Dry Bean Classification Using Multi-Layer Perceptron

1. Introduction

Machine learning is a branch of artificial intelligence that is focused on developing algorithms and models that enable computers to learn and acquire knowledge through data, and then make predictions or decisions Machine learning algorithms are used in a wide range of applications such as speech recognition, agriculture, healthcare and many more.

In the given scenario, the primary aim is to develop machine learning models to classify the different varieties of dry beans. Our requirement is to classify the dry beans into 7 distinct types of beans that are provided in the given dataset. This dataset is based on the study performed on 7 unique types of beans based on their physical attributes.

This report will focus on the implementing machine learning model to perform multiclass classification of seven types of dry beans from the provided dry bean dataset. In this report, I will be implementing Multi-layer Perceptron and K nearest neighbors. In this report we will discuss the working of these models and evaluate each model using evaluation metrics. Then we will discuss the best model among the two models.

2. Problem Outline

In the agriculture sector, there are diverse types of dry beans that are cultivated around the world. Seed quality and type are crucial factors for crop production. So, determining the best seed species for cultivation is a major concern for farmers and the market. Classification of the distinct genetic types of dry beans is one of the main processes in crop production.

Our problem is to classify the different genetic varieties dry bean provided in the database. There are 7 different varieties of dry beans that are identified in the given dataset. Based on the physical features such as size, area, shape given in the dataset, we must classify it into several types of dry beans. These 7 different varieties of dry beans that are identified are Barbunya, Dermason, Cali, Seker, Sira, Horoz, Bombay.

3. Dataset

The given dataset is based on the study which concentrated on seven different varieties of dry bean and their physical attributes. This study focused on the features such as form, shape, type, and structure, which are crucial in the market context. For the research, the images of 13,611 grains of 7 different varieties captured with high resolution camera were considered. Segmentation and feature extraction methods were performed on the obtained images which produced 16 features out of which 12 described dimension and others included shape form. The 16 features that were considered are area, perimeter, major axis length, minor axis length, aspect ratio, eccentricity, convex area, equivalent diameter, extent, solidity, roundness, compactness. Apart from 16 features we also obtained 4 shape features which are Shape factor 1, Shape factor 2, Shape Factor 3 and Shape Factor 4. These features are used to distinguish 7 varieties of beans which are Cali, Horoz, Dermason, Seker, Bombay, Barbunya, Sira.

Exploratory Data Analysis and the supervised model is created using the dataset. Since the dataset doesn't include any null values or duplicate values, we don't need to carry out data cleaning process.

Sorting and quality control are the two real-world applications in which the dataset can be used. Among the limitations of the dataset are dataset only takes into consideration only seven registered types which doesn't represent full diversity beans from different varieties may exhibit similar size and shape,

the dataset may be imbalanced, and the dataset is based on Turkey, so it is subject to bias and can be hard to generalize to different regions or cultivation practices.

4. Exploratory Data Analysis and Data Preprocessing

By conducting Exploratory analysis, we get valuable insight and better understanding of the data provided in the dataset. We get the overview of the data and understand the class distribution and correlations between the features.

In the EDA (Exploratory Data Analysis), we get basic statistical summary like mean, median, range etc. of each feature. As many algorithms work with numerical data rather than categorical data, LabelEncoder converts target labels into integer values (0 to n_classes-1). We get to know that data is imbalanced by plotting bar graph of count of grains in each variety shown in Fig 1. To get comprehensive visual of relationships and patterns among the features we are plotting correlation heatmap shown in Fig 2and cluster map, respectively.

