Machine Learning – Assignment 2(KNN Prediction)

Ganesh G (25MML0043)

https://drive.google.com/drive/u/3/folders/1mJeHBra-Ux5RHaG8-ymCq7bd67hj4NzU

KNN - K-Nearest Neighbors.

AIM:

To develop and simulate an online learning model for wine quality prediction that can incrementally update its parameters as new samples arrive, while effectively handling missing values and adapting to concept drift in wine quality standards over time.

INTRODUCTION:

In traditional machine learning systems, models are trained once on a fixed dataset and then deployed for prediction. However, in many real-world scenarios, data arrives continuously, and the underlying patterns may change over time — a phenomenon known as concept drift. Wine quality prediction is one such domain where continuous updates are valuable. New wine samples, changes in production techniques, and evolving consumer preferences can all alter the relationships between chemical features and perceived quality.

REAL WORLD APPLICATIONS:

Dynamic Quality Control in Food & Beverage

- Similar techniques can be applied to beer brewing, coffee roasting, or dairy production, where chemical properties shift due to seasonal changes in raw materials.

Agricultural and Vineyard Management

- Continuous learning models can process weather, soil, and grape growth data to predict future wine quality and guide harvesting decisions.

Supply Chain Optimization

- Distributors can forecast quality degradation over time, helping decide optimal storage conditions and delivery schedules.

Fraud Detection in Wine Industry

- Models can be trained to detect anomalies in chemical profiles, helping prevent counterfeit wine production.

ALGORITHM:

1. Import the necessary libraries: pandas, numpy, matplotlib, seaborn, and modules from sklearn.

2. Load the dataset using loading function.

3. Check for missing values and Handle missing values using imputation techniques.

4. Apply different scalers and observe how different scaling methods impact model performance.

5.Split data into training and testing sets(e.g., 80% train, 20% test).

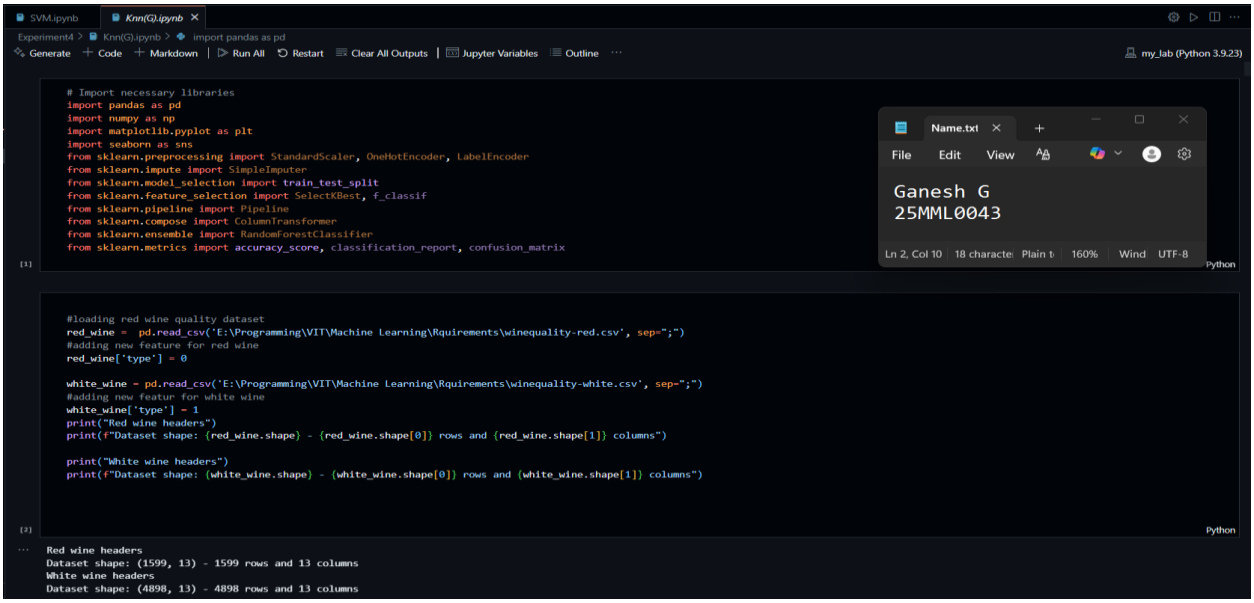
6. Initialize KNN classifier and train the model on the training set.

7.Predict the quality of wine on the test set and Evaluate model using metrics such as Accuracy, Confusion Matrix.

8. Compare performance across different scaling methods and identify which scaler works best with KNN for this dataset.

Implementation and results

1. Load dataset



2. Checking missing values

```
# Check data types and missing values
print("Dataset information:")
white_wine.info()

# Get statistical summary of numerical features
print("\nSummary statistics of numerical features:")
white_wine.describe()
```

[4] Python

```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64
 12  type                  4898 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 497.6 KB

Summary statistics of numerical features:
```

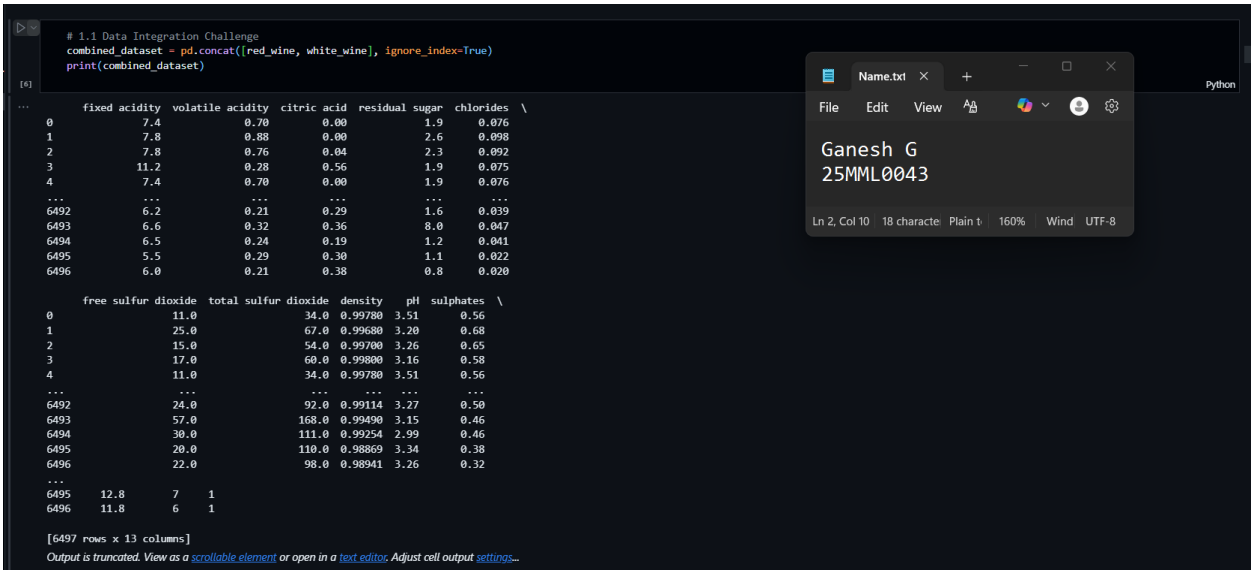
Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 160% | Wind | UTF-8

3. Data Preprocessing



```
# 1.1 Data Integration Challenge
combined_dataset = pd.concat([red_wine, white_wine], ignore_index=True)
print(combined_dataset)
```

[6] Python

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4              0.70         0.00             1.9      0.076
1               7.8              0.88         0.00             2.6      0.098
2               7.8              0.76         0.04             2.3      0.092
3              11.2              0.28         0.56             1.9      0.075
4               7.4              0.70         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
6492            6.2              0.21         0.29             1.6      0.039
6493            6.6              0.32         0.36             8.0      0.047
6494            6.5              0.24         0.19             1.2      0.041
6495            5.5              0.29         0.30             1.1      0.022
6496            6.0              0.21         0.38             0.8      0.020

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
6492                 24.0                  92.0  0.99114  3.27       0.50
6493                 57.0                 168.0  0.99490  3.15       0.46
6494                 30.0                 111.0  0.99254  2.99       0.46
6495                 20.0                 110.0  0.98869  3.34       0.38
6496                 22.0                  98.0  0.98941  3.26       0.32
...
6495     12.8       7     1
6496     11.8       6     1

[6497 rows x 13 columns]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 160% | Wind | UTF-8

```python
#missing value simulation
# MCAR - 5% missing in citric_acid
np.random.seed(42)
mcar_indices = combined_dataset.sample(frac=0.05).index
combined_dataset.loc[mcar_indices, 'citric acid'] = np.nan
```

```python
# MAR - 3% missing in sulphates where quality < 5
mar_condition = combined_dataset[combined_dataset['quality'] < 5]
mar_indices = mar_condition.sample(frac=0.03, random_state=42).index
combined_dataset.loc[mar_indices, 'sulphates'] = np.nan

print("checking missingness")
print(combined_dataset.isnull().sum())
```

```
checking missingness
fixed acidity           0
volatile acidity        0
citric acid           325
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               7
alcohol                 0
quality                 0
type                    0
dtype: int64
```

Name.txt
File Edit View
Ganesh G
25MML0043
Ln 2, Col 10 18 characte Plain t 160% Wind UTF-8

4. Imputation strategies

```python
#imputation startergies
#mean imputation
mean_imputed = combined_dataset.copy()
mean_imputed['citric acid'] = mean_imputed['citric acid'].fillna(mean_imputed['citric acid'].mean())
mean_imputed['sulphates'] = mean_imputed['sulphates'].fillna(mean_imputed['sulphates'].mean())
```

