In [33]:
```
import random
from scipy import linalg
import numpy

def doNonLinRegRun(num_train, num_validate, train_first):
    #IMPORT POINTS
    trainstrs = []
    teststrs = []
    trainlines = open('in.dta.txt', 'r')
    testlines = open('out.dta.txt', 'r')
    for line in trainlines:
        trainstrs.append(line.split())
    for line in testlines:
        teststrs.append(line.split())

    trainpts = []
    testpts = []
    validatepts = []
    count = 0
    for stri in trainstrs:
        if train_first:
            if count < num_train:
                trainpts.append([float(stri[0]), float(stri[1]), float
(stri[2])])
            elif count < num_train + num_validate:
                validatepts.append([float(stri[0]), float(stri[1]), fl
oat(stri[2])])
        else:
            if count < num_train:
                validatepts.append([float(stri[0]), float(stri[1]), fl
oat(stri[2])])
            elif count < num_train + num_validate:
                trainpts.append([float(stri[0]), float(stri[1]), float
(stri[2])])
        count += 1
    for stri in teststrs:
        testpts.append([float(stri[0]), float(stri[1]), float(stri[2])
])

    #TRANSFORM POINTS
    newtrainpoints3 = []
    newtestpoints3 = []
    newvalidatepoints3 = []
    for pt in trainpts:
        newtrainpoints3.append([pt[0], pt[1], pt[0]**2, pt[2]])
    for pt in validatepts:
        newvalidatepoints3.append([pt[0], pt[1], pt[0]**2, pt[2]])
    for pt in testpts:
```

```
        newtestpoints3.append([pt[0], pt[1], pt[0]**2, pt[2]])

    newtrainpoints4 = []
    newtestpoints4 = []
    newvalidatepoints4 = []
    for pt in trainpts:
        newtrainpoints4.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[2
]])
    for pt in validatepts:
        newvalidatepoints4.append([pt[0], pt[1], pt[0]**2, pt[1]**2, p
t[2]])
    for pt in testpts:
        newtestpoints4.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[2]
])

    newtrainpoints5 = []
    newtestpoints5 = []
    newvalidatepoints5 = []
    for pt in trainpts:
        newtrainpoints5.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[0
]*pt[1], \
                                pt[2]])
    for pt in validatepts:
        newvalidatepoints5.append([pt[0], pt[1], pt[0]**2, pt[1]**2, p
t[0]*pt[1], \
                                pt[2]])
    for pt in testpts:
        newtestpoints5.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[0]
*pt[1], \
                               pt[2]])

    newtrainpoints6 = []
    newtestpoints6 = []
    newvalidatepoints6 = []
    for pt in trainpts:
        newtrainpoints6.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[0
]*pt[1], \
                                numpy.abs(pt[0] - pt[1]), pt[2]])
    for pt in validatepts:
        newvalidatepoints6.append([pt[0], pt[1], pt[0]**2, pt[1]**2, p
t[0]*pt[1], \
                                numpy.abs(pt[0] - pt[1]), pt[2]])
    for pt in testpts:
        newtestpoints6.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[0]
*pt[1], \
                               numpy.abs(pt[0] - pt[1]), pt[2]])

    newtrainpoints7 = []
    newtestpoints7 = []
    newvalidatepoints7 = []
```

```
    for pt in trainpts:
        newtrainpoints7.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[0
]*pt[1], \
                                numpy.abs(pt[0] - pt[1]), numpy.abs(pt[
1] + pt[0]), pt[2]])
    for pt in validatepts:
        newvalidatepoints7.append([pt[0], pt[1], pt[0]**2, pt[1]**2, p
t[0]*pt[1], \
                                numpy.abs(pt[0] - pt[1]), numpy.abs(pt[
1] + pt[0]), pt[2]])
    for pt in testpts:
        newtestpoints7.append([pt[0], pt[1], pt[0]**2, pt[1]**2, pt[0]
*pt[1], \
                               numpy.abs(pt[0] - pt[1]), numpy.abs(pt[1
] + pt[0]), pt[2]])

    #APPLY ALGORITHM
    xmat = []
    ymat = []
    for pt in newtrainpoints3:
        xmat.append([1, pt[0], pt[1], pt[2]])
        ymat.append(pt[3])
    xmat = numpy.asarray(xmat)
    ymat = numpy.asarray(ymat)
    pseudoinverse = linalg.pinv(xmat)
    weight3 = numpy.matmul(pseudoinverse, ymat)

    xmat = []
    ymat = []
    for pt in newtrainpoints4:
        xmat.append([1, pt[0], pt[1], pt[2], pt[3]])
        ymat.append(pt[4])
    xmat = numpy.asarray(xmat)
    ymat = numpy.asarray(ymat)
    pseudoinverse = linalg.pinv(xmat)
    weight4 = numpy.matmul(pseudoinverse, ymat)

    xmat = []
    ymat = []
    for pt in newtrainpoints5:
        xmat.append([1, pt[0], pt[1], pt[2], pt[3], pt[4]])
