Automatic Evaluation of MathBERT Embeddings of Students' Algebra Questions

CS22B1009 N.Priyadarshini

CS22B1012 G.SatyaPriya

Introduction:

Grading algebraic responses in traditional educational settings is a labor-intensive task, particularly in large classrooms. These conventional methods can result in inconsistent assessments and delayed feedback for students. This project explores an automated solution using **MathBERT embeddings**, a specialized model for mathematical language processing, combined with machine learning (ML) classifiers to categorize student responses as either correct or incorrect. The proposed system aims to enhance grading efficiency, ensure uniformity, and provide timely feedback, addressing a significant gap in the evaluation of descriptive mathematical answers.

Problem Statement:

The goal is to automate the evaluation of open-ended student responses to high school algebra questions. By leveraging machine learning techniques and MathBERT embeddings, the project seeks to classify responses into correct or incorrect categories, thus reducing manual grading efforts and providing standardized assessments.

Objective:

1. **Efficiency**: Minimize manual grading time and effort for educators.
2. **Consistency**: Deliver unbiased and standardized evaluations.
3. **Scalability**: Handle large datasets of student responses effectively.

4. Advanced NLP Usage: Use MathBERT embeddings to understand mathematical language accurately.
5. Data Insights: Highlight common errors and provide actionable insights to improve teaching strategies.

Dataset Description:

The dataset includes responses from approximately 50 students to 21 algebraic questions. Each response was transformed into 385-dimensional MathBERT embeddings, yielding 1024 rows in total.

Methodology:

1. Preprocessing:
    a. Duplicate Removal: Eliminated redundant entries.
    b. Handling Missing Values: Ensured data completeness by addressing null values.
2. Feature Extraction:
    a. Using MathBERT: Each response was encoded into a 385-dimensional vector, capturing mathematical semantics and syntax.
3. Feature Selection:
    a. Applied correlation techniques to identify and remove irrelevant or redundant features.
    b. Reduced dimensionality from 385 to 384 features post-selection, improving model generalization.

TABLE I. MODELS AND HYPERPARAMETERS

Model	Hyperparameters
SVM	{'C':1,'gamma':0.1,'kernel':'rbf'}
KNN	{'K=8 best'}
Decision Tree	{'criterion':'entropy', 'max depth':'max features':'sqrt', 'min samples leaf': 4, 'min samples split': 10'}
RF	{'bootstrap': False,'max _depth': 30, 'min samples leaf': 2,'min samples split': 10, 'n estimators': 100'}

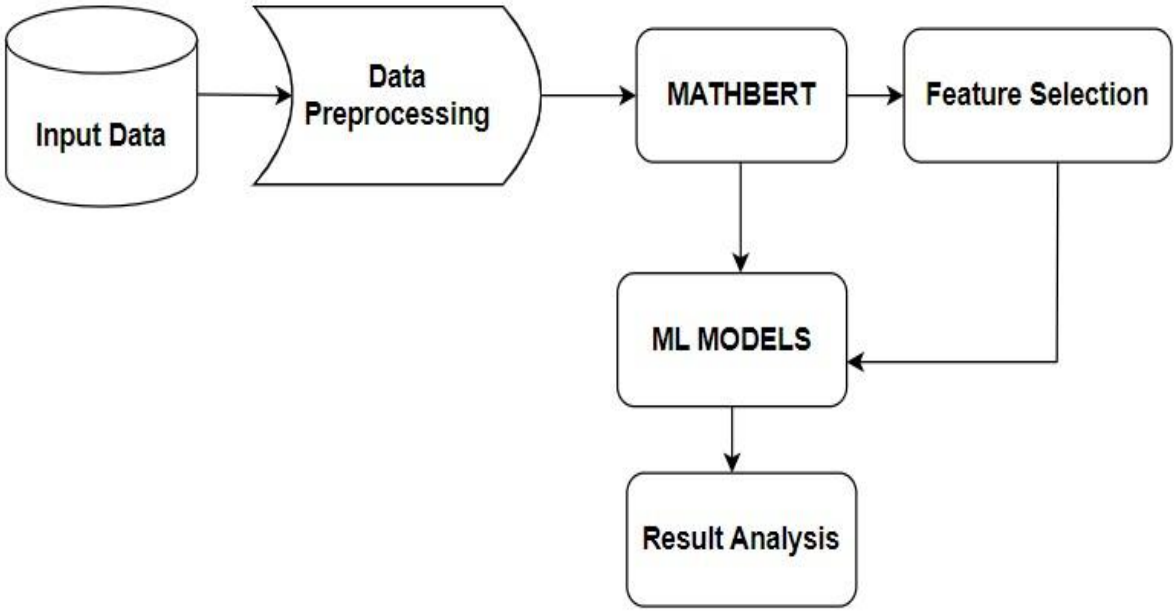
ML Models Employed:

c. Models tested include Support Vector Machines (SVM), Random Forest (RF), k-Nearest Neighbors (KNN), Decision Trees, and others.
d. Performed 4-fold cross-validation to ensure robust evaluation and avoid overfitting.

Performance Metrics:

Metrics such as accuracy, precision, recall, and F1-score were used to evaluate model effectiveness before and after feature selection.

Here are the step wise screenshots of the methodology:



INITIAL DATASET:training_mathbert.xlsx:

[1]:
```
import pandas as pd

# Read the Excel file
data = pd.read_excel('training_mathbert.xlsx')

# Create a new column 'Classification'
data['Classification'] = (data['output'] >= 4).astype(int)

# Display the updated DataFrame
print(data)
```

```
1125 -0.279187  0.099571  0.229438  0.197775 -0.326235 -0.254882 -0.000509

       embed_7   embed_8   embed_9  ...  embed_376  embed_377  embed_378  \
0     0.094621  0.330203 -0.258730  ...  -0.230662   0.173143  -0.259786
1    -0.081476  0.213762 -0.105293  ...  -0.090271   0.129022  -0.008138
2     0.014463  0.104336 -0.014190  ...  -0.117493  -0.118993  -0.046860
3     0.074028  0.149310 -0.147779  ...  -0.100965   0.236099  -0.286450
4     0.075697  0.055439 -0.200804  ...  -0.230057   0.296528  -0.526185
...        ...       ...       ...  ...        ...        ...        ...
1121  0.012244  0.254925 -0.056957  ...  -0.316320   0.013955  -0.030025
1122  0.027731  0.275638 -0.068072  ...  -0.419794   0.061882  -0.190783
1123  0.019028  0.268377 -0.188386  ...  -0.354851   0.093440  -0.093673
1124  0.074879  0.154743 -0.030652  ...  -0.428964   0.052585  -0.172558
1125  0.034733  0.283212 -0.207798  ...  -0.275116  -0.012660   0.011353

      embed_379  embed_380  embed_381  embed_382  embed_383  output  \
0     -0.316996  -0.389919   0.105596   0.196438   0.117199     0.0
1     -0.220774  -0.021343  -0.029695   0.335977  -0.197539     0.0
2      0.010008  -0.118400  -0.085768   0.512956   0.023334     0.0
3     -0.130198  -0.051258  -0.047492   0.241473  -0.095162     0.0
4     -0.251471   0.196795  -0.101786   0.570922   0.007743     0.0
...         ...        ...        ...        ...        ...     ...
1121  -0.306975   0.139429  -0.256867   0.331288  -0.045333     5.0
1122  -0.323777   0.185546  -0.144995   0.347215  -0.024521     5.0
1123  -0.271739   0.042851  -0.143179   0.410811  -0.105163     5.0
1124  -0.218717   0.201938  -0.085569   0.345646  -0.032904     5.0
1125  -0.323538   0.055882   0.001314   0.474825  -0.055474     5.0

      Classification
0                  0
1                  0
2                  0
3                  0
4                  0
...              ...
1121               1
1122               1
1123               1
1124               1
1125               1

[1126 rows x 386 columns]
```

