Multiclass classification using SVM

April 22, 2024

1 Support Vector Machine (SVM) - Multiclass Classification:

2 Assignment 4 (Multiclassification of urban land using Support Vector Machines)

[1]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Load the datasets
train_data = pd.read_csv("train_data.csv")
test_data = pd.read_csv("test_data.csv")
```

[2]:
```python
# Remove rows with missing data
train_data.dropna(inplace=True)
test_data.dropna(inplace=True)
# Separate target variable from features
X_train = train_data.drop(columns=["class"])
y_train = train_data["class"]

X_test = test_data.drop(columns=["class"])
y_test = test_data["class"]

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[31]:
```python
# Display the shapes of the train and test data
print("Train shape:", train_data.shape)
print("Test shape:", test_data.shape)
```

```
Train shape: (507, 148)
Test shape: (168, 148)
```

3 2. Random Forest Classifier - Base Model:

[32]:
```
from sklearn.ensemble import RandomForestClassifier

# RandomForestClassifier with default parameter
rf_classifier = RandomForestClassifier()

# Fit the model train data
rf_classifier.fit(X_train, y_train)

# Predicting on the test data
y_pred_test = rf_classifier.predict(X_test)
```

[6]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Calculating confusion matrix
conf_matrix_test = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix (Test Data):\n", conf_matrix_test)

# Calculating classification report
class_report_test = classification_report(y_test, y_pred_test)
print("\nClassification Report (Test Data):\n", class_report_test)
```

```
Confusion Matrix (Test Data):
 [[14  0  0  0  0  0  0  0  0]
 [ 1 22  0  2  0  0  0  0  0]
 [ 0  1 13  0  0  1  0  0  0]
 [ 0  5  0 18  0  0  0  0  0]
 [ 0  0  0  0 26  0  0  0  3]
 [ 1  0  1  0  0 13  0  0  0]
 [ 3  0  0  0  0  0 13  0  0]
 [ 0  1  0  5  2  0  0  6  0]
 [ 0  0  0  1  1  0  0  0 15]]

Classification Report (Test Data):
               precision    recall  f1-score   support

     asphalt       0.74      1.00      0.85        14
    building       0.76      0.88      0.81        25
         car       0.93      0.87      0.90        15
    concrete       0.69      0.78      0.73        23
       grass       0.90      0.90      0.90        29
        pool       0.93      0.87      0.90        15
      shadow       1.00      0.81      0.90        16
        soil       1.00      0.43      0.60        14
        tree       0.83      0.88      0.86        17

    accuracy                           0.83       168
```

```
   macro avg       0.86      0.82      0.83       168
weighted avg       0.85      0.83      0.83       168
```

[7]:
```
# Predicting on the training data
y_pred_train = rf_classifier.predict(X_train)

# Calculating confusion matrix for training data
conf_matrix_train = confusion_matrix(y_train, y_pred_train)
print("Confusion Matrix (Training Data):\n", conf_matrix_train)

# Calculating classification report for training data
class_report_train = classification_report(y_train, y_pred_train)
print("\nClassification Report (Training Data):\n", class_report_train)
```

```
Confusion Matrix (Training Data):
 [[45  0  0  0  0  0  0  0  0]
 [ 0 97  0  0  0  0  0  0  0]
 [ 0  0 21  0  0  0  0  0  0]
 [ 0  0  0 93  0  0  0  0  0]
 [ 0  0  0  0 83  0  0  0  0]
 [ 0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0 45  0  0]
 [ 0  0  0  0  0  0  0 20  0]
 [ 0  0  0  0  0  0  0  0 89]]

