COSC 757 Assignment 2: Dry Bean Dataset Classification

Doug Branton
Towson University
COSC 757: Data Mining
Source Code: https://github.com/dougbrn/Towson/tree/main/COSC757/assignment2
dougrbranton@gmail.com

***Abstract*—In this assignment, I explore the Dry Bean Dataset and define a standardized experiment to test three different classification algorithms in predicting the bean type from a set of features describing the size and shape of the beans. The beans are split into a Train and Test set using KFolds cross-validation and each partition is statistically validated to be representative of the full dataset. The performance of the three algorithms is compared by examining their accuracy and full confusion matrices. Finally, I discuss some takeaways and general findings from the behavior of the classification algorithms.**

I. Introduction

For this assignment, I chose the Dry Bean Dataset from the UCI Machine Learning Repository. The Dry Bean Dataset consists of 13,611 grains of 7 different labeled dry beans, which were imaged with a high-resolution camera. A set of 16 features were extracted from the images, consisting of 12 dimensions and 4 shape forms. The true label for each bean feature set is known, and provided along with the features. This makes the Dry Bean Dataset well suited for classification.

The objectives of this analysis are as follows:

1) Perform Exploratory Data Analysis (EDA) on the Dry Bean Dataset to better understand the features and indicate what preprocessing will need to occur.
2) Research and choose 3 classification algorithms for supervised learning on the Dry Bean Dataset.
3) Design an experiment using a train and test set to learn the Dry Bean Dataset Label given the full feature set.
4) Compare the results of the 3 classification algorithms using a standardized set of model evaluation metrics.

II. Methodology

A. Exploratory Data Analysis and Preprocessing

In the first step of exploring the Dry Bean Dataset, I produced Figure 1, which provides a histogram for each column in the dataset including both the features and the class label. In expecting the distribution for the Class label, it's clear that the distribution is weighted towards some classes rather than others, notably Dermason has almost 7 times the number of datapoints relative to Bombay. However, even in the case of Bombay we still have roughly 500 datapoints to work with and don't need to necessarily handle it as a special rare class. It is worth noting this for now and potentially iterating if the classifiers have difficulty with the Bombay class.

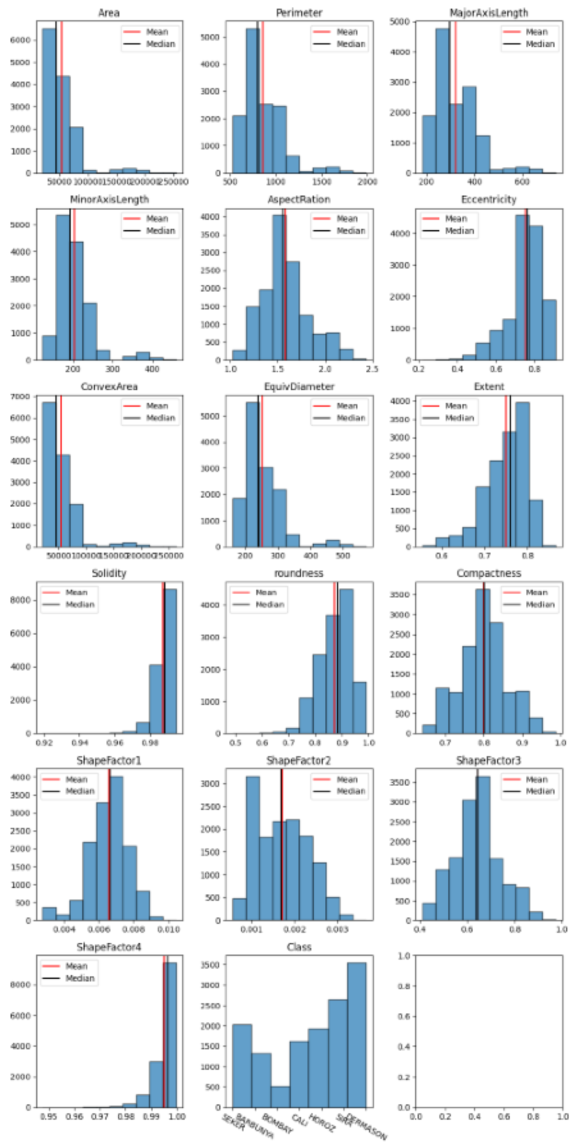

Fig. 1. Histograms for each raw feature

For the rest of the features, we see roughly Gaussian behavior with some high variance in the order of magnitude. This indicates that normalization will be important to perform on several of the features in the dataset. There also appears to be a small second Gaussian distribution that appears in several metrics, and due to the proportion of each class relative to the full dataset, it's reasonable to expect that these secondary Gaussians are coming from a bean class that is highly distinct from the other classes.