        ymat.append(pt[5])
    xmat = numpy.asarray(xmat)
    ymat = numpy.asarray(ymat)
    pseudoinverse = linalg.pinv(xmat)
    weight5 = numpy.matmul(pseudoinverse, ymat)

    xmat = []
    ymat = []
    for pt in newtrainpoints6:
```

```
        xmat.append([1, pt[0], pt[1], pt[2], pt[3], pt[4], pt[5]])
        ymat.append(pt[6])
    xmat = numpy.asarray(xmat)
    ymat = numpy.asarray(ymat)
    pseudoinverse = linalg.pinv(xmat)
    weight6 = numpy.matmul(pseudoinverse, ymat)

    xmat = []
    ymat = []
    for pt in newtrainpoints7:
        xmat.append([1, pt[0], pt[1], pt[2], pt[3], pt[4], pt[5], pt[6
]])
        ymat.append(pt[7])
    xmat = numpy.asarray(xmat)
    ymat = numpy.asarray(ymat)
    pseudoinverse = linalg.pinv(xmat)
    weight7 = numpy.matmul(pseudoinverse, ymat)

    #COMPUTE IN-SAMPLE ERROR
    #wrongtraincount = 0
    #for point in newtrainpoints:
    #    if numpy.sign(weight[0] + weight[1]*point[0] + weight[2]*poin
t[1] + weight[3]*point[2] + \
    #                  weight[4]*point[3] + weight[5]*point[4] + weigh
t[6]*point[5] + weight[7]*point[6]) != point[7]:
    #        wrongtraincount += 1
    #wrongtraincount /= len(newtrainpoints)

    #COMPUTE OUT-OF-SAMPLE ERROR
    wrongtestcount3 = 0
    for pt in newtestpoints3:
        if numpy.sign(weight3[0] + weight3[1]*pt[0] + weight3[2]*pt[1]
+ weight4[3]*pt[2]) != pt[3]:
            wrongtestcount3 += 1
    wrongtestcount3 /= len(newtestpoints3)

    wrongtestcount4 = 0
    for pt in newtestpoints4:
        if numpy.sign(weight4[0] + weight4[1]*pt[0] + weight4[2]*pt[1]
+ weight4[3]*pt[2] + \
                      weight5[4]*pt[3]) != pt[4]:
            wrongtestcount4 += 1
    wrongtestcount4 /= len(newtestpoints4)

    wrongtestcount5 = 0
    for pt in newtestpoints5:
        if numpy.sign(weight5[0] + weight5[1]*pt[0] + weight5[2]*pt[1]
+ weight5[3]*pt[2] + \
                      weight5[4]*pt[3] + weight6[5]*pt[4]) != pt[5]:
            wrongtestcount5 += 1
```

```
    wrongtestcount5 /= len(newtestpoints5)

    wrongtestcount6 = 0
    for pt in newtestpoints6:
        if numpy.sign(weight6[0] + weight6[1]*pt[0] + weight6[2]*pt[1]
+ weight6[3]*pt[2] + \
                      weight6[4]*pt[3] + weight6[5]*pt[4] + weight7[6]
*pt[5]) != pt[6]:
            wrongtestcount6 += 1
    wrongtestcount6 /= len(newtestpoints6)

    wrongtestcount7 = 0
    for pt in newtestpoints7:
        if numpy.sign(weight7[0] + weight7[1]*pt[0] + weight7[2]*pt[1]
+ weight7[3]*pt[2] + \
                      weight7[4]*pt[3] + weight7[5]*pt[4] + weight7[6]
*pt[5] + weight7[7]*pt[6]) != pt[7]:
            wrongtestcount7 += 1
    wrongtestcount7 /= len(newtestpoints7)

    #COMPUTE VALIDATION ERROR
    wrongvalidatecount3 = 0
    for pt in newvalidatepoints3:
        if numpy.sign(weight3[0] + weight3[1]*pt[0] + weight3[2]*pt[1]
+ weight4[2]*pt[1]) != pt[3]:
            wrongvalidatecount3 += 1
    wrongvalidatecount3 /= len(newvalidatepoints3)

    wrongvalidatecount4 = 0
    for pt in newvalidatepoints4:
        if numpy.sign(weight4[0] + weight4[1]*pt[0] + weight4[2]*pt[1]
+ weight4[3]*pt[2] + \
                      weight5[4]*pt[3]) != pt[4]:
            wrongvalidatecount4 += 1
    wrongvalidatecount4 /= len(newvalidatepoints4)

    wrongvalidatecount5 = 0
    for pt in newvalidatepoints5:
        if numpy.sign(weight5[0] + weight5[1]*pt[0] + weight5[2]*pt[1]
+ weight5[3]*pt[2] + \
                      weight5[4]*pt[3] + weight6[5]*pt[4]) != pt[5]:
            wrongvalidatecount5 += 1
    wrongvalidatecount5 /= len(newvalidatepoints5)

    wrongvalidatecount6 = 0
    for pt in newvalidatepoints6:
        if numpy.sign(weight6[0] + weight6[1]*pt[0] + weight6[2]*pt[1]
+ weight6[3]*pt[2] + \
                      weight6[4]*pt[3] + weight6[5]*pt[4] + weight7[6]
*pt[5]) != pt[6]:
```

```
            wrongvalidatecount6 += 1
    wrongvalidatecount6 /= len(newvalidatepoints6)

    wrongvalidatecount7 = 0
    for pt in newvalidatepoints7:
        if numpy.sign(weight7[0] + weight7[1]*pt[0] + weight7[2]*pt[1]
+ weight7[3]*pt[2] + \
                      weight7[4]*pt[3] + weight7[5]*pt[4] + weight7[6]
*pt[5] + weight7[7]*pt[6]) != pt[7]:
            wrongvalidatecount7 += 1
    wrongvalidatecount7 /= len(newvalidatepoints7)

    return [wrongvalidatecount3, wrongvalidatecount4, wrongvalidatecou
nt5, wrongvalidatecount6, wrongvalidatecount7, \
            wrongtestcount3, wrongtestcount4, wrongtestcount5, wrongte
stcount6, wrongtestcount7]
```