Classification Report (Training Data):
               precision    recall  f1-score   support

     asphalt       1.00      1.00      1.00        45
    building       1.00      1.00      1.00        97
         car       1.00      1.00      1.00        21
    concrete       1.00      1.00      1.00        93
       grass       1.00      1.00      1.00        83
        pool       1.00      1.00      1.00        14
      shadow       1.00      1.00      1.00        45
        soil       1.00      1.00      1.00        20
        tree       1.00      1.00      1.00        89

    accuracy                           1.00       507
   macro avg       1.00      1.00      1.00       507
weighted avg       1.00      1.00      1.00       507
```

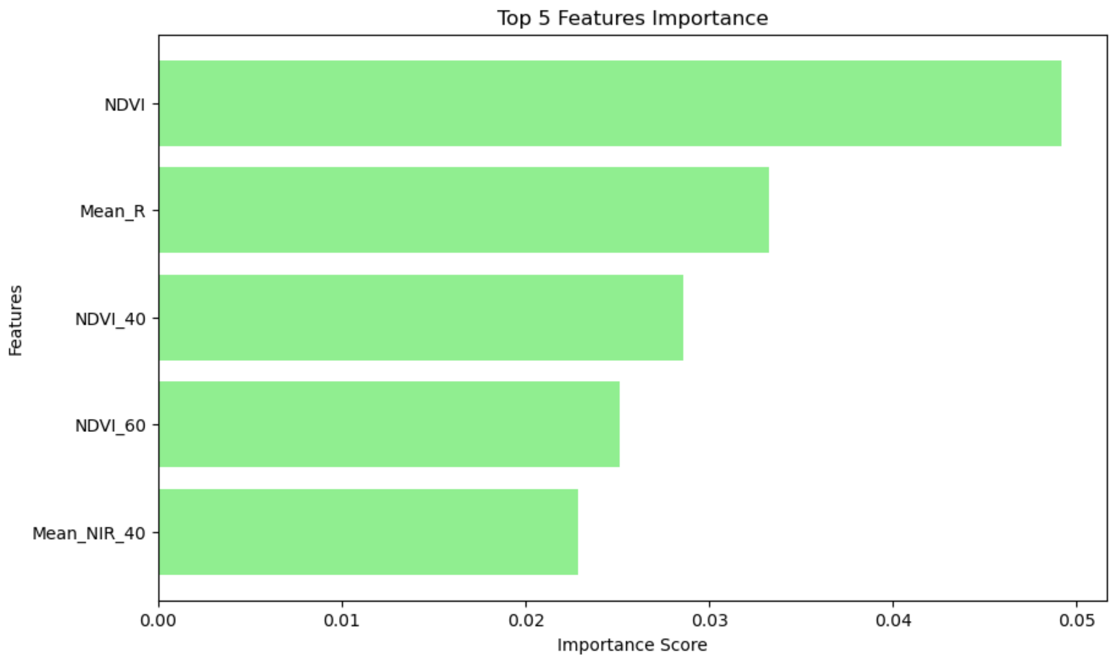
[8]:
```
import matplotlib.pyplot as plt

# Get feature importances
feature_importances = rf_classifier.feature_importances_
```

```
top5_indices = feature_importances.argsort()[-5:][::-1]

top5_features = X_train.columns[top5_indices]
print("Top 5 Features:", top5_features)
top5_importances = feature_importances[top5_indices]
# Plotting top 5 features
plt.figure(figsize=(10, 6))
plt.barh(top5_features, top5_importances, color='lightgreen')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.title('Top 5 Features Importance')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at␣
↪the top
plt.show()
```

```
Top 5 Features: Index(['NDVI', 'Mean_R', 'NDVI_40', 'NDVI_60', 'Mean_NIR_40'],
dtype='object')
```



4 3. LinearSVM Classifier - Base Model:

[9]:
```
# Suppressing FutureWarning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

[10]:
```python
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, classification_report

# LinearSVC classifier with 10000 max_iter
linear_svc = LinearSVC(max_iter=10000)  # You can adjust the value of max_iter␣
 ↪as needed

# Fitting the model on the training data
linear_svc.fit(X_train, y_train)

# Use the fitted model to predict on test data
y_pred_test = linear_svc.predict(X_test)
# Calculate predictions for the training data
y_pred_train = linear_svc.predict(X_train)
```

```
C:\Users\arpan\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1242:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
```