AFTER REMOVING OUTPUT:SAVED IN data1_no_output.xlsx file:

```
data.drop(['output'], axis=1)
```

	embed_0	embed_1	embed_2	embed_3	embed_4	embed_5	embed_6	embed_7	embed_8	embed_9	...	embed_375	embed_376	embed_377	embed_378
0	-0.089926	0.343874	0.176382	0.169358	-0.413337	-0.276315	0.188070	0.094621	0.330203	-0.258730	...	-0.272278	-0.230662	0.173143	-0.259786
1	0.303261	0.084930	0.047369	-0.017244	-0.524733	-0.104934	0.335107	-0.081476	0.213762	-0.105293	...	-0.310262	-0.090271	0.129022	-0.008138
2	-0.274291	0.216801	0.029110	0.259279	-0.655594	-0.289643	0.073369	0.014463	0.104336	-0.014190	...	-0.464926	-0.117493	-0.118993	-0.046860
3	0.118676	0.095572	0.157358	0.225097	-0.632885	-0.125629	0.204013	0.074028	0.149310	-0.147779	...	-0.030543	-0.100965	0.236099	-0.286450
4	0.298772	0.300674	0.366119	-0.022142	-0.748852	-0.035268	0.277504	0.075697	0.055439	-0.200804	...	-0.149575	-0.230057	0.296528	-0.526185
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1121	-0.316305	0.126331	0.084587	0.099225	-0.503260	-0.062559	0.115574	0.012244	0.254925	-0.056957	...	-0.514027	-0.316320	0.013955	-0.030025
1122	-0.342471	0.060391	-0.009947	0.156623	-0.511338	-0.070624	0.140290	0.027731	0.275638	-0.068072	...	-0.595396	-0.419794	0.061882	-0.190783
1123	-0.379174	0.197136	0.149639	0.060261	-0.347212	-0.064022	0.119379	0.019028	0.268377	-0.188386	...	-0.517010	-0.354851	0.093440	-0.093673
1124	-0.379726	0.075891	0.100093	0.118006	-0.429774	-0.140415	0.120826	0.074879	0.154743	-0.030652	...	-0.683223	-0.428964	0.052585	-0.172558
1125	-0.279187	0.099571	0.229438	0.197775	-0.326235	-0.254882	-0.000509	0.034733	0.283212	-0.207798	...	-0.574962	-0.275116	-0.012660	0.011353

1126 rows × 385 columns

[4]:
```
import pandas as pd

# Assuming 'data' is your DataFrame and 'Average' is the column with values
# Replace 'data' and 'Average' with your actual DataFrame and column names

# Create a new column 'Classification' based on the conditions you mentioned
data = pd.read_excel('training_mathbert.xlsx')
data1 = pd.DataFrame(data)
data1['Classification'] = (data1['output'] >= 4).astype(int)

# Drop the 'output' column
data1_no_output = data1.drop(['output'], axis=1)

# Specify the file path where you want to save the Excel file
excel_file_path = 'data1_no_output.xlsx'

# Write the DataFrame without the 'output' column to an Excel file
data1_no_output.to_excel(excel_file_path, index=False)

# Print a message indicating the successful write
print(f"DataFrame without 'output' column written to {excel_file_path}")
```

```
DataFrame without 'output' column written to data1_no_output.xlsx
```

AFTER REMOVING DUPLICATES:saved in no_duplicated.xlsx file:

[11]:
```
import pandas as pd

# Assuming 'data' is your DataFrame and 'Average' is the column with values
# Replace 'data' and 'Average' with your actual DataFrame and column names

# Create a new column 'Classification' based on the conditions you mentioned
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)
```

[12]:
```
df1
```

[12]:

	embed_0	embed_1	embed_2	embed_3	embed_4	embed_5	embed_6	embed_7	embed_8	embed_9	...	embed_375	embed_376	embed_377	embed_378
0	-0.089926	0.343874	0.176382	0.169358	-0.413337	-0.276315	0.188070	0.094621	0.330203	-0.258730	...	-0.272278	-0.230662	0.173143	-0.259786
1	0.303261	0.084930	0.047369	-0.017244	-0.524733	-0.104934	0.335107	-0.081476	0.213762	-0.105293	...	-0.310262	-0.090271	0.129022	-0.008138
2	-0.274291	0.216801	0.029110	0.259279	-0.655594	-0.289643	0.073369	0.014463	0.104336	-0.014190	...	-0.464926	-0.117493	-0.118993	-0.046860
3	0.118676	0.095572	0.157358	0.225097	-0.632885	-0.125629	0.204013	0.074028	0.149310	-0.147779	...	-0.030543	-0.100965	0.236099	-0.286450
4	0.298772	0.300674	0.366119	-0.022142	-0.748852	-0.035268	0.277504	0.075697	0.055439	-0.200804	...	-0.149575	-0.230057	0.296528	-0.526185
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1029	-0.316305	0.126331	0.084587	0.099225	-0.503260	-0.062559	0.115574	0.012244	0.254925	-0.056957	...	-0.514027	-0.316320	0.013955	-0.030025
1030	-0.342471	0.060391	-0.009947	0.156623	-0.511338	-0.070624	0.140290	0.027731	0.275638	-0.068072	...	-0.595396	-0.419794	0.061882	-0.190783
1031	-0.379174	0.197136	0.149639	0.060261	-0.347212	-0.064022	0.119379	0.019028	0.268377	-0.188386	...	-0.517010	-0.354851	0.093440	-0.093673

TRAINING AND TESTING SET SHAPE:

[13]:
```
from sklearn.model_selection import train_test_split
import pandas as pd

data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)
# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the dataset into training and testing sets (80% training, 20% testing)+
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Display the shapes of the training and testing sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (827, 384) (827,)
Testing set shape: (207, 384) (207,)
```

KNN CLASSIFIER:(calculating accuracy,precison,f1 score,recall):

1]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)
# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the parameter grid
param_grid = {'n_neighbors': np.arange(1, 11)}

# Create the KNN classifier
knn = KNeighborsClassifier()

# Create GridSearchCV object
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_k = grid_search.best_params_['n_neighbors']

# Train the final model with the best hyperparameters
final_model = KNeighborsClassifier(n_neighbors=best_k)
final_model.fit(X_train, y_train)

# Predict on the test set
y_pred = final_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Best k: {best_k}")
print(f"Test Accuracy: {accuracy}")
```

```
Best k: 8
Test Accuracy: 0.7246376811594203
```

[15]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)
# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert the continuous target values to discrete classes
y_train_classes = np.round(y_train)
y_test_classes = np.round(y_test)

# Initialize the KNN classifier
knn_model = KNeighborsClassifier(n_neighbors=8)  # You can set the desired hyperparameters here

# Train the KNN model on the training set
knn_model.fit(X_train, y_train_classes)

# Predict the output (classes) on the training set
y_train_pred = knn_model.predict(X_train)

# Predict the output (classes) on the testing set
y_test_pred = knn_model.predict(X_test)

# Calculate accuracy on the training set
train_accuracy = accuracy_score(y_train_classes, y_train_pred)

# Calculate accuracy on the testing set
test_accuracy = accuracy_score(y_test_classes, y_test_pred)

# Calculate F1 score, precision, and recall on the testing set
f1 = f1_score(y_test_classes, y_test_pred, average='weighted')  # 'weighted' takes class imbalance into account
precision = precision_score(y_test_classes, y_test_pred, average='weighted')
```

```python
# Calculate F1 score, precision, and recall on the testing set
f1 = f1_score(y_test_classes, y_test_pred, average='weighted')  # 'weighted' takes class imbalance into account
precision = precision_score(y_test_classes, y_test_pred, average='weighted')
recall = recall_score(y_test_classes, y_test_pred, average='weighted')

# Display the results
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Training Accuracy: 0.7859733978234583
Testing Accuracy: 0.7246376811594203
F1 Score: 0.7238517036192456
Precision: 0.7230999930656682
Recall: 0.7246376811594203
```

