Enhancing Dry Bean Classification: Leveraging K-Nearest Neighbors Algorithm with Parameter Optimization

Adhip Bhattarai[1*], Bishal Rijal[2*]
[1]Department of Electronics and Computer Engineering, Thapathali Campus, IOE, Tribhuvan University (e-mail: adhipbh200@gmail.com)
[2]Department of Electronics and Computer Engineering, Thapathali Campus, IOE, Tribhuvan University (e-mail: bishalrijal5467@gmail.com)

Corresponding author: First A. Author (e-mail: author@ boulder.nist.gov).

*Authors contributed equally

ABSTRACT In this report, we delve into the application of the K-Nearest Neighbors (KNN) algorithm for classifying various types of dry beans using a dataset sourced from Kaggle. Our primary objective revolves around training the KNN algorithm to accurately distinguish between different bean varieties based on their unique features. Initiating our investigation, we establish a baseline performance by implementing the KNN algorithm with its default hyperparameters. Through data preprocessing and model training, we attain an initial accuracy rate of 72%, serving as a foundational benchmark. Progressing further, we focus on the pivotal aspect of hyperparameter tuning, an integral facet of optimizing machine learning algorithms. Our pivotal breakthrough emerges through the utilization of the KNeighborsClassifier, with specific hyperparameters fine-tuned to enhance classification performance. By setting the number of neighbors (n_neighbors) to 10 and employing the 'distance' weighting scheme, we achieve an elevated accuracy rate of 74%. This outcome underscores the substantial impact of parameter optimization on model efficacy. Our study illuminates the intricate interplay between algorithmic hyperparameters and classification accuracy. This exploration not only elucidates the operational dynamics of the KNN algorithm in the context of dry bean classification but also underscores the critical role of parameter adjustments in machine learning model refinement.

INDEX TERMS Accuracy, Dry Beans, hyperparameter, K-Nearest Neighbors

I. INTRODUCTION

The K-Nearest Neighbors (KNN) algorithm serves as a fundamental building block within the realm of Machine Learning, employing the principles of Supervised Learning. With simplicity at its core, KNN operates on the premise of assessing similarities between new and existing data points, subsequently categorizing the new data based on its likeness to established categories. One of the distinct features of KNN is its ability to retain all available data points for reference during classification, making it an effective tool for sorting new data into appropriate categories.

Functioning effectively for both Classification and Regression tasks, KNN is particularly renowned for its application in solving Classification problems. Notably, KNN stands as a non-parametric algorithm, avoiding any preconceived assumptions about the underlying data distribution. This characteristic grants KNN versatility, enabling it to adapt to various types of data patterns.

KNN exhibits a unique learning style often referred to as "lazy learning." Unlike eager learners who immediately process training data, KNN conserves the training dataset until the classification phase. Upon encountering new data, KNN leverages its stored dataset to categorize the incoming information by comparing it with its accumulated knowledge. By facilitating this direct matching of data points, KNN efficiently classifies new instances into categories that closely resemble the new data's attributes, exemplifying its principle of proximity-based classification.

II. METHODOLOGY

A. KNN ALGORITHM

The K-Nearest Neighbors (KNN) algorithm is a straightforward and intuitive machine-learning technique that falls under the category of supervised learning. It's used for both classification and regression tasks, although it's most commonly associated with classification. The central idea behind KNN is to classify a new data point by looking at the "k" closest data points in the training dataset.

Steps:

- Choosing the Value of K:

The "k" in KNN represents the number of nearest neighbors that the algorithm considers when making a prediction for a new data point. This value is an important parameter that you need to determine based on your dataset and problem. A small "k" might make the model sensitive to noise, while a large "k" might oversimplify the classification boundaries.

- Calculating Distance:

The algorithm calculates the distance between the new data point and all the data points in the training set to determine the nearest neighbors. The distance can be calculated using various metrics, such as Euclidean distance, Manhattan distance, Minkowski distance, etc.. Each data point is represented as a vector in a high-dimensional space, and the distance between them is a measure of their similarity.

$$dist(x,z) = \sum_{r=1}^{d} |x_r - z_r|^p \qquad 1$$

- Finding the Nearest Neighbors:

The algorithm then identifies the "k" training data points that are closest to the new data point based on the calculated distance. These data points are neighbors with the most similar feature values to the new data point.