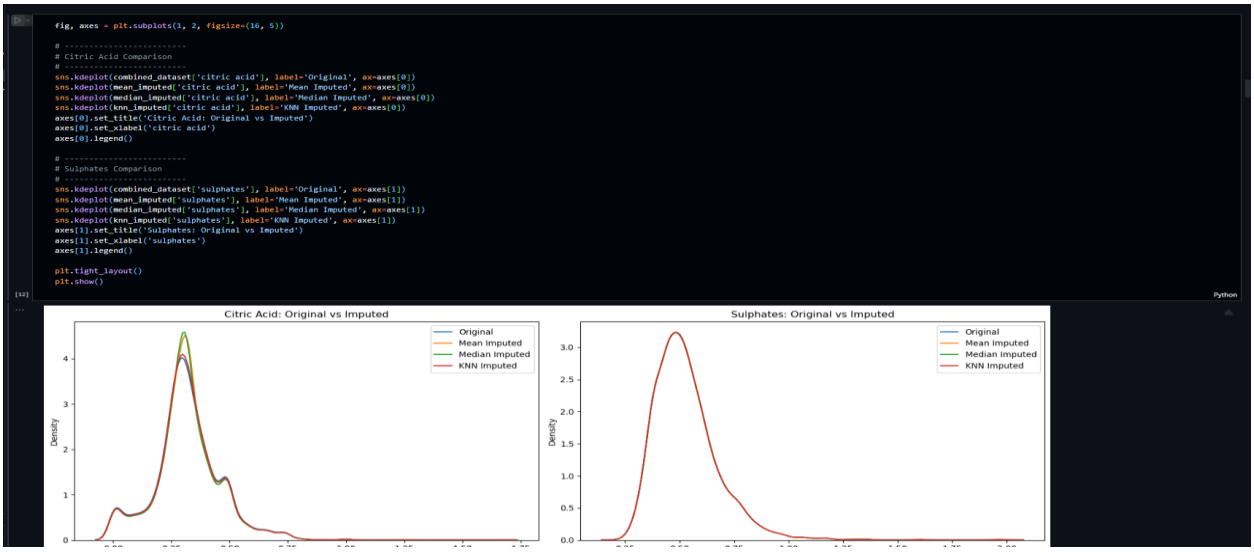
```python
#median imputation
median_imputed = combined_dataset.copy()
median_imputed['citric acid'] = median_imputed['citric acid'].fillna(median_imputed['citric acid'].median())
median_imputed['sulphates'] = median_imputed['sulphates'].fillna(median_imputed['sulphates'].median())
```

```python
from sklearn.impute import KNNImputer


knn_imputer = KNNImputer(n_neighbors=5)
knn_imputed = combined_dataset.copy()
# Only numeric columns for KNN
knn_imputed.iloc[:, :] = knn_imputer.fit_transform(knn_imputed)
```

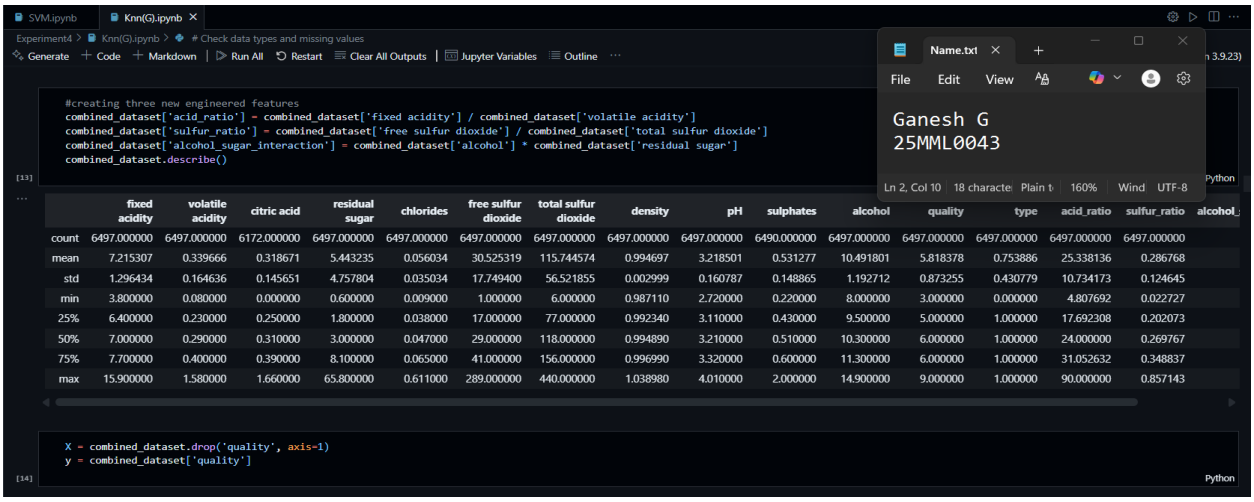
Name.txt
File Edit View
Ganesh G
25MML0043
Ln 2, Col 10 18 characte Plain t 160% Wind UTF-8

5. Comparison

```python
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# ------------------------
# Citric Acid Comparison
# ------------------------
sns.kdeplot(combined_dataset['citric acid'], label='Original', ax=axes[0])
sns.kdeplot(mean_imputed['citric acid'], label='Mean Imputed', ax=axes[0])
sns.kdeplot(median_imputed['citric acid'], label='Median Imputed', ax=axes[0])
sns.kdeplot(knn_imputed['citric acid'], label='KNN Imputed', ax=axes[0])
axes[0].set_title('Citric Acid: Original vs Imputed')
axes[0].set_xlabel('citric acid')
axes[0].legend()

# ------------------------
# Sulphates Comparison
# ------------------------
sns.kdeplot(combined_dataset['sulphates'], label='Original', ax=axes[1])
sns.kdeplot(mean_imputed['sulphates'], label='Mean Imputed', ax=axes[1])
sns.kdeplot(median_imputed['sulphates'], label='Median Imputed', ax=axes[1])
sns.kdeplot(knn_imputed['sulphates'], label='KNN Imputed', ax=axes[1])
axes[1].set_title('Sulphates: Original vs Imputed')
axes[1].set_xlabel('sulphates')
axes[1].legend()

plt.tight_layout()
plt.show()
```



6. New Engineered Features



7. Scale data and train KNN

8. Comparing different Scalars

```python
#Comparing the performance of different scalers using KNN
import matplotlib.pyplot as plt

# Use the 'results' dictionary from previous step
scaler_names = list(results.keys())
# Take the mean accuracy for each scaler
accuracies = [np.mean(acc_list) for acc_list in results.values()]

plt.figure(figsize=(8, 5))
bars = plt.bar(scaler_names, accuracies, color=['skyblue', 'lightgreen', 'salmon'])
plt.title('KNN Accuracy with Different Scalers')
plt.ylabel('Accuracy')
plt.ylim(0, 1)

# Annotate bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, f'{yval:.4f}', ha='center', va='bottom')

plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

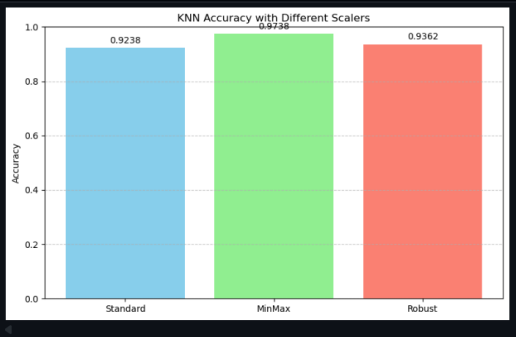
Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 160% | Wind | UTF-8

[17] Python



9. Analyze quality Distribution

```python
# 2. Quality Distribution by Wine Type
plt.figure(figsize=(10, 5))
sns.countplot(data=combined_dataset, x='quality', hue='type', palette='pastel')
plt.title('Wine Quality Distribution by Type (Red vs White)')
plt.xlabel('Quality Score')
plt.ylabel('Count')
plt.legend(title='Wine Type', labels=['Red', 'White'])
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

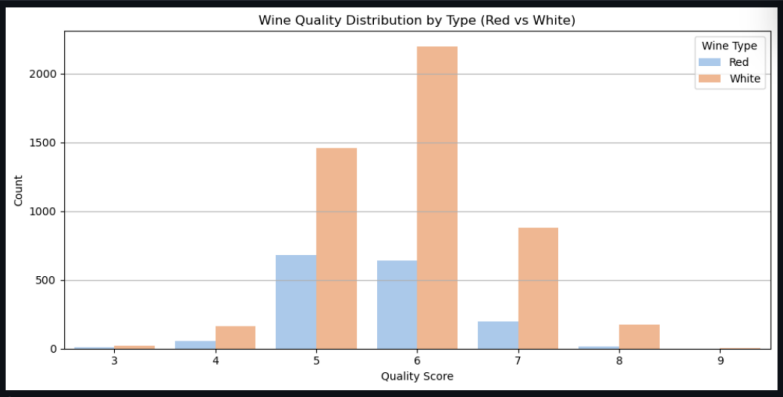
Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 160% | Wind | UTF-8

[19] Python



```python
from sklearn.feature_selection import mutual_info_classif
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'mean_imputed' is your clean dataset with 'quality' as the target
# Drop non-feature columns
X = mean_imputed.drop(columns=['quality', 'type'])  # keep only numerical features
y = mean_imputed['quality']

# Compute Mutual Information
mi_scores = mutual_info_classif(X, y, random_state=42)

# Create DataFrame for visualization
mi_df = pd.DataFrame({
    'Feature': X.columns,
    'MI Score': mi_scores
}).sort_values(by='MI Score', ascending=False)

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=mi_df, x='MI Score', y='Feature', color='skyblue')
plt.title('Mutual Information Feature Importance (All Wines)')
plt.tight_layout()
plt.show()
```

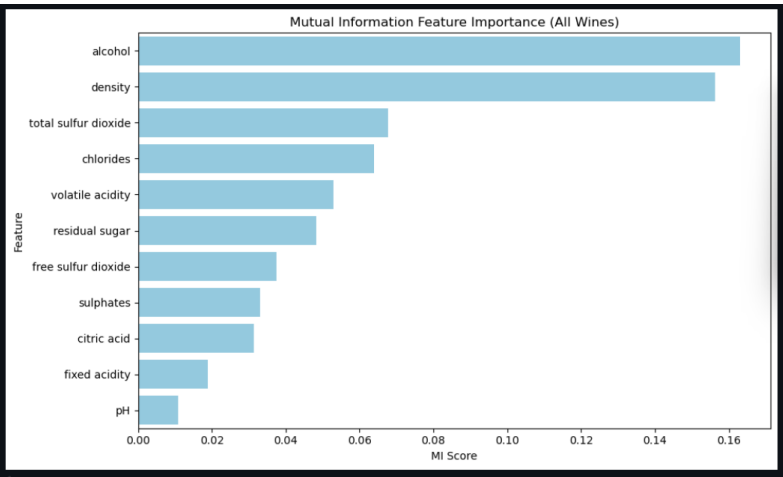
[20] Python

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 18 characte Plain t 160% Wind UTF-8



Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 18 characte Plain t 160% Wind UTF-8

```python
#Challenge: How do outliers affect KNN? Demonstrate with examples.
from sklearn.feature_selection import mutual_info_classif
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Split the dataset by wine type
red_wine = mean_imputed[mean_imputed['type'] == 0]
white_wine = mean_imputed[mean_imputed['type'] == 1]

# Define features and target
X_red = red_wine.drop(columns=['quality', 'type'])
y_red = red_wine['quality']

X_white = white_wine.drop(columns=['quality', 'type'])
y_white = white_wine['quality']

# Compute mutual information for each
mi_red = mutual_info_classif(X_red, y_red, random_state=42)
mi_white = mutual_info_classif(X_white, y_white, random_state=42)

# Create DataFrames
mi_red_df = pd.DataFrame({'Feature': X_red.columns, 'MI Score': mi_red, 'Wine Type': 'Red'})
mi_white_df = pd.DataFrame({'Feature': X_white.columns, 'MI Score': mi_white, 'Wine Type': 'White'})

# Combine both
mi_combined = pd.concat([mi_red_df, mi_white_df])
mi_combined.sort_values(by='MI Score', ascending=False, inplace=True)
```

