CURNEU MEDTECH INNOVATIONS PRIVATE LIMITED

SD03Q03

- **BAALA VIDHYA SREE K C (1832011)**

PROBLEM STATEMENT – 1

FRUIT CLASSIFICATION USING KNN ALGORITHM

ABSTRACT:

This document deals with the creation of a classification model for the given fruit dataset. The model here is implemented by using KNN classification algorithm. For the given dataset, we plot different scatter plots for different parameter combinations, do feature engineering, split data into train and test sets, define a KNN classifier and implement it for three test cases. The tool used here is Python.

INTRODUCTION:

Fruits in real – life can be identified based on their color, shape, size, etc..... as we humans know how each fruit looks like. For a machine to identify a fruit, we must provide basic information about the fruits like the prior mentioned attributes for which we can develop a model and implement and see as to how accurate the machine can able to identify the fruit.

ABOUT THE DATASET:

The given 'fruits.csv' contains the following attributes:

1. fruit_label
2. fruit_name
3. mass
4. width
5. height
6. color_score

The fruits dataset has information about the following fruits:

∅ apple ∅ orange ∅ lemon ∅ mandarin

About the attributes:

⇒ The fruit_label attribute consists of label numbers '1, 2, 3, 4' for the fruits 'apple, mandarin, orange, lemon' respectively.
⇒ The fruit_name attribute contains the names of the four fruits mentioned earlier.
⇒ The mass attribute contains the mass of each individual fruit observed and recorded.
⇒ The width attribute contains the width of each individual fruit observed and recorded.

⇒ The height attribute contains the height of each individual fruit observed and recorded.
⇒ The color_score attribute contains values from color chart that depict the respective colors.

A	B	C	D	E	F
fruit_label	fruit_name	mass	width	height	color_score
1	apple	192	8.4	7.3	0.55
1	apple	180	8	6.8	0.59
1	apple	176	7.4	7.2	0.6
2	mandarin	86	6.2	4.7	0.8
2	mandarin	84	6	4.6	0.79
2	mandarin	80	5.8	4.3	0.77
2	mandarin	80	5.9	4.3	0.81
2	mandarin	76	5.8	4	0.81
1	apple	178	7.1	7.8	0.92

CODE EXPLANATION:

Importing Required Libraries:

⇒ First, we import the required libraries for the code to run effectively.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.model_selection import train_test_split as TTS
from sklearn.preprocessing import MinMaxScaler as MMS
```

Importing Fruit Dataset:

⇒ Given Fruit dataset is imported
⇒ A dictionary is created for the fruit_name using fruit_label as index so that it will be helpful to display the desired output.

```
data = pd.read_csv('/content/fruits.csv')
pred = dict(zip(data.fruit_label.unique(),
            data.fruit_name.unique()))
pred
```

Knowing the dataset

⇒ To know about the dataset's format, we can use the following code lines:

```
data.info()
data.describe()
```

⇒ data.info() - From this, we can know about what are all the attributes present their data types, null – values (if present any).
⇒ data.describe() – From this, we can come to know about the summary statistics of the dataset.

Checking for null values

⇒ First, we check for any null values present or not. If present, then we replace them with any central location (mean, median, mode) values.

```
data.isnull().sum()
```

Plots comparing different attributes

⇒ To visually see the how each parameter affects the other parameters, we can use scatter plots, line graph and histogram.

#For all attributes
```
data.hist(figsize=(10,10))
plt.show()
```

#B/W Width and Height
```
plt.figure(figsize=(8,5))
plt.scatter(data['width'], data['height'])
plt.title('Width & Height')
plt.xlabel('Width')
plt.ylabel('Height')
plt.show()
```

#B/W Mass and Color Score
```
plt.figure(figsize=(8,5))
plt.scatter(data['mass'], data['color_score'])
plt.title('Mass & Color_Score')
plt.xlabel('Mass')
plt.ylabel('Color_Score')
plt.show()
```

#B/W Width and Height – Line Graph
```
plt.figure(figsize=(9,5))
plt.plot(data['height'], label='Height')
plt.plot(data['width'], label='Width')
plt.title('Hight & Weight Relation')
plt.legend()
plt.show()
```

Assigning Attributes to Variables

⇒ We categorize the attributes to Variables as: Dependent (X) & Independent (Y).

