```
#!/usr/bin/env python3

import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
from numpy.linalg import norm
from math import exp

def lstsq(A, b, lambda_=0):
    return np.linalg.solve(A.T @ A + lambda_ * np.eye(A.shape[1]), A.T @ b)

def ploss(X, y, w):
    return norm(X @ w - y)**2/X.shape[0]

def assemble_feature(x, D):
    n_feature = x.shape[1]
    Q = [(np.ones(x.shape[0]), 0, 0)]
    i = 0
    while Q[i][1] < D:
        cx, degree, last_index = Q[i]
        for j in range(last_index, n_feature):
            Q.append((cx * x[:, j], degree + 1, j))
        i += 1
    return np.column_stack([q[0] for q in Q])

def poly_kernel(x, z, p):
    return (1 + x.T @ z)**p

def rbf_kernel(x, z, sigma):
    return exp(-norm(x-z)**2/2/sigma**2)

def kernel_matrix(X, Y, kernel, parameter):
    """
    calculate the kernel matrix given kernel type and kernel parameter
    X, Y have same feature dimension
    kernel : can be 'poly' or 'rbf'
    parameter: if 'poly' then this is p; if 'rbf' then this is sigma
    """
    Z = np.zeros((X.shape[0], Y.shape[0]))
    for i in range(X.shape[0]):
        for j in range(Y.shape[0]):
            if kernel == 'poly':
                Z[i, j] = poly_kernel(X[i, :].T, Y[j, :].T, parameter)
            if kernel == 'rbf':
```

```
            Z[i, j] = rbf_kernel(X[i, :].T, Y[j, :].T, parameter)
    return Z

def kernel_ridge(X, y, lambda_, kernel, parameter):
    K = kernel_matrix(X, X, kernel, parameter)
    return np.linalg.solve(K + lambda_ * np.eye(X.shape[0]), y)

def kloss(X, y, alpha, X_train, kernel, parameter):
    K = kernel_matrix(X, X_train, kernel, parameter)
    return norm(K @ alpha - y)**2/X.shape[0]

def visualize(X, y, ax):
    pos = y[:] == +1.0
    neg = y[:] == -1.0
    ax.scatter(X[pos, 0], X[pos, 1], c='red', marker='+')
    ax.scatter(X[neg, 0], X[neg, 1], c='blue', marker='v')
    return ax

def heatmap(X, y, ax, f, clip=5):
    # example: heatmap(lambda x, y: x * x + y * y)
    # clip: clip the function range to [-clip, clip] to generate a clean plot
    #   set it to zero to disable this function

    xx = yy = np.linspace(np.min(X), np.max(X), 72)
    x0, y0 = np.meshgrid(xx, yy)
    x0, y0 = x0.ravel(), y0.ravel()
    z0 = f(x0, y0)


    if clip:
        z0[z0 > clip] = clip
        z0[z0 < -clip] = -clip

    hb = ax.hexbin(x0, y0, C=z0, gridsize=50, cmap=cm.jet, bins=None)
    #plt.colorbar(ax = ax)
    cs = ax.contour(
        xx, yy, z0.reshape(xx.size, yy.size), [-2, -1, -0.5, 0, 0.5, 1, 2], cmap=cm.jet)
    ax.clabel(cs, inline=1, fontsize=10)

    visualize(X, y, ax)

    return hb, ax
```

```
#def main():
   # example usage of heatmap
#    heatmap(lambda x, y: x * x + y * y)

def a():
   fig, axs = plt.subplots(1, 3, figsize=(12, 4))
   for i, name in enumerate(['circle', 'heart', 'asymmetric']):
      data = np.load(name+'.npz')
      X = data["x"]
      y = data["y"]
      X /= np.max(X)  # normalize the data
      visualize(X, y, axs[i])
   plt.savefig('5a.jpg')
   plt.show()

def get_train_valid(name, split):
   data = np.load(name+'.npz')
   X = data["x"]
   y = data["y"]
   X /= np.max(X)  # normalize the data

   n_train = int(X.shape[0] * split)
   X_train = X[:n_train:, :]
   X_valid = X[n_train:, :]