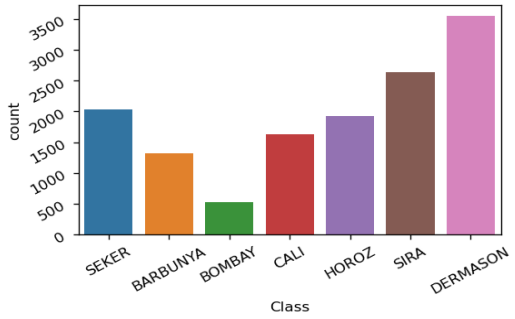


Fig 1 Bar graph of no of data for each class

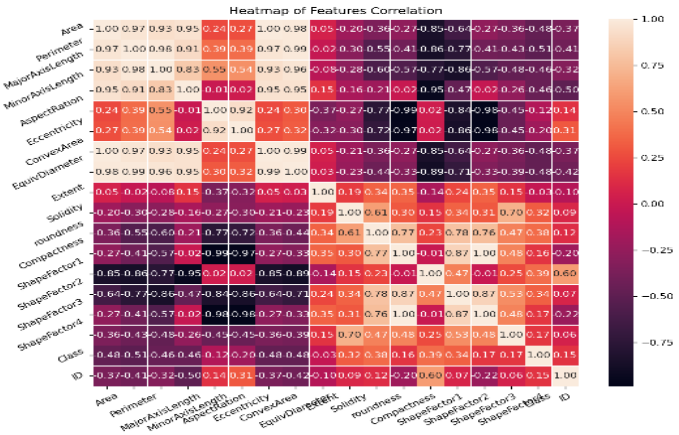


Fig 2 Heat map

For the given data, we must separate label and features into different dataframe. We then have to standardize the features so that each feature has zero mean and standard deviation 1 by scaling to unit variance. After this we split dataset into training and testing sets.

5. Model Creation

We will investigate the model used to classify different varieties of dry bean and evaluate the performance of the model.

5.1 Multi-Layer Perceptron

Multi-Layer Perceptron is a type of artificial neural networks (ANN) that consists of multiple layers of interconnected nodes also known as perceptron. Perceptron resembles a neuron in the human body. Each perceptron like a biological neuron receives input, applies activation function to the weighted sum of those input and passes result to output and from output layer each node passes the probability of each class and then predict label accordingly. MLP is a type of supervised learning model. It is a feedforward neural network hence the information flows in forward direction. MLP is used to classify different varieties of dry beans according to their features based on underlying patterns and relationships in the dataset. The MLP's multiple layers and non-linear activation functions enable it to

map the input features to higher-dimensional spaces, where the classes can be more effectively separated.

Each perceptron acts as a linear classifier and creates a hyperplane and the mathematical equation of the hyperplane is

$$x_n w=0$$

Here x_n is the input feature for data instance n and w is weight vector which include bias (w+b). The output of each perceptron in MLP can be calculated as:

$$z = \sum_{i=1}^{n} x_i w_i + b$$

Where z is z is the weighted sum of the inputs and biases, w represents the weights associated with each input feature, x denotes the input features, b is the bias term.

MLP has three layers: an input layer, hidden layer, and output layer. The neurons are arranged such that it takes input and perform activation function whose output is fed as input to hidden layer and the output of this layer in turn is given into output layer. MLP structure for the given problem is

- **Input layer:** There are 16 features for the dry beans in the dataset that we are considering which is used as input parameters of MLP
- **Hidden layer:** For the problem when we optimized hyperparameters we got number of nodes as 6 in the hidden layer
- **Output layer**: Output layer has 7 nodes which gives probability of each type of dry bean labelled in dataset.

Backpropagation is a key algorithm used to train multi-layer perceptron (MLP) models. It involves updating the weights and biases of the MLP by propagating the error gradients backward through the network. By repeatedly adjusting the weights and biases based on the calculated error gradients, backpropagation helps the MLP to minimize the difference between the predicted output and the actual output.

MLP model captures complex patterns and nonlinear relationships in the data. The MLP can automatically learn and extract relevant features from the dataset. An MLP can potentially identify meaningful representations and combinations of the 16 features extracted from data thus improving classification accuracy. Due to this it is an apt model for the classification of dry beans. But the MLPs may have limitations, such as the potential for overfitting and sensitivity to hyperparameter choices.

The hyperparameters that we have chosen are optimized using Randomized Search CV which are:

- hidden_layer_sizes: this determines the number of neurons in the hidden layer which determines the architecture of the MLP. After optimizing the hyperparameters, we got no of nodes as 6
- Activation: The activation function brings non-linearity into the output of a perceptron. The optimal value for activation that we got is tanh, which is hyperbolic tan function, returns f(x) = tanh(x).
- Solver: This parameter refers to the optimization algorithm used to train the multi-layer perceptron (MLP) model we used lbfgs. This solver uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm with limited memory. It is an efficient optimizer for small and medium-sized datasets.
- Learning_rate: the learning rate is a hyperparameter that determines the step size at which the weights are updated during the training process. The learning rate is set to constant. It means constant learning rate.

- Random_state: this parameter is used to control the random number generator for initializing the weights of the MLP model. We have got optimal value as 4.

