Evaluating Machine Learning Classifiers for Dry Bean Classification

EE 559 Course Project

Data Set: Dry Beans

Swetha Shankar, swethash@usc.edu

26 April 2024

1. Abstract

A brief (typically ~200 words), informative description of your project. Include the This project addresses the problem of classifying dry beans into distinct categories based on a given dataset with seven unique classes. The dataset contained 16 features, including traditional metrics like area, perimeter, and compactness, as well as shape-related factors such as eccentricity and solidity. The main goal was to build and evaluate various machine learning models to identify the most accurate classifier for distinguishing among the different types of dry beans.

The machine learning methods applied included Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Multilayer Perceptron (MLP), Nearest Means Classifier, and a trivial random classifier for baseline comparison. Dimensionality reduction techniques, such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), were used to improve model performance.

The key results revealed that Multilayer Perceptron (MLP) with PCA achieved the best overall test accuracy of 90.78% with minimal overfitting. Support Vector Machine (SVM) with PCA and K-Nearest Neighbors (KNN) with LDA also showed promising results, with test accuracies of 91.25% and 90.33%, respectively. The trivial random classifier, serving as a baseline, achieved a test accuracy of approximately 14%.

These findings suggest that the combination of MLP and PCA is the most effective approach for dry bean classification, with the potential for further improvements through additional hyperparameter tuning and regularization.

2. Introduction

2.1. Problem Statement and Goals

This project focuses on the analysis of a dataset containing 13,611 records of dry beans across seven distinct classes. The dataset provides detailed information on 16 features, including dimensions and shape forms of the beans, with the primary objective of building classification models to differentiate among the various classes of dry beans [5].

To identify patterns and relationships within the data, a classification approach is best suitable. The goal of this project is to build five machine learning models for classification, which include trivial, baseline, support vector classification, neural network and non-probabilistic (distribution-free) that can accurately categorize the dry beans into their respective classes based on these features.

These features comprise both traditional metrics such as area, perimeter, and compactness, as well as more complex factors like eccentricity, solidity, and shape factors. The extensive feature set provides a rich foundation for developing and evaluating various classification models. The dataset serves as a valuable resource for exploring data-driven insights and can inform agricultural or food science applications.

2.2. Literature Review

To explore existing methods for classifying dry beans, two research papers were reviewed along with existing sources [4], with a focus on machine learning concepts and their application to this domain.

The first paper [2] presented a study on haricot bean classification using three different machine learning models: InceptionV3, VGG16, and VGG19. While these models were pre-trained convolutional neural networks (CNNs), the paper also employed traditional machine learning techniques like Support Vector Machines (SVM) and Logistic Regression (LR). The authors extracted image features from the final layers of the CNNs and used them to train SVM and LR models. This approach demonstrates the utility of hybrid models, combining feature extraction with machine learning classifiers, to improve classification performance. The results indicated that InceptionV3, with SVM, achieved a success rate of 79.60%, while LR with VGG16 attained 83.71% success. This paper highlighted the effectiveness of machine learning classifiers when trained on extracted features, offering insights into advanced data processing and classification techniques.

The second paper [3] focused on a dataset with 13,611 grains from seven different types of dry beans, which aligns with the dataset used in this project. The paper employed 16 features, including 12 dimensions and four shape-related factors, to classify the beans. The study used machine learning classifiers like Multilayer Perceptron (MLP), Support Vector Machines (SVM), k-Nearest Neighbors (KNN), and Decision Trees (DT), implementing a 10-fold cross-validation approach. The results indicated that SVM achieved the highest accuracy at 93.13%, demonstrating the effectiveness of traditional machine learning models in bean classification. The paper's methodology and dataset are highly like the current project, offering a useful reference point for feature extraction, data preprocessing, and model selection.

Overall, these papers provide relevant insights into various machine learning concepts and their application to dry bean classification. They demonstrate the use of different classifiers, the importance of feature extraction, and the value of cross-validation in achieving high accuracy rates.

3. Approach and Implementation

3.1. Dataset Usage

The dataset used in this project consists of 13,611 records of dry beans from seven different classes. This dataset was split into training and testing sets, resulting in 10,889 data points for training and 2,722 data points for testing**.**

Data Splitting and Cross-Validation

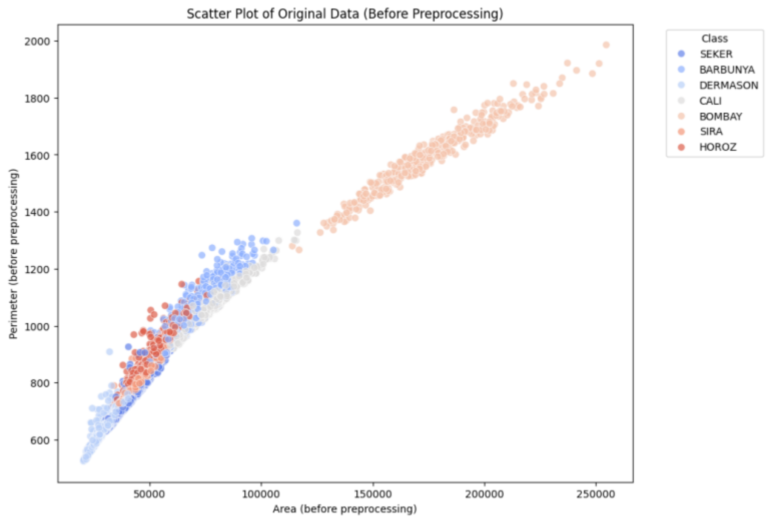
To improve model robustness, the training set underwent outlier removal, reducing the number of training data points to 9,970. After further preprocessing however, the training dataset became 19,523 which was later split into training and validation sets during cross-validation. To split the data, the project used 5-fold cross-validation in each of the five runs, 25 folds in total. In each fold, the training and testing data points were split into a 4:1 ratio, resulting in 15,619 data points for training and 3,904 data points for validation. The sequential loop iteration through the folds employed the KFold method, ensuring an even distribution of data across each fold. Cross-validation was used to ensure the robustness of the training process and to evaluate model performance. The validation sets were created during each cross-validation fold, ensuring that each model had a different subset of data for validation. This approach helped identify the best-performing model based on accuracy.

Test Set Usage

The test set was utilized whenever the dimensionality of the labels (y) was changed. This ensured that both the training and testing sets had the same number of features, maintaining consistency during model evaluation. The test set was used to assess the model's performance after changes in the feature set.

