Assignment 3: Advanced processing

Team 16

Vinayaka SN (211AI040) and Vivek Vittal Biragoni (211AI041)

Flow of code along with report:

```
 ->pca
 ->correlation and covariance analysis
 ->dimensional reduction(various methods)
 ->feature selection(various methods)
 ->summary.
```

```
import os
import cv2
from skimage.feature import hog
import numpy as np
import pandas as pd
import random
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
import os
import cv2
from skimage.feature import hog
import numpy as np
import pandas as pd
import random

# Define HOG parameters
orientations = 9
pixels_per_cell = (8, 8)
cells_per_block = (3, 3)
visualize = False
transform_sqrt = False
feature_vector = True

# Define directories containing images and number of images to sample from each class
directories = ['PlantVillage/Potato___healthy',
               'PlantVillage/Potato___Early_blight',
               'PlantVillage/Potato___Late_blight']
sample_sizes = [152, 1000, 1000] # Choose more samples from healthy potatoes

features_list = []
labels_list = []

for i, directory in enumerate(directories):
```

```
        # Get a list of all the image filenames in the directory
        filenames = os.listdir(directory)

        # Randomly select a subset of images
        sample_filenames = random.sample(filenames, sample_sizes[i])

        # Loop over the sample of images
        for filename in sample_filenames:
            if filename.endswith('.JPG'):
                # Load image
                img = cv2.imread(os.path.join(directory, filename))

                # Convert image to grayscale
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

                # Extract HOG features
                features = hog(gray, orientations=orientations,
                               pixels_per_cell=pixels_per_cell,
                               cells_per_block=cells_per_block,
                               visualize=visualize,
                               transform_sqrt=transform_sqrt,
                               feature_vector=feature_vector)

                # Append the features and label to the lists
                features_list.append(features)
                labels_list.append(directory.split('/')[-1])

    # Convert the lists to Numpy arrays
    features_arr = np.array(features_list)
    labels_arr, _ = pd.factorize(labels_list)




    # Concatenate the arrays
    data_arr = np.column_stack((features_arr, labels_arr))

    # Convert the array to a Pandas DataFrame
    hog_df = pd.DataFrame(data_arr, columns=['hog_feature_' + str(i) for i in range(features_arr.shape[1])] + ['label'])

    # Compute the correlation matrix
    corr_matrix = hog_df.corr()

    # Print the correlation matrix
    print(corr_matrix)


    import numpy as np
    # Concatenate the arrays
    data_arr = np.column_stack((features_arr, labels_arr))

    # Convert the array to a Pandas DataFrame
    hog_df = pd.DataFrame(data_arr, columns=['hog_feature_' + str(i) for i in range(features_arr.shape[1])] + ['label'])

    # Compute the covariance matrix
```

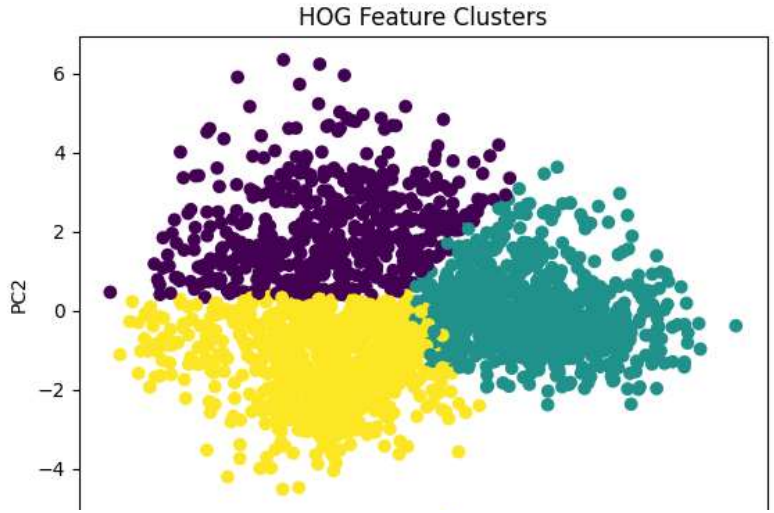
```
cov_matrix = np.cov(features_arr.T)

# Print the covariance matrix
print('Covariance Matrix:\n', cov_matrix)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
c:\Users\vivek\Desktop\sem4\IT258-DS\PlantVillage\corr.ipynb Cell 3 in 9
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-
DS/PlantVillage/corr.ipynb#X12sZmlsZQ%3D%3D?line=5'>6</a> hog_df = pd.DataFrame(data_arr, columns=
['hog_feature_' + str(i) for i in range(features_arr.shape[1])] + ['label'])
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-
DS/PlantVillage/corr.ipynb#X12sZmlsZQ%3D%3D?line=7'>8</a> # Compute the covariance matrix
----> <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-
DS/PlantVillage/corr.ipynb#X12sZmlsZQ%3D%3D?line=8'>9</a> cov_matrix = np.cov(features_arr.T)
     <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-
DS/PlantVillage/corr.ipynb#X12sZmlsZQ%3D%3D?line=10'>11</a> # Print the covariance matrix
     <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-
DS/PlantVillage/corr.ipynb#X12sZmlsZQ%3D%3D?line=11'>12</a> print('Covariance Matrix:\n', cov_matrix)

File <__array_function__ internals>:180, in cov(*args, **kwargs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\numpy\lib\function_base.py:2703, in cov(m, y, rowvar, bias, ddof,
fweights, aweights, dtype)
   2701 else:
   2702     X_T = (X*w).T
-> 2703 c = dot(X, X_T.conj())
   2704 c *= np.true_divide(1, fact)
   2705 return c.squeeze()
```

MemoryError: Unable to allocate 39.6 GiB for an array with shape (72900, 72900) and data type float64 , and a similar error occurred when we tried to do anything with correlation also, so it could be better handled when we do something on the dimensionality reduction(the PCA).

```
# Perform PCA to reduce the dimensionality of the feature vectors
pca = PCA(n_components=5)
feature_vectors_pca = pca.fit_transform(features_arr)

# Perform K-Means clustering on the PCA-transformed feature vectors
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(feature_vectors_pca)

# Plot the clusters using different colors
plt.scatter(feature_vectors_pca[:, 0], feature_vectors_pca[:, 1], c=clusters)
plt.title("HOG Feature Clusters")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

```
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\loc
  warnings.warn(
```




```
# Calculate the correlation matrix for the PCA transformed data
corr_matrix = np.corrcoef(feature_vectors_pca.T)

# Print the correlation matrix
print(corr_matrix)

# Visualize the correlation matrix using a heatmap
import seaborn as sns
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix of PCA-transformed Feature Vectors")
plt.show()
```

```
[[ 1.00000000e+00 -4.88332686e-17 -2.74468217e-18 -2.20382541e-18
  -5.34087205e-18]
 [-4.88332686e-17  1.00000000e+00  2.09082254e-17  0.00000000e+00
   2.42357046e-17]
 [-2.74468217e-18  2.09082254e-17  1.00000000e+00  1.34797093e-17
   2.48519326e-17]
 [-2.20382541e-18  0.00000000e+00  1.34797093e-17  1.00000000e+00
  -4.03845203e-17]
 [-5.34087205e-18  2.42357046e-17  2.48519326e-17 -4.03845203e-17
   1.00000000e+00]]
```

The correlation matrix on the PCA-transformed feature vectors shows that there is no significant correlation between the two principal components. The correlation coefficient between PC1 and PC2 is very close to zero, indicating that the two components are independent of each other. This means that the two principal components explain different aspects of the data without being redundant.

