Exploring Algorithms for Dry Bean Classification

EE 559 Course Project

Data Set : Dry Beans

Shyam Sunder Rajasekaran, srajasek@usc.edu

26 April 2024

1. Abstract

The project aims to analyze the best algorithm for classification of dry beans. The Dry Bean Dataset used contains characteristics of various dry bean types, including form, shape, type, and structure. In order to predict the type of dry bean based on these features, the project uses a variety of models, such as Artificial Neural Networks (ANN), Random Forest (RF), K-Nearest Neighbors (KNN), and Support Vector Machines (SVM) and also a baseline model is used.

A strong methodology that includes data preprocessing, feature engineering, oversampling using SMOTE, and principal component analysis for dimensionality reduction is used to train and test the models. Cross-validation is used to assess each model's performance, and accuracy and F1 scores are used to compare the outcomes. The best model for this classification task was the Random Forest and the neural network (mlp) which obtained the highest accuracies and F1 score. The results of this investigation may enhance the efficacy and precision of the classification of dry beans.

2. Introduction

2.1. Problem Statement and Goals

The dataset selected for this task consists of 13611 dry bean grains from 7 distinct classes. The objective of the problem is to correctly predict the class of a given dry bean grain based on its features, and it is approached as a classification task. This project's main objective is to train multiple models on the dataset, evaluate each model's performance, and choose the top model. The ultimate goal is to improve dry bean classification's effectiveness and accuracy, which could lead to higher agricultural productivity and sustainability.

There are sixteen features in the dataset used for this project. The dry bean different measurements and characteristics are provided by these features. Among the features are: Major axis length, minor axis length, area perimeter, and aspect ratio. A dry bean grain's class is largely determined by each of these characteristics. These features will be used to train machine learning models, which will subsequently be used to predict the class of previously undiscovered test class.

2.2. Literature Review

In traditional machine learning workflows, encoding categorical features often involves techniques like one-hot encoding, label encoding, or ordinal encoding. These methods can be memory-intensive, Feature hashing, also known as the hashing trick, offers a memory-efficient alternative for encoding categorical features. Rather than creating separate binary columns for each category, feature hashing maps categories to a fixed number of bins, providing numerical values for categorical features.

Scikit-learn [1] provides an implementation of feature hashing through the FeatureHasher class. This class offers a memory-efficient solution for encoding categorical features, making it suitable for large datasets and online learning scenarios. Enables machine learning models to handle categorical features more efficiently. Scikit-learn provides an implementation of feature hashing through the [2] FeatureHasher class, offering a memory-efficient solution for encoding categorical features.

.

3. Approach and Implementation

3.1. Dataset Usage

The Dry Bean Dataset, which includes features denoting various bean characteristics and a target variable designating the bean's class or type, is used in this project. Using the supplied dry_bean_classification_train.csv and dry_bean_classification_test.csv files, the dataset is divided into training and test sets. The training dataset contains 10889 training samples and 2722 test samples and the number of features present is 16. The target column, which represents the class labels, is identified as 'class'.

A five-fold stratified approach is used for cross-validation. This procedure is carried out by the CrossValidation class's cross_validate_and_predict function. It divides the training data into five folds while maintaining the class distribution in each fold using the StratifiedKFold class from scikit-learn. The function then computes the macro-averaged F1 scores for both the training and validation sets after training the designated models on the training folds and assessing their performance on the validation folds. The test set is only used once, at the very end of the process, to assess the chosen models' final performance. The Figure.1. shows the class distribution in the training data.