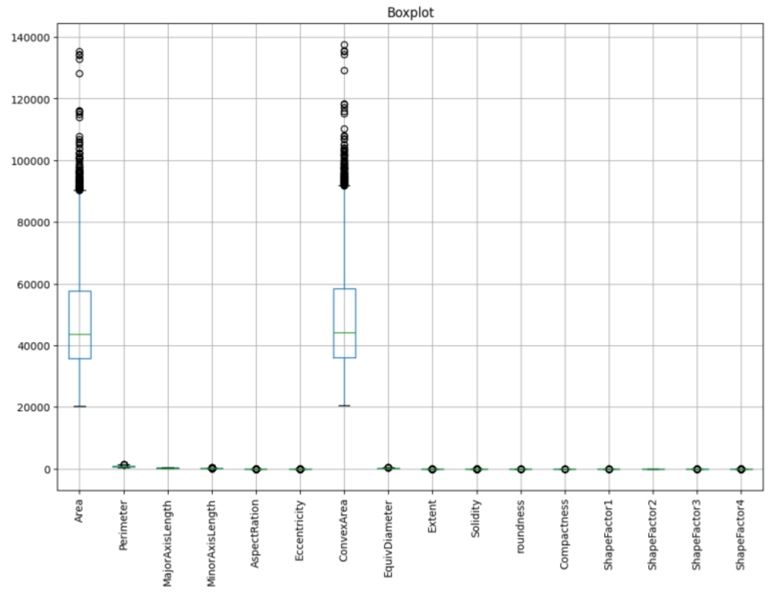
Dataset Visualization

To better understand the dataset's characteristics, a scatter plot was created using the "Area" and "Perimeter" features, with data points colored by class. This visualization helped identify patterns in the dataset and provided insights into the distribution of different classes before preprocessing.

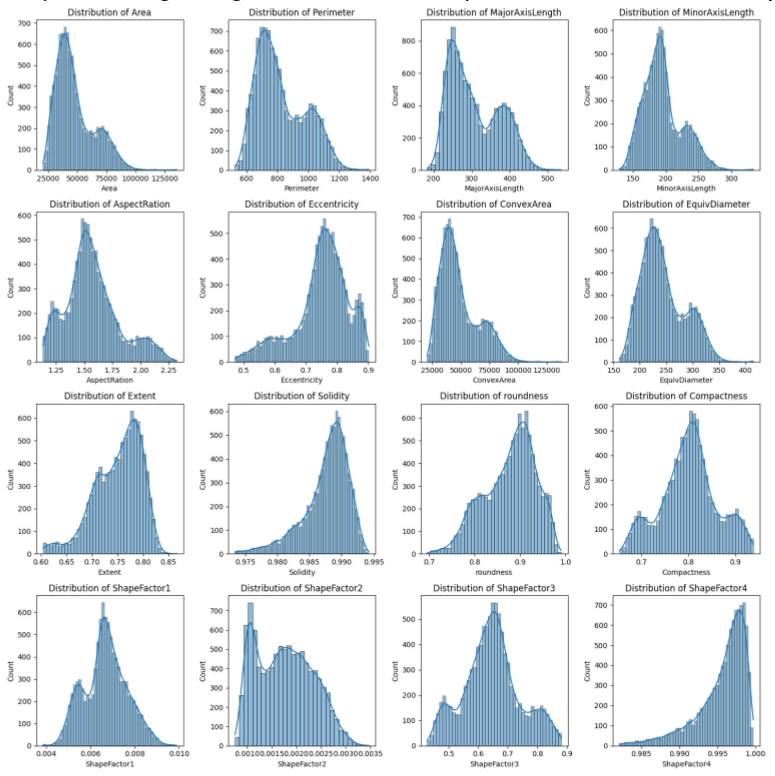


Boxplot

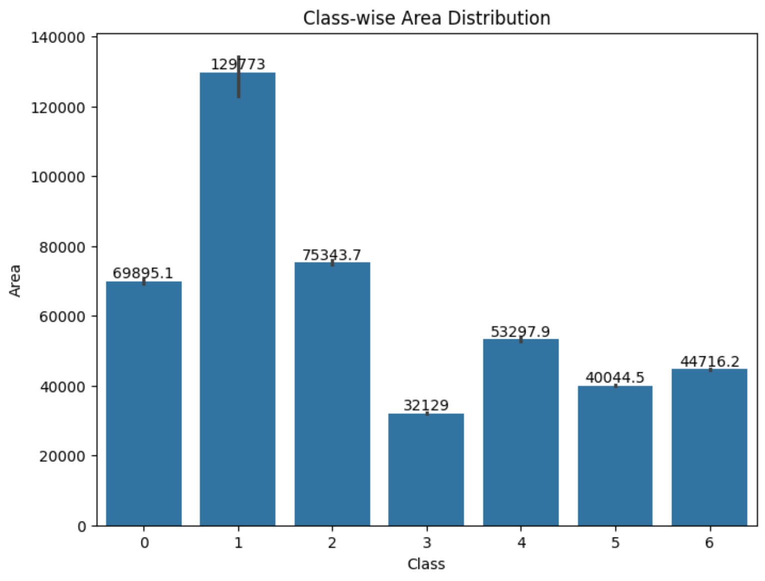
A boxplot visualization [1] was created to assess the distribution and spread of different features across the dataset. The boxplot provided a comprehensive view of the central tendency and variability of numerical features, helping to identify potential outliers.



Histograms
Histograms were generated to examine the distribution of numerical features across the dataset. This visualization revealed the frequency distribution for each feature, providing insights into the shape of the data and any skewness.

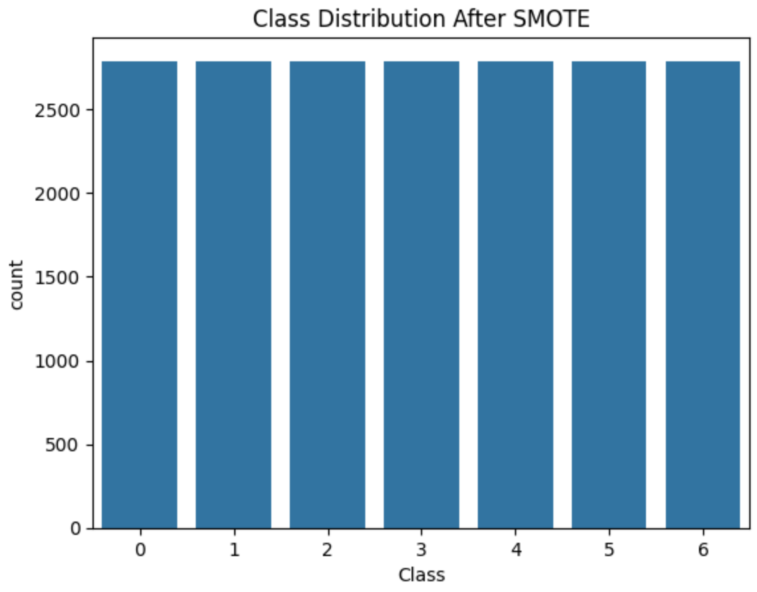


Barplot
A barplot was created to visualize the class-wise distribution of the "Area" feature, offering insights into the average values for each class.