In [34]:
```
result = doNonLinRegRun(25, 10, True);
print('Validation error for k = 3: ' + str(result[0]));
print('Validation error for k = 4: ' + str(result[1]));
print('Validation error for k = 5: ' + str(result[2]));
print('Validation error for k = 6: ' + str(result[3]));
print('Validation error for k = 7: ' + str(result[4]));

print('Out-of-sample error for k = 3: ' + str(result[5]));
print('Out-of-sample error for k = 4: ' + str(result[6]));
print('Out-of-sample error for k = 5: ' + str(result[7]));
print('Out-of-sample error for k = 6: ' + str(result[8]));
print('Out-of-sample error for k = 7: ' + str(result[9]));
```

```
Validation error for k = 3: 0.3
Validation error for k = 4: 0.4
Validation error for k = 5: 0.8
Validation error for k = 6: 0.0
Validation error for k = 7: 0.1
Out-of-sample error for k = 3: 0.44
Out-of-sample error for k = 4: 0.412
Out-of-sample error for k = 5: 0.72
Out-of-sample error for k = 6: 0.088
Out-of-sample error for k = 7: 0.072
```

Problem 1: We can see that the lowest validation error is for k = 6, so the answer is D. Problem 2: We can see that the lowest out-of-sample error is for k = 7, so the answer is E.