A number of neural network architectures can be used as alternatives to MLP, including Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), as well as traditional machine learning algorithms, such as Support Vector Machines (SVMs) and Random Forests.

5.2 KFold Cross Validation

Cross-validation is a technique used for evaluating how well a model performs and generalizes to unseen data. It divides the dataset into subsets or folds and trains the model on some folds while testing it on others. By repeating this process multiple times and averaging the performance, cross-validation gives a reliable estimate of the model's performance. It helps in identifying issues like overfitting or underfitting and allows for fair comparisons between different models or parameters. As a result of cross-validation, the best model for real-world deployment can be selected. We have used KFold cross validation method for cross validation. It splits data into k fold. Here we have given k=5.

6. Model Evaluation

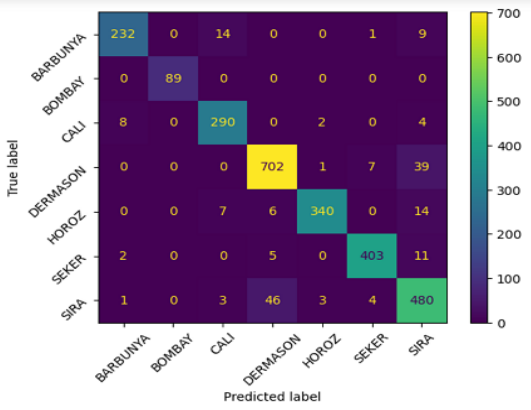
In this report, the evaluation is done using accuracy, precision, recall and F1 score, respectively. The MLP model evaluation is done using performance metrics, confusion matrices, classification report and cross-validation, the modes used in this model.

Performance metrics measure the model's accuracy and ability to correctly classify instances across different classes and provide valuable insights into its performance on a per-class basis and overall performance. Confusion matrix compares the values of the labelled class against the predicted class thus it is suitable for multiclass evaluation. It helps to identify the number of test inputs that are correctly classified and misclassified. The classification report offers a concise overview of the model's performance for individual classes and helps in identifying classes where the model excels or requires improvement. Cross-validation is employed to assess the model's performance on multiple subsets of the data. It helps in estimating the model's generalization ability and provides a more robust evaluation by mitigating the impact of specific train-test splits. Here, for this analysis the value of k is taken as 5.

Fig 3: Classification Report

Dry Bean Type	precision	recall	f1-score	support
BARBUNYA	0.95	0.91	0.93	256
BOMBAY	1	1	1	89
CALI	0.92	0.95	0.94	304
DERMASON	0.92	0.94	0.93	749
HOROZ	0.98	0.93	0.95	367
SEKER	0.97	0.96	0.96	421
SIRA	0.86	0.89	0.88	537

Fig 4: Confusion matrix (Optimized MLP)



The accuracy, precision, recall and F1 score achieved by MLP are 93.13%, 94.55%, 93.92% and 94.21 % respectively. From the classification report, we can see that for Bombay class the model performed well with all metrics values as 1, whereas for class Sira the performed worse with 0.87.099,0,0.88 for precision, recall and F1, respectively. For the rest of the classes a good score was obtained for precision,

recall and F1 as observed in Fig 3. The K-Fold cross-validation was performed with a k value of 5 and an output obtained with an F1 score of 0.9424, 0.9464, 0.9412, 0.9398 and 0.9229 on different folds.

As shown in Fig 4, the value along the diagonal is correctly predicted and the others are misclassified. Here, Dermason has the highest number of correctly predicted classes i.e., 702 correct predictions, the percentage of correctly predicted value is highest for Bombay while Sira has the lowest percentage.

Comparing these values with the study conducted by (Murat Koklu,2020), the accuracy, precision, recall and F1 scores have increased since the hyperparameters were optimized.

7. Extra Task:

7.1 K nearest Neighbors

K nearest Neighbors is a type of supervised machine learning algorithm which is used classification and regression. It is a non-parametric intuitive learning model. KNN is a type of instance-based learning, where predictions for new data points are made based on the "k" nearest training examples in the feature space. KNN is based on the idea that similar things will remain in close proximity to each other.

In the KNN algorithm, the training data consists of labeled examples, where each example is a data point with its corresponding class. The algorithm will store and memorize the training dataset while training, so that during prediction it can be used to find nearest neighbors. During classification, when new unlabeled data that needs to be predicted is given as input, the algorithm calculates the distance between this point and all the training examples. After the distance is calculated, the algorithm selects k training datapoints that are nearest to the unlabeled datapoint. The algorithm counts the occurrences of each class among the "k" nearest neighbors and assigns the highest count to the new point. After determining the class or regression value for the new data point, the algorithm predicts the class label. In the case of evaluation, it compares the predicted labels or values with the true labels or values to assess the algorithm's performance.

