```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install scikit-learn-extra
```

```
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 12.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.25.2)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.11.4)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-ext
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-le
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.3.0
```

```
import os

# Define the path to your dataset folder
dataset_path = '/content/drive/MyDrive/Seeds'

# Check the contents of the folder
os.chdir(dataset_path)
```

```
from skimage.feature import canny
from skimage.feature import daisy
from sklearn_extra.cluster import KMedoids
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import os
import cv2
import numpy as np
from skimage import exposure
from skimage.filters import gabor
from skimage.feature import local_binary_pattern
from skimage.feature import greycomatrix, greycoprops
from skimage.feature import hog
from sklearn.preprocessing import MinMaxScaler


# Function to extract color histograms
def extract_color_histogram(image):
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist_hue = cv2.calcHist([hsv_image], [0], None, [256], [0, 256])
    hist_saturation = cv2.calcHist([hsv_image], [1], None, [256], [0, 256])
    hist_value = cv2.calcHist([hsv_image], [2], None, [256], [0, 256])
    return hist_hue.flatten(), hist_saturation.flatten(), hist_value.flatten()


# Function to extract Gabor filter features
def extract_gabor_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gabor_result, _ = gabor(gray_image, frequency=0.6, theta=1.5)
    return gabor_result.flatten()


# Function to extract GLCM (Gray Level Co-occurrence Matrix) features
def extract_glcm_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    distances = [1, 2, 3]
    angles = [0, np.pi/4, np.pi/2, 3*np.pi/4]
    glcm = greycomatrix(gray_image, distances=distances, angles=angles, symmetric=True, normed=True)
    contrast = greycoprops(glcm, 'contrast')
    dissimilarity = greycoprops(glcm, 'dissimilarity')
    homogeneity = greycoprops(glcm, 'homogeneity')
    energy = greycoprops(glcm, 'energy')
    correlation = greycoprops(glcm, 'correlation')
    glcm_features = np.hstack((contrast.flatten(), dissimilarity.flatten(), homogeneity.flatten(), energy.flatten(), correlation.flatter
    return glcm_features
```

```
# Function to extract HOG (Histogram of Oriented Gradients) features using skimage
def extract_skimage_hog_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    features, _ = hog(gray_image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    return features.flatten()


# Function to extract Local Binary Pattern (LBP) features using skimage
def extract_skimage_lbp_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    radius = 3
    n_points = 8 * radius
    lbp = local_binary_pattern(gray_image, n_points, radius, method='uniform')
    hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, n_points + 3), range=(0, n_points + 2))
    return hist.flatten()




# Function to extract Canny edge features
def extract_canny_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = canny(gray_image)
    return edges.flatten()


# Function to extract DAISY (Descrete Anisotropic Scale Space) features
# def extract_daisy_features(image):
#     gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#     features = daisy(gray_image, step=180, radius=58, rings=2, histograms=6, orientations=8)
#     return features.flatten()



# Function to extract texture features using Haralick features
def extract_haralick_features(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    textures = cv2.imgproc.textureFast(gray_image)
    # Extract Haralick texture features
    haralick = cv2.imgproc.haralick(textures)
    features = np.ravel(haralick)
    return features


# Function to perform advanced image processing
def advanced_image_processing(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    equalized_image = exposure.equalize_hist(gray_image)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    clahe_image = clahe.apply(gray_image)
    bilateral_filtered = cv2.bilateralFilter(image, 9, 75, 75)
    return equalized_image, clahe_image, bilateral_filtered
```

```
def extract_features_and_clustering(dataset_path):
    images = []
    min_height = 100
    min_width = 100

    for subdir, dirs, files in os.walk(dataset_path):
        for file in files:
            file_path = os.path.join(subdir, file)
            img = cv2.imread(file_path)
            if img.shape[0] >= min_height and img.shape[1] >= min_width:
                images.append(img)
            else:
                print(f"Ignoring {file_path} due to small dimensions.")

    color_histograms = []
    gabor_features_list = []
    glcm_features_list = []
    skimage_hog_features_list = []
    skimage_lbp_features_list = []
    canny_features_list = []
    haralick_features_list = []

    for img in images:
        hist_hue, hist_saturation, hist_value = extract_color_histogram(img)
        color_histograms.append(np.concatenate([hist_hue, hist_saturation, hist_value]))

        gabor_features = extract_gabor_features(img)
        gabor_features_list.append(gabor_features)

        glcm_features = extract_glcm_features(img)
        glcm_features_list.append(glcm_features)

        skimage_hog_features = extract_skimage_hog_features(img)
        skimage_hog_features_list.append(skimage_hog_features)

        skimage_lbp_features = extract_skimage_lbp_features(img)
        skimage_lbp_features_list.append(skimage_lbp_features)

        canny_features = extract_canny_features(img)
        canny_features_list.append(canny_features)


    # Truncate or pad lists to ensure equal lengths
    # Ensure equal lengths of lists
    min_length = min(len(color_histograms), len(gabor_features_list), len(glcm_features_list),
                     len(skimage_hog_features_list), len(skimage_lbp_features_list), len(canny_features_list))

    color_histograms = [feature[:min_length] for feature in color_histograms]
    gabor_features_list = [feature[:min_length] for feature in gabor_features_list]
    glcm_features_list = [feature[:min_length] for feature in glcm_features_list]
    skimage_hog_features_list = [feature[:min_length] for feature in skimage_hog_features_list]
    skimage_lbp_features_list = [feature[:min_length] for feature in skimage_lbp_features_list]
    canny_features_list = [feature[:min_length] for feature in canny_features_list]


    # Concatenate the features
    features = np.concatenate([color_histograms, gabor_features_list, glcm_features_list,
                               skimage_hog_features_list, skimage_lbp_features_list, canny_features_list], axis=1)

    scaler = MinMaxScaler()
    features_scaled = scaler.fit_transform(features)

    k = 5
    kmedoids = KMedoids(n_clusters=k)
    clusters = kmedoids.fit_predict(features_scaled)

    return features_scaled, clusters
```