- Voting (Classification) or Averaging (Regression):

For classification tasks, KNN takes a majority vote from the "k" neighbors to determine the class of the new data point. The class that occurs most frequently among the neighbors is assigned to the new point. In regression tasks, the algorithm averages the target values of the "k" nearest neighbors to predict the target value for the new data point.

- Making Predictions:

Once the majority vote or average is determined, the algorithm assigns the class label (for classification) or the predicted value (for regression) to the new data point.

B. MODEL ARCHITECTURE

The System Block Diagram is represented in the Appendix and Explained in the Working Principle section.

C. INSTRUMENTATION

- scikit-learn (sklearn):

Scikit-learn is a powerful machine-learning library that provides a user-friendly interface for various algorithms, including KNN. It offers efficient tools for data preprocessing, model training, evaluation, etc.

- numpy:

Numpy is a fundamental library for numerical computations and working with arrays. It's often used to manage data in numerical form, which is essential for distance calculations and array manipulations in KNN.

- pandas:

Pandas is another essential library for data manipulation and analysis. It helps with loading, cleaning, and preprocessing datasets, which is crucial before applying KNN.

- Matplotlib:

Matplotlib is a popular library for creating data visualizations. While not strictly necessary for KNN, it can be useful for visualizing results, data distributions, and plotting correlations as heatmap, etc.

III. WORKING PRINCIPLE

A. DATASET PREPARATION

Consider a dataset

$$X = (x_{ij})_{m \times n} \qquad 2$$

For In this notation, "m" represents the total number of instances or data points, and "n" signifies the number of features present in each instance. Specifically, x_ij represents the value of the "j"-th feature for the "i"-th data point. In the context of the dry beans dataset, "i" corresponds to the 13611 rows, which are individual instances of beans. The "j" index denotes the 17 different numerical attributes associated with each bean.

Each attribute in the dry beans dataset provides quantitative information about different aspects of the beans. These attributes cover characteristics such as size, shape, and texture. Unlike pixel intensities in image data, these attributes are numerical values that capture various traits relevant to the beans' classification.

The dry beans dataset is structured with 13611 instances, and each instance is described by 17 attributes. These attributes, excluding the target column, contribute to the multidimensional representation of each bean. The dataset's primary objective is to employ these attributes to accurately classify the beans into seven distinct classes: SEKER, BARBUNYA, BOMBAY, CALI, HOROZ, SIRA, and DERMASON. The attributes' values aid in determining the specific class to which each bean belongs, facilitating the classification process.

B. DATASET PREPROCESSING

To ensure the accuracy and significance of our analysis on the dry beans dataset [1], it is imperative to effectively preprocess the data. This involves a sequence of essential steps that ready the dataset for utilization by the K-Nearest Neighbors (KNN) algorithm:

- Label Encoding:

  Unlike datasets with traditional categorical variables, the dry beans dataset's attributes are primarily numerical. Consequently, the label encoding process, typically employed for categorical data, is not applicable. However, as the classification task involves distinct bean classes (SEKER, BARBUNYA, BOMBAY, CALI, HOROZ, SIRA, DERMASON), no additional encoding is required for the output variable.

- Normalization:

  Normalization stands as a pivotal operation to ensure uniform scaling of feature values. In the context of the dry beans dataset, where attributes encompass various numerical aspects, normalization enhances the data's convergence during model training. Since all attributes are already numeric and span different ranges, we executed normalization to ensure their values fall within a consistent range. The scaling was performed on the entire dataset, adjusting all values proportionally.

- Handling Missing Values:

  The dry beans dataset, being meticulously curated, thankfully does not contain any missing values. Each instance boasts complete and well-defined attribute values. Consequently, there is no requirement for imputation or special handling of missing data.

IV. WORKING PRINCIPLE

The working block diagram of our lab is shown in Figure 1. We used 2 methods for training: default and customized.

Default Mode Training:

Initially, the KNN algorithm was trained in its default mode [1], where several parameters were set to their standard values. The following settings were used during this training phase:

- weights: 'uniform'

  In the default mode, the weight function used for prediction was set to 'uniform'. This implies that all data points within the neighborhood were assigned equal weights, thus treating them with uniform importance.

- algorithm: 'auto'

  The algorithm used to compute nearest neighbors was set to 'auto'. This mode allowed the algorithm to automatically select the most suitable neighbor search method based on the specific data and its characteristics.