Countplot

A count plot visualization was used to depict the class distribution after applying SMOTE for balancing. This plot helped visualize the effectiveness of SMOTE in equalizing class representation.



3.2. Preprocessing

To ensure the data is consistent, clean, and suitable for the chosen algorithms, the following preprocessing techniques were applied to the dry beans dataset:

Imputation

Missing data can impact the accuracy and performance of machine learning models. To address this issue, a SimpleImputer was used to fill in missing values with the median of the respective feature. The median was chosen as the imputation strategy because it is less sensitive to outliers than the mean, providing a more robust estimate of central tendency.

Outlier Removal

Outliers are data points that deviate significantly from the rest of the dataset. They can distort statistical analyses and reduce the accuracy of machine learning models. To address this, the z-score method was used. Data points with z-scores exceeding three standard deviations from the mean were removed, as these were considered significant outliers.

However, if there are many outliers in a dataset, removing them could lead to data loss. For instance, techniques like Interquartile Range (IQR) and Isolation Forest were overly aggressive in outlier removal, leading to the elimination of a significant number of data points, including those belonging to the 'BOMBAY' class. In such cases, it is common to replace outliers with the mean or mode to maintain the dataset's structure. Since the dry beans dataset contained a relatively small number of outliers and had a large overall size, the decision was made to remove them. This approach helped ensure data quality without excessively reducing the dataset size.

Label Encoding

For the machine learning algorithms to work with categorical data, the class labels, the only column with non-numerical data were transformed into numerical values using a LabelEncoder. This step involved mapping each unique class label to a unique integer. Label encoding was applied to both the training and testing sets to ensure consistency in the transformed class labels. This conversion is necessary for many machine learning models that require numerical input.

Standardization

To standardize the numeric features, a StandardScaler was applied to ensure that all features had a mean of zero and a standard deviation of one. This process helps normalize the data, allowing the machine learning algorithms to work with consistent feature scales.

Data Augmentation

To address class imbalance and enhance the robustness of the dataset, Synthetic Minority Over-sampling Technique (SMOTE) was used for data augmentation. SMOTE creates synthetic samples to balance the dataset, ensuring that each class has an adequate representation. This technique helps prevent models from becoming biased toward the majority class.

Additionally, to further increase dataset variability, small random **noise** was added to each feature. This approach enhances generalization by exposing the model to slight variations in the data. The noise was generated from a normal distribution with a mean of zero and a standard deviation of 0.1, which added a subtle amount of variability without distorting the underlying patterns.

3.3. Feature engineering

Feature engineering involves creating new features from existing ones to enhance the performance of machine learning models.

Standardization

To standardize the numeric features, a StandardScaler was applied to ensure that all features had a mean of zero and a standard deviation of one. This process helps normalize the data, allowing the machine learning algorithms to work with consistent feature scales.

Statistical Aggregations

To create new features, statistical aggregations were computed. A function was designed to calculate the mean, median, and standard deviation for each data point, adding these as new features. This approach provides additional insights into the data's distribution and can improve model accuracy.

Interaction Terms and Log Transformation

PolynomialFeatures was used to generate interaction terms between existing features. This technique creates new features representing the interaction between original features, allowing the models to capture more complex relationships. The PolynomialFeatures module was configured to generate second-degree interactions without including bias terms.

A log transformation was applied using a FunctionTransformer to handle skewed data distributions. This transformation helps to reduce the impact of large values, providing a more even distribution of data. The log transformation applied np.log1p, a common approach to ensure non-negative transformations.

Feature Expansion Pipeline

A feature engineering pipeline was created to streamline the transformation process. The pipeline included interaction terms, log transformation, statistical aggregations, and standardization. This pipeline was applied to both the resampled training set(.fit_transform) and the testing set(.transform) to maintain consistency across the data.

Handling Infinities and NaNs

A function was created to replace infinities with NaN and then convert those values to zero. This step addressed any issues introduced during feature engineering or scaling. This cleaning process was applied to both the resampled and testing datasets.

3.4. Feature dimensionality adjustment

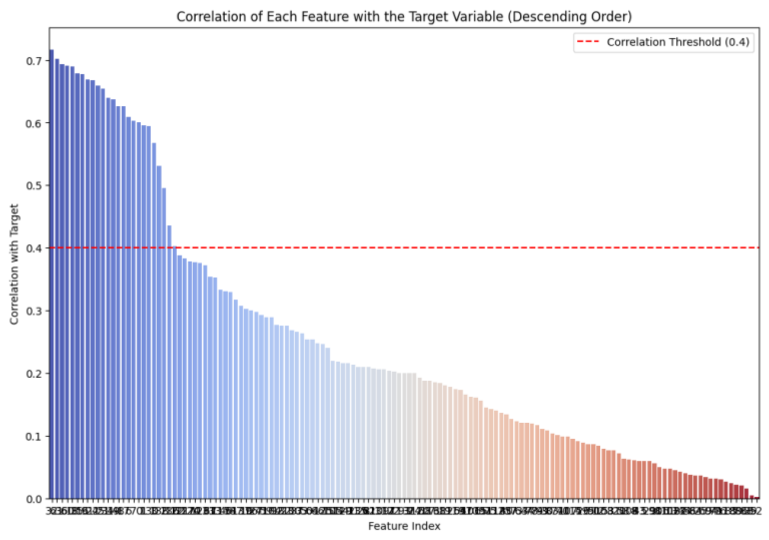
This section discusses how feature selection was performed using Pearson correlation, followed by the application of Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) to further refine the feature set.

Pearson Correlation-Based Feature Selection

To select the most relevant features, the Pearson correlation coefficient was used to determine the relationship between each feature and the target variable. This approach provided a measure of linear correlation, allowing for the identification of features with a strong relationship to the target.

