Classification and Clustering Model for leaves

Fundamentals of Machine Learning's second project

Ali Akbar Ahrari
ISFAHAN UNIVERSITY

second project

Classification and Clustering Model for leaves

Course: Fundamentals of Machine Learning

Student Name: Ali Akbar Ahrari - 4003613001

Teacher Name: Mr. Mohammad Kiani

Table of Contents

Note

Data available in *data* directory.
Python files available in *codes* directory.
Documents available in *docs* directory.

Overview

This script performs classification and clustering tasks on a dataset of leaf images and dataset (leaves.csv), aiming to classify leaf species and cluster similar instances based on extracted features. We start by classifying the data based on features extracted from the provided images and dataset. After that we jump into clustering the data and finding a relation between the clusters and classes. Let's start.

Set-Up

We start by importing necessary libraries we'll need later. Here is a description of what each library and import used for:

- **Pandas** and **NumPy**: Data manipulation and numerical operations.
- **Scikit-learn**: Machine learning utilities for model training, feature selection, metrics, clustering, and dimensionality reduction.
- **Matplotlib** and **Seaborn**: Data visualization libraries.
- **OpenCV** (cv2): Image processing library.
- **scikit-image** (skimage): Image processing and feature extraction utilities.
- **SciPy**: Scientific computing tools.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.pipeline import Pipeline
from sklearn.linear_model  import LogisticRegression

import os
import cv2
from skimage.feature import greycomatrix, greycoprops
from skimage import io, color

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score,
normalized_mutual_info_score, v_measure_score, homogeneity_score,
completeness_score, silhouette_score
from sklearn.manifold import TSNE
from scipy import stats
```

Let’s prepare the data for the models. First, we read the leaves.csv file and turn it into a data-frame using the **pandas** csv reader. After that, we can read the images and processes the appropriate ones for performing the feature extraction.

```
file_path = '/content/drive/My Drive/programming/leaf-classification-
clustering/data/leaves.csv'
df = pd.read_csv(file_path, header=None)
column_names = [f'feature_{i}' for i in range(df.shape[1])]
df.columns = column_names
```

Note: Since the dataset does not have any column name, we add it manually.

The **load_images** function Loads grayscale leaf images based on folder structure and DataFrame entries using scikit-image. We should define where and what images are proper based on the Dataframe we read. As you can see, Image names and class directory of each leaf has a specific rule that we defined it.

```
root_folder = '/content/drive/My Drive/programming/leaf-classification-
clustering/data/leaves'

unique_classes = df['feature_0'].unique()

def load_images(root_folder, df):
    images = []
    for index, row in df.iterrows():
        class_number = int(row['feature_0'])
        image_number = int(row['feature_1'])
        image_name = f"iPAD2_C{class_number:02d}_EX{image_number:02d}.JPG"
        class_folder = os.path.join(root_folder, f"{class_number}.
{row['class_Name']}")
        image_path = os.path.join(class_folder, image_name)
        if os.path.exists(image_path):
            image = io.imread(image_path)
            image = color.rgb2gray(image)
            images.append(image)
        else:
            images.append(None)  # Handle missing images appropriately
    return images

class_name_mapping = {
    1: "Quercus suber",
    2: "Salix atrocinerea",
    3: "Populus nigra",
    4: "Alnus sp",
    5: "Quercus robur",
    6: "Crataegus monogyna",
    7: "Ilex aquifolium",
    8: "Nerium oleander",
    9: "Betula pubescens",
    10: "Tilia tomentosa",
    11: "Acer palmaturu",
    12: "Celtis sp",
    13: "Corylus avellana",
    14: "Castanea sativa",
    15: "Populus alba",
    22: "Primula vulgaris",
    23: "Erodium sp",
    24: "Bougainvillea sp",
    25: "Arisarum vulgare",
    26: "Euonymus japonicus",
    27: "Ilex perado ssp azorica",
    28: "Magnolia soulangeana",
    29: "Buxus sempervirens",
    30: "Urtica dioica",
    31: "Podocarpus sp",
    32: "Acca sellowiana",
    33: "Hydrangea sp",
    34: "Pseudosasa japonica",
    35: "Magnolia grandiflora",
    36: "Geranium sp"
}
df['class_Name'] = df['feature_0'].map(class_name_mapping)
# Load the images
images = load_images(root_folder, df)
```

After reading the images, we can now extract some features for later data exploration and making a better model. **extract_texture_features** Utilizes **GLCM** from **skimage.feature** to compute texture features like contrast, dissimilarity, homogeneity, energy, and correlation. We also handle any missing image here. After this process, **texture_features_df** will keep the extracted data.

