```
# for questions #2-10

import numpy as np
from sklearn import svm
from sklearn.model_selection import KFold

def importData(filename):
    data = open(filename, 'r')
    lines = data.read().splitlines()

    matx = np.zeros(shape=(len(lines),3))
    vecty = np.zeros(shape=(len(lines),1))

    for i in range(len(lines)):
        point = lines[i].strip(' ').split('  ')
        matx[i] = np.matrix([1, float(point[1]), float(point[2])])
        vecty[i] = float(point[0])

    return matx, vecty

# change the y vectors to one_v_all binary classifiers
def one_v_all(vecty, digit):
    for i in range(len(vecty)):
        if vecty[i] == digit:
            vecty[i] = 1
        else:
            vecty[i] = -1

    return vecty

# 2-3
# input parameter digit represents the one digit with class +1 for
# one-versus-all binary classifier
def poly_kernel(digit):
    matx, vecty = importData('features.train.txt')
    newVecty = one_v_all(vecty, digit)

    clf = svm.SVC(C=0.01, kernel='poly', degree=2, gamma=1, coef0=1)
    clf.fit(matx, newVecty.ravel())
    disagreement_svm = 1 - clf.score(matx, newVecty.ravel())

    return disagreement_svm

# 2
# for i in range(0, 9, 2):
#     print(poly_kernel(i))

# 3
# for i in range(1, 10, 2):
#     print(poly_kernel(i))


# 4
# input parameter digit represents the one digit with class +1 for
# one-versus-all binary classifier
def num_SV(digit):
    matx, vecty = importData('features.train.txt')
    newVecty = one_v_all(vecty, digit)

    clf = svm.SVC(C=0.01, kernel='poly', degree=2, gamma=1, coef0=1)
    clf.fit(matx, newVecty.ravel())
    return len(clf.support_vectors_)

# print(abs(num_SV(0) - num_SV(1)))
```

```
# 5-6
# generate a new dataset reflecting one-versus-one binary classifier
def one_v_one(dataset, d1, d2):
    matx, vecty = dataset
    newMatx = np.empty((0, 3))
    newVecty = np.empty(0)
    for i in range(len(vecty)):
        if vecty[i] == d1:
            newMatx = np.append(newMatx, [matx[i]], 0)
            newVecty = np.append(newVecty, 1)
        elif vecty[i] == d2:
            newMatx = np.append(newMatx, [matx[i]], 0)
            newVecty = np.append(newVecty, -1)
    return newMatx, newVecty

def poly_kernel_2(q, c):
    matx, vecty = one_v_one(importData('features.train.txt'), 1, 5)
    testMatx, testVecty = one_v_one(importData('features.test.txt'), 1, 5)

    clf = svm.SVC(C=c, kernel='poly', degree=q, gamma=1, coef0=1)
    clf.fit(matx, vecty)

    e_in = 1 - clf.score(matx, vecty)
    e_out = 1 - clf.score(testMatx, testVecty)
    num_sv = len(clf.support_vectors_)
    print('Q =', q, 'C =', c)
    print('E_in:', e_in)
    print('E_out:', e_out)
    print('num_sv:', num_sv, '\n')

# 5
# for c in [0.001, 0.01, 0.1, 1]:
#     poly_kernel_2(2, c)

# 6
# for c in [0.0001, 0.001, 0.01, 1]:
#     poly_kernel_2(2, c)
#     poly_kernel_2(5, c)


# 7-8
def cross_val(c):
    matx, vecty = one_v_one(importData('features.train.txt'), 1, 5)
    kf = KFold(n_splits=10, shuffle=True)

    total_e_n = 0
    for train_index, test_index in kf.split(matx):
        x_train, x_test = matx[train_index], matx[test_index]
        y_train, y_test = vecty[train_index], vecty[test_index]
        clf = svm.SVC(C=c, kernel='poly', degree=2, gamma=1, coef0=1)
        clf.fit(x_train, y_train)
        total_e_n += 1 - clf.score(x_test, y_test)

    return total_e_n / 10

# 7
# scores = {0.0001: 0, 0.001: 0, 0.01: 0, 0.1: 0, 1: 0}
# for i in range(100):
#     minE_cv = 2
#     for c in [0.0001, 0.001, 0.01, 0.1, 1]:
#         e_cv = cross_val(c)
#         if e_cv < minE_cv:
#             minE_cv = e_cv
#             selectedC = c
#     scores[selectedC] += 1
# print(scores)
```

```
# 8
# totalE_cv = 0
# for i in range(100):
#     totalE_cv += cross_val(0.001)
# print(totalE_cv / 100)


# 9-10
def rbf_kernel(c):
    matx, vecty = one_v_one(importData('features.train.txt'), 1, 5)
    testMatx, testVecty = one_v_one(importData('features.test.txt'), 1, 5)

    clf = svm.SVC(C=c, kernel='rbf', gamma=1)
    clf.fit(matx, vecty)

    e_in = 1 - clf.score(matx, vecty)
    e_out = 1 - clf.score(testMatx, testVecty)
    print('C =', c)
    print('E_in:', e_in)
    print('E_out:', e_out, '\n')

# for c in [0.01, 1, 100, 10 ** 4, 10 ** 6]:
#     rbf_kernel(c)
```