- **Calculation of Pearson Correlations**: The Pearson correlation coefficient was calculated for each feature in the dataset against the target variable using pearsonr from scipy. The correlations were sorted in descending order to identify the most influential features.
- **Threshold-Based Selection**: A threshold of 0.4 was applied to select features with a correlation value exceeding this limit. This threshold was chosen to focus on features with a strong relationship to the target, reducing the influence of weaker correlations.

A bar plot was created to visualize the correlation values, providing a clear view of the most significant features. A threshold line was added to indicate the correlation threshold used for feature selection.



Application of LDA and PCA

Following Pearson correlation-based feature selection, LDA or PCA were applied to further reduce dimensionality and enhance model performance. Different values of n_components were used for both PCA and LDA and the results were compared and analyze.

This approach serves multiple purposes:

- **Efficiency Improvement**: By reducing the feature set, computational costs associated with training machine learning models decrease.
- **Performance Enhancement**: Techniques like LDA and PCA focus on the most relevant information, potentially improving model accuracy and mitigating overfitting.
- **Consistent Model Comparison**: Applying LDA and PCA to every model ensures a consistent basis for comparison across different models. This enables a clear assessment of how dimensionality reduction affects model performance.

3.5. Training, classification, and model selection

The project employed five models, applying Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) to each to evaluate performance differences. These models were trained using 5-fold cross-validation, allowing for validation during the training process. Training and validation accuracies were recorded for each fold to monitor the model's performance. These results were then plotted to visualize accuracy trends over the epochs.

- **Training Accuracy**: Indicates the model's accuracy on the training set.
- **Validation Accuracy**: Measures accuracy on the validation set during cross-validation. After cross-validation, each model was re-trained with the best parameters and then evaluated on the test set. The following metrics were calculated to assess the model's performance:
- **Test Accuracy**: Accuracy on the test set after final model training.
- **Confusion Matrix**: A matrix showing the classifier's predictions against the actual class labels, indicating where the model performed well and where it had difficulties.
- **Macro and Micro F1 Scores**: These scores provide a measure of precision and recall, with macro averaging across all classes and micro considering each instance's contribution.

3. 5. 1 Random Classifier

The random classifier served as a trivial model; it is a simple classification system that assigns class labels randomly to new data points based on the proportions of each class observed in the training data.

- **Training**:

The system is trained on a dataset that contains examples from multiple classes (S0, S1, ..., S6). For each class Si, the system counts the number of training data points labeled with that class, denoted as Ni. It then calculates the total number of training data points, denoted as N.

- **Classification**:

When presented with a new data point to classify, the system randomly selects a class label based on the probabilities calculated from the training data. Probability of selecting class Si as the label for the new data point is given by Ni/N. The system repeats this process for each new data point, independently assigning class labels based on the observed proportions in the training data.

In essence, this "Trivial Solution" does not make any use of the features of the data points or attempt to learn any patterns or relationships between the features and the class labels. It simply mimics the class distribution observed in the training data by assigning class labels randomly according to the observed proportions. As a result, its classification decisions are entirely arbitrary and do not reflect any underlying structure in the data. And hence a good comparison model to other models to see how the feature analysis improves performance.

Interpretation of Results

The Random Classifier demonstrated limited accuracy (~14%), as expected from a random model. However, the trivial model provided a useful baseline to compare against more complex models, helping to highlight the potential improvements gained from other classifiers. Overall, the results from the trivial system demonstrate that dimensionality reduction, whether through PCA or LDA, impacts both training and test accuracy. Reducing the number of components can lead to improved test accuracy, indicating better generalization.

PCA:

n_components	Train accuracy	Test accuracy
24	13.82	13.07
12	14.51	13.65
6	14.76	15.65

LDA:

n_components	Train accuracy	Test accuracy
6	14.37	14.54
4	14.17	13.22
2	13.6	15.9

3. 5. 2 Nearest Means Classifier

The Nearest Means Classifier is a simple machine learning algorithm that predicts the class of a sample based on the nearest mean (centroid) among the classes.

- **Model**: The Nearest Means Classifier calculates the mean (centroid) of each class and uses it as a reference point for prediction. During prediction, the class with the closest mean (Euclidean distance) to the sample is chosen.
- **Training**: The classifier was trained on the transformed data, with each class's mean calculated from the training set. The model has no significant adjustable parameters, relying on the class means for prediction.

Interpretation of Results

The results from PCA and LDA suggest that while reducing the number of components may decrease overfitting, it can also lower test accuracy if reduced too much. This finding implies that model selection must carefully consider the optimal number of components for effective accuracy and generalization.

PCA:

n_components	Train accuracy	Test accuracy
24	76.02	71.41
12	75.83	71.34
6	74.47	70.20

LDA:

n_components	Train accuracy	Test accuracy
6	88.3	87.65
4	85.72	83.17
2	75.13	74.54

The Nearest Means Classifier demonstrated significantly higher accuracy compared to the Random Classifier, indicating that it could effectively identify patterns among the different classes of dry beans. The results and analysis provide a basis for comparison with more complex models in subsequent sections. The best model chosen here for comparison is the classifier with LDA.

3. 5. 3 Support Vector Machine (SVM)

Support Vector Machines (SVM) are powerful classifiers that aim to find the hyperplane that best separates different classes in a dataset.

Hyperparameter Tuning

To optimize the SVM model, a grid search with cross-validation was employed to identify the best hyperparameters. The following parameter grid was used:

- **C**: [0.1, 1, 10, 100]
- **Kernel**: ['linear', 'rbf']
- **Gamma**: [0.001, 0.01, 0.1, 1]

Grid search allows systematic exploration of different hyperparameter combinations, helping to find the optimal settings for the SVM model.
A plot of accuracy versus epochs was created to visualize the model's performance over time, showing both training and validation scores.

Best Hyperparameters

- The best parameters for PCA-based SVM are 12 components, gamma=0.01, C=100, and kernel='rbf'. This setup provides a high-test accuracy of 91.25% with minimal overfitting.
- The best parameters for LDA-based SVM are 4 components, gamma=0.1, C=1, and kernel='rbf'. This configuration results in a test accuracy of 89.78% with a balanced train-test accuracy gap, suggesting good generalization.

These were then used to re-train the best SVM model before evaluation on the test set.

Interpretation of Results

PCA:

n_components (best parameters)	Train accuracy	Test accuracy
24(gamma=0.1, c=10, rbf)	91.97	88.09
12(gamma=0.01, c=100, rbf)	93.01	91.25
6(gamma=100, c=0.1, rbf)	93.38	70.20

LDA:

n_components,	Train accuracy	Test accuracy
6(gamma=0.1, c=10, rbf)	91.3	89.05
4(gamma=0.1, c=1, rbf)	91.13	89.78
2(gamma=1, c=1, rbf)	78.37	77.72

These results indicate that while PCA-based SVM with 12 components yields the highest accuracy, LDA-based SVM with four components offers a good balance between overfitting and generalization. Therefore, to achieve the highest accuracy on unseen data, SVM with PCA is the best choice for this dataset.

