Importing all required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import os
import xml.etree.ElementTree as ET
from PIL import Image
from pathlib import Path
import random
import warnings
warnings.filterwarnings("ignore")

images_dir = r'C:\Users\V Varunkumar\Desktop\programming assignment 2\
images'
annotation_dir = r'C:\Users\V Varunkumar\Desktop\programming
assignment 2\annotation''

# Function to List directories inside a given path
def list_directories (path):
    return [d for d in os.listdir(path) if
os.path.isdir(os.path.join(path, d))]

images_subdirs = list_directories (images_dir)
annotations_subdirs = list_directories (annotation_dir)
```

1. Use images from ALL FOUR classes.

```
print("Directories in Images folder:", images_subdirs)
print("\nDirectories in Annotations folder:", annotations_subdirs)

Directories in Images folder: ['n02091831-Saluki', 'n02093859-
Kerry_blue_terrier', 'n02108551-Tibetan_mastiff', 'n02111277-
Newfoundland']

Directories in Annotations folder: ['n02091831-Saluki', 'n02093859-
Kerry_blue_terrier', 'n02108551-Tibetan_mastiff', 'n02111277-
Newfoundland']

import os
from skimage import filters, exposure
from skimage.color import rgb2gray
from skimage.io import imread
import numpy as np

def angle(dx, dy):
    """Calculate the angles between horizontal and vertical
operators."""
```

```
    return np.mod(np.arctan2(dy, dx), np.pi)

def compute_edge_histogram(image, grid_size=(4, 4), nbins=36):
    """Compute the edge histogram for an image."""
    # Convert to grayscale
    gray_img = rgb2gray(image)

    # Get the Sobel gradients
    dx = filters.sobel_h(gray_img)
    dy = filters.sobel_v(gray_img)

    # Compute the gradient angle
    angle_sobel = angle(dx, dy)

    # Divide the image into sub-regions and compute histograms for
each region
    height, width = gray_img.shape
    h_step = height // grid_size[0]
    w_step = width // grid_size[1]

    edge_hist = []

    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            # Extract sub-region
            sub_region = angle_sobel[i*h_step:(i+1)*h_step, j*w_step:
(j+1)*w_step]

            # Compute histogram for the sub-region
            hist, _ = exposure.histogram(sub_region, nbins=nbins)

            # Normalize histogram
            hist = hist / hist.sum()

            # Append to the edge histogram
            edge_hist.extend(hist)

    return edge_hist
```

1. Convert the images to edge histograms. (Assignment 1 - These will be the vector representations of the images). This will be your dataset for Part 3.

```
# Main code to process all images in the directory
histograms = []
labels = []

for index, name in enumerate(os.listdir(images_dir)):
    label = []
    for image in os.listdir(os.path.join(images_dir, name)):
        img = imread(os.path.join(images_dir, name, image.strip()))
```

```
        # Compute the edge histogram
        edge_hist = compute_edge_histogram(img, grid_size=(4, 4),
nbins=36)

        # Append histogram and label
        histograms.append(edge_hist)
        labels.append(index)

# Convert histograms and labels to arrays for further processing
histograms = np.array(histograms)
labels = np.array(labels)


histograms.shape

(726, 576)
```

1. Split the dataset into a training set and a test set: For each class, perform a training/test split of 80/20

```
# Convert the list of histograms and labels to numpy arrays for
further processing
X = histograms  # Features (edge histograms)
y = labels      # Labels (corresponding class indices)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