[16]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Convert the continuous target values to discrete classes
    y_train_classes = np.round(y_train)
    y_test_classes = np.round(y_test)

    # Initialize the KNN classifier
    knn_model = KNeighborsClassifier(n_neighbors=8)

    # Train the KNN model on the training set
    knn_model.fit(X_train, y_train_classes)

    # Predict the output (classes) on the testing set
    y_test_pred = knn_model.predict(X_test)
```

```
    f1 = f1_score(y_test_classes, y_test_pred, average='weighted')
    precision = precision_score(y_test_classes, y_test_pred, average='weighted')
    recall = recall_score(y_test_classes, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.7318840579710144
Mean F1 Score: 0.7149047366793011
Mean Precision: 0.714629858188255
Mean Recall: 0.7318840579710144
Standard Deviation Accuracy: 0.02319343745730975
Standard Deviation F1 Score: 0.024610261248219932
Standard Deviation Precision: 0.02538862962701297
Standard Deviation Recall: 0.02319343745730975
```

SUPER VECTOR MACHINE(SVM): (calculating accuracy,precison,f1 score,recall):

[17]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)
# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto', 0.1, 0.01, 0.001]
}

# Create the SVM classifier
svm = SVC()

# Create GridSearchCV object
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the final model with the best hyperparameters
final_model = SVC(**best_params)
final_model.fit(X_train, y_train)

# Predict on the test set
y_pred = final_model.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Best hyperparameters: {best_params}")
print(f"Test Accuracy: {accuracy}")
```

```
Best hyperparameters: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Test Accuracy: 0.7729468599033816
```

8]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the SVM classifier
svm_model = SVC(kernel='rbf', C=1.0)  # You can set the desired hyperparameters here

# Train the SVM model on the training set
svm_model.fit(X_train, y_train)

# Predict the output on the training set
y_train_pred = svm_model.predict(X_train)

# Predict the output on the testing set
y_test_pred = svm_model.predict(X_test)

# Calculate accuracy on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculate accuracy on the testing set
test_accuracy = accuracy_score(y_test, y_test_pred)

# Calculate F1 score, precision, and recall on the testing set
f1 = f1_score(y_test, y_test_pred, average='weighted')  # 'weighted' takes class imbalance into account
```

```
# Calculate F1 score, precision, and recall on the testing set
f1 = f1_score(y_test, y_test_pred, average='weighted')  # 'weighted' takes class imbalance into account
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')

# Display the results
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Training Accuracy: 0.7654171704957679
Testing Accuracy: 0.782608695652174
F1 Score: 0.747911515619481
Precision: 0.7748109640831757
Recall: 0.782608695652174
```

[19]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the SVM model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the SVM classifier
    svm_model = SVC(kernel='rbf', C=1.0)

    # Train the SVM model on the training set
    svm_model.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = svm_model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
```

```
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.7521739130434782
Mean F1 Score: 0.6994897563938427
Mean Precision: 0.7526735341015847
Mean Recall: 0.7521739130434782
Standard Deviation Accuracy: 0.023373855851986387
Standard Deviation F1 Score: 0.03505564471430314
Standard Deviation Precision: 0.03030597081159757
```

DECISION TREE: (calculating accuracy,precison,f1 score,recall):

```
import pandas as pd
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the parameter grid for Decision Tree
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

# Create the Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Create GridSearchCV object
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the final model with the best hyperparameters
final_model = DecisionTreeClassifier(**best_params)
final_model.fit(X_train, y_train)
```

```
# Fit the model
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the final model with the best hyperparameters
final_model = DecisionTreeClassifier(**best_params)
final_model.fit(X_train, y_train)

# Predict on the test set
y_pred = final_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Best hyperparameters: {best_params}")
print(f"Test Accuracy: {accuracy}")
```

```
Best hyperparameters: {'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5}
Test Accuracy: 0.6763285024154589
```

[21]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from math import sqrt  # Add this import statement

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree classifier
dt_model = DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_split=10, min_samples_leaf=4, max_features='sqrt')

# Train the Decision Tree model on the training set
dt_model.fit(X_train, y_train)

# Predict the output on the training set
y_train_pred = dt_model.predict(X_train)

# Predict the output on the testing set
y_test_pred = dt_model.predict(X_test)

# Calculate accuracy on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculate accuracy on the testing set
test_accuracy = accuracy_score(y_test, y_test_pred)

# Calculate F1 score, precision, and recall on the testing set
f1 = f1_score(y_test, y_test_pred, average='weighted')  # 'weighted' takes class imbalance into account
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')
```

```
# Calculate F1 score, precision, and recall on the testing set
f1 = f1_score(y_test, y_test_pred, average='weighted')  # 'weighted' takes class imbalance into account
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')

# Display the results
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Training Accuracy: 0.939540507859734
Testing Accuracy: 0.6811594202898551
F1 Score: 0.6860340443612366
Precision: 0.6919927133207591
Recall: 0.6811594202898551
```

[22]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the Decision Tree model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the Decision Tree classifier
    dt_model = DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_split=10, min_samples_leaf=4, max_features='sqrt')

    # Train the Decision Tree model on the training set
    dt_model.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = dt_model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
```

```
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results+
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.6690821256038648
Mean F1 Score: 0.6664828862927621
Mean Precision: 0.6659753793759758
Mean Recall: 0.6690821256038648
Standard Deviation Accuracy: 0.049419605116123734
Standard Deviation F1 Score: 0.04995152623662305
Standard Deviation Precision: 0.05172054676452486
Standard Deviation Recall: 0.049419605116123734
```

RANDOM FOREST: (calculating accuracy,precison,f1 score,recall):

[23]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
    # Additional parameters for Random Forest can be added here
}

# Create the Random Forest classifier
rf = RandomForestClassifier()

# Create GridSearchCV object
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the final model with the best hyperparameters
```

```python
# Train the final model with the best hyperparameters
final_model = RandomForestClassifier(**best_params)
final_model.fit(X_train, y_train)

# Predict on the test set
y_pred = final_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Best hyperparameters: {best_params}")
print(f"Test Accuracy: {accuracy}")
```

```
Best hyperparameters: {'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 50}
Test Accuracy: 0.7536231884057971
```

[*]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Read the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the Random Forest model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the Random Forest classifier
    rf = RandomForestClassifier()

    # Train the Random Forest model on the training set
    rf.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = rf.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.7493212669683258
Mean F1 Score: 0.7179199306111091
Mean Precision: 0.7267727187692207
Mean Recall: 0.7493212669683258
Standard Deviation Accuracy: 0.017664453661487025
Standard Deviation F1 Score: 0.01999324826848087
```

AFTER FEATURE SELECTION CODES:

KNN CLASSIFIER:

[5]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.neighbors import KNeighborsClassifier

# Load the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Calculate the correlation matrix
correlation_matrix = X.corr().abs()

# Create a mask to ignore the upper triangle of the correlation matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set the threshold for high correlation
threshold = 0.8  # Example threshold (change as needed)

# Find index of feature columns with correlation greater than the threshold
high_correlation = correlation_matrix.mask(mask).stack() > threshold
high_correlation_features = high_correlation[high_correlation].index.tolist()

# Get unique features involved in high correlation
features_to_drop = set()
for feat1, feat2 in high_correlation_features:
    features_to_drop.add(feat1 if correlation_matrix.loc[feat1, 'Classification'] < correlation_matrix.loc[feat2, 'Classification'] else feat2)

# Drop highly correlated features from the dataset
X_filtered = X.drop(columns=features_to_drop)

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the KNN model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_filtered, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=8)  # Adjust parameters as needed

