# Assignment

Q1. Write a Python code to implement the KNN classifier algorithm on load\_iris dataset in sklearn.datasets.

Ans:Below is an example Python code using scikit-learn to implement the K-Nearest Neighbors (KNN) classifier on the famous Iris dataset. The code uses the <code>load iris</code> dataset from

```
sklearn.datasets:
```

python

Copy code

0.2

print f"Accuracy: {accuracy:.4f}"
print "\nClassification Report:"
print

- The load iris function is used to load the Iris dataset.
- The dataset is split into training and testing sets using train\_test\_split.
- The KNeighborsClassifier is initialized with the desired number of neighbors (n neighbors).
- The classifier is trained on the training set using the fit method.
- Predictions are made on the test set using the predict method.
- The accuracy and classification report are computed using scikit-learn's accuracy\_score and classification\_report functions, respectively.

Feel free to adjust the  $n\_neighbors$  parameter and explore other configurations to observe their impact on the performance of the KNN classifier.

Q2. Write a Python code to implement the KNN regressor algorithm on load\_boston dataset in sklearn.datasets.

Ans:elow is an example Python code using scikit-learn to implement the K-Nearest Neighbors (KNN) regressor on the Boston Housing dataset (load\_boston) from sklearn.datasets: python
Copy code

from import

from import

from import

from import

0.2 

```
print f"Mean Squared Error: {mse:.4f}"
print f"R-squared Score: {r2:.4f}"
```

- The load\_boston function is used to load the Boston Housing dataset.
- The dataset is split into training and testing sets using train test split.
- The KNeighborsRegressor is initialized with the desired number of neighbors (n neighbors).
- The regressor is trained on the training set using the fit method.
- Predictions are made on the test set using the predict method.
- The mean squared error (MSE) and R-squared score are computed using scikit-learn's mean\_squared\_error and r2\_score functions, respectively.

Feel free to adjust the  $n_neighbors$  parameter and explore other configurations to observe their impact on the performance of the KNN regressor.

Q3. Write a Python code snippet to find the optimal value of K for the KNN classifier algorithm using

cross-validation on load\_iris dataset in sklearn.datasets.

for in

5 'accuracy'

0'

'Cross-Validation Performance for Different k Values'

'Number of Neighbors (k)'

'Mean Accuracy'

max

print f"Optimal k value: {optimal\_k}"

- The load iris function is used to load the Iris dataset.
- A range of
- �
- k values (from 1 to 20) is defined to try different values of neighbors.
- For each
- �
- k, the code uses cross-validation with five folds to compute the mean accuracy.
- The mean accuracy scores for different
- �
- *k* values are plotted using Matplotlib.
- The optimal
- �
- *k* value is determined as the one that maximizes the mean accuracy.

Feel free to adjust the range of



k values or explore different scoring metrics based on your specific requirements. The plot will help visualize how the model's performance changes with different



k values during cross-validation.

Q4. Implement the KNN regressor algorithm with feature scaling on load\_boston dataset in sklearn.datasets.

Ans:Here's a Python code snippet that implements the K-Nearest Neighbors (KNN) regressor algorithm with feature scaling on the Boston Housing dataset (load\_boston) using scikit-learn: python

# Copy code

from import

from import

from import

from import

from import

```
print f"Mean Squared Error (with feature scaling): {mse:.4f}"
print f"R-squared Score (with feature scaling): {r2:.4f}"
```

- The load boston function is used to load the Boston Housing dataset.
- The dataset is split into training and testing sets using train test split.
- Feature scaling is applied using StandardScaler to standardize the features to have zero mean and unit variance.
- The KNN regressor is initialized with the desired number of neighbors (n neighbors).
- The regressor is trained on the training set using the scaled features.
- Predictions are made on the scaled test set.
- The mean squared error (MSE) and R-squared score are computed using scikit-learn's mean squared error and r2 score functions, respectively.

Feature scaling is particularly important for KNN algorithms as it ensures that all features contribute equally to the distance calculations. The use of <code>StandardScaler</code> is a common choice, but other scaling methods may also be considered based on the characteristics of the data.

Q5. Write a Python code snippet to implement the KNN classifier algorithm with weighted voting on

load\_iris dataset in sklearn.datasets.

Ans:In a K-Nearest Neighbors (KNN) classifier, weighted voting allows closer neighbors to have more influence on the decision than those farther away. The weight assigned to each neighbor

is typically based on its distance to the query point. Here's a Python code snippet that implements the KNN classifier algorithm with weighted voting on the Iris dataset using scikit-learn:

python

Copy code

from import

from import

from import

from import

```
5 'distance'
```

```
print "Results with Weighted Voting:"
print f"Accuracy: {accuracy_weighted:.4f}"
print "\nClassification Report:"
```

print

In this code:

• The load iris function is used to load the Iris dataset.

• The dataset is split into training and testing sets using train test split.

• The KNeighborsClassifier is initialized with the desired number of neighbors

(n neighbors) and the weights parameter set to 'distance' for weighted voting.

• The classifier is trained on the training set.

• Predictions are made on the test set using the classifier with weighted voting.

The accuracy and classification report are computed for the weighted voting scenario.