1. Perform standardization on the training dataset.

```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform the training data
X_train_standardized = scaler.fit_transform(X_train)



# Output the results to verify
print("Standardization complete.")
print(f"Mean of standardized training data:
{np.mean(X_train_standardized, axis=0)}")
print(f"Variance of standardized training data:
{np.var(X_train_standardized, axis=0)}")
```

```
Standardization complete.
Mean of standardized training data: [-7.01067556e-16  2.54987602e-15 -
5.83830159e-16 -4.06140638e-15
  5.95500660e-16  3.79562282e-15  3.46657569e-16  1.78248221e-15
  6.12919677e-16  4.29541460e-16  5.98754762e-16  4.53621814e-15
  4.65527999e-16  2.83202580e-15 -8.06251617e-16 -3.11340991e-16
  1.56943424e-15  8.61667060e-16  3.46753277e-16  1.17051962e-16
 -4.18056394e-16 -9.28950403e-16  7.04991621e-16  4.26191649e-15
  8.89709762e-16  2.97597196e-15  1.57833516e-15  2.27059750e-15
 -1.20239068e-15  1.89541869e-15  1.02944473e-15 -2.89653359e-15
 -7.45380769e-16 -1.43659031e-16  8.99854903e-16  3.48188911e-16
 -1.66533454e-17  1.63164501e-15 -1.76903511e-15  9.50006357e-16
  1.48296126e-15  1.56196894e-15 -3.43116340e-15 -1.52777695e-15
 -5.51005601e-15  2.33395678e-15 -3.09139687e-16  1.63495893e-15
  3.86453321e-15 -2.49082364e-15 -1.88278512e-15 -2.50271547e-15
 -3.68664629e-16  2.22934698e-15  1.69605709e-15 -6.79317713e-16
  3.27514596e-15 -4.20066281e-16  1.20344348e-15 -3.84366868e-16
 -2.41913769e-15 -4.01556183e-15 -5.01705956e-16  2.28935644e-15
  2.05984655e-15  3.80155677e-16 -1.67720244e-15  1.56962566e-17
 -2.05180700e-15  3.43728877e-15  1.96107498e-16  8.82914431e-17
 -8.28743204e-16  1.23148618e-15 -3.04277676e-15 -8.57551578e-17
  2.20905669e-15 -2.48364547e-15  8.96026548e-16  2.12588567e-15
  1.63279352e-15  5.54154423e-16 -2.85241179e-15  3.40111081e-15
 -6.55701547e-16  4.69806186e-15 -3.25314488e-16  4.85579010e-16
  2.86564355e-15  1.57929225e-15 -3.37469516e-16  1.64714985e-15
  3.16509271e-16  8.55637400e-17  3.15839309e-18 -4.04972093e-15
 -1.49238859e-15  2.64888686e-15  1.87551124e-15  3.89745707e-15
 -3.82261272e-16  4.07145582e-16  6.73407690e-16  1.35906612e-16
 -3.29822376e-15 -9.54408966e-16  1.62015994e-15 -2.86674270e-16
 -3.55462786e-16  1.58953310e-15 -9.46752255e-16  1.36643570e-15
 -1.30020515e-16  2.74155997e-15 -2.25765288e-16  2.78742546e-15
  1.33389468e-15  1.85298376e-15  3.92157571e-15 -6.96760657e-17
 -1.07193947e-17 -1.25378635e-15 -1.17798491e-15  1.39313848e-15
 -1.77635684e-16  1.29216561e-15  6.07452307e-17 -2.54968460e-15
  3.31535565e-16 -3.16528413e-15  3.87046717e-16  5.13286731e-16
 -2.35252431e-16 -6.92405903e-16  2.80711757e-15  2.99310385e-15
  1.37796862e-15  3.03186595e-15 -2.44823319e-15  6.84318502e-17
 -1.49229288e-15 -1.46013470e-15  1.18602446e-15 -6.03157371e-16
  2.13909350e-16  1.48693318e-15  6.85849844e-16  1.18296177e-16
  1.24383262e-15 -7.08245723e-16 -6.59099212e-16 -2.15344983e-16
  2.38697950e-16  1.23177330e-15 -1.57919654e-16  7.21262130e-16
  5.23049037e-15  1.97629269e-15 -1.66533454e-16  6.02391700e-16
 -2.11248643e-15  1.02695630e-15  5.99137598e-16 -2.97080368e-16
  4.05231404e-16  5.42477940e-16 -5.87939659e-16  8.19303916e-16
 -3.88099514e-15  4.95389170e-16  1.94097612e-15  2.30677546e-15
  7.61842696e-17  9.29524656e-16  1.50033243e-15  1.27331096e-15
 -1.53440479e-15  6.04880131e-16 -2.17871698e-15 -5.31567127e-16
  3.65272946e-16  2.27595720e-16  3.55309651e-15 -1.59068161e-15
  1.29283557e-15  2.79546501e-15  2.15316271e-15  3.99527155e-15
 -2.58988233e-15  3.83105903e-15 -1.36366014e-15 -3.27898628e-16
```

```
  3.93605168e-15  4.11261064e-16 -2.94496228e-16 -4.26383067e-17
 -4.54330060e-16  2.49733184e-15 -1.30106653e-15 -3.57367392e-15
  3.72424792e-15  2.15344983e-15  6.29132761e-15  7.22984890e-16
 -6.84012234e-15 -4.51707637e-15 -4.77501180e-15  1.01298280e-15
  1.08581726e-16 -1.16872508e-15 -3.73283779e-15 -4.27244447e-16
  1.66179331e-15 -1.15999164e-16 -9.36032860e-17 -6.08708486e-17
  1.77922811e-16  2.49149360e-15 -1.52062271e-15  8.25010558e-15
  2.45474139e-15  2.32993701e-15 -5.42171672e-15  1.17109387e-15
  1.37672440e-15 -2.33701947e-15  1.06256000e-15 -1.14678382e-15
 -1.96279774e-15  7.53420315e-16 -1.00264624e-15 -6.63071131e-16
 -6.64602473e-16 -6.73790525e-16 -4.40260855e-16 -1.73041658e-16
 -5.78081644e-17 -1.38509893e-15  2.57916294e-15  5.07103937e-15
  4.56071962e-15  1.93446791e-15 -1.26220873e-15  3.19093411e-16
  6.50916103e-16  3.64229719e-15  5.05364428e-15  2.95549026e-16
 -3.66603299e-15  2.85250750e-15  9.00429157e-16  1.29934377e-15
  3.59195432e-16 -2.70090463e-16 -1.34834672e-15 -2.20819531e-15
  3.13886848e-15 -1.75300387e-15 -9.23016453e-16  6.74939032e-16
  1.87503270e-15  2.43134057e-15  3.51911986e-15  1.92834254e-15
  2.71238970e-16 -4.31455638e-15 -1.48367908e-15 -1.79951839e-15
  7.15902433e-17  1.17683641e-15 -4.83425560e-16  4.48491818e-16
  8.13908328e-16 -3.17600352e-15  5.32141381e-16 -1.16271935e-15
