Assignment-4

December 13, 2023

[1]:
```
import os
import glob
import matplotlib.pyplot as plt
import cv2
import numpy as np
import pandas as pd
from sklearn import preprocessing
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
dog_images = glob.glob('/Users/Assignment/Images/*/*')
breeds = glob.glob('/Users/Assignment/Images/*')
annotations = glob.glob('/Users/Assignment/Annotation/*/*')
cropped = "./Cropped_1/"
```

1 Cropping and Resize Images in Your 4-class Images Dataset

[3]:
```
four_classes=['papillon','bluetick','Dandie_Dinmont','Pembroke']
```

[4]:
```
def get_bounding_boxes(annot):
    xml = annot
    tree = ET.parse(xml)
    root = tree.getroot()
    objects = root.findall('object')
    bbox = []
    for o in objects:
        bndbox = o.find('bndbox')
        xmin = int(bndbox.find('xmin').text)
        ymin = int(bndbox.find('ymin').text)
        xmax = int(bndbox.find('xmax').text)
        ymax = int(bndbox.find('ymax').text)
        bbox.append((xmin,ymin,xmax,ymax))
    return bbox
```

[5]:
```
def get_image(annot):
    img_path = '/Users/Assignment/Images/'
    file = annot.split('/')
```

```
    img_filename = img_path + file[-2]+'/'+file[-1]+'.jpg'
    return img_filename
```

[6]:
```
import xml.etree.ElementTree as ET
from keras_preprocessing import image
from PIL import Image
from pathlib import Path
plt.figure(figsize=(10,6))
for i in range(len(dog_images)):
    if str(dog_images[i]).split('/')[-2].split('-')[1] in four_classes:
        bbox = get_bounding_boxes(annotations[i])
        dog = get_image(annotations[i])
        im = Image.open(dog)
        for j in range(len(bbox)):
            im2 = im.crop(bbox[j])
            im2 = im2.resize((224,224), Image.ANTIALIAS)
            new_path = dog.replace('/Users/Assignment/Images/','/Users/
↪Assignment/Cropped_1/')
            new_path = new_path.replace('.jpg','-' + str(j) + '.jpg')
            im2=im2.convert('RGB')
            head, tail = os.path.split(new_path)
            Path(head).mkdir(parents=True, exist_ok=True)
            im2.save(new_path)

# Refernce ----->https://www.kaggle.com/code/espriella/
↪stanford-dogs-transfer-crop-stack/notebook
```

<Figure size 1000x600 with 0 Axes>

[7]:
```
images=glob.glob('/Users/Assignment/Cropped_1/*/*')
```

[8]:
```
# Normalize Dataset
```

[9]:
```
import albumentations as A
from albumentations.pytorch import ToTensorV2

transforms = A.Compose([A.Resize(height = 128, width = 128),
                        A.Normalize(),
                        ToTensorV2()])
```

[10]:
```
hist=[]
classes=[]
for i in images:
    image = plt.imread(i)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    normalized_image = gray_image.astype(float) / 255.0
    n = transforms(image = image)['image']
    hist.append(n)
```

```
    classes.append(str(i).split('/')[4].split('-')[1])
```

[11]:
```
import torch
from torch.utils.data import Dataset, DataLoader
data_loader = DataLoader(hist,
                         batch_size  = 32,
                         shuffle     = False,
                         num_workers = 2)
```

[12]:
```
import timm
import torch.nn as nn
```

[13]:
```
model    = timm.create_model(model_name = 'resnet18', pretrained = True)
model.fc = nn.Linear(512, 2)
```

[14]:
```
def get_features(name):
    def hook(model, input, output):
        features[name] = output.detach()
    return hook
```

[15]:
```
model.global_pool.register_forward_hook(get_features('feats'))
```

[15]:
```
<torch.utils.hooks.RemovableHandle at 0x16038dd90>
```

[16]:
```
##### FEATURE EXTRACTION LOOP

# placeholders
PREDS = []
FEATS = []

# placeholder for batch features
features = {}

# loop through batches
for idx, inputs in enumerate(data_loader):



    # forward pass [with feature extraction]
    preds = model(inputs)

    # add feats and preds to lists
    PREDS.append(preds.detach().numpy())
    FEATS.append(features['feats'].numpy())

    # early stop
    if idx == 9:
```

```
        break
```

[17]:
```
PREDS = np.concatenate(PREDS)
FEATS = np.concatenate(FEATS)

print('- preds shape:', PREDS.shape)
print('- feats shape:', FEATS.shape)
# Reference---https://kozodoi.me/blog/20210527/extracting-features
```

```
- preds shape: (320, 2)
- feats shape: (320, 512)
```

