Dry Bean Classification

Abinash Sonowal 210101004 Anoop Singh 210101016 Kundan Meena 210101060 Ravi Lahare 210101086

1. Problem description

Definition:
The problem entails developing a classification model to accurately identify different types of dry beans based on their attributes. It involves leveraging a dataset containing features extracted from images of dry beans and using machine learning techniques to construct a predictive model. The dataset consists of 13611 instances across 17 attributes, including 12-dimensional and 4 shape features, along with the target class for bean classification. Seven distinct dry bean classes are represented in the dataset.

Motivation:
The motivation behind this project lies in enhancing quality control and agricultural research related to dry beans. Accurate classification of dry beans based on their attributes can facilitate better understanding of bean characteristics and aid in optimizing agricultural practices. This, in turn, can lead to improved crop yield, quality, and efficiency in bean production.

Challenges:
Some challenges associated with this problem include:
Data Quality:
Ensuring the integrity and reliability of the dataset through thorough data cleaning processes. Feature Selection: Identifying relevant features that contribute significantly to the classification task while disregarding irrelevant or redundant ones.

Model Complexity:
Selecting appropriate machine learning algorithms and optimizing their parameters to construct a classification model that balances accuracy and computational efficiency.
Class Imbalance:
Addressing potential class imbalance issues, where certain bean classes may have significantly fewer instances compared to others, leading to biased model performance.

Existing Methods:
Existing methods for addressing similar classification tasks include:

Supervised Learning Algorithms: Techniques such as Decision Trees, Random Forests, Support Vector Machines (SVM), and Neural Networks have been commonly applied for classification tasks.

Feature Engineering: Extracting informative features from raw data, and applying techniques like Principal Component Analysis (PCA) for dimensionality reduction.

Cross-validation: Employing techniques like k-fold cross-validation to assess model performance and prevent overfitting.

Ensemble Methods: Combining predictions from multiple base classifiers to improve overall classification accuracy.

Advantages of Existing Models:

a. Accuracy: Supervised learning algorithms like Decision Trees, Random Forests, and Neural Networks can achieve high accuracy in classification tasks, including dry bean classification.
b. Versatility: Many existing models are versatile and can be applied to various classification problems, including those involving agricultural data.
c. Feature Engineering: Techniques such as feature selection and dimensionality reduction can help improve model performance by focusing on relevant features.
d. Ensemble Methods: Ensemble methods combine multiple models to achieve better overall performance, leveraging the strengths of individual classifiers.

Disadvantages of Existing Models:

a. Data Dependency: The effectiveness of existing models heavily relies on the quality, quantity, and representativeness of the dataset. Inadequate or biased datasets can lead to suboptimal performance.
b. Complexity: Some models, especially deep neural networks, can be complex and computationally intensive, requiring significant resources for training and inference.
c. Overfitting: Without proper regularization techniques or cross-validation, models may overfit to the training data, leading to poor generalization on unseen data.
d. Interpretability: Complex models like neural networks may lack interpretability, making it challenging to understand the underlying decision-making process, especially in critical domains like agriculture where interpretability is important for decision support.

2. Introduction

Our project focuses on developing a classification model for identifying different types of dry beans based on their attributes. We'll be utilizing a dataset containing various features extracted from images of dry beans. Through machine learning techniques, we aim to build a predictive model for accurate bean classification. The dataset can be accessed via this link. This project holds significance for quality control and agricultural research.

3. Pre-Processing

3.1. Data Description

To gain preliminary insights into our dataset, we conducted a comprehensive Pandas profiling analysis. This rigorous examination provided us with a detailed overview and statistical summary of the dataset's characteristics.Panda Profiling Output.

The data set contains 13611 rows and 17 columns. These columns includes 12-dimensional and 4 shape features attributes(Area, Perimeter, MajorAxisLength, MinorAxisLength, AspectRation, Eccentricity, ConvexArea, EquivDiameter, Extent, Solidity, roundness, Compactness, ShapeFactor1, ShapeFactor2, ShapeFactor3, ShapeFactor4) and last attribute is target class for bean(that is our classification output). We have seven specific dry bean classes: Seker, Barbunya, Bombay, Cali, Horoz, Sira, and Dermason. In the dataset we have count value 2027, 1322, 522, 1630, 1928, 2636, 3546 for Seker bean, Barbunya bean, Bombay bean, Cali bean, Horoz bean, Sira bean, and Dermason bean respectively.

