DRY BEANS CLASSIFICATION PREDICTION

A Data Mining Project

ABSTRACT

This research utilizes Data Mining models to accurately classify Dry Bean varieties, enhancing crop production through improved marketing and production management.

Submitted By,

Ashlesh Umesh Khajbage

Contents

Summary

Project Objective

Seed categorization helps in easier crop production. It is necessary to classify these seeds which will assist in improved marketing and production management. To achieve this objective, several types of Dry Beans are used in a research to perform the classification. The purpose of this project is to design Data Mining models that will help classify the varieties of Dry Beans utilizing its features. These results could help in providing a method to distinguish the different varieties of beans. Using accuracy performance measures, the best possible classification model can be suggested.

Techniques/Algorithms Used

All the techniques used in this project are under the Supervised Learning algorithm where the goal is to predict the single classification outcome variable and evaluate the accuracy of the predictions. Therefore, we would be using techniques: Nominal Logistic Regression, Decision Tree and Neural Networks to perform the Data Mining Tasks.

Data Mining Results

It is expected that the results obtained from the classification model need to have good accuracy rate (low misclassification rate). Hence, the confusion matrices of Training and Validation Partition should be close to each other thereby avoiding the possibility of overfitting. It is also important to interpret and understand the results to determine the best classification model. Additionally, the recommended model should be able to predict the classification of new Dry Bean records.

Introduction

Seed classification in agricultural systems is important for both marketing and production. To enhance this purpose, understanding of the seed quality and variety is required. There is a wide range of genetic diversity of dry beans and this study involves a method for obtaining uniform seed varieties from crop production. Here, images of 13,611 grains of 7 different registered dry beans and 16 features were taken with a high-resolution camera.

In this dataset, there are 13,611 records with 16 predictors, that depict dimensions and shape of dry beans, and 1 outcome variable with 7 different classes, which are the varieties of dry beans. The objective of the project is to classify and predict the varieties of dry beans using various data mining methods and techniques.

The data set is obtained from the UCI Machine Learning Repository which is Center for Machine Learning and Intelligent Systems. Following are the attributes and its descriptions of the data set:

```
Attribute Information:
1. Area (A): The area of a bean zone and the number of pixels within its boundaries.
2. Perimeter (P): Bean circumference is defined as the length of its border.
3. Major axis length (L): The distance between the ends of the longest line that can be
   drawn from a bean.
4. Minor axis length (l): The longest line that can be drawn from the bean while standing
   perpendicular to the main axis.
5. Aspect ratio (K): Defines the relationship between L and l.
6. Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7. Convex area (C): Number of pixels in the smallest convex polygon that can contain the
   area of a bean seed.
8. Equivalent diameter (Ed): The diameter of a circle having the same area as a bean
   seed area.
9. Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10.Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell
   to those found in beans.
11.Roundness (R): Calculated with the following formula: (4piA)/(P^2)
12.Compactness (CO): Measures the roundness of an object: Ed/L
13.ShapeFactor1 (SF1)
14.ShapeFactor2 (SF2)
15.ShapeFactor3 (SF3)
16.ShapeFactor4 (SF4)
17.Class (Seker, Barbunya, Bombay, Cali, Dermason, Horoz and Sira)
```

MAIN CHAPTER

A Data Mining Project must have its purpose defined along with a data set. The data set must be pre-processed to apply various Machine Learning algorithms to learn directly from data.

Obtain Data for Analysis

The dataset (Dry_Bean_Dataset.csv) for the project is obtained from the UCI Machine Learning Repository. Seven different types of dry beans were used in this research, considering the features such as form, shape, type, and structure by the market situation. A total of 16 features; 12 dimensions and 4 shape forms, were obtained from the grains.

Explore, Clean and Preprocess Data

Prior employing any Data Mining Machine Learning algorithm, it is essential to clean and preprocess data that will satisfy the requirements of the algorithms.

Dry beans dataset dimensions are as follows:

```
In [16]: np.shape(df)
Out[16]: (10834, 17)

In [17]: df['y'].unique()
Out[17]: array(['HOROZ', 'SEKER', 'DERMASON', 'SIRA', 'BARBUNYA', 'CALI', 'BOMBAY'],
               dtype=object)
```

- There are 7 unique target classes

Below are the columns/variables in the dry bean dataset. All are already one-word variables. all column data types are numeric (‘int64’ or ‘float64’) except ‘Class’ column which is a nominal variable with categorical data.

```
In [7]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10834 entries, 0 to 10833
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             10834 non-null  int64
 1   Perimeter        10834 non-null  float64
 2   MajorAxisLength  10834 non-null  float64
 3   MinorAxisLength  10834 non-null  float64
 4   AspectRation     10834 non-null  float64
 5   Eccentricity     10834 non-null  float64
 6   ConvexArea       10834 non-null  int64
 7   EquivDiameter    10834 non-null  float64
 8   Extent           10834 non-null  float64
 9   Solidity         10834 non-null  float64
 10  roundness        10834 non-null  float64
 11  Compactness      10834 non-null  float64
 12  ShapeFactor1     10834 non-null  float64
 13  ShapeFactor2     10834 non-null  float64
```

- No missing values in the training dataset but we have to take into consideration if any missing data may appear in the test set.

Some Data Mining Techniques may require the data types to be changed from an object to a 'category'. In this project, the 'Class' column needs to be updated to a 'category' data type wherever required.

Note: There are no outliers and missing values in this dataset.

Data Mining Tasks

The main task involved in this project is 'Classification'. The outcome variable : 'Class' has different types of Dry beans and has rest of the columns as predictors.

Partition Data

To prevent the overfitting of data, dataset is partitioned into two parts, Training and Validation partition. Here, Training partition is used to train and develop the classification model. On the other hand, Validation partition is used to implement the model and evaluate its performance on new data. In this project, data set is partitioned as 60% training partition and 40% validation partition.

Data Mining Techniques

Nominal Logistic Regression

A logistic regression model is used to make predictions (classifications) for a categorical variable based on identifying probabilities of categorical outcomes, and therefore, can be used in this project. Since it is a multi-classification outcome variable, Nominal Logistic Regression Technique is used.

Data Type Conversion:

It is required to change 'object' datatype of class variable into 'category' datatype. Here, class column is not split into dummy variables because it is being used as an outcome variable. Below are the preprocessed datatypes of all variables.

```
In [8]: df.info()
```

```
---  ------           --------------  -----
 0   Area             10834 non-null  int64
 1   Perimeter        10834 non-null  float64
 2   MajorAxisLength  10834 non-null  float64
 3   MinorAxisLength  10834 non-null  float64
 4   AspectRation     10834 non-null  float64
 5   Eccentricity     10834 non-null  float64
 6   ConvexArea       10834 non-null  int64
 7   EquivDiameter    10834 non-null  float64
 8   Extent           10834 non-null  float64
 9   Solidity         10834 non-null  float64
 10  roundness        10834 non-null  float64
 11  Compactness      10834 non-null  float64
 12  ShapeFactor1     10834 non-null  float64
 13  ShapeFactor2     10834 non-null  float64
 14  ShapeFactor3     10834 non-null  float64
 15  ShapeFactor4     10834 non-null  float64
 16  y                10834 non-null  object
dtypes: float64(14), int64(2), object(1)
memory usage: 1.5+ MB
```

Here, the 7 types of 'Class' variable is replaced with numbers from 0 to 6 for classification.