[21] Python

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 18 characte Plain t 160% Wind UTF-8

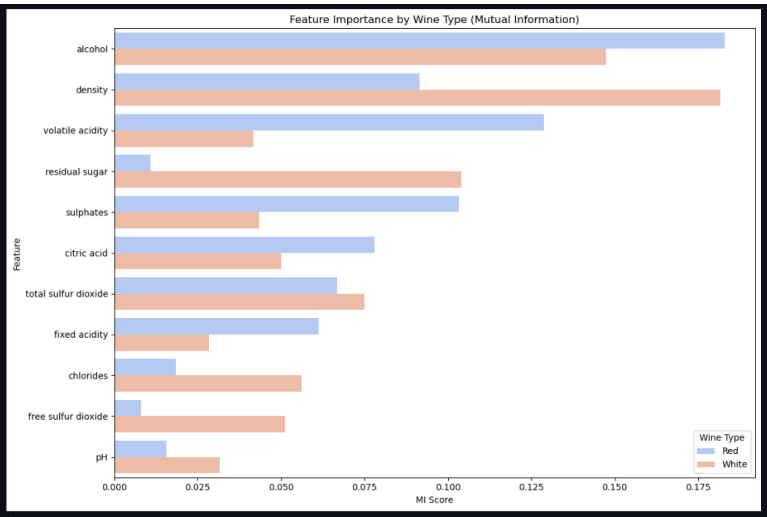
```python
plt.figure(figsize=(12, 8))
sns.barplot(data=mi_combined, x='MI Score', y='Feature', hue='Wine Type', palette='coolwarm')
plt.title('Feature Importance by Wine Type (Mutual Information)')
plt.tight_layout()
plt.show()
```

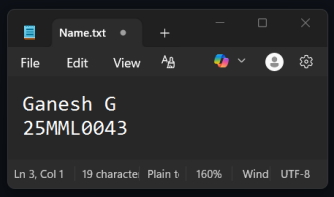
[22]

⚠ The version of Python associated with the selected kernel is no longer supported. Please consider selecting a different kernel.

```
plt.figure(figsize=(12, 8))
sns.barplot(data=mi_combined, x='MI Score', y='Feature', hue='Wine Type', palette='coolwarm')
plt.title('Feature Importance by Wine Type (Mutual Information)')
plt.tight_layout()
plt.show()
```

[22] Python

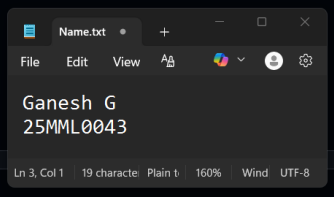




10. Model Development & Evaluation

```
# Phase 3: Model Development & Evaluation
# Implement stratified sampling to maintain quality distribution

from sklearn.model_selection import train_test_split

# Define features and target
X = mean_imputed.drop(columns=['quality'])
y = mean_imputed['quality']

# Stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,      # this ensures proportional distribution
    random_state=42
)
```

[23] Python



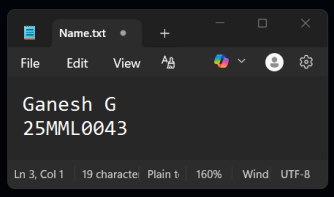

```
import pandas as pd

# Check target distribution
print("Original distribution:\n", y.value_counts(normalize=True).sort_index())
print("\nTrain distribution:\n", y_train.value_counts(normalize=True).sort_index())
print("\nTest distribution:\n", y_test.value_counts(normalize=True).sort_index())
```

[24] Python

```
Original distribution:
 quality
3    0.004618
4    0.033246
5    0.329075
6    0.436509
7    0.166077
8    0.029706
9    0.000770
Name: proportion, dtype: float64

Train distribution:
 quality
3    0.004618
4    0.033288
5    0.329036
6    0.436598
7    0.166057
8    0.029632
9    0.000770
```

```python
# Use 70-15-15 split (train-validation-test)
from sklearn.model_selection import train_test_split

# Full features and target
X = mean_imputed.drop(columns=['quality'])
y = mean_imputed['quality']

# Remove classes with only 1 sample (required for stratified split)
value_counts = y.value_counts()
valid_classes = value_counts[value_counts > 1].index
X = X[y.isin(valid_classes)]
y = y[y.isin(valid_classes)]

# Step 1: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.30,
    stratify=y,
    random_state=42
)

# Step 2: Validation (15%) and Test (15%) from Temp
# Remove classes with only 1 sample in y_temp (very rare, but safe)
temp_value_counts = y_temp.value_counts()
temp_valid_classes = temp_value_counts[temp_value_counts > 1].index
X_temp = X_temp[y_temp.isin(temp_valid_classes)]
y_temp = y_temp[y_temp.isin(temp_valid_classes)]

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.5,  # 0.5 * 30% = 15%
    stratify=y_temp,
    random_state=42
)
```

[25] Python

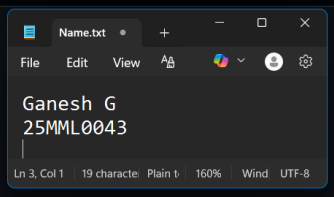



```python
print("Original:", y.value_counts(normalize=True).sort_index())
print("Train   :", y_train.value_counts(normalize=True).sort_index())
print("Val     :", y_val.value_counts(normalize=True).sort_index())
print("Test    :", y_test.value_counts(normalize=True).sort_index())
```

[26] Python

```
Original: quality
3    0.004618
4    0.033246
5    0.329075
6    0.436509
7    0.166077
8    0.029706
9    0.000770
Name: proportion, dtype: float64
Train   : quality
3    0.004618
4    0.033209
5    0.329008
6    0.436552
7    0.166044
8    0.029690
9    0.000880
Name: proportion, dtype: float64
Val     : quality
3    0.005133
4    0.032854
5    0.329569
6    0.436345
7    0.166324
8    0.029774
...
6    0.436923
7    0.166154
8    0.029744
Name: proportion, dtype: float64
```

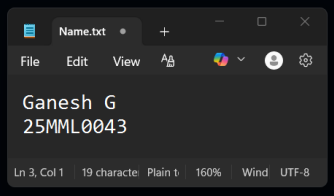
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...




```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_absolute_error, root_mean_squared_error
import numpy as np

# Use the stratified 70-15-15 split created earlier:
# X_train, X_val, X_test, y_train, y_val, y_test

# Train KNN with k=3 (Euclidean is the default distance metric)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict on validation set
y_pred = knn.predict(X_val)
# Classification metrics
acc = accuracy_score(y_val, y_pred)
prec = precision_score(y_val, y_pred, average='weighted', zero_division=0)
rec = recall_score(y_val, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_val, y_pred, average='weighted', zero_division=0)

# Regression-style metrics
mae = mean_absolute_error(y_val, y_pred)
rmse = root_mean_squared_error(y_val, y_pred)

# Print results
print("KNN Evaluation Metrics (k=3):")
print(f"Accuracy       : {acc:.4f}")
print(f"Precision      : {prec:.4f}")
print(f"Recall         : {rec:.4f}")
print(f"F1 Score       : {f1:.4f}")
print(f"MAE            : {mae:.4f}")
print(f"RMSE           : {rmse:.4f}")
```

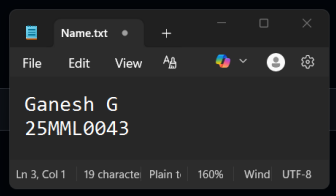
[27] Python

```
KNN Evaluation Metrics (k=3):
Accuracy       : 0.4784
Precision      : 0.4818
Recall         : 0.4784
F1 Score       : 0.4736
MAE            : 0.6622
RMSE           : 1.0005
```

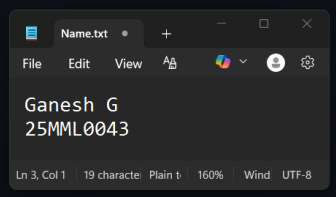


```python
# 3.3 Advanced Evaluation
# • Implement custom evaluation metric: Quality-Weighted Accuracy
import numpy as np

def quality_weighted_accuracy(y_true, y_pred):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    correct_within_1 = np.abs(y_true - y_pred) <= 1
    return np.mean(correct_within_1)

# Apply to validation set predictions
qwa = quality_weighted_accuracy(y_val, y_pred)
print(f"Quality-Weighted Accuracy (±1): {qwa:.4f}")
```

```
Quality-Weighted Accuracy (±1): 0.8819
```

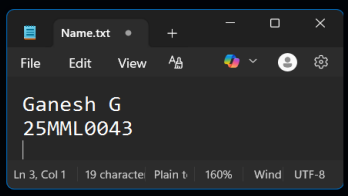


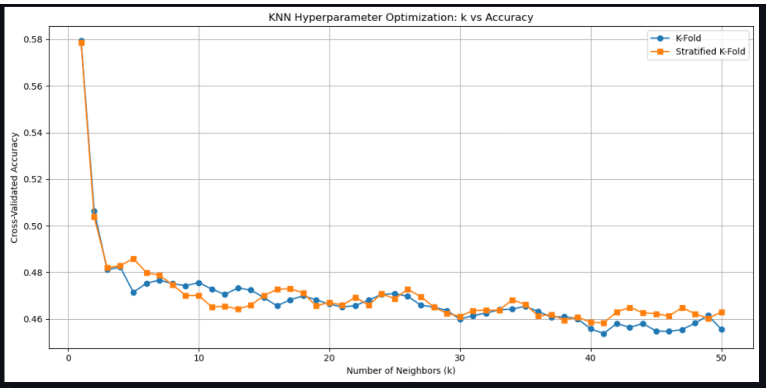

```python
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Confusion Matrix
cm = confusion_matrix(y_val, y_pred, labels=sorted(y.unique()))

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=sorted(y.unique()), yticklabels=sorted(y.unique()))
plt.title("Confusion Matrix: True vs Predicted Quality")
plt.xlabel("Predicted Quality")
plt.ylabel("Actual Quality")
plt.tight_layout()
plt.show()
```