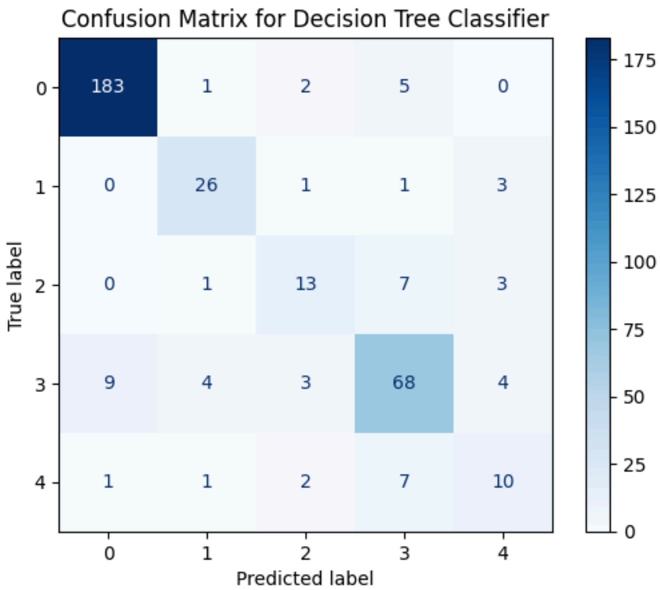
```
def encode_features(features):
    from sklearn.decomposition import PCA

    # Determine the maximum value for n_components based on the minimum of samples or features
    n_components = min(features.shape[0], features.shape[1])

    # Reduce n_components to avoid exceeding the minimum of samples or features
    n_components = min(n_components, 90)  # Set a maximum value of 90 for n_components

    pca = PCA(n_components=n_components)
    encoded_features = pca.fit_transform(features)
    return encoded_features



# # Function to perform Decision Tree Classification
# def decision_tree_classification(encoded_features, clusters):
#     X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
#     tree_classifier = DecisionTreeClassifier(random_state=42)
#     tree_classifier.fit(X_train, y_train)
#     y_pred = tree_classifier.predict(X_test)
#     accuracy = accuracy_score(y_test, y_pred)
#     return accuracy




import matplotlib.pyplot as plt


from sklearn.metrics import roc_curve, confusion_matrix, roc_auc_score

# Function to plot ROC curve
def plot_roc_curve(y_test, y_pred_proba):
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba[:, 1])
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label='ROC Curve')
    plt.plot([0, 1], [0, 1], 'k--', label='Random')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Decision Tree Classifier')
    plt.legend()
    plt.show()

# Function to display confusion matrix
def display_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)

# Function to perform Decision Tree Classification and get metrics
def decision_tree_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    tree_classifier = DecisionTreeClassifier(random_state=42)
    tree_classifier.fit(X_train, y_train)
    y_pred = tree_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate ROC AUC score
    y_pred_proba = tree_classifier.predict_proba(X_test)
    roc_auc = roc_auc_score(y_test, y_pred_proba[:, 1])

    # Plot ROC curve
    plot_roc_curve(y_test, y_pred_proba)

    # Display confusion matrix
    display_confusion_matrix(y_test, y_pred)

    return accuracy, roc_auc



# Define the path to your dataset folder
dataset_path = '/content/drive/MyDrive/Seeds'

# Extract features and perform clustering
features_scaled, clusters = extract_features_and_clustering(dataset_path)
```

```
Ignoring /content/drive/MyDrive/Seeds/Litchi/Litchi (59).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Orange/Orange (82).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Sapodilla/Sapodilla (45).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Hog_Plum/Hog_Plum (63).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Hog_Plum/Hog_Plum (66).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Hog_Plum/Hog_Plum (64).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Hog_Plum/Hog_Plum (65).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Olive/Olive (76).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Olive/Olive (75).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Amla/Amla (24).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Amla/Amla (27).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Amla/Amla (23).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Amla/Amla (26).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Amla/Amla (25).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Black_Plum/Black_Plum (1).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Black_Plum/Black_Plum (2).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (85).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (84).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (86).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (88).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (90).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (89).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (83).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (81).jpg due to small dimensions.
Ignoring /content/drive/MyDrive/Seeds/Guava/Guava (87).jpg due to small dimensions.
Streaming output truncated to the last 5000 lines.
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:35: skimage_deprecation: Function ``greycomatrix`` is depreca
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:35: skimage_deprecation: Function ``greycomatrix`` is depreca
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
  removed_version='1.0')
/usr/local/lib/python3.10/dist-packages/skimage/feature/__init__.py:42: skimage_deprecation: Function ``greycoprops`` is deprecat
```

```
# Encode features
encoded_features = encode_features(features_scaled)
```

```
# # Perform Decision Tree Classification
# accuracy = decision_tree_classification(encoded_features, clusters)
# print("Accuracy of Decision Tree Classifier using K-medoids clusters:", accuracy)
```