 -2.15651252e-15 -1.07304012e-15  2.02692269e-15  1.09681182e-15
  1.13989278e-15 -2.27978556e-16 -1.92987389e-15 -3.16566696e-15
  1.17721924e-15 -2.27710571e-15  1.38261050e-15 -1.83952470e-16
  2.39511476e-16 -1.33475606e-15 -4.35283993e-16  2.90648731e-15
 -1.25072366e-15  3.49490551e-15 -1.07576783e-15 -7.16266127e-15
  2.09640734e-15  3.27745494e-15  1.99840144e-15 -4.04752860e-16
  3.03186595e-15  2.14426178e-15 -1.92949105e-16  7.10542736e-16
  1.68103079e-15 -1.05988015e-15  4.28201536e-16  1.11022302e-16
 -3.61836997e-15  2.44670184e-15  5.83249923e-16 -2.79067956e-15
 -2.42870858e-15  3.80021685e-15 -1.05949732e-15 -3.51079319e-15
 -3.56008326e-15 -2.38085413e-15 -1.95514103e-15  5.74712691e-15
  2.60902411e-15 -9.00811992e-16  2.40210151e-15 -4.60359720e-16
 -4.55287149e-16 -9.90395505e-16 -3.40168506e-15 -6.13225945e-15
 -2.99109396e-15  1.72926807e-15  5.81460167e-15 -8.05868782e-17
  2.92256640e-15 -1.47822367e-15 -4.91393324e-16 -7.13509711e-16
 -1.36940268e-15  2.25413558e-15 -5.69305139e-15 -3.84898052e-15
  1.06782399e-15 -1.92594982e-15 -6.86424098e-16 -2.23106974e-15
 -1.31159451e-15 -7.11576392e-15 -1.00532609e-15 -9.56897397e-16
  1.33762733e-15 -3.18078897e-15 -1.51507159e-15  5.80570075e-16
 -7.34929359e-15 -6.23265807e-15 -1.11764046e-15 -5.39683241e-15
  1.33638311e-15 -1.28441319e-16  3.38924291e-15 -3.92980667e-16
 -4.77108774e-17  2.02730553e-15 -7.56195872e-16 -1.69366437e-15
 -4.35819962e-15  5.43128760e-15  2.46182385e-15  3.21155937e-15
  1.81081204e-15 -8.96349565e-16 -2.62634742e-15 -1.43333621e-15
  2.69707628e-15  7.97254982e-16  5.31313499e-15  5.57791361e-16
 -1.67911662e-15  1.71012629e-15 -6.19236463e-16 -3.44532831e-15
  3.26175868e-16  3.41642423e-15  7.67872356e-16  3.97545981e-15
  2.50718986e-15 -1.83588776e-15  2.77084390e-15  2.60414295e-15
 -1.79281877e-15 -1.44137575e-15  4.54042934e-16 -1.46568581e-15
```

```
 -3.74604562e-16 -2.75756429e-15  4.86392535e-16 -2.07611706e-15
 -2.51235814e-16  2.46928914e-16 -2.88945113e-15  3.10680600e-15
 -3.66641583e-15 -4.24468889e-16 -1.27292812e-15  1.65002112e-15
  6.70431144e-15  1.30010945e-15  4.68183921e-15 -6.02764964e-15
 -1.12046387e-15  1.25550911e-15 -2.85212467e-16  6.00984779e-15
 -3.57070695e-15 -7.89024019e-16 -2.45780408e-16 -3.55079950e-16
 -2.85116758e-16 -1.28537028e-16  2.14000273e-15  4.56166474e-16
 -2.80618440e-16  4.63116135e-15  6.19207750e-15 -1.32748219e-16
 -8.56403071e-16  4.09940281e-15  1.51545443e-15  3.46045032e-15
 -3.77772526e-15 -1.14754949e-15 -2.38640525e-15  2.48058279e-15
 -9.10095754e-16  2.34008215e-15 -1.19817949e-15 -1.22488226e-15
  2.27844563e-15  8.60997097e-16  9.05980272e-16  3.48447325e-15
 -7.64355054e-17  2.24781879e-15  5.67075122e-16 -4.46960476e-16
 -3.96627176e-15 -1.19837090e-15 -3.83792615e-17 -1.52942793e-16
  1.67452259e-15  3.03052602e-15  7.44710806e-16  1.23349606e-15
  1.15597187e-15 -2.70970985e-15  1.24679960e-15  1.61462827e-15
  2.19994042e-15 -7.33895703e-16 -3.11273995e-15  3.14319930e-15
 -7.62416949e-16 -3.37469516e-15 -4.85052611e-16 -2.41913769e-15
 -2.55504430e-15 -5.84589848e-16 -3.12068379e-15  2.23958783e-16
  4.37054607e-16 -1.70361809e-16  3.62851511e-15 -3.68689753e-15
  7.94575134e-16 -2.28648518e-15  3.46753277e-15 -2.96620965e-15
 -2.51982343e-15  3.06172712e-15 -4.07911253e-16  1.92719404e-15
 -2.82800603e-15 -3.95756225e-16 -1.70601081e-16  3.92033149e-15
  6.39574600e-16  2.29892733e-15  1.35026090e-15  1.63958885e-15
 -1.70419234e-15 -3.61970990e-16  2.74110236e-16  9.71540855e-16
 -2.41033247e-15  1.40739910e-15 -9.39035727e-16  2.77641894e-15
 -1.70151249e-15 -3.24922082e-15 -2.12569426e-15  2.37836570e-15
  1.63154930e-15 -2.97721617e-15 -1.33454072e-15  1.69275513e-15
 -2.73392420e-16 -8.68270972e-16  1.86421759e-15 -8.05103111e-16
  1.99916712e-15  2.82168924e-15  2.21393785e-15 -4.71863927e-15
  3.13456158e-15  2.25528408e-15 -1.25220715e-15 -3.20986054e-15
  1.61121116e-15 -1.92480131e-15  1.78688482e-16  1.24392833e-15
 -3.26750121e-16  1.51315742e-15  7.04417367e-17  6.11962588e-16
 -1.51449734e-15 -2.00739808e-15 -1.34183852e-16 -3.84366868e-16
 -2.39693323e-15 -1.03365592e-17  1.29910450e-15 -2.25802076e-15
 -1.70438376e-15  4.09787147e-15 -1.20334777e-15 -1.51478447e-15
  1.84411873e-15 -6.43355101e-16 -2.28227399e-15  1.22354234e-15
 -3.40149365e-16  1.36133920e-15 -2.80905567e-16  1.89235600e-15
  7.55047366e-16 -1.45190373e-15 -1.78516206e-15 -1.40328362e-15
 -1.57211409e-15 -1.40299649e-15 -5.37577645e-15 -1.00468006e-15
 -3.97924031e-15 -6.13338403e-16 -2.66348246e-15  3.18327740e-16
  8.97103273e-16  7.90268234e-16 -1.33092771e-15  1.66983285e-15
 -5.90416126e-16  3.59291141e-15  1.78363071e-15  4.68774923e-15
 -1.75702365e-15 -8.51139083e-16 -5.50460061e-15 -1.44989384e-15
 -5.09281314e-15  3.10623175e-15 -4.74103515e-15  2.12655564e-15
  1.63482733e-17 -1.87053438e-15  4.26048086e-16  7.96795131e-16]
Variance of standardized training data: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
```

```
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.]
```