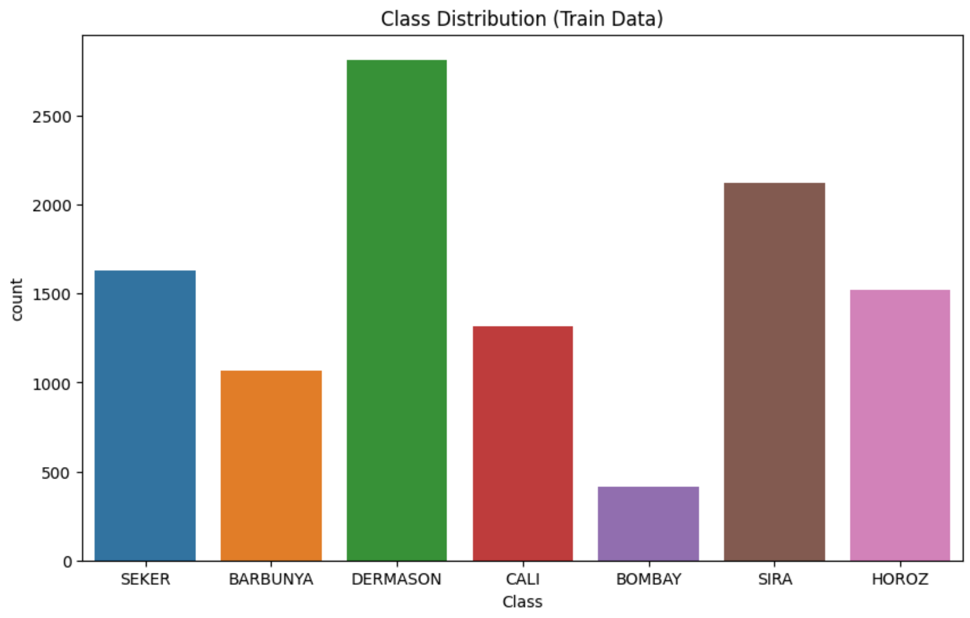

Figure.1. Class distribution

The class distribution in the train data is as follows: class 3 has 2815 samples, class 6 has 2123 samples, class 5 has 1631 samples, class 4 has 1522 samples, class 2 has 1316 samples, class 0 has 1068 samples, and class 1 has 414 samples.

3.2. Preprocessing

The data is preprocessed using multiple methods used to Label encoding: To translate the class labels into numerical values, the target variables (y_train and y_test) are label-encoded using Scikit-Learn's LabelEncoder. Both in the training and test sets, the code looks for missing values. There are no missing values in the dataset. The Scikit-learn's StandardScaler is used to scale the features (X_train and X_test). It does this by taking the mean out of the features and scaling them to the variance.The minority classes in the training data were oversampled using the SMOTE class from the imblearn.over_sampling module seen in the Figure.2

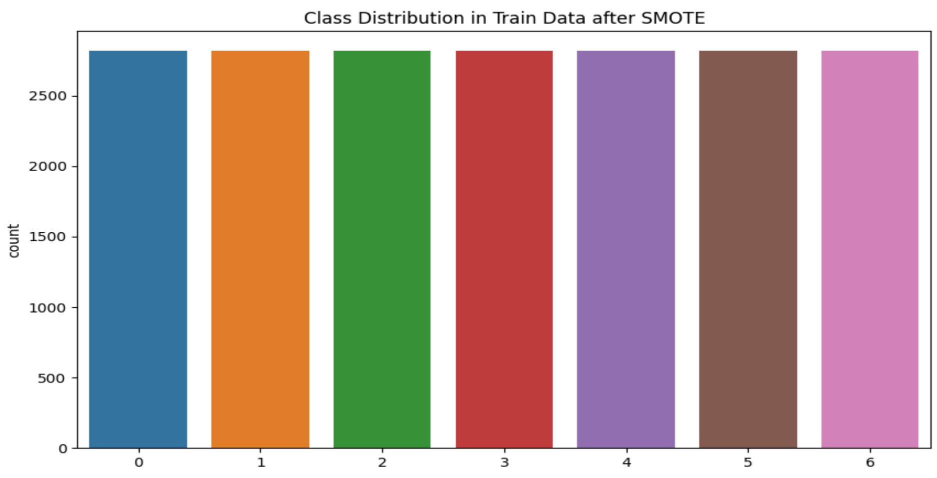

Figure.2.Class Distribution after SMOTE

Data augmentation involves creating an augmented dataset (X_train_aug) by adding Gaussian noise with a mean of 0 and a standard deviation of 0.1 to the scaled training data (X_train_scaled).. This action was taken to rectify the dataset's class imbalance. From the augmented training data (X_train_aug), outliers are found and eliminated using an EllipticEnvelope detector from scikit-learn. Since the contamination parameter is set to 0.1, 10% of the data points are thought to be anomalies.

