Predicting Bean Types from Defining Features Using Various Machine Learning Techniques

## DATA SCIENCE 4 COURSE: MACHINE LEARNING

## University of Toronto School of Continuing Education

## University of Waterloo School of Professional Development

## December 13th, 2021

Group 7:

Amr Abdalla

Mitchell Stevens

Wen Tang

Matt Walsh

Treshan Waas

Ting Zhang

\*the authors contributed equally to this project and paper

# 1.0 Introduction

Originally from America, dry beans have been widely spread in the world with broad genetic diversity, by adapting to temperatures, humidity, climate, and soil quality of the region that they grow (Gentry, 1969). Thus, it is very important for us to understand and classify dry beans according to their phenotypes.

The dry bean dataset is a popular open-source dataset available for users to experiment with machine learning models (Koklu et al. 2020). The dataset consists of dry beans of seven types, each recorded with its physical characteristics (size across various dimensions, shape, etc.). Computer-aided images were used of 13,611 grains and 16 features were recorded classifying the type of the bean.

Building classification models of dry beans can have wide agricultural applications. Farmers can use this prediction technique to choose the proper types of dry beans that benefit their local agricultural economy. In addition, identification of beans based on their phenotypes can help farmers to expand the marketing according to a standard (Chen et al. 2010).

# 2.0 Objective and methods

The objective of our analysis was to build, train, test and evaluate the performance of multiple machine learning models to determine the most effective model for our dataset. We used a Dry Bean dataset obtained from the UC Irvine Machine Learning Repository which contains 7 different types of beans with 16 features. Our approach was to build models using three different classifiers (k-Nearest Neighbours, Random Forest, and multiclass Support Vector Machine) and then combine all three into a voting classifier model.

# 

# 3.0 Data Preparation

This dataset was chosen for our analysis because it is known to have clearly defined features for each bean type, resulting in accurate machine learning prediction. Because the dataset contains 7 unique bean types, we prepared the data for use in classification models rather than regression.

The first step in data preparation was to confirm that the data was successfully downloaded into a pandas dataframe by checking the head. Once that was confirmed, we displayed the correlation matrix to look for correlation between columns.