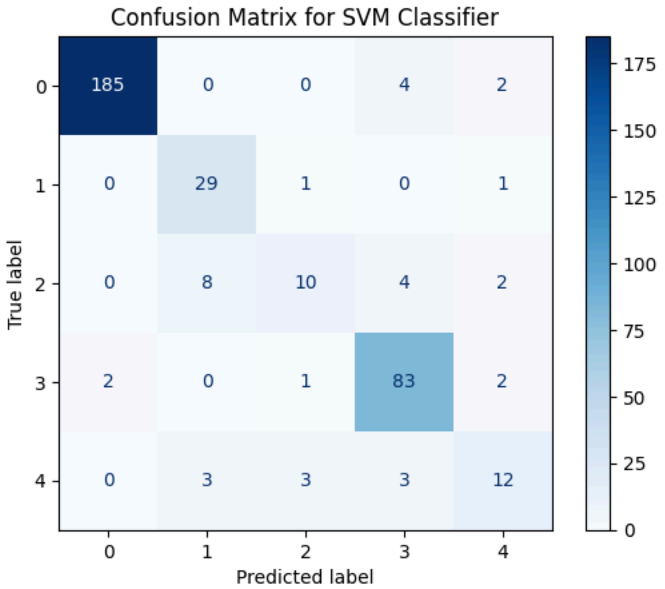
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

def decision_tree_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    tree_classifier = DecisionTreeClassifier(random_state=42)
    tree_classifier.fit(X_train, y_train)
    y_pred = tree_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Display confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix for Decision Tree Classifier')
    plt.show()

    return accuracy

# Perform Decision Tree Classification and plot confusion matrix
dec_accuracy = decision_tree_classification_metrics(encoded_features, clusters)
print("Accuracy of Decision Tree Classifier using K-medoids clusters:", dec_accuracy)
```