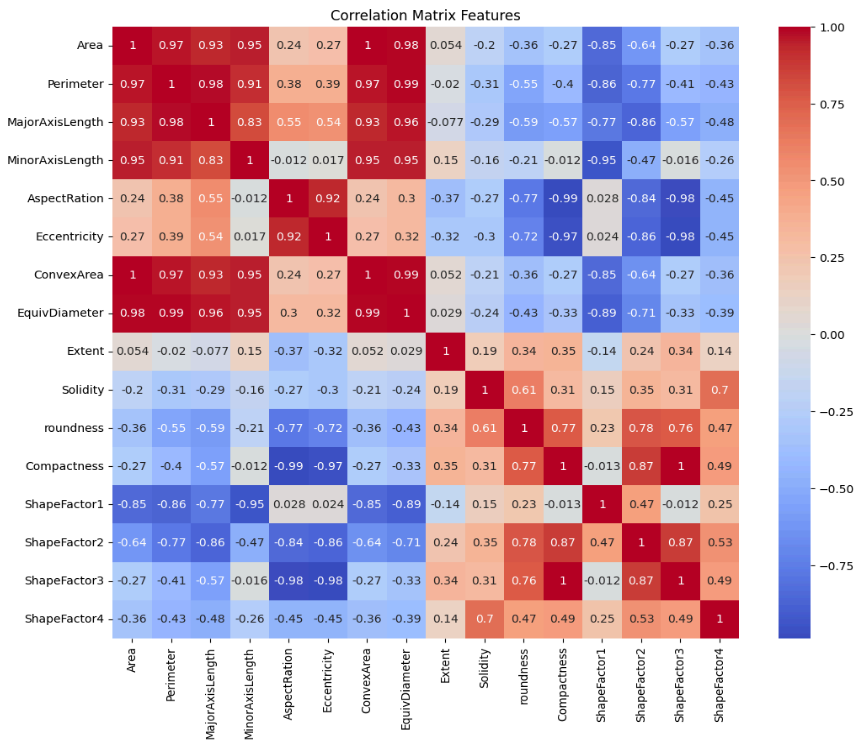

Figure.3. Correlation Matrix

The Figure.3. shows the correlation matrix features of the data, also computes the pairwise correlation coefficients between all numerical features and visualizes them using a heatmap.

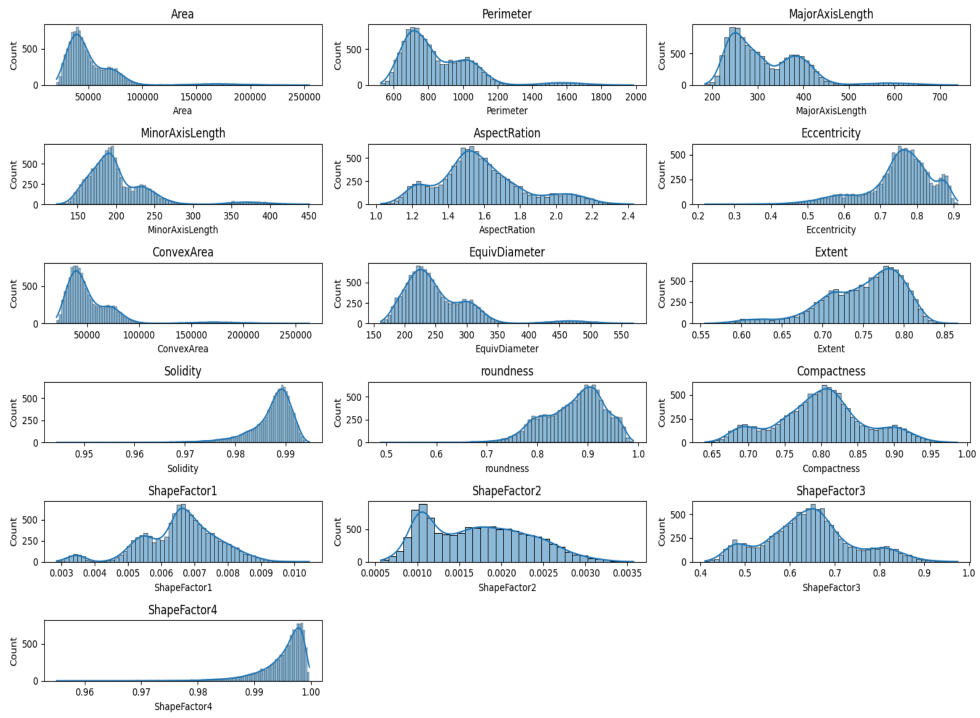

Figure.4. Numerical Features

The Figure.4. visualizes the numerical features, The subplot displays a histogram and a kernel density estimate for a single numerical feature.