```
def extract_texture_features(image):
    if image.dtype == np.float64:
        image = (image * 255).astype(np.uint8)
    # Compute GLCM
    glcm = greycomatrix(image, distances=[5], angles=[0], levels=256,
symmetric=True, normed=True)

    # Compute texture features
    contrast = greycoprops(glcm, prop='contrast')[0, 0]
    dissimilarity = greycoprops(glcm, prop='dissimilarity')[0, 0]
    homogeneity = greycoprops(glcm, prop='homogeneity')[0, 0]
    energy = greycoprops(glcm, prop='energy')[0, 0]
    correlation = greycoprops(glcm, prop='correlation')[0, 0]

    return [contrast, dissimilarity, homogeneity, energy, correlation]
```

```
# Extract texture features for all images
texture_features = []
for image in images:
    if image is not None:
        features = extract_texture_features(image)
        texture_features.append(features)
    else:
        texture_features.append([np.nan] * 5)  # Handle missing images
appropriately

texture_features_df = pd.DataFrame(texture_features, columns=['contrast',
'dissimilarity', 'homogeneity', 'energy', 'correlation'])
```

Now, we can concatenate the available datasets and use them for further actions. Now it's time to get the **X** and **y** from the **df_combined**.

```
# Concatenate the original dataset with the new texture features
df_combined = pd.concat([df, texture_features_df], axis=1)
# Split the data into features and target
X = df_combined.drop(columns=['feature_0','feature_1','class_Name'])
y = df_combined['feature_0']
```

Note: removing the labels, second feature and class_name column we added before is necessary.

Classification

We split the data into training and testing sets using **train_test_split**. The **test_size=0.1** parameter specifies that 30% of the data will be used for testing. The **random_state=42** parameter ensures reproducibility, and **stratify=y** ensures that the target distribution is similar in both the training and testing sets.

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42, stratify=y)
```

This part builds a pipeline (clf) that selects features using **Logistic Regression** (**SelectFromModel**) and performs classification using **Extra Trees Classifier**.

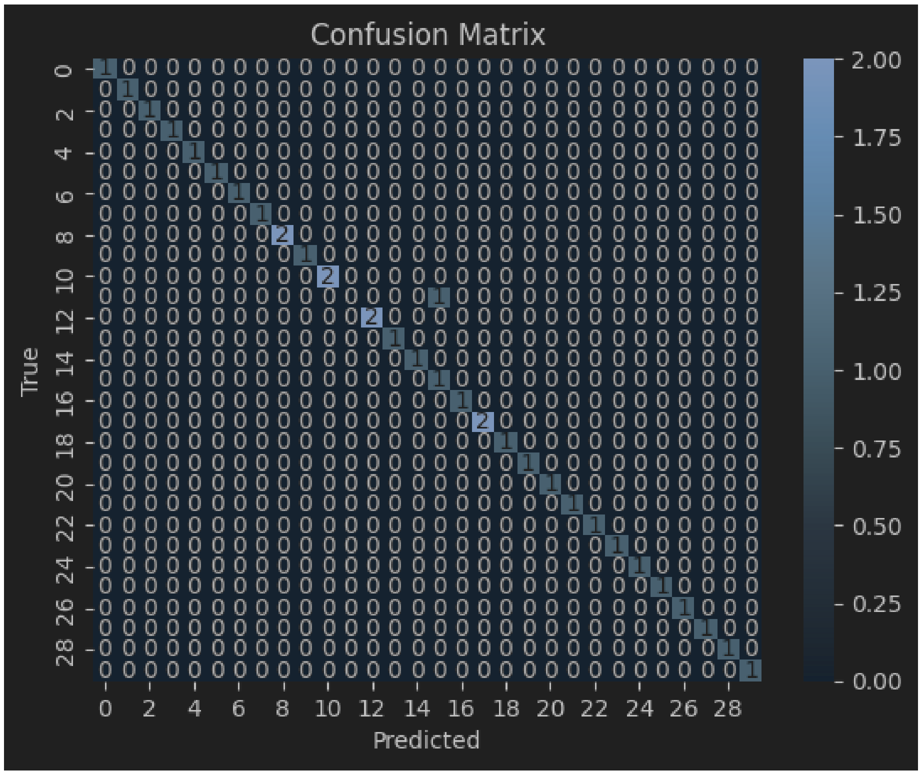
Random Forest uses bagging to select different variations of the training data to ensure decision trees are sufficiently different. However, Extra Trees uses the entire dataset to train decision trees.

```
clf = Pipeline([
  ('feature_selection', SelectFromModel(estimator=LogisticRegression())),
  ('classification', ExtraTreesClassifier(n_estimators=65, random_state=43))
])
clf.fit(X_train, y_train)
```

```
# Predictions and evaluation
y_pred = clf.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

This classifier performs classification with an accuracy of 97%.

Here is an overview of the **Confusion Matrix**:



Clustering

Before performing clustering, we do some pre-processing on the data by **Standardizing Features** and Applying **PCA**.

RobustScaler standardizes features by removing the median and scaling the data according to the interquartile range. **np.clip** limits (clips) the values in an array. Here, it ensures that all values in **X_scaled** are within the range of -3 to 3. This is another step to mitigate the impact of outliers. **PCA** transforms the data into a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

