```python
# for questions #13-18

import math
import numpy as np
import random
from sklearn import svm

# input param n is the number of points in the dataset
def createData(n):
    matx = np.zeros(shape=(n,3))
    vecty = np.zeros(shape=(n,1))
    for i in range(0, n):
        x = np.matrix([1, random.uniform(-1, 1), random.uniform(-1, 1)])
        matx[i] = x
        vecty[i] = np.sign(x[0,2] - x[0,1] + 0.25 * math.sin(math.pi * x[0,1]))

    return matx, vecty

def non_sep_rbf_kernel():
    matx, vecty = createData(100)

    # large c means approximately hard margin
    clf = svm.SVC(C=100000000, kernel='rbf', gamma=1.5)
    clf.fit(matx, vecty.ravel())

    e_in = 1 - clf.score(matx, vecty.ravel())
    return (e_in != 0)

# 13
# nonSepCount = 0
# for i in range(1000):
#     nonSepCount += non_sep_rbf_kernel()
# print(nonSepCount/1000)


# calculate error given dataset, centers and their weights, and gamma
def calculateError(dataset, centers, w, gamma):
    matx, vecty = dataset

    errorCount = 0
    for i in range(len(matx)):
        x = matx[i, 1:]
        rbfs = ([math.exp(-gamma * np.linalg.norm(x - centers[j]) ** 2)
                 for j in range(len(centers))])
        output = np.matrix([1] + rbfs).dot(w)
        if np.sign(output) != np.sign(vecty[i]):
            errorCount += 1

    return errorCount / len(matx)


def rbf_kernel():
    matx, vecty = createData(100)

    # large c means approximately hard margin
    clf = svm.SVC(C=100000000, kernel='rbf', gamma=1.5)
    clf.fit(matx, vecty.ravel())

    e_in = 1 - clf.score(matx, vecty.ravel())

    # discard the run and repeat if dataset is not separable in z space
    if e_in != 0:
        return rbf_kernel()

    testMatx, testVecty = createData(1000)
    e_out = 1 - clf.score(testMatx, testVecty.ravel())
    return (e_in, e_out)


# regular rbf classification
# input param k is number of cluster centers
def lloyd_pinv(k, gamma):
    dataset = createData(100)
    matx, vecty = dataset

    # initialize centers
    centers = [np.matrix([random.uniform(-1, 1), random.uniform(-1, 1)])
        for i in range(k)]
    # print(matx)
    # print(matx[[-1, -2, -4]])
    # print(matx[[-1, -2, -4]].mean(0))
    # x = matx[0, 1:]

    while True:
        # contains indices of points for each cluster
        clusters = [[] for i in range(k)]

        # go through each point in the dataset to put them in clusters
        for i in range(len(matx)):
            x = matx[i, 1:]

            # find closest center
            closestCenterNorm = 2
            for j in range(k):
                # Euclidean distance from point to current center
                dist = np.linalg.norm(x - centers[j])
                if (dist < closestCenterNorm):
                    closestCenterNorm = dist
                    closestCenter = j
            clusters[closestCenter].append(i)

        # discard and repeat the run if any cluster is empty
        if any(clusters[i] == [] for i in range(k)):
            return lloyd_pinv(k, gamma)

        oldCenters = centers.copy()

        # find new center for each cluster, i is cluster index
        for i in range(k):
            #print('new cluster')
            #print(matx[clusters[i], 1:])
            centers[i] = matx[clusters[i], 1:].mean(0)

        # check convergence
        if all((oldCenters[i] == centers[i]).all() for i in range(k)):
            break

    # choose weights
    # phi matrix of rbf applied to each point and each center
    phi = np.zeros(shape=(len(matx), k+1))

    for i in range(len(matx)):
        x = matx[i, 1:]
        row = ([1] + [math.exp(-1 * gamma * np.linalg.norm(x - centers[j]) ** 2)
            for j in range(k)])
        phi[i] = row

    w = np.linalg.pinv(phi).dot(vecty)

    # calculate in-sample errorCount
    e_in = calculateError(dataset, centers, w, gamma)
    # calculate out-of-sample error
    testData = createData(1000)
    e_out = calculateError(testData, centers, w, gamma)

    return e_in, e_out


# 14
# beatCount = 0
# for i in range(100):
#     beatCount += (rbf_kernel()[1] < lloyd_pinv(9, 1.5)[1])
# print(beatCount / 100)


# 15
# beatCount = 0
# for i in range(100):
#     beatCount += (rbf_kernel()[1] < lloyd_pinv(12, 1.5)[1])
# print(beatCount / 100)


# 16
# dictionary of frequencies for 5 scenarios (a through e)
# freqs = {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'e': 0}
# for i in range(1000):
#     e_in9, e_out9 = lloyd_pinv(9, 1.5)
#     e_in12, e_out12 = lloyd_pinv(12, 1.5)
#     if e_in9 > e_in12 and e_out9 < e_out12:
#         freqs['a'] += 1
#     elif e_in9 < e_in12 and e_out9 > e_out12:
#         freqs['b'] += 1
#     elif e_in9 < e_in12 and e_out9 < e_out12:
#         freqs['c'] += 1
#     elif e_in9 > e_in12 and e_out9 > e_out12:
#         freqs['d'] += 1
#     elif e_in9 == e_in12 and e_out9 == e_out12:
#         freqs['e'] += 1
# print(freqs)


# 17
# dictionary of frequencies for 5 scenarios (a through e)
# freqs = {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'e': 0}
# for i in range(1000):
#     e_in1, e_out1 = lloyd_pinv(9, 1.5)
#     e_in2, e_out2 = lloyd_pinv(9, 2)
#     if e_in1 > e_in2 and e_out1 < e_out2:
#         freqs['a'] += 1
#     elif e_in1 < e_in2 and e_out1 > e_out2:
#         freqs['b'] += 1
#     elif e_in1 < e_in2 and e_out1 < e_out2:
#         freqs['c'] += 1
#     elif e_in1 > e_in2 and e_out1 > e_out2:
#         freqs['d'] += 1
#     elif e_in1 == e_in2 and e_out1 == e_out2:
#         freqs['e'] += 1
# print(freqs)


# 18
# count = 0
# for i in range(100):
#     if lloyd_pinv(9, 1.5)[0] == 0:
#         count += 1
# print(count / 100)
```