- leaf_size: 30

  A leaf size of 30 was employed for BallTree or KDTree. This parameter affects the construction and query speed of the tree as well as the memory requirements. The value of 30 was chosen to balance these considerations.

- p: 2

  The power parameter for the Minkowski metric was set to 2, which corresponds to using the Euclidean distance (l2) for distance calculations.

- metric: 'minkowski'

  The metric used for distance computation was 'minkowski', resulting in the standard Euclidean distance when p = 2. The Euclidean distance measures the straight-line distance between data points in a multi-dimensional space.

- n_jobs: None

  The number of parallel jobs used for neighbor search was set to None, which implies that the algorithm runs using a single job. The value of n_jobs can be adjusted to utilize multiple processors if available.

Customized Training: n_neighbors = 10, weights = 'distance':

In the subsequent phase of training, the KNN algorithm was fine-tuned by modifying specific parameters to enhance its predictive performance. The following settings were employed:

- n_neighbors: 10

  To influence the classification outcome, the number of neighbors considered for prediction was set to 10. This means that the algorithm evaluated the 10 nearest neighbors to make a prediction for a given data point.

- weights: 'distance'

The weight function was altered to 'distance'. In this configuration, the influence of each neighbor on the prediction was determined by the inverse of their distance. Closer neighbors held more influence, with their weight inversely proportional to their distance from the data point being predicted.

V. RESULTS:

In the context of the dry beans dataset, our focus was on harnessing the power of the K-Nearest Neighbors (KNN) algorithm. This dataset encompasses attributes that define different types of beans, prompting us to construct a classification model tailored to this unique domain. The overarching goal was to establish a model proficient in accurately categorizing the various bean types based on their distinctive attributes.

In the dry beans dataset, we have information about seven different types of beans. To make this data ready for analysis, we started by normalizing it. This means we adjusted the values so they all fit within the same range. This helps the data work better when we train our model. Since KNN uses distance metrics, normalization is a crucial step.

The Pearson Heat Mapping of Features illustrates the relation of Features with one another which can be seen in Figure 5. It shows that the Perimeter is more correlated with MajorAxisLength and the Perimeter is correlated with Area. It also shows other correlations between features like Solidity vs Eccentricity as well as others.

The sci-kit learn provides KNN in its library. It is very easy to implement the KNN using sci-kit learn. We obtain the classification report as shown in Figure 5. We were able to obtain an accuracy of 72% using default parameters and 74% using hyperparameter tuning.

The confusion matrices for both the approaches are shown in Figure 2 and Figure 6.

VI. DISCUSSION AND ANALYSIS

During our exploration of creating a K-Nearest Neighbors (KNN) model for the dry beans dataset, we embarked on a thorough investigation to evaluate the model's performance and extract valuable insights from our experimental setup. Our primary aim was to optimize the model's effectiveness by systematically adjusting different parameters and gaining a deeper understanding of its behavior.

Throughout our experimentation, we rigorously assessed the KNN model using the dry beans dataset, which features diverse attributes defining various types of beans. By systematically fine-tuning and training the model, we garnered intriguing results that unveiled the model's efficacy within this specific context.

In the initial phase of our analysis, we established a baseline performance with default parameters. The model's accuracy was recorded at 72%, highlighting its capability to make accurate classifications based on the input attributes of the dry beans.

Subsequently, we endeavored to enhance the model's accuracy by customizing parameters. By selecting 10 neighbors and assigning weights based on distance, we observed an increase in accuracy to 74%. This parameter adjustment demonstrated the model's adaptability and ability to make more precise classifications, showcasing its effectiveness in categorizing the various types of beans in the dataset.

While KNN is a versatile algorithm, exploring other machine learning algorithms like Random Forest, Support Vector Machines, or Neural Networks might offer better predictive capabilities for the dry beans dataset.

VII. REFERENCES:

[1] *Dry Bean Dataset,* 2020.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, V, J. erplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "scikit-learn," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

Adhip Bhattarai is a dedicated individual pursuing a Bachelor's degree in Computer Engineering at Tribhuvan University. With a strong passion for machine learning and data science, he is constantly exploring the latest advancements in these fields. Although he may not have notable accomplishments just yet, Adhip's enthusiasm and drive for learning and applying cutting-edge technologies make him a promising and ambitious individual in the world of computer engineering.

