot add them, so we should not add their numeric representations.

A **regression problem** has a real number (a number with a decimal point) as its output. For example, we could use the data in the table below to estimate someone's weight given their height.

Height(Inches)	Weight(Pounds)
65.78	112.99
71.52	136.49
69.40	153.03
68.22	142.34
67.79	144.30
68.70	123.30
69.80	141.49
70.01	136.46
67.90	112.37
66.78	120.67
66.49	127.45
67.62	114.14
68.30	125.61
67.12	122.46
68.28	116.09

Data used in a regression analysis will look similar to the data shown in the image above. We have an independent variable (or set of independent variables) and a dependent variable (the thing we are trying to guess given our independent variables). For instance, we could say height is the independent variable and weight is the dependent variable.

Also, each row is typically called an **example, observation, or data point**, while each column (not including the label/dependent variable) is often called a **predictor, dimension, independent variable, or feature.**

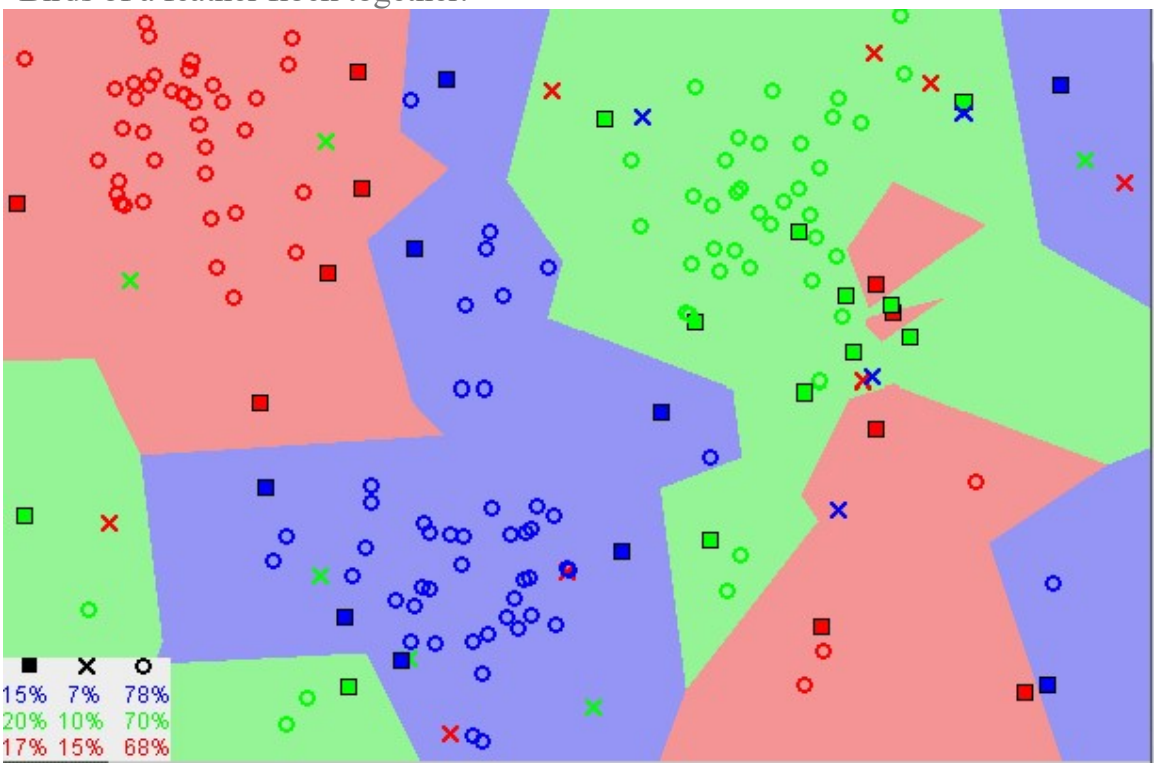
An **unsupervised machine learning** algorithm makes use of input data without any labels —in other words, no teacher (label) telling the child (computer) when it is right or when it has made a mistake so that it can self-correct.

Unlike supervised learning that tries to learn a function that will allow us to make predictions given some new unlabeled data, unsupervised learning tries to learn the basic structure of the data to give us more insight into the data.

K-Nearest Neighbors

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

"Birds of a feather flock together."



Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

Code without inbuilt function:

```
import numpy as np
from collections import Counter

class KNN:

    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_distance(self, X):
        return np.sqrt(np.sum((self.X_train - X)**2, axis=1))

    def predict(self, X):
        y_pred = []

        for i in range(len(X)):
            distances = self.euclidean_distance(X[i])
            indices = np.argsort(distances)[:self.k]
            k_nearest_labels = [self.y_train[idx] for idx in indices]
            most_common_label = Counter(k_nearest_labels).most_common(1)[0][
0]
            y_pred.append(most_common_label)

        return np.array(y_pred)
```

```
InterviewS = [70, 70, 30, 10]
ExamR = [70, 40, 40, 40]
TypeH = ['Not Hired', 'Hired', 'Not Hired', 'Not Hired']

X = np.array(list(zip(InterviewS, ExamR)))
y = np.array(TypeH)

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=42)

knn = KNN(k=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Calculate the accuracy of the model
# accuracy = accuracy_score(y_test, y_pred)
# print(f"Accuracy: {accuracy:.2f}")

knn = KNN(k=3)
knn.fit(X, y)

# Input new data to classify
new_data = np.array([[80, 40]])

predicted_label = knn.predict(new_data)

print(f"Predicted label: {predicted_label[0]}")
```

Output:

```
Predicted label: Not Hired
```

Code with inbuilt function:

```
import numpy as np

InterviewS = [90, 60, 70, 50, 80, 70, 60, 40, 90, 80]
ExamR = [80, 50, 60, 40, 90, 50, 70, 60, 60, 70]
```

```
TypeH = ['Hired', 'Not Hired', 'Hired', 'Not Hired', 'Hired', 'Not Hired', 'Hired', '
Not Hired', 'Hired', 'Hired']

X = np.array(list(zip(InterviewS, ExamR)))
y = np.array(TypeH)

from sklearn.neighbors import KNeighborsClassifier

# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the classifier to the data
knn.fit(X, y)

# Predict if a person with interview score 70 and exam rank 80 will be hired or
not
X_new = np.array([[70, 80]])
prediction = knn.predict(X_new)

print("Prediction:", prediction)
```

Output:

```
Prediction: ['Hired']
```

Conclusion: We implemented KNN in python without using any inbuilt function and without inbuilt function.