   y_train = y[:n_train]
   y_valid = y[n_train:]
   return X, y, X_train, y_train, X_valid, y_valid

def b(name):
   SPLIT = 0.8
   LAMBDA = 0.001
   ps = np.arange(1, 17, 1)

   X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

   train_loss = np.zeros(ps.shape[0])
   valid_loss = np.zeros(ps.shape[0])
   for p in ps:
      w = lstsq(assemble_feature(X_train, p), y_train, LAMBDA)
      train_loss[p-1] = ploss(assemble_feature(X_train, p), y_train, w)
      valid_loss[p-1] = ploss(assemble_feature(X_valid, p), y_valid, w)
   print('for p = {} the training loss is {} and validation loss is {}'.format(ps, train_loss, valid_loss))
   plt.semilogy(ps, train_loss, label = 'training loss')
   plt.semilogy(ps, valid_loss, label = 'valid loss')
```

```
    plt.legend()
    plt.title('poly_loss_'+name)
    plt.savefig('poly_loss_'+name+'.jpg')
    plt.show()

    #ps = np.arange(2, 13, 2)
    #fig, axs = plt.subplots(1, ps.shape[0], figsize = (ps.shape[0]*4, 4))
    #for i, p in enumerate(ps):
    #    w = lstsq(assemble_feature(X_train, p), y_train, LAMBDA)
    #    f = lambda x, y: assemble_feature(np.hstack((x.reshape(x.shape[0], 1),
y.reshape(y.shape[0], 1))), p) @ w
    #    heatmap(X, y, axs[i], f, clip=5)
    #    axs[i].set_title('p=' + str(p))
    #plt.savefig('poly_heatmap_'+name+'.jpg')
    #plt.show()

def c(name):
    SPLIT = 0.8
    LAMBDA = 0.001
    ps = np.arange(1, 17, 1)

    X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

    #train_loss = np.zeros(ps.shape[0])
    #valid_loss = np.zeros(ps.shape[0])
    #for p in ps:
    #    alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='poly', parameter=p)
    #    train_loss[p-1] = kloss(X_train, y_train, alpha, X_train, kernel='poly', parameter=p)
    #    valid_loss[p-1] = kloss(X_valid, y_valid, alpha, X_train, kernel='poly', parameter=p)
    #print('for p = {} the training loss is {} and validation loss is {}'.format(ps, train_loss,
valid_loss))
    #plt.semilogy(ps, train_loss, label = 'training loss')
    #plt.semilogy(ps, valid_loss, label = 'valid loss')
    #plt.legend()
    #plt.title('poly_kernel_loss_'+name)
    #plt.savefig('poly_kernel_loss_'+name+'.jpg')
    #plt.show()

    ps = np.arange(2, 13, 2)
    fig, axs = plt.subplots(1, ps.shape[0], figsize = (ps.shape[0]*5, 4))
    for i, p in enumerate(ps):
        alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='poly', parameter=p)
        f = lambda x, y: kernel_matrix(np.hstack((x.reshape(x.shape[0], 1), y.reshape(y.shape[0],
1))),\
```

```
                                X_train, kernel='poly', parameter=p) @ alpha
            hb, ax = heatmap(X, y, axs[i], f, clip=0)
            axs[i].set_title('p=' + str(p))
            fig.colorbar(hb, ax=ax)
        plt.savefig('poly_kernel_heatmap_noclip_'+name+'.jpg')
        plt.show()

    def c_part2(name):
        SPLIT = 0.15
        LAMBDA = 0.001
        ps = np.arange(1, 25, 1)

        X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

        train_loss = np.zeros(ps.shape[0])
        valid_loss = np.zeros(ps.shape[0])
        for p in ps:
            alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='poly', parameter=p)
            train_loss[p-1] = kloss(X_train, y_train, alpha, X_train, kernel='poly', parameter=p)