So, in KNN model it learns the training dataset for the given problem and memorizes it and when the new data is given to the model it calculates the distance between the point and the training examples. After that algorithm selects k nearest datapoints assigns the highest count to the input. Then it predicts which of the class of dry bean the new datapoint belongs to.

KNN, unlike MLP, doesn't learn from explicitly learn from training dataset it memorizes the training dataset makes predictions based on the similarity of new data to the stored instances. But in MLP model are trained using backpropagation. Due to lazy learning the KNN doesn't have an explicit complex model and cannot capture the intricate patterns.

MLP is able to learn complex, non-linear relationships between input features and target variables and capture hidden patterns in the data but KNN assumes a simple proximity-based decision boundary and hence cannot capture complex patterns. After being trained with labeled data the MLP can generalize well with unseen data, but this is not the case with KNN. MLP can more effectively handle high-dimensional data than KNN. MLP can handle Imbalance dataset better than KNN. Hence due to these advantages of MLP over KNN, it can be justified that MLP is a better choice than KNN.

Hyperparameters

For our classification model after optimizing using randomized Search CV, we got optimal value for n_neighbors as 12 which indicates the number of points that are considered as neighbors. For

calculating distance between the 2 datapoints the optimal value we obtained is Euclidean distance. For weight parameter that is used in prediction we got the optimal value as `distance`

7.2 Model evaluation

Here for evaluating the performance of KNN we are using performance metrics, confusion matrices, classification report and cross-validation like what we have used for primary model evaluation. The achieved accuracy, precision, recall and F1 score for KNN are 92.72%, 94.31%, 93.32% and 93.75 % which is lower than those values of MLP model. From the classification report, Bombay class performed well with precision, recall, F1 score as 1 for all the score whereas for Sira class the model performed bad. It achieved precision of 0.87, recall and F1 score each with 0.88. MLP performs slightly better than KNN because it can handle high-dimensional and imbalanced data better than KNN. The performance for both models is shown in Table 1. The performance of the model based on F1 score is shown in Fig 5

Performance	Accuracy	Precision	Recall	F1_score
ML Perceptron optimized	0.931326	0.945529	0.939282	0.942107
K nearest neighbours optimized	0.927286	0.943115	0.933261	0.937543

Table 1: performance of each model

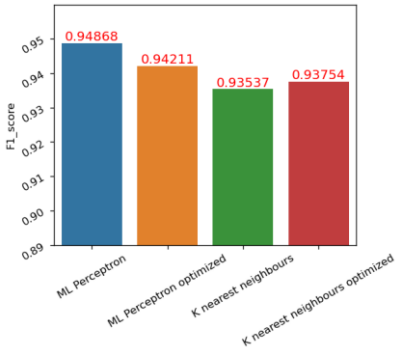


Fig 5: Bar graph of F1 score

8. Conclusion

In this study, we performed the MLP model to classify the data into 7 distinct types of dry beans. MLP captures the complex patterns and relationship among the features and also it performs well with high dimensional and imbalance data, it is ideal choice for classification. We also developed K nearest neighbors model. It gave a superior performance, but the MLP performed better than KNN. For MLP we got the accuracy, F1 score of 0.9313 and 0.9455 which is slightly better than MLP. Due to MLP interconnected neural network architecture it captures complex pattern thus resulting in high performance. We also optimized the hyperparameter using Randomized Search CV, thus giving a better result than those performed by M. Koklu and I.A. Ozkan work.

9. References

- Murat Koklu and I.A. Ozkan, 2020, 'Multiclass classification of dry beans using computer vision and machine learning techniques', Elsevier,
- Dry bean dataset**,** https://archive-beta.ics.uci.edu/dataset/602/dry+bean+dataset
- Tuning the hyperparameter, https://scikit-learn.org/stable/modules/grid_search.html
- Jason Brownlee, 2016, 'Crash Course on Multi-Layer Perceptron', https://machinelearningmastery.com/neural-networks-crash-course/
- Multi-Layer Perceptron, scikit learn, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- KNN classifier, scikit learn, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

- Label Encoder, scikit learn, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
- Standard Scaler, scikit learn, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html