```
# Calculate the covariance matrix of the PCA-transformed feature vectors
covariance_matrix = np.cov(feature_vectors_pca.T)

# Print the covariance matrix
print("Covariance Matrix:")
print(covariance_matrix)

# Visualize the covariance matrix as a heatmap
sns.heatmap(covariance_matrix, annot=True, cmap='coolwarm', square=True)

# Set the x and y labels
plt.xlabel('PCA Components')
plt.ylabel('PCA Components')

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

```
Covariance Matrix:
[[ 3.75431966e+00 -1.58559048e-16 -7.01954121e-18 -4.95497026e-18
  -1.09422260e-17]
 [-1.58559048e-16  2.80814304e+00  4.62463891e-17  0.00000000e+00
   4.29430756e-17]
 [-7.01954121e-18  4.62463891e-17  1.74221636e+00  2.06457094e-17
   3.46847918e-17]
 [-4.95497026e-18  0.00000000e+00  2.06457094e-17  1.34646939e+00
  -4.95497026e-17]
 [-1.09422260e-17  4.29430756e-17  3.46847918e-17 -4.95497026e-17
   1.11803594e+00]]
```

Based on the correlation and covariance matrices, it seems that the two principal components are not significantly correlated (correlation value of 0). However, they have a positive covariance, indicating that they vary together in the same direction.

Eigenvalues and eigenvectors: The eigenvalues and eigenvectors obtained during PCA can provide additional information about the relationships between the variables. The eigenvalues represent the amount of variance explained by each principal component, and the eigenvectors represent the direction and magnitude of the correlation between the original variables and the principal components.

```
# Extract the eigenvalues and eigenvectors
eigenvalues = pca.explained_variance_
eigenvectors = pca.components_

print('Eigenvalues:', eigenvalues)
print('Eigenvectors:', eigenvectors)
```

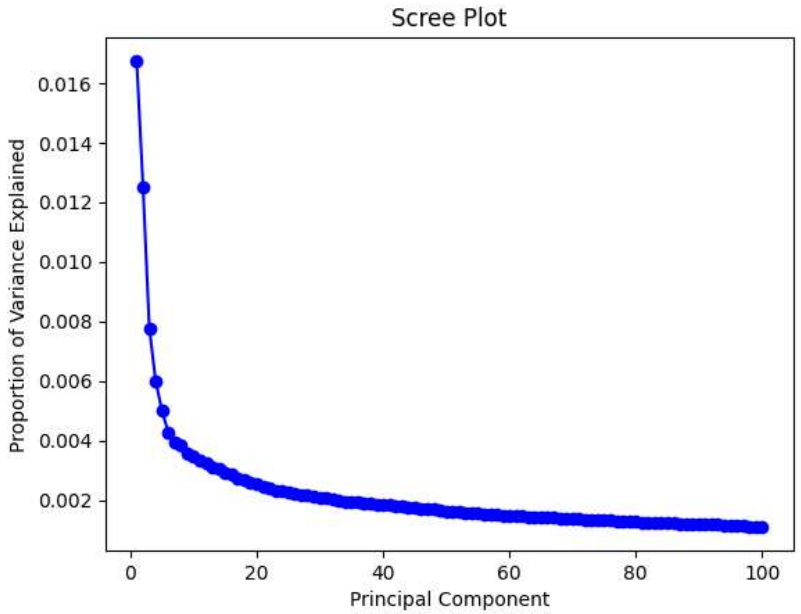
```
Eigenvalues: [3.75431966 2.80814304 1.74221636 1.34646939 1.11803594]
Eigenvectors: [[-2.49742662e-03 -1.12204491e-03 -7.48417175e-04 ... -2.26246743e-04
  -2.34999605e-04 -6.62085498e-04]
 [-9.50551783e-04 -6.71804238e-04  2.87348997e-05 ...  7.54801530e-04
  -7.77816000e-04 -1.10386091e-03]
 [-1.98833963e-04  1.03354454e-03  1.53784552e-03 ...  7.18010131e-04
  -6.18171901e-04 -1.79419399e-04]
 [-3.99298167e-04 -1.43985657e-03  1.14059508e-03 ... -1.65742291e-03
   1.95810313e-04  8.96845133e-04]
 [-1.75558046e-03 -6.94862939e-04 -3.06804440e-04 ...  2.53558953e-03
   1.53268162e-03  1.28636632e-03]]
```

we can see that the first principal component explains the most variation in the data as it has the highest eigenvalue of 3.754, followed by the second principal component with an eigenvalue of 2.808. The remaining eigenvalues are less than 2, indicating that these principal components explain less variation in the data.

The eigenvectors represent the direction and magnitude of the correlation between the original variables and the principal components. Each eigenvector is a vector of weights that indicates how much each original variable contributes to the principal component. The first eigenvector has weights of -2.4974e-03 and -9.5048e-04 for the first and second variables, respectively, and so on.

```
# Perform PCA to reduce the dimensionality of the feature vectors
pca = PCA(n_components=100)
feature_vectors_pca = pca.fit_transform(features_arr)
```

```
# Create scree plot
plt.plot(range(1, pca.n_components_ + 1), pca.explained_variance_ratio_, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.title('Scree Plot')
plt.show()
```