3. 5. 4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple yet effective classification algorithm that predicts the class of a sample based on the majority class among its nearest neighbors.

Hyperparameter Tuning

A grid search with cross-validation was used to tune the hyperparameters for KNN. The following parameter grid was explored:

- **n_neighbors**: [50, 100, 150, 200, 250]
- **Weights**: ['uniform', 'distance']
- **Algorithm**: ['auto', 'ball_tree', 'kd_tree', 'brute']

A plot of accuracy versus epochs revealed high training accuracy and variability in validation accuracy, indicating that KNN can quickly adapt to the training data but may exhibit overfitting tendencies.

Best Hyperparameters

- **PCA-based KNN**: The optimal configuration uses 12 components, algorithm='auto', n_neighbors=50, and weights='distance'. This setup achieves a test accuracy of 89.89%, demonstrating high accuracy and reliability without overfitting.
- **LDA-based KNN**: The best parameters include 6 components, algorithm='auto', n_neighbors=50, and weights='distance'. This configuration leads to a test accuracy of 90.33%, showcasing the highest accuracy among KNN models with the best generalization.

Interpretation of Results

KNN achieved high training accuracy, but the variability in validation accuracy suggests high overfitting. The test accuracy was approximately 89%, lower than expected due to overfitting risks. The variability observed in the validation accuracy indicates that KNN may require additional tuning or regularization to avoid overfitting.

PCA:

n_components (best parameters)	Train accuracy	Test accuracy
24(auto, 50, distance)	100	88.83
12(auto, 50, distance)	100	89.89
6(auto, 50, distance)	100	87.61

LDA:

n_components,	Train accuracy	Test accuracy
6(auto, 50, distance)	100	90.33
4(auto, 50, uniform)	90.46	89.53
2(auto, 100, uniform)	80.84	82.43

For this dataset, KNN with LDA (6 components) is the most suitable approach, yielding the highest test accuracy. If the goal is to maximize test accuracy, this setup should be chosen. Although KNN with PCA (12 components) provides a good result, LDA with 6 components offers a slightly better test accuracy, suggesting that LDA's class-separating capability works better with KNN in this context.

3. 5. 5 Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) is a type of artificial neural network that can capture complex relationships in data. This section describes the application of MLP with Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA), focusing on hyperparameter tuning, training with cross-validation, and performance evaluation.

Hyperparameter Tuning with Grid Search

The following parameter grid was examined:

- **Hidden Layer Sizes**: [(10,), (50,), (100,), (100, 50)]
- **Activation**: ['relu', 'tanh', 'logistic']
- **Solver**: ['adam', 'sgd']
- **Learning Rate**: ['constant', 'adaptive']

Best Hyperparameters

The grid search identified the best hyperparameters for MLP:

- **PCA-based MLP**: The optimal setup is with 12 components, activation='logistic', hidden_layer_sizes=(100, 50), learning_rate='constant', and solver='adam'. This configuration achieves a high test accuracy of 90.66% while maintaining a training accuracy of 93.27%.
- **LDA-based MLP**: The best configuration for this model uses 6 components, activation='logistic', hidden_layer_sizes=(100, 50), learning_rate='adaptive', and solver='adam'. This setup leads to a test accuracy of 90.15% and a training accuracy of 91.28%.

Interpretation of Results

Overall, MLP demonstrated the effectiveness of artificial neural networks in achieving high classification accuracy when combined with dimensionality reduction techniques.

PCA:

n_components (best parameters)	Train accuracy	Test accuracy
24(tanh, (50,), constant, adam)	92.24	91.03
12(logistic, (100,50), constant, adam)	93.27	90.66
6(tanh, (100,), adaptive, adam)	92.82	90.89

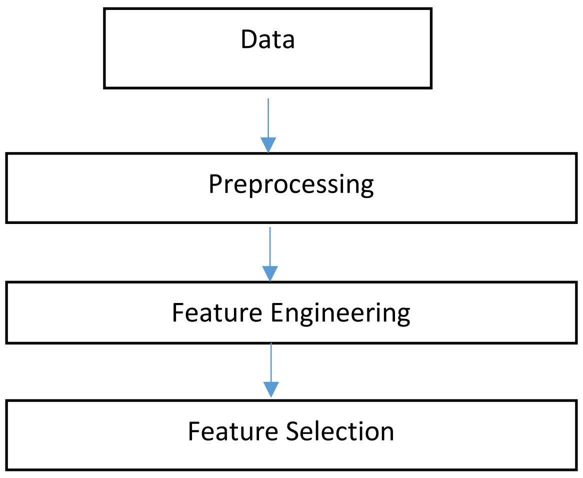
LDA:

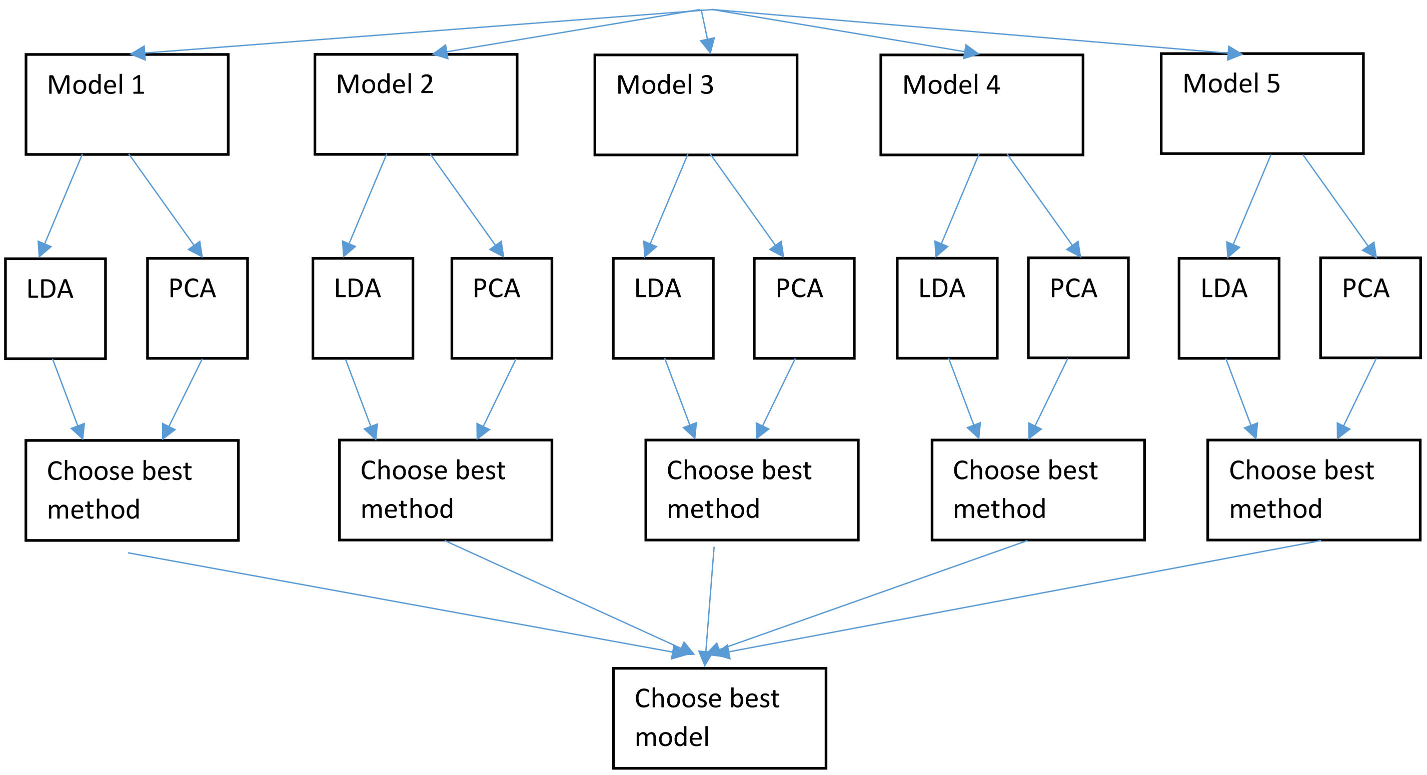
n_components,	Train accuracy	Test accuracy
6(logistic, (100, 50), adaptive, adam)	91.28	90.15
4(relu, (50,), constant, adam)	90.70	89.63
2(relu, (50,), adaptive, adam)	78.47	77.84

When comparing the performance of these models, PCA-based MLP with 12 components demonstrates the best results for this dataset. With a test accuracy of 90.66% and the highest training accuracy, this configuration balances generalization and overfitting control effectively. Thus, PCA seems to be the better dimensionality reduction technique for MLP in this context.

4. Results and Analysis: Comparison and Interpretation

The flowchart illustrates the steps followed in this project and how the various models are compared. It begins with the initial data processing phase followed by feature engineering, and feature selection. After feature selection, five different models are developed, with each model using either Linear Discriminant Analysis (LDA) or Principal Component Analysis (PCA) for dimensionality reduction.
Each of the five models is evaluated with both LDA and PCA, leading to a total of 10 model evaluations. The best-performing method (LDA or PCA) is chosen for each model, and these are further compared to select the overall best model.
This comprehensive process ensures a thorough evaluation and comparison of different machine learning techniques for the dry bean classification task.

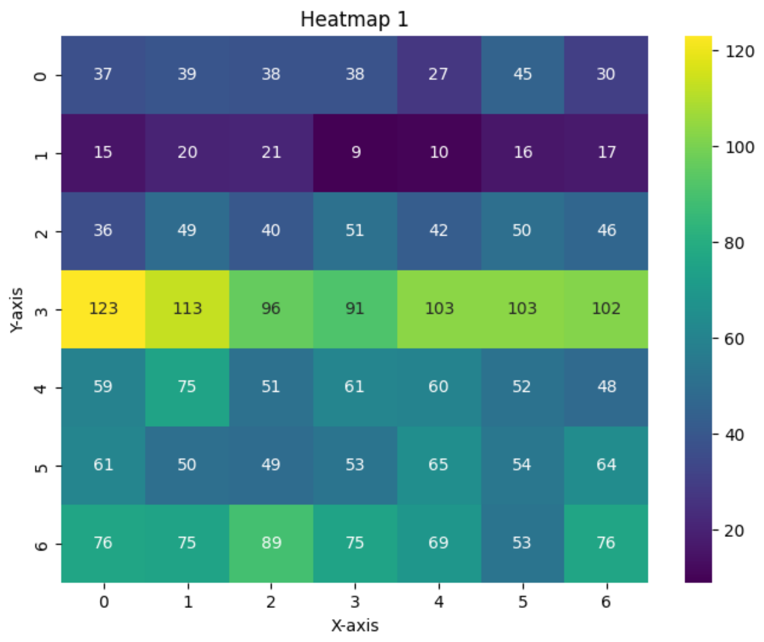




Performance Comparison:

- **Random Classifier with LDA**

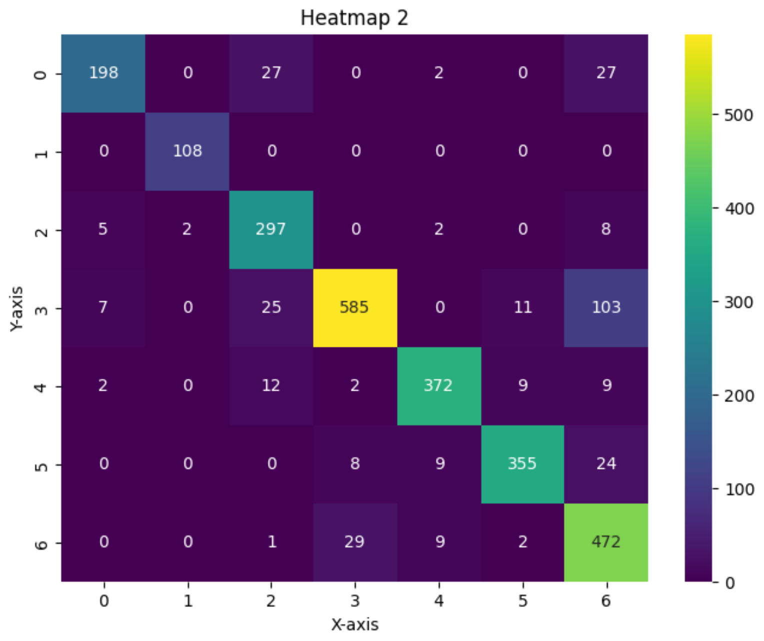
Training Accuracy: 14.46%
Test Accuracy: 13.89%
Training Macro F1: 14.45%
Test Macro F1: 13.28%
Test Micro F1: 13.88%
Confusion Matrix (Test): A broad distribution of predictions, indicating a lack of distinctive accuracy.