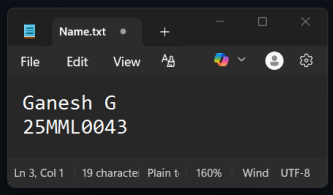


11. Hyper Parameter Optimization

```python
# Phase 4: Hyperparameter Optimization
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

import matplotlib.pyplot as plt

# Use the mean_imputed dataset and features/target as before
X = mean_imputed.drop(columns=['quality'])
y = mean_imputed['quality']

# Range of k values to test
k_range = range(1, 51)

# Prepare results storage
cv_results = {
    'KFold': [],
    'StratifiedKFold': []
}

# K-Fold Cross-Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=kf, scoring='accuracy')
    cv_results['KFold'].append(scores.mean())

# Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=skf, scoring='accuracy')
    cv_results['StratifiedKFold'].append(scores.mean())

# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(k_range, cv_results['KFold'], label='K-Fold', marker='o')
plt.plot(k_range, cv_results['StratifiedKFold'], label='Stratified K-Fold', marker='s')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Cross-Validated Accuracy')
plt.title('KNN Hyperparameter Optimization: k vs Accuracy')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Find optimal k for each strategy
best_k_kfold = np.argmax(cv_results['KFold']) + 1
best_k_stratkfold = np.argmax(cv_results['StratifiedKFold']) + 1
print(f"Best k (K-Fold): {best_k_kfold} | Accuracy: {cv_results['KFold'][best_k_kfold-1]:.4f}")
print(f"Best k (Stratified K-Fold): {best_k_stratkfold} | Accuracy: {cv_results['StratifiedKFold'][best_k_stratkfold-1]:.4f}")
```

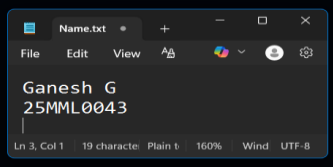






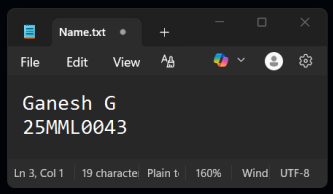

```
Best k (K-Fold): 1 | Accuracy: 0.5793
Best k (Stratified K-Fold): 1 | Accuracy: 0.5787
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split

# Use the same train/val split as before
# X_train, X_val, y_train, y_val are already defined

# Distance metrics to compare
distance_metrics = {
    'Euclidean': {'metric': 'euclidean'},
    'Manhattan': {'metric': 'manhattan'},
    'Minkowski (p=3)': {'metric': 'minkowski', 'p': 3}
}

results_dist = {}

for name, params in distance_metrics.items():
    knn = KNeighborsClassifier(n_neighbors=5, **params)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_val)
    acc = accuracy_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred, average='weighted', zero_division=0)
    results_dist[name] = {'accuracy': acc, 'f1': f1}

# Advanced: Weighted distance based on feature importance (Mutual Information)
# Use MI scores from mi_df (already computed)
# Ensure only features present in X_train are used (exclude 'type' if not present)
mi_scores_aligned = mi_df.set_index('Feature').reindex(X_train.columns)['MI Score'].fillna(0).values
# Normalize MI scores to sum to 1, then invert so higher MI = lower weight (closer)
feature_weights = 1 / (mi_scores_aligned + 1e-6)
feature_weights /= feature_weights.sum()

from scipy.spatial.distance import minkowski

def weighted_minkowski(u, v, w, p=2):
    # u, v: 1D arrays; w: 1D array of weights
    return np.power(np.sum(w * np.abs(u - v) ** p), 1/p)

# Closure to pass weights and p to the distance function
def weighted_minkowski_distance(w, p):
    def dist(u, v):
        return weighted_minkowski(u, v, w, p)
    return dist

# Create the classifier with the custom metric
knn_weighted = KNeighborsClassifier(
    n_neighbors=5,
    metric=weighted_minkowski_distance(feature_weights, 2)
)
```




```
Euclidean: Accuracy = 0.4713, F1 = 0.4574
Manhattan: Accuracy = 0.4928, F1 = 0.4781
Minkowski (p=3): Accuracy = 0.4682, F1 = 0.4522
Weighted Euclidean (MI): Accuracy = 0.4589, F1 = 0.4421
```

```python
# 4.3 Advanced KNN Variants
from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier, LocalOutlierFactor
from sklearn.metrics import accuracy_score, f1_score

# Weighted KNN (distance-based weights)
knn_weighted_dist = KNeighborsClassifier(n_neighbors=5, weights='distance')
knn_weighted_dist.fit(X_train, y_train)
y_pred_weighted_dist = knn_weighted_dist.predict(X_val)
acc_weighted_dist = accuracy_score(y_val, y_pred_weighted_dist)
f1_weighted_dist = f1_score(y_val, y_pred_weighted_dist, average='weighted', zero_division=0)

# Radius-based neighbors
radius_clf = RadiusNeighborsClassifier(radius=2.0, weights='distance', outlier_label='most_frequent')
radius_clf.fit(X_train, y_train)
y_pred_radius = radius_clf.predict(X_val)
acc_radius = accuracy_score(y_val, y_pred_radius)
f1_radius = f1_score(y_val, y_pred_radius, average='weighted', zero_division=0)

# Local Outlier Factor (for outlier detection, not classification)
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.05)
outlier_flags = lof.fit_predict(X_train)
n_outliers = (outlier_flags == -1).sum()

print(f"Weighted KNN (distance): Accuracy = {acc_weighted_dist:.4f}, F1 = {f1_weighted_dist:.4f}")
print(f"RadiusNeighborsClassifier: Accuracy = {acc_radius:.4f}, F1 = {f1_radius:.4f}")
print(f"Local Outlier Factor: Outliers detected in training set = {n_outliers}")
```




```
Weighted KNN (distance): Accuracy = 0.5811, F1 = 0.5748
RadiusNeighborsClassifier: Accuracy = 0.5883, F1 = 0.5687
Local Outlier Factor: Outliers detected in training set = 228
```

```python
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

dims = range(2, 101, 10)  # from 2 to 100 dimensions
accuracies = []

for d in dims:
    n_informative = max(2, d // 5)  # Keep informative features low (e.g., 20% of d)

    X, y = make_classification(
        n_samples=1000,
        n_features=d,
        n_informative=n_informative,
        n_redundant=0,
        n_repeated=0,
        random_state=42
    )

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))

# Plotting the drop in accuracy
plt.plot(dims, accuracies, marker='o', color='teal')
plt.xlabel("Number of Features (Dimensions)")
plt.ylabel("KNN Accuracy")
plt.title("Curse of Dimensionality on KNN Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()
```

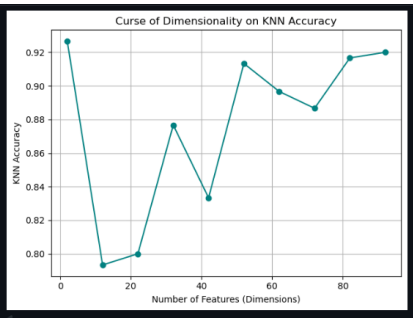
[33] Python

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 3, Col 1 | 19 characte | Plain t | 160% | Wind | UTF-8



Name.txt

File Edit View

Ganesh G
25MML0043

Ln 3, Col 1 | 19 characte | Plain t | 160% | Wind | UTF-8

```python
import time
import numpy as np

sample_sizes = [1000, 2000, 5000, 10000]
times = []

for n in sample_sizes:
    X, y = make_classification(n_samples=n, n_features=20, n_informative=10, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)

    start = time.time()
    knn.predict(X_test)
    end = time.time()

    times.append(end - start)

plt.plot(sample_sizes, times, marker='o', color='crimson')
plt.xlabel("Training Set Size")
plt.ylabel("Prediction Time (s)")
plt.title("KNN Prediction Time vs Dataset Size")
plt.grid()
plt.tight_layout()
plt.show()
```

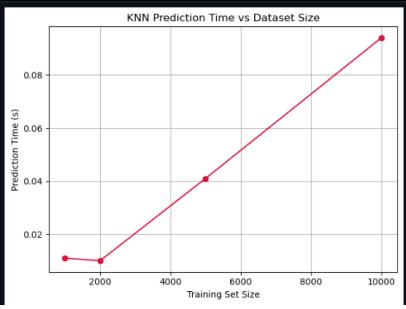
[34] Python

Name.txt

File Edit View

Ganesh G
25MML0043

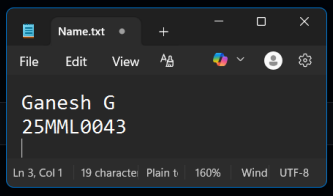
Ln 3, Col 1 | 19 characte | Plain t | 160% | Wind | UTF-8



```python
import numpy as np

n_samples = 10000
n_features = 20
dtype_size = 8  # float64

memory_MB = (n_samples * n_features * dtype_size) / (1024 ** 2)
print(f"Estimated memory usage for training data: {memory_MB:.2f} MB")
```

[35] Python

```
Estimated memory usage for training data: 1.53 MB
```

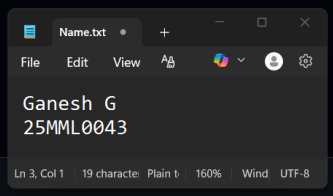



```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split
import time

# Reuse stratified train/val/test splits (assumes X, y already defined)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.15, stratify=y, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.176, stratify=y_train_val, random_state=42)  # 0.176 ≈ 15% of total

results = {}

# -------------------- KNN --------------------
start_train = time.time()
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
train_time = time.time() - start_train

start_pred = time.time()
y_pred_knn = knn.predict(X_val)
pred_time = time.time() - start_pred

results['KNN'] = {
    'accuracy': accuracy_score(y_val, y_pred_knn),
    'f1_score': f1_score(y_val, y_pred_knn, average='weighted'),
    'train_time': train_time,
    'pred_time': pred_time
}

# -------------------- Random Forest --------------------
start_train = time.time()
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
train_time = time.time() - start_train

start_pred = time.time()
y_pred_rf = rf.predict(X_val)
pred_time = time.time() - start_pred
```

[36]

```python
results['Random Forest'] = {
    'accuracy': accuracy_score(y_val, y_pred_rf),
    'f1_score': f1_score(y_val, y_pred_rf, average='weighted'),
    'train_time': train_time,
    'pred_time': pred_time
}

# -------------------- Print Results --------------------
for model, metrics in results.items():
    print(f"\n{model} Performance:")
    for metric, value in metrics.items():
        print(f"{metric.capitalize()}: {value:.4f}")
```