[11]:
```python
# Calculating confusion matrix for test data
conf_matrix_test = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix (Test Data):")
print(conf_matrix_test)

# Calculating the classification report for test data
class_report_test = classification_report(y_test, y_pred_test)
print("\nClassification Report (Test Data):")
print(class_report_test)


# Calculatinge confusion matrix for training data
conf_matrix_train = confusion_matrix(y_train, y_pred_train)
print("\nConfusion Matrix (Training Data):")
print(conf_matrix_train)

# Calculating classification report for training data
class_report_train = classification_report(y_train, y_pred_train)
print("\nClassification Report (Training Data):")
print(class_report_train)
```

```
Confusion Matrix (Test Data):
[[ 2  0  0  2  7  0  3  0  0]
 [ 0 20  0  2  2  1  0  0  0]
 [ 0  3  9  1  0  0  1  1  0]
 [ 0  3  1  6 10  3  0  0  0]
 [ 0  0  0  0 27  0  0  0  2]
 [ 1  0  3  0  0 10  1  0  0]
```

```
 [ 0  0  0  0  0  0 14  0  2]
 [ 0  1  2  2  8  1  0  0  0]
 [ 0  0  0  0  5  0  0  0 12]]

Classification Report (Test Data):
              precision    recall  f1-score   support

     asphalt       0.67      0.14      0.24        14
    building       0.74      0.80      0.77        25
         car       0.60      0.60      0.60        15
    concrete       0.46      0.26      0.33        23
       grass       0.46      0.93      0.61        29
        pool       0.67      0.67      0.67        15
      shadow       0.74      0.88      0.80        16
        soil       0.00      0.00      0.00        14
        tree       0.75      0.71      0.73        17

    accuracy                           0.60       168
   macro avg       0.56      0.55      0.53       168
weighted avg       0.57      0.60      0.55       168


Confusion Matrix (Training Data):
[[ 9  2  1  1 27  0  4  0  1]
 [ 0 70  2  4 11 10  0  0  0]
 [ 0  0 17  4  0  0  0  0  0]
 [ 0  4  4 32 47  5  0  0  1]
 [ 0  0  2  1 64  2  0  0 14]
 [ 0  1  0  0  0 13  0  0  0]
 [ 0  0  0  0  3  0 42  0  0]
 [ 0  3  1  0 13  0  0  2  1]
 [ 0  0  0  0  3  0  0  0 86]]

Classification Report (Training Data):
              precision    recall  f1-score   support

     asphalt       1.00      0.20      0.33        45
    building       0.88      0.72      0.79        97
         car       0.63      0.81      0.71        21
    concrete       0.76      0.34      0.47        93
       grass       0.38      0.77      0.51        83
        pool       0.43      0.93      0.59        14
      shadow       0.91      0.93      0.92        45
        soil       1.00      0.10      0.18        20
        tree       0.83      0.97      0.90        89

    accuracy                           0.66       507
   macro avg       0.76      0.64      0.60       507
```

```
weighted avg       0.76      0.66      0.64       507
```

[ ]:
```
##Yes there are signs of overfitting. High precision, recall, F-1 score for␣
↪most classes on training set. But performance
#dropped in test set for all scores.
```

5 4. Support Vector Machine Classifier + Linear Kernel +grid Search:

[13]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import numpy as np

# Defining the parameter grid
param_grid = {'C': np.arange(0.01, 10, 0.2)}

# Creating SVC with linear kernel
svc_linear = SVC(kernel='linear')

# Running GridSearchCV
grid_search = GridSearchCV(svc_linear, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Printing best parameters
print("Best Parameters:", grid_search.best_params_)