	0	1	2	3	4	5	6
0	37	39	38	38	27	45	30
1	15	20	21	9	10	16	17
2	36	49	40	51	42	50	46
3	123	113	96	91	103	103	102
4	59	75	51	61	60	52	48
5	61	50	49	53	65	54	64
6	76	75	89	75	69	53	76

- **Nearest Means Classifier with LDA**

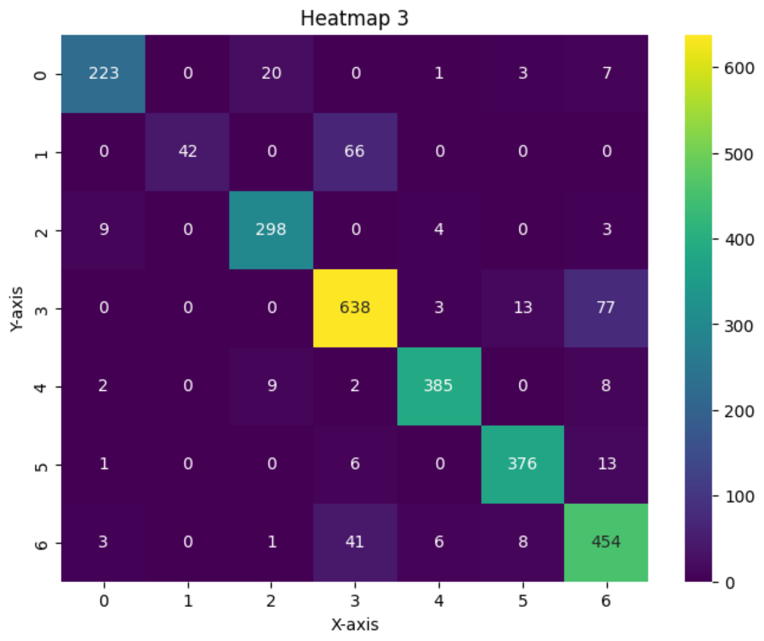
Training Accuracy: 87.83%
Test Accuracy: 87.69%
Training Macro F1: 87.93%
Test Macro F1: 89.26%
Test Micro F1: 87.69%
Confusion Matrix (Test): Significant class distinctions, with accurate predictions for the known classes.



	0	1	2	3	4	5	6
0	198	0	27	0	2	0	27
1	0	108	0	0	0	0	0
2	5	2	297	0	2	0	8
3	7	0	25	585	0	11	103
4	2	0	12	2	372	9	9
5	0	0	0	8	9	355	24
6	0	0	1	29	9	2	472

- **SVM with PCA**

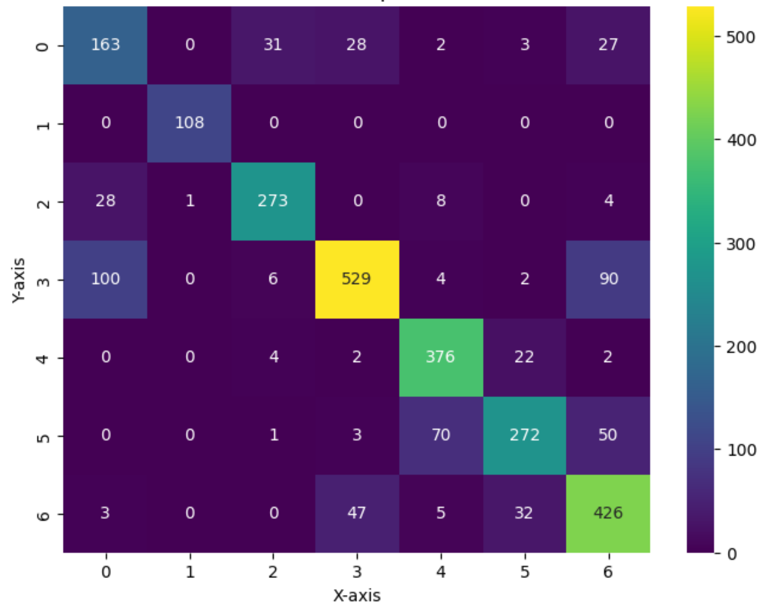
Training Accuracy: 93.56%
Test Accuracy: 88.76%
Training Macro F1: 93.58%
Test Macro F1: 85.72%
Test Micro F1: 88.75%
Confusion Matrix (Test): This classifier has a distinct separation among classes, with high training accuracy indicating some level of overfitting.



- **KNN with LDA**

Training Accuracy: 78.53%
Test Accuracy: 78.88%
Training Macro F1: 78.43%
Test Macro F1: 80.36%
Test Micro F1: 78.87%
Confusion Matrix (Test): The confusion matrix suggests relatively balanced class accuracy with some misclassifications.

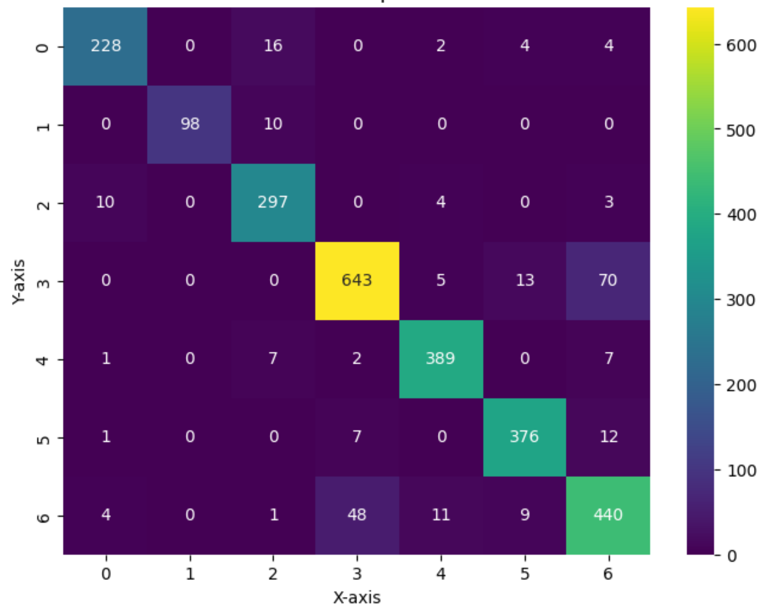
Heatmap 4



- **MLP with PCA**

Training Accuracy: 93.40%
Test Accuracy: 90.78%
Training Macro F1: 93.41%
Test Macro F1: 91.72%
Test Micro F1: 90.77%
Confusion Matrix (Test): This classifier shows good generalization, with fewer misclassifications among known classes.

