ASSIGNMENT 3

Name: Vaishnavi Gosavi
Roll No: 71
Year & Class: TY CSAI-A
Batch: 2

Title: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Python ML library classes can be used for this problem.

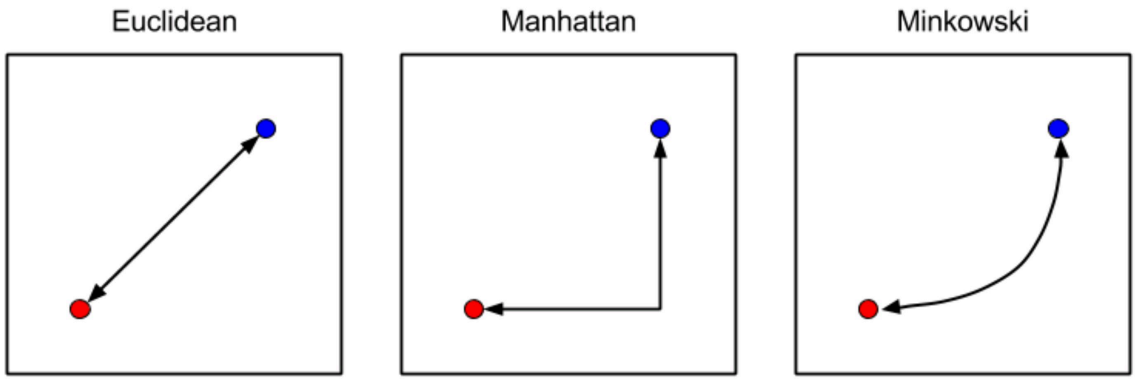
Description:-

1. Introduction to k-Nearest Neighbors (k-NN)

k-Nearest Neighbors (k-NN) is one of the simplest and most effective supervised machine learning algorithms used for both classification and regression tasks. The algorithm classifies a new data point based on the majority class of its nearest neighbors in the feature space. It relies on the assumption that similar data points exist close to each other.

2. Distance Metrics Used in k-NN

The performance of k-NN largely depends on the choice of distance metric, which measures the similarity between two data points. The commonly used distance metrics are:



- **Euclidean Distance:** the most widely used metric for calculating the straight-

$$d(x,\ y)\ = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

line distance between two points in space. It represents the shortest distance between two data points. The formula is:
In simple terms, it calculates the root of the sum of the squared differences between the corresponding features of two points.

- **Manhattan Distance:** also known as the L1 distance, calculates the distance between two points by summing the absolute differences of their corresponding coordinates. The formula is:

$$d(x, y)\ = \sum_{i=1}^{n} |x_i -\ y_i|$$

It measures the distance by following the grid-based path between points, like moving along the streets of a city.

- **Minkowski Distance** is a generalization of both Euclidean and Manhattan distances. It is defined as:

$$D(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Where:

  - p = 2 gives Euclidean Distance,
  - p = 1 gives Manhattan Distance,

Higher values of make the metric more sensitive to large differences in feature values. This metric allows flexibility by choosing different values of to adapt to different data distributions.

Algorithm

1. Choose the value of k (number of nearest neighbors).
2. Compute the distance between the query point and all training samples (commonly using Euclidean distance).
3. Sort the distances and select the k nearest neighbors.
4. Perform majority voting (for classification) or calculate the average (for regression).
5. Assign the predicted class label or value to the query point.

Implementation

1. Load the Iris dataset from sklearn.datasets.
2. Convert the dataset into a Pandas DataFrame.
3. Map species names for better readability.
4. Visualize relationships between features using pairplot from seaborn.
5. Split the dataset into Training (80%) and Testing (20%) sets.
6. Standardize the feature values using StandardScaler.
7. Build the k-NN model using KNeighborsClassifier with k = 5.
8. Make predictions on the test set.
9. Evaluate the model using:
10. Accuracy Score
11. Classification Report
12. Confusion Matrix
13. Find the optimal k value by plotting accuracy vs. k values.

Model Evaluation Metrics

To assess the performance of the k-NN model, the following evaluation metrics are used:

- **Accuracy Score:** Overall performance of the model.
- **Precision:** Proportion of true positives among predicted positives.
- **Recall:** Proportion of true positives among actual positives.
- **F1-Score:** Harmonic mean of precision and recall.
- **Confusion Matrix:** Summary of prediction results

→ KNN (K-Nearest Neighbour)

KNN classification for predicting if a person is underweight or overweight.