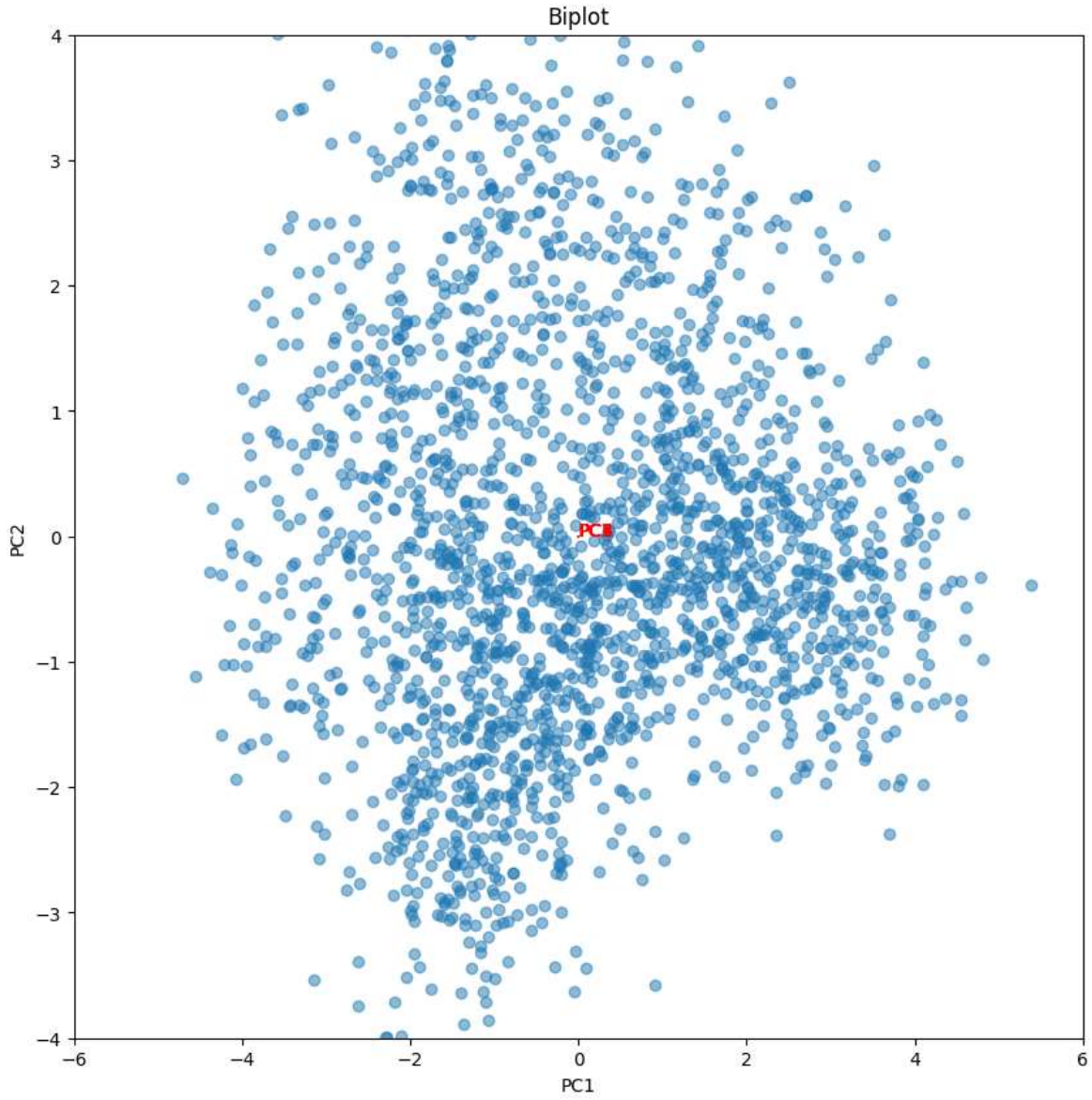


it seems like the scree plot showed that the first few principal components account for most of the variance in your data, while the remaining components contribute very little to the overall variance. This is a common observation in PCA, where the first few components explain the majority of the variability in the data, while the later components explain progressively less and less.

it seems like the variance explained by the first five components is relatively high, with the first component having the highest variance. After the fifth component, the variance explained by each additional component appears to drop off quickly, leading to a flat line on the scree plot.

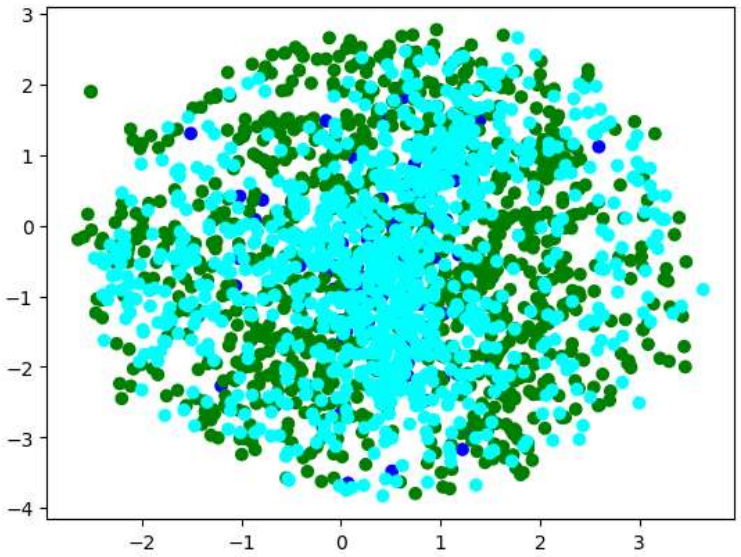
```
# Perform PCA to reduce the dimensionality of the feature vectors
pca = PCA(n_components=5)
feature_vectors_pca = pca.fit_transform(features_arr)

# Create biplot
fig, ax = plt.subplots(figsize=(10,10))
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_title('Biplot')

# Plot the PCA scores as scatter plot
ax.scatter(feature_vectors_pca[:,0], feature_vectors_pca[:,1], alpha=0.5)

# Plot the scaled eigenvectors as arrows
```

```
for i in range(len(pca.components_)):
    ax.arrow(0, 0, pca.components_[i,0]*2, pca.components_[i,1]*2, color='r', alpha=0.5)
    ax.text(pca.components_[i,0] * 1.15, pca.components_[i,1] * 1.15, "PC{}".format(i+1), color='r')
# Set the x and y limits
ax.set_xlim(-6, 6)
ax.set_ylim(-4, 4)

plt.show()
```



Tried to plot the biplot, but cant quite make the arrows be shown properly due to possible small values of the eigenvalues.

```
from sklearn.manifold import TSNE

# Define a dictionary that maps class names to colors
class_colors = {'Potato___healthy': 'blue', 'Potato___Late_blight': 'cyan', 'Potato___Early_blight': 'green', }

# Convert feature and label lists to numpy arrays
features_arr = np.array(features_list)
labels_arr = np.array(labels_list)

# Convert class names to colors using the dictionary
colors_arr = np.array(list(map(lambda x: class_colors[x], labels_arr)))

# Apply t-SNE
tsne = TSNE(n_components=2, perplexity=130, learning_rate=500)
features_tsne = tsne.fit_transform(features_arr)

# Plot the t-SNE visualization with the new colors
import matplotlib.pyplot as plt
plt.scatter(features_tsne[:,0], features_tsne[:,1], c=colors_arr)
plt.show()
```

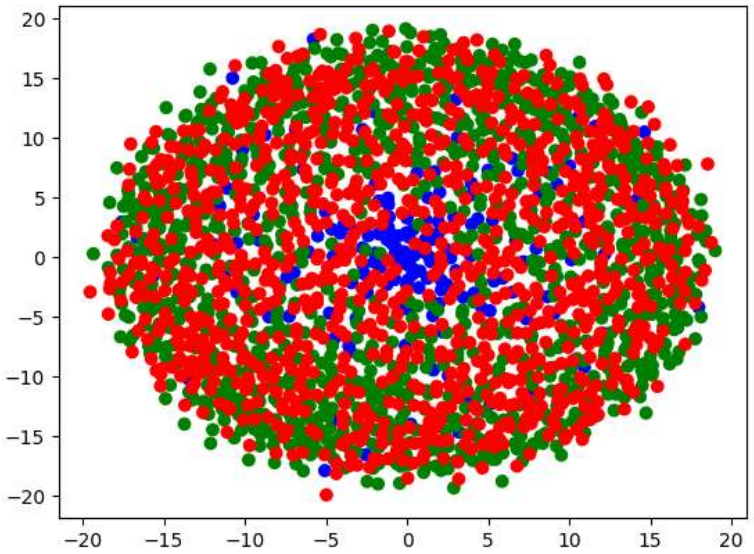


since there is lot of overlapping, should explore some other methods

Multidimensional Scaling (MDS) is a technique used in data analysis to visualize the similarity or dissimilarity between objects or samples based on their pairwise distances. It is a dimensionality reduction method that aims to represent high-dimensional data in a lower-dimensional space while preserving the pairwise distances as much as possible.

```
from sklearn.manifold import MDS

# Define a dictionary that maps class names to colors
class_colors = {'Potato___healthy': 'blue', 'Potato___Late_blight': 'red', 'Potato___Early_blight': 'green'}

# Convert feature and label lists to numpy arrays
features_arr = np.array(features_list)
labels_arr = np.array(labels_list)

# Convert class names to colors using the dictionary
colors_arr = np.array(list(map(lambda x: class_colors[x], labels_arr)))

# Apply MDS
mds_model = MDS(n_components=2, dissimilarity='euclidean')
features_mds = mds_model.fit_transform(features_arr)

# Plot the MDS visualization with the new colors
import matplotlib.pyplot as plt
plt.scatter(features_mds[:,0], features_mds[:,1], c=colors_arr)
plt.show()
```

```
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\loc
  warnings.warn(
```