Generalized Linear Model

Since all the variables in the dry bean data set are the individual characteristics of dry beans, calculated from the high-resolution images, there are no unneeded variables. Below figure shows the Generalized Linear Model Regression Results

Generalized Linear Model Regression Results

Dep. Variable:	Class	No. Observations:	8166
Model:	GLM	Df Residuals:	8150
Model Family:	Binomial	Df Model:	15
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	nan
Date:	Sat, 15 May 2021	Deviance:	nan
Time:	21:17:08	Pearson chi2:	3.57e+20
No. Iterations:	100		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Area	3.365e+14	6828.840	4.93e+10	0.000	3.36e+14	3.36e+14
Perimeter	-6.857e+15	2.05e+05	-3.34e+10	0.000	-6.86e+15	-6.86e+15
MajorAxisLength	-5.046e+15	2.62e+06	-1.92e+09	0.000	-5.05e+15	-5.05e+15
MinorAxisLength	2.786e+16	4.7e+06	5.92e+09	0.000	2.79e+16	2.79e+16
AspectRation	1.384e+19	1.94e+08	7.14e+10	0.000	1.38e+19	1.38e+19
Eccentricity	-1.623e+19	1.54e+08	-1.05e+11	0.000	-1.62e+19	-1.62e+19
ConvexArea	-9.096e+13	6593.449	-1.38e+10	0.000	-9.1e+13	-9.1e+13
EquivDiameter	-1.84e+17	7.21e+06	-2.55e+10	0.000	-1.84e+17	-1.84e+17
Extent	-6.426e+17	1.68e+07	-3.84e+10	0.000	-6.43e+17	-6.43e+17
Solidity	1.18e+19	5.82e+08	2.03e+10	0.000	1.18e+19	1.18e+19
roundness	8.005e+17	1.27e+08	6.31e+09	0.000	8.01e+17	8.01e+17
Compactness	1.828e+20	1.49e+09	1.23e+11	0.000	1.83e+20	1.83e+20
ShapeFactor1	-3.017e+21	1.62e+10	-1.86e+11	0.000	-3.02e+21	-3.02e+21
ShapeFactor2	-1.491e+21	4.45e+10	-3.35e+10	0.000	-1.49e+21	-1.49e+21
ShapeFactor3	-1.106e+20	8.52e+08	-1.3e+11	0.000	-1.11e+20	-1.11e+20
ShapeFactor4	-3.788e+19	1.34e+09	-2.82e+10	0.000	-3.79e+19	-3.79e+19

As shown, p-values of all the predictor variables are 0.000 which is less than 0.05, therefore all the predictor variables are statistically important. Hence, no dimension reduction is needed.

LogisticRegression()

For developing Nominal Logistic Regression model, we are using *LogisticRegression()* function from *scikit-learn* package from Python. Since, 'Class' variable is a nominal categorical variable , we are setting the parameter *multi_class = 'multinomial'*. Below is the model function used:

```
In [42]: model_logistic = LogisticRegression(multi_class='multinomial', solver='lbfgs')
         model_logistic.fit(X_train, y_train)
```

As shown, to train the model, it uses data of training partition.

Following are the Intercepts and Coefficients for the Nominal Logistic Regression model.

```
Nominal Logistic Regression for Dry Beans dataset
  Intercepts [-0.00057134 -0.00053085  0.00012553  0.00075401 -0.000334    0.00044613
  0.00011052]
```

```
Coefficients [[-3.89911211e-04  1.10031754e-01 -1.52431555e-01 -5.80263236e-02
 -1.11701513e-03 -4.82566274e-04  4.51497345e-04 -7.94520937e-02
 -5.61511454e-04 -5.64018258e-04 -1.13350935e-03 -3.70734568e-04
 -5.53052536e-06 -9.89827629e-07 -2.36090513e-04 -4.59549926e-04]
[ 7.91331392e-04 -2.88955528e-01 -1.09272604e-01 -6.71195271e-02
 -8.51623898e-04 -3.99600810e-04  3.63022975e-03 -8.51145642e-02
 -3.92780553e-04 -5.23723818e-04 -4.62877158e-04 -4.22786417e-04
 -4.17787540e-06 -1.08126387e-06 -3.39180931e-04 -5.28595970e-04]
[-3.20373958e-03 -4.34361227e-02  1.05434013e-01  1.75148418e-03
  6.53228244e-04  2.67801705e-04  3.36686069e-03  3.46254843e-02
  5.17203382e-04  1.24316126e-04  3.15025475e-04 -4.49027089e-05
  1.13587301e-06 -7.70880187e-07 -1.52576897e-04  6.49822738e-05]
[ 3.79286117e-03  6.90956150e-02  6.82810348e-02  8.47540751e-02
  6.45047951e-04  9.02102897e-04 -5.88234880e-03  9.30249191e-02
  3.01549759e-04  7.41416278e-04  1.21327091e-03  6.22424100e-04
  6.87021670e-06  1.09814674e-06  4.40475696e-04  7.66783347e-04]
[-4.92640658e-03  3.89612804e-02  2.39116550e-01 -2.00691791e-01
  2.60526301e-03  4.59896118e-04  4.03302200e-03 -5.14890234e-02
 -5.80950817e-04 -3.24001935e-04 -7.10732268e-04 -9.46289992e-04
  3.80760006e-06 -4.33114511e-06 -1.23635526e-03 -3.87542307e-04]
[ 4.95663990e-03  7.18765719e-02 -2.07846419e-01  2.29257641e-01
 -2.14201118e-03 -1.35456343e-03 -5.70764559e-03  5.07327459e-02
  3.66832635e-04  4.41212634e-04  6.88330666e-04  1.24973438e-03
 -2.34326709e-06  8.31870090e-06  1.81288424e-03  4.84582520e-04]
[-1.02077508e-03  4.24264299e-02  5.67189797e-02  1.00744415e-02
  2.07111008e-04  6.06929794e-04  1.08384609e-04  3.76725318e-02
  3.49657048e-04  1.04798973e-04  9.04917179e-05 -8.74447885e-05
  2.37978075e-07 -2.24373084e-06 -2.89156335e-04  5.93400610e-05]]
```

The logistic response functions for each class of the model are specified as below:

$$p(n) = \frac{e^{nx}}{1 + e^{0x} \ldots + e^{nx}}$$

where ***p(n)*** = Probability of class n. For ex: p(0) is probability of BARBUNYA class, p(1) is probability of BOMBAY class and so on.

e^{nx} = odds of outcome belonging to class n. For ex: e^{1x} is odds of outcome belonging to class 1 BOMBAY and so on.

Classification Probabilities

Following are the first 10 records depicting the classification predictions for the different classes of dry beans.

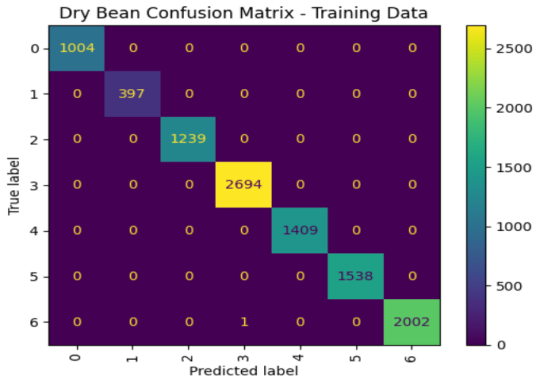
Classification for First 10 Records in Validation Data Set

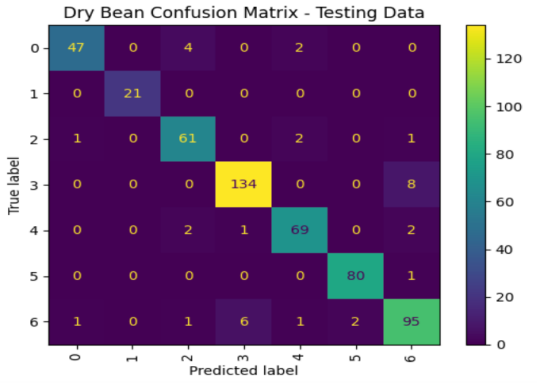
	Actual	p(0)	p(1)	p(2)	p(3)	p(4)	p(5)	p(6)	Classification
10057	6	0.0159	0.0	0.6125	0.0000	0.1549	0.0000	0.2167	2
13354	3	0.0001	0.0	0.0001	0.4724	0.0006	0.0010	0.5257	6
13141	3	0.0001	0.0	0.0000	0.6952	0.0000	0.0068	0.2979	3
8634	6	0.0004	0.0	0.0011	0.0070	0.2430	0.0000	0.7485	6
12579	3	0.0000	0.0	0.0000	0.7854	0.0000	0.1692	0.0453	3
11088	3	0.0000	0.0	0.0000	0.9968	0.0001	0.0000	0.0030	3
13111	3	0.0000	0.0	0.0000	0.6947	0.0007	0.0003	0.3042	3
9369	6	0.0006	0.0	0.0015	0.0004	0.6108	0.0000	0.3867	4
9826	6	0.0198	0.0	0.0567	0.0002	0.0055	0.0001	0.9178	6
13193	3	0.0000	0.0	0.0001	0.4825	0.0058	0.0001	0.5115	6

Here, classification 0 is BARBUNYA, 1 is BOMBAY, 2 is CALI, 3 is DERMASON, 4 is HOROZ, 5 is SEKER and 6 is SIRA.