1. Perform standardization on the test dataset using the means and variances you obtained from the training dataset.

```python
# Use the mean and variance from the training data to transform the test data
X_test_standardized = scaler.transform(X_test)

from sklearn.model_selection import StratifiedKFold

# Initialize stratified 5-fold cross-validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

1. Perform stratified 5-fold cross-validation on the 4-class classification problem using the three classification methods (available on canvas) assigned to you.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report

classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
    "Neural Network": MLPClassifier(max_iter=10000),
    "AdaBoost": AdaBoostClassifier(n_estimators=1000)
}
```

Plot the (3) confusion matrices for using three approaches (clearly label the classes) on the test set

## References

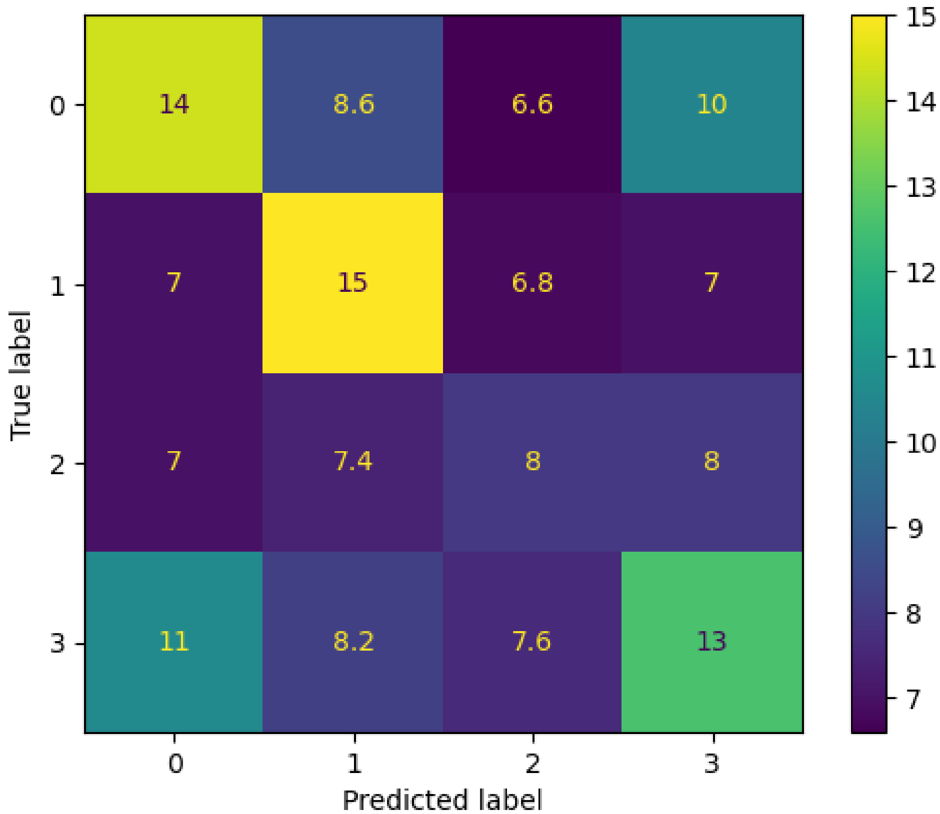
Classification Report for Prediction
https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.classification_report.html

Confusion matrices for classification
https://scikit-learn.org/dev/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

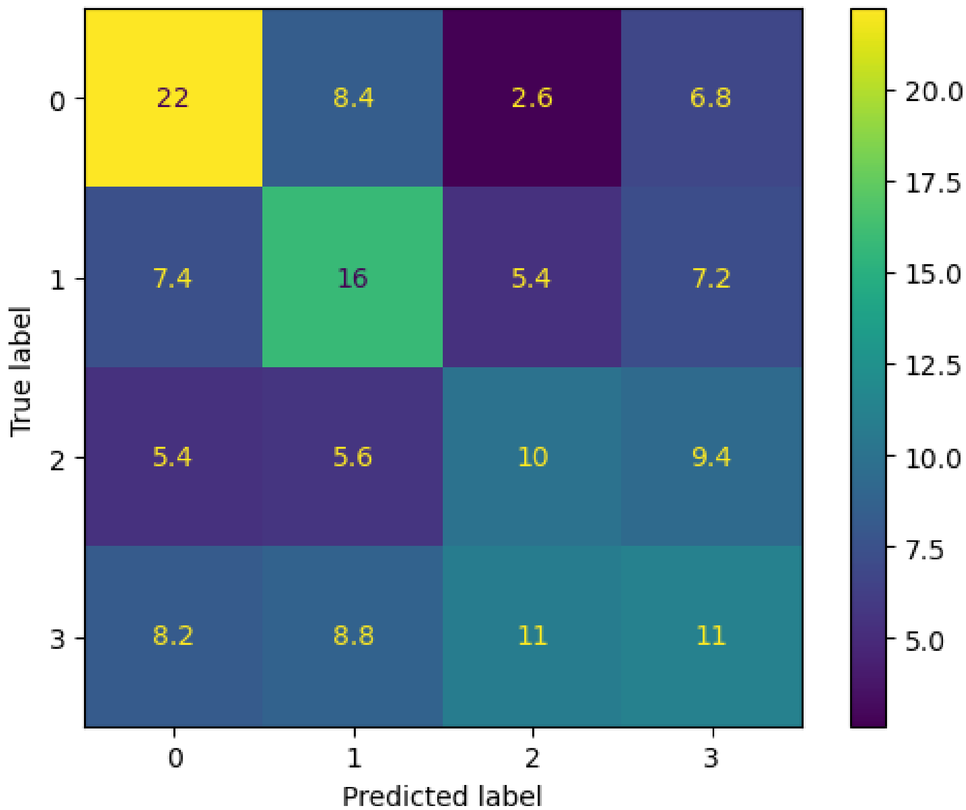
```python
from sklearn import metrics

for clf in classifiers.values():
    print(str(clf)+"\n\n")
    clf.fit(X_train_standardized,y_train)
    predictions=clf.predict(X_test_standardized)
    confusion_matrix = metrics.confusion_matrix(y_test, predictions)
    report=metrics.classification_report(y_test, predictions)
    print(report)
    truelabels, predictlabels, cm, val_a=[], [], [], []
    for traini, testi in kfold.split(X,y):
        xtrain, xtest=X[traini], X[testi]
        ytrain, ytest=y[traini], y[testi]

        clf.fit(xtrain, ytrain)
        p=clf.predict(xtest)
```

```
        truelabels.extend(ytest)
        predictlabels.extend(p)
        val_a.append(metrics.accuracy_score (ytest,p))
        cm.append(metrics.confusion_matrix(ytest,p))
    print("Mean validation acc: "+str(np.mean(val_a)))
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
sum(cm)/len(cm))
    cm_display.plot()
    plt.show()
```