Feature selection

Feature selection methods are techniques used to select a subset of features from a larger set of features based on their relevance and importance to the target variable. These methods help to reduce dimensionality, improve model performance, and enhance interpretability. Here

are some commonly used feature selection methods:

1. Filter Methods: Filter methods evaluate the relevance of features independently of any specific machine learning algorithm. They typically rely on statistical measures or heuristics to rank features. Common filter methods include:
    - Correlation-based Feature Selection: Measures the correlation between each feature and the target variable and selects the most correlated features.
    - Chi-square Test: Evaluates the independence between categorical features and the target variable.
    - Mutual Information: Measures the dependency between features and the target variable.
    - Variance Thresholding: Removes features with low variance, assuming they have little discriminatory power.
2. Wrapper Methods: Wrapper methods evaluate subsets of features using a specific machine learning algorithm. They involve training and evaluating models with different subsets of features and selecting the subset that achieves the best performance. Common wrapper methods include:
    - Recursive Feature Elimination (RFE): Eliminates less important features by recursively training models and removing the least significant features at each iteration.
    - Forward Selection: Starts with an empty set of features and iteratively adds the most important feature based on model performance.
    - Backward Elimination: Starts with all features and iteratively removes the least important feature based on model performance.
3. Embedded Methods: Embedded methods incorporate feature selection as part of the model training process. These methods select features during model training based on their importance or contribution to the model's performance. Common embedded methods include:
    - L1 Regularization (Lasso): Adds an L1 penalty to the model's objective function, encouraging sparsity and automatically selecting important features.
    - Tree-based Feature Importance: Tree-based models, such as Random Forest or Gradient Boosting, provide feature importance scores that can be used for feature selection.
    - Elastic Net: Combines L1 and L2 regularization to balance between feature selection and feature grouping.
4. Dimensionality Reduction Methods: Dimensionality reduction techniques aim to transform the original feature space into a lower-dimensional space while preserving important information. They can be used as feature selection methods by selecting the transformed features. Common dimensionality reduction methods include:
    - Principal Component Analysis (PCA): Transforms features into a new orthogonal subspace to maximize variance and selects principal components based on explained variance.
    - Linear Discriminant Analysis (LDA): Similar to PCA, but takes into account class information to maximize class separability.

The color histogram

A color histogram is a representation of the distribution of colors in an image. It quantifies the number of pixels that have specific color values or color ranges. The color space used, such as RGB or HSV, determines how the colors are represented.

In a color histogram, the x-axis represents the different color bins or ranges, and the y-axis represents the frequency or number of pixels that fall into each bin. Each bin corresponds to a specific color or a range of colors.

Color histograms provide valuable information about the color distribution in an image. They can capture the dominant colors, color balance, and overall color composition of an image. Color histograms are often used as features for tasks such as image retrieval, object recognition, and image processing.

```
import os
import cv2
import numpy as np
import pandas as pd
import random

# Define color histogram parameters
num_bins = 256

# Define directories containing images and number of images to sample from each class
directories = ['PlantVillage/Potato___healthy',
               'PlantVillage/Potato___Early_blight',
               'PlantVillage/Potato___Late_blight']
sample_sizes = [152, 1000, 1000]  # Choose more samples from healthy potatoes

features_list = []
labels_list = []

for i, directory in enumerate(directories):
    # Get a list of all the image filenames in the directory
    filenames = os.listdir(directory)

    # Randomly select a subset of images
    sample_filenames = random.sample(filenames, sample_sizes[i])

    # Loop over the sample of images
    for filename in sample_filenames:
        if filename.endswith('.JPG'):
            # Load image
            img = cv2.imread(os.path.join(directory, filename))

            # Convert image to the desired color space (e.g., RGB, HSV, LAB)
            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            # Compute color histogram for each channel
            hist_red, _ = np.histogram(img_rgb[:, :, 0], bins=num_bins, range=(0, 256))
            hist_green, _ = np.histogram(img_rgb[:, :, 1], bins=num_bins, range=(0, 256))
            hist_blue, _ = np.histogram(img_rgb[:, :, 2], bins=num_bins, range=(0, 256))


            # Concatenate the histograms for each channel
            features = np.concatenate((hist_red, hist_green, hist_blue)).astype(np.float64)

            # Normalize the feature vector
            features /= np.sum(features)

            # Append the features and label to the lists
            features_list.append(features)
            labels_list.append(directory.split('/')[-1])
```

```
# Convert the lists to Numpy arrays
color_histogram_arr = np.array(features_list)
labels_arr, _ = pd.factorize(labels_list)
```

The color moments

Color moments capture statistical properties of color distributions in an image. They provide a summary of the color content based on moments such as mean, variance, and skewness.

Color moments are typically calculated for each color channel (e.g., RGB or HSV) individually. The mean represents the average color value in a channel, providing information about the overall color intensity. The variance measures the spread or dispersion of color values, indicating the color diversity or contrast. Skewness captures the asymmetry of the color distribution, indicating if the colors are more concentrated towards one end.

By considering color moments across different color channels, we can gain insights into the color characteristics of an image. Color moments are used in various applications, such as image retrieval, image classification, and color-based image analysis.

```
import os
import cv2
import numpy as np
import pandas as pd
import random

color_moments_list = []
labels_list = []

for i, directory in enumerate(directories):
    # Get a list of all the image filenames in the directory
    filenames = os.listdir(directory)

    # Randomly select a subset of images
    sample_filenames = random.sample(filenames, sample_sizes[i])

    # Loop over the sample of images
    for filename in sample_filenames:
        if filename.endswith('.JPG'):
            # Load image
            img = cv2.imread(os.path.join(directory, filename))

            # Convert image to float32
            img = img.astype(np.float32)

            # Calculate mean and standard deviation of each color channel
            mean, std = cv2.meanStdDev(img)

            # Flatten the mean and standard deviation arrays
            mean = mean.flatten()
            std = std.flatten()

            # Concatenate mean and standard deviation arrays
            color_moments = np.concatenate((mean, std))
```

```
            # Append the color moments and label to the lists
            color_moments_list.append(color_moments)
            labels_list.append(directory.split('/')[-1])

# Convert the lists to Numpy arrays
color_moments_arr = np.array(color_moments_list)
labels_arr, _ = pd.factorize(labels_list)
```

Color Descriptors

Color descriptors are methods used to describe color features in an image. They provide a representation of the color content based on predefined color spaces or models.