Bishal Rijal is a dedicated individual currently studying Bachelor's in Computer and Technology at Tribhuvan University. Bishal's enthusiasm for research and innovation has led him to undertake various projects and engage in practical applications of his knowledge. He continually seeks to deepen in understanding of the subject matter, staying up-to-date with the latest advancements and trends. With his relentless determination, inquisitive mindset, and expertise in machine learning and data science, Bishal Rijal is poised to make significant contributions to the ever-evolving field of technology.

VIII. APPENDICES

A. FIGURES

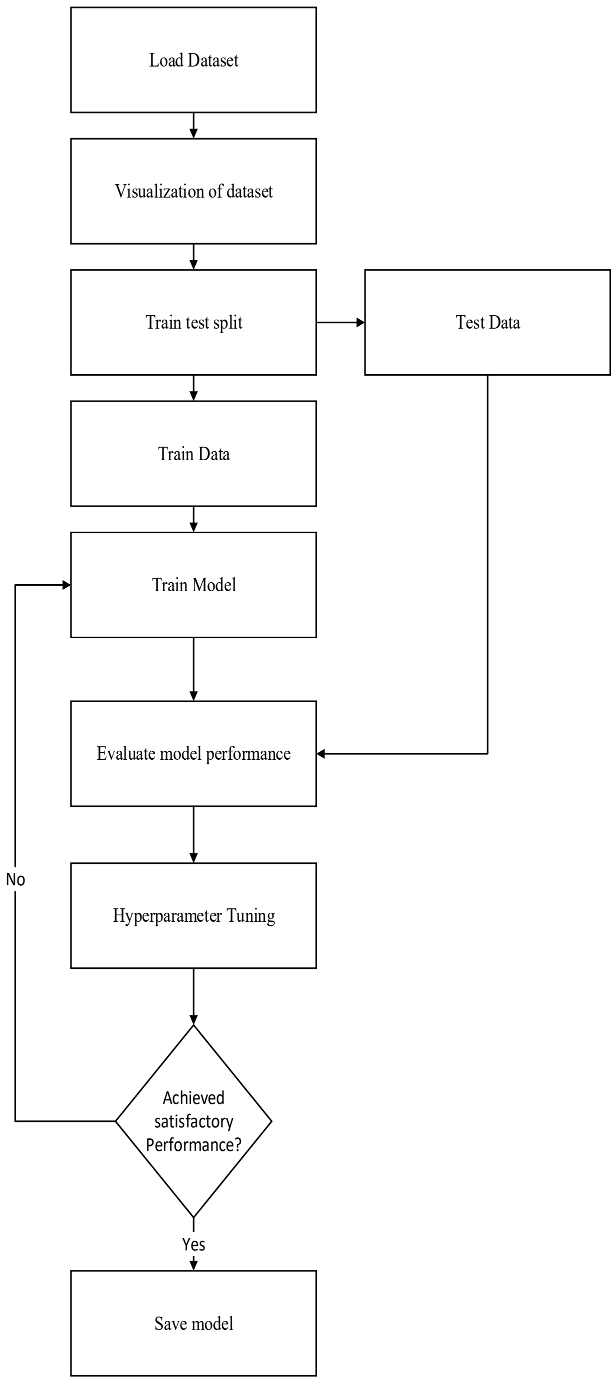
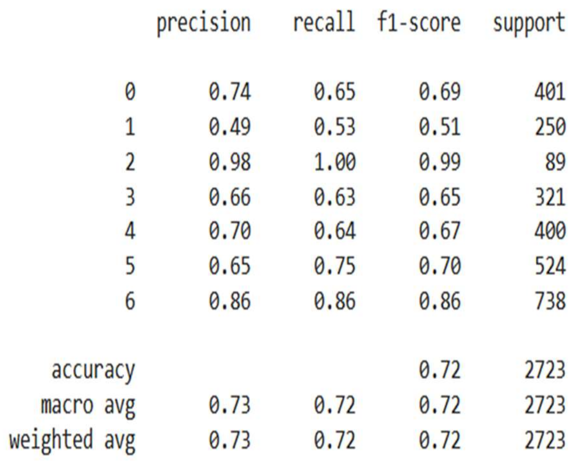


Figure 1. Block Diagram



```
              precision    recall  f1-score   support

           0       0.74      0.65      0.69       401
           1       0.49      0.53      0.51       250
           2       0.98      1.00      0.99        89
           3       0.66      0.63      0.65       321
           4       0.70      0.64      0.67       400
           5       0.65      0.75      0.70       524
           6       0.86      0.86      0.86       738

    accuracy                           0.72      2723
   macro avg       0.73      0.72      0.72      2723
weighted avg       0.73      0.72      0.72      2723
```