In [35]:
```
result = doNonLinRegRun(25, 10, False);
print('Validation error for k = 3: ' + str(result[0]));
print('Validation error for k = 4: ' + str(result[1]));
print('Validation error for k = 5: ' + str(result[2]));
print('Validation error for k = 6: ' + str(result[3]));
print('Validation error for k = 7: ' + str(result[4]));

print('Out-of-sample error for k = 3: ' + str(result[5]));
print('Out-of-sample error for k = 4: ' + str(result[6]));
print('Out-of-sample error for k = 5: ' + str(result[7]));
print('Out-of-sample error for k = 6: ' + str(result[8]));
print('Out-of-sample error for k = 7: ' + str(result[9]));
```

```
Validation error for k = 3: 0.56
Validation error for k = 4: 0.6
Validation error for k = 5: 0.68
Validation error for k = 6: 0.08
Validation error for k = 7: 0.12
Out-of-sample error for k = 3: 0.384
Out-of-sample error for k = 4: 0.488
Out-of-sample error for k = 5: 0.656
Out-of-sample error for k = 6: 0.192
Out-of-sample error for k = 7: 0.196
```

Problem 3: We can see that the lowest validation error is for k = 6, so the answer is D. Problem 4: We can see that the lowest out-of-sample error is for k = 6, so the answer is D.

Problem 5: We can see that the out-of-sample errors corresponding to k=6 and k=6 for problems 1 and 3 respectively are 0.088 and 0.192 respectively, so (by obervation) the closest values in euclidean distance are 0.1 and 0.2 so the answer is B.

In [39]:
```
avg1 = 0
avg2 = 0
avgmin = 0
for a in range(1000000):
    pt1 = random.uniform(0,1)
    pt2 = random.uniform(0,1)
    avg1 += pt1
    avg2 += pt2
    avgmin += numpy.minimum(pt1, pt2)
avg1 /= 1000000
avg2 /= 1000000
avgmin /= 1000000
print('Average e_1: ' + str(avg1));
print('Average e_2: ' + str(avg1));
print('Average min e_1, e_2: ' + str(avgmin));
```

```
Average e_1: 0.4996145309094419
Average e_2: 0.4996145309094419
Average min e_1, e_2: 0.3331019692313675
```

Problem 6: We can see that e_1 and e_2 are closest to 0.5 and the minimum of e_1 and e_2 is closest to 0.4. Therefore the answer is D.