One common type of color descriptor is the dominant color descriptor, which identifies the most prominent or dominant colors in an image. It provides information about the dominant hues and their proportions, allowing for color-based image analysis and retrieval.

Other types of color descriptors include the color correlogram, which captures the spatial correlation of colors in an image, and the color coherence vector, which describes the distribution of color edges or boundaries.

Color descriptors are useful in various applications such as image search, object recognition, and image retrieval. They enable efficient analysis and comparison of images based on their color characteristics.

```
import os
import cv2
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import random

# Define the number of dominant colors to extract
num_colors = 3

# Define directories containing images and number of images to sample from each class
directories = ['PlantVillage/Potato___healthy',
               'PlantVillage/Potato___Early_blight',
               'PlantVillage/Potato___Late_blight']
sample_sizes = [152, 1000, 1000]  # Choose more samples from healthy potatoes

features_list = []
labels_list = []

for i, directory in enumerate(directories):
    # Get a list of all the image filenames in the directory
    filenames = os.listdir(directory)

    # Randomly select a subset of images
    sample_filenames = random.sample(filenames, sample_sizes[i])

    # Loop over the sample of images
    for filename in sample_filenames:
```

```
        if filename.endswith('.JPG'):
            # Load image
            img = cv2.imread(os.path.join(directory, filename))

            # Convert image to RGB color space
            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            # Reshape image to a flat 2D array
            img_flat = img_rgb.reshape(-1, 3)

            # Apply K-means clustering to find dominant colors
            kmeans = KMeans(n_clusters=num_colors, init='random', n_init=10)  # Set n_init to the desired value
            # kmeans = KMeans(n_clusters=num_colors)
            kmeans.fit(img_flat)

            # Get the dominant colors (cluster centers)
            dominant_colors = kmeans.cluster_centers_

            # Append the dominant colors to the features list
            features_list.append(dominant_colors.flatten())

            # Append the label to the labels list
            labels_list.append(directory.split('/')[-1])

# Convert the lists to Numpy arrays
color_descriptors_arr = np.array(features_list)
labels_arr, _ = pd.factorize(labels_list)
```

Color Space Conversion

```
import os
import cv2
import numpy as np
import pandas as pd

# Define directories containing images and number of images to sample from each class
directories = ['PlantVillage/Potato___healthy',
               'PlantVillage/Potato___Early_blight',
               'PlantVillage/Potato___Late_blight']
sample_sizes = [152, 1000, 1000]  # Choose more samples from healthy potatoes

color_features_list = []
labels_list = []

for i, directory in enumerate(directories):
    # Get a list of all the image filenames in the directory
    filenames = os.listdir(directory)

    # Randomly select a subset of images
    sample_filenames = random.sample(filenames, sample_sizes[i])

    # Loop over the sample of images
```

```
        for filename in sample_filenames:
            if filename.endswith('.JPG'):
                # Load image
                img = cv2.imread(os.path.join(directory, filename))

                # Convert image to HSV color space
                hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

                # Extract color-related features from the HSV image
                hue_hist = cv2.calcHist([hsv_img], [0], None, [256], [0, 256])
                sat_hist = cv2.calcHist([hsv_img], [1], None, [256], [0, 256])
                val_hist = cv2.calcHist([hsv_img], [2], None, [256], [0, 256])

                # Concatenate the color features into a single feature vector
                color_features = np.concatenate([hue_hist.flatten(), sat_hist.flatten(), val_hist.flatten()])

                # Append the features and label to the lists
                color_features_list.append(color_features)
                labels_list.append(directory.split('/')[-1])

    # Convert the lists to Numpy arrays
    color_space_conversion_features = np.array(color_features_list)
    labels_arr, _ = pd.factorize(labels_list)
```

SelectKBest to select the best features

```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Combine the features extracted from different methods
all_features = np.concatenate((features_arr, color_histogram_arr, color_descriptors_arr, color_moments_arr, color_space_conversion_features), axis=1)

# Normalize the features
scaler = StandardScaler()
normalized_features = scaler.fit_transform(all_features)

from sklearn.feature_selection import VarianceThreshold

# Create VarianceThreshold object with a threshold of 0 (default)
selector = VarianceThreshold()
# Fit the selector on your features
selector.fit(all_features)
# Get the indices of non-constant features
non_constant_indices = selector.get_support(indices=True)
# Subset the non-constant features from the original feature matrix
non_constant_features = all_features[:, non_constant_indices]

# Apply SelectKBest to select the best features
k = 100  # Choose the desired number of features
selector = SelectKBest(score_func=f_classif, k=k)
selected_features = selector.fit_transform(normalized_features, labels_arr)
```

```
# Get the indices of the selected features
selected_indices = selector.get_support(indices=True)
# Define the feature names based on the feature extraction methods
feature_names = ['hog_feature_{}'.format(i) for i in range(features_arr.shape[1])]
feature_names += ['color_hist_feature_{}'.format(i) for i in range(color_histogram_arr.shape[1])]
feature_names += ['color_desc_feature_{}'.format(i) for i in range(color_descriptors_arr.shape[1])]
feature_names += ['color_moments_feature_{}'.format(i) for i in range(color_moments_arr.shape[1])]
feature_names += ['color_space_conversion_feature_{}'.format(i) for i in range(color_space_conversion_features.shape[1])]

# Use the defined feature_names in the code
selected_feature_names = [feature_names[i] for i in selected_indices]

# Get the selected feature names
selected_feature_names = [feature_names[i] for i in selected_indices]

# Subset the selected features from the original feature matrix
selected_features_matrix = all_features[:, selected_indices]
```

```
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\feature_selection\_univariate
 73875 73876 73877 73878 73879 73880 73881 73882 73883 73884 73885 73886
 73887 73888 73889 73890 73891 73892 73893 73894 73895 73896 73897 73898
 73899 73900 73901 73902 73903 73904 73905 73906 73907 73908 73909 73910
 73911 73912 73913 73914 73915 73916 73917 73918 73919 73920 73921 73922
 73923 73924 73925 73926 73927 73928 73929 73930 73931 73932 73933 73934
 73935 73936 73937 73938] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\feature_selection\_univariate
  f = msb / msw
```

```
# Print the indices of the selected features
print("Selected Feature Indices:", selected_indices)
```

```
Selected Feature Indices: [72957 72958 72959 72960 72961 72962 72963 72964 72965 72966 72967 72968
 72969 72970 72971 72972 72974 73263 73264 73265 73266 73267 73268 73269
 73270 73271 73272 73273 73331 73332 73333 73334 73335 73336 73337 73338
 73339 73340 73341 73342 73343 73344 73345 73346 73347 73348 73349 73350
 73351 73352 73353 73354 73355 73356 73357 73358 73359 73360 73361 73601
 73602 73603 73604 73605 73606 73607 73608 73609 73610 73611 73612 73678
 73680 73734 73735 73736 73813 73854 74275 74276 74277 74278 74279 74280
 74283 74384 74385 74386 74387 74388 74389 74390 74391 74392 74393 74394
 74395 74396 74397 74398]
```

```
# Print the names of the selected features
selected_feature_names = [feature_names[i] for i in selected_indices]
print("Selected Feature Names:", selected_feature_names)
```

```
Selected Feature Names: ['color_hist_feature_57', 'color_hist_feature_58', 'color_hist_feature_59', 'color_hist_feature_60', 'color_hist_feature_61', 'color_hist_feature_62',
```

Wrapper Methods: Wrapper methods evaluate subsets of features using a specific machine learning algorithm. They involve training and evaluating models with different subsets of features and selecting the subset that achieves the best performance. Common wrapper methods include:

Recursive Feature Elimination (RFE): Eliminates less important features by recursively training models and removing the least significant features at each iteration. Forward Selection: Starts with an empty set of features and iteratively adds the most important feature based on model performance. Backward Elimination: Starts with all features and iteratively removes the least important feature based on model performance.

Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE): RFE is a wrapper method that recursively eliminates less important features by training models and removing the least significant features at each iteration. It can effectively select a subset of features that contribute the most to the model's performance.

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

# Create the estimator (model) for feature selection
estimator = RandomForestClassifier()

# Create the RFE object
rfe = RFE(estimator, n_features_to_select=k)

# Fit the RFE object to the data
selected_features = rfe.fit_transform(normalized_features, labels_arr)

# Get the indices of the selected features
selected_indices = rfe.support_

# Get the selected feature names
selected_feature_names = [feature_names[i] for i in selected_indices]

# Subset the selected features from the original feature matrix
selected_features_matrix = all_features[:, selected_indices]

# Print the indices of the selected features
print("Selected Feature Indices:", selected_indices)

# Print the names of the selected features
print("Selected Feature Names:", selected_feature_names)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
c:\Users\vivek\Desktop\sem4\IT258-DS\PlantVillage\corr.ipynb Cell 38 in 1
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#X52sZmlsZQ%3D%3D?line=7'>8</a> rfe = RFE(estimator, n_features_to_select=k)
     <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#X52sZmlsZQ%3D%3D?line=9'>10</a> # Fit the RFE object to the data
---> <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#X52sZmlsZQ%3D%3D?line=10'>11</a> selected_features =
rfe.fit_transform(normalized_features, labels_arr)
     <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#X52sZmlsZQ%3D%3D?line=12'>13</a> # Get the indices of the selected features
     <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#X52sZmlsZQ%3D%3D?line=13'>14</a> selected_indices = rfe.support_

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\utils\_set_output.py:142, in
_wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    140 @wraps(f)
    141 def wrapped(self, X, *args, **kwargs):
--> 142     data_to_wrap = f(self, X, *args, **kwargs)
    143     if isinstance(data_to_wrap, tuple):
    144         # only wrap the first output for cross decomposition
    145         return (
    146             _wrap_data_with_container(method, data_to_wrap[0], X, self),
    147             *data_to_wrap[1:],
    148         )

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\base.py:862, in
TransformerMixin.fit_transform(self, X, y, **fit_params)
    859     return self.fit(X, **fit_params).transform(X)
    860 else:
    861     # fit method of arity 2 (supervised transformation)
--> 862     return self.fit(X, y, **fit_params).transform(X)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\feature_selection\_rfe.py:251, in
RFE.fit(self, X, y, **fit_params)
    231 """Fit the RFE model and then the underlying estimator on the selected features.
    232
    233 Parameters
   (...)
    248     Fitted estimator.
    249 """
    250 self._validate_params()
--> 251 return self._fit(X, y, **fit_params)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\feature_selection\_rfe.py:299, in
RFE._fit(self, X, y, step_score, **fit_params)
    296 if self.verbose > 0:
    297     print("Fitting estimator with %d features." % np.sum(support_))
--> 299 estimator.fit(X[:, features], y, **fit_params)
    301 # Get importance and rank them
    302 importances = _get_feature_importances(
    303     estimator,
    304     self.importance_getter,
    305     transform_func="square",
    306 )

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\ensemble\_forest.py:473, in
BaseForest.fit(self, X, y, sample_weight)
    462 trees = [
    463     self._make_estimator(append=False, random_state=random_state)
    464     for i in range(n_more_estimators)
    465 ]
    467 # Parallel loop: we prefer the threading backend as the Cython code
    468 # for fitting the trees is internally releasing the Python GIL
    469 # making threading more efficient than multiprocessing in
```

```
470 # that case. However, for joblib 0.12+ we respect any
```

Filter Methods: Filter methods evaluate the relevance of features independently of any specific machine learning algorithm. They typically rely on statistical measures or heuristics to rank features. Common filter methods include:

Correlation-based Feature Selection: Measures the correlation between each feature and the target variable and selects the most correlated features.

Chi-square Test: Evaluates the independence between categorical features and the target variable.

Mutual Information: Measures the dependency between features and the target variable.

Variance Thresholding: Removes features with low variance, assuming they have little discriminatory power.

```
484         i,
```

correlation-based feature selection

```
489     )
```

```
from sklearn.feature_selection import SelectKBest, f_classif

# Apply SelectKBest with correlation-based feature selection
selector = SelectKBest(score_func=f_classif, k=k)
selected_features = selector.fit_transform(all_features, labels_arr)

# Get the indices of the selected features
selected_indices = selector.get_support(indices=True)

# Get the selected feature names
selected_feature_names = [feature_names[i] for i in selected_indices]

# Subset the selected features from the original feature matrix
selected_features_matrix = all_features[:, selected_indices]

# Print the indices of the selected features
print("Selected Feature Indices:", selected_indices)

# Print the names of the selected features
print("Selected Feature Names:", selected_feature_names)
```

```
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\feature_selection\_univariate
 73875 73876 73877 73878 73879 73880 73881 73882 73883 73884 73885 73886
 73887 73888 73889 73890 73891 73892 73893 73894 73895 73896 73897 73898
 73899 73900 73901 73902 73903 73904 73905 73906 73907 73908 73909 73910
 73911 73912 73913 73914 73915 73916 73917 73918 73919 73920 73921 73922
 73923 73924 73925 73926 73927 73928 73929 73930 73931 73932 73933 73934
 73935 73936 73937 73938] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
C:\Users\vivek\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\feature_selection\_univariate
  f = msb / msw
Selected Feature Indices: [72957 72958 72959 72960 72961 72962 72963 72964 72965 72966 72967 72968
 72969 72970 72971 72972 72974 73263 73264 73265 73266 73267 73268 73269
 73270 73271 73272 73273 73331 73332 73333 73334 73335 73336 73337 73338
 73339 73340 73341 73342 73343 73344 73345 73346 73347 73348 73349 73350
 73351 73352 73353 73354 73355 73356 73357 73358 73359 73360 73361 73601
 73602 73603 73604 73605 73606 73607 73608 73609 73610 73611 73612 73678
 73680 73734 73735 73736 73813 73854 74275 74276 74277 74278 74279 74280
```

```
 74283 74384 74385 74386 74387 74388 74389 74390 74391 74392 74393 74394
 74395 74396 74397 74398]
Selected Feature Names: ['color_hist_feature_57', 'color_hist_feature_58', 'color_hist_feature_59', 'color_hist_feature_60', 'color_hist_feature_61', 'color_hist_feature_62',
```

Principal Component Analysis (PCA) based feature selection

```
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\joblib\_parallel_backends.py:597, in
```

Principal Component Analysis (PCA) is a popular dimensionality reduction technique that can also be used for feature selection. It identifies the most important features by transforming the original features into a new set of orthogonal features called principal components.