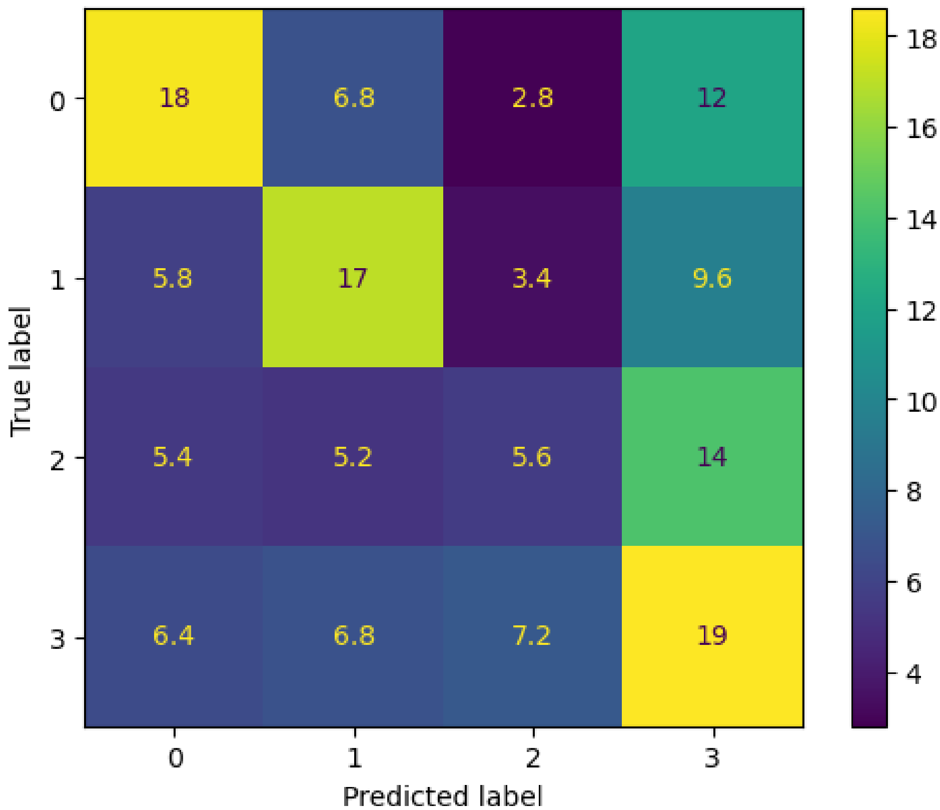
```
DecisionTreeClassifier()


              precision    recall  f1-score   support

           0       0.31      0.30      0.30        40
           1       0.40      0.44      0.42        36
           2       0.19      0.16      0.17        31
           3       0.35      0.36      0.35        39

    accuracy                           0.32       146
   macro avg       0.31      0.32      0.31       146
weighted avg       0.32      0.32      0.32       146

Mean validation acc: 0.3443268776570618
```




```
MLPClassifier(max_iter=10000)


              precision    recall  f1-score   support

           0       0.57      0.72      0.64        40
           1       0.54      0.53      0.54        36
           2       0.45      0.29      0.35        31
           3       0.42      0.44      0.43        39

    accuracy                           0.51       146
   macro avg       0.50      0.49      0.49       146
weighted avg       0.50      0.51      0.50       146

Mean validation acc: 0.40775625885687294
```




```
AdaBoostClassifier(n_estimators=1000)


              precision    recall  f1-score   support

           0       0.59      0.42      0.49        40
           1       0.63      0.61      0.62        36
           2       0.27      0.13      0.17        31
           3       0.34      0.59      0.43        39

    accuracy                           0.45       146
   macro avg       0.46      0.44      0.43       146
weighted avg       0.46      0.45      0.44       146

Mean validation acc: 0.410533774208786
```




1. The MLPClassifier confusion matrix performs best, showing the highest diagonal values indicating correct predictions across classes, with fewer misclassifications than the first and slightly better than the third.

2. Best Method According to Mean Validation Accuracy The mean validation accuracies for the three methods are: Decision Tree: 0.3319 AdaBoost: 0.4105 Neural Network (MLP): 0.4174 The highest mean validation accuracy is achieved by MLPClassifier (Neural Network) with a mean accuracy of 0.4174. Hence, the MLPClassifier is the best method based on mean validation accuracy.

3. Best Method According to Test Set Accuracy The test set accuracies for the three methods are:

Decision Tree: 0.35 AdaBoost: 0.45 Neural Network (MLP): 0.49 The highest test set accuracy is achieved by MLPClassifier (0.49). Therefore, MLPClassifier is also the best method based on test set accuracy.

1. Best Method According to F-Measure Comparing the weighted average F-measures on the test set:

Decision Tree: 0.35 AdaBoost: 0.44 Neural Network (MLP): 0.49 The MLPClassifier again has the highest weighted average F-measure (0.49). Thus, it is the best method according to the F-measure on the test set.

Use images from TWO classes.

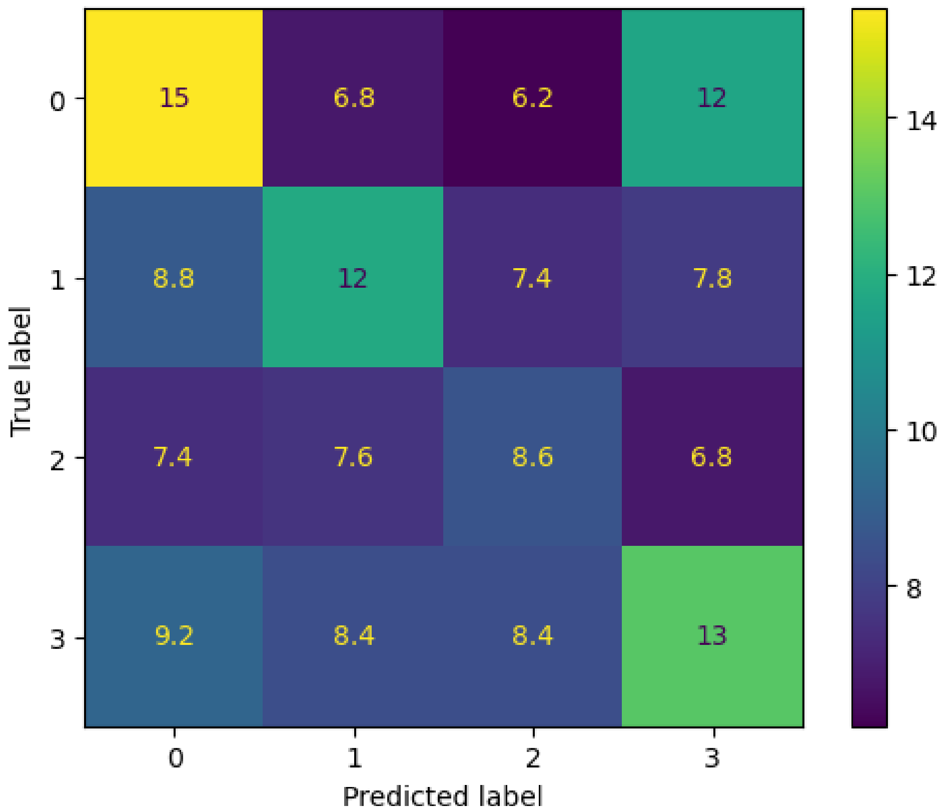
```
all_classes = os.listdir(images_dir)
selected_classes = random.sample(all_classes, 2)  # Randomly select 2
classes

# Initialize lists for histograms and labels
sel_histograms = []
sel_labels = []

# Process only the selected classes
for index, class_name in enumerate(selected_classes):
    class_dir = os.path.join(images_dir, class_name)
    for image_name in os.listdir(class_dir):
        img_path = os.path.join(class_dir, image_name)
        img = imread(img_path)

        # Compute the edge histogram
        edge_hist = compute_edge_histogram(img, grid_size=(4, 4),
nbins=36)

        # Append histogram and label
        sel_histograms.append(edge_hist)
        sel_labels.append(index)  # Label as 0 or 1 based on the
selected class

# Convert histograms and labels to arrays for further processing
selected_classes_histograms = np.array(sel_histograms)
selected_classes_labels = np.array(sel_labels)

Selected_X = selected_classes_histograms
Selected_Y = selected_classes_labels

selected_X_train, selected_X_test, selected_y_train, selected_y_test =
train_test_split(
    Selected_X, Selected_Y, test_size=0.2, random_state=42)
```