Confusion Matrices for Nominal Logistic Regression

Following is the confusion matrix for Training and Validation partition for Nominal Logistic Regression.





For this model, the confusion matrices for the training and validation partitions show a very high accuracy of around 90%, and thus the trained logistic regression model fits well for the validation date set and can be used for classification of the dry beans. The misclassification rate for the training partition is 1 - 0.9166 = 0.0834 or 8.34%, and for the validation partition 1 - 0.9085 = 0.0915 or 9.15%. The accuracy of the model for the validation records is closer to that of the accuracy for the training records, and therefore, there is no overfitting in this case.

DECISION TREE

The Decision Tree technique in Data mining performs both classification and prediction. Here, Decision Tree is used to classify the "Class" of the bean, based on a set of predictors. Output will be represented by tree diagrams. Different models and algorithm will be used to build trees.

Cross-validation

Cross-Validation is used to evaluate variability in performance accuracy on different data partitioning options.

```
# Use cross_val_score() function to identify performance
# accuracy for 5 folds (cv=5) of cross-validation partitioning.
scores = cross_val_score(treeClassifier, train_X, train_y, cv=5)
```

A five-fold partition has been opted where each one has 20% of records. Each time, one of the folds is used as a validation set and remaining 4 folds serve as training set. Cross-validation is done using cross_val_score() function from scikit-learn library.

```
Performance Accuracy of 5-Fold Cross-Validation
Accuracy scores of each fold:  ['0.892', '0.897', '0.893', '0.889', '0.879']

Two Standard Deviation (95%) Confidence Interval for Mean Accuracy
Accuracy: 0.890 (+/- 0.012)
```

DecisionTreeClassifier()

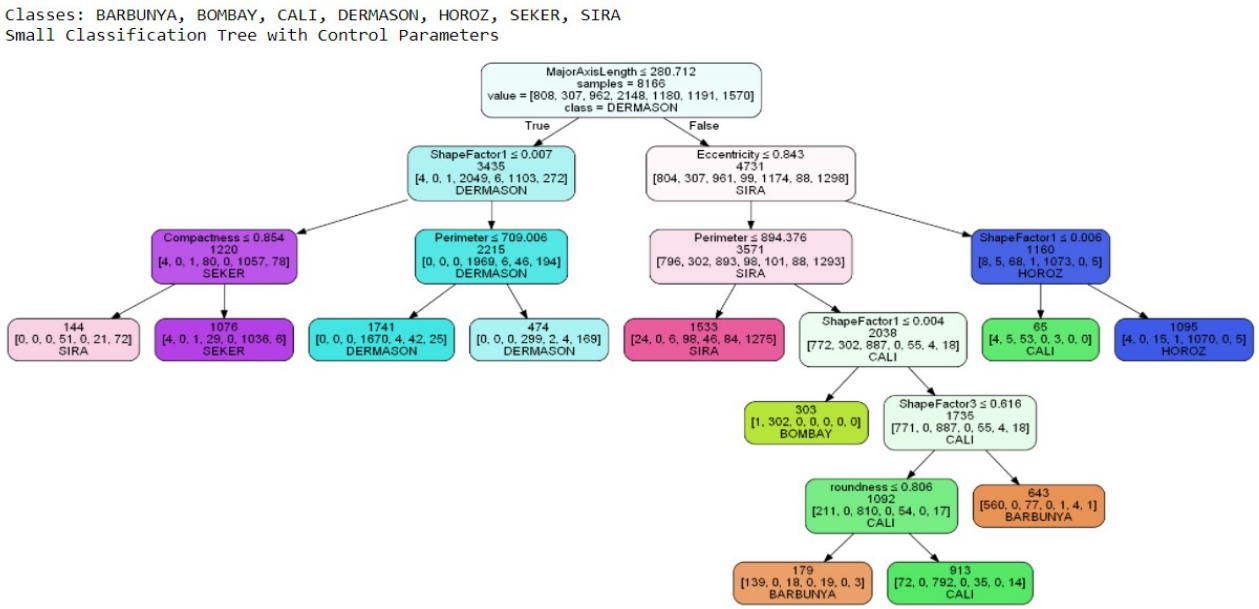
A classification tree model is trained using DecisionTreeClassifier() with the training data set and the following parameters:

- maximum depth (number of splits) equals 12.
- minimum impurity decreases per split of 0.01.
- minimum number of node records (samples) to split equals to 20.

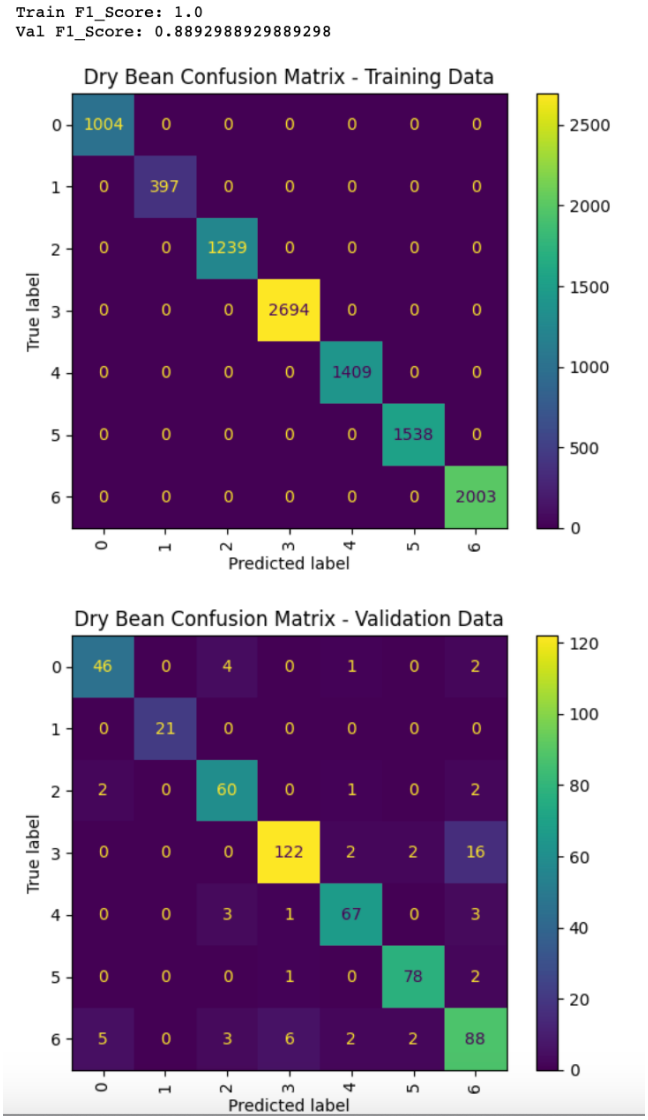
```
smallClassTree = DecisionTreeClassifier(max_depth=12,
        min_impurity_decrease=0.01, min_samples_split=20)
smallClassTree.fit(train_X, train_y)
```

Plot Decision Tree

Using plotDecisionTree() with the feature_names and class_names, classification tree is developed.



