```
import argparse
import os
import pathlib
import math

from multiprocessing import Process

import numpy as np
from numba import jit
import scipy.io as sio
import matplotlib.pyplot as plt
from PIL import Image
from tqdm import tqdm

import utils

try:
    from icecream import ic
except ImportError:  # Graceful fallback if IceCream isn't installed.
    ic = lambda *a: None if not a else (a[0] if len(a) == 1 else a)  # noqa


def mu_n(sample_mean, prior_mu0, sample_cov, prior_sigma0, n: int, a_inv:
np.ndarray):
    """
    a_inv = np.linalg.inv(Sigma_0 + Sigma / n)
    """
    # (64,64) @ (64,64) @ (64, 1) + (64, 64) @ (64, 64) @ (64, 1) -> (64,
1) + (64,1)
    return (prior_sigma0 @ a_inv @ sample_mean) + (sample_cov @ a_inv @
prior_mu0) / n


@jit(nopython=True)
def sigma_n(sigma, sigma_0, n: int, a_inv: np.ndarray):
    return sigma_0 @ a_inv * sigma / n


@jit(nopython=True)
def g(x, W, w, w0):
    """
    Decision boundary function g_i(x).
    """
    return x.T @ W @ x + w.T @ x + w0


def ML_result(BG, FG):
    current_dir = pathlib.Path(__file__).parent.resolve()
    data_dir = current_dir / "data"
    TrainsampleDCT_BG = BG
    TrainsampleDCT_FG = FG
```

```
    m_FG_ML, n_FG = TrainsampleDCT_FG.shape
    m_BG_ML, n_BG = TrainsampleDCT_BG.shape

    P_FG_ML = m_FG_ML / (m_FG_ML + m_BG_ML)
    P_BG_ML = m_BG_ML / (m_FG_ML + m_BG_ML)
    assert P_FG_ML + P_BG_ML == 1

    #
================================================================================
============
    # ML mean mu
    mu_FG_ML = np.mean(TrainsampleDCT_FG, axis=0).reshape(-1, 1)
    mu_BG_ML = np.mean(TrainsampleDCT_BG, axis=0).reshape(-1, 1)

    # ML covariance Sigma
    cov_FG_ML, cov_BG_ML = np.cov(TrainsampleDCT_FG.T),
np.cov(TrainsampleDCT_BG.T)

    img = np.asarray(Image.open(os.path.join(data_dir, "cheetah.bmp"),
"r"))

    # Convert to double and / 255
    img = utils.im2double(img)
    assert img.min() == 0 and img.max() <= 1

    ground_truth = np.asarray(
        Image.open(os.path.join(data_dir, "cheetah_mask.bmp"), "r")
    )

    processed_img = np.zeros([img.shape[0] - 8, img.shape[1] - 8],
dtype=bool)

    # constants

    logp_FG_ML = np.log(P_FG_ML)
    logp_BG_ML = np.log(P_BG_ML)

    logdet_FG_ML = np.log(np.linalg.det(cov_FG_ML))
    logdet_BG_ML = np.log(np.linalg.det(cov_BG_ML))

    W_FG = np.linalg.inv(cov_FG_ML)
    W_BG = np.linalg.inv(cov_BG_ML)

    w_FG = -2 * W_FG @ mu_FG_ML
    w_BG = -2 * W_BG @ mu_BG_ML

    w0_FG = mu_FG_ML.T @ W_FG @ mu_FG_ML + logdet_FG_ML - 2 * logp_FG_ML
    w0_BG = mu_BG_ML.T @ W_BG @ mu_BG_ML + logdet_BG_ML - 2 * logp_BG_ML

    # Feature vector 64 x 1
    x_64 = np.zeros((64, 1), dtype=np.float64)
    for i in range(processed_img.shape[0]):
        for j in range(processed_img.shape[1]):
            # 8 x 8 block
```

```
            block = img[i : i + 8, j : j + 8]
            # DCT transform on the block
            block_DCT = utils.dct2(block)
            # zigzag pattern mapping
            for k in range(block_DCT.shape[0]):
                for p in range(block_DCT.shape[1]):
                    loc = utils.zigzag[k, p]
                    x_64[loc, :] = block_DCT[k, p]

            if g(x_64, W_FG, w_FG, w0_FG) >= g(x_64, W_BG, w_BG, w0_BG):
                processed_img[i, j] = 0
            else:
                processed_img[i, j] = 1
    errors_ML, _, _ = utils.calculate_error(processed_img, ground_truth,
verbose=False)
    return errors_ML