Dataset has two features: height (cm) and weight (kg)

it is classified to: underweight or overweight

Height (cm)	Weight (kg)	Category
150	50	underweight
160	60	underweight
170	65	overweight
180	80	overweight

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

$x_2 = 165$ $y_2 = 62$

distance from new point to (150, 50)

$= \sqrt{(165-150)^2 + (62-50)^2}$

$= \sqrt{(15)^2 + (12)^2}$

$= \sqrt{225 + 144}$

$= \sqrt{369}$

$= 19.20$

dist from new point to (160, 60)

Date: / /

$= \sqrt{(165-160)^2 + (62-60)^2}$

$= \sqrt{(5)^2 + (2)^2}$

$= \sqrt{25+4}$

$= \sqrt{29}$

$= 5.38$

dist from new point (170,6

$= \sqrt{(165-170)^2 + (62-65)^2}$

$= \sqrt{(-5)^2 + (-3)^2}$

$= \sqrt{25+9}$

$= \sqrt{34}$

$= 5.83$

dist from new point to (180, 80)

$= \sqrt{(165-180)^2 + (62-80)^2}$

$= \sqrt{(-15)^2 + (-18)^2}$

$= \sqrt{(225) + (324)}$

$= \sqrt{549}$

$= 23.4$

K points are

150	50	19.2	un
160	60	5.38	un
170	65	5.83	ov
180	80	23.4	ov

if we take k=3

nearst points are (160, 60), (170, 65) & (150,5

since majority is underweight

we will classify

new point (165, 62) as underweight.

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target
```

```
df = pd.DataFrame(data=X, columns=breast_cancer.feature_names)
df['target'] = y
```

```
df
```

ean ter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness
2.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622
2.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238
0.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444
7.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098
5.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374
...	...	...	...	...	...	...	...	...	...	...	...	..
2.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.1410
1.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.1166
3.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.1139
0.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.1650
7.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.0899

```
df.isnull().sum()
```

	0
mean radius	0
mean texture	0
mean perimeter	0
mean area	0
mean smoothness	0
mean compactness	0
mean concavity	0
mean concave points	0
mean symmetry	0
mean fractal dimension	0
radius error	0
texture error	0
perimeter error	0
area error	0
smoothness error	0
compactness error	0
concavity error	0
concave points error	0
symmetry error	0
fractal dimension error	0
worst radius	0
worst texture	0
worst perimeter	0
worst area	0
worst smoothness	0
worst compactness	0
worst concavity	0
worst concave points	0
worst symmetry	0
worst fractal dimension	0
target	0

dtype: int64

```
X
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
import numpy as np
prior_probs = {}
for class_label in np.unique(y_train):
    prior_probs[class_label] = np.sum(y_train == class_label) / len(y_train)
```

```
likelihoods = {}
for class_label in np.unique(y_train):
    X_class = X_train[y_train == class_label]
    likelihoods[class_label] = {
        'mean': np.mean(X_class, axis=0),
        'std': np.std(X_class, axis=0)
    }
```

```
def predict(X):
    predictions = []
    for sample in X:
        class_probs = {}
        for class_label in prior_probs:
            prior = prior_probs[class_label]
            likelihood = 1
            for i, feature_val in enumerate(sample):
                mean = likelihoods[class_label]['mean'][i]
                std = likelihoods[class_label]['std'][i]
                likelihood *= (1 / (np.sqrt(2 * np.pi) * std)) * np.exp(-((feature_val - mean) ** 2) / (2 * (std ** 2)))
            class_probs[class_label] = prior * likelihood
        predicted_class = max(class_probs, key=class_probs.get)
        predictions.append(predicted_class)
    return predictions
```

```
y_pred = predict(X_test)
```

```
accuracy = np.sum(y_pred == y_test) / len(y_test)
print(f"Accuracy: {accuracy}")
```

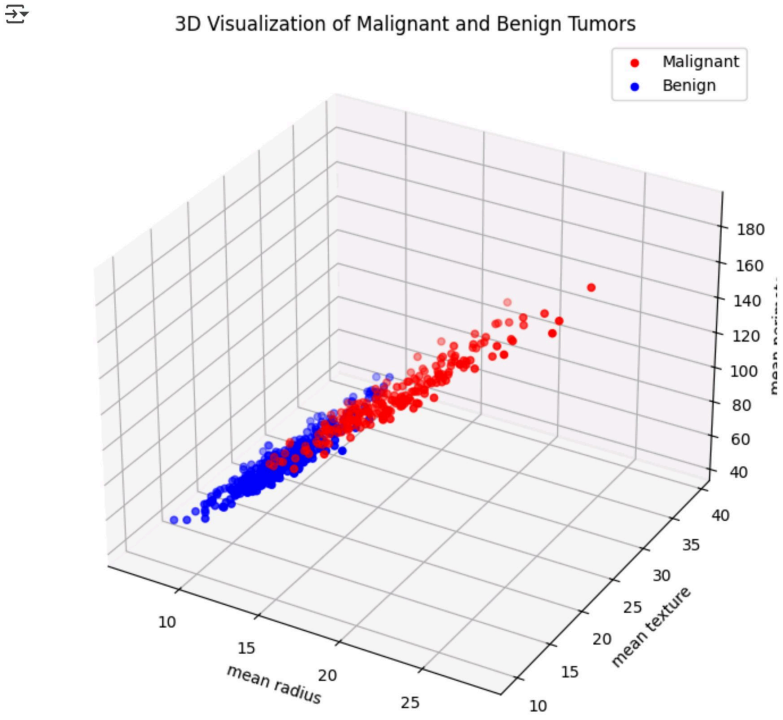
```
Accuracy: 0.9122807017543859
```

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

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88        63
           1       0.94      0.92      0.93       108

    accuracy                           0.91       171
   macro avg       0.90      0.91      0.91       171
```