```
X = data.iloc[:, 2:].values
y = data['fruit_label'].values
```

✓ Here, X variable contains the attribute: mass, width, height, color_score
✓ Y variable contains the attribute 'fruit_label' (known as the 'target variable')

Feature Scaling

⇒ Feature Scaling is done to normalize the data's range or feature.

```
scaler = MMS()
X = scaler.fit_transform(X)
```

Splitting Dataset into Training and Test Sets

⇒ The Variable Datasets are split into Training and Test Sets.
⇒ Training set is used to build the model while test set is used to test the model's accuracy.

```
X_train, X_test, y_train, y_test = TTS(X, y, test_size = 0.3
,random_state = 1234)
```

KNN Classifier Function

About KNN:

It's a simple classification algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

```
def euclidean_distance(a, b):
        return np.sqrt(np.sum((a - b)**2))

class KNN:
    def __init__(self, k=5):
        self.k = k
    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
    def predict(self, X):
        y_predict = [self._predict(x) for x in X]
        return np.array(y_predict)
    def _predict(self, x):
      import numpy as np
```

```
        distances = [euclidean_distance(x, x_train) for
                                        x_train in self.X_train]
        k_index = np.argsort(distances)[:self.k]
        k_neighbor_labels = [self.y_train[i] for i in k_index]
        most_common = Counter(k_neighbor_labels).most_common(1)
        return most_common[0][0]
```

To determine the Best k – value

Elbow Method:

This method can be used to determine the optimal number of clusters that can be formed in a dataset.

```
from sklearn.neighbors import KNeighborsClassifier
k = range(1,15)
total_acc = []

for i in k:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    acc = knn.score(X_test,y_test)
    total_acc.append(acc)

plt.figure(figsize=(9,5))
plt.xlabel("value of K")
plt.ylabel("Accuracy")
plt.title("Select best value of k")
plt.plot(k,total_acc)
plt.show()
```

⇒ We can also determine the optimal k value by determining the error rate. The k value of least error rate will be taken as optimal k – value.

```
error_rate = []
for i in range(1,25):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(9, 5))
plt.plot(range(1,25),error_rate,color='blue', linestyle =
'dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:",min(error_rate),"at K =",
                      error_rate.index(min(error_rate)))
```

Prediction using the Classifier

Using the above KNN - Classifier Function, we can fit the training sets to the model and predict results for the new data.

```
k = 1
clf = KNN(k=k)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
```

Classification Report

A Classification report is used to measure the quality of predictions from a classification algorithm.

The classification report displays the following:

- ⇒ Precision (percent of predictions that were correct)
- ⇒ Recall (percent of positive instances found)
- ⇒ F1 (percent of positive predictions that were correct)
- ⇒ Support scores (number of actual occurrences of the class in the specified dataset)

```
def Accuracy(y_true, y_predict):
    accuracy = np.sum(y_true == y_predict) / len(y_true)
    return accuracy

def precision_score(y_test, y_predict):
    true_positives = len([a for a, p in zip(y_test, y_predict) if a ==
                                                    p and p == 1])
    false_positives = len([a for a, p in zip(y_test, y_predict) if a !=
                                                    p and p == 1])
    return true_positives / (true_positives + false_positives)

def f1_score(y_test,y_predict):
    return 2 * (precision_score(y_test, predictions) * recall_score(
      y_test, predictions)) / (precision_score(y_test, predictions) +
      recall_score(y_test, predictions))

def recall_score(actual, predicted):
    true_positives = len([a for a, p in zip(actual, predicted) if a ==
                                                    p and p == 1])
    false_negatives = len([a for a, p in zip(actual, predicted) if a !=
                                                    p and p == 0])
    return true_positives / (true_positives + false_negatives)


print("\nCLASSIFICATION REPORT")
print("\nAccuracy :", round(Accuracy(y_test, predictions), 2))
print("\nPrecision:", round(precision_score(y_test, predictions), 2))
```

```
print("\nF1 Score :", round(f1_score(y_test, predictions), 2))
print("\nRecall   :", round(recall_score(y_test, predictions), 2))
```

Confusion Matrix

The Confusion Matrix is used to describe the classification model performance on a test dataset for which the true values are known.