3.3. Feature engineering and dimensionality adjustment

From the oversampled training data (X_train_sm), polynomial features of degree 2 are generated using the scikit-learn PolynomialFeatures class. For the training and test sets, the generated polynomial features are kept in X_train_poly and X_test_poly, respectively. The Figure.5. shows a scatter plot of cleaned data after the data processing before feature engineering.

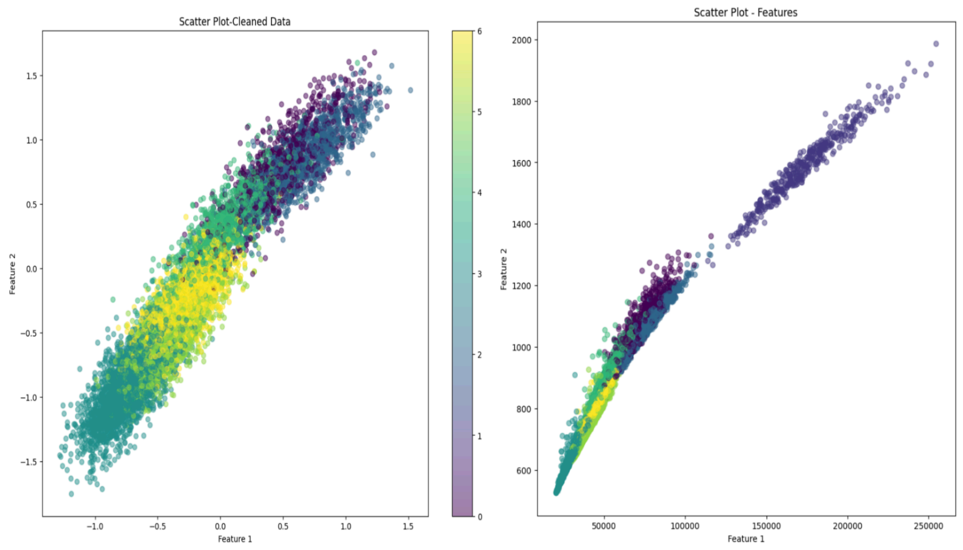

Figure.5. Scatter plot of data

Choosing features: The mutual information score between each feature and the target variable is calculated using the scikit-learn mutual_info_classif function. After the selection of the top 8 features with the highest mutual information scores, the training and test data (X_train_selected and X_test_selected) are modified appropriately. FeatureHasher from scikit-learn is used to encode the target variables (y_train and y_test) after they have been converted to a string. The chosen numerical features (X_train_selected and X_test_selected) are concatenated with the encoded features that are produced.