    # Train the KNN model on the training set
    knn.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = knn.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
```

```
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.7294685990338163
Mean F1 Score: 0.7085109487242736
Mean Precision: 0.7130388948897278
Mean Recall: 0.7294685990338163
Standard Deviation Accuracy: 0.021926194623981113
Standard Deviation F1 Score: 0.03031753228764195
Standard Deviation Precision: 0.027489986023943192
Standard Deviation Recall: 0.021926194623981113
```

SVM(SUPPORT VECTOR MACHINE):

[15]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Load the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Calculate the correlation matrix
correlation_matrix = X.corr().abs()

# Create a mask to ignore the upper triangle of the correlation matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set the threshold for high correlation
threshold = 0.8  # Example threshold (change as needed)

# Find index of feature columns with correlation greater than the threshold
high_correlation = correlation_matrix.mask(mask).stack() > threshold
high_correlation_features = high_correlation[high_correlation].index.tolist()

# Get unique features involved in high correlation
features_to_drop = set()
for feat1, feat2 in high_correlation_features:
    features_to_drop.add(feat1 if correlation_matrix.loc[feat1, 'Classification'] < correlation_matrix.loc[feat2, 'Classification'] else feat2)

# Drop highly correlated features from the dataset
X_filtered = X.drop(columns=features_to_drop)

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_filtered_standardized = scaler.fit_transform(X_filtered)
```

```python
# Standardize the features using StandardScaler
scaler = StandardScaler()
X_filtered_standardized = scaler.fit_transform(X_filtered)

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the SVM model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_filtered_standardized, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the SVM classifier
    svm_model = SVC(kernel='rbf', C=1.0)
    # Train the SVM model on the training set
    svm_model.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = svm_model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
```

```python
# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.7468599033816424
Mean F1 Score: 0.7118455765929317
Mean Precision: 0.732581213416721
Mean Recall: 0.7468599033816424
Standard Deviation Accuracy: 0.024840502670883943
Standard Deviation F1 Score: 0.027977276752283075
Standard Deviation Precision: 0.030488572383062372
Standard Deviation Recall: 0.024840502670883943
```

DECISION TREE:

[20]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.tree import DecisionTreeClassifier

# Load the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Calculate the correlation matrix
correlation_matrix = X.corr().abs()

# Create a mask to ignore the upper triangle of the correlation matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set the threshold for high correlation
threshold = 0.8  # Example threshold (change as needed)

# Find index of feature columns with correlation greater than the threshold
high_correlation = correlation_matrix.mask(mask).stack() > threshold
high_correlation_features = high_correlation[high_correlation].index.tolist()

# Get unique features involved in high correlation
features_to_drop = set()
for feat1, feat2 in high_correlation_features:
    features_to_drop.add(feat1 if correlation_matrix.loc[feat1, 'Classification'] < correlation_matrix.loc[feat2, 'Classification'] else feat2)

# Drop highly correlated features from the dataset
X_filtered = X.drop(columns=features_to_drop)

# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the Decision Tree model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_filtered, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the Decision Tree classifier
    decision_tree = DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_split=10, min_samples_leaf=4, max_features='sqrt')  # Adjust pa

    # Train the Decision Tree model on the training set
    decision_tree.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = decision_tree.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
```

```python
# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

```
Mean Accuracy: 0.6492753623188406
Mean F1 Score: 0.6481860006601761
Mean Precision: 0.6483451174923448
Mean Recall: 0.6492753623188406
Standard Deviation Accuracy: 0.02682790990708179
Standard Deviation F1 Score: 0.02815674220787503
Standard Deviation Precision: 0.030436615337206763
Standard Deviation Recall: 0.02682790990708179
```

RANDOM FOREST:

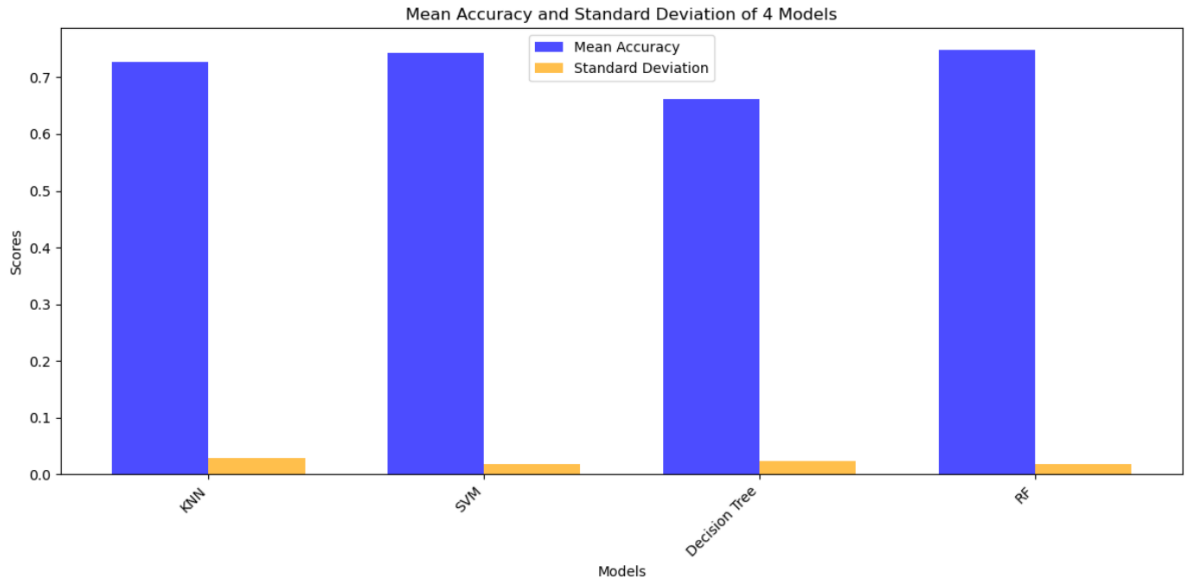
[23]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

# Load the data from the Excel file
data = pd.read_excel('no_duplicates.xlsx')
df1 = pd.DataFrame(data)

# Separate features (X) and target variable (y)
X = df1.drop('Classification', axis=1)
y = df1['Classification']

# Calculate the correlation matrix
correlation_matrix = X.corr().abs()

# Create a mask to ignore the upper triangle of the correlation matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set the threshold for high correlation
threshold = 0.8  # Example threshold (change as needed)

# Find index of feature columns with correlation greater than the threshold
high_correlation = correlation_matrix.mask(mask).stack() > threshold
high_correlation_features = high_correlation[high_correlation].index.tolist()

# Get unique features involved in high correlation
features_to_drop = set()
for feat1, feat2 in high_correlation_features:
    features_to_drop.add(feat1 if correlation_matrix.loc[feat1, 'Classification'] < correlation_matrix.loc[feat2, 'Classification'] else feat2)

# Drop highly correlated features from the dataset
X_filtered = X.drop(columns=features_to_drop)

# Standardize the features using StandardScaler
scaler = StandardScaler()
X_filtered_standardized = scaler.fit_transform(X_filtered)
```

```python
# Initialize lists to store evaluation metrics
accuracy_list = []
f1_list = []
precision_list = []
recall_list = []

# Set the number of iterations for random train-test splits
num_iterations = 10

# Run the Random Forest model on multiple random train-test splits
for _ in range(num_iterations):
    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_filtered_standardized, y, test_size=0.2, random_state=np.random.randint(1, 100))

    # Initialize the Random Forest classifier
    param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
    # Additional parameters for Random Forest can be added here
}

    rf = RandomForestClassifier(n_estimators=50, random_state=42)

    # Train the Random Forest model on the training set
    rf.fit(X_train, y_train)

    # Predict the output on the testing set
    y_test_pred = rf.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred, average='weighted')
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')

