```
import pandas as pd
import cv2
import os
import numpy as np
import warnings
from skimage.color import rgb2gray
from skimage import io, exposure, filters
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, SpectralClustering,
BisectingKMeans
from sklearn.cluster import DBSCAN, AgglomerativeClustering
from sklearn.metrics import fowlkes_mallows_score, silhouette_score

warnings.filterwarnings('ignore')
```

1. Feature Extraction

```
def calculate_gradient_angle(dx, dy):
    """Calculate the angles between horizontal and vertical
operators."""
    return np.mod(np.arctan2(dy, dx), np.pi)

data_directory = "/content/drive/MyDrive/data_mining/processed"

class_labels =
['Basenji','Airedale','Brittany_spaniel','Bouvier_des_flandres']

dataframe = pd.DataFrame(columns = list(range(0,36))+['class'])
class_folders = os.listdir(data_directory)
for folder in class_folders:
  class_path = os.path.join(data_directory, folder)
  for i, label in enumerate(class_labels):
    if label.lower() == folder.split("-")[-1].lower():
      class_num = i
  for filename in os.listdir(class_path):
    img = io.imread(os.path.join(class_path,filename))
    gray_image = rgb2gray(img)
    angle_sobel =
calculate_gradient_angle(filters.sobel_h(gray_image),
                     filters.sobel_v(gray_image))
    hist,bins = exposure.histogram(angle_sobel,nbins=36)
    dataframe.loc[len(dataframe)] = list(hist)+[class_num]


scaler = StandardScaler()
scaler.fit(dataframe[dataframe.columns[:-1]])

data = dataframe[dataframe.columns[:-1]]
```

```
original_class_labels = np.array(dataframe[dataframe.columns[-1]])

scaled_data = scaler.transform(data)
```

2. PCA Dimension Reduction

```
pca = PCA(n_components=2)

transformed_data = pca.fit_transform(scaled_data)
```

3. Clustering

```
# K-means clustering with init='random'
kmeans_random = KMeans(n_clusters=4, init='random', random_state=42)
kmeans_random.fit(transformed_data)
kmeans_random_labels = kmeans_random.labels_

# K-means clustering with init='k-means++'
kmeans_kmeans_pp = KMeans(n_clusters=4, init='k-means++',
random_state=42)
kmeans_kmeans_pp.fit(transformed_data)
kmeans_kmeans_pp_labels = kmeans_kmeans_pp.labels_

# Bisecting K-means clustering with init='random'
bisecting_kmeans_random = BisectingKMeans(n_clusters=4, init='random',
random_state=42)
bisecting_kmeans_random.fit(transformed_data)
bisecting_kmeans_random_labels = bisecting_kmeans_random.labels_

# Spectral clustering with default parameters
spectral_clustering = SpectralClustering(n_clusters=4,
random_state=42)
spectral_clustering.fit(transformed_data)
spectral_clustering_labels = spectral_clustering.labels_

# DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=2)
dbscan.fit(data)
dbscan_labels = dbscan.labels_

# Agglomerative clustering with different linkage methods
agglomerative_single = AgglomerativeClustering(n_clusters=4,
linkage='single')
agglomerative_single.fit(data)
agglomerative_single_labels = agglomerative_single.labels_
```

```
agglomerative_complete = AgglomerativeClustering(n_clusters=4,
linkage='complete')
agglomerative_complete.fit(data)
agglomerative_complete_labels = agglomerative_complete.labels_

agglomerative_average = AgglomerativeClustering(n_clusters=4,
linkage='average')
agglomerative_average.fit(data)
agglomerative_average_labels = agglomerative_average.labels_

agglomerative_ward = AgglomerativeClustering(n_clusters=4,
linkage='ward')
agglomerative_ward.fit(data)
agglomerative_ward_labels = agglomerative_ward.labels_
```

4. Calculate Scores

```
fowlkes_mallows_scores = {
    'K-means (Random)': fowlkes_mallows_score(original_class_labels,
kmeans_random_labels),
    'K-means (k-means++)':
fowlkes_mallows_score(original_class_labels, kmeans_kmeans_pp_labels),
    'Bisecting K-means': fowlkes_mallows_score(original_class_labels,
bisecting_kmeans_random_labels),
    'Spectral Clustering':
fowlkes_mallows_score(original_class_labels,
spectral_clustering_labels),
    'DBSCAN': fowlkes_mallows_score(original_class_labels,
dbscan_labels),
    'Agglomerative (Single link)':
fowlkes_mallows_score(original_class_labels,
agglomerative_single_labels),
    'Agglomerative (Complete link)':
fowlkes_mallows_score(original_class_labels,
agglomerative_complete_labels),
    'Agglomerative (Group Average)':
fowlkes_mallows_score(original_class_labels,
agglomerative_average_labels),
    'Agglomerative (Ward)':
fowlkes_mallows_score(original_class_labels,
agglomerative_ward_labels)
}

silhouette_scores = {
    'K-means (Random)': silhouette_score(transformed_data,
kmeans_random_labels),
    'K-means (k-means++)': silhouette_score(transformed_data,
```

```
kmeans_kmeans_pp_labels),
    'Bisecting K-means': silhouette_score(transformed_data,
bisecting_kmeans_random_labels),
    'Spectral Clustering': silhouette_score(transformed_data,
spectral_clustering_labels),
    'DBSCAN': silhouette_score(transformed_data, dbscan_labels),
    'Agglomerative (Single link)': silhouette_score(transformed_data,
agglomerative_single_labels),
    'Agglomerative (Complete link)':
silhouette_score(transformed_data, agglomerative_complete_labels),
    'Agglomerative (Group Average)':
silhouette_score(transformed_data, agglomerative_average_labels),
    'Agglomerative (Ward)': silhouette_score(transformed_data,
agglomerative_ward_labels)
}
```

Rank Scores

```
ranked_methods_fm = sorted(fowlkes_mallows_scores.items(), key=lambda
x: x[1], reverse=True)
print("Ranking based on Fowlkes-Mallows index:")
for method, score in ranked_methods_fm:
    print(f"{method}: {score}")
```

```
Ranking based on Fowlkes-Mallows index:
DBSCAN: 0.5002698125334627
Agglomerative (Single link): 0.4984317900355655
Agglomerative (Complete link): 0.433116968386077
Agglomerative (Group Average): 0.4323712267130742
Agglomerative (Ward): 0.3494824532576536
Spectral Clustering: 0.3448680331177179
K-means (k-means++): 0.29276677234334564
K-means (Random): 0.29264973176469217
Bisecting K-means: 0.283495760257754
```

```
ranked_methods_silhouette = sorted(silhouette_scores.items(),
key=lambda x: x[1], reverse=True)
print("\nRanking based on Silhouette Coefficient:")
for method, score in ranked_methods_silhouette:
    print(f"{method}: {score}")
```

```
Ranking based on Silhouette Coefficient:
DBSCAN: 0.5191030010671098
K-means (Random): 0.44865130193551783
K-means (k-means++): 0.4473471041006935
Agglomerative (Complete link): 0.442676295573699
Agglomerative (Group Average): 0.4149510208189473
```

```
Spectral Clustering: 0.3890738513559013
Bisecting K-means: 0.3879733401290354
Agglomerative (Ward): 0.37429613326368116
Agglomerative (Single link): 0.16868010189535632
```