Perform a standard 5-fold cross-validation and a stratified 5-fold cross-validation on the training set

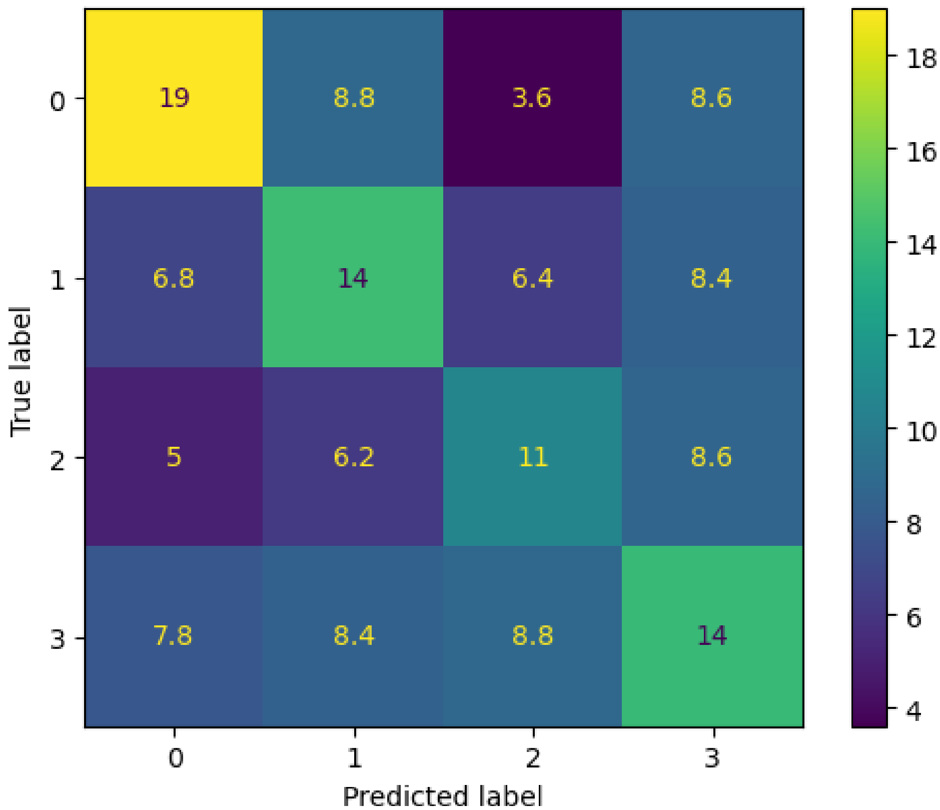
```
# Initialize the StandardScaler
Selected_scaler = StandardScaler()

# Fit the scaler on the training data and transform the training data
selected_X_train_standardized =
Selected_scaler.fit_transform(selected_X_train)

# Use the mean and variance from the training data to transform the
test data
selected_X_test_standardized = scaler.transform(selected_X_test)
```

```python
# Initialize stratified 5-fold cross-validation
selected_kfold = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)

for clf in classifiers.values():
    print(str(clf)+"\n\n")
    clf.fit(selected_X_train_standardized,selected_y_train)
    selected_predictions=clf.predict(selected_X_test_standardized)
    confusion_matrix = metrics.confusion_matrix(selected_y_test,
selected_predictions)
    report=metrics.classification_report(selected_y_test,
selected_predictions)
    print(report)
    truelabels, predictlabels, cm, val_a=[], [], [], []
    for traini, testi in selected_kfold.split(X,y):
        xtrain, xtest=X[traini], X[testi]
        ytrain, ytest=y[traini], y[testi]

        clf.fit(xtrain, ytrain)
        p=clf.predict(xtest)

        truelabels.extend(ytest)
        predictlabels.extend(p)
        val_a.append(metrics.accuracy_score (ytest,p))
        cm.append(metrics.confusion_matrix(ytest,p))
    print("Mean validation acc: "+str(np.mean(val_a)))
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
sum(cm)/len(cm))
    cm_display.plot()
    plt.show()
```