Beyond the secondary peaks, this dataset appears fairly well behaved and I felt comfortable not rejecting any datapoints as outliers. At this point, I normalized the following features using z-score scaling; Area, Perimeter, MajorAxisLength, MinorAxisLength, ConvexArea, and EquivDiameter. This ensured that all features were now on the same order of magnitude, mitigating the risk of several larger features being more heavily weighted in any of the classification algorithms. From this normalized dataset, I produced Figure 2, which contains scatter plots of the 16 features plotted against one another, color coded by bean type.

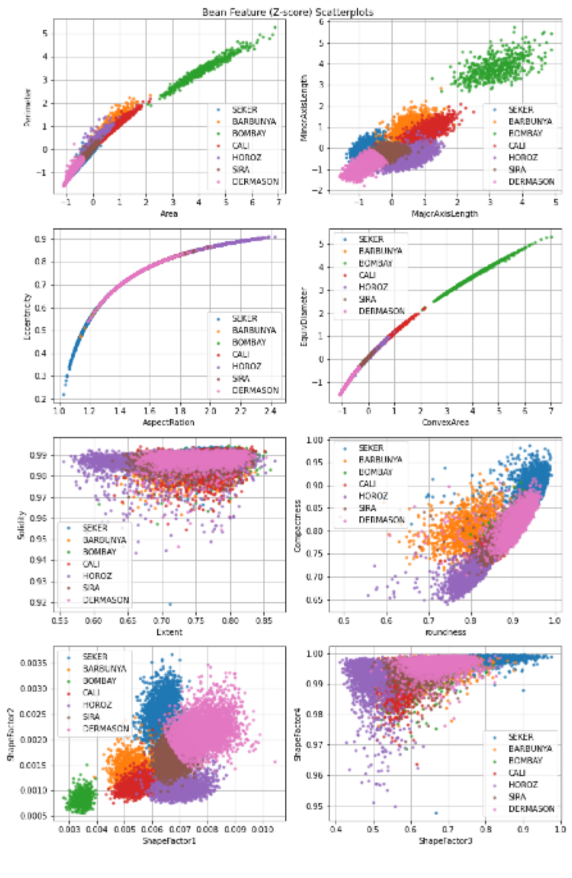

Fig. 2. Scatter Plots of Features, several features normalized by z-score

As can be seen in the plots, the Bean classes do have distinct clustering within most of the feature space. Bombay beans appear distinctly separate and are likely the source of the secondary Gaussians in the histograms. Within these plots, we do see several features that have a high correlation with one another. For example, Area and Perimeter have a clear relationship, as the two quantities are highly related mathematically. This potentially warrants rejecting several features in order to not have these properties be assigned additional weight compared to unique features. It's worth noting that just from this plot the full scope of correlation between the features isn't know, as this is just 8 of 120 total pairs. However, because the context of this data is all spatial attributes (shape and size), it seems difficult to isolate unique features as most appear to have some correlation. For this, I chose to let the full feature set be included in analysis, with the understanding that performing dimensionality reduction may be a more effective method of finding several higher-impact features.

B. *Classification Algorithms*

The following classification algorithms were implementations chosen from Python's Scikit-Learn package.

1) Decision Tree Classifier: Decision Trees create a set of simple to understand decision rules based on thresholds on the features in the dataset. They are capable of handling multi-output data and scale reasonably well with data volume, scaling logarithmically with the size of the training set. The maximum depth is an important tunable hyper parameter that determines the complexity of the overall tree. Decision Trees are vulnerable to overfitting, where the classifier learns the training data itself rather than the general patterns contained in the training data, when the maximum depth is set too large. I will discuss in the results section some of my findings regarding overfitting with applying Decision Trees to the Dry Bean Dataset.

2) Naive Bayes Classifier: Naive Bayes Classifiers apply Bayes' Theorem and assume that there is strong independence between the features in the dataset (Naive). As discussed in preprocessing, the features in this dataset don't adhere to this assumption, and so for this test I was curious to see how much that will impact the performance of the classifier. Bayes classifiers aim to find the posterior probability, which is the probability that a given hypothesis holds provided the sample dataset. The specific implementation I chose was the Gaussian Naive Bayes algorithm, which additionally assumes that the likelihood of features is Gaussian.

3) Multi-Layer Perceptron (Neural Network): A Multi-Layer Perceptron (MLP) is a Neural Network that learns a function that maps an input layer of m features, to an output layer of n output dimensions. Between these layers, any number of hidden layers with a set of p nodes can exist and define the general network architecture. Scikit-Learns MLP implementation trains the classifier by back-propagation, where each node is assigned a weight, which is tuned to predict the correct class label of the input layer.