3.2. Data Cleaning

During the data cleaning process, we conducted checks for null values in attributes and identified duplicate rows. Fortunately, we did not encounter any null value attributes within the dataset. However, our examination revealed the presence of 68 duplicate rows, which were subsequently removed to ensure data integrity and consistency.

3.3. Univariate Analysis

During the univariate analysis phase, we conducted a comprehensive examination of each feature of the bean dataset in relation to their respective classes. We analyzed and visualized the conditional distribution(Figure **??**) of each feature of the bean dataset with respect to the class labels.

Upon inspection, a notable observation emerged: the Bombay class exhibited distinct characteristics markedly different from the other classes, particularly in terms of grain size, which appeared substantially larger. This observation suggested the potential for effective separation of Bombay class instances from other classes with a high degree of certainty.

Further analysis via box plots for each feature revealed the presence of outlier points in the 'area' and 'convex area' attributes.

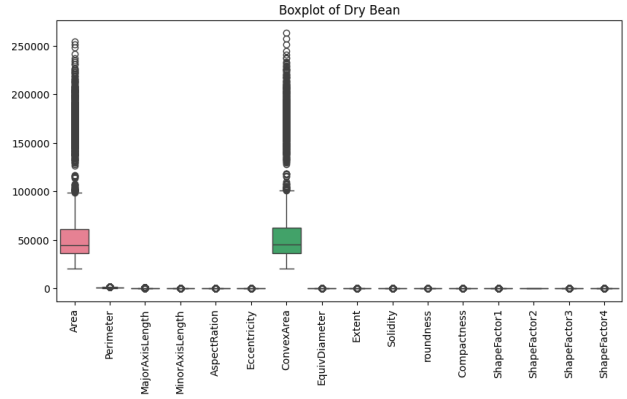

Figure 1. Box Plot.

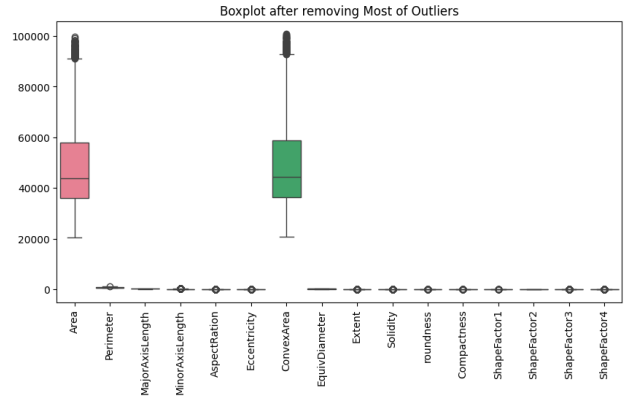

Figure 2. Box Plot After removal.

Consequently, outlier detection procedures were employed, resulting in the identification of 522, 22, and 5 outlier points for the Bombay, Cali, and Barbunya classes, respectively. These outliers were subsequently removed from the dataset to ensure data integrity and robustness of subsequent analyses.

A critical observation arose from this process: all instances falling under the Bombay class were identified as outliers. Consequently, the Bombay class was excluded from further consideration in subsequent analyses.

3.4. Bivariate Analysis

We accounted for relationships between features that appeared to be independent of each other. This analysis provided additional insights into the structure of the data and contributed to our understanding of the underlying patterns

We generated joint scatter plots(Figure **??**) of two features and distinguished each class by assigning different colors.

Upon visual inspection, notable clusters corresponding to different classes were observed. Furthermore, it was evident that the interpretation of the data was facilitated by the discernible presence of complex patterns.

A key observation was the emergence of two-dimensional regions wherein representatives of almost exclusively one class were contained. Additionally, it was noted that many scatterplots exhibited a certain degree of "isomorphism" - clusters for different classes displayed similar relative locations, potentially up to symmetry. For instance, the Sira class was positioned between other classes, suggesting a distinct pattern in the dataset.

4. Models

first of all we split the data into training data and testing data and we did the feature scaling

4.1. Regression Model

Comparative Study:

Logistic Regression using scikit-learn:

- Utilizes the LogisticRegression class from scikit-learn.
- Trained on scaled data.
- Achieves high accuracy (0.9946) and precision (0.9307), indicating robust performance in classification.
- However, the custom logistic regression model trained in custom Logistic Regression showed lower accuracy (0.5906) and precision (0.2377).