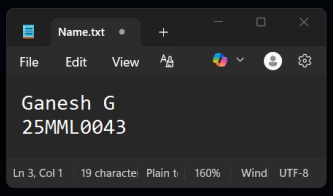
[36] Python




```
KNN Performance:
Accuracy: 0.9378
F1_score: 0.9378
Train_time: 0.0020
Pred_time: 0.0680

Random Forest Performance:
Accuracy: 0.9164
F1_score: 0.9164
Train_time: 4.1474
Pred_time: 0.0407
```

```python
from sklearn.feature_selection import SelectKBest, f_classif, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Example dataset (replace with your own)
# X should be a DataFrame; y should be a Series or array
# If X is a NumPy array, define feature names manually
feature_names = X.columns.tolist() if isinstance(X, pd.DataFrame) else [f'feature_{i}' for i in range(X.shape[1])]
if isinstance(X, np.ndarray):
    X_df = pd.DataFrame(X, columns=feature_names)
else:
    X_df = X.copy()

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_df, y, test_size=0.2, random_state=42, stratify=y)

# ---------- Feature Selection ----------
# SelectKBest (ANOVA F-test)
skb = SelectKBest(score_func=f_classif, k=10)
X_train_skb = skb.fit_transform(X_train, y_train)
X_test_skb = skb.transform(X_test)
selected_features_skb = [feature_names[i] for i in skb.get_support(indices=True)]

# RFE with Logistic Regression
rfe = RFE(estimator=LogisticRegression(max_iter=1000), n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test)
selected_features_rfe = [feature_names[i] for i in rfe.get_support(indices=True)]

print("Top 10 features by SelectKBest:", selected_features_skb)
print("Top 10 features by RFE:", selected_features_rfe)

# ---------- Train and Evaluate KNN ----------
knn = KNeighborsClassifier(n_neighbors=5)

# Evaluate on SelectKBest features
knn.fit(X_train_skb, y_train)
y_pred_skb = knn.predict(X_test_skb)
acc_skb = accuracy_score(y_test, y_pred_skb)

# Evaluate on RFE features
knn.fit(X_train_rfe, y_train)
y_pred_rfe = knn.predict(X_test_rfe)
acc_rfe = accuracy_score(y_test, y_pred_rfe)

print(f"KNN Accuracy with SelectKBest features: {acc_skb:.4f}")
print(f"KNN Accuracy with RFE features: {acc_rfe:.4f}")
```

[37]



```
Top 10 features by SelectKBest: ['feature_3', 'feature_4', 'feature_7', 'feature_8', 'feature_10', 'feature_13', 'feature_14', 'feature_15', 'feature_16', 'feature_17']
Top 10 features by RFE: ['feature_1', 'feature_3', 'feature_7', 'feature_8', 'feature_11', 'feature_13', 'feature_14', 'feature_15', 'feature_16', 'feature_17']
KNN Accuracy with SelectKBest features: 0.9100
KNN Accuracy with RFE features: 0.9330
```

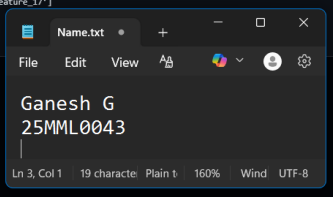
```python
import numpy as np
from sklearn.metrics import pairwise_distances

# Define weights (should align with feature order in X)
feature_weights = np.array([1.0, 2.0, 1.2, 1.5, 1.0, 1.2, 1.5, 0.8, 0.8, 2.0, 3.0])  # Example for 11 features

def custom_distance(u, v):
    return np.sqrt(np.sum(feature_weights * (u - v) ** 2))
```

[38] Python

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assume X and y are already preprocessed
# Select only the first 11 features to match feature_weights
X_selected = X[:, :11]
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, stratify=y, random_segment=42)

knn_custom = KNeighborsClassifier(n_neighbors=3, metric=custom_distance)
knn_custom.fit(X_train, y_train)
y_pred = knn_custom.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"Accuracy with custom distance: {acc:.4f}")
```

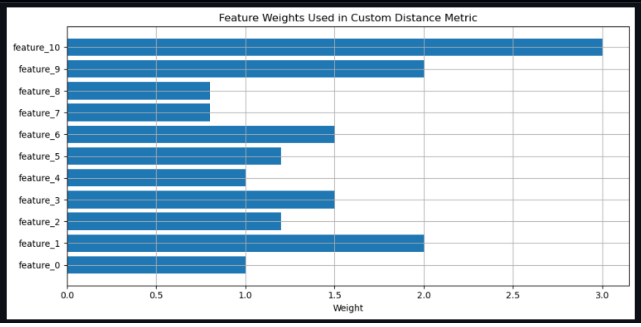
[41] Python

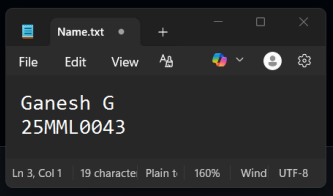
```
Accuracy with custom distance: 0.7804
```



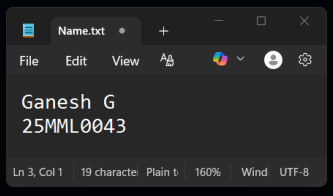

```python
import matplotlib.pyplot as plt

features = X.columns if hasattr(X, 'columns') else [f"feature_{i}" for i in range(len(feature_weights))]

plt.figure(figsize=(10, 5))
plt.barh(features, feature_weights)
plt.title("Feature Weights Used in Custom Distance Metric")
plt.xlabel("Weight")
plt.grid(True)
plt.tight_layout()
plt.show()
```

[42] Python

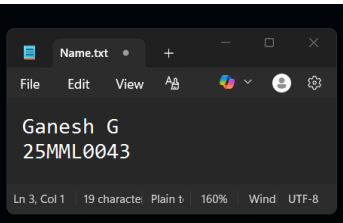





```python
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler

# Assume combined_dataset is preprocessed and features/labels are ready
# For example:
# X = combined_dataset.drop('quality', axis=1)
# y = combined_dataset['quality']

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

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define feature subsets (example: select different indexes)
feature_sets = [
    list(range(0, X.shape[1])),                # All features
    list(range(0, X.shape[1] // 2)),           # First half
    list(range(X.shape[1] // 2, X.shape[1]))   # Second half
]

# Define models with different parameters
knn1 = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn2 = KNeighborsClassifier(n_neighbors=5, metric='manhattan')
knn3 = KNeighborsClassifier(n_neighbors=7, metric='minkowski', p=3)

# Wrap KNNs with feature selection
from sklearn.base import BaseEstimator, ClassifierMixin

class FeatureSubsetKNN(BaseEstimator, ClassifierMixin):
    def __init__(self, knn, features):
        self.knn = knn
        self.features = features

    def fit(self, X, y):
        self.knn.fit(X[:, self.features], y)
        return self

    def predict(self, X):
        return self.knn.predict(X[:, self.features])
```



```python
# Build wrapped classifiers
model1 = ('knn1', FeatureSubsetKNN(knn1, feature_sets[0]))
model2 = ('knn2', FeatureSubsetKNN(knn2, feature_sets[1]))
model3 = ('knn3', FeatureSubsetKNN(knn3, feature_sets[2]))

# Ensemble using soft voting
ensemble = VotingClassifier(
    estimators=[model1, model2, model3],
    voting='hard'  # Use 'soft' if using predict_proba
)

# Fit ensemble
ensemble.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = ensemble.predict(X_test_scaled)
print("Ensemble Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

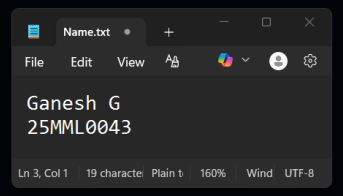


Python

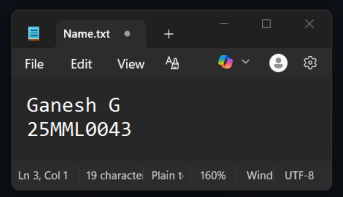
```
Ensemble Accuracy: 0.8815
              precision    recall  f1-score   support

           0       0.88      0.89      0.88      1000
           1       0.89      0.87      0.88      1000

    accuracy                           0.88      2000
   macro avg       0.88      0.88      0.88      2000
weighted avg       0.88      0.88      0.88      2000
```

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Load & prepare data
X = combined_dataset.drop('quality', axis=1)
y = combined_dataset['quality']

# Impute missing values with mean (or use median/0 as needed)
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split for streaming and test
X_stream, X_test, y_stream, y_test = train_test_split(
    X_imputed, y, test_size=0.2, stratify=y, random_state=42
)

# Scale
scaler = StandardScaler()
X_stream_scaled = scaler.fit_transform(X_stream)
X_test_scaled = scaler.transform(X_test)

# Simulate streaming (in chunks)
n_chunks = 10
chunk_size = len(X_stream_scaled) // n_chunks
model = SGDClassifier(loss='log_loss', max_iter=1, warm_start=True)

accuracies = []

for i in range(n_chunks):
    start = i * chunk_size
    end = start + chunk_size
    X_chunk = X_stream_scaled[start:end]
    y_chunk = y_stream.iloc[start:end]

    # Simulate concept drift
    if i >= n_chunks // 2:
        y_chunk = y_chunk.apply(lambda q: q - 1 if q >= 7 else q)

    # Incremental training
    model.partial_fit(X_chunk, y_chunk, classes=np.unique(y))

    # Evaluate on fixed test set
    y_pred = model.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)
    print(f"Batch {i+1}: Accuracy = {acc:.4f}")
```

[44]




