```python
import argparse
from pathlib import Path
from collections import defaultdict
from datetime import datetime

import numpy as np

from my_logger import get_my_logger
from utils import load_data, Trainer, plot_accs, tr_va_split
from preprocess import ImagePreprocessor, TextPreprocessor
from save_csv import results_to_csv

if __name__ == "__main__":
    # logger
    logger = get_my_logger(Path(__file__).name)
    logger.info('--- [START {0:%Y-%m-%d_%H-%M-%S}] {1}'.format(datetime.now(), '-' * 100))

    parser = argparse.ArgumentParser(description='cs289 hw1')
    parser.add_argument('problem', type=int, help='problem number (e.g. 2, 3, 4)')
    parser.add_argument('data_name', type=str, help='dataset name (e.g. mnist, spam, cifar10)')
    args = parser.parse_args()

    # -- configuration --------------------
    data_name = args.data_name  # cifar10 mnist spam

    split_and_save = 0
    preprocess = 0
    tune_n_tr = 0
    tune_hparameter = 0
    train_and_predict = 0
    split = 'holdout'  # kfold holdout
    n_fold = 1
    kernel = 'linear'

    n_trs = {
        'mnist': None,  # 5000
        'spam': None,
        'cifar10': None  # 5000
    }

    if args.problem == 2:
        split_and_save = 1
    elif args.problem == 3:
        tune_n_tr = 1
    elif args.problem == 4:
        tune_hparameter = 1
        n_trs['mnist'] = 10000
    elif args.problem == 5:
        tune_hparameter = 1
        split = 'kfold'
        n_fold = 5
    elif args.problem == 6:
        preprocess = 1
        train_and_predict = 1
        split = 'kfold'
        n_fold = 5
        kernel = 'rbf'

    logger.info(f'data:\t{data_name}')
    logger.info(f'prepro:\t{preprocess}')
    logger.info(f'split:\t{split}')
    logger.info(f'n_fold:\t{n_fold}')

    # seed
    seed = 189
    np.random.seed(seed)
    logger.info(f'seed:\t{seed}')

    n_vas = {
        'mnist': 10000,
        'spam': 0.2,
        'cifar10': 5000
    }
    n_va = n_vas[data_name]

    preprocessors = {
        'mnist': ImagePreprocessor(
            normalize=False, hog_tf=True, pca_tf=False, lbp_tf=False,
win_size=(28, 28), block_size=(8, 8),
            block_stride=(4, 4), cell_size=(8, 8), nbins=9
        ),
        'spam': TextPreprocessor(),
        'cifar10': ImagePreprocessor(
            normalize=False, hog_tf=False, pca_tf=True, lbp_tf=False
        )
    }
    preprocessor = preprocessors[data_name]

    n_tr_spaces = {
        'mnist': [100, 200, 500, 1000, 2000, 5000, 10000],
        'spam': [100, 200, 500, 1000, 2000, 4138],
        'cifar10': [100, 200, 500, 1000, 2000, 5000]
    }
    n_tr_space = n_tr_spaces[data_name]

    all_params = {
        'mnist': {
            'C': 2e1,
            'kernel': kernel,
            'gamma': 'scale',
            'random_state': seed
        },
        'spam': {
            'C': 1e1,
            'kernel': kernel,
            'gamma': 'scale',
            'random_state': seed
        },
        'cifar10': {
            'C': 1e1,
            'kernel': kernel,
            'gamma': 'scale',
            'random_state': seed
        }
    }
    params = all_params[data_name]

    all_parameter_spaces = {
        'mnist': {
            'C': np.logspace(-10, 10, 11, base=10)
        },
        'spam': {
            'C': np.logspace(-10, 4, 8, base=10)
        },
        'cifar10': {
            'C': np.logspace(-10, 10, 11, base=10)
        }
    }
    parameter_spaces = all_parameter_spaces[data_name]
    # ------------------------------------

    # load data
    x_train, x_test, y_train = load_data(data_name)
    logger.info(f'x_train:\t{x_train.shape}')
    logger.info(f'y_train:\t{y_train.shape}')
    logger.info(f'x_test:\t{x_test.shape}')

    # preprocessing