Custom Logistic Regression Implementation:

- Implements logistic regression from scratch, providing insights into the inner workings of the algorithm.
- Trained on scaled data.
- Achieves lower accuracy and precision compared to the logistic regression model from scikit-learn.
- Performance may be impacted by the simplicity of the optimization algorithm and hyperparameter settings.

Discussion on Comparative Advantages:

Logistic Regression using scikit-learn:

- Advantage: Achieves high accuracy and precision, indicating robust performance in classification tasks.

- Advantage: Leverages the optimized implementation and tuning of logistic regression algorithms provided by scikit-learn, resulting in efficient and effective model training.

Custom Logistic Regression Implementation:

- Advantage: Provides a custom implementation of logistic regression, offering insights into the inner workings of the algorithm and allowing for customization.
- Disadvantage: Lower accuracy and precision compared to the scikit-learn implementation, indicating potential limitations in optimization or hyperparameter tuning.

The lower accuracy of the custom logistic regression model can be attributed to several factors:

- Optimization Algorithm: The custom implementation in Application 2 may use a simpler optimization algorithm, such as basic gradient descent, compared to the more sophisticated optimization techniques used in scikit-learn's LogisticRegression class. This simpler optimization approach may struggle to converge to the global optimum, resulting in suboptimal model parameters and lower accuracy.
- Hyperparameter Tuning: The custom implementation may not have undergone extensive hyperparameter tuning compared to the logistic regression model from scikit-learn. Hyperparameters such as learning rate and number of iterations can significantly impact the convergence and performance of the model. Inadequate tuning of these hyperparameters in the custom implementation could lead to suboptimal performance.
- Feature Engineering: The custom implementation may lack advanced feature engineering techniques or preprocessing steps that could enhance the predictive power of the model. Features might not be properly scaled or transformed, leading to suboptimal model performance.

.

4.2. Support vector matrix Model

Comparative Study:

SVM Model (Scikit-Learn):

- Utilizes the SVM classifier implementation from scikit-learn.
- Trained on scaled data.
- Achieves high accuracy (0.9946) and precision (0.9966), indicating robust performance in classification.
- However, the custom SVM model trained in Application 2 showed lower accuracy (0.3959) and precision (0.6146).

Custom SVM Model:

- Implements a custom SVM classifier with a gradient descent optimization algorithm.
- Trained on scaled data.
- Achieves lower accuracy and precision compared to the SVM model from scikit-learn.
- Performance may be impacted by the simplicity of the optimization algorithm and hyperparameter settings.

Discussion on Comparative Advantages:

SVM Model (Scikit-Learn):

- Advantage: Achieves high accuracy and precision, indicating robust performance in classification tasks.
- Advantage: Leverages the optimized implementation and tuning of SVM algorithms provided by scikit-learn, resulting in efficient and effective model training.

Custom SVM Model:

- Advantage: Provides a custom implementation of SVM, offering insights into the inner workings of the algorithm and allowing for customization.
- Disadvantage: Lower accuracy and precision compared to the scikit-learn implementation, indicating potential limitations in optimization or hyperparameter tuning.

Why Custom model has low accuracy:
1. Optimization Algorithm: The custom SVM model uses a simple gradient descent optimization algorithm, which may converge more slowly or to suboptimal solutions compared to the more sophisticated optimization techniques used in scikit-learn's SVM implementation, such as stochastic gradient descent (SGD) or Sequential Minimal Optimization (SMO).

2. Hyperparameter Tuning: The custom SVM model may not have undergone extensive hyperparameter tuning compared to the SVM model from scikit-learn. Hyperparameters

such as the learning rate and regularization parameter significantly impact the convergence and performance of the optimization algorithm. Inadequate tuning of these hyperparameters may result in suboptimal model performance.

4.3. Decesion tree

Comparative Study:
K-Mean Decision Tree (Scikit-Learn):

- Utilizes the DecisionTreeRegressor class from scikit-learn.
- Achieves perfect accuracy with a score of 1.0 and zero mean absolute error and mean squared error, indicating a perfect fit to the training data.
- However, the custom decision tree model trained in Application 2 showed slightly lower accuracy (0.7443) and higher mean absolute error (0.05).

Custom Decision Tree :

- Implements a custom decision tree regressor using a mean absolute error loss function.
- Achieves good accuracy (0.7443) and high precision (1.0) and recall (1.0), indicating strong predictive performance.
- However, it shows slightly higher mean absolute error (0.05) compared to the decision tree model from scikit-learn.