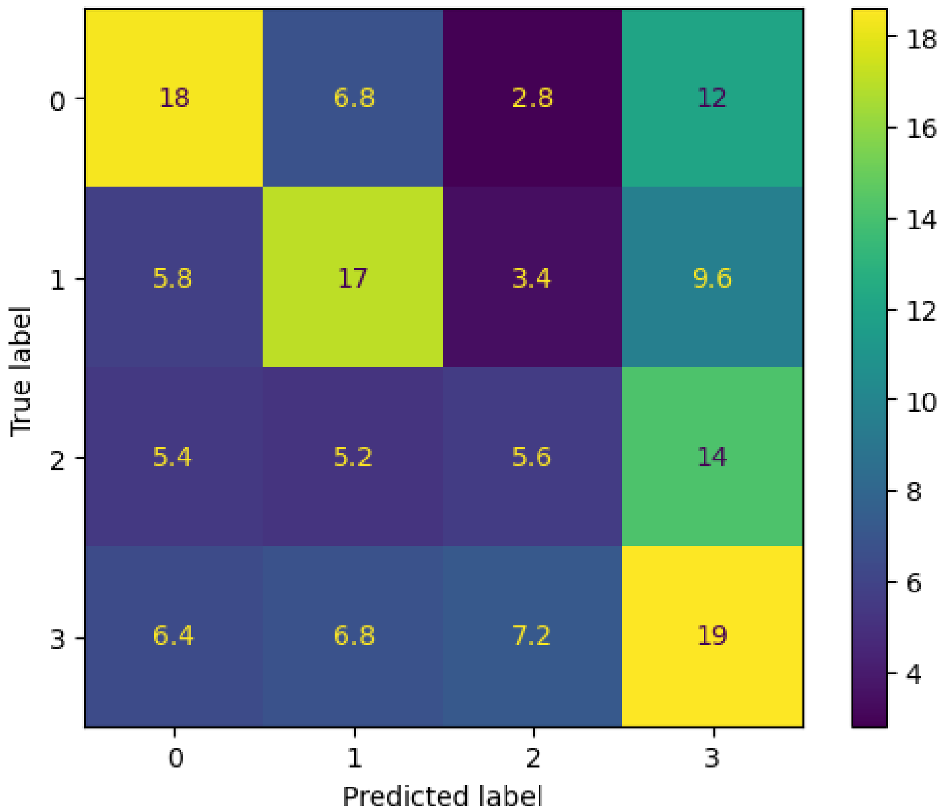
```
DecisionTreeClassifier()


              precision    recall  f1-score   support

           0       0.72      0.65      0.68        43
           1       0.58      0.66      0.62        32

    accuracy                           0.65        75
   macro avg       0.65      0.65      0.65        75
weighted avg       0.66      0.65      0.66        75

Mean validation acc: 0.3360415682569674
```




```
MLPClassifier(max_iter=10000)

              precision    recall  f1-score   support

           0       0.73      0.70      0.71        43
           1       0.62      0.66      0.64        32

    accuracy                           0.68        75
   macro avg       0.67      0.68      0.68        75
weighted avg       0.68      0.68      0.68        75

Mean validation acc: 0.3981105337742088
```




```
AdaBoostClassifier(n_estimators=1000)


              precision    recall  f1-score   support

           0       0.81      0.58      0.68        43
           1       0.59      0.81      0.68        32

    accuracy                           0.68        75
   macro avg       0.70      0.70      0.68        75
weighted avg       0.71      0.68      0.68        75

Mean validation acc: 0.410533774208786
```




Support Vector Classifiers using LinearSVC such that parameter $C$ = 0.1, 1, 10, 100 and other parameters set as default.

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import KFold, StratifiedKFold,
cross_val_score
from sklearn.preprocessing import StandardScaler
import numpy as np

# Assuming histograms and labels are preprocessed and standardized as
X and y respectively
# Example: X = standardized histograms, y = corresponding labels

# Define the values for parameter C
C_values = [0.1, 1, 10, 100]

# Initialize results dictionary
results_standard = {C: [] for C in C_values}
results_stratified = {C: [] for C in C_values}

# Perform standard 5-fold cross-validation for each C value
kf = KFold(n_splits=5, shuffle=True, random_state=42)
for C in C_values:
    model = LinearSVC(C=C, max_iter=10000)
    scores = cross_val_score(model, X_train, y_train, cv=kf,
```

```
scoring='accuracy')
    results_standard[C] = scores
    print(f"Standard 5-fold CV | C={C} | Mean Accuracy:
{np.mean(scores):.4f} | Std Dev: {np.std(scores):.4f}")

# Perform stratified 5-fold cross-validation for each C value
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for C in C_values:
    model = LinearSVC(C=C, max_iter=10000)
    stratified_scores = cross_val_score(model, X_train, y_train,
cv=skf, scoring='accuracy')
    results_stratified[C] = stratified_scores
    print(f"Stratified 5-fold CV | C={C} | Mean Accuracy:
{np.mean(stratified_scores):.4f} | Std Dev:
{np.std(stratified_scores):.4f}")
```

```
Standard 5-fold CV | C=0.1 | Mean Accuracy: 0.3000 | Std Dev: 0.0264
Standard 5-fold CV | C=1 | Mean Accuracy: 0.3586 | Std Dev: 0.0286
Standard 5-fold CV | C=10 | Mean Accuracy: 0.3897 | Std Dev: 0.0320
Standard 5-fold CV | C=100 | Mean Accuracy: 0.3879 | Std Dev: 0.0299
Stratified 5-fold CV | C=0.1 | Mean Accuracy: 0.3121 | Std Dev: 0.0127
Stratified 5-fold CV | C=1 | Mean Accuracy: 0.3672 | Std Dev: 0.0410
Stratified 5-fold CV | C=10 | Mean Accuracy: 0.3862 | Std Dev: 0.0158
Stratified 5-fold CV | C=100 | Mean Accuracy: 0.3914 | Std Dev: 0.0339
```

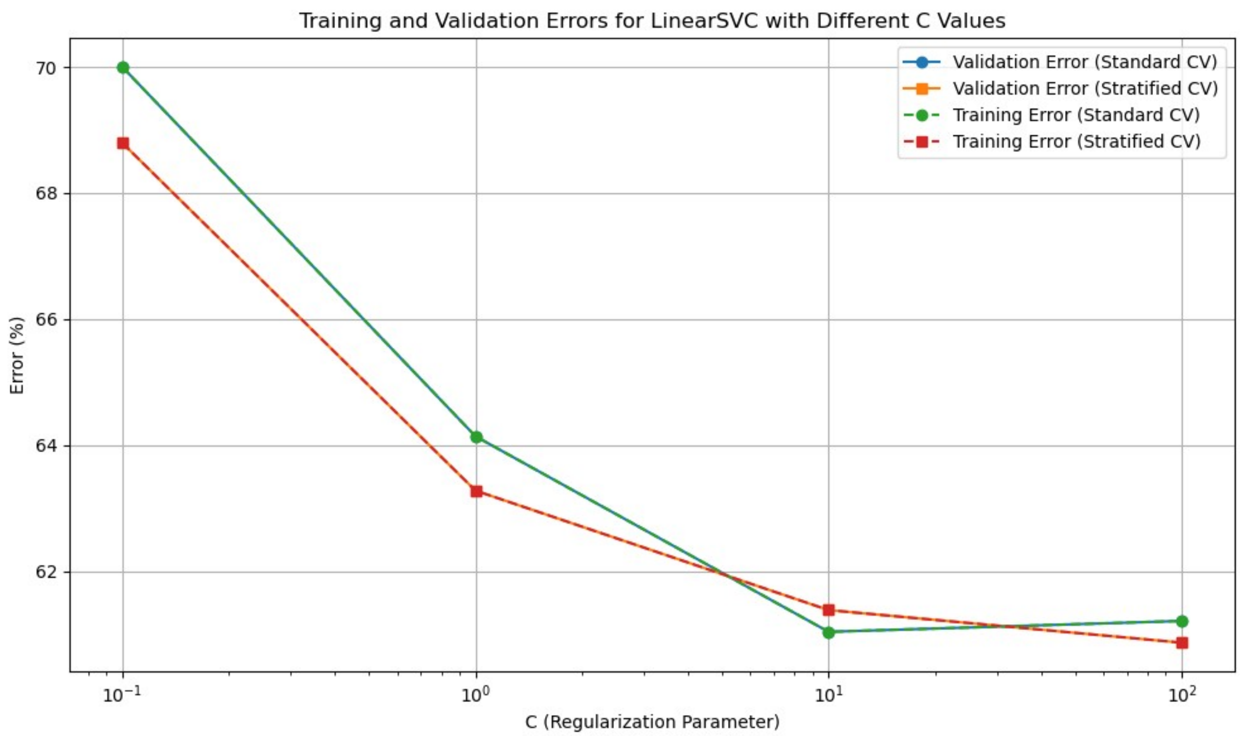
Comment about (1) the model complexity for SVM in relation to $C$, and (2) when/whether there is overfitting/underfitting.

The model is under fitting because the training data is not necessarily the same as the test and the accuracy is low and the standard deviation is not very good

• Plot a graph (x-axis: $C$; y-axis: mean validation/training error (%)) containing four error curves (2 validation error curves and 2 training error curves - label them clearly using a legend to define the curves).