# Getting best estimator
best_svc_linear = grid_search.best_estimator_
```

```
Best Parameters: {'C': 0.01}
```

[14]:
```
# Best performing model
best_svc_linear = grid_search.best_estimator_
```

[15]:
```
# Predicting on test data
y_pred_svc_linear = best_svc_linear.predict(X_test_scaled)
```

[16]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix for test data
conf_matrix_test = confusion_matrix(y_test, y_pred_svc_linear)
print("Confusion Matrix (Test Data):")
print(conf_matrix_test)

# Classification report for test data
class_report_test = classification_report(y_test, y_pred_svc_linear)
```

```
print("\nClassification Report (Test Data):")
print(class_report_test)
```

```
Confusion Matrix (Test Data):
[[13  0  0  0  0  0  1  0  0]
 [ 0 22  0  2  1  0  0  0  0]
 [ 0  1 14  0  0  0  0  0  0]
 [ 0  5  0 17  0  0  0  1  0]
 [ 0  0  0  1 25  0  0  0  3]
 [ 0  0  0  0  0 14  1  0  0]
 [ 1  0  0  0  0  0 15  0  0]
 [ 0  3  0  5  2  0  0  4  0]
 [ 0  0  0  1  2  0  0  0 14]]

Classification Report (Test Data):
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.71      0.88      0.79        25
         car       1.00      0.93      0.97        15
    concrete       0.65      0.74      0.69        23
       grass       0.83      0.86      0.85        29
        pool       1.00      0.93      0.97        15
      shadow       0.88      0.94      0.91        16
        soil       0.80      0.29      0.42        14
        tree       0.82      0.82      0.82        17

    accuracy                           0.82       168
   macro avg       0.85      0.81      0.82       168
weighted avg       0.83      0.82      0.81       168
```

[17]:

```
# Predictions for training data
y_train_pred_svc_linear = best_svc_linear.predict(X_train_scaled)

# Confusion matrix for training data
conf_matrix_train = confusion_matrix(y_train, y_train_pred_svc_linear)
print("Confusion Matrix (Training Data):")
print(conf_matrix_train)

# Classification report for training data
class_report_train = classification_report(y_train, y_train_pred_svc_linear)
print("\nClassification Report (Training Data):")
print(class_report_train)
```

```
Confusion Matrix (Training Data):
[[40  0  0  0  0  0  5  0  0]
 [ 2 87  0  7  0  0  1  0  0]
```

```
 [ 0  1 19  1  0  0  0  0  0]
 [ 0  9  0 83  1  0  0  0  0]
 [ 0  1  0  0 70  0  0  0 12]
 [ 0  1  0  0  1 12  0  0  0]
 [ 1  0  0  0  0  0 43  0  1]
 [ 0  3  0  4  2  0  0 11  0]
 [ 0  0  0  0  3  0  1  0 85]]

Classification Report (Training Data):
              precision    recall  f1-score   support

    asphalt       0.93      0.89      0.91        45
   building       0.85      0.90      0.87        97
        car       1.00      0.90      0.95        21
   concrete       0.87      0.89      0.88        93
      grass       0.91      0.84      0.88        83
       pool       1.00      0.86      0.92        14
     shadow       0.86      0.96      0.91        45
       soil       1.00      0.55      0.71        20
       tree       0.87      0.96      0.91        89

    accuracy                           0.89       507
   macro avg       0.92      0.86      0.88       507
weighted avg       0.89      0.89      0.89       507
```

[ ]:
```
### Model does not exhibit severe signs of overfitting. However,tuning the␣
↪model could -
#help improve its generalization performance
```

6 5. Support Vector Machine Classifier + Polynomial Kernel + Grid Search:

[18]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'C': np.arange(0.01, 10, 0.2),
    'degree': [2, 3, 4, 5, 6]
}

# Create the SVC model with polynomial kernel
svm_poly = SVC(kernel='poly')

# Run GridSearchCV
```

```
grid_search = GridSearchCV(svm_poly, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