```
Batch 1: Accuracy = 0.4031
Batch 2: Accuracy = 0.4023
Batch 3: Accuracy = 0.3815
Batch 4: Accuracy = 0.3980
Batch 5: Accuracy = 0.4346
Batch 6: Accuracy = 0.4900
Batch 7: Accuracy = 0.4569
Batch 8: Accuracy = 0.4808
Batch 9: Accuracy = 0.4877
Batch 10: Accuracy = 0.4500
```



SVM - Support Vector Machine.

AIM:

To implement Support Vector Machine (SVM) classifiers with different kernels for various synthetic datasets (linear, moon-shaped, and circular), perform feature scaling, tune hyperparameters using GridSearchCV, and evaluate the models using accuracy, precision, recall, F1-score, confusion matrices, and decision boundary visualization.

INTRODUCTION:

Support Vector Machine is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal separating hyperplane that maximizes the margin between classes in the feature space. If the data is not linearly separable, SVM uses the kernel trick to project data into a higher-dimensional space where separation is possible.

REAL WORLD APPLICATIONS:

SVM is widely used in various domains, including:

1. **Medical Diagnosis** – In Medical diagnosis, it is used for detecting diseases such as cancer, diabetes, heart disease.
2. **Image Recognition** –It is widely used for Face detection and object classification.
3. **Text Categorization** – Spam filtering, sentiment analysis.
4. **Bioinformatics** – Protein classification, gene expression analysis.

ALGORITHM:

1.Import the necessary libraries.

2.Load the Dataset - Import the Breast Cancer Wisconsin dataset from sklearn.datasets.

3. Explore the Data - Check number of samples, features, and class distribution.

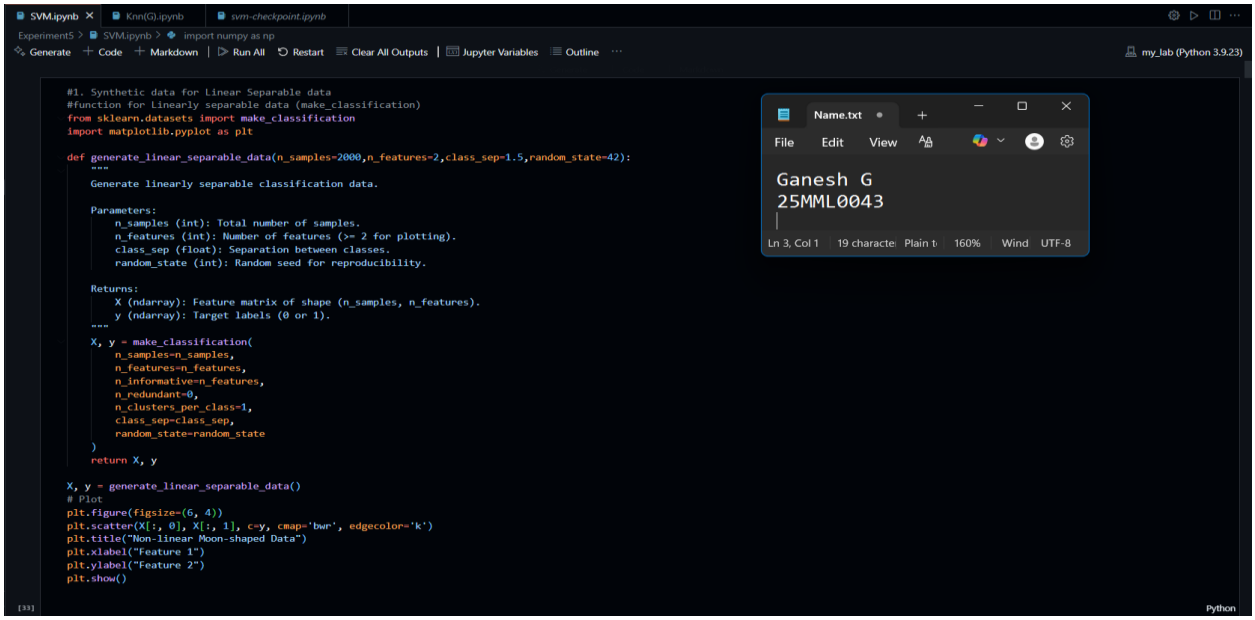
4.Split the Data and divide into training set (80%) and testing set (20%).

5. Feature Scaling - Use Standard Scaler to normalize the features for better SVM performance.

1. Choose Kernel - Use linear, poly, rbf, and sigmoid kernels for experimentation.
2. Train the SVM Model - For each kernel, fit the SVM on training data.
3. Make Predictions - Use the trained model to predict on test data.
4. Evaluate the Model - Calculate confusion matrix, accuracy, precision, recall, and F1 score.
5. Visualize Decision Boundaries - Reduce features to 2D using PCA and plot boundaries for each kernel.
6. Hyperparameter Tuning - Use GridSearchCV to find best `C` and `gamma` values for the RBF kernel.

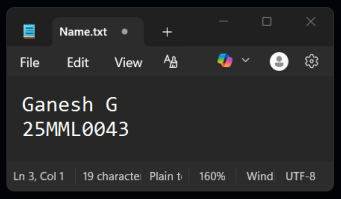
IMPLEMENTATION AND RESULTS:

1. Linear Separable data



```python
# Train SVM classifiers with different kernels: 'linear', 'poly', 'rbf', 'sigmoid'
from sklearn.svm import SVC

def train_svm_models(X_train, y_train, degree=3, C=1.0):
    """
    Train SVM classifiers with different kernels and return them in a dictionary.

    Parameters:
        X_train (ndarray): Training feature matrix.
        y_train (ndarray): Training labels.
        degree (int): Degree for polynomial kernel (default=3).
        C (float): Regularization parameter for all SVMs.

    Returns:
        dict: A dictionary where keys are kernel names and values are trained SVC models.
    """
    kernels = ['linear', 'poly', 'rbf', 'sigmoid']
    models = {}

    for kernel in kernels:
        if kernel == 'poly':
            model = SVC(kernel=kernel, degree=degree, C=C)
        else:
            model = SVC(kernel=kernel, C=C)

        model.fit(X_train, y_train)
        models[kernel] = model

    return models
```

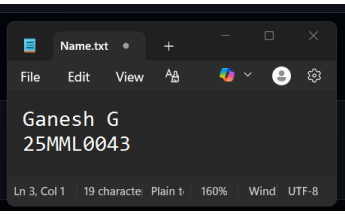
[13] Python



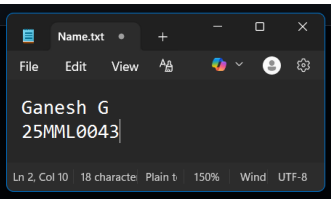

```python
svm_models = train_svm_models(X_train_scaled, y_train)
# Display trained model types
for kernel, model in svm_models.items():
    print(f"{kernel} kernel -> Trained model: {model}")
```

[14] Python

```
linear kernel -> Trained model: SVC(kernel='linear')
poly kernel -> Trained model: SVC(kernel='poly')
rbf kernel -> Trained model: SVC()
sigmoid kernel -> Trained model: SVC(kernel='sigmoid')
```




```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

def tune_svm_hyperparameters(X_train, y_train, cv=5):
    """
    Perform hyperparameter tuning for different SVM kernels using GridSearchCV.

    Parameters:
        X_train (ndarray): Training features.
        y_train (ndarray): Training labels.
        cv (int): Number of cross-validation folds.

    Returns:
        dict: Best estimator per kernel.
    """
    param_grids = {
        'linear': {
            'C': [0.1, 1, 10, 100]
        },
        'poly': {
            'C': [0.1, 1, 10],
            'degree': [2, 3, 4]
        },
        'rbf': {
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1]
        },
        'sigmoid': {
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1]
        }
    }

    best_models = {}

    for kernel, params in param_grids.items():
        svc = SVC(kernel=kernel)
        grid_search = GridSearchCV(svc, params, cv=cv, scoring='accuracy', n_jobs=-1)
        grid_search.fit(X_train, y_train)
        best_models[kernel] = grid_search.best_estimator_
        print(f"[{kernel.upper()}] Best params: {grid_search.best_params_} | Best score: {grid_search.best_score_:.4f}")

    return best_models
```



```python
def evaluate_and_visualize(models, X_train, y_train, X_test, y_test):
    """
    Evaluate SVM models on test data and visualize confusion matrix + decision boundaries.

    Parameters:
        models (dict): Dictionary of trained SVM models per kernel.
        X_train, y_train: Training set.
        X_test, y_test: Test set.
    """
    results = {}

    for kernel, model in models.items():
        print(f"\n=== {kernel.upper()} Kernel ===")

        # Predictions
        y_pred = model.predict(X_test)

        # Metrics
        acc = accuracy_score(y_test, y_pred)
        precision, recall, f1, _ = precision_recall_fscore_support(
            y_test, y_pred, average='binary', zero_division=0
        )

        results[kernel] = {
            'accuracy': acc,
            'precision': precision,
            'recall': recall,
            'f1': f1
        }

        print(f"Accuracy: {acc:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall: {recall:.4f}")
        print(f"F1-score: {f1:.4f}")

        # Confusion Matrix
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(4, 3))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=['Class 0', 'Class 1'],
                    yticklabels=['Class 0', 'Class 1'])
        plt.title(f"{kernel.upper()} Kernel - Confusion Matrix")
        plt.ylabel("True Label")
        plt.xlabel("Predicted Label")
        plt.show()

        # Decision Boundary (2D only)
        if X_train.shape[1] == 2:
            plt.figure(figsize=(5, 4))
            plot_decision_boundary(model, X_train, y_train, title=f"{kernel.upper()} Kernel")
            plt.show()
```

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

meet.google.com

```python
def plot_decision_boundary(model, X, y, title="Decision Boundary"):
    """
    Plot decision boundary for a trained SVM model with 2D features.
    Highlights support vectors.
    """
    # Create meshgrid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 300),
        np.linspace(y_min, y_max, 300)
    )

    # Predictions for grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolors='k')

    # Highlight support vectors
    plt.scatter(
        model.support_vectors_[:, 0],
        model.support_vectors_[:, 1],
        s=100, facecolors='none', edgecolors='k', linewidths=1.5, label='Support Vectors'
    )

    plt.title(title)
    plt.legend()
    plt.xlabel("Feature 1 (scaled)")
    plt.ylabel("Feature 2 (scaled)")


# ===== Example workflow =====


# Hyperparameter tuning
best_svm_models = tune_svm_hyperparameters(X_train_scaled, y_train)

# Evaluation & visualization
metrics_results = evaluate_and_visualize(best_svm_models, X_train_scaled, y_train, X_test_scaled, y_test)
```

Name.txt

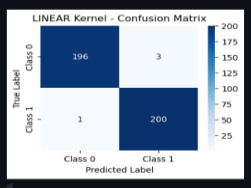
File Edit View

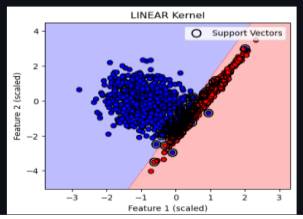
Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

Python

```
[LINEAR] Best params: {'C': 100} | Best score: 0.9838
[POLY] Best params: {'C': 10, 'degree': 3} | Best score: 0.9869
[RBF] Best params: {'C': 10, 'gamma': 1} | Best score: 0.9850
[SIGMOID] Best params: {'C': 10, 'gamma': 0.01} | Best score: 0.9731

=== LINEAR Kernel ===
Accuracy: 0.9900
Precision: 0.9852
Recall: 0.9950
F1-score: 0.9901
```





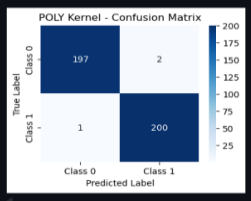
Name.txt

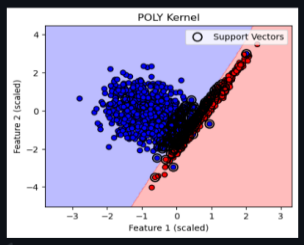
File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```
=== POLY Kernel ===
Accuracy: 0.9925
Precision: 0.9901
Recall: 0.9950
F1-score: 0.9926
```





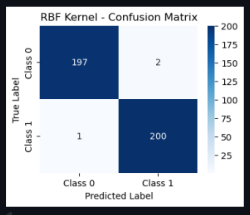
Name.txt

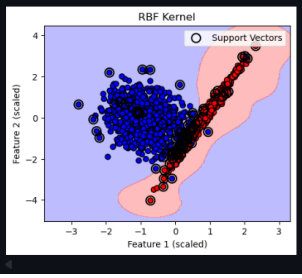
File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```
=== RBF Kernel ===
Accuracy: 0.9925
Precision: 0.9901
Recall: 0.9950
F1-score: 0.9926
```





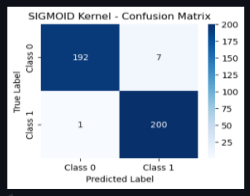
Name.txt

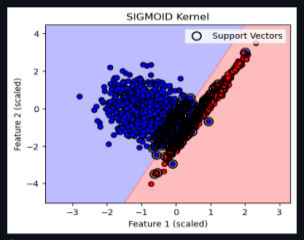
File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```
=== SIGMOID Kernel ===
Accuracy: 0.9800
Precision: 0.9662
Recall: 0.9950
F1-score: 0.9804
```





Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

2. Nonlinear moon data

```python
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

def generate_moon_data(n_samples, noise, random_state):
    """
    Generate synthetic non-linear moon-shaped classification data.

    Parameters:
        n_samples (int): Total number of samples.
        noise (float): Standard deviation of Gaussian noise added to the data.
        random_state (int): Random seed for reproducibility.

    Returns:
        X (ndarray): Feature matrix of shape (n_samples, 2).
        y (ndarray): Target labels (0 or 1).
    """
    X, y = make_moons(
        n_samples=n_samples,
        noise=noise,
        random_state=random_state
    )
    return X, y
X, y = make_moons(n_samples=2000, noise=0.2, random_state=42)
# Plot
plt.figure(figsize=(6, 4))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolor='k')
plt.title("Non-linear Moon-shaped Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

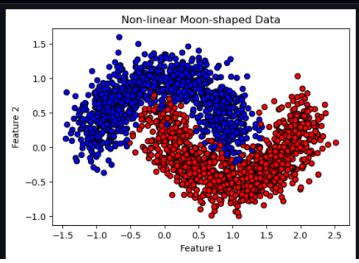
[1] ✓ 8.2s Python

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8




```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
)

#Scaling
# 📏 Standard scaling (fit on train, transform both)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[2] ✓ 0.2s Python

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```python
from sklearn.svm import SVC

def train_svm_models(X_train, y_train, degree=3, C=1.0):
    """
    Train SVM classifiers with different kernels and return them in a dictionary.

    Parameters:
        X_train (ndarray): Training features.
        y_train (ndarray): Training labels.
        degree (int): Degree for polynomial kernel (default=3).
        C (float): Regularization parameter for all SVMs.

    Returns:
        dict: Dictionary {kernel_name: trained_model}.
    """
    kernels = ['linear', 'poly', 'rbf', 'sigmoid']
    models = {}

    for kernel in kernels:
        if kernel == 'poly':
            model = SVC(kernel=kernel, degree=degree, C=C)
        else:
            model = SVC(kernel=kernel, C=C)

        model.fit(X_train, y_train)
        models[kernel] = model

    return models
```

[3] ✓ 0.2s Python

```python
svm_models = train_svm_models(X_train_scaled, y_train)
# Show trained models
for kernel, model in svm_models.items():
    print(f"{kernel} kernel -> Trained model: {model}")
```

[6] ✓ 0.7s Python

```
linear kernel -> Trained model: SVC(kernel='linear')
poly kernel -> Trained model: SVC(kernel='poly')
rbf kernel -> Trained model: SVC()
sigmoid kernel -> Trained model: SVC(kernel='sigmoid')
```

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

def tune_svm_hyperparameters(X_train, y_train, cv=5):
    """
    Perform hyperparameter tuning for different SVM kernels using GridSearchCV.

    Parameters:
        X_train (ndarray): Training features.
        y_train (ndarray): Training labels.
        cv (int): Number of cross-validation folds.

    Returns:
        dict: Best estimator per kernel.
    """
    # Define parameter grids for each kernel
    param_grids = {
        'linear': {
            'C': [0.1, 1, 10, 100]
        },
        'poly': {
            'C': [0.1, 1, 10],
            'degree': [2, 3, 4]
        },
        'rbf': {
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1]
        },
        'sigmoid': {
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1]
        }
    }

    best_models = {}

    # Loop through kernels and run GridSearch
    for kernel, params in param_grids.items():
        svc = SVC(kernel=kernel)
        grid_search = GridSearchCV(
            svc, params, cv=cv, scoring='accuracy', n_jobs=-1
        )
        grid_search.fit(X_train, y_train)

        best_models[kernel] = grid_search.best_estimator_

        print(f"[{kernel.upper()}] Best params: {grid_search.best_params_} | "
              f"Best CV score: {grid_search.best_score_:.4f}")

    return best_models
```

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 18 characte Plain t 150% Wind UTF-8

```
# Tune hyperparameters
best_svm_models = tune_svm_hyperparameters(X_train_scaled, y_train)
```

[7] ✓ 19.9s

```
[LINEAR] Best params: {'C': 1} | Best CV score: 0.8694
[POLY] Best params: {'C': 1, 'degree': 3} | Best CV score: 0.8681
[RBF] Best params: {'C': 10, 'gamma': 1} | Best CV score: 0.9719
[SIGMOID] Best params: {'C': 10, 'gamma': 0.01} | Best CV score: 0.8688
```

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 18 characte Plain t 150% Wind UTF-8

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import precision_recall_fscore_support, accuracy_score, confusion_matrix

def evaluate_and_visualize(models, X_train, y_train, X_test, y_test):
    """
    Evaluate SVM models on test data and visualize confusion matrix + decision boundaries.

    Parameters:
        models (dict): Dictionary {kernel: trained_model}.
        X_train, y_train: Training data.
        X_test, y_test: Test data.
    """
    results = {}

    for kernel, model in models.items():
        print(f"\n=== {kernel.upper()} Kernel ===")

        # Predictions
        y_pred = model.predict(X_test)

        # Metrics
        acc = accuracy_score(y_test, y_pred)
        precision, recall, f1, _ = precision_recall_fscore_support(
            y_test, y_pred, average='binary', zero_division=0
        )

        results[kernel] = {
            'accuracy': acc,
            'precision': precision,
            'recall': recall,
            'f1': f1
        }

        print(f"Accuracy: {acc:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall: {recall:.4f}")
        print(f"F1-score: {f1:.4f}")

        # Confusion Matrix
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(4, 3))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=['Class 0', 'Class 1'],
                    yticklabels=['Class 0', 'Class 1'])
        plt.title(f"{kernel.upper()} Kernel - Confusion Matrix")
        plt.ylabel("True Label")
        plt.xlabel("Predicted Label")
        plt.show()
```

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 18 characte Plain t 150% Wind UTF-8

```
        # Decision Boundary
        if X_train.shape[1] == 2:
            plt.figure(figsize=(5, 4))
            plot_decision_boundary(model, X_train, y_train,
                                   title=f"{kernel.upper()} Kernel Decision Boundary")
            plt.show()

    return results


def plot_decision_boundary(model, X, y, title="Decision Boundary"):
    """
    Plot decision boundary for a trained SVM model with 2D features.
    Highlights support vectors.
    """
    # Create meshgrid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 300),
        np.linspace(y_min, y_max, 300)
    )

    # Predictions for meshgrid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolors='k')

    # Highlight support vectors
    plt.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=100, facecolors='none', edgecolors='k',
                linewidths=1.5, label='Support Vectors')

    plt.title(title)
    plt.xlabel("Feature 1 (scaled)")
    plt.ylabel("Feature 2 (scaled)")
    plt.legend()
# Evaluate & visualize
results = evaluate_and_visualize(best_svm_models, X_train_scaled, y_train, X_test_scaled, y_test)
```

30.5s Python

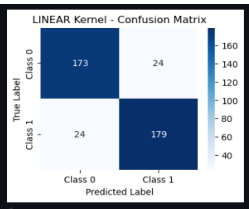
```
=== LINEAR Kernel ===
Accuracy:  0.8800
Precision: 0.8818
Recall:    0.8818
F1-score:  0.8818
```

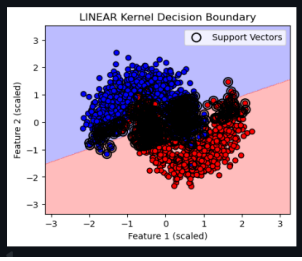
Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8





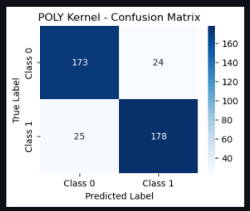
Name.txt

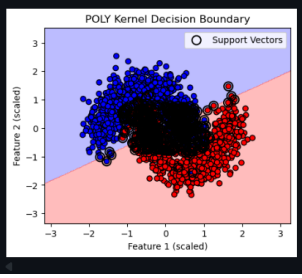
File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```
=== POLY Kernel ===
Accuracy:  0.8775
Precision: 0.8812
Recall:    0.8768
F1-score:  0.8790
```