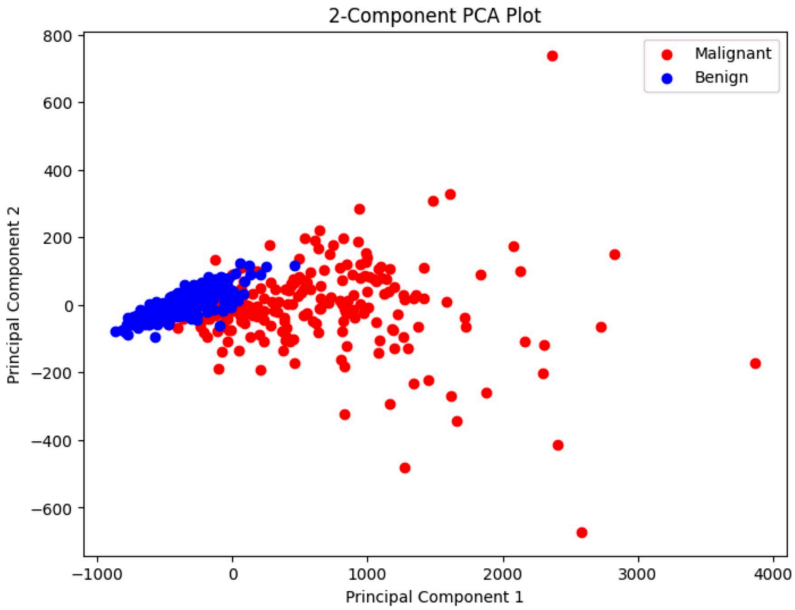
```
weighted avg       0.91      0.91      0.91       171
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import classification_report

# Load the breast cancer dataset
breast_cancer = load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target

# Create a DataFrame
df = pd.DataFrame(data=X, columns=breast_cancer.feature_names)
df['target'] = y

# Select three features for 3D visualization (you can change these)
features = ['mean radius', 'mean texture', 'mean perimeter']
X_subset = df[features]

# Create the 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot for malignant tumors (target = 0)
malignant = ax.scatter(X_subset[y == 0][features[0]], X_subset[y == 0][features[1]], X_subset[y == 0][features[2]], c='red', label='Malignant')

# Scatter plot for benign tumors (target = 1)
benign = ax.scatter(X_subset[y == 1][features[0]], X_subset[y == 1][features[1]], X_subset[y == 1][features[2]], c='blue', label='Benign')

# Set labels and title
ax.set_xlabel(features[0])
ax.set_ylabel(features[1])
ax.set_zlabel(features[2])
ax.set_title('3D Visualization of Malignant and Benign Tumors')

# Add legend
ax.legend(handles=[malignant, benign])

# Display the plot
plt.show()
```




```
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame with 'target' column
# and features for PCA (e.g., all except 'target')

# Separate features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

# Apply PCA with 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

```
X_pca = pca.fit_transform(X)

# Create a DataFrame with PCA components
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['target'] = y

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(pca_df[pca_df['target'] == 0]['PC1'], pca_df[pca_df['target'] == 0]['PC2'], label='Malignant', c='red')
plt.scatter(pca_df[pca_df['target'] == 1]['PC1'], pca_df[pca_df['target'] == 1]['PC2'], label='Benign', c='blue')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2-Component PCA Plot')
plt.legend()
plt.show()
```



Conclusion:

k-Nearest Neighbors is a simple yet powerful algorithm that performs well on small datasets with low dimensionality. However, its computational complexity and sensitivity to noise require careful preprocessing and feature selection. Selecting an appropriate k value and standardizing the features can significantly improve the model's performance.