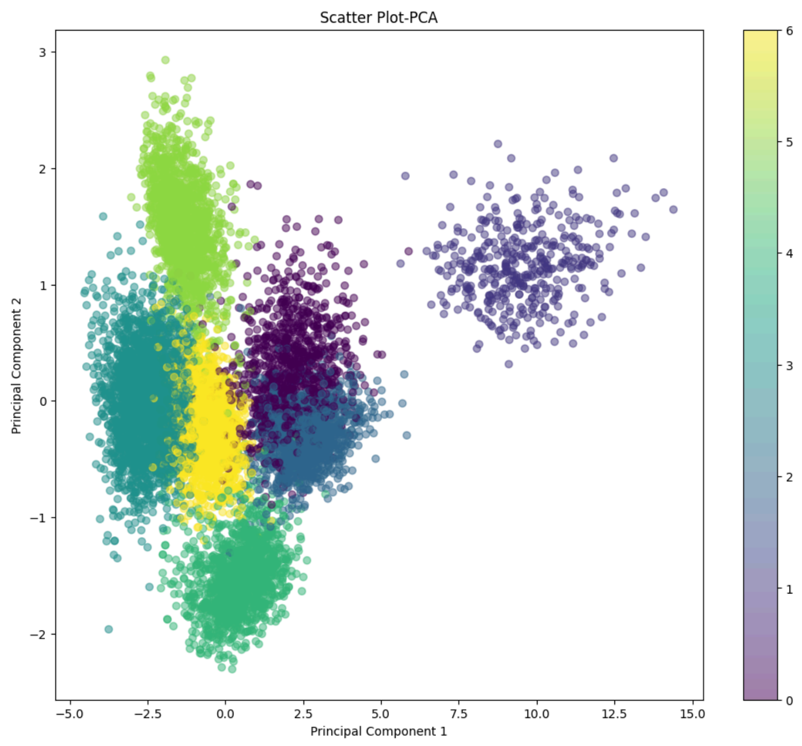

Figure.6. Scatter plot after PCA

Using the PCA class from scikit-learn, PCA is applied to the encoded and selected features (X_train_selected and X_test_selected). To preserve 94% of the variance in the data, the number of components was selected. For the training and test sets, respectively, the transformed data is kept in X_train_pca and X_test_pca. The scatter plot between two features is shown in Figure.6.

3.4. Training, classification and/or regression, and model selection

The following machine learning models are put into practice and assessed by the project: Trivial System, Baseline System, artificial neural network, KNeighborsClassifier, Support Vector Machine.

Based on the class distribution in the training data, a trivial system randomly allocated class labels to the test data. This provides a reference point for comparison. A simple baseline model [3] represented by the scikit-learn NearestCentroid classifier. On the training and validation sets, the baseline system is trained and assessed, and its results are reported for the test set. A scikit-learn MLPClassifier [4] with a hidden layer size of 100 and a maximum of 500 iterations is an example of an artificial neural network (ANN). The training and validation sets are used to train and assess the ANN, and the test set performance is reported. A Random Forest Classifier [5] with 150 estimators, a maximum depth of 10, and a minimum of 5 samples per leaf is available from scikit-learn. On the training and validation sets, the Random Forest is trained and assessed, and its results on the test set are reported.

A KNeighborsClassifier [6] from scikit-learn with five neighbors is called K-Nearest Neighbors (KNN). The KNN model's performance on the test set is reported after it has been trained and assessed using the training and validation sets. Support Vector Machine (SVM) [7] An automatic gamma value selection feature and an RBF kernel are features of this scikit-learn SVC. The SVM's performance on the test set is reported after it has been trained and assessed using the training and validation sets.

Model	Accuracy (Train)	Macroavg F1score (Train)	Microavg F1score (Train)	Accuracy (Validation)	Macroavg F1score (Validation)	Microavg F1score (Validation)
Support Vector Machine	0.960	0.948	0.960	0.954	0.943	0.954
KNearest Neighbors	0.969	0.959	0.969	0.945	0.932	0.945
Random Forest	0.971	0.961	0.971	0.956	0.946	0.956
ANN	0.969	0.958	0.969	0.959	0.947	0.959
Baseline	0.895	0.883	0.895	0.881	0.869	0.881
Trivial System	0.168	0.140	0.168	-	-	-

Table.1. Train and Validation Performance

From Dividing the data using train_test_split from scikit-learn into training and validation sets. The training set is utilized and trained on the model. Accuracy, macro-averaged and micro-averaged F1 scores, and confusion matrices on the test set are used to compare each model's performance. Table 1. displays the models' respective performance scores.

Model	Data	Mean	Std
Random Forest	Train	0.960	0.001
Random Forest	Validati on	0.945	0.004
KNN	Train	0.880	0.001
KNN	Validati on	0.813	0.008

Table.2. Cross-Validation scores

The Random Forest model outperforms the KNN model in terms of cross-validation [8] mean accuracy scores on both the training and validation datasets, as can be seen in the Table.2.. The Random Forest model also performs better consistently across multiple cross-validation runs, as evidenced by a lower standard deviation, which gauges the degree of variation in the scores. However, the KNN model's performance appears to be less stable and more dependent on the selection of the training and validation sets, as evidenced by its lower mean accuracy score and higher standard deviation, particularly on the validation dataset.

4. Results and Analysis: Comparison and Interpretation

After evaluating the performance of the test set, it is known that with a micro-averaged F1 score of 0.970, a macro-averaged F1 score of 0.959, and an accuracy of 0.970, the Artificial Neural Network performs best overall across all evaluation metrics inside EE559 scope. With accuracies and F1 scores that are comparable to the best model, the Random Forest and Support Vector Machine models likewise function exceptionally well as shown in Table.2.

