```
#for questions #8-10

import numpy as np
import random
from sklearn import svm

def createData(f, 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] = 1 if (x[0,2] > f(x[0,1])) else -1

    # if all data points are on one side of the line, discard the run and start
    # a new run
    if -1 not in vecty or 1 not in vecty:
        return createData(f, n)

    return matx, vecty


def setup(n):
    #Set up the target function f
    x = random.uniform(-1, 1), random.uniform(-1, 1)
    y = random.uniform(-1, 1), random.uniform(-1, 1)
    z = np.polyfit(x, y, 1)
    f = np.poly1d(z)

    #create data
    dataset = createData(f, n)

    return f, dataset


def calculateError(w, dataset):
    matz, vecty = dataset
    errorCount = 0
    for i in range(len(matz)):
        if np.sign(w.dot(matz[i, :])) != np.sign(vecty[i]):
            errorCount += 1

    return errorCount / len(matz)


#8-9
def plasvm(n):
    f, dataset = setup(n)
    matx, vecty = dataset
    w = np.matrix([0, 0, 0])
    misclassified = [0]
    while misclassified:
        misclassified = []
        for i in range(n):
            x = matx[i,:]
            y = vecty[i]
            if np.sign(w.dot(x.transpose())) != np.sign(y):
                misclassified.append(i)
        if misclassified:
            index = random.choice(misclassified)
            w = w + vecty[index] * matx[index,:]

    #calculate disagreement for pla
    testData = createData(f, 1000)
    disagreement_pla = calculateError(w, testData)
```

```
    #find g_svm and calculate disagreement
    clf = svm.SVC(kernel='linear', C=10000000)
    clf.fit(matx, vecty.ravel())
    testMatx, testMaty = testData
    disagreement_svm = 1 - clf.score(testMatx, testMaty.ravel())

    return disagreement_svm < disagreement_pla


def runplasvm(runCount, n):
    totalsvmbetter = 0
    for i in range(0, runCount):
        svmbetter = plasvm(n)
        totalsvmbetter += svmbetter
    return (totalsvmbetter / runCount)

#print(runplasvm(1000, 100))


#10
def numSV(n):
    f, dataset = setup(n)
    matx, vecty = dataset
    clf = svm.SVC(kernel='linear', C=10000000)
    clf.fit(matx, vecty.ravel())
    return len(clf.support_vectors_)

totalNum = 0
for i in range(1000):
    totalNum += numSV(100)
print(totalNum / 1000)
```