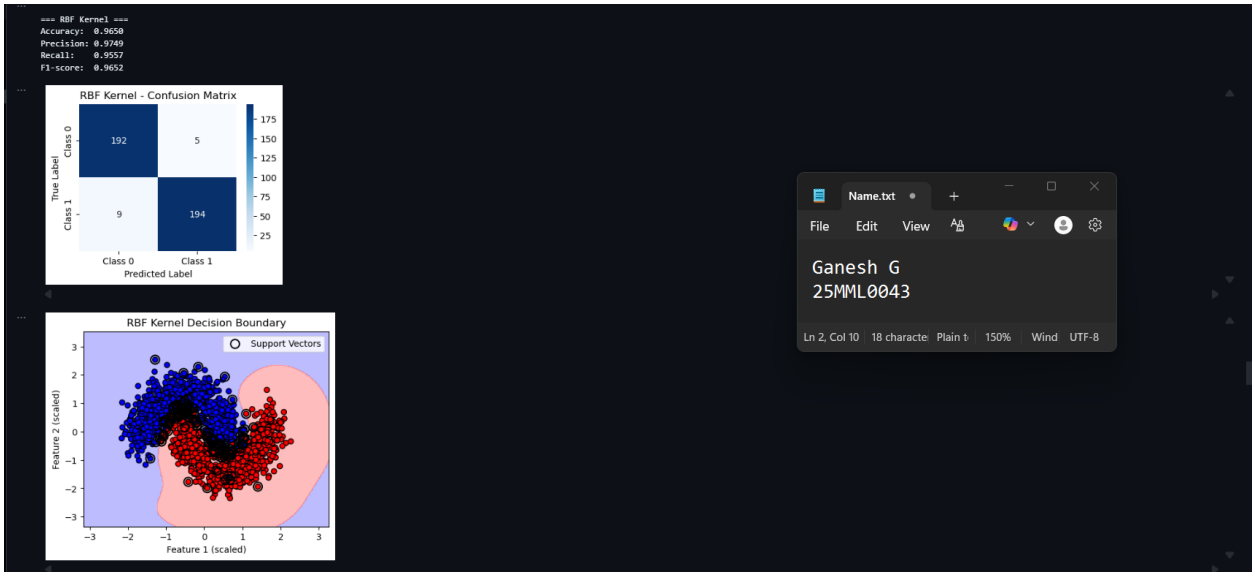


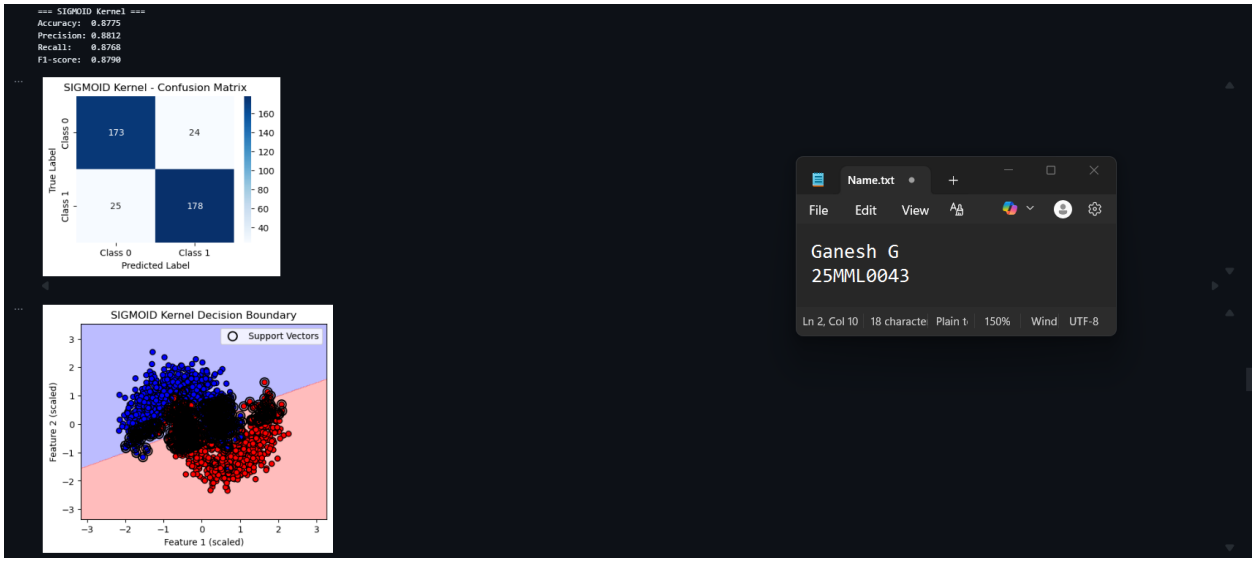
Name.txt

File Edit View

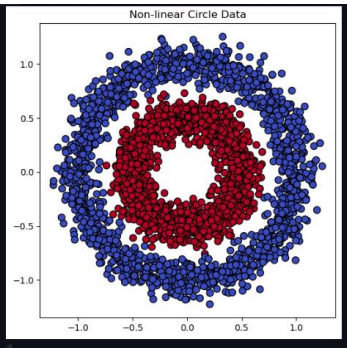
Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8





3. Nonlinear Circular Data



Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```python
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Scale features (fit on train, transform on both train/test)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[15] ✓ 0.0s Python

```python
svm_models = train_svm_models(X_train_scaled, y_train)
# Show trained models
for kernel, model in svm_models.items():
    print(f"{kernel} kernel -> Trained model: {model}")
```

[18] ✓ 2.6s Python

```
linear kernel -> Trained model: SVC(kernel='linear')
poly kernel -> Trained model: SVC(kernel='poly')
rbf kernel -> Trained model: SVC()
sigmoid kernel -> Trained model: SVC(kernel='sigmoid')
```

Name.txt

File Edit View

Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

def tune_svm_hyperparameters(X_train, y_train, cv=5):
    """
    Perform hyperparameter tuning for different SVM kernels using GridSearchCV.

    Parameters:
        X_train (ndarray): Training features.
        y_train (ndarray): Training labels.
        cv (int): Number of cross-validation folds.

    Returns:
        dict: Best estimator per kernel.
    """
    # Define parameter grids for each kernel
    param_grids = {
        'linear': {
            'C': [0.1, 1, 10, 100]
        },
        'poly': {
            'C': [0.1, 1, 10],
            'degree': [2, 3, 4]
        },
        'rbf': {
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1]
        },
        'sigmoid': {
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1]
        }
    }

    best_models = {}
```

SVM.ipynb | Knn(G).ipynb | svm-checkpoint.ipynb

Experiment5 > SVM.ipynb > #Function for Non-linear circle data (make_circles)

Generate | Code | Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline

my_lab (Python 3.9.23)

```python
    # Loop through kernels and run GridSearch
    for kernel, params in param_grids.items():
        svc = SVC(kernel=kernel)
        grid_search = GridSearchCV(
            svc, params, cv=cv, scoring='accuracy', n_jobs=-1
        )
        grid_search.fit(X_train, y_train)

        best_models[kernel] = grid_search.best_estimator_

        print(f"[{kernel.upper()}] Best params: {grid_search.best_params_} | "
              f"Best CV score: {grid_search.best_score_:.4f}")

    return best_models


# Tune hyperparameters
best_svm_models = tune_svm_hyperparameters(X_train_scaled, y_train)
```

[19] ✓ 22.8s Python

```
[LINEAR] Best params: {'C': 1} | Best CV score: 0.5379
[POLY] Best params: {'C': 10, 'degree': 2} | Best CV score: 0.9979
[RBF] Best params: {'C': 10, 'gamma': 0.01} | Best CV score: 0.9979
[SIGMOID] Best params: {'C': 10, 'gamma': 0.1} | Best CV score: 0.6275
```

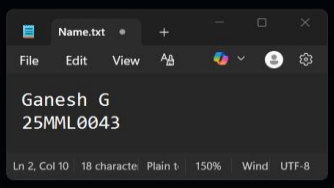
Name.txt

File Edit View

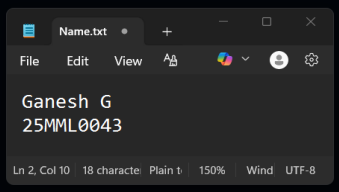
Ganesh G
25MML0043

Ln 2, Col 10 | 18 characte | Plain t | 150% | Wind | UTF-8

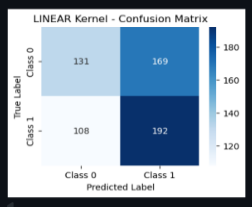
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import precision_recall_fscore_support, accuracy_score, confusion_matrix

def evaluate_and_visualize(models, X_train, y_train, X_test, y_test):
    """
    Evaluate SVM models on test data and visualize confusion matrix + decision boundaries.

    Parameters:
        models (dict): Dictionary {kernel: trained_model}.
        X_train, y_train: Training data.
        X_test, y_test: Test data.
    """
    results = {}

    for kernel, model in models.items():
        print(f"\n=== {kernel.upper()} Kernel ===")

        # Predictions
        y_pred = model.predict(X_test)

        # Metrics
        acc = accuracy_score(y_test, y_pred)
        precision, recall, f1, _ = precision_recall_fscore_support(
            y_test, y_pred, average='binary', zero_division=0
        )

        results[kernel] = {
            'accuracy': acc,
            'precision': precision,
            'recall': recall,
            'f1': f1
        }

        print(f"Accuracy:  {acc:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall:    {recall:.4f}")
        print(f"F1-score:  {f1:.4f}")

        # Confusion Matrix
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(4, 3))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=['Class 0', 'Class 1'],
                    yticklabels=['Class 0', 'Class 1'])
        plt.title(f"{kernel.upper()} Kernel - Confusion Matrix")
        plt.ylabel("True Label")
        plt.xlabel("Predicted Label")
        plt.show()
```

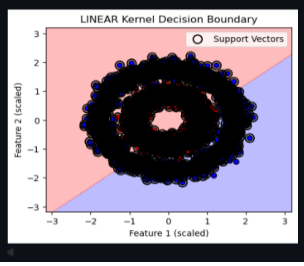


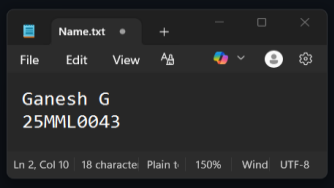

```python
        # Decision Boundary
        if X_train.shape[1] == 2:
            plt.figure(figsize=(5, 4))
            plot_decision_boundary(model, X_train, y_train,
                                   title=f"{kernel.upper()} Kernel Decision Boundary")
            plt.show()

    return results


def plot_decision_boundary(model, X, y, title="Decision Boundary"):
    """
    Plot decision boundary for a trained SVM model with 2D features.
    Highlights support vectors.
    """
    # Create meshgrid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 300),
        np.linspace(y_min, y_max, 300)
    )

    # Predictions for meshgrid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolors='k')

    # Highlight support vectors
    plt.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=100, facecolors='none', edgecolors='k',
                linewidths=1.5, label='Support Vectors')

    plt.title(title)
    plt.xlabel("Feature 1 (scaled)")
    plt.ylabel("Feature 2 (scaled)")
    plt.legend()
# Evaluate & visualize
results = evaluate_and_visualize(best_svm_models, X_train_scaled, y_train, X_test_scaled, y_test)
```



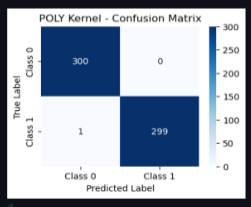

```
=== LINEAR Kernel ===
Accuracy:  0.5383
Precision: 0.5319
Recall:    0.6400
F1-score:  0.5809
```

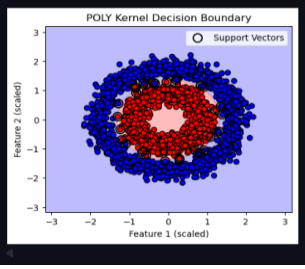


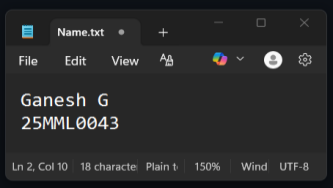




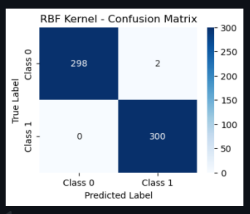
```
=== POLY Kernel ===
Accuracy:  0.9983
Precision: 1.0000
Recall:    0.9967
F1-score:  0.9983
```

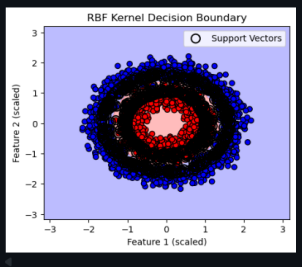


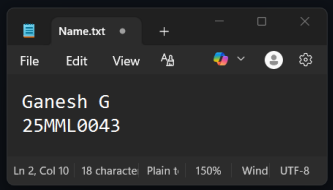





```
=== RBF Kernel ===
Accuracy:  0.9967
Precision: 0.9934
Recall:    1.0000
F1-score:  0.9967
```








```
=== SIGMOID Kernel ===
Accuracy:  0.6333
Precision: 0.6064
Recall:    0.7600
F1-score:  0.6746
```