```
classes = set(y_test)
number_of_classes = len(classes)

conf_matrix = pd.DataFrame(
    np.zeros((number_of_classes, number_of_classes),dtype=int),
    index=classes,
    columns=classes)

for i, j in zip(y_test, predictions):
        conf_matrix.loc[i, j] += 1

print("CONFUSION MATRIX:\n\n", conf_matrix.values)
```

Test Cases

The test cases can be used to verify the performance of the classifier on new data.

```
test_case_1 = clf.predict([[1.0,1.0,0.8,0.5]])
print('The Fruit is: ', pred[test_case_1[0]])


test_case_2 = clf.predict([[0.0, 0.0, 0.0, 0.6]])
print('The Fruit is: ', pred[test_case_2[0]])


test_case_3 = clf.predict([[0.4, 0.3, 0.9, 0.4]])
print('The Fruit is: ', pred[test_case_3[0]])
```

OUTPUT AND INFERENCE:

⇒ The dictionary created for each fruit_name using fruit_label respectively.

```
{1: 'apple', 2: 'mandarin', 3: 'orange', 4: 'lemon'}
```

⇒ Results of data.info() tells us that the data has 6 attributes, all attributes are free of null values, and of data types int (fruit_label, mass), float (width, height, color_space) and object (fruit_name).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59 entries, 0 to 58
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   fruit_label  59 non-null     int64  
 1   fruit_name   59 non-null     object 
 2   mass         59 non-null     int64  
 3   width        59 non-null     float64
 4   height       59 non-null     float64
 5   color_score  59 non-null     float64
dtypes: float64(3), int64(2), object(1)
memory usage: 2.9+ KB
```

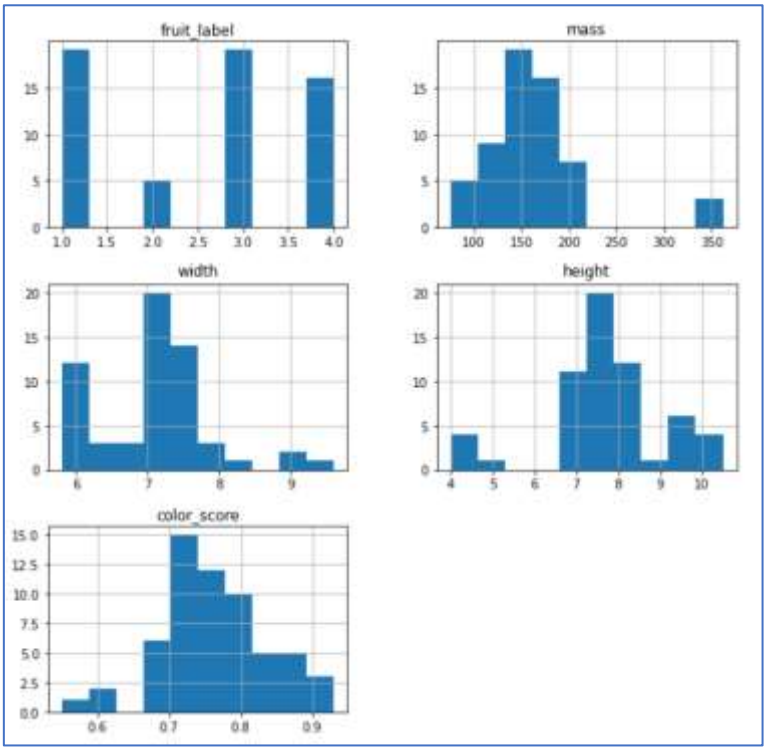
⇒ Results of data.describe() give details about the summary statistics like mean, median, standard deviation, quantile values, maximum and minimum values and count of each attribute.

	fruit_label	mass	width	height	color_score
count	59.000000	59.000000	59.000000	59.000000	59.000000
mean	2.542373	163.118644	7.105085	7.693220	0.762881
std	1.208048	55.018832	0.816938	1.361017	0.076857
min	1.000000	76.000000	5.800000	4.000000	0.550000
25%	1.000000	140.000000	6.600000	7.200000	0.720000
50%	3.000000	158.000000	7.200000	7.600000	0.750000
75%	4.000000	177.000000	7.500000	8.200000	0.810000
max	4.000000	362.000000	9.600000	10.500000	0.930000

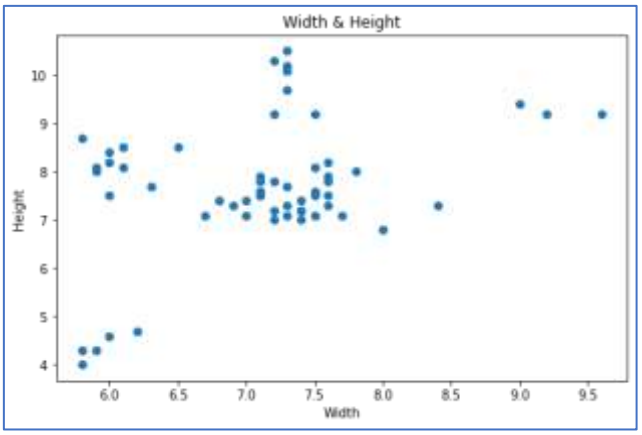
⇒ Results of data.isnull().sum() shows that all attributes are free of null values.

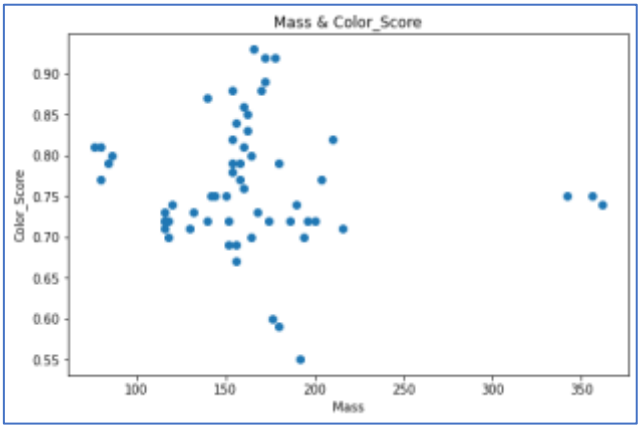
```
fruit_label    0
fruit_name     0
mass           0
width          0
height         0
color_score    0
dtype: int64
```

⇒ The Histogram plots shows how each attribute is distributed, range of attributes.

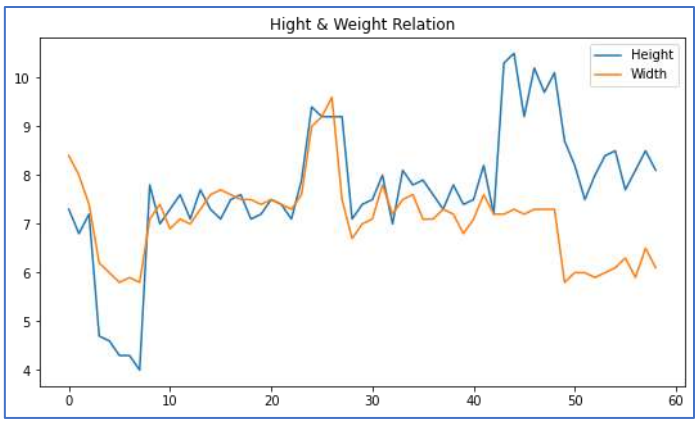


⇒ The scatter plots show how width and height and mass and color_score pair of attributes are related.