    # Append metrics to the lists
    accuracy_list.append(accuracy)
    f1_list.append(f1)
    precision_list.append(precision)
    recall_list.append(recall)

# Calculate mean and standard deviation for each metric
mean_accuracy = np.mean(accuracy_list)
mean_f1 = np.mean(f1_list)
mean_precision = np.mean(precision_list)
mean_recall = np.mean(recall_list)

std_accuracy = np.std(accuracy_list)
std_f1 = np.std(f1_list)
std_precision = np.std(precision_list)
std_recall = np.std(recall_list)

# Display the results
print("Mean Accuracy:", mean_accuracy)
print("Mean F1 Score:", mean_f1)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)

print("Standard Deviation Accuracy:", std_accuracy)
print("Standard Deviation F1 Score:", std_f1)
print("Standard Deviation Precision:", std_precision)
print("Standard Deviation Recall:", std_recall)
```

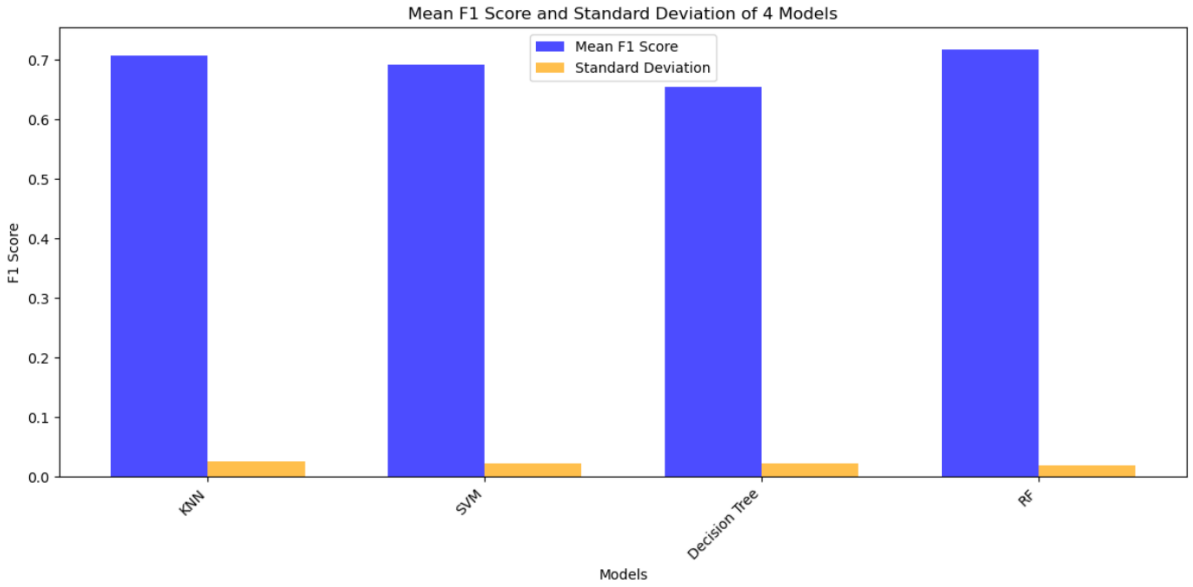
```
Mean Accuracy: 0.7314009661835749
Mean F1 Score: 0.6901696657909827
Mean Precision: 0.7035321660212347
Mean Recall: 0.7314009661835749
Standard Deviation Accuracy: 0.027768324373333174
Standard Deviation F1 Score: 0.03552162984686795
Standard Deviation Precision: 0.032355436373300656
Standard Deviation Recall: 0.027768324373333174
```

CODES AND OUTPUTS OF PLOTTINGS:

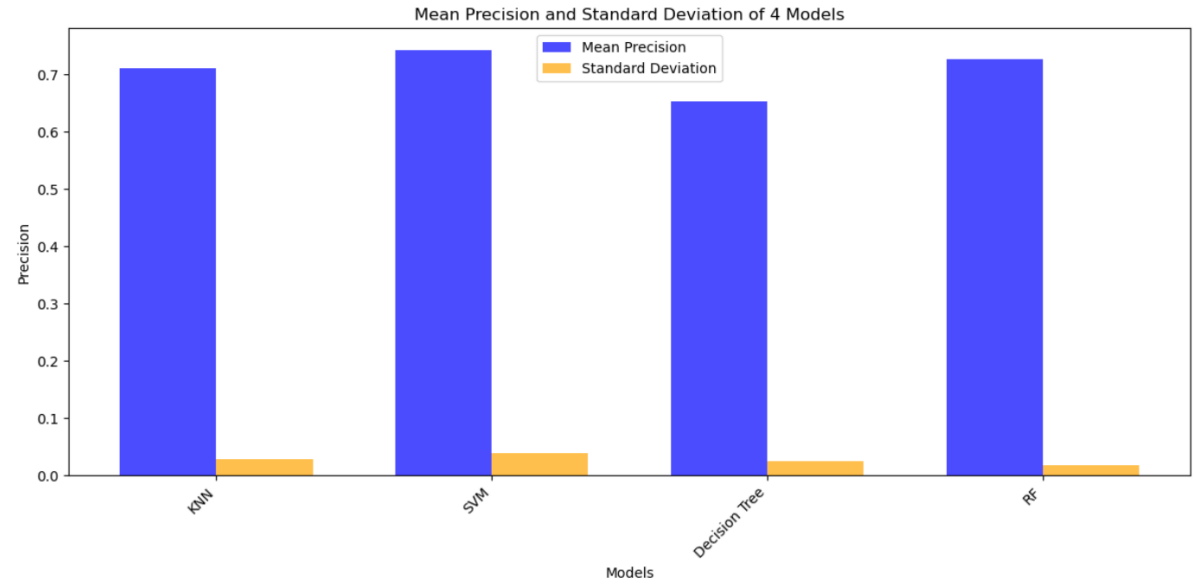
[17]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean accuracy, and standard deviation
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_accuracy = [0.73, 0.75, 0.66, 0.73]
std_accuracy = [0.02, 0.02, 0.04, 0.02]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean accuracy and standard deviation
fig, ax = plt.subplots(figsize=(12, 6))

bar1 = ax.bar(np.arange(len(models)), mean_accuracy, bar_width, color='blue', alpha=0.7, label='Mean Accuracy')
bar2 = ax.bar(np.arange(len(models)) + bar_width, std_accuracy, bar_width, color='orange', alpha=0.7, label='Standard Deviation')

# Labeling and formatting
ax.set_title('Mean Accuracy and Standard Deviation of 4 Models')
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



•[21]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean F1 score, and standard deviation
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_f1 = [0.71, 0.69, 0.66, 0.71]
std_f1 = [0.024, 0.03, 0.04, 0.01]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean F1 score and standard deviation
fig, ax = plt.subplots(figsize=(12, 6))

bar1 = ax.bar(np.arange(len(models)), mean_f1, bar_width, color='blue', alpha=0.7, label='Mean F1 Score')
bar2 = ax.bar(np.arange(len(models)) + bar_width, std_f1, bar_width, color='orange', alpha=0.7, label='Standard Deviation')

# Labeling and formatting
ax.set_title('Mean F1 Score and Standard Deviation of 4 Models')
ax.set_xlabel('Models')
ax.set_ylabel('F1 Score')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

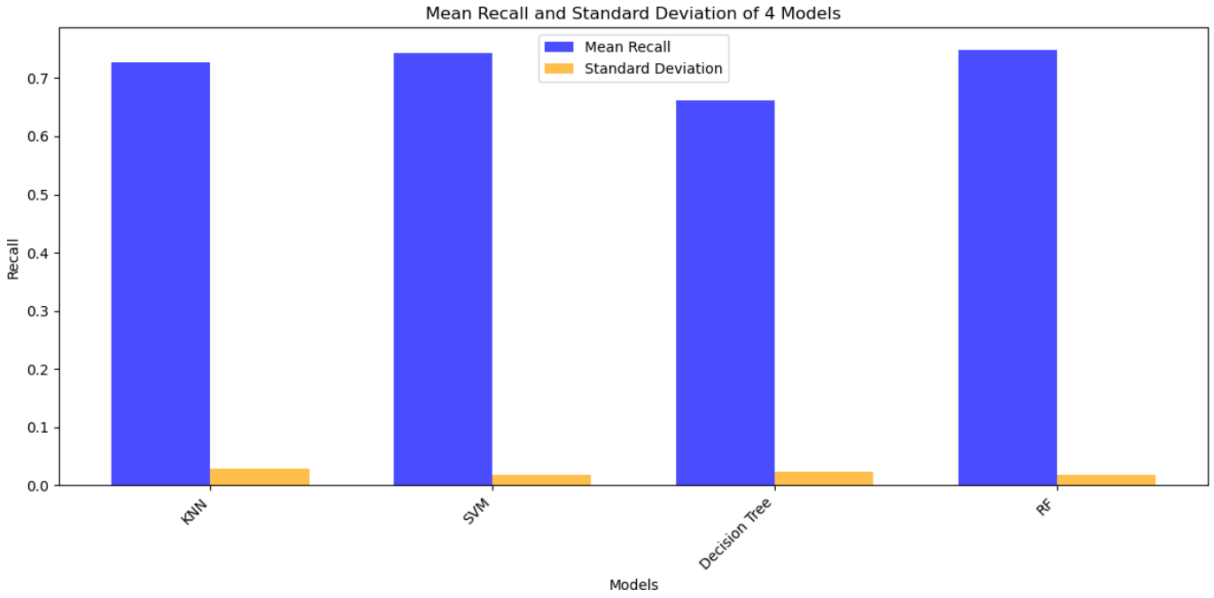



```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean precision, and standard deviation
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_precision = [0.71, 0.75, 0.66, 0.72]
std_precision = [0.02, 0.03, 0.05, 0.01]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean precision and standard deviation
fig, ax = plt.subplots(figsize=(12, 6))

bar1 = ax.bar(np.arange(len(models)), mean_precision, bar_width, color='blue', alpha=0.7, label='Mean Precision')
bar2 = ax.bar(np.arange(len(models)) + bar_width, std_precision, bar_width, color='orange', alpha=0.7, label='Standard Deviation')

# Labeling and formatting
ax.set_title('Mean Precision and Standard Deviation of 4 Models')
ax.set_xlabel('Models')
ax.set_ylabel('Precision')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

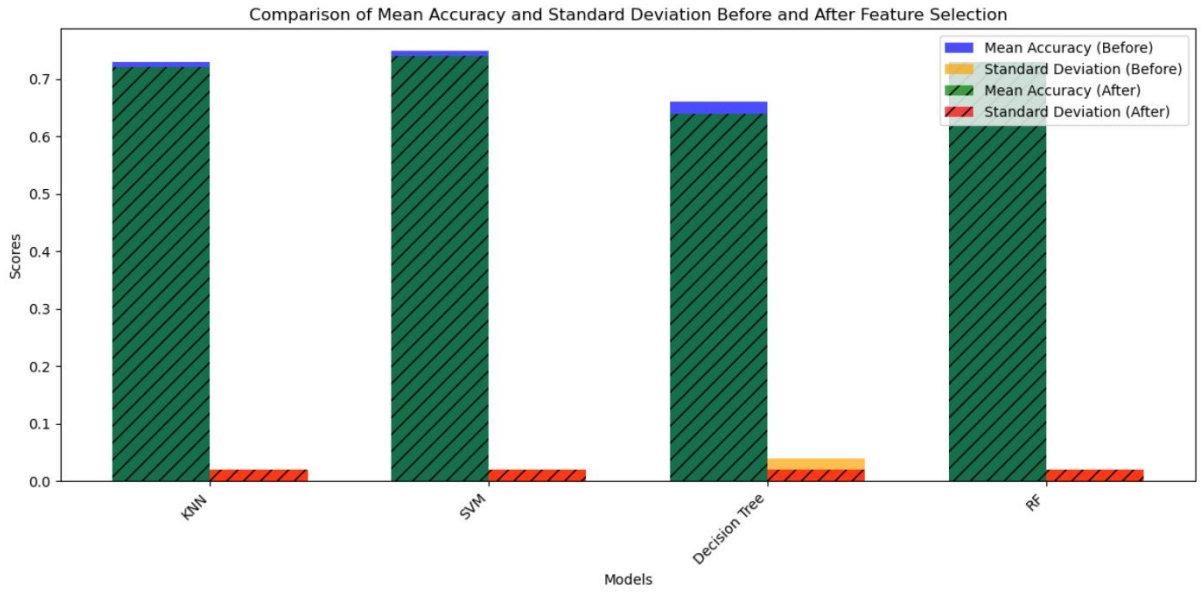


[25]:

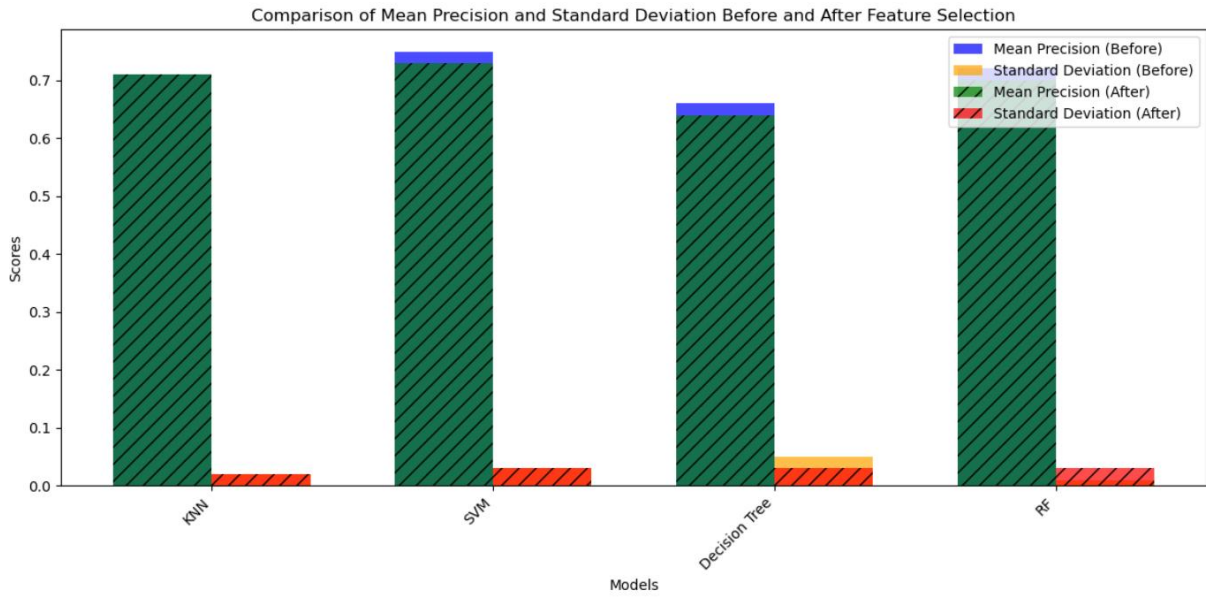
```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean recall, and standard deviation
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_recall = [0.73, 0.75, 0.66, 0.74]
std_recall = [0.02, 0.02, 0.04, 0.01]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean recall and standard deviation
fig, ax = plt.subplots(figsize=(12, 6))

bar1 = ax.bar(np.arange(len(models)), mean_recall, bar_width, color='blue', alpha=0.7, label='Mean Recall')
bar2 = ax.bar(np.arange(len(models)) + bar_width, std_recall, bar_width, color='orange', alpha=0.7, label='Standard Deviation')

# Labeling and formatting
ax.set_title('Mean Recall and Standard Deviation of 4 Models')
ax.set_xlabel('Models')
ax.set_ylabel('Recall')  # Updated y-axis label
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