```
Accuracy of Decision Tree Classifier using K-medoids clusters: 0.8450704225352113
```

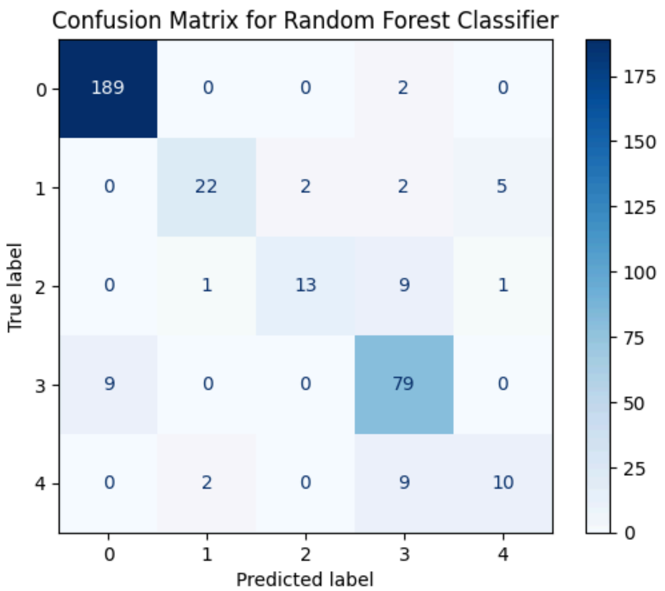
```
from sklearn.svm import SVC
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

def svm_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    svm_classifier = SVC(kernel='linear', probability=True, random_state=42)
    svm_classifier.fit(X_train, y_train)
    y_pred = svm_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Display confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix for SVM Classifier')
    plt.show()

    return accuracy

# Perform SVM Classification and plot confusion matrix
svm_accuracy = svm_classification_metrics(encoded_features, clusters)
print("Accuracy of SVM Classifier using K-medoids clusters:", svm_accuracy)
```



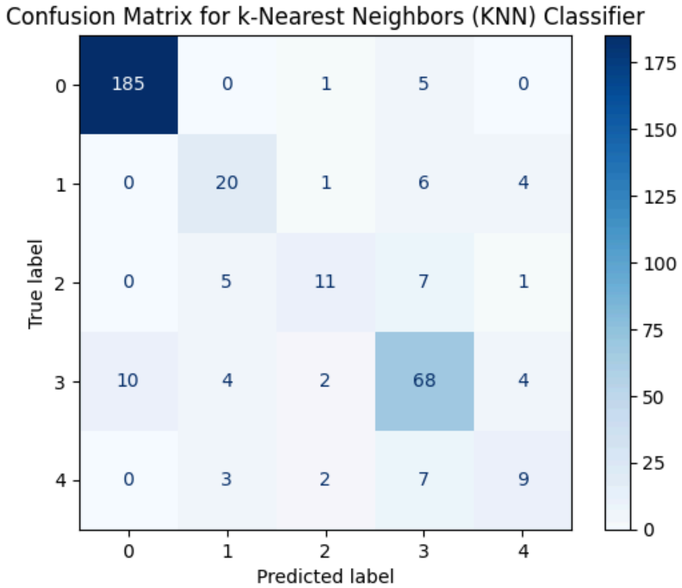

```
Accuracy of SVM Classifier using K-medoids clusters: 0.8985915492957747
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# Function to perform Random Forest Classification
def random_forest_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    rf_classifier = RandomForestClassifier(random_state=42)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Display confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix for Random Forest Classifier')
    plt.show()

    return accuracy

# Perform Random Forest Classification and plot confusion matrix
rf_accuracy = random_forest_classification_metrics(encoded_features, clusters)
print("Accuracy of Random Forest Classifier using K-medoids clusters:", rf_accuracy)
```



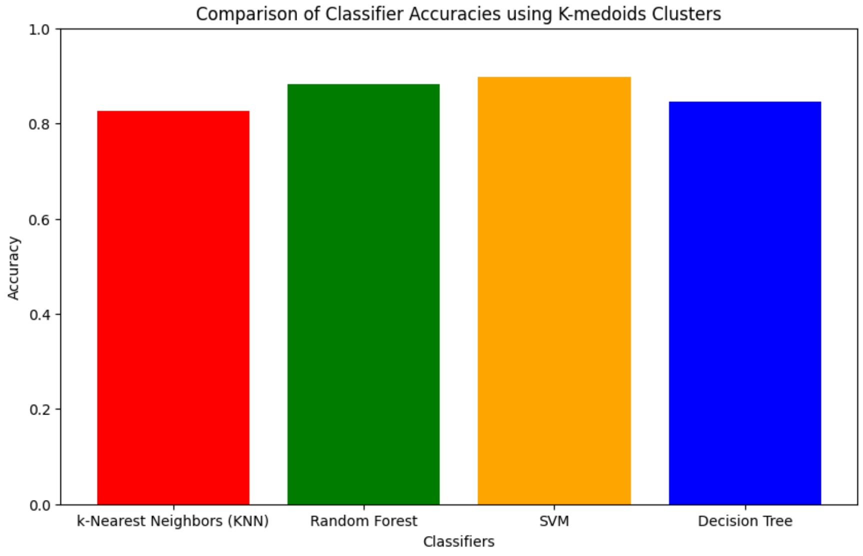