Figure 2. Classification Report for default parameters

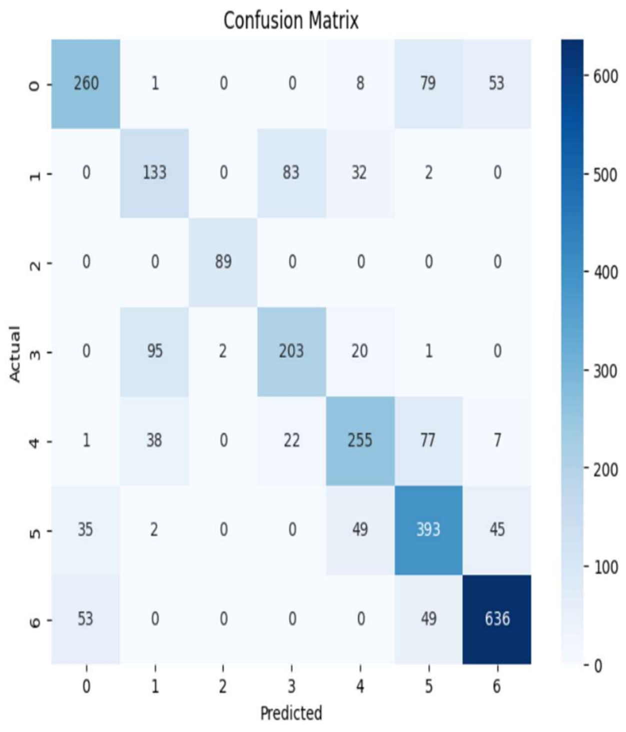


Figure 3. Confusion matrix for default parameters