[18]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
import numpy as np
pca = PCA(n_components=2)
```

[19]:
```
listt=[]
clas=[]
for i in images:
    img = plt.imread(i)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    hist1 = cv2.calcHist([gray], [0], None, [256], [0,256])

    hist=hist1.flatten()
    normalized_data = (hist - np.min(hist)) / (np.max(hist) - np.min(hist))

    listt.append(normalized_data)
    clas.append(str(i).split('/')[-2].split('-')[1])
transform=pca.fit_transform(listt)
transform.shape
```

[19]:
```
(786, 2)
```

[20]:
```
data_set=pd.DataFrame(transform)
```

[21]:
```
data_set.head()
```

[21]:

	0	1
0	-1.739353	-1.265062
1	-0.964784	0.884294
2	0.842698	-2.213536
3	-2.330110	1.106226
4	-3.633325	0.084845

[22]:
```
from sklearn.cluster import KMeans
kmean = KMeans(n_clusters=4,n_init=20,init='random')
kmean.fit(data_set)
```

[22]: KMeans(init='random', n_clusters=4, n_init=20)

[23]:
```
kmean_lab=kmean.labels_
```

[24]:
```
kmeanp = KMeans(init="k-means++", n_clusters=4, n_init=10)
kmeanp.fit(data_set)
```

[24]: KMeans(n_clusters=4, n_init=10)

[25]:
```
set(kmeanp.labels_)
kp=kmeanp.labels_
```

[26]:
```
from sklearn.cluster import BisectingKMeans
Bikmean = BisectingKMeans(init="random", n_clusters=4, n_init=10)
Bikmean.fit(data_set)
```

[26]: BisectingKMeans(n_clusters=4, n_init=10)

[27]:
```
Bi=Bikmean.labels_
```

[28]:
```
from sklearn.cluster import SpectralClustering
spec = SpectralClustering(n_clusters=4, n_init=10)
spec.fit(data_set)
```

[28]: SpectralClustering(n_clusters=4)

[29]:
```
sp=spec.labels_
```

[30]:
```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.5, min_samples=22).fit(data_set)
m=db.labels_
print(len(set(db.labels_)))
```

4

2 min_samples=22, ep=0.5

[31]:
```
from sklearn.cluster import AgglomerativeClustering
clus_ward = AgglomerativeClustering(n_clusters=4,linkage='ward').fit(data_set)
labb=clus_ward.labels_
```

[32]:
```
set(labb)
```

[32]: {0, 1, 2, 3}

[33]:
```
clus_single = AgglomerativeClustering(n_clusters=4,linkage='single').
 ↪fit(data_set)
labb_single=clus_single.labels_
```

```
print(set(labb_single))
```

{0, 1, 2, 3}

[34]:
```
clus_complete = AgglomerativeClustering(n_clusters=4,linkage='complete').
  ↪fit(data_set)
labb_complete=clus_complete.labels_
print(set(labb_complete))
```

{0, 1, 2, 3}

[35]:
```
clus_average = AgglomerativeClustering(n_clusters=4,linkage='average').
  ↪fit(data_set)
labb_average=clus_average.labels_
print(set(labb_average))
```

{0, 1, 2, 3}

[36]:
```
z,s=[],[]
```

[37]:
```
from sklearn.metrics.cluster import fowlkes_mallows_score
from sklearn.metrics import silhouette_score
z_kmean=fowlkes_mallows_score(clas,kmean_lab )
si_kmean=silhouette_score(data_set,kmean_lab )
print(f"kmeans fowlkes_mallows_score{z_kmean} and silhouette_score{si_kmean}")
z.append(z_kmean)
s.append(si_kmean)
```

kmeans fowlkes_mallows_score0.2601336093068398 and
silhouette_score0.38604470366077026

[38]:
```
z_kmeanp=fowlkes_mallows_score(clas,kp )
si_kmeanp=silhouette_score(data_set,kp )
print(f"kmeans++ fowlkes_mallows_score{z_kmeanp} and␣
  ↪silhouette_score{si_kmeanp}")
z.append(z_kmeanp)
s.append(si_kmean)
```

kmeans++ fowlkes_mallows_score0.2601336093068398 and
silhouette_score0.38604470366077026

[39]:
```
z_bi=fowlkes_mallows_score(clas,Bi )
si_bi=silhouette_score(data_set,Bi )
print(f"Bisecting kmeans fowlkes_mallows_score{z_bi} and␣
  ↪silhouette_score{si_bi}")
z.append(z_bi)
s.append(si_kmean)
```

Bisecting kmeans fowlkes_mallows_score0.26415645395055587 and
silhouette_score0.32355639772562117

[40]:
```
z_sp=fowlkes_mallows_score(clas,sp )
si_sp=silhouette_score(data_set,sp )
print(f"spectral fowlkes_mallows_score{z_sp} and silhouette_score{si_sp}")
z.append(z_sp)
s.append(si_sp)
```