def run(strategy, plot: bool = True, save: bool = False, test: bool =
False):
    current_dir = pathlib.Path(__file__).parent.resolve()
    data_dir = current_dir / "data"
    prior1_fname = data_dir / "Prior_1.mat"
    prior2_fname = data_dir / "Prior_2.mat"
    alpha_fname = data_dir / "Alpha.mat"
    mat_fname_subset = data_dir / "TrainingSamplesDCT_subsets_8.mat"
    plot_dir = current_dir / "plots"

    # Create the directory if it does not exist
    for d in [data_dir, plot_dir]:
        if not os.path.exists(d):
            os.mkdir(d)

    #
============================================================================
============
    # Load the data
    subsets8 = sio.loadmat(mat_fname_subset)
    prior_1 = sio.loadmat(prior1_fname)
    prior_2 = sio.loadmat(prior2_fname)
    alpha_dict = sio.loadmat(alpha_fname)

    # weights
    alpha = alpha_dict["alpha"].ravel()

    #
============================================================================
============
    # Load Images
    # load Image (original_img has dtype=uint8)
    img = np.asarray(Image.open(os.path.join(data_dir, "cheetah.bmp"),
"r"))
    # Convert to double and / 255
    img = utils.im2double(img)
```

```
    assert img.min() == 0 and img.max() <= 1
    ground_truth = np.asarray(
        Image.open(os.path.join(data_dir, "cheetah_mask.bmp"), "r")
    )

    #
============================================================================
============
    # Handle stragtegys
    if strategy == 1:
        prior = prior_1
    elif strategy == 2:
        prior = prior_2
    else:
        raise ValueError("Invalid strategy. Choice:[1, 2]")

    print(f"Strategy: {strategy}")
    err_bayes = []
    err_mle = []
    err_map = []

    #
============================================================================
============
    pbar_idx = 1
    for subset_idx in tqdm(
        range(1, 5), dynamic_ncols=True, desc=f"Dataset ({pbar_idx})"
    ):
        pbar_idx += 1
        D_BG = subsets8[f"D{subset_idx}_BG"]
        D_FG = subsets8[f"D{subset_idx}_FG"]

        # n_samples and m_features
        n_BG, m_BG = D_BG.shape
        n_FG, m_BG = D_FG.shape

        # prior
        total_samples = n_BG + n_FG
        P_BG = n_BG / total_samples
        P_FG = n_FG / total_samples
        print(f"\tprior_BG: {P_BG}")
        print(f"\tprior_FG: {P_FG}")

        prior_mu0_BG = prior["mu0_BG"].reshape(-1, 1)
        prior_mu0_FG = prior["mu0_FG"].reshape(-1, 1)

        # \hat{\mu_n} sample mean
        BG_mean = np.mean(D_BG, axis=0).reshape(-1, 1)
        FG_mean = np.mean(D_FG, axis=0).reshape(-1, 1)

        # \Sigma sample covariance using unbiased estimator
        BG_cov = np.cov(D_BG.T, bias=False)
        FG_cov = np.cov(D_FG.T, bias=False)
```

```
        # log prior
        logp_FG = math.log(P_FG)
        logp_BG = math.log(P_BG)

        # a) Bayesian Estimation
        img_lst = []
        print(f"\tBayesian Estimation with Strategy {strategy}")
        for a in range(alpha.shape[0]):
            processed_img = np.empty([img.shape[0] - 8, img.shape[1] - 8],
dtype=bool)
            # Sigma_0 (with weight = alpha[i] )
            prior_sigma0 = np.diag((alpha[a] * prior["W0"]).flat)

            # import ipdb; ipdb.set_trace()
            # * pre-compute the inverse of Sigma_0 + (Sigma / n)
            a_BG_inv = np.linalg.inv(prior_sigma0 + BG_cov / n_BG)
            a_FG_inv = np.linalg.inv(prior_sigma0 + FG_cov / n_FG)