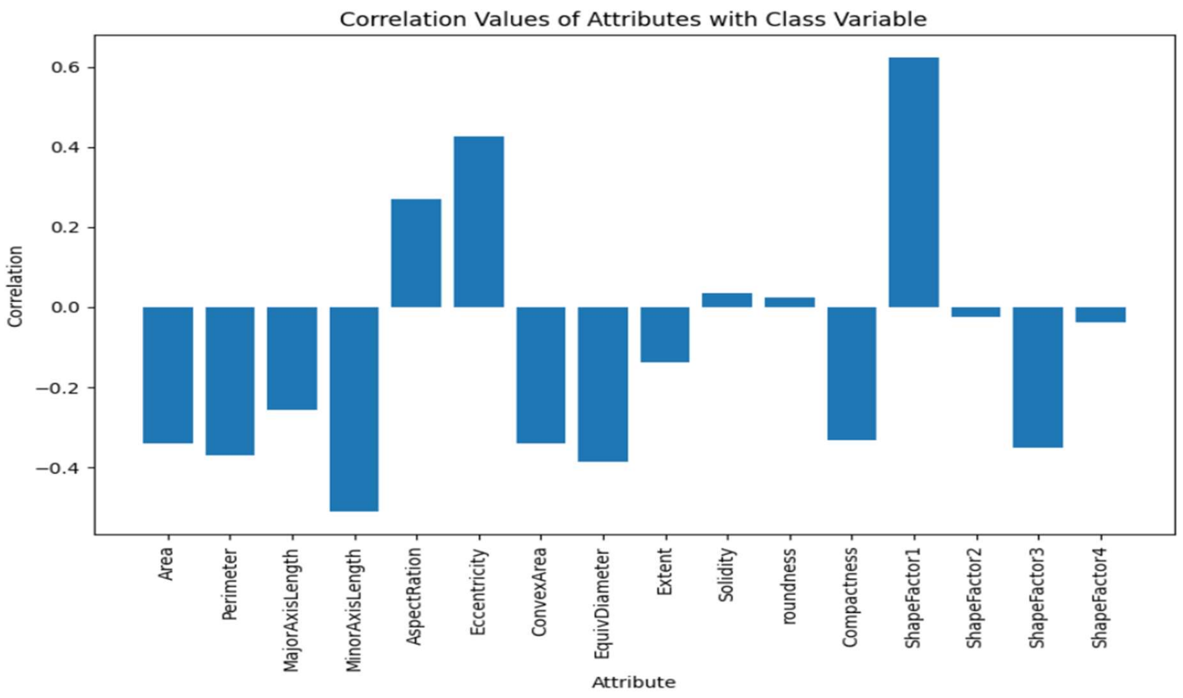


Figure 4. Correlation values of Attributes with Class Variable

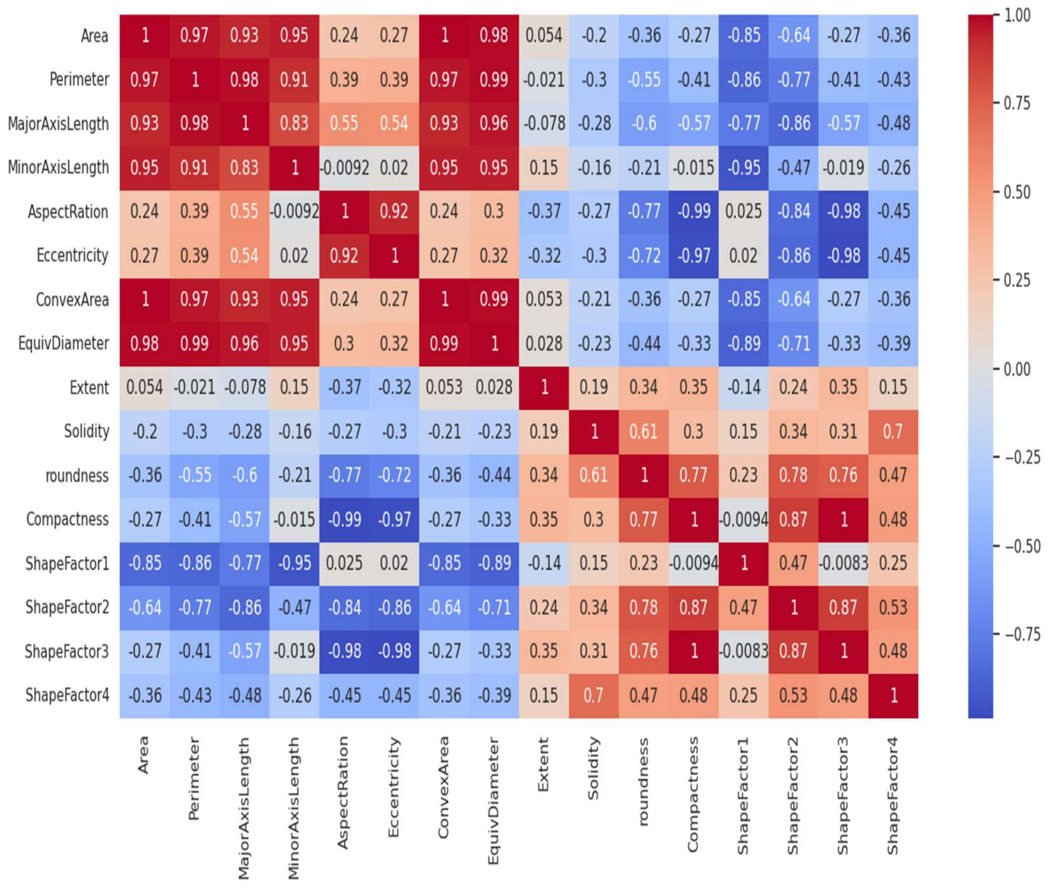
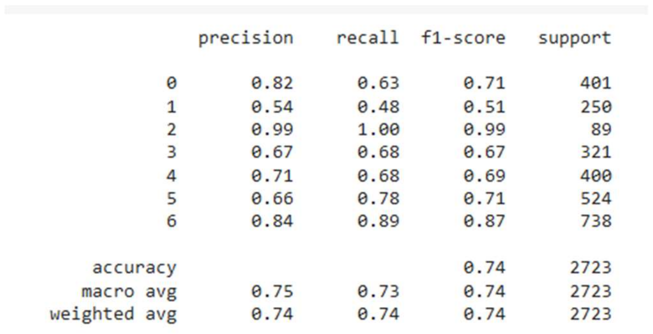