```
import matplotlib.pyplot as plt

# Store mean validation and training errors
train_errors_standard = []
val_errors_standard = []
train_errors_stratified = []
val_errors_stratified = []

# Perform cross-validation and calculate errors
kf = KFold(n_splits=5, shuffle=True, random_state=42)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for C in C_values:
    # Standard cross-validation
    model_standard = LinearSVC(C=C, max_iter=10000)
```

```
    val_scores_standard = cross_val_score(model_standard, X_train,
y_train, cv=kf, scoring='accuracy')
    train_scores_standard = cross_val_score(model_standard, X_train,
y_train, cv=kf, scoring='accuracy')
    val_errors_standard.append(1 - np.mean(val_scores_standard))
    train_errors_standard.append(1 - np.mean(train_scores_standard))

    # Stratified cross-validation
    model_stratified = LinearSVC(C=C, max_iter=50000)
    val_scores_stratified = cross_val_score(model_stratified, X_train,
y_train, cv=skf, scoring='accuracy')
    train_scores_stratified = cross_val_score(model_stratified,
X_train, y_train, cv=skf, scoring='accuracy')
    val_errors_stratified.append(1 - np.mean(val_scores_stratified))
    train_errors_stratified.append(1 -
np.mean(train_scores_stratified))

# Plotting
plt.figure(figsize=(10, 6))

# Validation error curves
plt.plot(C_values, np.array(val_errors_standard) * 100,
label='Validation Error (Standard CV)', marker='o')
plt.plot(C_values, np.array(val_errors_stratified) * 100,
label='Validation Error (Stratified CV)', marker='s')

# Training error curves
plt.plot(C_values, np.array(train_errors_standard) * 100,
label='Training Error (Standard CV)', linestyle='--', marker='o')
plt.plot(C_values, np.array(train_errors_stratified) * 100,
label='Training Error (Stratified CV)', linestyle='--', marker='s')

# Formatting the plot
plt.xscale('log')
plt.xlabel('C (Regularization Parameter)')
plt.ylabel('Error (%)')
plt.title('Training and Validation Errors for LinearSVC with Different
C Values')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
print('train_errors_standard --', train_errors_standard )
print('train_errors_stratified --', train_errors_stratified)
print('val_errors_standard --', val_errors_standard)
print('val_errors_stratified --', val_errors_stratified)

train_errors_standard -- [0.7, 0.6413793103448275, 0.6103448275862069,
0.6120689655172413]
train_errors_stratified -- [0.6879310344827586, 0.6327586206896552,
0.6137931034482759, 0.6086206896551725]
val_errors_standard -- [0.7, 0.6413793103448275, 0.6103448275862069,
0.6120689655172413]
val_errors_stratified -- [0.6879310344827586, 0.6327586206896552,
0.6137931034482759, 0.6086206896551725]

val_errors_standard

[0.7, 0.6413793103448275, 0.6103448275862069, 0.6120689655172413]
```

Which $C$ has/have the lowest mean error for each curve?

```
# Finding the C value with the lowest mean validation error
optimal_C = min(results_stratified, key=lambda c: np.mean(1 -
results_stratified[c]))
print(f"Optimal C (from stratified 5-fold CV): {optimal_C}")
# Finding the C value with the lowest mean validation error
optimal_stan = min(results_standard, key=lambda c: np.mean(1 -
results_standard[c]))
```

```
print(f"Optimal C (from stratified 5-fold CV): {optimal_C}")
print('The lowest mean error for each curve', min(optimal_stan,
optimal_C))
```

```
Optimal C (from stratified 5-fold CV): 100
Optimal C (from stratified 5-fold CV): 100
The lowest mean error for each curve 10
```

Use the $C$ value with the lowest mean validation error for your SVM classifier from the stratified 5-fold cross-validation. What is the error for the test dataset

```
# Train the final model with this optimal C on the entire training set
final_model = LinearSVC(C=optimal_C, max_iter=10000)
final_model.fit(X_train, y_train)

# Predict on the test dataset
y_test_pred = final_model.predict(X_test)

# Calculate test error
test_accuracy = np.mean(y_test == y_test_pred)
test_error = 1 - test_accuracy
print(f"Test Error: {test_error * 100:.2f}%")
```

```
Test Error: 58.22%
```