**Dry Bean Classification Project: Report**

**1. Introduction**

This report outlines a machine learning classification project using the Dry Bean Dataset. The workflow includes data preprocessing, feature engineering, model training, and performance evaluation. Various algorithms were tested to identify the best method for classifying bean varieties.

**2. Project Implementation**

**2.1 Data Preprocessing**

The preprocessing pipeline included the following steps:

1. **Handling missing values**:
   * Added 5% missing values to two randomly selected columns
   * Added 35% missing values to one column
   * Applied mean/median imputation for columns with 5% missing values
   * Dropped the column with 35% missing values
2. **Outlier detection and treatment**:
   * Used the IQR method to identify outliers
   * Replaced outliers with boundary values
3. **Feature scaling**:
   * Applied StandardScaler to normalize numerical features
   * Ensured all features had comparable scales for optimal model performance
4. **Categorical encoding**:
   * Used LabelEncoder to convert class labels to numeric form

Sample visualizations from the preprocessing step:

A group of blue boxes

AI-generated content may be incorrect.A group of blue and white graphs

AI-generated content may be incorrect.A diagram of a variety of colors

AI-generated content may be incorrect.

**2.2 Feature Engineering**

Three data representations were created for comparison:

1. **Raw data**: Preprocessed data with the original features (minus the dropped column)
2. **PCA-transformed data**: Dimensionality reduction using Principal Component Analysis
3. **LDA-transformed data**: Dimensionality reduction using Linear Discriminant Analysis

**PCA Components**

PCA was used to reduce dimensionality while preserving as much variance as possible:

A diagram of different colored dots

AI-generated content may be incorrect.A graph of colored dots

AI-generated content may be incorrect.

**LDA Components**

LDA was used to create a lower-dimensional representation while maximizing class separation:

A diagram of a diagram of a number of discriminants

AI-generated content may be incorrect.A graph of colored dots

AI-generated content may be incorrect.

**2.3 Model Training and Evaluation**

Five classification algorithms were implemented and evaluated across the three data representations:

1. **Logistic Regression**
2. **Decision Tree**
3. **Random Forest**
4. **XGBoost**
5. **Naive Bayes**

Nested cross-validation was used for robust evaluation:

* Outer loop: 5-fold CV for performance evaluation
* Inner loop: 3-fold CV for hyperparameter tuning

**3. Results and Analysis**

**3.1 Performance Metrics**

The table below summarizes the performance metrics for each model and data representation combination:

| **Classifier** | **Data Representation** | **Accuracy (Mean)** | **Precision (Mean)** | **Recall (Mean)** | **F1 Score (Mean)** |
| --- | --- | --- | --- | --- | --- |
| Logistic Regression | raw | 0.9204 | 0.9210 | 0.9204 | 0.9206 |
| Logistic Regression | pca | 0.8876 | 0.8895 | 0.8876 | 0.8882 |
| Logistic Regression | lda | 0.8390 | 0.8453 | 0.8390 | 0.8397 |
| Decision Tree | raw | 0.9008 | 0.9014 | 0.9008 | 0.9010 |
| Decision Tree | pca | 0.8694 | 0.8728 | 0.8694 | 0.8706 |
| Decision Tree | lda | 0.8561 | 0.8604 | 0.8561 | 0.8574 |
| Random Forest | raw | 0.9183 | 0.9187 | 0.9183 | 0.9185 |
| Random Forest | pca | 0.8882 | 0.8895 | 0.8882 | 0.8887 |
| Random Forest | lda | 0.8774 | 0.8799 | 0.8774 | 0.8784 |
| XGBoost | raw | 0.9195 | 0.9199 | 0.9195 | 0.9197 |
| XGBoost | pca | 0.8913 | 0.8925 | 0.8913 | 0.8918 |
| XGBoost | lda | 0.8765 | 0.8786 | 0.8765 | 0.8773 |
| Naive Bayes | raw | 0.8928 | 0.8966 | 0.8928 | 0.8938 |
| Naive Bayes | pca | 0.8768 | 0.8803 | 0.8768 | 0.8779 |
| Naive Bayes | lda | 0.8261 | 0.8324 | 0.8261 | 0.8273 |

**3.2 Performance Visualizations**

The comparative performance of different models across data representations is illustrated in the following figures:

A graph of different colored bars

AI-generated content may be incorrect.A graph of different colored bars

AI-generated content may be incorrect.