In [65]:
```
from sklearn import svm
import random
import numpy

def isPointPositive(point, line):
    result = (point[0] * line[0]) + (point[1] * line[1]) + line[2]
    if result == 0:
        return 3
    elif result > 0:
        return True
    else:
        return False

def doRun(NUMPOINTS):
    centerpt = [random.uniform(-1,1), random.uniform(-1,1)]
    otherpt = [random.uniform(-1,1), random.uniform(-1,1)]

    goodset = False
    points = []
    while not goodset:
        for a in range(NUMPOINTS):
            points.append([random.uniform(-1,1), random.uniform(-1,1)]
)
```

```
        linevect = [otherpt[0] - centerpt[0], otherpt[1] - centerpt[1]
, 0]
        origlinevect = [linevect[0], linevect[1], 0]

        havepos = False
        haveneg = False
        for point in points:
            if point[1] > (otherpt[1]-centerpt[1])/(otherpt[0]-centerp
t[0])*(point[0]-centerpt[0]):
                point.append(1)
                havepos = True
            else:
                point.append(-1)
                haveneg = True
            if havepos and haveneg:
                goodset = True

    linevect = [0,0,0]
    counter = 0
    for a in range(1000):
        counter += 1
        misclassified = []
        for point in points:
            score = isPointPositive(point, linevect)
            if score == 3:
                misclassified.append(point)
            elif score:
                if point[2] == -1:
                    misclassified.append(point)
            elif not score:
                if point[2] == 1:
                    misclassified.append(point)
            else:
                misclassified.append(point)

        #print("Iteration number " + str(counter))
        #print("Number of misclassified points: " + str(len(misclassif
ied)))

        if len(misclassified) == 0:
            break

        targetpoint = random.choice(misclassified)

        linevect[0] += targetpoint[2]*targetpoint[0]
        linevect[1] += targetpoint[2]*targetpoint[1]
        linevect[2] += targetpoint[2]

    #print(counter)
```

```
    wrongcount = 0
    testpoints = []
    for a in range(1000):
        point1 = [random.uniform(-1,1), random.uniform(-1,1)]
        testpoints.append(point1)
        if point1[1] > (otherpt[1]-centerpt[1])/(otherpt[0]-centerpt[0
])*(point1[0]-centerpt[0]):
            point1.append(1)
        else:
            point1.append(-1)
        score = isPointPositive(point1, linevect)
        if score == True and point1[2] == -1:
            wrongcount+=1
        elif score == False and point1[2] == 1:
            wrongcount+=1

    #print(wrongcount/1000.0)
    return [counter, wrongcount/1000.0, testpoints, points]
```

In [68]:
```
from sklearn import svm
import random
import numpy

NUM_RUNS = 1000
N = 10
avg_vectors = 0
SVM_better = 0
for a in range(NUM_RUNS):
    PLA_result = doRun(N)
    testpoints = PLA_result[2]
    trainpoints = PLA_result[3]

    inputs = []
    outputs = []
    for point in trainpoints:
        inputs.append([point[0], point[1]])
        outputs.append(point[2])

    svc = svm.SVC(kernel = 'linear', C = 99999999999999999999999999999999999999)
    svc.fit(inputs, outputs)

    avg_vectors += len(svc.support_)

    misclassifiedct = 0
    for pt in testpoints:
        if svc.predict([[pt[0], pt[1]]]) != pt[2]:
            misclassifiedct += 1

    if PLA_result[1] > misclassifiedct/len(testpoints):
        SVM_better += 1

print('How often SVM is better than PLA (N=10): '+ str(SVM_better/NUM_RUNS))
print('Average number of support vectors (N=10): ' + str(avg_vectors/NUM_RUNS))
```

```
How often SVM is better than PLA (N=10): 0.602
Average number of support vectors (N=10): 2.848
```

Problem 8: SVM is better than PLA is closest to 60%, so the answer to question 8 is C.

In [69]:
```
from sklearn import svm
import random
import numpy

NUM_RUNS = 1000
N = 100
avg_vectors = 0
SVM_better = 0
for a in range(NUM_RUNS):
    PLA_result = doRun(N)
    testpoints = PLA_result[2]
    trainpoints = PLA_result[3]

    inputs = []
    outputs = []
    for point in trainpoints:
        inputs.append([point[0], point[1]])
        outputs.append(point[2])

    svc = svm.SVC(kernel = 'linear', C = 99999999999999999999999999999999999999)
    svc.fit(inputs, outputs)

    avg_vectors += len(svc.support_)

    misclassifiedct = 0
    for pt in testpoints:
        if svc.predict([[pt[0], pt[1]]]) != pt[2]:
            misclassifiedct += 1

    if PLA_result[1] > misclassifiedct/len(testpoints):
        SVM_better += 1

print('How often SVM is better than PLA (N=100): '+ str(SVM_better/NUM_RUNS))
print('Average number of support vectors (N=100): ' + str(avg_vectors/NUM_RUNS))
```

```
How often SVM is better than PLA (N=100): 0.61
Average number of support vectors (N=100): 3.0
```

Problem 9: SVM is better than PLA closest to 65% of the time, so the answer to question 9 is D. Problem 10: The average number of support vectors on average is closest to 3, so the answer to question 10 is B.

In [ ]: