Chaitanya Machhindra Nawale
BECOB215
ML Assignment No. 2

PAGE NO.
DATE :

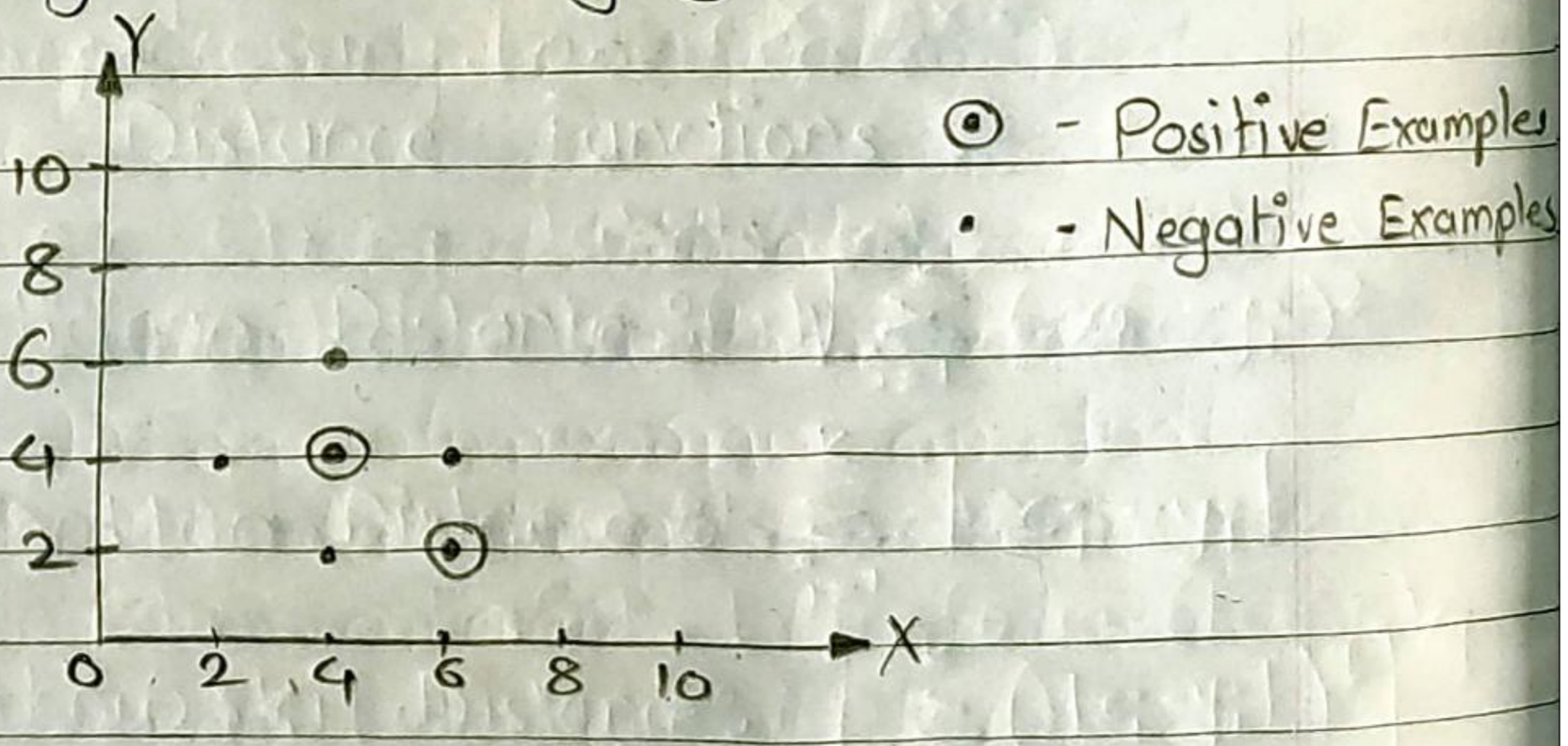
* Title :- Implementation of K-Nearest Neighbour technique.