```
]: import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean accuracy, and standard deviation before feature selection
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_accuracy_before = [0.73, 0.75, 0.66, 0.73]
std_accuracy_before = [0.02, 0.02, 0.04, 0.02]

# Data for the models, mean accuracy, and standard deviation after feature selection
mean_accuracy_after = [0.72, 0.74, 0.64, 0.73]
std_accuracy_after = [0.02, 0.02, 0.02, 0.02]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean accuracy and standard deviation before and after feature selection
fig, ax = plt.subplots(figsize=(12, 6))

bar1_before = ax.bar(np.arange(len(models)), mean_accuracy_before, bar_width, color='blue', alpha=0.7, label='Mean Accuracy (Before)')
bar2_before = ax.bar(np.arange(len(models)) + bar_width, std_accuracy_before, bar_width, color='orange', alpha=0.7, label='Standard Deviation (Before)')

bar1_after = ax.bar(np.arange(len(models)), mean_accuracy_after, bar_width, color='green', alpha=0.7, label='Mean Accuracy (After)', hatch='//')
bar2_after = ax.bar(np.arange(len(models)) + bar_width, std_accuracy_after, bar_width, color='red', alpha=0.7, label='Standard Deviation (After)', hatch=

# Labeling and formatting
ax.set_title('Comparison of Mean Accuracy and Standard Deviation Before and After Feature Selection')
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