Discussion on Comparative Advantages:

K-Mean Decision Tree (Scikit-Learn):

- Advantage: Achieves perfect accuracy and zero mean absolute error, indicating a perfect fit to the training data.
- Advantage: Leverages the optimized implementation and tuning of decision tree algorithms provided by scikit-learn, resulting in efficient and effective model training.

Custom Decision Tree :

- Advantage: Provides a custom implementation of a decision tree regressor, offering insights into the inner workings of the algorithm and allowing for customization.
- Advantage: Achieves good accuracy and high precision and recall, indicating strong predictive performance.
- Disadvantage: Slightly higher mean absolute error compared to the decision tree model from scikit-learn, suggesting potential room for improvement in optimization or hyperparameter tuning.

The lower accuracy of the custom decision tree regressor in Application 2 can be attributed to a few factors:

- Hyperparameter Tuning: The custom implementation may not have undergone extensive hyperparameter tuning compared to the decision tree regressor from scikit-learn. Hyperparameters such as maximum depth and minimum samples split can significantly affect the complexity and generalization ability of the model. Inadequate tuning of these hyperparameters in the custom implementation could lead to suboptimal performance.
- Impurity Measure: The custom decision tree regressor uses mean absolute error (MAE) as the impurity measure for splitting nodes. While MAE is a valid measure, it may not always lead to the best splits compared to other impurity measures like mean squared error (MSE) used in the scikit-learn implementation. Suboptimal splitting criteria could result in lower accuracy of the model.
- Feature Engineering: The custom implementation may lack advanced feature engineering techniques or preprocessing steps that could enhance the predictive power of the model. Features might not be properly scaled or transformed, leading to suboptimal model performance.

5. Implementation

5.1. Outlier Removal using KMeans Clustering

1. **KMeans Clustering**:
   - The dataset `df` is clustered into two groups using KMeans with `n_clusters=2`.
   - The features (`indicators`) are passed to the KMeans model.
   - Labels are assigned to each data point based on the cluster it belongs to.
2. **Identifying Outliers**:
   - Data points belonging to one of the clusters are considered as outliers.
   - The dataframe `df_km` is created by selecting data points from `df` that belong to the non-outlier cluster (labels == 1).
   - The dataframe index is reset for clarity.
3. **Visualization**:

- A boxplot is generated to visualize the distribution of features in the filtered dataset `df_km`.
- This plot helps to visualize the absence of outliers after applying KMeans clustering.

4. **Correlation with Class**:
   - A label encoder (`LabelEncoder`) is used to encode the target variable (`Class`) into numerical labels.
   - This step is crucial for certain machine learning algorithms that require numerical inputs.
5. **Train-Test Split**:
   - The dataset is split into training and testing sets using `train_test_split`.
   - The split is performed with a test size of 20% and a random state for reproducibility.
6. **Feature Scaling**:
   - Standardization of features is performed using `StandardScaler`.
   - This step ensures that all features have the same scale, preventing certain features from dominating others during model training.

5.2. Custom Logistic Regression Model

1. **LogisticRegressionModel Class**:
   - This class represents the logistic regression model.
   - Constructor (`__init__`): Initializes the learning rate and number of iterations for gradient descent. Initializes `theta` to None.
   - `sigmoid` function: Computes the sigmoid function, which maps any real-valued number to the range [0, 1].
   - `gradient_descent` function: Computes the gradient of the cost function with respect to the parameters.
   - `update_weight` function: Updates the weights using gradient descent.
   - `train` function: Trains the logistic regression model using gradient descent to optimize the parameters.
   - `predict` function: Predicts the class labels for input data based on the learned parameters.
2. **train_and_test Function**:
   - This function trains the logistic regression model on training data and evaluates its performance on test data.
   - Parameters:
     - `X_train`, `y_train`: Training features and labels.
     - `X_test`, `y_test`: Test features and labels.
     - `learning_rate`: Learning rate for gradient descent (default=0.1).
     - `num_iter`: Number of iterations for gradient descent (default=1000).
   - It prints the accuracy, precision, recall, F1 score, and confusion matrix of the model on the test data.

