Project: NAÏVE BEES: PREDICT SPECIES FROM IMAGES

BİLGİSAYARLI GÖRME İÇİN DERİN ÖĞRENME UYGULAMALARI

İstanbul Üniversitesi

Rayees Ahmad Lone

## **1. Import Python libraries**

*# used to change filepaths*  
import os  
  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
from IPython.display import display  
%matplotlib inline  
  
import pandas as pd  
import numpy as np  
  
*# import Image from PIL*  
from PIL import Image  
  
from skimage.feature import hog  
from skimage.color import rgb2grey  
  
from sklearn.preprocessing import StandardScaler  
from sklearn.decomposition import PCA  
  
*# import train\_test\_split from sklearn's model selection module*  
from sklearn.model\_selection import train\_test\_split  
  
*# import SVC from sklearn's svm module*  
from sklearn.svm import SVC  
  
*# import accuracy\_score from sklearn's metrics module*  
from sklearn.metrics import roc\_curve, auc, accuracy\_score

**2. Display image of each bee type**

*# load the labels using pandas*  
labels = pd.read\_csv("datasets/labels.csv", index\_col=0)  
  
*# show the first five rows of the dataframe using head*  
display(labels.head)  
  
**def** get\_image(row\_id, root="datasets/"):  
 *"""*  
 *Converts an image number into the file path where the image is located,*   
 *opens the image, and returns the image as a numpy array.*  
 *"""*  
 filename = "{}.jpg".format(row\_id)  
 file\_path = os.path.join(root, filename)  
 img = Image.open(file\_path)  
 **return** np.array(img)  
  
*# subset the dataframe to just Apis (genus is 0.0) get the value of the sixth item in the index*  
apis\_row = labels[labels.genus == 0.0].index[5]  
  
*# show the corresponding image of an Apis*  
plt.imshow(get\_image(apis\_row))  
plt.show()  
  
*# subset the dataframe to just Bombus (genus is 1.0) get the value of the sixth item in the index*  
bombus\_row = labels[labels.genus == 1.0].index[5]  
  
*# show the corresponding image of a Bombus*  
plt.imshow(get\_image(bombus\_row))  
plt.show()

**3. Image manipulation with rgb2grey**

*# load a bombus image using our get\_image function and bombus\_row from the previous cell*  
bombus = get\_image(bombus\_row)  
  
*# print the shape of the bombus image*  
print('Color bombus image has shape: ', bombus.shape)  
  
*# convert the bombus image to greyscale*  
grey\_bombus = rgb2grey(bombus)  
  
*# show the greyscale image*  
plt.imshow(grey\_bombus, cmap=mpl.cm.gray)  
  
*# greyscale bombus image only has one channel*  
print('Greyscale bombus image has shape: ', grey\_bombus.shape)

**4. Histogram of oriented gradients**

*# run HOG using our greyscale bombus image*  
hog\_features, hog\_image = hog(grey\_bombus,  
 visualize=True,  
 block\_norm='L2-Hys',  
 pixels\_per\_cell=(16, 16))  
  
*# show our hog\_image with a grey colormap*  
plt.imshow(hog\_image, cmap=mpl.cm.gray)

**5. Create image features and flatten into a single row**

**def** create\_features(img):  
 *# flatten three channel color image*  
 color\_features = np.ndarray.flatten(img)  
 *# convert image to greyscale*  
 grey\_image = rgb2grey(img)  
 *# get HOG features from greyscale image*  
 hog\_features = hog(grey\_image, block\_norm='L2-Hys', pixels\_per\_cell=(16, 16))  
 *# combine color and hog features into a single array*  
 flat\_features = np.hstack((color\_features,hog\_features))  
 **return** flat\_features  
  
bombus\_features = create\_features(bombus)  
  
*# print shape of bombus\_features*  
print(bombus\_features.shape)

**6. Loop over images to preprocess**

**def** create\_feature\_matrix(label\_dataframe):  
 features\_list = []  
   
 **for** img\_id in label\_dataframe.index:  
 *# load image*  
 img = get\_image(img\_id)  
 *# get features for image*  
 image\_features = create\_features(img)  
 features\_list.append(image\_features)  
   
 *# convert list of arrays into a matrix*  
 feature\_matrix = np.array(features\_list)  
 **return** feature\_matrix  
  
*# run create\_feature\_matrix on our dataframe of images*  
feature\_matrix = create\_feature\_matrix(labels)

**7. Scale feature matrix + PCA**

*# get shape of feature matrix*  
print('Feature matrix shape is: ', feature\_matrix.shape)  
  
*# define standard scaler*  
ss = StandardScaler()  
*# run this on our feature matrix*  
bees\_stand = ss.fit\_transform(feature\_matrix)  
  
pca = PCA(n\_components=500)  
*# use fit\_transform to run PCA on our standardized matrix*  
bees\_pca = pca.fit\_transform(bees\_stand)  
*# look at new shape*  
print('PCA matrix shape is: ', bees\_pca.shape)

Feature matrix shape is: (500, 31296)  
PCA matrix shape is: (500, 500)

**8. Split into train and test sets**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(bees\_pca,  
 labels.genus.values,  
 test\_size=.3,  
 random\_state=1234123)  
  
*# look at the distrubution of labels in the train set*  
pd.Series(y\_train).value\_counts()

**9. Train model**

*# define support vector classifier*  
svm = SVC (kernel='linear',probability=True,random\_state=42)  
  
*# fit model*  
svm.fit(X\_train,y\_train)

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,  
 decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='linear',  
 max\_iter=-1, probability=True, random\_state=42, shrinking=True,  
 tol=0.001, verbose=False)

**10. Score model**

*# generate predictions*  
y\_pred = svm.predict(X\_test)  
  
*# calculate accuracy*  
accuracy = accuracy\_score(y\_pred, y\_test)  
print('Model accuracy is: ', accuracy)

Model accuracy is: 0.68

**11. ROC curve + AUC**

*# predict probabilities for X\_test using predict\_proba*  
probabilities = svm.predict\_proba(X\_test)  
  
*# select the probabilities for label 1.0*  
y\_proba = probabilities[:, 1]  
  
*# calculate false positive rate and true positive rate at different thresholds*  
false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_proba, pos\_label=1)  
  
*# calculate AUC*  
roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)  
  
plt.title('Receiver Operating Characteristic')  
*# plot the false positive rate on the x axis and the true positive rate on the y axis*  
roc\_plot = plt.plot(false\_positive\_rate,  
 true\_positive\_rate,  
 label='AUC = {:0.2f}'.format(roc\_auc))  
  
plt.legend(loc=0)  
plt.plot([0,1], [0,1], ls='--')  
plt.ylabel('True Positive Rate')  
plt.xlabel('False Positive Rate');