Confusion Matrix for DecisionTreeClassifier()



The accuracy for the training partition is rather high (0.8900 or 89.00%) and misclassification rate is 1- 0.8900 = 0.11 or 11%. This represents a very good fit of the classification tree. The accuracy values for both training and validation data sets are very close to each other, and thus the overfitting is not a problem for the trained classification tree.

Grid Search Algorithm

GridSearchCV() involves exhaustive search to find most accurate parameters and leads to the tree with the highest accuracy (smallest error).

```
# Define the hyperparameters and their possible values for the grid search
hyperparameters = {
    'hidden_layer_sizes': [(100,), (200,), (100, 100)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001, 0.01],
}

# Perform grid search using cross-validation
grid_search = GridSearchCV(classifier, hyperparameters, cv=5)
grid_search.fit(X_train, y_train)
```

Initial Score & Parameter Values

Firstly, the initial score and parameter values are predicted using the above function. Following are the list of values assigned to parameters to make predictions.

```
'max_depth': [10, 20, 30, 40],
'min_impurity_decrease': [0, 0.0005, 0.001, 0.005, 0.01],
'min_samples_split': [20, 40, 60, 80, 100],
```

Initial Prediction:

```
Initial score:0.9085
Initial parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.0005, 'min_samples_split': 20}
```

Improved Score & Parameter Values

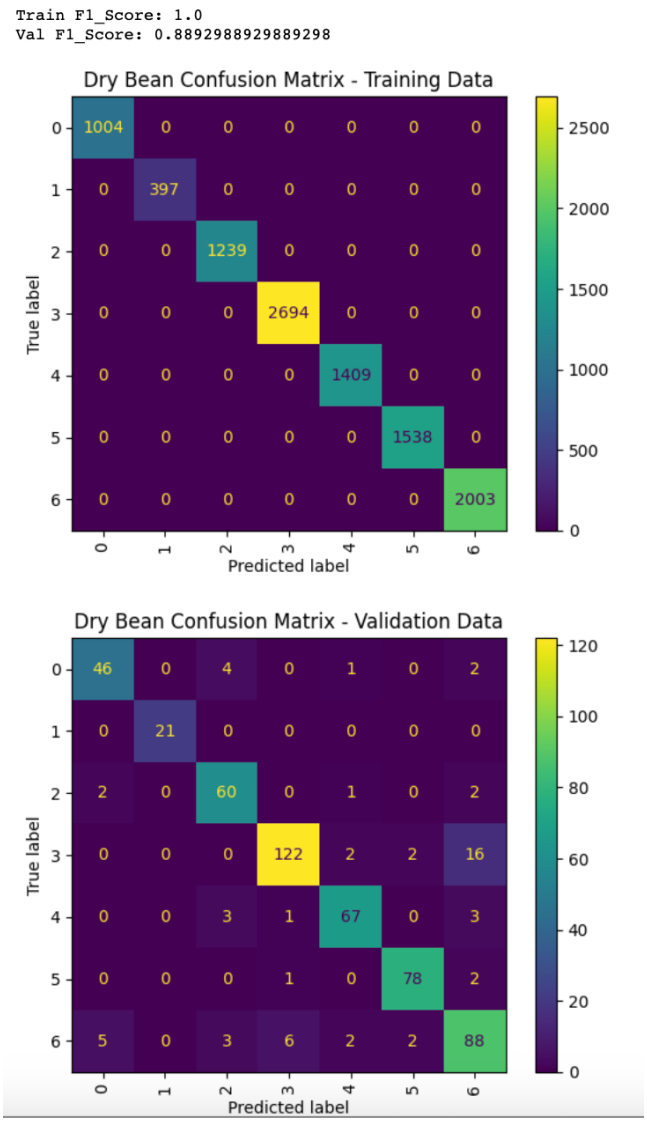
Then using same algorithm, the the classification tree control parameters are optimized with below parameters:

```
'max_depth': list(range(2, 20)),
'min_impurity_decrease': [0.0005, 0.001,0.005],
'min_samples_split': list(range(10, 30)),
```

Improved Prediction:

```
Improved score:0.9096
Improved parameters:  {'max_depth': 15, 'min_impurity_decrease': 0.0005, 'min_samples_split': 14}
```

Confusion Matrix for Grid Search



The accuracy for the training partition is rather high (0.889 or 88.9%) and misclassification rate is 1- 0. 889 = 0.111 or 11.1%. This represents a very good fit of the classification tree. The accuracy values for both training and validation data sets are very close to each other, and thus the overfitting is not a problem for the trained classification tree.

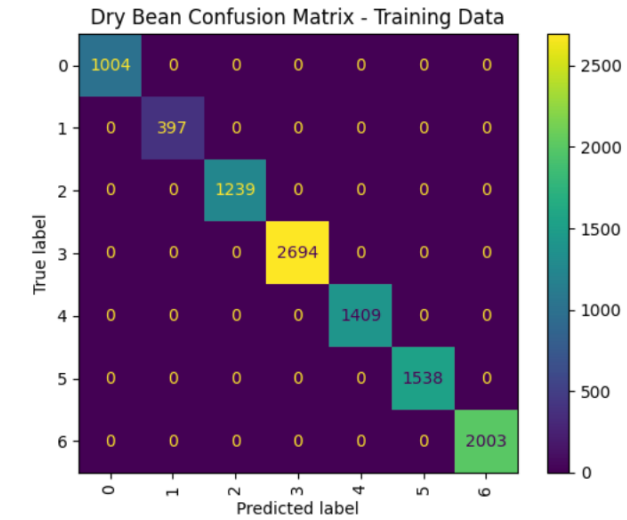
Random Forest

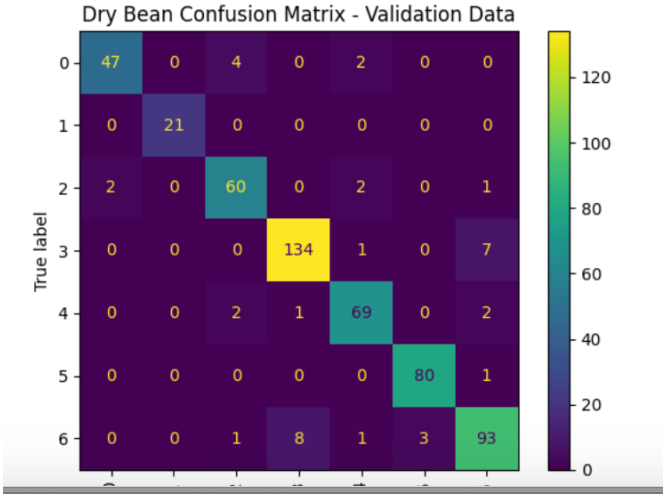
Results from multiple trees can be combined to improve performance with resulting in a model called Ensemble Tree model. Random Forest is one of the Ensemble Tree models, that combines the classifications/predictions and takes an average of multiple estimates (models), which is more reliable than just using a single estimate.

```
rf = RandomForestClassifier(n_estimators=500, random_state=1)
rf.fit(train_X, train_y)
```

Confusion matrix for Random Forest

```
Train F1_Score: 1.0
Val F1_Score: 0.9298892988929889
```





The accuracy for the training partition is rather high (1.0). This represents a very good fit of the classification tree. The accuracy values for both training and validation data sets are close to each other, and thus the overfitting is not a problem for the trained classification tree.