```
# Standardizing the features
scaler = RobustScaler()
X_scaled = scaler.fit_transform(X)

# Clip outliers
```

```
X_scaled = np.clip(X_scaled, -3, 3)


# Applying PCA
pca = PCA(n_components=11)
X_pca = pca.fit_transform(X_scaled)
```

For clustering, after several examinations, we found that **Agglomerative model** performs better compared to k-means, GMM, DBSCAN and several algorithms.

__AgglomerativeClustering__ is a type of hierarchical clustering method that builds nested clusters by repeatedly merging or splitting them. In this case, it is used to group the data into clusters.

n_clusters=30: Specifies the number of clusters to find.(Since 30 classes are available in the dataset and we also like to find the relation between them.)

fit_predict: This method fits the agglomerative clustering model to the data and then returns the cluster labels for each data point.

```
agglo = AgglomerativeClustering(n_clusters=30)
agglo_labels = agglo.fit_predict(X_pca)

# Evaluate Agglomerative Clustering
homogeneity_agglo = homogeneity_score(y, agglo_labels)
completeness_agglo = completeness_score(y, agglo_labels)
v_measure_agglo = v_measure_score(y, agglo_labels)
ari_agglo = adjusted_rand_score(y, agglo_labels)
nmi_agglo = normalized_mutual_info_score(y, agglo_labels)
silhouette_avg = silhouette_score(X_pca, agglo_labels)

print(f'Silhouette Score: {silhouette_avg}')
print(f'Agglomerative Clustering Homogeneity: {homogeneity_agglo:.2f}')
print(f'Agglomerative Clustering Completeness: {completeness_agglo:.2f}')
print(f'Agglomerative Clustering V-measure: {v_measure_agglo:.2f}')
print(f'Agglomerative Clustering Adjusted Rand Index: {ari_agglo:.2f}')
print(f'Agglomerative Clustering Normalized Mutual Information:
{nmi_agglo:.2f}')
```

the model’s evaluation:

1. **Silhouette Score: 0.2824671604156972 (**Moderate cluster separation, with some overlap**)**
2. **Agglomerative Clustering Homogeneity: 0.74 (**Clusters are fairly pure, mostly containing members of a single class**)**
3. **Agglomerative Clustering Completeness: 0.77 (**Members of the same class are largely grouped together**)**
4. **Agglomerative Clustering V-measure: 0.75 (**Balanced measure of homogeneity and completeness; fairly good clustering**)**

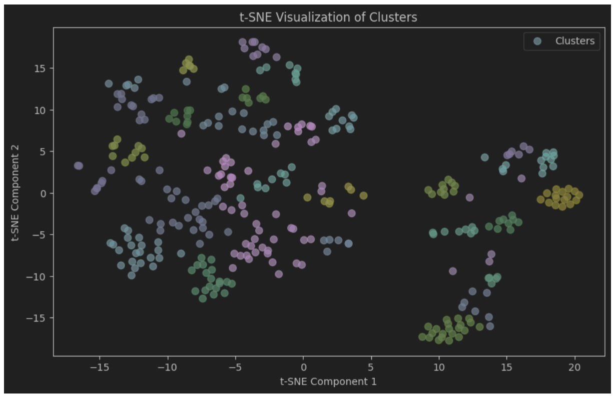
5. **Agglomerative Clustering Adjusted Rand Index: 0.42 (**Moderate agreement with true labels, indicating reasonable clustering**)**
6. **Agglomerative Clustering Normalized Mutual Information: 0.75 (**High level of agreement between clusters and true labels; informative clustering**)**

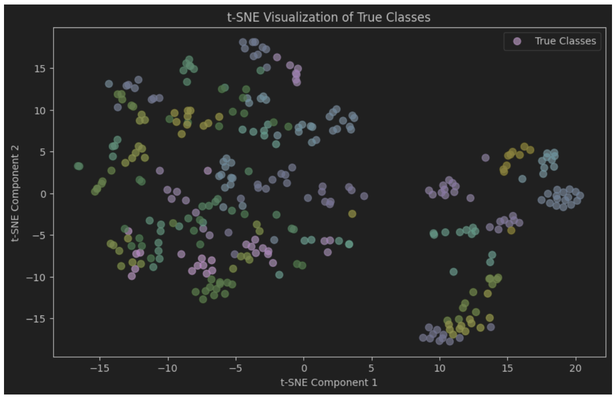
In the end, We Use t-SNE to visualize the clusters and true classes in 2D space, providing a visual representation of how well the clustering algorithm performed.

```
# Visualize the clusters and true classes using t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(10, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=agglo_labels, cmap='viridis', s=50,
alpha=0.7, label='Clusters')
plt.title('t-SNE Visualization of Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis', s=50, alpha=0.7,
label='True Classes')
plt.title('t-SNE Visualization of True Classes')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()
```


t-SNE Visualization of Clusters
Clusters
t-SNE Component 1
t-SNE Component 2


t-SNE Visualization of True Classes
True Classes
t-SNE Component 1
t-SNE Component 2