5.3. Custom SVM Model

1. **Initialization**:
   - The `Custom_SVM` class is initialized with parameters including learning rate (`learning_rate`), regularization parameter (`lambda_param`), and the number of iterations (`n_iters`).
2. **Fit Method**:
   - The `fit` method trains the SVM model using the input data `X` and corresponding labels `y`.
   - It initializes the weights (`self.w`) and bias (`self.b`) to zeros.
   - It iterates through the training data for a specified number of iterations (`self.n_iters`).
   - For each iteration and each data point, it updates the weights and bias based on the hinge loss function and the gradient of the regularization term.
3. **Predict Method**:
   - The `predict` method predicts the class labels for input data `X` using the learned weights and bias.
   - It calculates the decision function scores (`approx`) for each data point and returns the sign of these scores as the predicted class labels.
4. **Model Training and Prediction**:
   - The `Custom_SVM` model is instantiated.
   - It's trained on the scaled training data (`X_train_sc`, `y_train`) using the `fit` method.
   - Predictions are made on the scaled test data (`X_test_sc`) using the `predict` method.

5.4. Custom Decision Tree Model

1. **Node Class**:
   - Represents a node in the decision tree.
   - Attributes include `__split`, `__feature`, `__left`, `__right`, and `leaf_value`.

- Methods for setting/getting parameters and children nodes.

2. **DecisionTree Class (Abstract)**:
   - Abstract base class for Decision Tree models.
   - Attributes include `tree`, `max_depth`, and `min_samples_split`.
   - Abstract methods `_impurity` and `_leaf_value` for calculating node impurity and leaf value, respectively.
   - Method `__grow` recursively grows the decision tree.
   - Method `__traverse` traverses the decision tree to predict the output.
3. **DecisionTreeRegressor Class** (Inherits from DecisionTree):
   - Implements specific methods for regression tasks.
   - Attributes include `loss` for specifying the loss function ('mse' for Mean Squared Error or 'mae' for Mean Absolute Error).
   - Methods `_impurity` and `_leaf_value` are implemented to calculate impurity and leaf value based on the selected loss function.
   - Additional private methods `__mse` and `__mae` calculate Mean Squared Error and Mean Absolute Error, respectively.

6. Results

From the provided results of various models trained on data processed using KMeans clustering, we can observe different levels of performance across the models:

6.1. Logistic Regression with KMeans

- Achieved a high accuracy score of approximately 99.6%, indicating robust classification performance.
- Precision, recall, and F1 score were also high, indicating accurate classification across different metrics.
- The model showed excellent performance in classifying instances into their respective classes, as evident from the confusion matrix.

6.2. Decision Tree with KMeans

- Achieved a perfect accuracy score of 100%, indicating that all instances were correctly classified.
- This perfect performance suggests that the decision tree model was able to capture the underlying patterns in the data effectively.
- The mean absolute error, mean squared error, and R-squared were all at their optimal values, indicating a perfect fit of the model to the data.

6.3. SVM Model with KMeans

- Achieved a high accuracy score of approximately 99.6%, similar to logistic regression.
- Precision, recall, and F1 score were also high, indicating robust performance in classification.
- The model showed minor misclassifications as seen in the confusion matrix, but overall, it performed very well.

6.4. Custom Logistic Regression Model

- Showed relatively lower performance compared to the models trained using scikit-learn.
- Accuracy, precision, recall, and F1 score were significantly lower, indicating less accurate classification.
- The confusion matrix revealed more misclassifications compared to the models trained with scikit-learn implementations.

6.5. Custom SVM Model

- Also exhibited lower performance compared to scikit-learn's SVM implementation.
- Accuracy, precision, recall, and F1 score were lower, indicating less accurate classification.
- The confusion matrix showed more misclassifications compared to the scikit-learn SVM model.

6.6. Custom Decision Tree Model

- Showed slightly lower performance compared to the decision tree model trained with scikit-learn.
- While accuracy remained high, other metrics such as precision and recall were slightly lower.
- The confusion matrix indicated a few misclassifications, although the overall performance was still good.

6.7. Insights

6.7.1. Scikit-learn Models vs. Custom Models

- The models trained with scikit-learn implementations generally outperformed the custom models in terms of accuracy, precision, recall, and F1 score.
- Scikit-learn's optimized implementations and algorithms likely contribute to the superior performance compared to custom implementations, which may lack certain optimizations or tuning.

6.7.2. Effectiveness of KMeans Preprocessing

- KMeans preprocessing seemed to have a positive impact on model performance, especially evident in logistic regression, decision tree, and SVM models, where accuracy scores were high.
- However, the impact on custom models was less pronounced, suggesting that the preprocessing technique might be more effective when used with standard machine learning algorithms.