Model	Accuracy	Macroavg F1score	Microavg F1score
Trivial	0.175	0.145	0.175
Baseline	0.893	0.878	0.893
Artificial Neural Network	0.970	0.959	0.970
Random Forest	0.964	0.952	0.964
KNearest Neighbors	0.956	0.942	0.956
Support Vector Machine	0.964	0.952	0.964

Table.2. Test set performance

While it performs worse than the other models, the NearestCentroid classifier-based Baseline System outperforms the Trivial System. As predicted, the Trivial System performs poorly, assigning class labels at random based on the distribution of training data.

Additional information about each model's classification performance across various classes can be found in its confusion matrices below. While the Trivial System and Baseline System have higher misclassification rates for some classes, the Random Forest and Support Vector Machine models have the lowest misclassification rates overall. The Random Forest and Support Vector Machine models have more parameters and possibly more complex decision boundaries in terms of degrees of freedom and constraints.

The heatmaps of the best-performing models, such as the Support Vector Machine, Random Forest, and Artificial Neural Network as shown on Figure.7. and Figure.8. and Figure.9., exhibit a strong diagonal pattern, signifying a high concentration of accurate predictions along the main diagonal. For these models the misclassifications represented by the off-diagonal elements are relatively small.

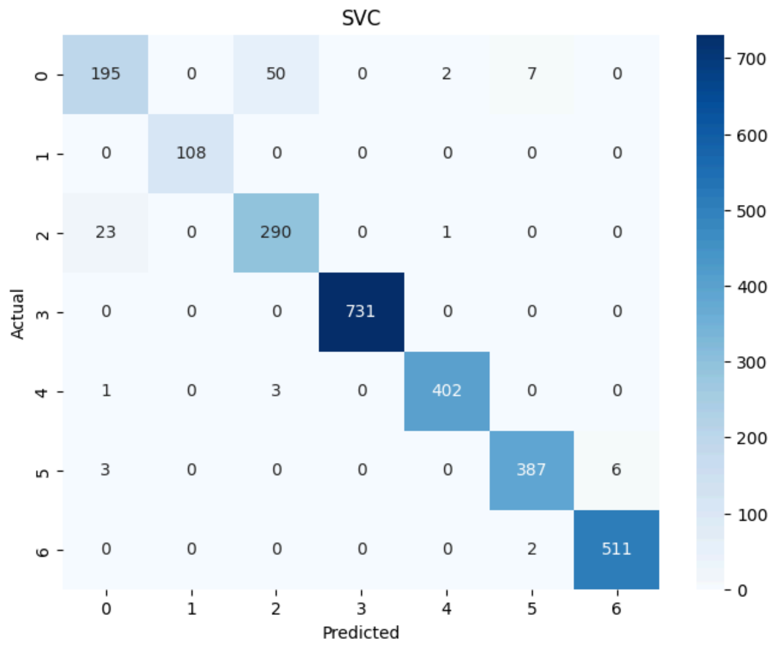

Figure.7. Heatmap of SVC

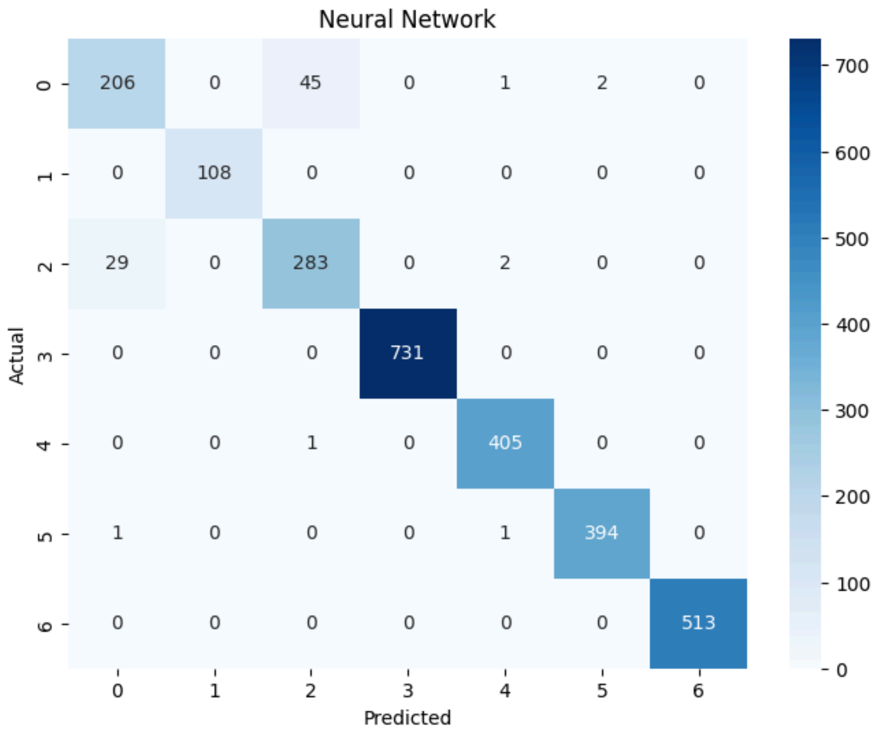

Figure.8. Heatmap of mlp

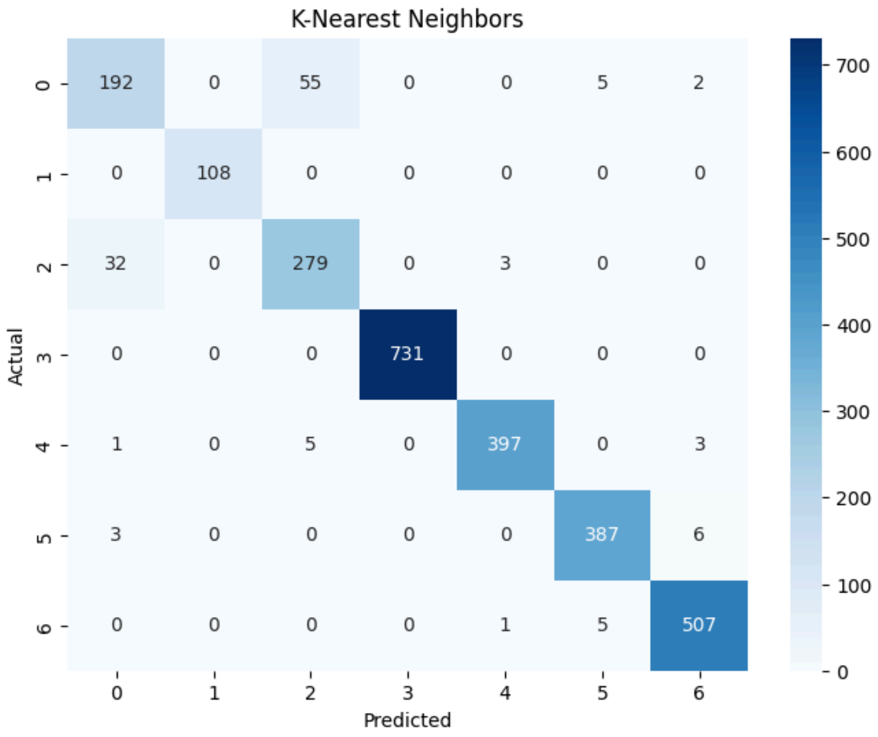

Figure.9. Heatmap of KNN

The Trivial System's heatmap as shown in Figure.10., on the other hand, is showing a more uniform distribution because class labels are assigned at random without taking into account the real data patterns.