```
--> 597     self.results = batch()
```

```
from sklearn.decomposition import PCA

# Apply PCA for feature selection
pca = PCA(n_components=k)  # Choose the desired number of components
selected_features = pca.fit_transform(all_features)

# Get the explained variance ratio of the selected components
explained_variance_ratio = pca.explained_variance_ratio_

# Get the indices of the top selected components
selected_indices = np.argsort(explained_variance_ratio)[::-1][:k]

# Get the selected feature names
selected_feature_names = [feature_names[i] for i in selected_indices]

# Subset the selected features from the original feature matrix
selected_features_matrix = all_features[:, selected_indices]

# Print the indices of the selected features
print("Selected Feature Indices:", selected_indices)

# Print the names of the selected features
print("Selected Feature Names:", selected_feature_names)
```

```
Selected Feature Indices: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
Selected Feature Names: ['hog_feature_0', 'hog_feature_1', 'hog_feature_2', 'hog_feature_3', 'hog_feature_4', 'hog_feature_5', 'hog_feature_6', 'hog_feature_7', 'hog_feature_8
```

```
860         Build a decision tree classifier from the training set (X, y).
```

Variance Thresholding

```
887     """
```

Variance Thresholding is a simple yet effective method for feature selection, especially when dealing with features that have low variance and are assumed to have little discriminatory power.

```
893         check_input=check_input
```

```
from sklearn.feature_selection import VarianceThreshold

# Apply Variance Thresholding for feature selection
threshold = 0.1  # Choose the desired threshold value
selector = VarianceThreshold(threshold=threshold)
selected_features = selector.fit_transform(all_features)

# Get the indices of the selected features
selected_indices = selector.get_support(indices=True)

# Get the selected feature names
selected_feature_names = [feature_names[i] for i in selected_indices]

# Subset the selected features from the original feature matrix
selected_features_matrix = all_features[:, selected_indices]

# Print the indices of the selected features
print("Selected Feature Indices:", selected_indices)

# Print the names of the selected features
print("Selected Feature Names:", selected_feature_names)
```

```
      73704 73705 73706 73707 73708 73709 73710 73711 73712 73713 73714 73715
      73716 73717 73718 73719 73720 73721 73722 73723 73724 73725 73726 73727
      73728 73729 73730 73731 73732 73733 73734 73735 73736 73737 73738 73739
      73740 73741 73742 73743 73744 73745 73746 73747 73748 73749 73750 73751
      73752 73753 73754 73755 73756 73757 73758 73759 73760 73761 73762 73763
      73764 73765 73766 73767 73768 73769 73770 73771 73772 73773 73774 73775
      73776 73777 73778 73779 73780 73781 73782 73783 73784 73785 73786 73787
      73788 73789 73790 73791 73792 73793 73794 73795 73796 73797 73798 73799
      73800 73801 73802 73803 73804 73805 73806 73807 73808 73809 73810 73811
      73812 73813 73814 73815 73816 73817 73818 73819 73820 73821 73822 73823
      73824 73825 73826 73827 73828 73829 73830 73831 73832 73833 73834 73835
      73836 73837 73838 73839 73840 73841 73842 73843 73844 73845 73846 73847
      73848 73849 73850 73851 73852 73853 73854 73855 73856 73857 73858 73859
      73860 73861 73862 73939 73940 73941 73942 73943 73944 73945 73946 73947
      73948 73949 73950 73951 73952 73953 73954 73955 73956 73957 73958 73959
      73960 73961 73962 73963 73964 73965 73966 73967 73968 73969 73970 73971
      73972 73973 73974 73975 73976 73977 73978 73979 73980 73981 73982 73983
      73984 73985 73986 73987 73988 73989 73990 73991 73992 73993 73994 73995
      73996 73997 73998 73999 74000 74001 74002 74003 74004 74005 74006 74007
      74008 74009 74010 74011 74012 74013 74014 74015 74016 74017 74018 74019
      74020 74021 74022 74023 74024 74025 74026 74027 74028 74029 74030 74031
      74032 74033 74034 74035 74036 74037 74038 74039 74040 74041 74042 74043
      74044 74045 74046 74047 74048 74049 74050 74051 74052 74053 74054 74055
      74056 74057 74058 74059 74060 74061 74062 74063 74064 74065 74066 74067
      74068 74069 74070 74071 74072 74073 74074 74075 74076 74077 74078 74079
      74080 74081 74082 74083 74084 74085 74086 74087 74088 74089 74090 74091
      74092 74093 74094 74095 74096 74097 74098 74099 74100 74101 74102 74103
      74104 74105 74106 74107 74108 74109 74110 74111 74112 74113 74114 74115
      74116 74117 74118 74119 74120 74121 74122 74123 74124 74125 74126 74127
      74128 74129 74130 74131 74132 74133 74134 74135 74136 74137 74138 74139
      74140 74141 74142 74143 74144 74145 74146 74147 74148 74149 74150 74151
      74152 74153 74154 74155 74156 74157 74158 74159 74160 74161 74162 74163
      74164 74165 74166 74167 74168 74169 74170 74171 74172 74173 74174 74175
```

```
 74225 74226 74227 74228 74229 74230 74231 74232 74233 74234 74235 74236
 74237 74238 74239 74240 74241 74242 74243 74244 74245 74246 74247 74248
 74249 74250 74251 74252 74253 74254 74255 74256 74257 74258 74259 74260
 74261 74262 74263 74264 74265 74266 74267 74268 74269 74270 74271 74272
 74273 74274 74275 74276 74277 74278 74279 74280 74281 74282 74283 74284
 74285 74286 74287 74288 74289 74290 74291 74292 74293 74294 74295 74296
 74297 74298 74299 74300 74301 74302 74303 74304 74305 74306 74307 74308
 74309 74310 74311 74312 74313 74314 74315 74316 74317 74318 74319 74320
 74321 74322 74323 74324 74325 74326 74327 74328 74329 74330 74331 74332
 74333 74334 74335 74336 74337 74338 74339 74340 74341 74342 74343 74344
 74345 74346 74347 74348 74349 74350 74351 74352 74353 74354 74355 74356
 74357 74358 74359 74360 74361 74362 74363 74364 74365 74366 74367 74368
 74369 74370 74371 74372 74373 74374 74375 74376 74377 74378 74379 74380
 74381 74382 74383 74384 74385 74386 74387 74388 74389 74390 74391 74392
 74393 74394 74395 74396 74397 74398 74399 74400 74401 74402 74403 74404
 74405 74406 74407 74408 74409 74410 74411 74412 74413 74414 74415 74416
 74417 74418 74419 74420 74421 74422 74423 74424 74425 74426 74427 74428
 74429 74430 74431 74432 74433 74434 74435 74436 74437 74438 74439 74440
 74441 74442 74443 74444 74445 74446 74447 74448 74449 74450]
Selected Feature Names: ['color_desc_feature_0', 'color_desc_feature_1', 'color_desc_feature_2', 'color_desc_feature_3', 'color_desc_feature_4', 'color_desc_feature_5', 'col
```

Lasso regularization

L1 regularization, also known as Lasso regularization, is a powerful technique for feature selection that adds an L1 penalty to the model's objective function. This penalty encourages sparsity in the feature weights, resulting in automatic feature selection.