Importance scores

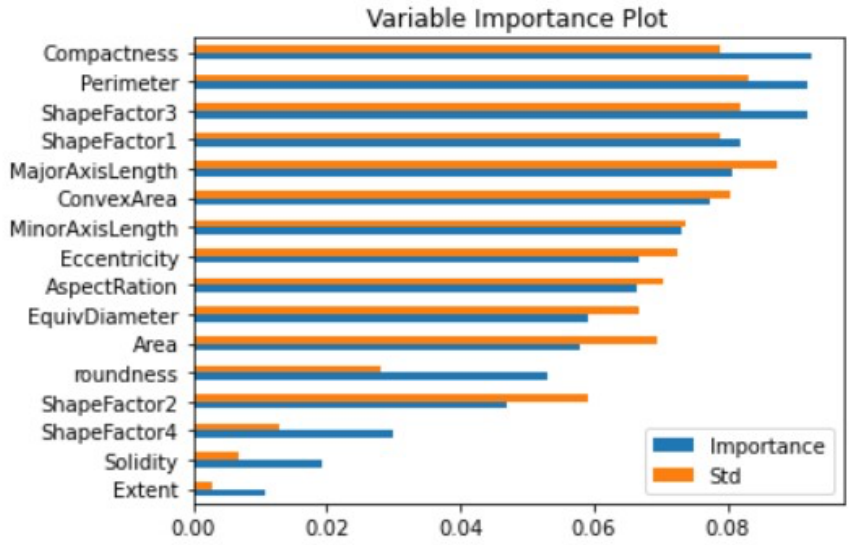
Variable Importance Scores for Random Forest

Variable Importance Scores for Random Forest

	Feature	Importance	Std
11	Compactness	0.092699	0.079035
1	Perimeter	0.091958	0.083172
14	ShapeFactor3	0.091917	0.081914
12	ShapeFactor1	0.081912	0.078824
2	MajorAxisLength	0.080599	0.087524
6	ConvexArea	0.077455	0.080435
3	MinorAxisLength	0.072978	0.073645
5	Eccentricity	0.066677	0.072481
4	AspectRation	0.066571	0.070333
7	EquivDiameter	0.059260	0.066783
0	Area	0.057875	0.069345
10	roundness	0.053095	0.028136
13	ShapeFactor2	0.046968	0.059117
15	ShapeFactor4	0.030051	0.012809
9	Solidity	0.019296	0.006805
8	Extent	0.010690	0.002975

Importance plot

Variable Importance Plot:



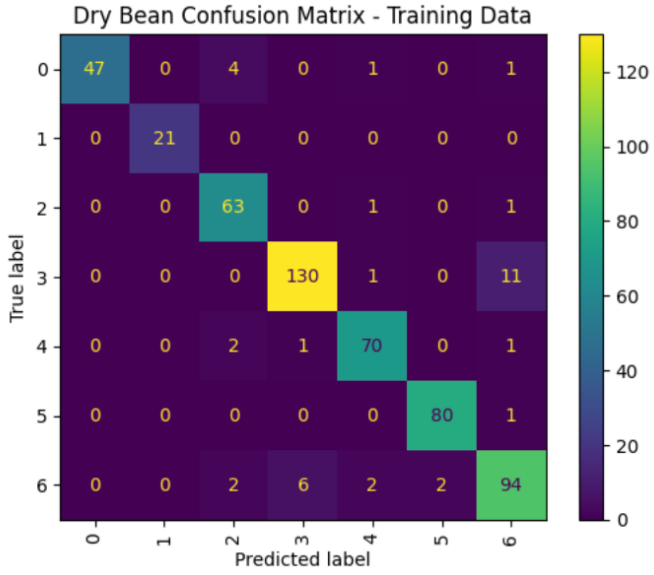
Boosted Tree

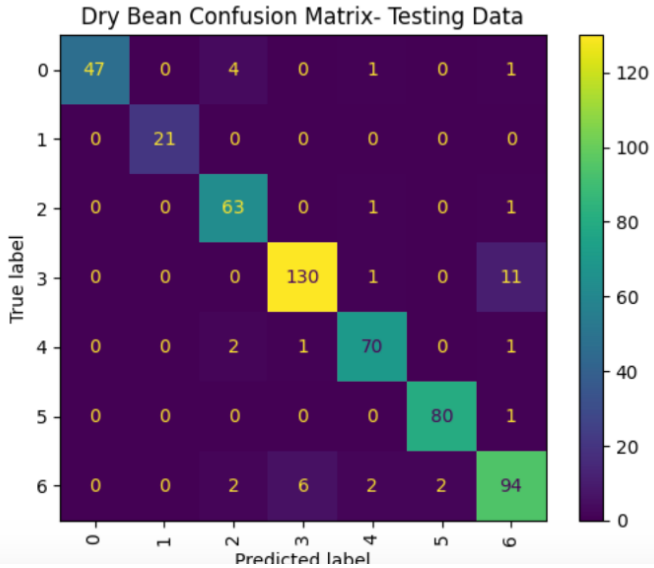
Boosted Tree is also an ensemble method where sequence of trees is fitted, so that each tree concentrates on misclassified records from the previous tree.

```
model_xgb= xgb.XGBClassifier(random_state=42,verbosity=0, min_child_weight=2,
                             max_depth=4, learning_rate=0.15, gamma=0.22, colsample_bytree=0.5)

model_xgb.fit(X_train, y_train)

print("Train F1_Score: ", metrics.f1_score(y_train, model_xgb.predict(X_train), average='micro'))
print("Val F1_Score: ", metrics.f1_score(y_val, model_xgb.predict(X_val), average='micro'))
```

Confusion Matrix Boosted Tree

```
Train F1_Score:  0.9614935822637106
Val F1_Score:  0.9317343173431735
```



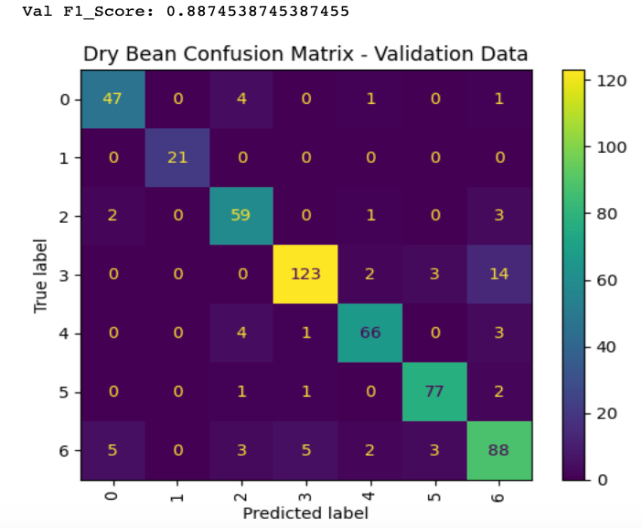


The accuracy for the training partition is rather high (1.0). This represents a very good fit of the classification tree. The accuracy values for both training and validation data sets are close to each other, and thus the overfitting is not a problem for the trained classification tree.

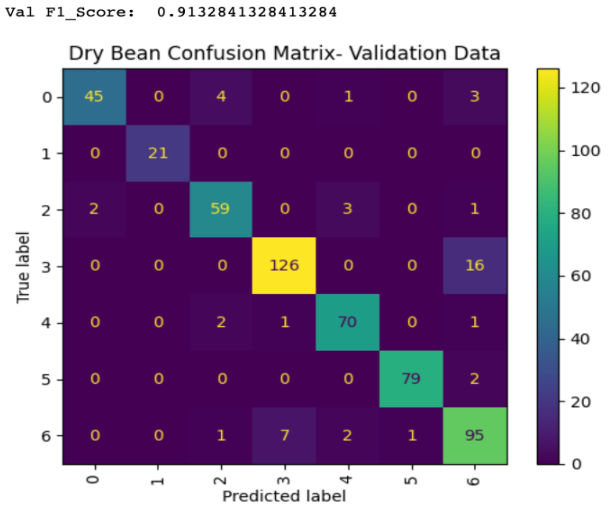
Accuracy Measure Comparison

Since there are four Confusion Matrices developed for each model, we can compare the validation partition for each using the accuracy value (misclassification rate).