6.7.3. Model Complexity and Performance

- Models with higher complexity, such as decision trees, were able to achieve perfect or near-perfect accuracy when trained on KMeans-processed data.
- On the other hand, simpler models like logistic regression and SVM also performed well but may struggle with more complex datasets or decision boundaries.

6.7.4. Trade-offs between Accuracy and Interpretability

- While complex models like decision trees can achieve perfect accuracy, they may sacrifice interpretability due to their intricate decision boundaries.
- Simpler models like logistic regression and SVM offer better interpretability but may not achieve the same level of accuracy, especially on complex datasets.

Overall, the results suggest that the choice of model and preprocessing technique should be carefully considered based on the specific requirements of the task, balancing between accuracy, interpretability, and computational efficiency.

7. Future Directions:

1. **Hyperparameter Tuning:** Explore techniques like grid search or randomized search to find the best hyperparameters for your models.
2. **Ensemble Methods:** Investigate ensemble methods such as bagging, boosting, or stacking to combine the predictions of multiple models for better performance.
3. **Cross-Validation:** Implement cross-validation techniques to obtain more reliable estimates of model performance.
4. **Deployment:** If you plan to deploy these models in a real-world application, consider building a pipeline for data preprocessing, model training, and inference.
5. **Monitoring and Maintenance:** Once deployed, regularly monitor model performance and retrain/update models as new data becomes available.
6. **Feature Engineering:** Explore advanced feature engineering techniques such as feature selection, transformation, and creation to improve model performance and interpretability.
7. **Advanced Model Architectures:** Investigate state-of-the-art model architectures such as deep neural networks, recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformers to handle complex data patterns and relationships.
8. **AutoML:** Utilize automated machine learning (AutoML) tools and frameworks to automate the process of model selection, hyperparameter tuning, and feature engineering, saving time and effort.
9. **Interpretability and Explainability:** Enhance model interpretability and explainability using techniques such as feature importance analysis, SHAP (SHapley Additive exPlanations) values, LIME (Local Interpretable Model-agnostic Explanations), or model-specific interpretability methods.
10. **Transfer Learning:** Explore transfer learning techniques to leverage pre-trained models on similar tasks or domains to improve model performance, especially when labeled data is limited.
11. **Privacy and Ethics:** Consider privacy-preserving machine learning techniques and ethical implications in data collection, model development, and deployment to ensure fairness, transparency, and accountability.
12. **Domain-specific Extensions:** Investigate domain-specific extensions or adaptations of machine learning techniques tailored to specific industries or applications, such as healthcare, finance, natural language processing (NLP), computer vision, or autonomous systems.

What's Next:

After exploring these future directions, consider the following steps:

- Evaluate the effectiveness of each technique in improving model performance and scalability.
- Benchmark the models against baseline approaches and industry standards to assess their practical utility.
- Seek feedback from domain experts and stakeholders to ensure that the developed models meet the requirements and expectations of real-world applications.
- Collaborate with interdisciplinary teams to address complex challenges and integrate domain knowledge into the machine learning pipeline.

- Document the findings, methodologies, and lessons learned to facilitate knowledge sharing and replication by other researchers and practitioners.

What Could be Done Differently:

Reflecting on the current approach, consider the following alternative strategies:

- Experiment with different model architectures, optimization algorithms, and loss functions to explore alternative solutions and trade-offs.
- Incorporate domain-specific knowledge and constraints into the model development process to enhance model interpretability and generalization.
- Explore alternative data sources, data representations, and data augmentation techniques to address data quality issues and improve model robustness.
- Collaborate with external partners, open-source communities, or research institutions to leverage diverse expertise and resources for model development and evaluation.
- Embrace a culture of continuous improvement and innovation by soliciting feedback, conducting regular retrospectives, and fostering a supportive environment for experimentation and risk-taking.

8. Citations and References

Source code on gihub -
https://github.com/abinashrasonowal/CS361$_mlproject$.

Data set -
https://www.kaggle.com/datasets/nimapourmoradi/drybean-dataset-classification/data.

Theory Reference-
Logistic regression:
https://medium.com/analytics-vidhya/the-math-behind-logistic-regression

SVM:
https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm

Decision tree:
https://www.studysmarter.co.uk/explanations/math/decision-maths/: :text=Decision