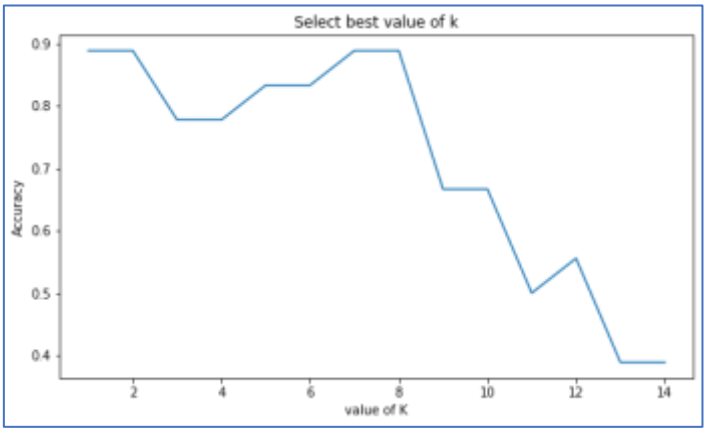


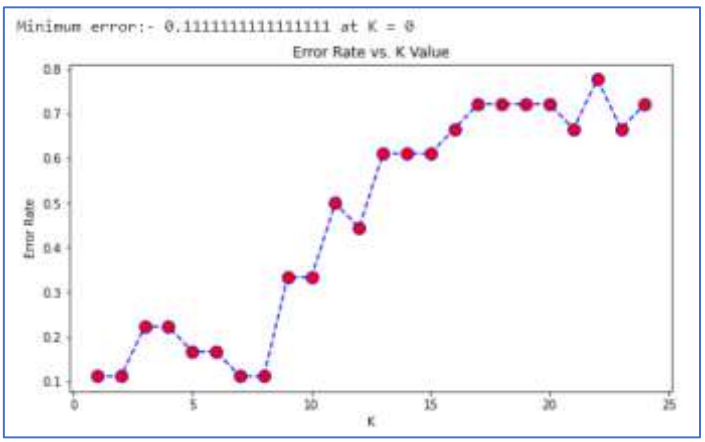


⇒ The line graph tells how both width and height are distributed across the data for individual fruits.



⇒ From the below 2 graphs, we can say that the optimal value for k is 1.





⇒ Classification Report Results:

- ∅ Accuracy for the model is **89% accurate**, which is fairly a good model and can predict well.
- ∅ Precision Score for the model is 0.86, which means 86% of predictions that were correct

- ∅ Recall value for the model 1.0, which means all the instance identified were positive.
- ∅ F1 score is 0.92, which is used to compare with other classifier models.

```
CLASSIFICATION REPORT

Accuracy : 0.89

Precision: 0.86

F1 Score : 0.92

Recall   : 1.0
```

⇒ Confusion matrix results:

From the matrix, it is clear that most of the predictions where true positives, and only one is false negative and only one false positive values.

```
CONFUSION MATRIX:

 [[6 0 0 0]
 [0 2 0 0]
 [1 0 3 1]
 [0 0 0 5]]
```

⇒ Prediction of New Results (Test Cases):

The following are the results for the test cases:

```
test_case_1 = clf.predict([[1.0,1.0,0.8,0.5]])
print('The Fruit is: ', pred[test_case_1[0]])

The Fruit is:  orange

[20] test_case_2 = clf.predict([[0.0, 0.0, 0.0, 0.6]])
     print('The Fruit is: ', pred[test_case_2[0]])

     The Fruit is:  mandarin

[21] test_case_3 = clf.predict([[0.4, 0.3, 0.9, 0.4]])
     print('The Fruit is: ', pred[test_case_3[0]])

     The Fruit is:  lemon
```

CONCLUSION

From the results, we can conclude that the scratch code has indeed performed well in predicting the results for test data and also for any new data given. Thus, fruit dataset can be well predicted with this model and also can identify any new fruit for the given attribute values.