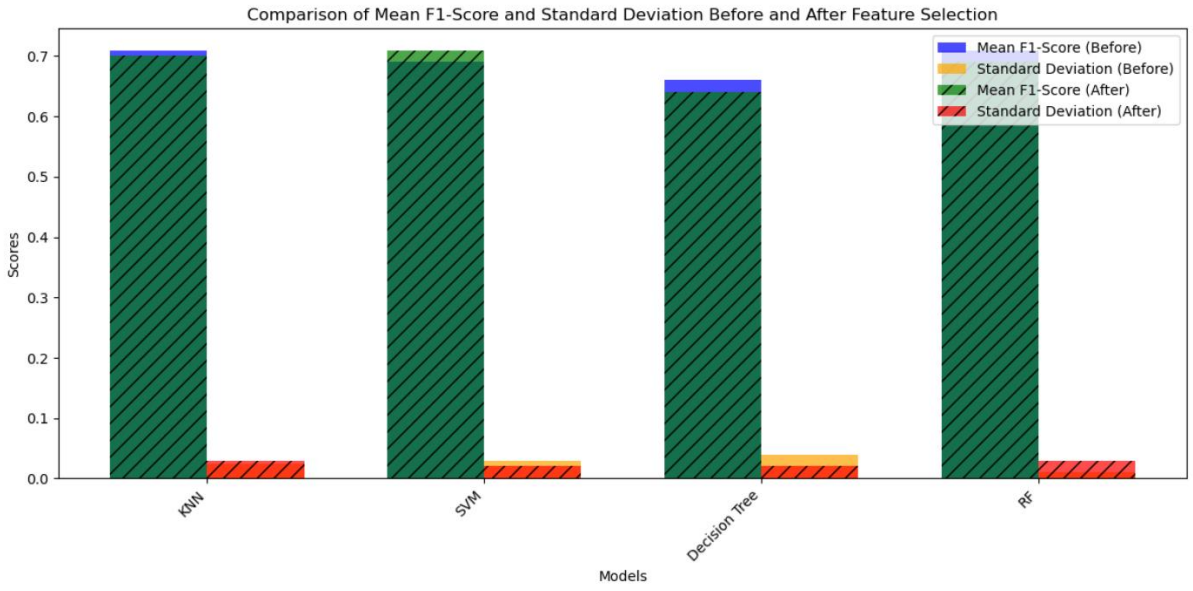


```
7]:
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean Precision, and standard deviation before feature selection
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_precision_before = [0.71, 0.75, 0.66, 0.72]
std_precision_before = [0.02, 0.03, 0.05, 0.01]

# Data for the models, mean Precision, and standard deviation after feature selection
mean_precision_after = [0.71, 0.73, 0.64,0.70]
std_precision_after = [0.02, 0.03, 0.03, 0.03]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean Precision and standard deviation before and after feature selection
fig, ax = plt.subplots(figsize=(12, 6))

bar1_before = ax.bar(np.arange(len(models)), mean_precision_before, bar_width, color='blue', alpha=0.7, label='Mean Precision (Before)')
bar2_before = ax.bar(np.arange(len(models)) + bar_width, std_precision_before, bar_width, color='orange', alpha=0.7, label='Standard Deviation (Before)')

bar1_after = ax.bar(np.arange(len(models)), mean_precision_after, bar_width, color='green', alpha=0.7, label='Mean Precision (After)', hatch='//')
bar2_after = ax.bar(np.arange(len(models)) + bar_width, std_precision_after, bar_width, color='red', alpha=0.7, label='Standard Deviation (After)', hatch

# Labeling and formatting
ax.set_title('Comparison of Mean Precision and Standard Deviation Before and After Feature Selection')
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

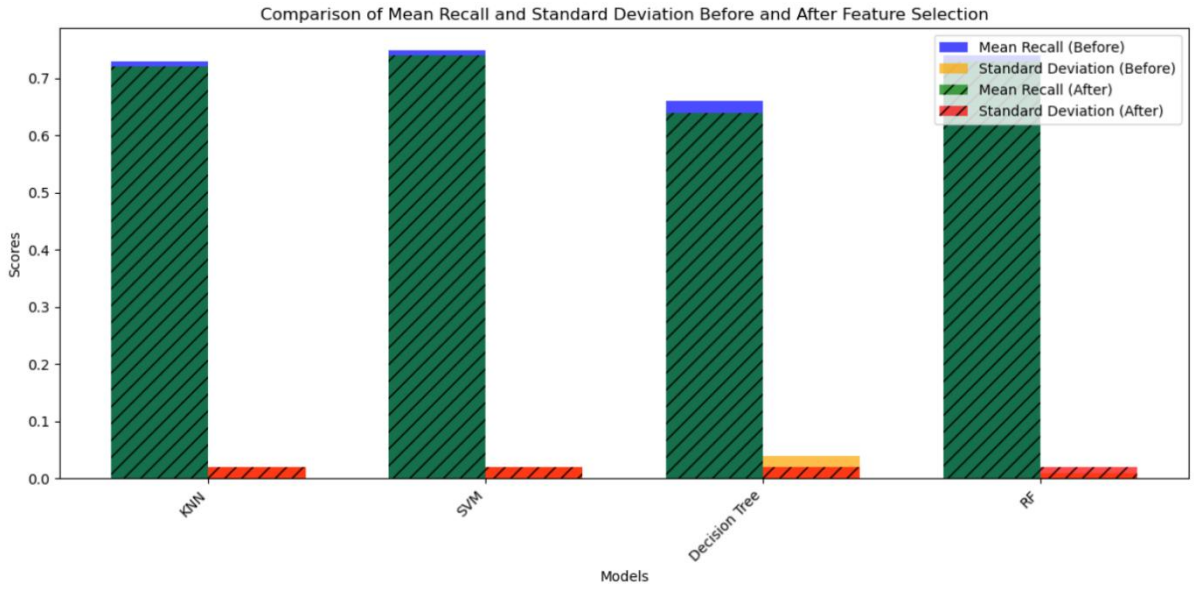


[45]:

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean F1-Score, and standard deviation before feature selection
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_f1_before = [0.71, 0.69, 0.66, 0.71]
std_f1_before = [0.024, 0.03, 0.04, 0.01]

# Data for the models, mean F1-Score, and standard deviation after feature selection
mean_f1_after = [0.70, 0.71, 0.64, 0.69]
std_f1_after = [0.03, 0.02, 0.02, 0.03]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean F1-Score and standard deviation before and after feature selection
fig, ax = plt.subplots(figsize=(12, 6))

bar1_before = ax.bar(np.arange(len(models)), mean_f1_before, bar_width, color='blue', alpha=0.7, label='Mean F1-Score (Before)')
bar2_before = ax.bar(np.arange(len(models)) + bar_width, std_f1_before, bar_width, color='orange', alpha=0.7, label='Standard Deviation (Before)')

bar1_after = ax.bar(np.arange(len(models)), mean_f1_after, bar_width, color='green', alpha=0.7, label='Mean F1-Score (After)', hatch='//')
bar2_after = ax.bar(np.arange(len(models)) + bar_width, std_f1_after, bar_width, color='red', alpha=0.7, label='Standard Deviation (After)', hatch='//')

# Labeling and formatting
ax.set_title('Comparison of Mean F1-Score and Standard Deviation Before and After Feature Selection')
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



[43]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the models, mean Recall, and standard deviation before feature selection
models = ['KNN', 'SVM', 'Decision Tree', 'RF']
mean_recall_before = [0.73, 0.75, 0.66, 0.74]
std_recall_before = [0.02, 0.02, 0.04, 0.01]
# Data for the models, mean Recall, and standard deviation after feature selection
mean_recall_after = [0.72, 0.74, 0.64, 0.73]
std_recall_after = [0.02, 0.02, 0.02,0.02,]

# Set the bar width
bar_width = 0.35

# Create a grouped bar plot for mean Recall and standard deviation before and after feature selection
fig, ax = plt.subplots(figsize=(12, 6))

bar1_before = ax.bar(np.arange(len(models)), mean_recall_before, bar_width, color='blue', alpha=0.7, label='Mean Recall (Before)')
bar2_before = ax.bar(np.arange(len(models)) + bar_width, std_recall_before, bar_width, color='orange', alpha=0.7, label='Standard Deviation (Before)')

bar1_after = ax.bar(np.arange(len(models)), mean_recall_after, bar_width, color='green', alpha=0.7, label='Mean Recall (After)', hatch='//')
bar2_after = ax.bar(np.arange(len(models)) + bar_width, std_recall_after, bar_width, color='red', alpha=0.7, label='Standard Deviation (After)', hatch='/

# Labeling and formatting
ax.set_title('Comparison of Mean Recall and Standard Deviation Before and After Feature Selection')
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_xticks(np.arange(len(models)) + bar_width / 2)
ax.set_xticklabels(models, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



Result and Analysis:

MODEL	TRAINING ACCURACY	TESTING ACCURACY	F1-SCORE	PRECISION	RECALL
KNN	78.58%	72.46%	0.72	0.72	0.72
SVM	76.54%	78.26%	0.74	0.77	0.78
DECISION TREE	93.95%	68.11%	0.68	0.69	0.68
RF	99.03%	77.77%	0.76	0.76	0.77

Before Feature Selection:

a. **SVM** emerged as the best-performing model, with superior accuracy and precision compared to other models like RF, KNN, and Decision Trees.

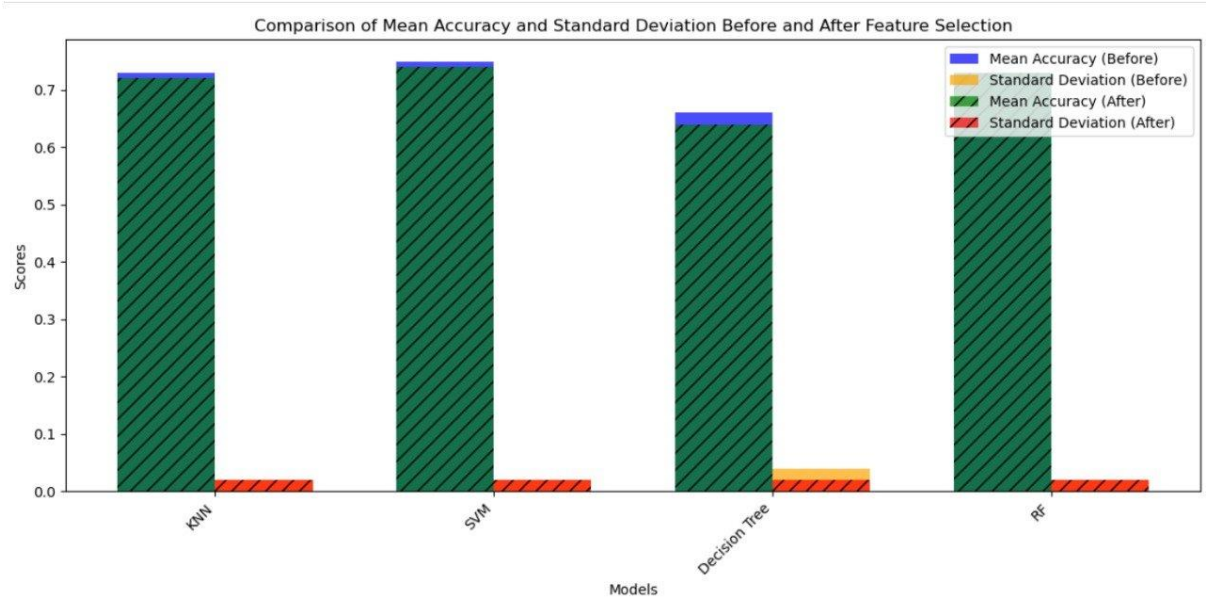
Models	Accuracy		Precision		Recall		F1-Score	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
KNN	0.73	0.02	0.71	0.02	0.73	0.02	0.71	0.02
SVM	0.75	0.02	0.75	0.03	0.75	0.02	0.69	0.03
DECISION TREE	0.66	0.04	0.66	0.05	0.66	0.04	0.66	0.01
RF	0.73	0.02	0.72	0.01	0.74	0.01	0.71	0.01

2. **After Feature Selection**:

a. SVM maintained its position as the most accurate model, demonstrating resilience to dimensionality reduction.
b. Mean accuracy slightly decreased post-feature selection, but the reduction in standard deviation indicated greater consistency.

Models	**Accuracy**		**Precision**		**Recall**		**F1-Score**	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
KNN	0.72	0.02	0.71	0.02	0.72	0.02	0.70	0.03
SVM	0.74	0.02	0.73	0.03	0.74	0.02	0.71	0.02
DECISION TREE	0.64	0.02	0.64	0.03	0.64	0.02	0.64	0.02
RF	0.73	0.02	0.70	0.03	0.73	0.02	0.69	0.03

3. **Performance Metrics Visualization**:
    a. Bar plots of accuracy, precision, recall, and F1-scores highlighted SVM as the leading model across all metrics, followed by ensemble methods like Random Forest.



Explanation of MathBert:

MathBERT is a fine-tuned variant of BERT tailored for mathematical contexts. It processes mathematical language, equations, and problem statements effectively by leveraging contextual embeddings. These embeddings are dense vectors that encapsulate the meaning and structure of student responses, enabling accurate classification by ML models.

Transformation Process:

1. Convert raw text responses into embeddings using MathBERT.
2. Encode mathematical relationships and semantics into 385-dimensional vectors for downstream ML tasks.

Conclusion:

- This project demonstrates the feasibility of automating the evaluation of descriptive algebra responses using MathBERT embeddings and machine learning classifiers.
- Among the tested models, SVM consistently outperformed others, proving to be the most effective for this task.
- Dimensionality reduction through feature selection further enhanced the consistency of the results.