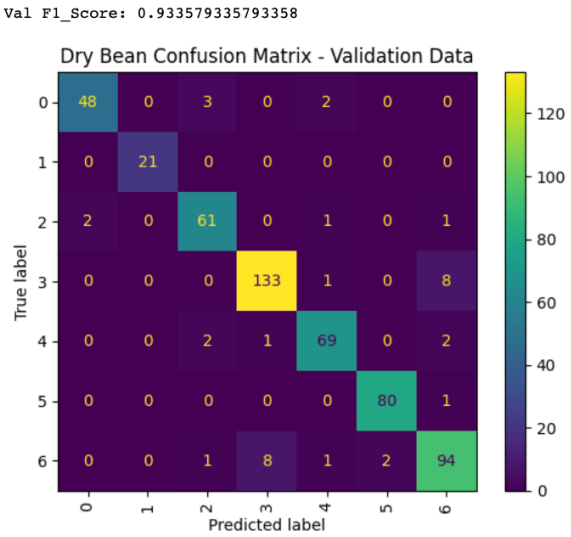
→Validation Partition for Decision Tree classifier with 12 as max_depth



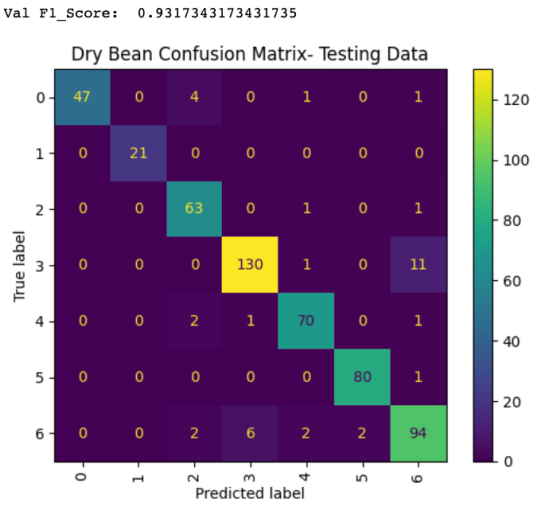
→ Validation Partition for Grid Search



→ Validation Partition for Random Forests



→ Validation Partition for Boosted Tree



On Comparing the four validation confusion matrices, it leads to the conclusion that the highest accuracy in classification may be achieved with the boosted classification tree (accuracy is 93.17% and misclassification is 6.83%). Therefore, the boosted classification can be recommended for making classification prediction for the Dry bean class.

Neural Networks

Neural networks are models used for classification and prediction. This model supports capturing complex relationships between predictors and outcome variable. Neural Networks is an iterative process that supports backwards propagation of error to distribute error all over the nodes to update weights (Coefficients) and theta values (Intercepts). Neural Networks is a feed forward method with input layer (predictors), hidden layer and output layer (outcome variable).

MLPClassifier()

For developing Neural Networks model, we are using *MLPClassifier()* algorithm from *scikit-learn* neural network package of Python. Since, 'Class' variable is a nominal categorical variable, we are using the above function with parameter activation = *'logistic'*. The hidden layer for this model is set to 12.

```
: model_mlp= MLPClassifier(random_state=1, max_iter=500, alpha=0.005)
  model_mlp.fit(X_train, y_train)
  print("Train F1_Score: ", metrics.f1_score(y_train, model_mlp.predict(X_train), average='micro'))
```

The values from all the predictors are normalized using *StandardScaler()* function. The scaled values for the training partition are the Z-score values. Z-score is a number of standard deviations of a column value from the column mean. For a numeric value xj in a column j in the training partition, the standard score or scaled value Zj is calculated as:

$$Zj = (xj - Uj)/Sj$$

where: *Uj* = mean of values in column j of training partition,

Sj = standard deviation of values in column j of training partition

The utilization of the scaled training predictors may lead to better prediction results.

Following are the Intercepts (Bias values) and Coefficients (Network Weights) used in the input, hidden and output layer of neural network model.

```
Final Intercepts for Dry Beans Neural Network Model
[array([-4.79598706, 27.39847153,  1.27073573,  1.13308799,  8.95352137,
       -0.55029283, -0.16815019, 27.17133122,  5.61343474, -0.83445201,
        0.89136045, -1.51868691]), array([-14.39710023,  12.13522694,   4.06840907,  -1.68199515,
        25.00185735,   7.4553393 , -32.27495493])]
```

The first array of the Final Intercepts contains node bias values (theta) for 12 nodes in the hidden layer. The second array of the Final Intercepts contains the (theta) value for the 7 nodes (classes) in the output layer.

Following are the Coefficients (Network Weights) used in the input, hidden and output layer of neural network model.

```
Network Weights for Dry Beans Neural Network Model
[array([[ 3.37309210e+00,  6.92002702e+00, -4.12051125e+00,
         -5.71104114e+00, -3.02332319e+00, -3.67606073e+00,
         -3.88386728e+00, -1.22090212e+01,  1.00039762e+01,
         -1.51264358e-01,  9.95858818e-01,  3.10787782e+00],
        [-5.82394721e+00,  5.01784387e+00, -2.64565082e+00,
         -3.14540335e-01, -1.95500890e+01,  3.58385167e+00,
          4.31595795e-01, -1.60369644e+01, -1.68618428e+01,
          4.51395236e+00,  5.93792829e+00, -3.11348460e+00],
        [-3.46596682e+00,  1.00418889e+01, -2.89727492e+00,
          9.57597104e+00, -1.15999195e+01, -1.17056493e+01,
          9.26796960e-01, -8.88233751e+00,  3.93997963e+00,
          8.85623117e-01,  1.43070646e+00,  1.17412423e+01],
        [ 3.09383926e+00,  1.45585719e+01,  5.52040491e+00,
          9.67026706e-01,  5.60463230e+00,  5.23943288e+00,
         -3.20270171e+00, -2.12080708e+01, -7.17675358e+00,
         -3.57665722e+00,  1.14398908e+00,  2.88422030e-01],
        [-4.49900457e+00, -1.99288628e+00, -3.45349917e-01,
         -1.67546813e-01, -6.23935626e+00,  9.59550212e+00,
         -8.79255136e+00, -1.62158905e+01,  8.55036697e+00,
         -4.64709675e+00,  7.32615429e+00,  4.81083478e-01],
        [-1.52477940e+00, -1.24689982e+00, -2.17957413e+00,
          3.76419315e+00, -7.36935039e+00, -1.42403887e-01,
         -1.22851020e+01,  1.85083067e+01, -4.91073558e+00,
         -7.45828574e+00,  1.08900310e+01,  1.87257619e+00],
```

```
[ 2.29068492e+00,  6.09115051e+00, -1.36023657e+00,
 -1.13962071e+01, -5.52525687e+00, -9.19782000e+00,
 -3.16823449e+00, -1.22532358e+01,  4.06852971e+00,
 -1.07654838e+00, -7.79182797e+00, -1.96657659e+01],
[-7.86325700e-01,  1.31800303e+01,  7.93924280e+00,
  1.26233865e+01, -8.93308555e-01, -5.02777096e-01,
 -1.40084100e+00, -1.28969483e+01, -1.77930664e+00,
  1.74053387e+00, -5.65679530e-02,  8.27049820e+00],
[ 2.87879498e-02,  8.26355505e-02, -3.13692034e-01,
  1.58321739e-01,  1.67345942e+00,  3.39117877e-01,
 -1.87984223e-02, -3.50274310e+00, -1.97345117e+00,
 -4.05816473e-02, -5.49803767e-01, -8.60802368e-02],
[ 4.40975393e+00,  5.10353463e+00, -4.58474874e-02,
  1.14767905e-01, -3.97155753e+00, -6.40688959e-01,
  1.86665017e+00, -7.56874586e+00, -5.29176275e-01,
 -2.36920760e+00, -2.26073638e-01, -2.16066880e-01],
[-4.20315763e+00, -1.42989461e+01, -1.01039980e+00,
 -6.36088709e-01,  4.70516960e-01, -5.27692180e+00,
 -2.32424664e+00,  1.22192733e+01,  9.73054550e+00,
  6.17767947e+00,  2.20317147e-01, -9.41751396e-01],
[-6.02316834e+00,  4.96169563e-01, -3.14688951e+00,
  3.49181021e+00, -1.06771129e+00,  1.61437730e+00,
 -6.01437042e+00, -2.62019847e+00,  2.74158258e+00,
 -1.73539818e+00,  7.83913487e+00,  1.46325363e+01],
[-4.16167995e+00, -2.81046081e+01,  4.18542607e-01,
  5.45950191e+00, -2.09076549e+01, -5.92936001e+00,
 -2.48409579e-01,  3.75731662e-01, -5.58638549e+00,
 -1.93914946e-01,  8.11575110e-01,  2.89118091e+00],
[ 7.30337663e+00, -1.20545037e+01,  6.70212341e+00,
  2.05794707e+00, -2.70101444e+01, -3.97362894e+00,
 -3.93933614e+00,  1.01446841e+01, -1.10275249e+01,
 -6.60079307e+00,  2.19782909e+00,  9.28417243e-01],
[-5.88511236e+00, -2.35582345e-01, -3.91695150e+00,
  2.07400791e+00, -2.44055069e+00,  7.71142696e+00,
 -6.66840737e+00, -8.94272136e+00,  1.85697214e+00,
 -6.66268110e+00,  9.69528878e+00, -8.83500302e+00],
```

```
[ 1.77008757e+00,  1.62897062e+00,  7.05227445e-01,
 -7.79795309e-01,  1.03110370e+00,  5.55088197e-01,
 -8.03659926e-02,  1.17506379e+01, -2.41391565e-02,
  5.54747309e-01, -7.80164404e-01, -4.35499756e-01]]), array([[ -2.13836109,  -0.66875438,   9.89715239,   0.43597649,
 -22.99763082,  11.08421188,   4.86505114],
[  8.6653354 ,  13.91811409,  17.67496447, -29.4790147 ,
  -6.11047176,  -1.90283844,  -3.15472071],
[ 19.06965229,  -2.26023018, -42.99789001,  18.1029343 ,
  -4.75703462,   4.6645464 ,   7.64158383],
[ 16.67445513, -30.36000875, -14.10419409,   8.89026841,
   6.05098397, -11.54211659,  25.41914234],
[  0.36604902,  -7.03571811,   6.92378   ,  -2.49853796,
  -6.55497652,   3.08220229,   5.54118412],
[  6.9006582 , -12.28801789,   7.88561585,  -4.53558199,
  -0.07079152,   0.04916548,   2.47930004],
[ -3.62323714,  -4.72186037,  -1.04746625,   2.72626012,
  -5.74559372,   1.48519662,  10.88047514],
[ -1.77504032,  -1.96148434,  -2.50052059,   1.77844969,
  -3.57990438,   5.18844027,   3.45982891],
[ -7.57917992,  -4.42180484,   8.52493566,   2.05034907,
   1.32976606,  -3.30855009,   4.28538337],
[  3.52745881,  14.39321484, -10.69934745,  -0.5848756 ,
 -12.6240937 ,   1.2048914 ,   4.4816463 ],
[ -3.48130901,  -8.66418879,  28.18704852, -12.45082273,
   2.23090594,  -5.99506115,   0.66858   ],
[-10.10785972,  -3.47330641, -36.05301901,  22.48331862,
   7.27448116,  17.12106951,   3.23560717]])]
```

The first array of the Network Weights contains weights from each of the 16 nodes in the input layer (16 predictors') to the 12 nodes in the hidden layer. The second array of the Network Weights contains weights from each of the 12 nodes in the hidden layer to the 7 nodes in the output layer.

As shown in the below formula, new bias values(theta) and network weight values are updated using the backwards propagation of error.

$$\theta_j^{new} = \theta_j^{old} + l(err_j)$$

$$w_j^{new} = w_j^{old} + l(err_j)$$

where l = constant between 0 and 1, reflects the *learning rate*.

Classification Probabilities

The updating of bias values and network weight stops when backwards propagation of error leads to very little change of weights from one iteration to the next or misclassification rate reaches required threshold.

Following are the first 10 records with classification results for Validation data set.

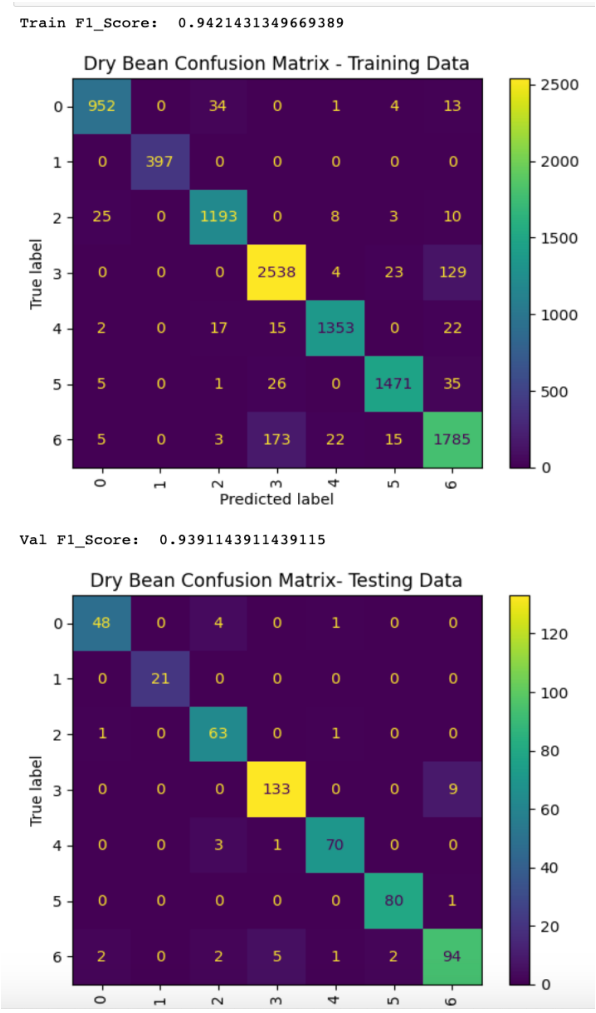
Classification for Dry Beans Data for Validation Set

	Actual	p(0)	p(1)	p(2)	p(3)	p(4)	p(5)	p(6)
10057	SIRA	0.0176	0.0	0.0126	0.0000	0.0125	0.0061	0.9512
13354	DERMASON	0.0000	0.0	0.0000	0.7951	0.0009	0.0007	0.2032
13141	DERMASON	0.0000	0.0	0.0000	0.8450	0.0011	0.0002	0.1538
8634	SIRA	0.0087	0.0	0.0000	0.0000	0.0358	0.0093	0.9461
12579	DERMASON	0.0000	0.0	0.0000	0.8446	0.0009	0.0081	0.1463
11088	DERMASON	0.0000	0.0	0.0000	0.9996	0.0004	0.0000	0.0000
13111	DERMASON	0.0000	0.0	0.0000	0.9210	0.0010	0.0002	0.0778
9369	SIRA	0.0000	0.0	0.0039	0.0000	0.7933	0.0002	0.2026
9826	SIRA	0.0002	0.0	0.0165	0.0000	0.0006	0.0005	0.9822
13193	DERMASON	0.0000	0.0	0.0000	0.7808	0.0007	0.0001	0.2184

	Classification
10057	SIRA
13354	DERMASON
13141	DERMASON
8634	SIRA
12579	DERMASON
11088	DERMASON
13111	DERMASON
9369	HOROZ
9826	SIRA
13193	DERMASON

Accuracy Performance Measures

Following are the confusion matrices for the training and validation partitions of the neural network model for dry beans.

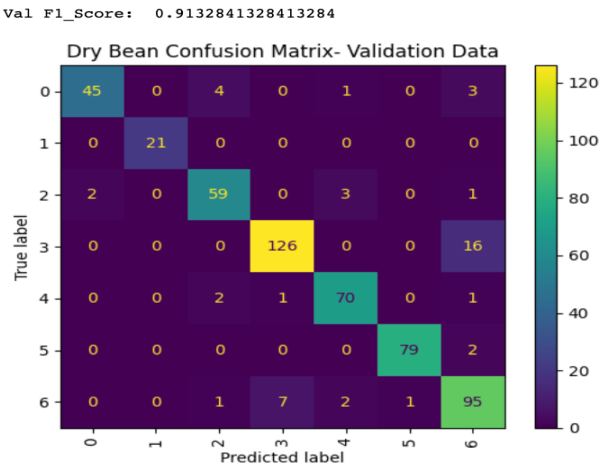


The confusion matrices for the training and validation partitions show a very high accuracy of around 94.21% for training and 93.91% for validation partition, and thus the trained neural network model fits well for the validation data set and can be used for classification of the dry beans. The misclassification rate for the training partition is 1 - 0.9421 = 0.0579 or 5.79%, and for the validation partition 1 - 0.9391 = 0.0709 or 7.09%. The accuracy of the model for the validation records is closer to that of the accuracy for the training records, and therefore, there is no overfitting in this model.