```
from sklearn.linear_model import LassoCV

# Fit Lasso regression model with L1 regularization
lasso = LassoCV(cv=5)  # Choose the desired number of cross-validation folds
lasso.fit(all_features, labels_arr)

# Get the feature importance scores
feature_importances = np.abs(lasso.coef_)

# Get the indices of the selected features
selected_indices = np.argsort(feature_importances)[::-1]

# Get the selected feature names
selected_feature_names = [feature_names[i] for i in selected_indices]

# Subset the selected features from the original feature matrix
selected_features_matrix = all_features[:, selected_indices]

# Print the indices of the selected features
print("Selected Feature Indices:", selected_indices)

# Print the names of the selected features
print("Selected Feature Names:", selected_feature_names)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
c:\Users\vivek\Desktop\sem4\IT258-DS\PlantVillage\corr.ipynb Cell 50 in 5
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#Y100sZmlsZQ%3D%3D?
line=2'>3</a> # Fit Lasso regression model with L1 regularization
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#Y100sZmlsZQ%3D%3D?
line=3'>4</a> lasso = LassoCV(cv=5)  # Choose the desired number of cross-validation folds
----> <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#Y100sZmlsZQ%3D%3D?
line=4'>5</a> lasso.fit(all_features, labels_arr)
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#Y100sZmlsZQ%3D%3D?
line=6'>7</a> # Get the feature importance scores
      <a href='vscode-notebook-cell:/c%3A/Users/vivek/Desktop/sem4/IT258-DS/PlantVillage/corr.ipynb#Y100sZmlsZQ%3D%3D?
line=7'>8</a> feature_importances = np.abs(lasso.coef_)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\sklearn\linear_model\_coordinate_descent.py:1668, in LinearModelCV.fit(self, X, y,
sample_weight)
   1648 # We do a double for loop folded in one, in order to be able to
   1649 # iterate in parallel on l1_ratio and folds
   1650 jobs = (
   1651     delayed(_path_residuals)(
   1652         X,
   (...)
   1666     for train, test in folds
   1667 )
-> 1668 mse_paths = Parallel(
   1669     n_jobs=self.n_jobs,
   1670     verbose=self.verbose,
   1671     prefer="threads",
   1672 )(jobs)
   1673 mse_paths = np.reshape(mse_paths, (n_l1_ratio, len(folds), -1))
   1674 # The mean is computed over folds.

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\sklearn\utils\parallel.py:63, in Parallel.__call__(self, iterable)
     58 config = get_config()
     59 iterable_with_config = (
     60     (_with_config(delayed_func, config), args, kwargs)
     61     for delayed_func, args, kwargs in iterable
     62 )
---> 63 return super().__call__(iterable_with_config)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\joblib\parallel.py:1088, in Parallel.__call__(self, iterable)
   1085 if self.dispatch_one_batch(iterator):
   1086     self._iterating = self._original_iterator is not None
-> 1088 while self.dispatch_one_batch(iterator):
   1089     pass
   1091 if pre_dispatch == "all" or n_jobs == 1:
   1092     # The iterable was consumed all at once by the above for loop.
   1093     # No need to wait for async callbacks to trigger to
   1094     # consumption.

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\joblib\parallel.py:901, in Parallel.dispatch_one_batch(self, iterator)
    899     return False
    900 else:
--> 901     self._dispatch(tasks)
    902     return True

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
```

```
packages\Python310\site-packages\joblib\parallel.py:819, in Parallel._dispatch(self, batch)
```

High time consumption

```
--> 819     job = self._backend.apply_async(batch, callback=cb)
```

What could be done with the selected features?

Use the selected features as input variables to train machine learning models and improve their performance.

Evaluate the models using various metrics to assess the impact of feature selection on performance.

Gain insights and interpretability by analyzing the importance and contribution of each selected feature.

```
    207     """Schedule a func to be run"""
```

Summary

1. Using Hog feature extraction we did the feature extraction part.

the correlation analysis and covariance analysis

1. On these features we tried to do the correlation analysis and covariance analysis, but since matrix is of huge size, got the memory error for it.
2. So we went on to do the PCA analysis on the extracted feature set by hog.
3. On these PC's we did the correlation analysis and covariance analysis.

Dimensionality reduction

1. As per as the dimensionality reduction we have used the PCA, whose results we used for the correlation analysis and covariance analysis.
2. since we have the image data set we also tried to explore other reductions methods that do not use linear combination of the features.
3. we used t-SNE, which works based on nonlinear combination of the features.
4. Then we moved on to the Multidimensional Scaling (MDS) dimensionality reduction method.

Feature Selection

1. We have mostly done all of works on the features extracted usign the HOG.
2. so before moving on to the Feature selection we extracted features using other methods like Color Histogram Color Descriptors Color Moments Color Space Conversion
3. We combined all the features from these and then explored the various selection methods on them.
4. We tried to implement various selection methods which were taught in class, like SelectKBest (Filter method) Recursive Feature Elimination (RFE) (Wrapper method) Tree-based Feature Importance (Wrapper method) Variance Thresholding (Filter method) L1 Regularization (Lasso) (Wrapper method) Principal Component Analysis (PCA) based feature selection

```
sample_weight, train, test, fit_intercept, path, path_params, alphas, l1_ratio, X_order, dtype)
```

Results

```
    1202 del X_train, y_train
```

Results of each task done is associated with the code as a mardown cell after the code cell in the notebook.

```
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
```

```
packages\Python310\site-packages\sklearn\linear_model\_coordinate_descent.py:332, in lasso_path(X, y, eps, n_alphas,
alphas, precompute, Xy, copy_X, coef_init, verbose, return_n_iter, positive, **params)
    176 def lasso_path(
    177     X,
    178     y,
   (...)
    190     **params,
    191 ):
    192     """Compute Lasso path with coordinate descent.
    193
    194     The Lasso optimization function varies for mono and multi-outputs.
   (...)
    330      [0.2159048  0.4425765  0.23668876]]
    331     """
--> 332     return enet_path(
    333         X,
    334         y,
    335         l1_ratio=1.0,
    336         eps=eps,
    337         n_alphas=n_alphas,
    338         alphas=alphas,
    339         precompute=precompute,
    340         Xy=Xy,
    341         copy_X=copy_X,
    342         coef_init=coef_init,
    343         verbose=verbose,
    344         positive=positive,
    345         return_n_iter=return_n_iter,
    346         **params,
    347     )

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\sklearn\linear_model\_coordinate_descent.py:631, in enet_path(X, y, l1_ratio, eps,
n_alphas, alphas, precompute, Xy, copy_X, coef_init, verbose, return_n_iter, positive, check_input, **params)
    617     model = cd_fast.enet_coordinate_descent_gram(
    618         coef_,
    619         l1_reg,
   (...)
    628         positive,
    629     )
    630 elif precompute is False:
--> 631     model = cd_fast.enet_coordinate_descent(
    632         coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
    633     )
```