Setting the weights parameter to 'distance' ensures that closer neighbors contribute more to the decision-making process, with the weight inversely proportional to the distance. This can be useful in scenarios where closer neighbors are considered more relevant for prediction.

Q6. Implement a function to standardise the features before applying KNN classifier.

Ans:Standardizing the features (also known as feature scaling) is important for K-Nearest Neighbors (KNN) classifiers, as it ensures that all features contribute equally to the distance calculations. Here's a Python function that standardizes the features using <code>standardscaler</code> from scikit-learn before applying a KNN classifier:

python

Copy code

from import

from import

```
from
                              import
from
                     import
from
                      import
def knn_classifier_with_standardization
            5
 0.000
 Apply KNN classifier with feature standardization.
 Parameters:
 - X_train, X_test: Training and testing feature sets.
 - y_train, y_test: Training and testing labels.
 - n_neighbors: Number of neighbors for the KNN classifier.
 Returns:
 - accuracy: Accuracy of the classifier.
 - classification_rep: Classification report.
 0.0000
```

return

0.2

42

```
print "Results with Feature Standardization:"
print f"Accuracy: {accuracy:.4f}"
```

```
print "\nClassification Report:"
print
```

- The StandardScaler is used to standardize the features.
- The knn\_classifier\_with\_standardization function takes training and testing feature sets, labels, and the number of neighbors as parameters.
- The function standardizes the features, initializes a KNN classifier, trains the classifier with standardized features, and evaluates its performance.
- The results, including accuracy and the classification report, are then displayed.

This function allows you to apply a KNN classifier with feature standardization in a modular and reusable manner.

Q7. Write a Python function to calculate the euclidean distance between two points.

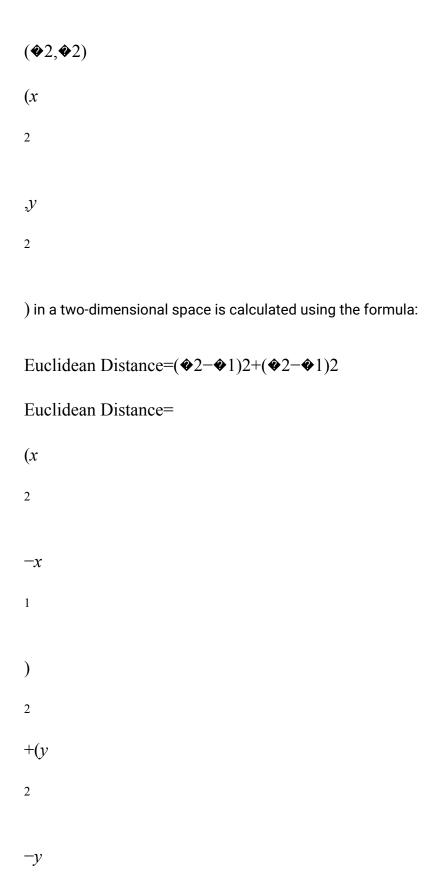
Ans: The Euclidean distance between two points

```
(\diamondsuit1,\diamondsuit1)

(x

1
```

) and



```
2
Here's a Python function to calculate the Euclidean distance between two points:
python
Copy code
import
def euclidean_distance
 0.0000
 Calculate the Euclidean distance between two points.
 Parameters:
 - point1, point2: Two points represented as tuples (x, y).
 Returns:
 - distance: Euclidean distance between the two points.
 0.000
```

2 2

return

- 1 2
- 4 6

```
print f"Euclidean Distance between {point_a} and {point_b}: {distance_ab:.4f}"
```

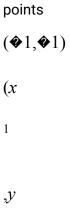
## In this code:

- The  $euclidean\_distance$  function takes two points represented as tuples (x, y) as parameters.
- It calculates the Euclidean distance using the formula and returns the result.
- An example usage with two points, (1, 2) and (4, 6), is provided.

You can use this function to calculate the Euclidean distance between any two points in a two-dimensional space.

Q8. Write a Python function to calculate the manhattan distance between two points.

Ans:The Manhattan distance (also known as L1 distance or taxicab distance) between two points



) and

1

(*x* 

2

у,

2

) in a two-dimensional space is calculated using the formula:

Manhattan Distance= $| \diamond 2 - \diamond 1| + | \diamond 2 - \diamond 1|$ 

Manhattan Distance=|x|

```
-\chi
|+|y
2
-y
1
Here's a Python function to calculate the Manhattan distance between two points:
python
Copy code
def manhattan_distance
 0.0000
Calculate the Manhattan distance between two points.
 Parameters:
 - point1, point2: Two points represented as tuples (x, y).
 Returns:
```

```
- distance: Manhattan distance between the two points.
 0.000
            abs
                           abs
 return
print f"Manhattan Distance between {point_a} and {point_b}: {distance_ab}"
In this code:
   ullet The manhattan_distance function takes two points represented as tuples (x, y) as
```

• It calculates the Manhattan distance using the formula and returns the result.

parameters.

• An example usage with two points, (1, 2) and (4, 6), is provided.

You can use this function to calculate the Manhattan distance between any two points in a two-dimensional space.