Figure 5. Correlation as a Heatmap

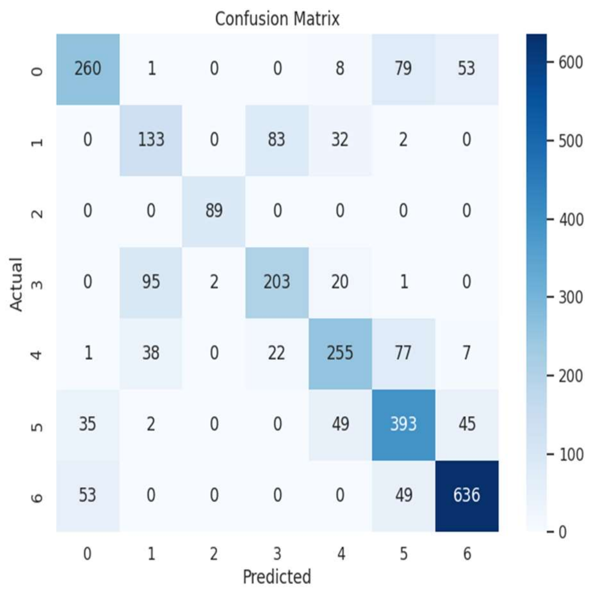


```
              precision    recall  f1-score   support

           0       0.82      0.63      0.71       401
           1       0.54      0.48      0.51       250
           2       0.99      1.00      0.99        89
           3       0.67      0.68      0.67       321
           4       0.71      0.68      0.69       400
           5       0.66      0.78      0.71       524
           6       0.84      0.89      0.87       738

    accuracy                           0.74      2723
   macro avg       0.75      0.73      0.74      2723
weighted avg       0.74      0.74      0.74      2723
```

Figure 6. Classification Report for Customized Parameters



Actual \ Predicted	0	1	2	3	4	5	6
0	260	1	0	0	8	79	53
1	0	133	0	83	32	2	0
2	0	0	89	0	0	0	0
3	0	95	2	203	20	1	0
4	1	38	0	22	255	77	7
5	35	2	0	0	49	393	45
6	53	0	0	0	0	49	636

Figure 7. Confusion Matrix for customized Parameters

B. CODE

```
import pandas as pd
import numpy as np
from scipy.io import arff
from google.colab import drive
from sklearn.preprocessing import
StandardScaler
from sklearn.model_selection import
train_test_split
from sklearn.neighbors import
KNeighborsClassifier
import random

drive.mount('/content/drive')
```

```
file_path =
'/content/drive/MyDrive/Dry_Bean_Dataset.
arff'

data, meta = arff.loadarff(file_path)
df = pd.DataFrame(data)
df

unique_values = df['Class'].unique()
print(unique_values)



mapping = {value: index for index, value
in enumerate(unique_values)}
df['Class'] = df['Class'].map(mapping)
df

for i in range (0,17):
  data_type = df.iloc[:, i].dtype
  print(data_type)

x = df.iloc[:,0:16]
y = df.iloc[:,-1]

x


y

x.describe()

scaler = StandardScaler()
scaler.fit(x)
x_scaled = scaler.transform(x)
x_scaled_df = pd.DataFrame(x_scaled,
columns=x.columns)
x_scaled_df

x_scaled_df.describe()

x_train, x_test, y_train, y_test =
train_test_split(x, y, test_size=0.2,
random_state=43)

neigh = KNeighborsClassifier()
neigh.fit(x_train, y_train)
```

```
y_pred = neigh.predict(x_test)

from sklearn.metrics import
classification_report,confusion_matrix
print(classification_report(y_test,
y_pred))

import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix as a
heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

correlation_matrix =
x_scaled_df.corrwith(y)

correlation_df =
pd.DataFrame({'Attribute':
correlation_matrix.index, 'Correlation':
correlation_matrix.values})
correlation_df['Correlation'] =
correlation_df['Correlation']

plt.figure(figsize=(10, 6))
plt.bar(correlation_df['Attribute'],
correlation_df['Correlation'])
plt.xlabel('Attribute')
plt.ylabel('Correlation')
plt.title('Correlation Values of
Attributes with Class Variable')
plt.xticks(rotation=90)
plt.show()

correlation_matrix

sns.set(rc = {'figure.figsize':(16,8)})
sns.heatmap(x_scaled_df.corr(), annot =
True, fmt = '.2g', cmap='coolwarm')

neigh1 =
KNeighborsClassifier(n_neighbors=10,
weights = 'distance')
neigh1.fit(x_train, y_train)

y_pred1 = neigh1.predict(x_test)

print(classification_report(y_test,
y_pred1))

cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix as a
heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```