    if preprocess:
        logger.info('** preprocess **')
        x_train, x_test = preprocessor.scale(x_train, x_test)
        logger.info(f'x_train:\t{x_train.shape}')
        logger.info(f'x_test:\t{x_test.shape}')

    # tune training data size
    if tune_n_tr:
        logger.info('** tune training data size**')
        accs = defaultdict(dict)
        for n_tr in n_tr_space:
            trainer = Trainer(params, x_train, x_test, y_train,
verbose=False)
            trainer.validate(split, n_fold=n_fold, n_va=n_va, n_tr=n_tr,
shuffle=True)

            tr_acc = trainer.scores_mean['tr_acc']
            va_acc = trainer.scores_mean['va_acc']
            accs['train'][n_tr] = tr_acc
            accs['valid'][n_tr] = va_acc
            logger.info(f'[{n_tr}]\ttr:{tr_acc:.5f}\tva:{va_acc:.5f}')

        path = Path(f'figures/{data_name}_acc_vs_n_tr.png')
        xlabel = 'Training data size'
        xlim = (0, max(n_tr_space) * 1.1)
        plot_accs(accs, data_name, path, xlabel, xlim, False)

    # tune C of SVM with holdout
    if tune_hparameter:
        logger.info('** tune hyper parameter **')
        n_tr = n_trs[data_name]
        best_acc = 0
        best_params = {}

        for parameter, space in parameter_spaces.items():
            accs = defaultdict(dict)
            for value in space:
                params[parameter] = value
                trainer = Trainer(params, x_train, x_test, y_train, verbose=False)
                trainer.validate(split, n_fold=n_fold, n_va=n_va, n_tr=n_tr, shuffle=True)

                tr_acc = trainer.scores_mean['tr_acc']
                va_acc = trainer.scores_mean['va_acc']
                accs['train'][value] = tr_acc
                accs['valid'][value] = va_acc
                logger.info(f'[{parameter}:{value:.0e}]\ttr:{tr_acc:.5f}\tva:{va_acc:.5f}')

                if va_acc > best_acc:
                    best_params = params.copy()
                    best_acc = va_acc

            path = Path(f'figures/{data_name}_acc_vs_{parameter}.png')
            xlabel = parameter
            xlim = (min(space) * 0.9, max(space) * 1.1)
            plot_accs(accs, data_name, path, xlabel, xlim, log_scale=True)

        params = best_params.copy()
        logger.info(f'[best_acc]\t{best_acc:.5f}')
        logger.info(f'[best_params]\t{best_params}')

    # split and save
    if split_and_save:
        logger.info('** split and save **')
        n_tr = n_trs[data_name]
        cv = tr_va_split(x_train, x_test, n_va, shuffle=True)
        ind_tr, ind_va = next(cv)
        x_tr, x_va = x_train[ind_tr[:n_tr]], x_train[ind_va]
        y_tr, y_va = y_train[ind_tr[:n_tr]], y_train[ind_va]

        logger.info(f'x_tr: {x_tr.shape}')
        logger.info(f'x_va: {x_va.shape}')
        logger.info(f'y_tr: {y_tr.shape}')
        logger.info(f'y_va: {y_va.shape}')

        save_dir = Path('data', 'split')
        np.save(Path(save_dir, f'{data_name}_x_tr.npy'), x_tr)
        np.save(Path(save_dir, f'{data_name}_x_va.npy'), x_va)
        np.save(Path(save_dir, f'{data_name}_y_tr.npy'), y_tr)
        np.save(Path(save_dir, f'{data_name}_y_va.npy'), y_va)

    # train and predict
    if train_and_predict:
        logger.info('** train and predict **')
        n_tr = n_trs[data_name]

        trainer = Trainer(params, x_train, x_test, y_train, verbose=True)
        trainer.validate(split, n_fold=n_fold, n_va=n_va, n_tr=n_tr, shuffle=True)

        tr_acc = trainer.scores_mean['tr_acc']
        va_acc = trainer.scores_mean['va_acc']
        logger.info(f'[all]\ttr:{tr_acc:.5f}\tva:{va_acc:.5f}')

        submit_path = Path(f'data/output/submit_{data_name}_{va_acc:.5f}.csv', index=False)
        results_to_csv(trainer.y_test_pred, submit_path)

```