[18]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='poly'),
             param_grid={'C': array([0.01, 0.21, 0.41, 0.61, 0.81, 1.01, 1.21,
1.41, 1.61, 1.81, 2.01,
       2.21, 2.41, 2.61, 2.81, 3.01, 3.21, 3.41, 3.61, 3.81, 4.01, 4.21,
       4.41, 4.61, 4.81, 5.01, 5.21, 5.41, 5.61, 5.81, 6.01, 6.21, 6.41,
       6.61, 6.81, 7.01, 7.21, 7.41, 7.61, 7.81, 8.01, 8.21, 8.41, 8.61,
       8.81, 9.01, 9.21, 9.41, 9.61, 9.81]),
                         'degree': [2, 3, 4, 5, 6]})
```

[19]:
```
# Printing the best parameters
print("Best Parameters:", grid_search.best_params_)

# Getting the best estimator
best_svm_poly = grid_search.best_estimator_
```

```
Best Parameters: {'C': 8.41, 'degree': 2}
```

[20]:
```
# Predicting on the test data
y_pred_poly = best_svm_poly.predict(X_test)
```

[29]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report

# Defininf the parameter grid
param_grid = {
    'C': np.arange(0.01, 10, 0.2),
    'degree': [2, 3, 4, 5, 6]
}

# SVC model with polynomial kernel
svm_poly = SVC(kernel='poly')

# Running GridSearchCV with default scoring and 5 cross-fold
grid_search = GridSearchCV(svm_poly, param_grid)
grid_search.fit(X_train, y_train)

# b) Identifying the best performing model:
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# c) Using the best estimator model to predict on test data:
y_pred_test = best_model.predict(X_test)

# d) Calculating the confusion matrix and classification report for test data:
conf_matrix_test = confusion_matrix(y_test, y_pred_test)
```

```
class_report_test = classification_report(y_test, y_pred_test,␣
↪zero_division='warn')

print("Best Parameters:", best_params)
print("Confusion Matrix (Test Data):")
print(conf_matrix_test)
print("\nClassification Report (Test Data):")
print(class_report_test)

# Use the best estimator model to predict on train data:
y_pred_train = best_model.predict(X_train)

# Calculating the confusion matrix and classification report for train data:
conf_matrix_train = confusion_matrix(y_train, y_pred_train)
class_report_train = classification_report(y_train, y_pred_train,␣
↪zero_division='warn')

print("\nConfusion Matrix (Train Data):")
print(conf_matrix_train)
print("\nClassification Report (Train Data):")
print(class_report_train)
```

```
Best Parameters: {'C': 8.41, 'degree': 2}
Confusion Matrix (Test Data):
[[ 7  2  0  1  2  0  0  0  2]
 [ 0 21  0  0  2  0  0  0  2]
 [ 0  9  6  0  0  0  0  0  0]
 [ 0  8  0 13  1  0  0  0  1]
 [ 0  6  0  4  8  0  0  0 11]
 [ 0  1  0  0  0  0  0  0 14]
 [ 0  3  0  0  1  0  0  0 12]
 [ 0  8  0  5  1  0  0  0  0]
 [ 0  0  0  1  1  0  0  0 15]]

Classification Report (Test Data):
              precision    recall  f1-score   support

     asphalt       1.00      0.50      0.67        14
    building       0.36      0.84      0.51        25
         car       1.00      0.40      0.57        15
    concrete       0.54      0.57      0.55        23
       grass       0.50      0.28      0.36        29
        pool       0.00      0.00      0.00        15
      shadow       0.00      0.00      0.00        16
        soil       0.00      0.00      0.00        14
        tree       0.26      0.88      0.41        17
```

```
    accuracy                           0.42       168
   macro avg       0.41      0.38      0.34       168
weighted avg       0.41      0.42      0.36       168


Confusion Matrix (Train Data):
[[19  9  0  1  5  0  0  0 11]
 [ 0 84  0  3  1  0  0  0  9]
 [ 0  7 11  1  0  0  0  0  2]
 [ 1 19  0 65  4  0  0  0  4]
 [ 1  7  0  8 31  0  0  0 36]
 [ 0  1  0  0  0  0  0  0 13]
 [ 2 10  0  0  2  0  0  0 31]
 [ 0  7  0  5  2  0  0  1  5]
 [ 0  0  0  0  2  0  0  0 87]]

Classification Report (Train Data):
              precision    recall  f1-score   support

    asphalt       0.83      0.42      0.56        45
   building       0.58      0.87      0.70        97
        car       1.00      0.52      0.69        21
   concrete       0.78      0.70      0.74        93
      grass       0.66      0.37      0.48        83
       pool       0.00      0.00      0.00        14
     shadow       0.00      0.00      0.00        45
       soil       1.00      0.05      0.10        20
       tree       0.44      0.98      0.61        89

    accuracy                           0.59       507
   macro avg       0.59      0.43      0.43       507
weighted avg       0.59      0.59      0.54       507

C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
## Yes, there are signs of overfitting in the model. This is evident from the␣
↪significant difference in performance metrics
# between the training and test data, including accuracy, precision, recall,␣
↪F1-score, and the confusion matrix.
```