A screenshot of a chart

AI-generated content may be incorrect.

These visualizations clearly show that:

* Raw data consistently outperformed both PCA and LDA across all classifiers
* For each classifier, the performance ranking was consistently: Raw > PCA > LDA
* Logistic Regression, XGBoost, and Random Forest were the top-performing classifiers

**3.3 Confusion Matrix Analysis**

Confusion matrices provide insights into classification patterns and error types for each model:

**Decision Tree Confusion Matrices**

A graph of blue squares with numbers and labels

AI-generated content may be incorrect.A graph of blue squares with numbers

AI-generated content may be incorrect.

**A graph of a graph with numbers and a number

AI-generated content may be incorrect.**

**Logistic Regression Confusion Matrices**

A graph of blue squares with numbers

AI-generated content may be incorrect.A graph of a graph with numbers and labels

AI-generated content may be incorrect.A graph of a graph with numbers and labels

AI-generated content may be incorrect.

**Naive Bayes Confusion Matrices**

A graph of blue squares with numbers

AI-generated content may be incorrect.A graph of blue squares

AI-generated content may be incorrect.A graph of a graph with numbers and a bar chart

AI-generated content may be incorrect.

**Random Forest Confusion Matrices**

A graph of a graph with numbers and labels

AI-generated content may be incorrect.A graph of a graph with numbers and labels

AI-generated content may be incorrect.A graph of a graph

AI-generated content may be incorrect.

**XGBoost Confusion Matrices**

A graph of blue squares with numbers and labels

AI-generated content may be incorrect.A graph of blue squares with numbers

AI-generated content may be incorrect.A graph of a graph with numbers and a number

AI-generated content may be incorrect.

Key observations from the confusion matrices:

1. **Class separability**:
   * Class 3 (represented by index 3) shows the highest classification accuracy across all models
   * Classes 0 and 2 show some degree of confusion in most models
   * Class 5 is often confused with Class 3, particularly in LDA-based models
2. **Model-specific patterns**:
   * Logistic Regression with raw data shows the most balanced confusion matrix with the fewest misclassifications
   * Naive Bayes consistently shows more confusion between classes compared to other models
   * LDA-based models generally show more misclassifications than PCA or raw data models

**3.4 ROC Curve Analysis**

The ROC curves illustrate the diagnostic ability of classifiers at various thresholds. Since this is a multi-class problem, ROC curves were generated using a one-vs-all approach:

**Decision Tree ROC Curves**

A graph of a line graph

AI-generated content may be incorrect.A graph of a line

AI-generated content may be incorrect.A graph of a tree

AI-generated content may be incorrect.

**Logistic Regression ROC Curves**

A graph of a logistic regression

AI-generated content may be incorrect.A graph of a logistic regression

AI-generated content may be incorrect.A graph of a logistic regression

AI-generated content may be incorrect.

**Naive Bayes ROC Curves**

A graph of a line

AI-generated content may be incorrect.A graph of a line

AI-generated content may be incorrect.A graph of a function

AI-generated content may be incorrect.

**Random Forest ROC Curves**

A graph of a curve

AI-generated content may be incorrect.A graph of a graph

AI-generated content may be incorrect.A graph of a function

AI-generated content may be incorrect.

**XGBoost ROC Curves**

A graph of a line graph

AI-generated content may be incorrect.A graph of a line

AI-generated content may be incorrect.A graph of a function

AI-generated content may be incorrect.