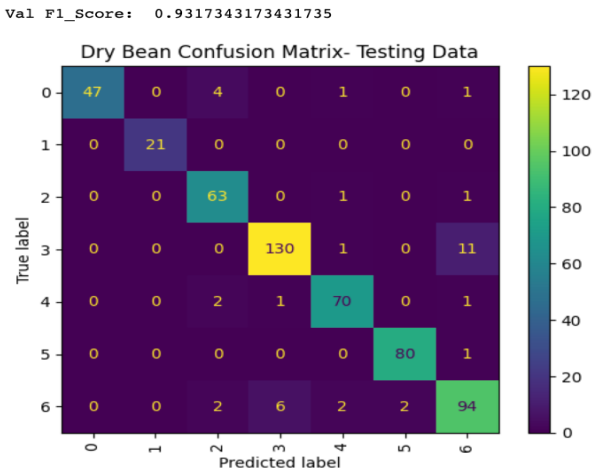
Conclusion

After developing the classification models and reviewing the results of each technique, it is necessary to interpret and compare the accuracy performance measures of all the techniques/models designed in the project.

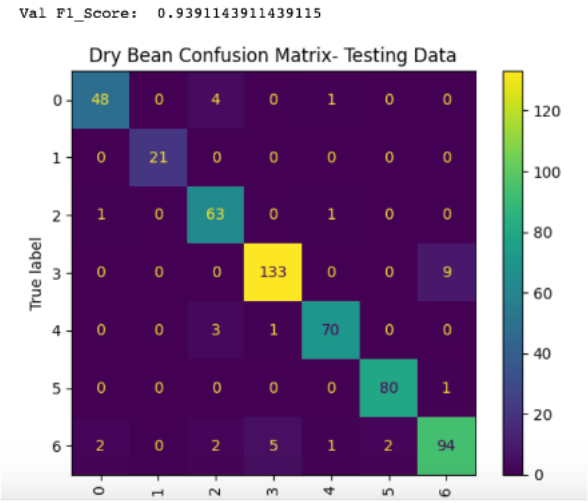
→Final Validation Partition of Nominal Logistic Regression



→Final Validation Partition of Decision Tree using Boosted Tree algorithm



→Final Validation Partition of Improved Neural Network Model with hidden layer size as 10



On Comparing the three validation confusion matrices, it leads to the conclusion that the highest accuracy in classification may be achieved with Improved Neural Network Model with hidden layer size as 10 (accuracy is 93.91% and misclassification is 7.09%).

Recommendation

Hence, we would recommend, using the Neural network model for classifying the Type of the Dry Beans for the outcome variable ‘Class’.

Remarks

There were few observations during the development of Classification Models.

- The initial Guess parameter for the Decision Tree was predicted as max_depth equal to 10. When tried to optimize the model using GridSearch() algorithm using the results of the initial parameters, the max_depth appeared as 15.
- To test the statistical importance of all the predictors of dry beans data set, we used Generalize Linear Model(GLM) from statsmodels library. It seems that all the predictors are statistically important and therefore there is no dimension reduction.

Data mining analysis and results

On analysing the results from each model, following are some inferences:

- There was no overfitting observed in any of the three techniques used as the accuracy values for training and validation confusion matrices were close to each other.
- All accuracies achieved for the developed classification model were greater than 85%.
- It is worthwhile to note that the Accuracy value for the validation partition of Boosted Tree model was 92.16% and only marginally lower than the Accuracy value of Neural Network model. Thus, Boosted Tree model may be recommended for classification as an alternative to the Neural Network model

Possible Benefits

Following are some of the benefits:

- Approaches to estimating weights in neural networks involve using the errors iteratively to update the estimated weights. Errors being distributed across the hidden nodes helps in computing an accurate outcome.
- All the graphs and charts developed in the models helped in better visualization of the classification results.
- The next best alternative is Boosted tree model and is efficient for large data sets, like the one used in the project.

Limitations

Following are few of the limitations:

- Models developed using GridSearch() took more time to predict the results.
- Max_Iteration had to be increased to a value very higher than the default value for the Nominal Logistic Regression model.

Bibliography

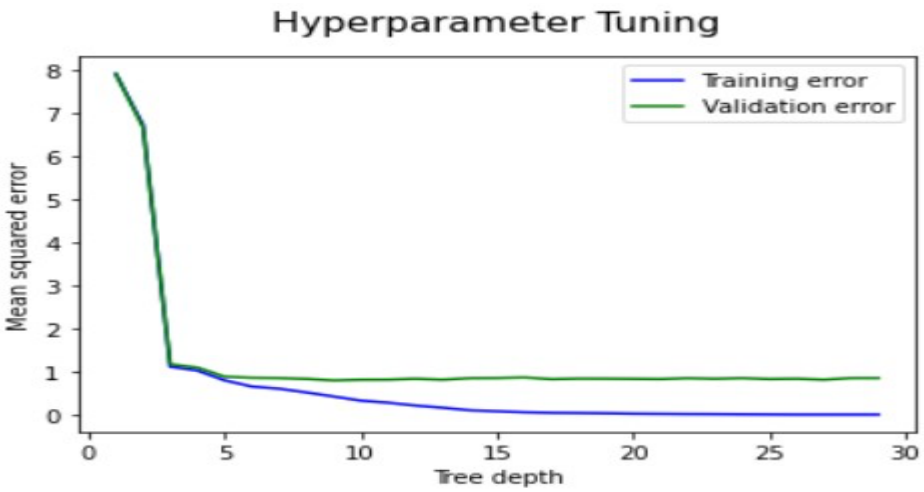
- The data set is obtained from the below UCI Machine Learning Repository URL https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset
- Data Mining Concepts https://docs.oracle.com/cd/B19306_01/datamine.102/b14339/3predictive.htm#BABFABG
- Decision Tree in Sklearn https://kanoki.org/2020/05/13/decision-tree-in-sklearn/
- Understanding Decision Trees for Classification in Python https://www.kdnuggets.com/2019/08/understanding-decision-trees-classificationpython.html
- KOKLU, M. and OZKAN, I.A., (2020), â€œMulticlass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques.â€• Computers and Electronics in Agriculture, 174, 105507.DOI: https://www.sciencedirect.com/science/article/abs/pii/S0168169919311573?via%3Dihub
- Logistic Regression Analysis https://www.sciencedirect.com/topics/nursing-and-health-professions/logistic-regressionanalysis
- API Reference https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model
- How to tune a Decision Tree? https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680

Appendices

Appendix A. Parameter tuning for Decision Tree.

Link : https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3

The above link helped in understanding the importance of parameters and how to use graphs to tune parameters. The usage of max_depth and min_samples_split was described well and that assisted in predicting a max_depth for this case. The graph was developed for Decision Tree in this project.



Appendix B. Detecting Outliers

Title: Statistics for Business and Economics Authors: Anderson, Sweeney, Williams, Camm, and Cochran Publisher: Cengage Learning ISBN: 10: 1337901067 ISBN 13: 978-1337901062 (Page:106)

Standardized values (z-scores) can be used to identify outliers. Recall that the empirical rule allows us to conclude that for data with a bell-shaped distribution, almost all the data values will be within three standard deviations of the mean. Hence, in using z-scores to identify outliers, we recommend treating any data value with a z-score less than -3 or greater than +3 as an outlier.