C. *Experimental Design*

The classification goal of this project is to predict the "Class" label, given the full set of 16 features. To ensure a standardized environment for the 3 classification algorithms, all algorithms are trained against an identical train set, and validated against an identical test set. The train and test datasets are partitioned using the KFolds cross-validation algorithm, where the data is split K times (K=5 in this case, which is a recommended standard) and one "fold" acts as the test set while the other k-1 folds act as the training set. This process is repeated K times so that every data point serves in the test set once. Choosing K=5 yields an approximate 80/20 split between the train and test sets. Additionally, I needed to shuffle my dataset as it was ordered, meaning the train and test partitions without shuffle would be heavily weighted towards certain bean classes depending on which partition was chosen.

In order to validate the train and test sets, I performed a Two-Tailed Hypothesis Test on each fold against the full dataset. In these tests, the null hypothesis was chosen to be:

$$H_0 : \mu_{fold} = \mu_{dataset}$$

For each fold, the test statistic was calculated as per the following equation:

$$t = \frac{(\bar{x} - \mu_0)}{s/\sqrt{n}}$$

The test statistic was then used to retrieve a p-value and the full table of results is shown below:

Fold	t	p-value	Valid Partition ($p > 0.05$)
1	0.2496	0.803	Yes
2	-0.0695	0.945	Yes
3	0.2683	0.788	Yes
4	-0.0231	0.982	Yes
5	-0.4302	0.667	Yes

All 5 folds have p-values greater than 0.05, thus the null hypothesis cannot be rejected and these partitions are all valid representatives of the full dataset.

D. Accuracy Metrics

To evaluate these algorithms, the balanced accuracy is computed for the result of predicted classes vs the true class labels. The balanced accuracy is defined as the average of the recall on each class, and is particularly useful for classes with higher than two possible labels. The balanced accuracy is also computed for the training set, as the comparison of this metric in the train and test sets will highlight any overfitting that may be occurring. In addition, the full confusion matrix is provided to show visually how the classifier fared across the full class-space.

III. Results

A. Decision Tree Classifier

The table below shows the final model balanced accuracy on the train and test partitions. Figure 3 shows the complete Confusion Matrix for the Decision Tree Classifier.

Accuracy (Test)	Accuracy (Train)
0.9097	0.9151

Agreement within 1% between the test and train set gives confidence that we are not overfitting the train set. In my first attempt to train the Decision Tree, the hyper-parameter constraining the maximum depth of the tree was left unset. While the accuracy, remained high for the test set, the training set had a perfect accuracy of 1.0, indicating that the model was overfitting. In my testing, I settled on a maximum depth of 6 which post-pruned the tree and achieved a reasonable balance of performance and agreement between train and test. The confusion matrix yields some general understanding of the model performance. Bombay beans were classified perfectly with no confusion with any other bean type. This makes intuitive sense because Bombay beans appeared the most distinct. There are some more significant sources of confusion between Barbunya/Cali and Dermason/Sira. These pairs are limiting the overall accuracy, but from EDA we see that they also are partially entangled in the feature space.

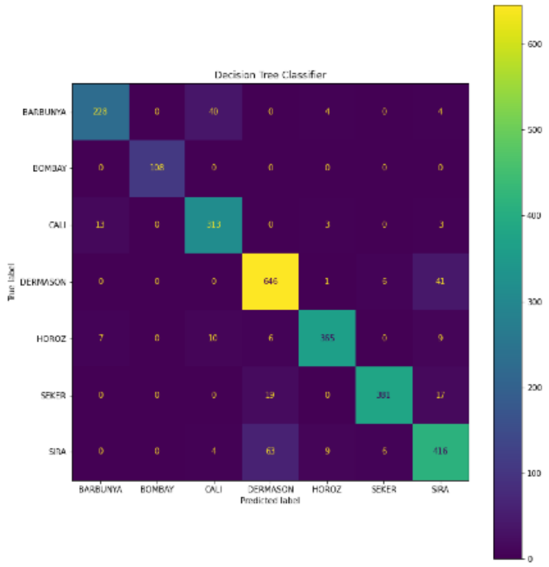

Fig. 3. Confusion Matrix for the Decision Tree Classifier

B. Naive Bayes Classifier

The table below shows the final model balanced accuracy on the train and test partitions. Figure 4 shows the complete Confusion Matrix for the Naive Bayes Classifier.

Accuracy (Test)	Accuracy (Train)
0.9124	0.9060

In both the Confusion Matrix and the Accuracy Scores, the Naive Bayes Classifier performs similarly to the Decision Tree Classifier. Again, we see no evidence of overfitting, and the Confusion Matrix yields similar structure in terms of pairs that are contributing to the loss in the classifier. As noted, our dataset does not fit the naive assumption that the Naive Bayes Classifier asserts, but that does not appear to be compromising the overall quality of the classifier in this case.

