In [2]:
```
from sklearn import svm
import random
import numpy

def formatOneToOne2(points, digit1, digit2):
    formattedInputs = []
    formattedOutputs = []

    for point in points:
        if point[0] == digit1:
            formattedInputs.append([point[1], point[2]])
            formattedOutputs.append(1)
        elif point[0] == digit2:
            formattedInputs.append([point[1], point[2]])
            formattedOutputs.append(-1)

    return [formattedInputs, formattedOutputs]

def formatOneToOne(points, digit1, digit2):
    formatted = []

    for point in points:
        if point[0] == digit1:
            formatted.append([point[1], point[2], 1])
        elif point[0] == digit2:
            formatted.append([point[1], point[2], -1])

    return formatted

def formatOneToMany2(points, digit):
    formattedInputs = []
    formattedOutputs = []

    for point in points:
        if point[0] == digit:
            formattedInputs.append([point[1], point[2]])
            formattedOutputs.append(1)
        else:
            formattedInputs.append([point[1], point[2]])
            formattedOutputs.append(-1)

    return [formattedInputs, formattedOutputs]

def formatOneToMany(points, digit):
    formatted = []

    for point in points:
        if point[0] == digit:
```

```
            formatted.append([point[1], point[2], 1])
        else:
            formatted.append([point[1], point[2], -1])

    return formatted;
```

In [1]:
```
# PROBLEM 12

def getSoftMarginSVMVectorNumber(trainInputs, trainOutputs, c, Q):
    tool = svm.SVC(kernel='poly', C=c, degree=Q, gamma=1, coef0=1)
    tool.fit(trainInputs, trainOutputs)
    return len(tool.support_)
```

In [3]:
```
# PROBLEM 12

#GET DATA
trainInputs = [[1,0],[0,1],[0,-1],[-1,0],[0,2],[0,-2],[-2,0]]
trainOutputs = [-1,-1,-1,1,1,1,1]

print('Number of support vectors: ')
print(getSoftMarginSVMVectorNumber(trainInputs, trainOutputs, float('i
nf'), 2))
```

```
Number of support vectors:
5
```

The number of support vectors is between 4 and 5, so the answer to question 12 is C.

In [28]:
```
# PROBLEMS 13-18
import random
import numpy
import sklearn
from sklearn.cluster import KMeans

def createLabeledPoints(numPoints):
    labeled = []
    for i in range(numPoints):
        x1 = random.uniform(-1,1)
        x2 = random.uniform(-1,1)
        out = numpy.sign(x2-x1+0.25*numpy.sin(numpy.pi*x1))
        labeled.append([x1,x2,out])
    return labeled

def doRBFKernel(trainInputs, trainOutputs, testInputs, testOutputs, ga
mmy):
    tool = svm.SVC(kernel='rbf', C=float('inf'), gamma=gammy)
    tool.fit(trainInputs, trainOutputs)
```

```
    trainErrorCount = 0
    testErrorCount = 0
    for i in range(len(trainInputs)):
        if tool.predict([trainInputs[i]]) != trainOutputs[i]:
            trainErrorCount += 1
    for i in range(len(testInputs)):
        if tool.predict([testInputs[i]]) != testOutputs[i]:
            testErrorCount += 1
    return [trainErrorCount/len(trainInputs), testErrorCount/len(testI
nputs)]

def doRBFRegular(trainInputs, trainOutputs, testInputs, testOutputs, g
ammy, K):
    #GET CLUSTERS
    cloister = KMeans(n_clusters=K).fit(trainInputs)
    centers = cloister.cluster_centers_
    #GET PHI MATRIX FROM TRAINING INPUTS
    phi = []
    for point in trainInputs:
        row = []
        for center in centers:
            row.append(numpy.exp((-gammy)*(numpy.linalg.norm(point-cen
ter)**2)))
        phi.append(row)
    #CALCULATE WEIGHTS
    intermediary = numpy.linalg.inv(numpy.matmul(numpy.transpose(phi),
phi))
    w = numpy.matmul(numpy.matmul(intermediary, numpy.transpose(phi)),
trainOutputs)
    #GET PHI_2 MATRIX FROM TEST INPUTS
    phi_2 = []
    for point in testInputs:
        row = []
        for center in centers:
            row.append(numpy.exp((-gammy)*(numpy.linalg.norm(point-cen
ter)**2)))
        phi_2.append(row)
    #CALCULATE E_IN
    wrongcount = 0
    for index in range(len(phi)):
        if numpy.sign(numpy.dot(phi[index],w)) != trainOutputs[index]:
            wrongcount += 1
    e_in = wrongcount / len(phi)
    #CALCULATE E_OUT
    wrongcount = 0
    for index in range(len(phi_2)):
        if numpy.sign(numpy.dot(phi_2[index],w)) != testOutputs[index]
:
            wrongcount += 1
    e_out = wrongcount / len(phi_2)
```

```
    return [e_in, e_out]
```

In [22]:
```
# PROBLEM 13
RUNS = 1000

notsepcount = 0
for i in range(RUNS):
    points = createLabeledPoints(100)
    points_in = []
    points_out = []
    for point in points:
        points_in.append([point[0], point[1]])
        points_out.append(point[2])
    result = doRBFKernel(points_in, points_out, [[0,0]], [1], 1.5)
    if result[0] != 0:
        notsepcount += 1
notsepprop = notsepcount/RUNS
print(notsepprop)
```

```
0.0
```

We can see that E_in is nonzero 0% of the time when gamma = 1.5, therefore the answer to question 13 is A.