Key observations from the ROC curves:

1. **Class-wise performance**:
   * Most classes show high AUC values (>0.95) across all models, indicating strong classification performance
   * Classes 1 and 3 consistently show the highest AUC values (near 1.0), indicating excellent separation
   * Classes 0 and 2 typically have slightly lower AUC values, aligning with the confusion seen in the confusion matrices
2. **Data representation comparison**:
   * Models trained on raw data consistently show higher AUC values compared to PCA and LDA
   * The gap in AUC values between data representations is most noticeable for Naive Bayes
   * Logistic Regression on raw data achieves the most balanced high AUC values across all classes
3. **Classifier comparison**:
   * Logistic Regression, Random Forest, and XGBoost show comparable ROC performance
   * Decision Tree models have slightly lower AUC values compared to the other models
   * Naive Bayes shows the most variability in AUC values across different classes

**4. Best Performing Model: Logistic Regression on Raw Data**

The best performing model was **Logistic Regression using raw data representation**, with the following metrics:

* Accuracy: 92.04%
* F1 Score: 92.06%
* Precision: 92.10%
* Recall: 92.04%

**4.1 Why Logistic Regression Performed Best**

1. **Feature Quality**: The raw data representation preserved all the relevant information for classification, while dimensionality reduction techniques (PCA and LDA) likely discarded some discriminative information.
2. **Model Suitability**: Logistic Regression works particularly well with:
   * Well-scaled features (achieved through StandardScaler in preprocessing)
   * Linearly separable classes (the bean varieties appear to be reasonably separable in the feature space)
   * Balanced datasets (the class distribution in this dataset is fairly balanced)
3. **Feature Relationships**: The feature correlation matrix shows moderate correlations between features and the target variable, providing good discriminative power for the Logistic Regression model.
4. **Generalization Capability**: Logistic Regression's ability to avoid overfitting, combined with the nested cross-validation approach, resulted in a model with excellent generalization performance.

**4.2 Limitations of Dimensionality Reduction**

The consistent performance drop observed with PCA and LDA representations suggests:

1. **Information Loss**: Both PCA and LDA discard some information that appears to be relevant for classification. While they aim to preserve variance (PCA) or maximize class separation (LDA), they may be eliminating subtle but important discriminative features.
2. **Feature Interactions**: Complex interactions between features might be lost during the transformation process, reducing the models' ability to capture the full complexity of the data.
3. **Bean-specific Characteristics**: Some discriminative features specific to certain bean varieties might be diluted or lost during dimensionality reduction.

**5. Code Implementation**

Key parts of the implementation are provided below:

**5.1 Data Preprocessing**

def handle\_missing\_values(df, missing\_cols):

"""

Handle missing values in the dataset:

- Fill columns with 5% missing values using mean/median

- Drop column/rows with 35% missing values

"""

print("Handling missing values...")

df\_handled = df.copy()

# Display missing values summary

print("Missing values summary:")

print(df\_handled.isnull().sum())

# Handle columns with 5% missing values

for col in missing\_cols["cols\_5pct"]:

# Choose mean or median based on distribution

if df\_handled[col].skew() > 1 or df\_handled[col].skew() < -1:

# For skewed distributions, use median

df\_handled[col].fillna(df\_handled[col].median(), inplace=True)

print(f"Filled missing values in {col} with median")

else:

# For normal distributions, use mean

df\_handled[col].fillna(df\_handled[col].mean(), inplace=True)

print(f"Filled missing values in {col} with mean")

# Handle column with 35% missing values

col\_35pct = missing\_cols["col\_35pct"]

# Decision: Drop the column if it's not crucial, otherwise drop rows

print(f"Dropping column {col\_35pct} with 35% missing values")

df\_handled.drop(columns=[col\_35pct], inplace=True)

return df\_handled

def detect\_and\_handle\_outliers(df, method='IQR'):

"""

Detect and handle outliers in the dataset

"""

print(f"Detecting outliers using {method} method...")