* Objective :-
  - To learn K-Nearest Neighbour to solve problems.

* Outcome :- To apply K-Nearest Neighbour to solve the real world problems.

* Software Requirements :-
  Python 3, Jupyter Notebook, scikit-learn

* Problem Statement :-
  In the following diagram let blue circles indicate negative examples. We want to use k-NN algorithm for classifying the points. If K=3, find the class of the point (6,6). Extend the same example for Distance weighted k-NN and Locally weighted averaging



BECOB215

PAGE NO.
DATE :

* Theory :-

KNN :-

K nearest neighbours is a simple algorithm that stores all ~~variables~~ available cases and classifies new cases based on a similarity measure (e.g. distance functions). In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k=1, then the object is simply assigned to the class of that single nearest neighbor.

Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance.

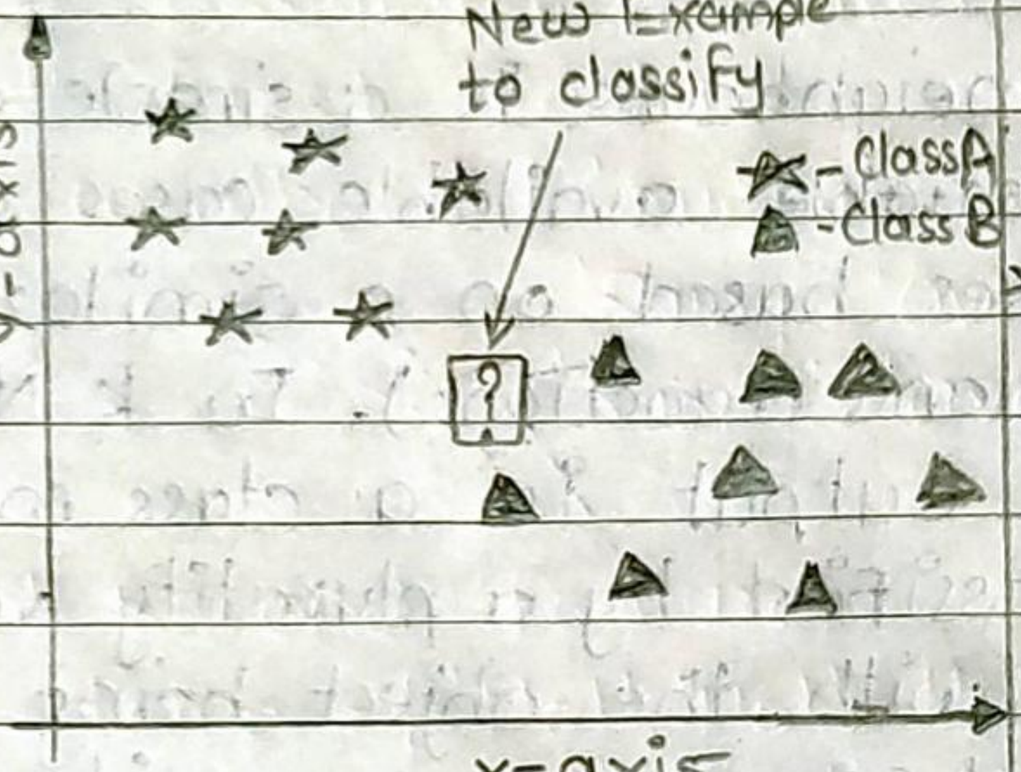
KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