7 6. Support Vector Machine Classifier + RBF Kernel + Grid Search:

[30]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.svm import SVC

# Defining the parameter grid
param_grid = {
    'C': np.arange(0.01, 10, 0.2),
    'gamma': [0.01, 0.1, 1, 10, 100]
}

# Creating the SVC model with RBF kernel
svm_rbf = SVC(kernel='rbf')

# Running GridSearchCV with default scoring and 5 cross-fold
grid_search = GridSearchCV(svm_rbf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# b) Identifying the best performing model:
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

```python
# c) Using the best estimator model to predict on test data:
y_pred_test = best_model.predict(X_test)

# d) Calculating the confusion matrix and classification report for test data:
conf_matrix_test = confusion_matrix(y_test, y_pred_test)
class_report_test = classification_report(y_test, y_pred_test)

print("Best Parameters:", best_params)
print("Confusion Matrix (Test Data):")
print(conf_matrix_test)
print("\nClassification Report (Test Data):")
print(class_report_test)

# e) Calculating predictions for the training data & build the classification␣
↪report & confusion matrix:
y_pred_train = best_model.predict(X_train)
conf_matrix_train = confusion_matrix(y_train, y_pred_train)
class_report_train = classification_report(y_train, y_pred_train)

print("\nConfusion Matrix (Train Data):")
print(conf_matrix_train)
print("\nClassification Report (Train Data):")
print(class_report_train)
```

```
Best Parameters: {'C': 0.01, 'gamma': 0.01}
Confusion Matrix (Test Data):
[[ 0 14  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  0  0  0]
 [ 0 15  0  0  0  0  0  0  0]
 [ 0 23  0  0  0  0  0  0  0]
 [ 0 29  0  0  0  0  0  0  0]
 [ 0 15  0  0  0  0  0  0  0]
 [ 0 16  0  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0  0  0]]

Classification Report (Test Data):
              precision    recall  f1-score   support

    asphalt       0.00      0.00      0.00        14
   building       0.15      1.00      0.26        25
        car       0.00      0.00      0.00        15
   concrete       0.00      0.00      0.00        23
      grass       0.00      0.00      0.00        29
       pool       0.00      0.00      0.00        15
     shadow       0.00      0.00      0.00        16
       soil       0.00      0.00      0.00        14
       tree       0.00      0.00      0.00        17
```

```
   accuracy                           0.15       168
  macro avg       0.02      0.11      0.03       168
weighted avg       0.02      0.15      0.04       168


Confusion Matrix (Train Data):
[[ 0 45  0  0  0  0  0  0  0]
 [ 0 97  0  0  0  0  0  0  0]
 [ 0 21  0  0  0  0  0  0  0]
 [ 0 93  0  0  0  0  0  0  0]
 [ 0 83  0  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  0  0]
 [ 0 45  0  0  0  0  0  0  0]
 [ 0 20  0  0  0  0  0  0  0]
 [ 0 89  0  0  0  0  0  0  0]]

Classification Report (Train Data):
              precision    recall  f1-score   support

    asphalt       0.00      0.00      0.00        45
   building       0.19      1.00      0.32        97
        car       0.00      0.00      0.00        21
   concrete       0.00      0.00      0.00        93
      grass       0.00      0.00      0.00        83
       pool       0.00      0.00      0.00        14
     shadow       0.00      0.00      0.00        45
       soil       0.00      0.00      0.00        20
       tree       0.00      0.00      0.00        89

   accuracy                           0.19       507
  macro avg       0.02      0.11      0.04       507
weighted avg       0.04      0.19      0.06       507

C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arpan\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
##No signs of overfitting. It is infact underfitting.Both the test and training␣
↪data results show signs of poor performance,
#indicating that the model is not able to effectively learn from the data and␣
↪generalize well to unseen instances -
```