```
Accuracy of Random Forest Classifier using K-medoids clusters: 0.8816901408450705
```

```
# Function to perform k-Nearest Neighbors (KNN) Classification
def knn_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    knn_classifier = KNeighborsClassifier()
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Display confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix for k-Nearest Neighbors (KNN) Classifier')
    plt.show()

    return accuracy

# Perform k-Nearest Neighbors (KNN) Classification and plot confusion matrix
knn_accuracy = knn_classification_metrics(encoded_features, clusters)
print("Accuracy of k-Nearest Neighbors (KNN) Classifier using K-medoids clusters:", knn_accuracy)
```




```
Accuracy of k-Nearest Neighbors (KNN) Classifier using K-medoids clusters: 0.82535211
```

```
import matplotlib.pyplot as plt

# Accuracy values
accuracies = [knn_accuracy, rf_accuracy, svm_accuracy,dec_accuracy]

# Classifier names
classifiers = ['k-Nearest Neighbors (KNN)', 'Random Forest', 'SVM','Decision Tree']

# Plotting the bar graph
plt.figure(figsize=(10, 6))
plt.bar(classifiers, accuracies, color=['red', 'green', 'orange','blue'])
plt.xlabel('Classifiers')
plt.ylabel('Accuracy')
plt.title('Comparison of Classifier Accuracies using K-medoids Clusters')
plt.ylim(0, 1)  # Setting y-axis limit to better visualize differences
plt.show()
```




```
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, precision_recall_curve, roc_curve

def calculate_metrics(y_test, y_pred, y_pred_proba):
    # Precision
    precision = precision_score(y_test, y_pred, average='weighted')

    # Recall
    recall = recall_score(y_test, y_pred, average='weighted')

    # F1 Score
    f1 = f1_score(y_test, y_pred, average='weighted')



    return precision, recall, f1

def decision_tree_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    tree_classifier = DecisionTreeClassifier(random_state=42)
    tree_classifier.fit(X_train, y_train)
    y_pred = tree_classifier.predict(X_test)


    # Calculate probabilities for positive class
    y_pred_proba = tree_classifier.predict_proba(X_test)

    # Calculate additional metrics
    precision, recall, f1 = calculate_metrics(y_test, y_pred, y_pred_proba)


    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)


decision_tree_classification_metrics(encoded_features, clusters)
```

```
Precision: 0.8419296118996915
Recall: 0.8450704225352113
F1 Score: 0.8432251946639185
```

```
def svm_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    svm_classifier = SVC(kernel='linear', probability=True, random_state=42)
    svm_classifier.fit(X_train, y_train)
    y_pred = svm_classifier.predict(X_test)

    # # Calculate probabilities for positive class
    y_pred_proba = svm_classifier.predict_proba(X_test)

    # Calculate additional metrics
    precision, recall, f1 = calculate_metrics(y_test, y_pred, y_pred_proba)

    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)


# Perform SVM Classification and compute all metrics
svm_classification_metrics(encoded_features, clusters)
```

```
Precision: 0.8968943797342201
Recall: 0.8985915492957747
F1 Score: 0.8942339272987956
```

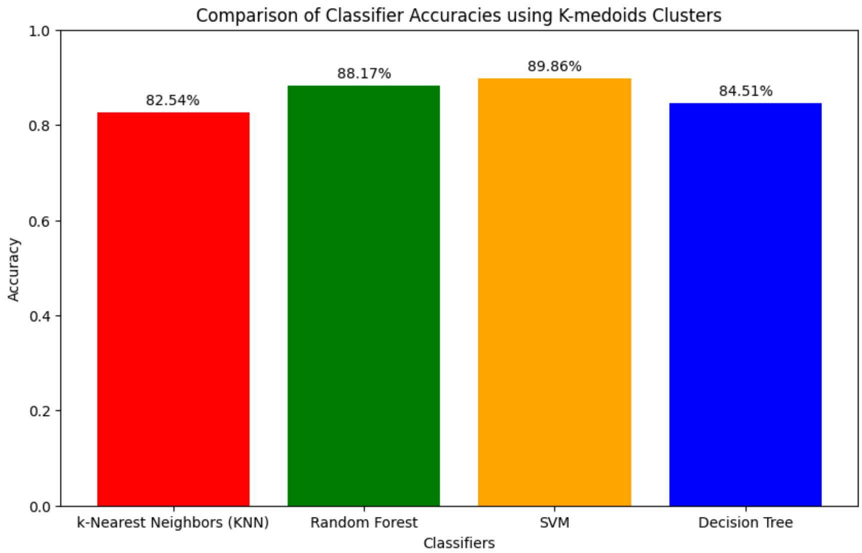
```
def random_forest_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    rf_classifier = RandomForestClassifier(random_state=42)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)