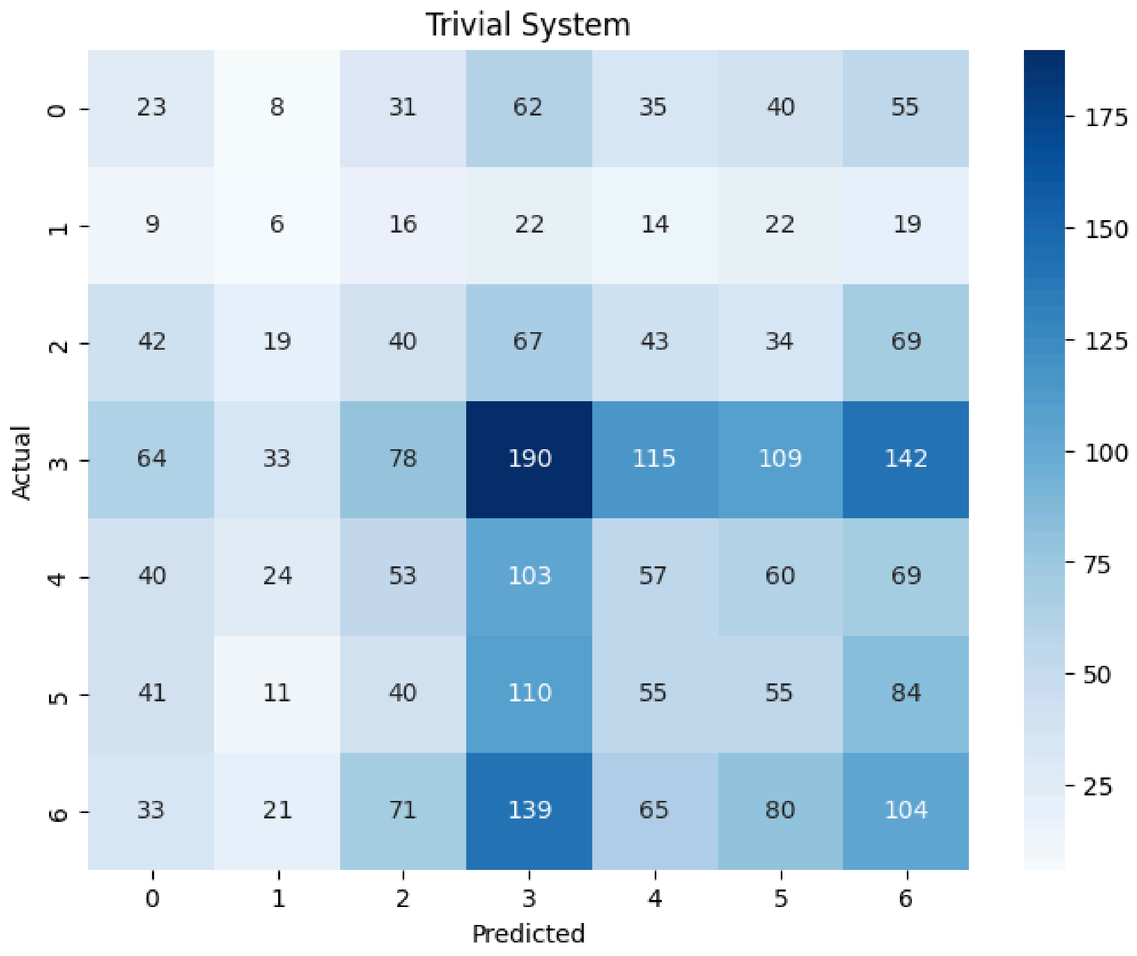

Figure.10. Heatmap of Trivial system

5. Libraries used and what you coded yourself

Several Python libraries are used in the project for feature engineering, data preprocessing, and the implementation and assessment of machine learning models: pandas, imblearn, scikit-learn, seaborn, matplotlib, and numpy. Pandas is used for preprocessing, loading, and manipulating data, and matplotlib and seaborn are used for data visualization features like heatmaps, correlation matrices, and plotting. Numpy is used for manipulating arrays and performing numerical operations. For machine learning models, preprocessing methods (like StandardScaler and

LabelEncoder), model evaluation metrics, and cross-validation, a lot of people use the scikit-learn library. In order to handle imbalanced data, the oversampling technique (SMOTE) also makes use of the imblearn library.

The FeatureHasher [2] is a technique for encoding categorical features by hashing them into a fixed-size feature vector

Preprocessing, model training, assessment, and visualization were all carried out with reference to the scikit-learn library and its features.. No additional libraries or functions outside of the EE 559 methods were used.

6. Contribution and Learning

Preprocessing the dry bean dataset, training and testing a variety of classification models (including Random Forest, SVM, ANN, KNN, and others), and carrying out cross-validation were all part of the project. When it came to evaluation metrics like accuracy and F1 scores, the Random Forest performed the best.

Hyperparameter tuning, ensemble approaches, model interpretability analysis, improved management of class imbalance, and investigation of transfer learning are some possible areas of future research.

Important Takeaways:

- Assessing several models and contrasting their output
- The significance of appropriate evaluation metrics (F1 scores, both micro- and macro-averaged)
- Performance and model complexity are traded off.
- Importance of confusion matrix heatmaps as visualizations to understand performance
- Using different machine learning techniques in a practical way

References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.FeatureHasher.html

[2] "Practical Lessons from Predicting Clicks on Ads at Facebook" by Hebich et al Available:https://research.fb.com/wp-content/uploads/2016/11/practical-lessons-from-predicting-clicks-on-ads-at-facebook.pdf

[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestCentroid.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[7] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html