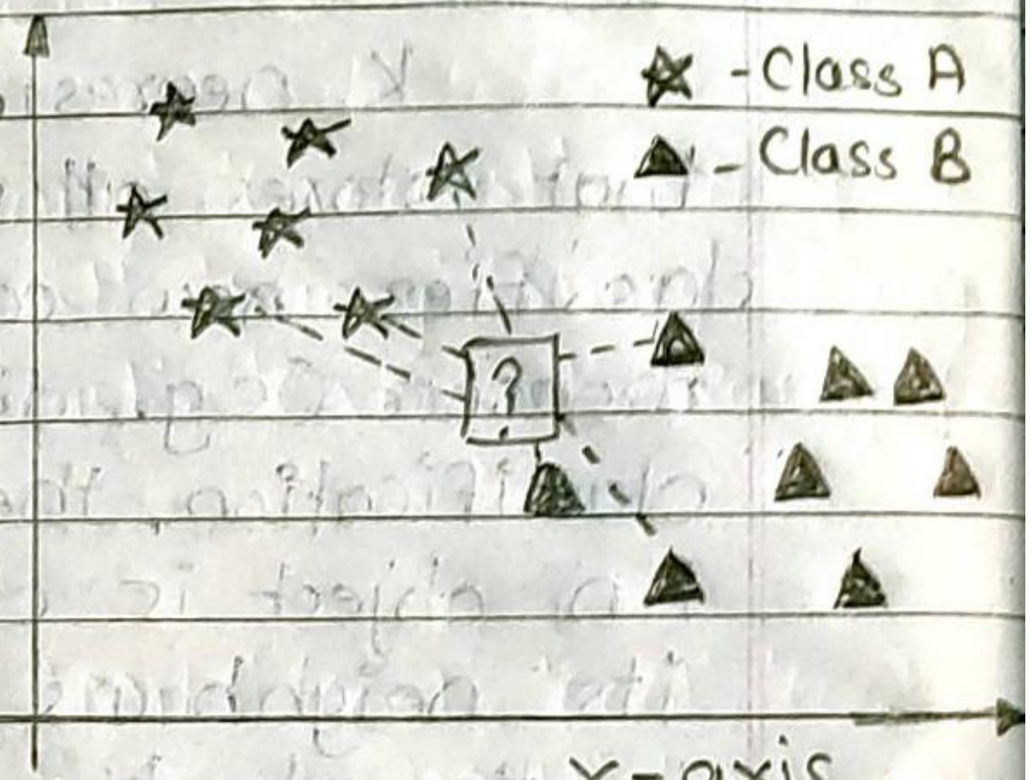
BEC0B215

PAGE NO.
DATE :

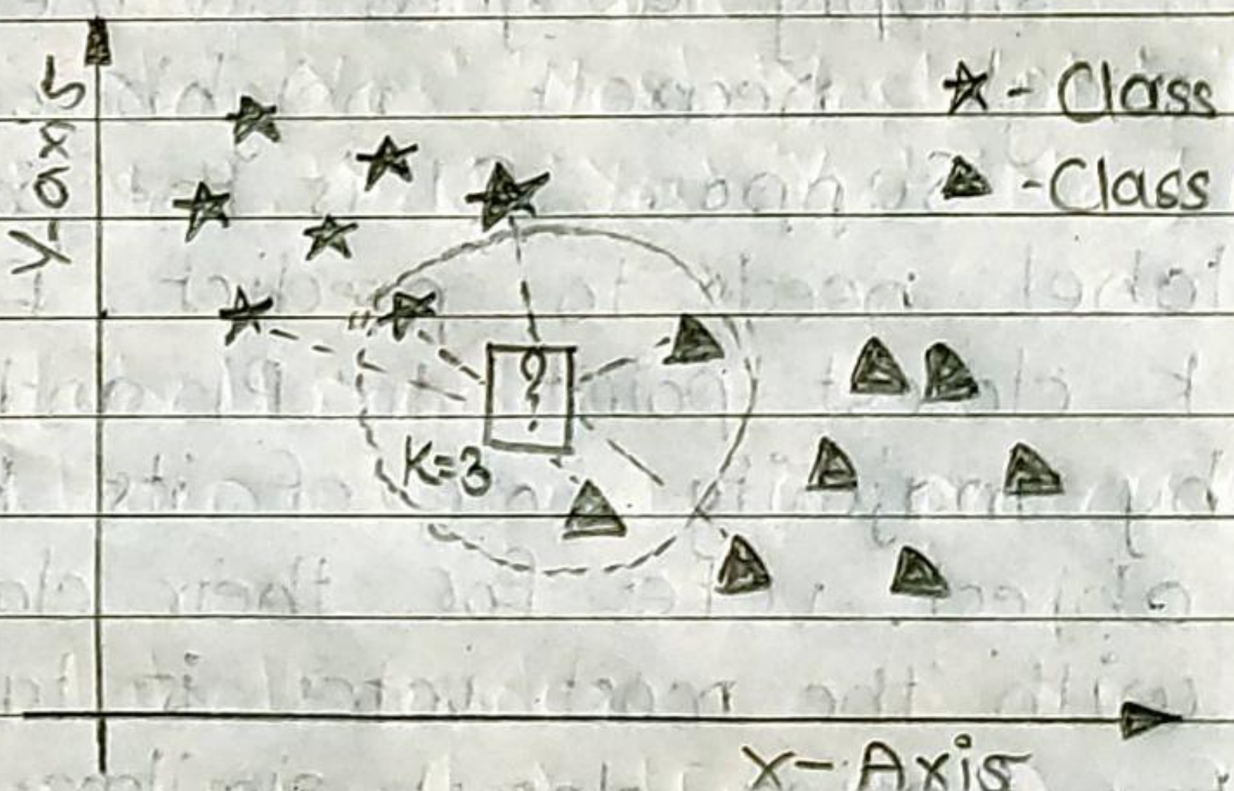
Initial Data



Calculate Distance



Finding Neighbors & Voting for Lables



Distance Functions :-

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

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

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

PAGE NO.
DATE :

Distance Weighted KNN:-

A refinement of the k-NN classification algorithm is to weigh the contribution of each of the k-neighbors according to their distance to the query point x_q, giving greater weight w_i to closer neighbors. This can be accomplished by replacing the final line in the algorithm by

$$F(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i \, \delta(v, f(x_i))$$

where the weight is

$$w_i = \frac{1}{d(x_q, x_i)^2}$$

(in case x_q exactly matches one of x_i, so that the denominator becomes zero, we assign $F(x_q)$ to $f(x_i)$ in this case.

For the version of k-NN for real-valued output the final line of the algorithm will be

$$F(x_q) \leftarrow \frac{\sum_{i=1}^{k} (w_i f(x_i))}{\sum_{i=1}^{k} w_i}$$

If weighting is used, it makes sense to use all traning examples, not just k-the algorithm then becomes a global one, since all training instances are used. The only disadvantage is that the algorithm will run more slowly.

PAGE NO.
DATE :

KNN Examples:-

```
# Assigning features and label variables
# First feature
weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy',
           'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy',
           'Sunny', 'Overcast', 'Overcast', 'Rainy']

# Second feature
temp = ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
        'Mild', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild',
        'Hot', 'Mild']

# Label or target variable
play = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
        'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

# Import LabelEncoder
from sklearn import preprocessing.

# creating labelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
weather_encoded = le.fit_transform(weather)

# converting string labels into numbers.
temp_encoded = le.fit_transform(temp)
label = le.fit_transform(play)
```

PAGE NO.
DATE :

```
# combining weather and temp into single
 list of tuples
features =list(zip(weather_encoded, temp_encoded))

from sklearn.neighbors import KNeighborsClassifier
 model =KNeighborsClassifier(n_neighbors= 3)

# Train the model using the training sets
  model.fit( features, label)

# Predict Output
 predicted, = model.predict([[0,2]])
       # 0: Overcast , 2: Mild
 print (predicted)
```

Output of example :-

[1]

In the above example, you have given input [0,2], where 0 means Overcast weather and 2 means Mild temperature. Model predicts [1], which means play.

* Conclusion :- Thus we have successfully implemented given problem using KNN technique.