    # Calculate probabilities for positive class
    y_pred_proba = rf_classifier.predict_proba(X_test)

    # Calculate additional metrics
    precision, recall, f1 = calculate_metrics(y_test, y_pred, y_pred_proba)


    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)


random_forest_classification_metrics(encoded_features, clusters)
```

```
Precision: 0.8798728591168977
Recall: 0.8816901408450705
F1 Score: 0.8757002915408361
```

```
def knn_classification_metrics(encoded_features, clusters):
    X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)
    knn_classifier = KNeighborsClassifier()
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)


    # Calculate probabilities for positive class
    y_pred_proba = knn_classifier.predict_proba(X_test)

    # Calculate additional metrics
    precision, recall, f1= calculate_metrics(y_test, y_pred, y_pred_proba)


    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)


knn_classification_metrics(encoded_features, clusters)
```

```
Precision: 0.8195876788488435
Recall: 0.8253521126760563
F1 Score: 0.8210066047072261
```

```
import matplotlib.pyplot as plt

# Accuracy values
accuracies = [knn_accuracy, rf_accuracy, svm_accuracy, dec_accuracy]

# Classifier names
classifiers = ['k-Nearest Neighbors (KNN)', 'Random Forest', 'SVM', 'Decision Tree']

# Plotting the bar graph
plt.figure(figsize=(10, 6))
bars = plt.bar(classifiers, accuracies, color=['red', 'green', 'orange', 'blue'])
plt.xlabel('Classifiers')
plt.ylabel('Accuracy')
plt.title('Comparison of Classifier Accuracies using K-medoids Clusters')
plt.ylim(0, 1)  # Setting y-axis limit to better visualize differences

# Adding text annotations on top of each bar
for bar, accuracy in zip(bars, accuracies):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.01, f'{accuracy:.2%}', ha='center', va='bottom')