8 7. Conceptual Questions:

[ ]:
```
#a) Among these models, the Random Forest Classifier demonstrates the highest␣
↪accuracy and macro F1-score, indicating-
#better overall performance compared to the other models.
```

[ ]:
```
#b) Polynomial and RBF kernels are capable of capturing complex, non-linear␣
↪relationships between features,
# which can be advantageous when the data is not linearly separable.
#Downside could be that polynomial and RBF kernels often involve higher␣
↪computational complexity compared to linear kernels-
#especially as the dimensionality of the feature space increases.
```

[ ]:
```
#c) The 'C' parameter in SVM models controls the trade-off between maximizing␣
↪the margin and minimizing the classification-
# error on the training data. A small 'C' value leads to a wider margin␣
↪decision boundary, allowing more training errors
# and emphasizing smoothness, potentially improving generalization to unseen␣
↪data. Conversely, a large 'C' value results
# in a narrower margin boundary, aiming to classify all training data points␣
↪correctly, which may lead to overfitting,
# particularly in noisy or poorly separated data. Proper tuning of 'C' is␣
↪essential to balance bias and variance, ensuring-
```

```
# optimal model performance and generalization capability.
```

[ ]:
```
#d) Scaling input data is crucial for Support Vector Machines (SVMs) because␣
 ↪SVMs are sensitive to the scale of features.
#SVMs aim to maximize the margin between different classes, and features with␣
 ↪larger scales can disproportionately-
#influence the decision boundary, leading to biased results. For example, in a␣
 ↪dataset containing features like 'area' and-
#'perimeter' of objects, if 'area' is measured in square meters and 'perimeter'␣
 ↪is measured in millimeters, the SVM -
#algorithm may prioritize 'area' due to its larger scale, potentially␣
 ↪neglecting valuable information from 'perimeter.'
#Without scaling, the SVM may struggle to converge or produce suboptimal␣
 ↪results, as it might not effectively learn the
#underlying patterns in the data.
```

[ ]:
```
#e)Conceptually, the purpose of a kernel in Support Vector Machines (SVMs) is␣
 ↪to transform the input data from its original-
#feature space into a higher-dimensional space where the data becomes linearly␣
 ↪separable. SVMs work by finding the optimal
#hyperplane that maximizes the margin between different classes in this␣
 ↪transformed space. Kernels enable SVMs to perform
#this transformation efficiently without explicitly computing the coordinates␣
 ↪of data points in the higher-dimensional
#space. Instead, they implicitly calculate the dot products between data points␣
 ↪in the higher-dimensional space, allowing
#SVMs to find complex decision boundaries in the original feature space without␣
 ↪directly operating in that space.
#In essence, kernels help SVMs capture nonlinear relationships in the data by␣
 ↪projecting it into a higher-dimensional space
#where such relationships may become linear.
```

9 The End