Heatmap 5



Degrees of Freedom (d.o.f.) and Constraints

The degrees of freedom (d.o.f.) is the number of components used for dimensionality reduction, and the constraint is that Nc > (3 to 10) * d.o.f. Since Nc is 15,619, it satisfies the constraint even with a maximum d.o.f. of 25 (25 * 10 = 250), allowing for more than enough data points to accurately train the model.

Best Performing System

The best-performing system is **MLP** with PCA, which demonstrates both high training accuracy (93.40%) and a closely matched test accuracy (90.78%). This system shows good generalization and a minimal overfitting pattern. The parameters for this model include:

- **Normalization and Preprocessing:** Data was normalized and preprocessed using LabelEncoder, StandardScaler, and SMOTE for balancing class distribution.
- **Feature Set:** PCA with 12 components.
- **Classifier/Regressor Used:** MLP with 'tanh' activation, hidden layer sizes of (50,), 'constant' learning rate, and 'adam' solver.

Analysis and interpretation of results

KNN exhibited overfitting, evidenced by a significant gap between training and test accuracy, which was unexpected. Similarly, SVM showed some degree of overfitting, though not as pronounced as in KNN. The performance of the Nearest Means Classifier, which was moderate, could have provided an early indication of KNN's behavior due to their similarities. As anticipated, the Random Classifier's low accuracy and the Nearest Means Classifier's average performance pointed to their limited predictive capabilities. PCA-based models generally provided better results, while LDA-based models offered a balanced approach to overfitting and generalization.

The overfitting observed in some models was surprising, likely resulting from the combination of LDA/PCA applied alongside feature selection using Pearson's correlation. A better approach could involve using either Pearson or either of LDA and PCA, but not both, to mitigate overfitting after preprocessing.

No additional libraries or methods or systems outside of EE-559 guidelines were used in this project.

5. Libraries used

For this project, the following libraries were used to implement data preprocessing, feature engineering, model training, and evaluation:

- **Pandas**: Used for csv parsing, data manipulation and analysis.
- **Numpy**: Utilized for numerical computations.
- **Matplotlib.pyplot**: Used for plotting and visualization.

- **Seaborn**: Employed for enhanced visualization and plots.
- **Sklearn Preprocessing**:

LabelEncoder: For encoding categorical labels into numerical format.
StandardScaler: To standardize features.
PolynomialFeatures: Used for creating polynomial features.
FunctionTransformer: To transform data.
SimpleImputer: To handle missing values.

- **Sklearn Model Selection**:

KFold: To set up cross-validation.
GridSearchCV: For hyperparameter tuning with grid search.

- **SMOTE** from **imblearn.over_sampling**: This library is used for synthetic minority over-sampling to address class imbalance in the dataset.
- **zscore** and **pearsonr** from **scipy.stats**: These functions are utilized for statistical analysis and calculating correlations between features.
- **Sklearn Decomposition**:

PCA: For Principal Component Analysis.
LDA: For Linear Discriminant Analysis.

- **Sklearn Metrics**:

confusion_matrix, f1_score, accuracy_score, make_scorer: To evaluate model performance.

- **Sklearn Neighbors**:

KNeighborsClassifier: Used for K-Nearest Neighbors.

- **Sklearn SVM**:

svm: To implement Support Vector Machines.

- **Sklearn Neural Network**:

MLPClassifier: To create Multilayer Perceptron (MLP) models.

Custom Code

While most of the methods utilized pre-built functions from these libraries, custom code was developed for specific tasks such as:

- Custom cross-validation loops.
- Data preprocessing and feature engineering tailored to the dataset.
- Specific visualization functions to plot accuracy across epochs.
- Implementation of training and validation workflows.
- Custom evaluation scripts to assess model performance.

6. Summary and conclusions

This project evaluated several machine learning classifiers for classifying dry beans based on a dataset of 13,611 records across seven classes. The selected models were Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Multilayer Perceptron (MLP) using insights from [3], Nearest Means Classifier, and a trivial random classifier, using Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) for dimensionality reduction.

Multilayer Perceptron (MLP) achieved the best results with a test accuracy of 90.78% and minimal overfitting. **Support Vector Machine (SVM)** followed closely, however, it showed signs of overfitting. **K-Nearest Neighbors (KNN)** yielded a test accuracy of 90.33% but exhibited overfitting tendencies due to high variability in validation accuracy. The **Nearest Means Classifier** provided a moderate baseline while the trivial random classifier served as a baseline, achieving a test accuracy of about 14%, indicating its limitations.

To reduce overfitting, future work should consider using only one dimensionality reduction technique. Further hyperparameter tuning and regularization might also help improve model performance. Investigating deep learning approaches[2] could yield even better results. In addition, it's noted from other studies[4] that **Random Forest** and **Naïve-Bayes** classifiers have shown better performances in similar contexts. This information suggests that these models could be valuable to explore in future work.

7. References

[1] "EDA + Outlier / Regression of Class w/ KMeans" 2021. [Kaggle]. Available: https://www.kaggle.com/code/ssyyhh/eda-outlier-regression-of-class-w-kmeans

[2] Taspinar, Y.S., Dogan, M., Cinar, I. et al. Computer vision classification of dry beans (Phaseolus vulgaris L.) based on deep transfer learning techniques. Eur Food Res Technol 248, 2707–2725 (2022). https://doi.org/10.1007/s00217-022-04080-1

[3] Murat Koklu, Ilker Ali Ozkan, Multiclass classification of dry beans using computer vision and machine learning techniques, Computers and Electronics in Agriculture, Volume 174, 2020, 105507, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2020.105507.

[4] "DryBeanClassification by priyanv25" [GitHub]. Available: https://github.com/priyanv25/drybeanclassification/blob/main/drybeansdata.ipynb

[5] “Dry Bean Classification dataset” [Kaggle]. Available: https://www.kaggle.com/datasets/nimapourmoradi/dry-bean-dataset-classification/data

[6] “Nearest-Means-Classifier” [GitHub]. Availbale: https://github.com/ClarenceCHL/Nearest-Means-Classifier

[7] “DiabetesPrediction” [GitHub] Available:
https://github.com/raynaji/DiabetesPrediction/blob/master/DiabetesPrediction_GridSearchCV_LR_SVM_RF.ipynb