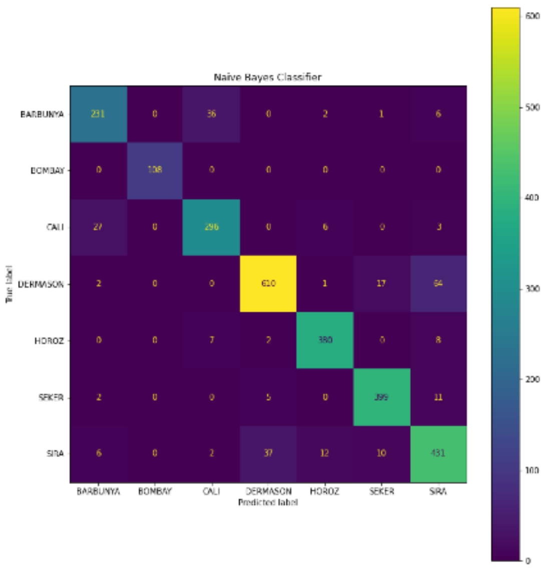

Fig. 4. Confusion Matrix for the Naive Bayes Classifier

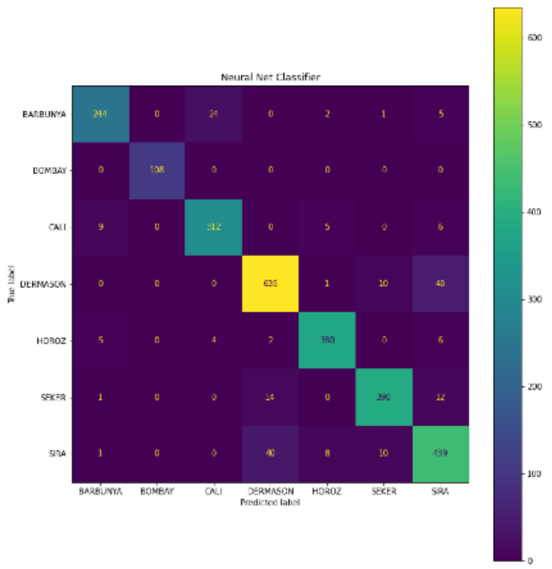

Fig. 5. Confusion Matrix for the Multi-Layer Perceptron

C. Multi-Layer Perceptron (Neural Network)

The table below shows the final model balanced accuracy on the train and test partitions. Figure 5 shows the complete Confusion Matrix for the MLP Classifier.

Accuracy (Test)	Accuracy (Train)
0.9304	0.9237

As with the other classifiers, the MLP Classifier achieves very similar, but overall better results in the accuracy and Confusion Matrix structure. However, in training I found that the MLP was very sensitive to the defined network architecture (The number and size of hidden layers), whereas the other classifiers more or less worked "out of the box". I found an optimal architecture in defining five hidden layers with four nodes each, but incremental deviations from this could drastically affect the performance. For example, defining a three layer/four node architecture yielded a balanced accuracy of approximately 0.63. That being said, the ability to affect performance with this tuning did result in a classifier that performed the best in this experiment.

IV. Conclusion

Overall, the three chosen classifiers all performed well in classifying the Dry Bean Dataset. The dataset itself is well-suited for classification tasks, and required only very limited preprocessing. The concern I found in EDA of potentially issues stemming from class imbalance appeared to not negatively impact the classifier performance, though it is certainly the case that the overall accuracy is being skewed by the individual classification accuracy's of more common bean types. Each of the classifiers settled on a similar performance level, highlighting the same sources of confusion. This, combined with what makes intuitive sense when examining the distribution of beans in the features themselves, yields strong confidence that the classifiers have learned the patterns behind the features.

There were several algorithm-specific takeaways from this experiment. The first from the Decision Tree Classifier was that constraining the maximum depth is important, as the potential for overfitting is very real. In fact, I didn't catch overfitting initially, and only in returing to the algorithm and calculating the accuracy on the train set did I find that it was occurring. For the Naive Bayes Classifier, I was expecting it to perform worse than the other classifiers due to the dataset not being tailored to meet the Naive condition. However, the performance was in line with the other classifiers, possibly because the dataset itself is very well-behaved for classification and this inefficiency may only appear in harder classification problems. Finally, for the MLP Classifier, it was clear that experimentally determining the network architecture was crucial to the overall success of the classifier. Because the individual nodes and hidden layers don't hold intuitive meaning, I found myself resorting to trial and error to accomplish this.