plt.show()
```



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)

# Initialize Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Accuracy: 0.8816901408450705
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       191
           1       0.88      0.71      0.79        31
           2       0.87      0.54      0.67        24
           3       0.78      0.90      0.84        88
           4       0.62      0.48      0.54        21

    accuracy                           0.88       355
   macro avg       0.82      0.72      0.76       355
weighted avg       0.88      0.88      0.88       355

Confusion Matrix:
[[189   0   0   2   0]
 [  0  22   2   2   5]
 [  0   1  13   9   1]
 [  9   0   0  79   0]
 [  0   2   0   9  10]]
```

```
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)

# Define individual classifiers
svm_classifier = SVC(probability=True)
rf_classifier = RandomForestClassifier()
knn_classifier = KNeighborsClassifier()

# Define the ensemble model using voting
voting_classifier = VotingClassifier(estimators=[('svm', svm_classifier), ('rf', rf_classifier), ('knn', knn_classifier)], voting='soft

# Train the ensemble model
voting_classifier.fit(X_train, y_train)

# Predict the labels
y_pred = voting_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate ROC AUC score
y_pred_proba = voting_classifier.predict_proba(X_test)


# Display confusion matrix and ROC curve
display_confusion_matrix(y_test, y_pred)

# Print accuracy and ROC AUC score
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
[[189   0   0   2   0]
 [  0  19   1   5   6]
 [  0   5   9   9   1]
 [  8   3   2  74   1]
 [  0   4   1   7   9]]
Accuracy: 0.8450704225352113
```

```
from sklearn.ensemble import GradientBoostingClassifier

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)

# Initialize Gradient Boosting classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Train the classifier
gb_classifier.fit(X_train, y_train)

# Predict the labels for test set
y_pred = gb_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)



# Display confusion matrix
def display_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)

# Display confusion matrix
display_confusion_matrix(y_test, y_pred)
```

```
Accuracy: 0.9126760563380282
Confusion Matrix:
[[189   0   0   2   0]
 [  0  26   0   0   5]
 [  0   1  16   7   0]
 [  8   0   0  80   0]
 [  0   1   0   7  13]]
```

```
from sklearn.ensemble import AdaBoostClassifier

# Initialize AdaBoost classifier
adaboost_classifier = AdaBoostClassifier(n_estimators=100, random_state=42)

# Train the classifier
adaboost_classifier.fit(X_train, y_train)

# Predict the labels for test set
y_pred = adaboost_classifier.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix

# Calculate accuracy
accuracy_adaboost = accuracy_score(y_test, y_pred)
print("AdaBoost Accuracy:", accuracy_adaboost)

# Compute confusion matrix
conf_matrix_adaboost = confusion_matrix(y_test, y_pred)
print("AdaBoost Confusion Matrix:")
print(conf_matrix_adaboost)
```

```
AdaBoost Accuracy: 0.6
AdaBoost Confusion Matrix:
[[125   0   0  66   0]
 [  0   3   9  19   0]
 [  0   0  17   6   1]
 [  1   1  19  66   1]
 [  0   2   8   9   2]]
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(encoded_features, clusters, test_size=0.2, random_state=42)

# Initialize Bagging classifier without specifying base_estimator
bagging_classifier = BaggingClassifier(n_estimators=100, random_state=42)

# Train the classifier
bagging_classifier.fit(X_train, y_train)

# Predict the labels for test set
y_pred = bagging_classifier.predict(X_test)

# Calculate accuracy
accuracy_bagging = accuracy_score(y_test, y_pred)
print("Bagging Accuracy:", accuracy_bagging)

# Compute confusion matrix
conf_matrix_bagging = confusion_matrix(y_test, y_pred)
print("Bagging Confusion Matrix:")
print(conf_matrix_bagging)
```

```
Bagging Accuracy: 0.8901408450704226
Bagging Confusion Matrix:
[[184   0   0   7   0]
 [  0  26   0   0   5]
 [  0   2  14   7   1]
 [  8   0   0  80   0]
 [  0   2   0   7  12]]
```

```
from sklearn.ensemble import ExtraTreesClassifier

# Initialize Extra Trees classifier
extra_trees_classifier = ExtraTreesClassifier(n_estimators=100, random_state=42)

# Train the classifier
extra_trees_classifier.fit(X_train, y_train)

# Predict the labels for test set
y_pred = extra_trees_classifier.predict(X_test)
# Calculate accuracy
accuracy_extra_trees = accuracy_score(y_test, y_pred)
print("Extra Trees Accuracy:", accuracy_extra_trees)

# Compute confusion matrix
conf_matrix_extra_trees = confusion_matrix(y_test, y_pred)
print("Extra Trees Confusion Matrix:")
print(conf_matrix_extra_trees)
```

```
Extra Trees Accuracy: 0.8422535211267606
Extra Trees Confusion Matrix:
[[188   0   0   3   0]
 [  0  23   0   3   5]