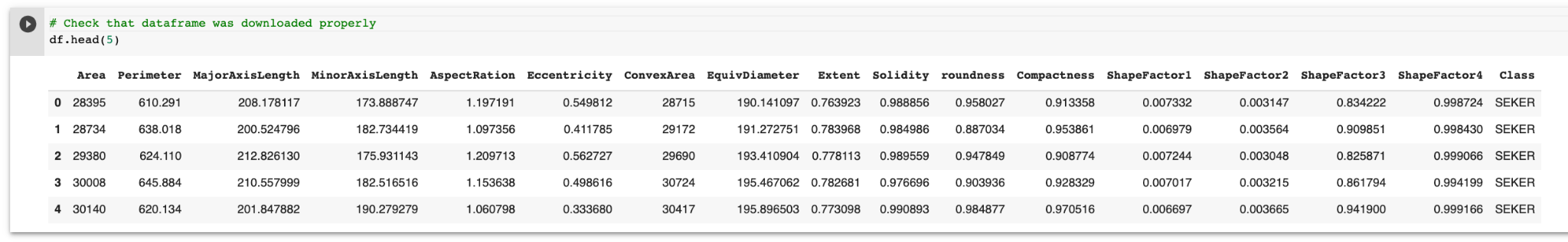


Table 1: First five rows of the imported data

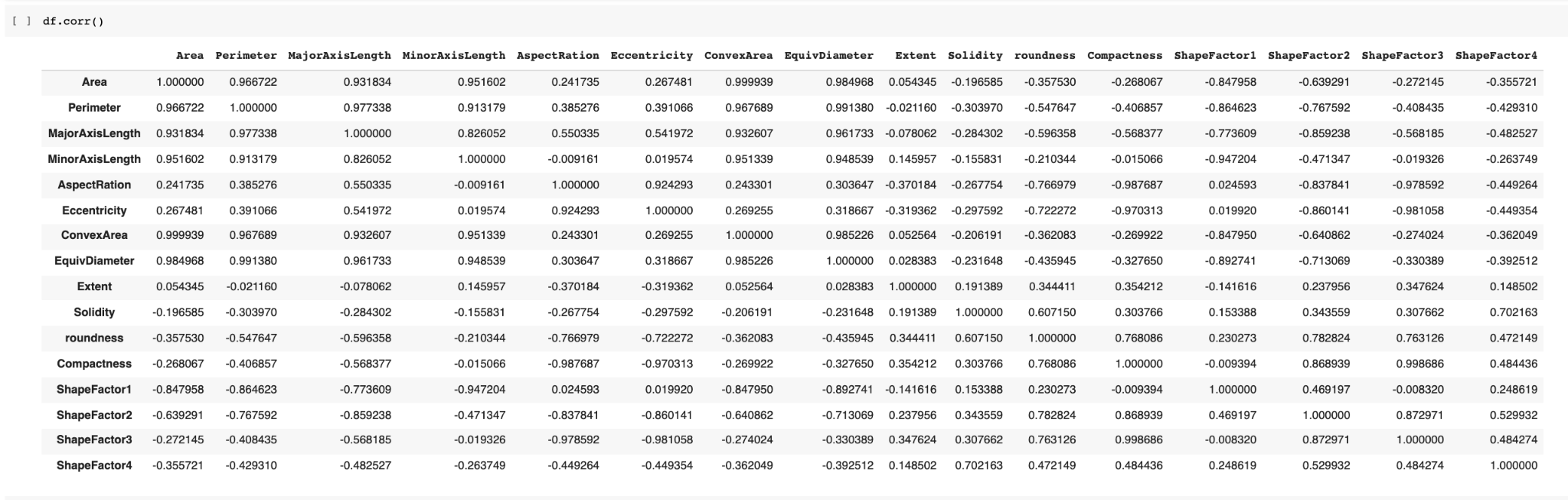


Table 2: Correlation matrix for the sixteen features of the data

Next, we grouped the dataset by bean type and plotted the distribution of quantity of each bean type. This showed that the dataset does not contain an even distribution of bean types, which could skew results. We also used the describe method to check common statistics on the original dataframe as well as the grouped dataframe. Finally we produced a pairplot coloured by bean type which displayed the most defining features of each bean.