In [30]:
```
# PROBLEM 14
RUNS = 500

kerbettercount = 0
for i in range(RUNS):
    points = createLabeledPoints(100)
    points_in = []
    points_out = []
    for point in points:
        points_in.append([point[0], point[1]])
        points_out.append(point[2])
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_ker = doRBFKernel(points_in, points_out, OUT_points_in, OUT
_points_out, 1.5)
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_reg = doRBFRegular(points_in, points_out, OUT_points_in, OU
T_points_out, 1.5, 9)
    if result_ker[1] < result_reg[1]:
        kerbettercount += 1

kerbetterprop = kerbettercount/500
print(kerbetterprop)
```

```
0.91
```

We have that the kernel form beats the regular form in terms of E_out over 75% of the time, so the answer to question 14 is E.

In [31]:
```
# PROBLEM 15
RUNS = 500

kerbettercount = 0
for i in range(RUNS):
    points = createLabeledPoints(100)
    points_in = []
    points_out = []
    for point in points:
        points_in.append([point[0], point[1]])
        points_out.append(point[2])
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_ker = doRBFKernel(points_in, points_out, OUT_points_in, OUT
_points_out, 1.5)
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_reg = doRBFRegular(points_in, points_out, OUT_points_in, OU
T_points_out, 1.5, 12)
    if result_ker[1] < result_reg[1]:
        kerbettercount += 1

kerbetterprop = kerbettercount/500
print(kerbetterprop)
```

```
0.76
```

We have that the kernel beats the regular form between 60% and 90% of the time, so the answer to question 15 is D.

In [33]:
```
# PROBLEM 16
RUNS = 500

counts = [0,0,0,0,0]
for i in range(RUNS):
    points = createLabeledPoints(100)
    points_in = []
    points_out = []
    for point in points:
        points_in.append([point[0], point[1]])
        points_out.append(point[2])
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_reg_9 = doRBFRegular(points_in, points_out, OUT_points_in,
OUT_points_out, 1.5, 9)
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_reg_12 = doRBFRegular(points_in, points_out, OUT_points_in,
OUT_points_out, 1.5, 12)

    if result_reg_9[0] > result_reg_12[0] and result_reg_9[1] < result
_reg_12[1]:
        counts[0] += 1
    elif result_reg_9[0] < result_reg_12[0] and result_reg_9[1] > resu
lt_reg_12[1]:
        counts[1] += 1
    elif result_reg_9[0] < result_reg_12[0] and result_reg_9[1] < resu
lt_reg_12[1]:
        counts[2] += 1
    elif result_reg_9[0] > result_reg_12[0] and result_reg_9[1] > resu
lt_reg_12[1]:
        counts[3] += 1
    elif result_reg_9[0] == result_reg_12[0] and result_reg_9[1] == re
sult_reg_12[1]:
        counts[4] += 1


print(counts)
```

```
[66, 29, 29, 280, 4]
```

We can see that most frequently, both E_in and E_out go down. Therefore the answer to question 16 is D.

In [34]:
```
# PROBLEM 17
RUNS = 500

counts = [0,0,0,0,0]
for i in range(RUNS):
    points = createLabeledPoints(100)
    points_in = []
    points_out = []
    for point in points:
        points_in.append([point[0], point[1]])
        points_out.append(point[2])
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_reg_9 = doRBFRegular(points_in, points_out, OUT_points_in,
OUT_points_out, 1.5, 9)
    OUT_points = createLabeledPoints(1000)
    OUT_points_in = []
    OUT_points_out = []
    for point in OUT_points:
        OUT_points_in.append([point[0], point[1]])
        OUT_points_out.append(point[2])
    result_reg_12 = doRBFRegular(points_in, points_out, OUT_points_in,
OUT_points_out, 2, 9)

    if result_reg_9[0] > result_reg_12[0] and result_reg_9[1] < result
_reg_12[1]:
        counts[0] += 1
    elif result_reg_9[0] < result_reg_12[0] and result_reg_9[1] > resu
lt_reg_12[1]:
        counts[1] += 1
    elif result_reg_9[0] < result_reg_12[0] and result_reg_9[1] < resu
lt_reg_12[1]:
        counts[2] += 1
    elif result_reg_9[0] > result_reg_12[0] and result_reg_9[1] > resu
lt_reg_12[1]:
        counts[3] += 1
    elif result_reg_9[0] == result_reg_12[0] and result_reg_9[1] == re
sult_reg_12[1]:
        counts[4] += 1


print(counts)
```

```
[25, 53, 149, 59, 4]
```

We can see that most frequently, both E_in and E_out go up. Therefore the answer to question 17 is C.

In [36]:
```
# PROBLEM 18
RUNS = 500

count = 0
for i in range(RUNS):
    points = createLabeledPoints(100)
    points_in = []
    points_out = []
    for point in points:
        points_in.append([point[0], point[1]])
        points_out.append(point[2])
    result = doRBFRegular(points_in, points_out, [[0,0]], [1], 1.5, 9)

    if result[0] == 0:
        count += 1


print(count/RUNS)
```

```
0.026
```

We can see that the percentage of time that regular RBF achieves E_in=0 with K=9 and gamma=1.5 is less than 10%. Therefore the answer to question 18 is A.

In [ ]:
```

```