```
spectral fowlkes_mallows_score0.26814316149844447 and
silhouette_score0.36885670800805337
```

[41]:
```
z_db=fowlkes_mallows_score(clas, m)
si_db=silhouette_score(data_set,m )
print(f"DBSCAN fowlkes_mallows_score{z_db} and silhouette_score{si_db}")
z.append(z_db)
s.append(si_db)
```

```
DBSCAN fowlkes_mallows_score0.3448185065139945 and
silhouette_score-0.0219352655727482
```

[42]:
```
z_w=fowlkes_mallows_score(clas, labb)
si_w=silhouette_score(data_set,labb )
print(f"Agglomerative clustering ward fowlkes_mallows_score{z_w} and␣
↪silhouette_score{si_w}")
z.append(z_w)
s.append(si_w)
```

```
Agglomerative clustering ward fowlkes_mallows_score0.27848809911961486 and
silhouette_score0.30175361784000293
```

[43]:
```
z_s=fowlkes_mallows_score(clas, labb_single)
si_s=silhouette_score(data_set,labb_single )
print(f"Agglomerative clustering single fowlkes_mallows_score{z_s} and␣
↪silhouette_score{si_s}")
z.append(z_s)
s.append(si_s)
```

```
Agglomerative clustering single fowlkes_mallows_score0.4982613464381047 and
silhouette_score-0.0692887304304368
```

[44]:
```
z_c=fowlkes_mallows_score(clas, labb_complete)
si_c=silhouette_score(data_set,labb_complete )
print(f"Agglomerative clustering complete fowlkes_mallows_score{z_c} and␣
↪silhouette_score{si_c}")
z.append(z_c)
s.append(si_c)
```

```
Agglomerative clustering complete fowlkes_mallows_score0.2669003337980466 and
silhouette_score0.31283073590136246
```

[45]:
```
z_a=fowlkes_mallows_score(clas, labb_average)
si_a=silhouette_score(data_set,labb_average )
print(f"Agglomerative clustering average fowlkes_mallows_score{z_a} and␣
↪silhouette_score{si_a}")
z.append(z_a)
s.append(si_a)
```

```
Agglomerative clustering average fowlkes_mallows_score0.2864314198602307 and
silhouette_score0.31049380001486576
```

[46]:
```
zip_data=␣
↪zip(["z_kmean","z_kmeanp","z_bi","z_sp","z_db","z_w","z_s","z_c","z_a"],z)
sorted_data = sorted(zip_data, key=lambda x: x[1])
```

[47]:
```
sorted_data
```

[47]:
```
[('z_kmean', 0.2601336093068398),
 ('z_kmeanp', 0.2601336093068398),
 ('z_bi', 0.26415645395055587),
 ('z_c', 0.2669003337980466),
 ('z_sp', 0.26814316149844447),
 ('z_w', 0.27848809911961486),
 ('z_a', 0.2864314198602307),
 ('z_db', 0.3448185065139945),
 ('z_s', 0.4982613464381047)]
```

[48]:
```
# Based on the fowlkes_mallows_score, spectral clustering performs better as it␣
↪has high value
# Based on the fowlkes_mallows_score, kmeans performs worst as it has low value
```

[49]:
```
zip_data=␣
↪zip(["si_kmean","si_kmeanp","si_bi","si_sp","si_db","si_w","si_s","si_c","si_a"],s)
sorted_data = sorted(zip_data, key=lambda x: x[1])
```

[50]:
```
sorted_data
```

[50]:
```
[('si_s', -0.0692887304304368),
 ('si_db', -0.0219352655727482),
 ('si_w', 0.30175361784000293),
 ('si_a', 0.31049380001486576),
 ('si_c', 0.31283073590136246),
 ('si_sp', 0.36885670800805337),
 ('si_kmean', 0.38604470366077026),
 ('si_kmeanp', 0.38604470366077026),
 ('si_bi', 0.38604470366077026)]
```

[51]:
```
# Based on the silhouette_score, Agglomerative single ward performs worst as it␣
↪has low value
# Based on the silhouette_score, bisecting kmeans performs best as it has high␣
↪value
```

3 Reference

4 https://kozodoi.me/blog/20210527/extracting-features

5 https://scikit-learn.org/stable/modules/clustering.html

[ ]:
```
```