 [  1   2  11   8   2]
 [ 19   0   0  69   0]
 [  1   2   0  10   8]]
```

```
import lightgbm as lgb

# Convert data to LightGBM Dataset format
train_data = lgb.Dataset(X_train, label=y_train)
test_data = lgb.Dataset(X_test)

# Define parameters for LightGBM
params = {
    'objective': 'binary',
    'metric': 'auc',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': -1,
    'seed': 42
}

# Train the model
lgb_classifier = lgb.train(params, train_data, num_boost_round=100)

# Predict probabilities for test set
y_pred_proba = lgb_classifier.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)



# Calculate accuracy
accuracy_lightgbm = accuracy_score(y_test, y_pred)
print("LightGBM Accuracy:", accuracy_lightgbm)

# Compute confusion matrix
conf_matrix_lightgbm = confusion_matrix(y_test, y_pred)
print("LightGBM Confusion Matrix:")
print(conf_matrix_lightgbm)
```

```
LightGBM Accuracy: 0.6197183098591549
LightGBM Confusion Matrix:
[[189   2   0   0   0]
 [  0  31   0   0   0]
 [  0  24   0   0   0]
 [  6  82   0   0   0]
 [  0  21   0   0   0]]
```

```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.3-cp310-cp310-manylinux2014_x86_64.whl (98.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.5/98.5 MB 939.7 kB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.25.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.11.4)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2023.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.50.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.1.2)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (8.2.3)
Installing collected packages: catboost
Successfully installed catboost-1.2.3
```

```python
from catboost import CatBoostClassifier

# Initialize CatBoost classifier
catboost_classifier = CatBoostClassifier(iterations=100, random_seed=42)

# Train the classifier
catboost_classifier.fit(X_train, y_train)

# Predict the labels for test set
y_pred = catboost_classifier.predict(X_test)
```

```
27:	learn: 0.1160075	total: 3.1s	remaining: 7.98s
28:	learn: 0.1131135	total: 3.23s	remaining: 7.9s
29:	learn: 0.1086183	total: 3.38s	remaining: 7.88s
30:	learn: 0.1070518	total: 3.55s	remaining: 7.89s
31:	learn: 0.1042950	total: 3.63s	remaining: 7.72s
32:	learn: 0.1006826	total: 3.72s	remaining: 7.55s
33:	learn: 0.0964659	total: 3.8s	remaining: 7.38s
34:	learn: 0.0917384	total: 3.89s	remaining: 7.22s
35:	learn: 0.0895130	total: 3.97s	remaining: 7.06s
36:	learn: 0.0872792	total: 4.06s	remaining: 6.91s
37:	learn: 0.0847562	total: 4.15s	remaining: 6.78s
38:	learn: 0.0831520	total: 4.24s	remaining: 6.63s
39:	learn: 0.0814695	total: 4.32s	remaining: 6.48s
40:	learn: 0.0777011	total: 4.41s	remaining: 6.34s
41:	learn: 0.0747309	total: 4.51s	remaining: 6.23s
42:	learn: 0.0730516	total: 4.6s	remaining: 6.1s
43:	learn: 0.0712693	total: 4.69s	remaining: 5.97s
44:	learn: 0.0686978	total: 4.77s	remaining: 5.83s
45:	learn: 0.0671743	total: 4.86s	remaining: 5.71s
46:	learn: 0.0662859	total: 4.96s	remaining: 5.59s
47:	learn: 0.0650100	total: 5.05s	remaining: 5.47s
48:	learn: 0.0631617	total: 5.14s	remaining: 5.35s
49:	learn: 0.0622850	total: 5.22s	remaining: 5.22s
50:	learn: 0.0613425	total: 5.31s	remaining: 5.1s
51:	learn: 0.0599545	total: 5.4s	remaining: 4.98s
52:	learn: 0.0574355	total: 5.49s	remaining: 4.87s
53:	learn: 0.0570203	total: 5.59s	remaining: 4.76s
54:	learn: 0.0548541	total: 5.68s	remaining: 4.65s
55:	learn: 0.0528318	total: 5.77s	remaining: 4.54s
56:	learn: 0.0511628	total: 5.86s	remaining: 4.42s
57:	learn: 0.0502654	total: 5.94s	remaining: 4.3s
58:	learn: 0.0499014	total: 6.03s	remaining: 4.19s
59:	learn: 0.0489920	total: 6.11s	remaining: 4.08s
60:	learn: 0.0481675	total: 6.25s	remaining: 3.99s
61:	learn: 0.0474997	total: 6.4s	remaining: 3.92s
62:	learn: 0.0470287	total: 6.58s	remaining: 3.86s
63:	learn: 0.0458852	total: 6.72s	remaining: 3.78s
64:	learn: 0.0448398	total: 6.89s	remaining: 3.71s
65:	learn: 0.0439723	total: 7.05s	remaining: 3.63s
66:	learn: 0.0435796	total: 7.18s	remaining: 3.54s
67:	learn: 0.0423085	total: 7.36s	remaining: 3.46s
68:	learn: 0.0414019	total: 7.56s	remaining: 3.4s
69:	learn: 0.0408835	total: 7.79s	remaining: 3.34s
70:	learn: 0.0404868	total: 8.02s	remaining: 3.27s
71:	learn: 0.0396479	total: 8.29s	remaining: 3.22s
72:	learn: 0.0387112	total: 8.54s	remaining: 3.16s
73:	learn: 0.0381882	total: 8.88s	remaining: 3.12s
74:	learn: 0.0372681	total: 9.11s	remaining: 3.04s
75:	learn: 0.0371015	total: 9.34s	remaining: 2.95s
76:	learn: 0.0365641	total: 9.64s	remaining: 2.88s
77:	learn: 0.0359164	total: 9.88s	remaining: 2.79s
78:	learn: 0.0355426	total: 10.2s	remaining: 2.7s
79:	learn: 0.0347783	total: 10.4s	remaining: 2.61s
80:	learn: 0.0339271	total: 10.6s	remaining: 2.5s
81:	learn: 0.0335851	total: 10.9s	remaining: 2.39s
```