df\_no\_outliers = df.copy()

# Get numerical columns (excluding class column)

numerical\_cols = df.select\_dtypes(include=np.number).columns.tolist()

if 'Class' in numerical\_cols:

numerical\_cols.remove('Class')

outliers\_summary = {}

for col in numerical\_cols:

if method == 'IQR':

# IQR method

Q1 = df\_no\_outliers[col].quantile(0.25)

Q3 = df\_no\_outliers[col].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Detect outliers

outliers = ((df\_no\_outliers[col] < lower\_bound) | (df\_no\_outliers[col] > upper\_bound))

outliers\_count = outliers.sum()

# Handle outliers by replacing with boundary values

df\_no\_outliers.loc[df\_no\_outliers[col] < lower\_bound, col] = lower\_bound

df\_no\_outliers.loc[df\_no\_outliers[col] > upper\_bound, col] = upper\_bound

outliers\_summary[col] = outliers\_count

return df\_no\_outliers

**5.2 Feature Engineering**

def apply\_pca(df, output\_dir=None):

"""

Apply PCA for dimensionality reduction

"""

print("Applying PCA for dimensionality reduction...")

# Separate features and target

X = df.drop('Class', axis=1) if 'Class' in df.columns else df

y = df['Class'] if 'Class' in df.columns else None

# Initialize and fit PCA

pca = PCA()

X\_pca = pca.fit\_transform(X)

# Get explained variance

explained\_variance = pca.explained\_variance\_ratio\_

# Determine number of components based on explained variance

avg\_var = 1.0 / len(explained\_variance)

n\_components = sum(explained\_variance > avg\_var)

print(f"Number of PCA components selected: {n\_components}")

# Create PCA with selected number of components

pca\_selected = PCA(n\_components=n\_components)

X\_pca\_selected = pca\_selected.fit\_transform(X)

# Create DataFrame with PCA components

pca\_cols = [f'PC{i+1}' for i in range(n\_components)]

df\_pca = pd.DataFrame(X\_pca\_selected, columns=pca\_cols)

# Add back the target column if it exists

if y is not None:

df\_pca['Class'] = y.values

return df\_pca, pca\_selected

def apply\_lda(df, output\_dir=None):

"""

Apply LDA for dimensionality reduction

"""

print("Applying LDA for dimensionality reduction...")

# Separate features and target

X = df.drop('Class', axis=1) if 'Class' in df.columns else df

y = df['Class'] if 'Class' in df.columns else None

if y is None:

raise ValueError("LDA requires a target variable 'Class' in the dataset")

# Initialize and fit LDA

# Number of components is min(n\_classes - 1, n\_features)

n\_classes = len(np.unique(y))

n\_features = X.shape[1]

n\_components = min(n\_classes - 1, n\_features, 3) # Capped at 3 as per project requirements

lda = LDA(n\_components=n\_components)

X\_lda = lda.fit\_transform(X, y)

# Create DataFrame with LDA components

lda\_cols = [f'LD{i+1}' for i in range(n\_components)]

df\_lda = pd.DataFrame(X\_lda, columns=lda\_cols)

# Add back the target column

df\_lda['Class'] = y.values

return df\_lda, lda

**5.3 Model Training and Evaluation**

def nested\_cross\_validation(df, classifier\_name, data\_representation='raw', output\_dir=None, n\_outer=5, n\_inner=3):

"""

Perform nested cross-validation for model evaluation

"""

print(f"Performing nested CV for {classifier\_name} on {data\_representation} data...")

# Separate features and target

X = df.drop('Class', axis=1).values

y = df['Class'].values

# Outer cross-validation

outer\_cv = StratifiedKFold(n\_splits=n\_outer, shuffle=True, random\_state=42)

# Initialize lists to store results

accuracy\_scores = []

precision\_scores = []

recall\_scores = []

f1\_scores = []

best\_params\_list = []

best\_models = []

best\_roc\_data = None

best\_confusion\_matrix = None

best\_accuracy = -1

# Perform outer cross-validation

for i, (train\_idx, test\_idx) in enumerate(outer\_cv.split(X, y)):

print(f"Outer fold {i+1}/{n\_outer}")

X\_train, X\_test = X[train\_idx], X[test\_idx]

y\_train, y\_test = y[train\_idx], y[test\_idx]

# Train and evaluate model

result = train\_and\_evaluate\_model(

X\_train, X\_test, y\_train, y\_test,

classifier\_name, None, random\_state=i

)

# Store results

accuracy\_scores.append(result['accuracy'])

precision\_scores.append(result['precision'])

recall\_scores.append(result['recall'])

f1\_scores.append(result['f1'])

best\_params\_list.append(result['best\_params'])

best\_models.append(result['model'])

# Keep track of best model for ROC curve

if result['accuracy'] > best\_accuracy:

best\_accuracy = result['accuracy']

best\_roc\_data = result['roc\_curve']

best\_confusion\_matrix = result['confusion\_matrix']

# Calculate mean and std of metrics

mean\_accuracy = np.mean(accuracy\_scores)

std\_accuracy = np.std(accuracy\_scores)

mean\_precision = np.mean(precision\_scores)

std\_precision = np.std(precision\_scores)

mean\_recall = np.mean(recall\_scores)

std\_recall = np.std(recall\_scores)

mean\_f1 = np.mean(f1\_scores)

std\_f1 = np.std(f1\_scores)

print(f"Mean Accuracy: {mean\_accuracy:.4f} (±{std\_accuracy:.4f})")

return {

'classifier': classifier\_name,

'data\_representation': data\_representation,

'accuracy': {'mean': mean\_accuracy, 'std': std\_accuracy},

'precision': {'mean': mean\_precision, 'std': std\_precision},

'recall': {'mean': mean\_recall, 'std': std\_recall},

'f1': {'mean': mean\_f1, 'std': std\_f1},

'best\_params': best\_params\_list,

'best\_roc\_data': best\_roc\_data,

'best\_models': best\_models

}

**6. Conclusions and Recommendations**

**6.1 Key Findings**

1. **Raw data superiority**: For this dataset, using the original features after proper preprocessing yielded the best classification results. Dimensionality reduction techniques, while potentially useful for visualization, resulted in information loss that negatively impacted classification performance.
2. **Model performance hierarchy**:
   * Top tier: Logistic Regression, XGBoost, and Random Forest (all >91% accuracy with raw data)
   * Middle tier: Decision Tree (90% accuracy with raw data)
   * Lower tier: Naive Bayes (89% accuracy with raw data)
3. **Feature importance**: The correlation matrix and model performance suggest that both size-related features (Area, Perimeter) and shape-related features (AspectRatio, Eccentricity) provide valuable discriminative information for bean classification.

**6.2 Recommendations**

1. **Preprocessing importance**: Proper handling of missing values, outliers, and feature scaling played a crucial role in achieving high classification performance. These steps should be emphasized in similar classification tasks.
2. **Feature preservation**: For similar classification tasks, preserving the original features after proper scaling and preprocessing is recommended over dimensionality reduction, as information loss can significantly impact performance.
3. **Model selection**: While Logistic Regression performed best, ensemble methods like Random Forest and XGBoost showed comparable performance and could be viable alternatives, especially if interpretability is not a primary concern.
4. **Production implementation**: For deployment in a real-world bean classification system, the Logistic Regression model with raw data representation offers the best balance of performance, interpretability, and computational efficiency.

This comprehensive analysis demonstrates the importance of evaluating multiple approaches (data representations and classification algorithms) to identify the optimal solution for a specific problem domain. The findings also highlight that simpler models with properly preprocessed data can outperform more complex approaches, reinforcing the principle that data quality and appropriate preprocessing often contribute more to model performance than algorithm complexity.