            # Parameter Distribution
            mu_n_BG = mu_n(BG_mean, prior_mu0_BG, BG_cov, prior_sigma0,
n_BG, a_BG_inv)
            mu_n_FG = mu_n(FG_mean, prior_mu0_FG, FG_cov, prior_sigma0,
n_FG, a_FG_inv)

            cov_n_BG = sigma_n(BG_cov, prior_sigma0, n_BG, a_BG_inv)
            cov_n_FG = sigma_n(FG_cov, prior_sigma0, n_FG, a_FG_inv)

            # Sum of two independent Gaussian is again a Gaussian
            # where mean is the sum of the means
            mu_BG = mu_n_BG
            mu_FG = mu_n_FG
            # and whose covariance matrixis is the sum of the covariance
matrices
            cov_BG = cov_n_BG + BG_cov
            cov_FG = cov_n_FG + FG_cov

            # guassian decsison bounday
            logdet_BG = math.log(np.linalg.det(cov_BG))
            logdet_FG = math.log(np.linalg.det(cov_FG))

            W_BG = np.linalg.inv(cov_BG)
            W_FG = np.linalg.inv(cov_FG)

            w_BG = -2 * W_BG @ mu_BG
            w_FG = -2 * W_FG @ mu_FG

            w0_FG = mu_FG.T @ W_FG @ mu_FG + logdet_FG - 2 * logp_FG
            w0_BG = mu_BG.T @ W_BG @ mu_BG + logdet_BG - 2 * logp_BG

            # Feature vector 64 x 1
            x_64 = np.zeros((64, 1), dtype=np.float64)
            for i in range(processed_img.shape[0]):
                for j in range(processed_img.shape[1]):
                    # # 8 x 8 block
```

```
                    block = img[i : i + 8, j : j + 8]
                    # DCT transform on the block
                    block_DCT = utils.dct2(block)
                    # zigzag pattern mapping
                    for k in range(block_DCT.shape[0]):
                        for p in range(block_DCT.shape[1]):
                            loc = utils.zigzag[k, p]
                            x_64[loc, :] = block_DCT[k, p]

                    if g(x_64, W_FG, w_FG, w0_FG) > g(x_64, W_BG, w_BG,
w0_BG):
                        processed_img[i, j] = 0
                    else:
                        processed_img[i, j] = 1
            img_lst.append(processed_img)
        error_lst_bayes = [
            utils.calculate_error(img, ground_truth, verbose=False)[0]
            for img in img_lst
        ]
        err_bayes.append(error_lst_bayes)

        print(f"\tMaximum Likelihood Estimation with Strategy {strategy}")
        error_lst_ml = ML_result(BG=D_BG, FG=D_FG)
        error_lst_ml = [error_lst_ml] * alpha.shape[0]
        err_mle.append(error_lst_ml)

        # b) Bayes MAP Approximation
        img_lst_MAP = []
        print(f"\tBayesian Estimation with MAP Approximation with Strategy
{strategy}")
        for a in range(alpha.shape[0]):
            processed_img = np.empty([img.shape[0] - 8, img.shape[1] - 8],
dtype=bool)

            # Sigma_0 (with alpha[:, i])
            prior_sigma_0 = np.diag((alpha[a] * prior["W0"]).flat)

            # * pre-compute the inverse of Sigma_0 + Sigma / n
            a_BG_inv = np.linalg.inv(prior_sigma_0 + BG_cov / n_BG)
            a_FG_inv = np.linalg.inv(prior_sigma_0 + FG_cov / n_FG)

            # Parameter Distribution
            mu_n_BG = mu_n(BG_mean, prior_mu0_BG, BG_cov, prior_sigma_0,
n_BG, a_BG_inv)
            mu_n_FG = mu_n(FG_mean, prior_mu0_FG, FG_cov, prior_sigma_0,
n_FG, a_FG_inv)

            # Sum of independent gaussian -- MAP
            mu_BG = mu_n_BG
            mu_FG = mu_n_FG

            cov_BG = BG_cov
            cov_FG = FG_cov
            #
```