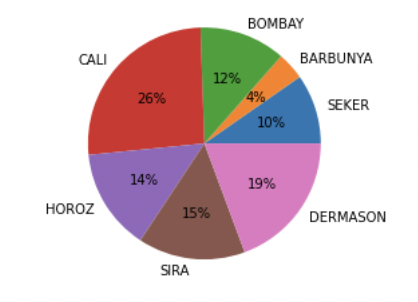


Figure 1: Pie chart of data distribution in Class

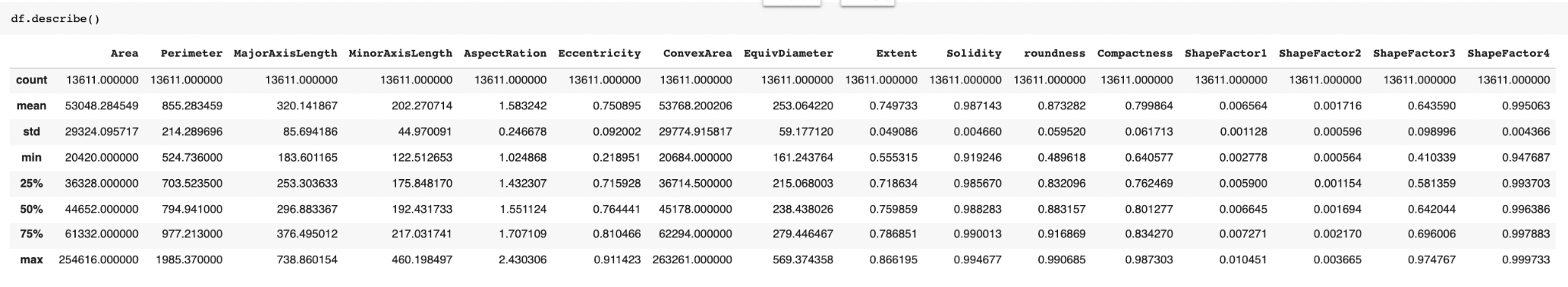


Table 3: Descriptive statistics for the features across the entire data set

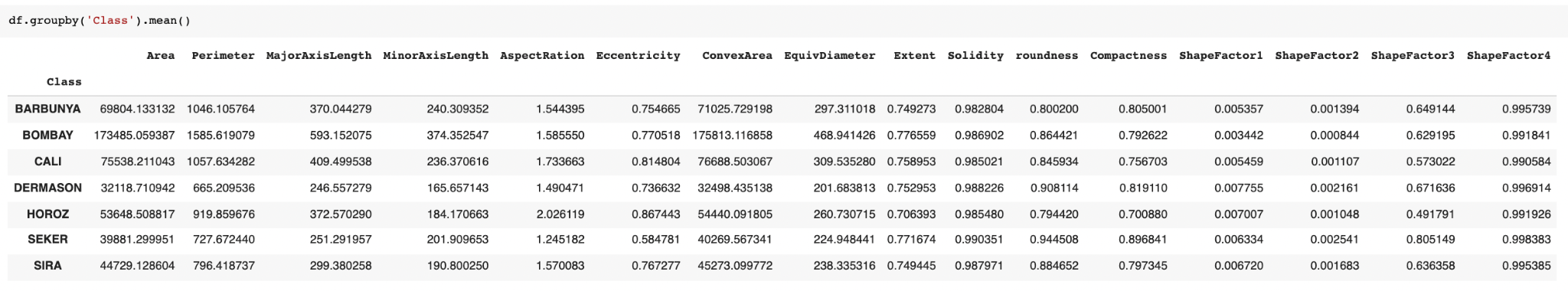


Table 4: Mean values for each feature by bean variety

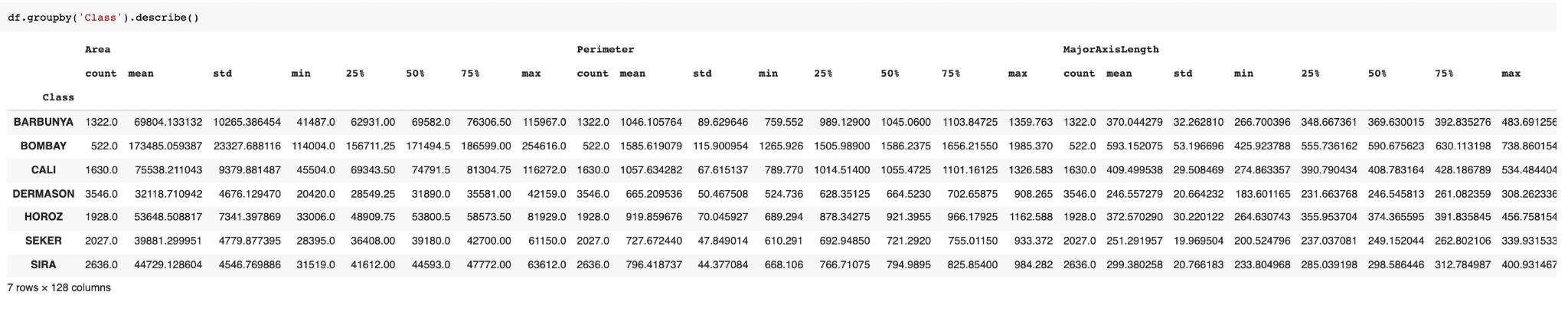


Table 5: Descriptive statistics for the features by bean variety

To begin preparing the dataframe for use with our classifiers, we replaced the bean type labels with integers. Next, we split the dataframe into feature columns and one target column, the bean types. Then we used sklearn’s train\_test\_split to split the dataframes into X and y testing and training sets. We made sure to use a random state variable to guarantee replicable results. Once we printed the shapes of our new arrays, they were ready for use in the models.

# 

# 4.0 Model Design

## 4.1 kNN Classifier

The KNN classifier defines classification based on a neighboring point. It is the most straightforward classification algorithm in supervised machine learning (Kumar 2020). Since our dataset of dry beans is a relatively small dataset with clean data, we decided to first use the kNN model to fit the dataset.

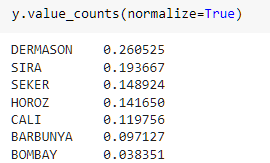
The target variable bean types is categorical. We initially used all the continuous variables as features in training input of X. X\_train and X\_test was transformed with scalers separately. We tried both the MinMax scaler and Standard Scaler; we discovered that MinMax scaler gave a model of a bit higher accuracy score.

We used a threefold cross validation on the training data. Randomized Search was done for hyperparameter tunings on {'n\_neighbors': stats.randint(1,50) , 'weights':['uniform', 'distance'], 'algorithm': ['auto', 'ball\_tree', 'kd\_tree', 'brute'], 'leaf\_size':stats.randint(2,50)}. According to RandomizedSearchCV results, these hyperparameters were picked up: {'algorithm': 'auto', 'leaf\_size': 19, 'n\_neighbors': 16, 'weights': 'distance'}

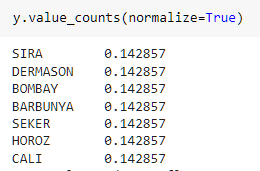
After initial fitting, a second approach was used with selected features based on pariplots. The selected 10 features only included: 'Area', 'Perimeter', 'MajorAxisLength','MinorAxisLength','ConvexArea', 'EquivDiameter', 'ShapeFactor1', 'ShapeFactor2', 'ShapeFactor3', 'ShapeFactor4'. For comparison, exactly the same hyperparameters were used with this second approach of reduced feature selections.

## 4.2 Random Forest Classifier

The second classifier used in our modeling was a random forest classifier, an ensemble technique based on building multiple decision trees. The first training of the model yielded low metrics scores – in the mid-70% range. We suspected that this might be mainly driven by the imbalance in the target column data.



We used SMOTE library to balance the target data column by applying a pipeline preprocessing step to both over sample and under sample the target column. This had immediate impact in increasing our metrics to above 90%.



In order to train the model we used a pipeline that included the following steps:

1. Preprocessing : Scaling numeric columns using sciklearn Standard Scaler
2. Feature selection : using sciKlearn PCA
3. Classification : using sciklearn Random Forest Classifier.

We further improved the model scores by hyper tuning the pipeline steps using Grid Search, to overcome overfitting through cross validation which led to the following optimal parameters:



which were then used in the training step.

Random Forest feature selection was hypertuned by PCA to select the 14 optimal column as follow:

1. [ ('Area', 0.23763497281650664),
2. ('Perimeter', 0.18889781330536506),
3. ('AspectRation', 0.11067768284099898),
4. ('Eccentricity', 0.08737258914851624),
5. ('ConvexArea', 0.06097183785034905),
6. ('Solidity', 0.05488122933538511),
7. ('EquivDiameter', 0.051849904962902885),
8. ('roundness', 0.04618557544067638),
9. ('Extent', 0.0436463633287806),
10. ('Compactness', 0.0322153457399157),
11. ('MajorAxisLength', 0.029167819169617155),
12. ('ShapeFactor1', 0.02097247479647778),
13. ('ShapeFactor2', 0.018458852094132155) ,
14. ('MinorAxisLength', 0.017067539170376407)]

## 4.3 Linear SVM Classifier

Our third chosen classifier was a multiclass implementation of Support Vector Machine, implemented in scikit-learn as LinearSVM. This method extends the binary classification of SVMs to three or more classes as “one vs. rest”: that is, a distinct SVM is built for each class to determine whether data points are in or out of that class; each data point is then scored by each of these SVMs, with the point being classified into the class that achieved the highest score.

Because of the scoring method’s sensitivity to geometric shearing, we standardized all of the features in the training data before fitting the model. We tuned our hyperparameters using a grid search, which yielded the values C=10.0, tolerance=0.0003, and penalty=l2. To compensate for the different numbers of elements in the classes, the class\_weight hyperparameter was set to “balanced”.

## 4.4 Voting Classifier

The last classifier used in our analysis is an ensemble learning method, voting classifier, which predicts a target value by using the majority vote (i.e. hard voting) or the average predicted probabilities (i.e. soft voting) of the pre-trained classifiers. The advantage of the ensemble methods is that multiple machine learning classifiers or regression models are combined to improve the results.

There are two types of voting classification, hard voting and soft voting. The hard voting returns the class most voted for by the classifiers. For example, if Classifier 1 and 2 predict a sample as Class 1 and Classifier 3 predicts it as Class 2, the voting classifier will label the sample as Class 1 based on the majority class labels. In the case of a tie, the voting classifier will label the class based on the ascending sorted order. For example, if Classifier 1 predicts a sample as Class 2 and Classifier 2 predicts it as Class 1, the class label 1 will be assigned to the sample. On the other hand, soft voting averages the output probabilities from all the classifiers. The final class label is derived from the class label with the highest average probability.

Both hard and soft voting were implemented in our analysis. KNN, random forest and linear SVM classifiers were all included in the hard voting classifier. The soft voting requires the input classifiers to calculate the class probabilities in order to predict the class labels. However, the linear SVM classifier is deterministic in nature and hence does not have the predict\_proba method. Thus, only KNN and random forest classifiers were included in the soft voting classifier.

# 

# 5.0 Model Evaluation

## 5.1 kNN Classifier

Most importantly, accuracy scores were calculated for this model using full 16 features or only 10 features. The accuracy score for 16-feature model is 0.908, while the one for 10-feature model is 0.889. The table below shows more details of the scores. The accuracy score is nearly perfect for 16-features. This led us to think of possible overfit of the model, so we choose a second approach with fewer features to check if overfitting is an issue.

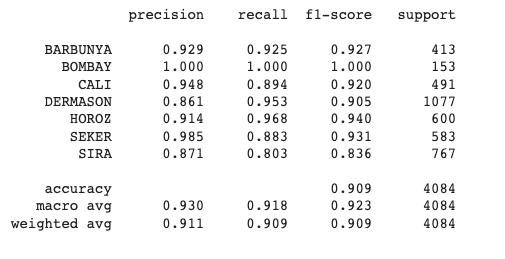


Table 6: Classification report of the kNN classifier with all 16 features

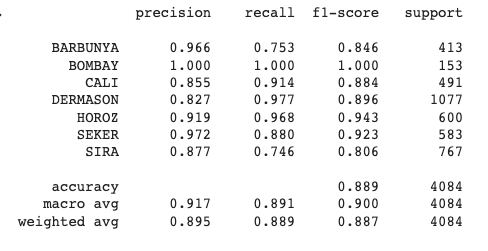


Table 7: Classification report of the kNN classifier with 10 features

Secondly, the confusion matrix of each approach was calculated and drawn as heatmaps. The figures below show the confusion matrix of both approaches. We can see that the model with 10 features is much less accurate in predicting Class 2 and 3 (Bombay and Cali) according to the confusion matrices.

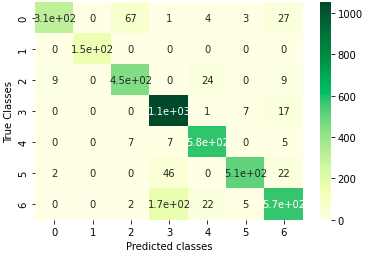


Figure 2: Confusion matrix for the KNN classifier with 10 features

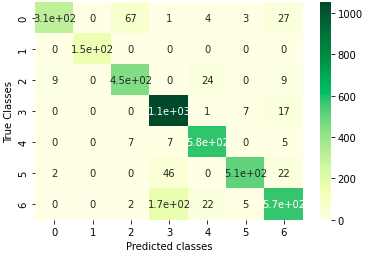


Figure 3: Confusion matrix for the KNN classifier with 16 features

Lastly, learning curves of the models were evaluated. The training scores are very high for both approaches. The scores are almost close to one. The dataset contains very organized data of high quality that are perfect for this machine learning training process.

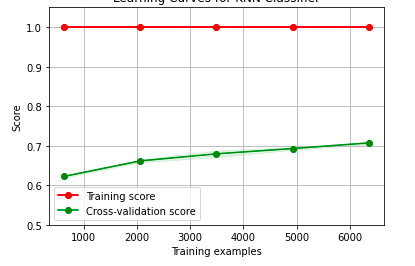


Figure 4: Learning curves for the KNN classifier with 16 features

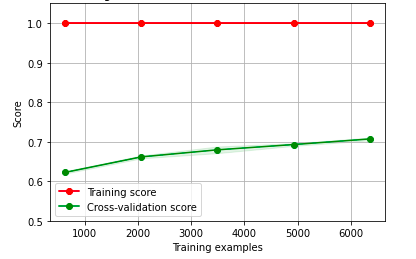


Figure 5: Learning curves for the kNN classifier with 10 features

## 5.2 Random Forest Classifier

Accuracy score for Random Forest was 0.95 , and f1 score for multi-classes ranged between (0.9-1)

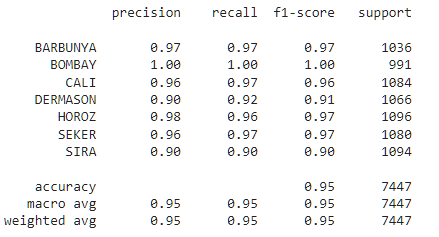


Table 8: Classification report of the random forest classifier

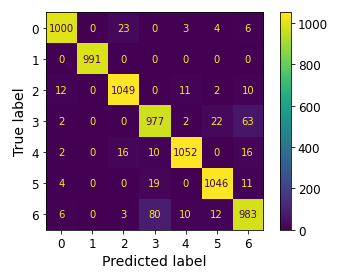


Figure 6: Confusion matrix for the random forest classifier

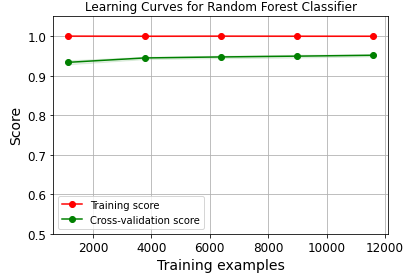


Figure 7: Learning curves for the random forest classifier

## 5.3 Linear SVM Classifier

The accuracy score for the SVM classifier was 0.94; f1-scores for individual classes ranged from 0.89 to 1.00. As with the kNN and RF classifiers, class 1 (Bombay) was always recognized, while the most common confusions were between classes 0 & 2 (Barbunya / Cali) and 3 & 6 (Dermason / Sira); these confusions were notably more symmetric for RF and SVM than they were for kNN.

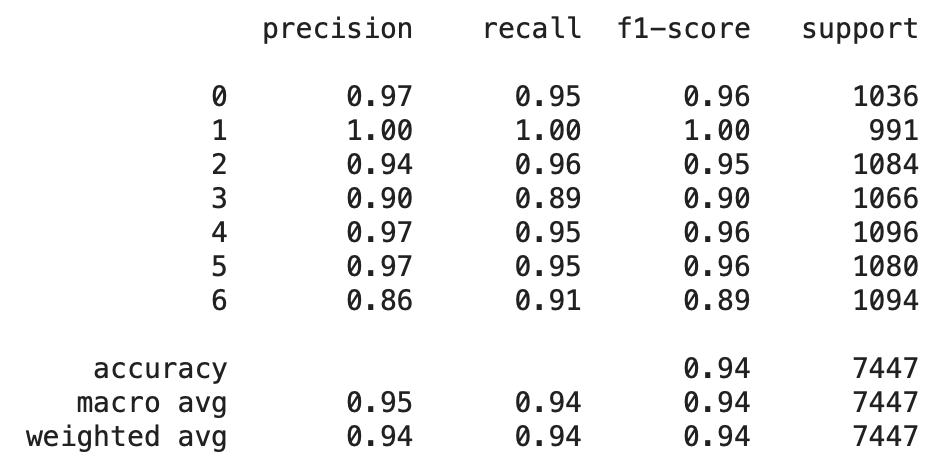


Table 9: Classification report of the linear SVM classifier

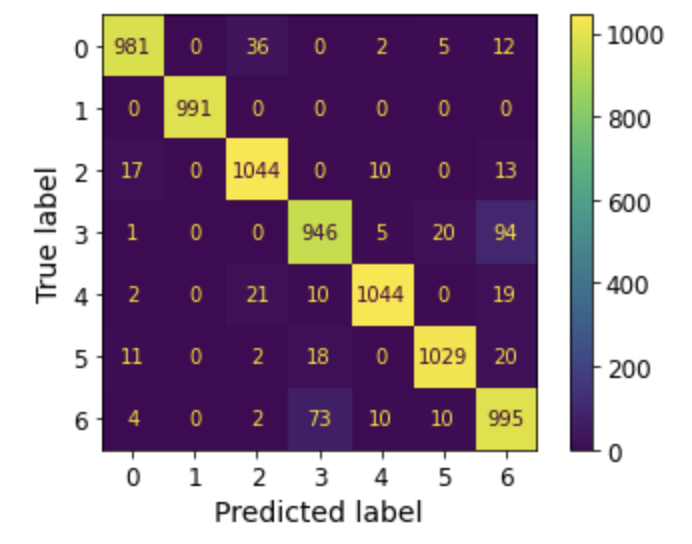


Figure 8: Confusion matrix for the linear SVM classifier

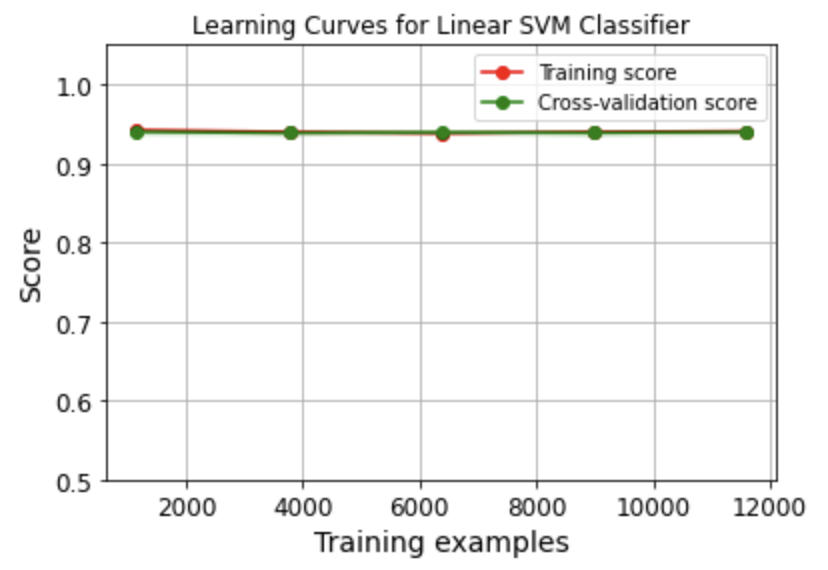


Figure 9: Learning curves for the linear SVM classifier

## 5.4 Voting Classifier

The accuracy scores for both hard and soft voting classifiers were 0.95. The f1-scores of hard voting for individual classes ranged from 0.90 to 1.00. The f1-scores of soft voting ranged from 0.89 to 1.00. The same as KNN, random forest and linear SVM classifiers, there was no misclassification in class 1 (Bombay). Most misclassifications occur between class 3 (Dermason) and 6 (Sira). These two classes had the lowest accuracy, precision, recall and f1-scores among all classes. This suggests that the qualities of these bean varieties that lead to these confusions are intrinsic, rather than an artifact of the classifier algorithms: given the disparate bases of the algorithms, we would not expect the same data points to be misclassified every time (or at least two out of three) unless there was a real phenomenon to observe.

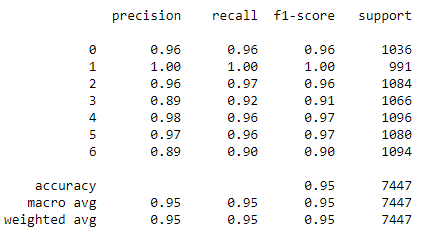


Table 10: Classification report of the hard voting classifier

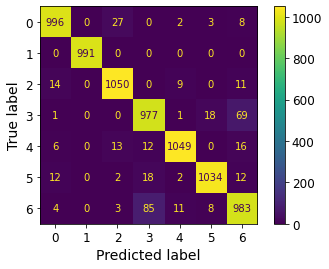


Figure 10: Confusion matrix for the hard voting classifier

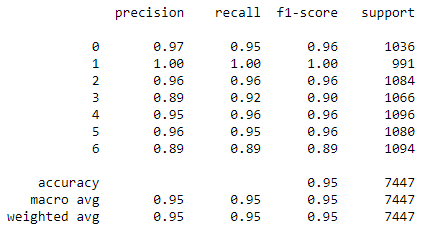


Table 11: Classification report of the soft voting classifier

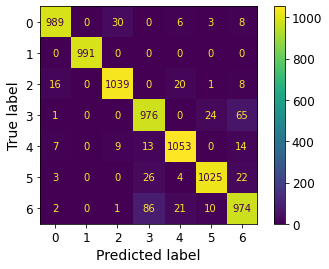


Figure 11: Confusion matrix for the soft voting classifier

The learning curves show that the training scores are very high for both hard and soft voting classifiers. The cross-validation scores increase a little as the training data set size increases. Overall, the data set is high quality and the classifiers perform very well on the data set.

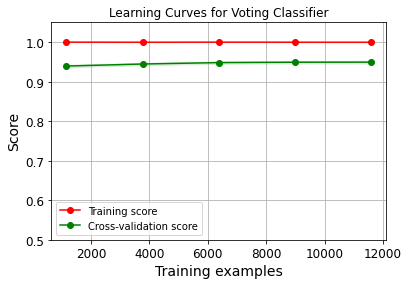


Figure 12: Learning curves for the hard voting classifier

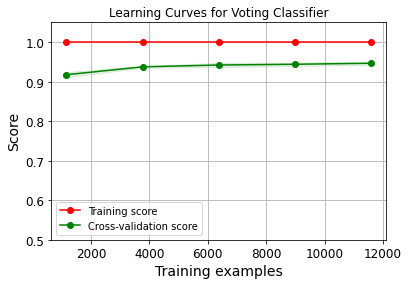


Figure 13: Learning curves for the soft voting classifier

# 

# 6.0 Conclusion

All the classifiers that we looked at have high accuracy scores and high training scores close to 1. According to the confusion matrices of each classifier, most wrong predictions/confusions occurred on class 3 and 6 (Dermanson and Sira). However, according to the confusion matrices, the performance of the models are: random forest = voting > linear SVM >kNN classifier. The dry bean dataset is very neat and clean so it fits well for any of the classifiers that we considered.

# 7.0 References

Gentry, H.S., 1969. Origin of the common bean, *Phaseolus vulgaris*. Econ. Bot. 23, 55–69 <https://www.jstor.org/stable/4253014>

Koklu, M. and Ozkan, I.A., (2020), Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques. Computers and Electronics in Agriculture, 174, 105507. DOI: <https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset#>

Chen, X., Xun, Y., Li, W., Zhang, J., 2010. Combining discriminant analysis and neural networks for corn variety identification. Comput. Electron. Agric. 71, S48–S53. <https://doi.org/10.1016/j.compag.2009.09.003>

Kumar A., 2020, KNN Algorithm: When? Why? How? <https://towardsdatascience.com/knn-algorithm-what-when-why-how-41405c16c36f>