            valid_loss[p-1] = kloss(X_valid, y_valid, alpha, X_train, kernel='poly', parameter=p)
        print('for p = {} the training loss is {} and validation loss is {}'.format(ps, train_loss, valid_loss))
        plt.semilogy(ps, train_loss, label = 'training loss')
        plt.semilogy(ps, valid_loss, label = 'valid loss')
        plt.legend()
        plt.title('few_training_poly_kernel_loss_'+name)
        plt.savefig('few_training_poly_kernel_loss_'+name+'.jpg')
        plt.show()

    def d(name):
        SPLIT = 0.8
        LAMBDAS = [0.0001, 0.001, 0.01]
        ps = [5, 6]

        X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)


        for j, p in enumerate(ps):
            for i, LAMBDA in enumerate(LAMBDAS):
                ns = np.arange(100, X_train.shape[0], 100)
                valid_loss = np.zeros(ns.shape[0])
                for k, n in enumerate(ns):
                    loss = np.zeros(10)
                    for h in range(10):
                        sample_index = np.random.choice(X_train.shape[0], size=n, replace=False)
```

```
            X_train_sample = X_train[sample_index, :]
            y_train_sample = y_train[sample_index]
            w = lstsq(assemble_feature(X_train_sample, p), y_train_sample, LAMBDA)

            loss[h] = ploss(assemble_feature(X_valid, p), y_valid, w)

        valid_loss[k] = np.mean(loss)

    plt.semilogx(ns, valid_loss, label = 'lambda = {}, p = {}'.format(LAMBDA, p))
  plt.legend()
  plt.title('poly_loss_vs_n_'+name)
  plt.savefig('poly_loss_vs_n_'+name+'.jpg')
  plt.show()

def e(name):
  SPLIT = 0.8
  LAMBDA = 0.001
  sigmas = [10, 3, 1, 0.3, 0.1, 0.03]

  X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

  #train_loss = np.zeros(len(sigmas))
  #valid_loss = np.zeros(len(sigmas))
  #for i, sigma in enumerate(sigmas):
  #   alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='rbf', parameter=sigma)
  #   train_loss[i] = kloss(X_train, y_train, alpha, X_train, kernel='rbf', parameter=sigma)
  #   valid_loss[i] = kloss(X_valid, y_valid, alpha, X_train, kernel='rbf', parameter=sigma)
  #print('for sigma = {} the training loss is {} and validation loss is {}'.format(sigmas, train_loss,
valid_loss))
  #plt.loglog(sigmas, train_loss, label = 'training loss')
  #plt.loglog(sigmas, valid_loss, label = 'valid loss')
  #plt.legend()
  #plt.title('rbf_kernel_loss_'+name)
  #plt.savefig('rbf_kernel_loss_'+name+'.jpg')
  #plt.show()

  fig, axs = plt.subplots(1, len(sigmas), figsize = (len(sigmas)*5, 4))
  for i, sigma in enumerate(sigmas):
    alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='rbf', parameter=sigma)
    f = lambda x, y: kernel_matrix(np.hstack((x.reshape(x.shape[0], 1), y.reshape(y.shape[0],
1))),\
                          X_train, kernel='rbf', parameter=sigma) @ alpha
    hb, ax = heatmap(X, y, axs[i], f, clip=0)
    axs[i].set_title('sigma=' + str(sigma))
```

```
        fig.colorbar(hb, ax=ax)
    plt.savefig('rbf_kernel_heatmap_no_clip_'+name+'.jpg')
    plt.show()

if __name__ == "__main__":
    #a()

    #b('circle')
    #b('heart')
    #b('asymmetric')

    #c('circle')
    #c('heart')
    #c_part2('circle')

    #d('asymmetric')
    e('heart')
```