```
================================================================================
==============
            # guassian decsison rule
            logdet_BG = np.log(np.linalg.det(cov_BG))
            logdet_FG = np.log(np.linalg.det(cov_FG))

            W_BG = np.linalg.inv(cov_BG)
            W_FG = np.linalg.inv(cov_FG)

            w_BG = -2 * W_BG @ mu_BG
            w_FG = -2 * W_FG @ mu_FG

            w0_BG = mu_BG.T @ W_BG @ mu_BG + logdet_BG - 2 * logp_BG
            w0_FG = mu_FG.T @ W_FG @ mu_FG + logdet_FG - 2 * logp_FG

            # Feature vector 64 x 1
            x_64 = np.empty((64, 1), dtype=np.float64)
            for i in range(processed_img.shape[0]):
                for j in range(processed_img.shape[1]):
                    # # 8 x 8 block
                    block = img[i : i + 8, j : j + 8]
                    # DCT transform on the block
                    block_DCT = utils.dct2(block)
                    # zigzag pattern mapping
                    for k in range(block_DCT.shape[0]):
                        for p in range(block_DCT.shape[1]):
                            loc = utils.zigzag[k, p]
                            x_64[loc, :] = block_DCT[k, p]

                    if g(x_64, W_FG, w_FG, w0_FG) >= g(x_64, W_BG, w_BG,
w0_BG):
                        processed_img[i, j] = 0
                    else:
                        processed_img[i, j] = 1
            img_lst_MAP.append(processed_img)
        error_lst_MAP = [
            utils.calculate_error(img, ground_truth, verbose=False)[0]
            for img in img_lst_MAP
        ]
        err_map.append(error_lst_MAP)

        if test:
            break

    #
================================================================================
==============
    # plot result
    assert len(err_bayes) == len(err_mle) == len(err_map)
    for idx in range(len(err_bayes)):
        plt.figure(figsize=(10, 6), dpi=300)
        plt.plot((alpha.flat), err_bayes[idx], "--o", label="Bayes_Error")
        plt.plot((alpha.flat), err_mle[idx], "--x", label="ML_Error")
        plt.plot((alpha.flat), err_map[idx], "--*", label="MAP_Error")
```

```
        plt.xlabel(r"$\log (\alpha)$")
        plt.ylabel("PoE")
        plt.xscale("log")
        plt.grid()
        plt.title(f"Dataset {idx+1} Strategy {strategy}: " + r"PE vs
$\alpha$")
        plt.legend()
        if save:
            plt.savefig(plot_dir / f"Dataset {idx+1} Strategy
{strategy}.png")
    if plot:
        plt.show()


def main():
    parser = argparse.ArgumentParser(description="HW2")
    parser.add_argument("--plot", "-p", action="store_true", help="Plot the
data")
    parser.add_argument("--save", "-s", action="store_true", help="Save
plots")
    parser.add_argument("--test", "-t", action="store_true", help="Test the
code")
    parser.add_argument("--strategy", type=int)
    args = parser.parse_args()

    if args.strategy == 1 or args.strategy == 2:
        run(args.strategy, args.plot, args.save, args.test)
    else:
        strategies = [1, 2]
        procs = []

        # instantiating process with arguments
        for s in strategies:
            # print(name)
            proc = Process(target=run, args=(s, args.plot, args.save,
args.test))
            procs.append(proc)
            proc.start()

        # complete the processes
        for proc in procs:
            proc.join()

    print("Done!")


if __name__ == "__main__":

    main()
```