(11) [C]

```
def q11():
    def z(x1, x2):
        return [1, (x2 ** 2) - (2 * x1) - 1, (x1 ** 2) - (2 * x2) + 1]
    X = [[1, 0], [0, 1], [0, -1], [-1, 0], [0, 2], [0, -2], [-2, 0]]
    Y = [-1, -1, -1, 1, 1, 1, 1]
    Z = []
    for x1,x2 in X:
        Z.append(z(x1, x2))
    classifier = svm.SVC(kernel='linear', C=numpy.inf)
    classifier.fit(Z, Y)
    print("Ein: ", 1 - classifier.score(Z, Y))
    print("Weight: ", classifier.coef_[0, 1:], 'Bias: ', classifier.intercept_)
    print('Normalized Weights: ',numpy.round(classifier.coef_[0, 1:]/2), 'Normalized Bias: ', (classifier.intercept_/2))
q11()


def q12():
    X = [[1, 0], [0, 1], [0, -1], [-1, 0], [0, 2], [0, -2], [-2, 0]]
    Y = [-1, -1, -1, 1, 1, 1, 1]

    classifier = svm.SVC(kernel='poly', degree=2, coef0=1, C=numpy.inf)
    classifier.fit(X, Y)
    print("Number Of Support Vectors: ", sum(classifier.n_support_))
def q13(runs, points):
```

hard margin

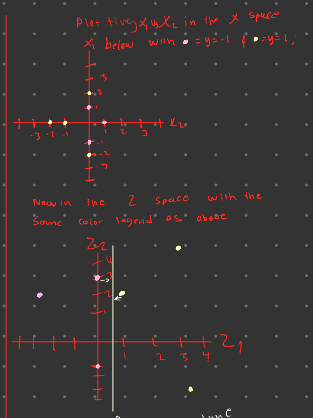
```
PROBLEMS 10  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  1
Ein:  0.0
Weight:  [1.999808e+00 6.400000e-05] Bias:  [-1.00002133]
Normalized Weights:  [1. 0.] Normalized Bias:  [-0.50001067]
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence according to the above code the correct answer is [C]

Ln 116, Col 28 Spaces: 4 UTF-8

Alternatively



Seperating plane only dependent on z_1 as it's just a vertical plane hence $w_2 = 0$. Only choice [C] matches $w_2 = 0$ however let's test it. As the most seperating plane pass through $(\frac{1}{2}, 0)$ as they equal distant from both classes of points hence:

$y = mx + b$ as it's linear

with $(\frac{1}{2}, 0)$ $\therefore$ $0 = w_1 \frac{1}{2} + b$

$= 0 = \frac{1}{2} + b$ $\therefore$ $b = -\frac{1}{2}$ hence

[C] works and is the correct answer as we commenced with our code.

(12) [C]

```
def q12():
    X = [[1, 0], [0, 1], [0, -1], [-1, 0], [0, 2], [0, -2], [-2, 0]]
    Y = [-1, -1, -1, 1, 1, 1, 1]

    classifier = svm.SVC(kernel='poly', degree=2, coef0=1, C=numpy.inf)
    classifier.fit(X, Y)
    print("Number Of Support Vectors: ", sum(classifier.n_support_))
q12()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Number Of Support Vectors:  5
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Number Of Support Vectors:  5
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Number Of Support Vectors:  5
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Number Of Support Vectors:  5
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence according to the above code, the correct answer is [C]

Ln 12, Col 35 Spaces: 4 UTF-8

(13) [a]

```
def q13(runs, points):
    def f(x1, x2):
        return numpy.sign(x2 - x1 + (0.25 * numpy.sin(numpy.pi * x1)))
    summ_fails = 0
    for r in range(runs):
        data = []
        score = []
        for i in range(points):
            point = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
            data.append(point)
            score.append(f(point[1], point[2]))
        classifier = svm.SVC(kernel='rbf', C=numpy.inf, coef0=1, gamma=1.5)
        classifier.fit(data, score)
        err = 1 - classifier.score(data, score)
        summ_fails += 0 if (err == 0) else 1
    print("Non seperable this many times: ",summ_fails/runs)

q13(1000, 100)
```

```
PROBLEMS 10  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  1
Non seperable this many times:  0.0
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py
Non seperable this many times:  0.0
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence according to the above code of 1000 runs the correct answer is [a].

Ln 148, Col 14

All helper functions used are the same throughout all code hence see original definition

(14) [e]

```python
def f(x1, x2):
    return numpy.sign(x2 - x1 + (0.25 * numpy.sin(numpy.pi * x1)))

def make_points(num, data):
    points = []
    for i in range(num):
        points.append(random.choice(data))
    return numpy.array(points)

def create_data(points):
    data = []
    score = []
    for i in range(points):
        point = [random.uniform(-1, 1), random.uniform(-1, 1)]
        data.append(point)
        score.append(f(point[0], point[1]))
    return [data, score]
```

```python

def find_centers(data, K):
    while True:
        centers = make_points(K, data)
        while True:
            clusters = [[] for j in range(K)]
            for q in data:
                curr_min = numpy.inf
                curr_min_idx = 0
                for i,p in enumerate(centers):
                    distance = numpy.sqrt(sum((q - p) ** 2))
                    if distance < curr_min:
                        curr_min_idx = i
                        curr_min = distance
                clusters[curr_min_idx].append(q)
            empty = 0
            for c in clusters:
                if len(c) == 0:
                    empty += 1
            if (empty == 0):
                break
            else:
                centers = make_points(K, data)
        curr_idx = 0
        for c in clusters:
            curr_x_mean = 0
            curr_y_mean = 0
            for point in c:
                curr_x_mean += point[0]
                curr_y_mean += point[1]
            curr_x_mean /= len(c)
            curr_y_mean /= len(c)
            centers[curr_idx] = [curr_x_mean, curr_y_mean]
            curr_idx += 1
        return centers

```

```python
def exp(gamma, x, c):
    return numpy.exp(-gamma * numpy.sum((x - c) ** 2))

def make_phi_matrix(data, K, gamma):
    phi_matrix = []
    centers = find_centers(data, K)
    for d in data:
        curr_point = [1]
        for c in centers:
            curr_point.append(exp(gamma, d, c))
        phi_matrix.append(curr_point)
    return [numpy.array(phi_matrix), centers]

def get_weight(data, K, gamma, Y):
    phi = make_phi_matrix(data, K, gamma)
    return [numpy.dot(numpy.linalg.pinv(phi[0]), Y), phi[1]]

def score_RBF(x, K, gamma, centers, w):
    tot = 0
    for i in range(1, K):
        tot += (w[i] * exp(gamma, x, centers[i]))
    return numpy.sign(tot + w[0])

def Eout_RBF(data, score, K, gamma, centers, w):
    summ_err = 0
    for d, s in zip(data, score):
        if (s != score_RBF(d, K, gamma, centers, w)):
            summ_err += 1
    return summ_err/(len(data) * K)
```

fin.py > Reg_Ein_and_Eout

```python
def Kernel_VS_Reg(K, gamma, runs):
    better = 0
    for r in range(runs):
        train_p = create_data(100)
        test_p = create_data(1000)
        d_train = train_p[0]
        s_train = train_p[1]
        d_test = test_p[0]
        s_test = test_p[1]
        #SVM
        classifier = svm.SVC(kernel='rbf', C=numpy.inf, coef0=1.0, gamma=1.5)
        classifier.fit(d_train, s_train)
        while 1 - classifier.score(d_train, s_train) != 0.0:
            train_p = create_data(100)
            d_train = train_p[0]
            s_train = train_p[1]
            classifier.fit(d_train, s_train)
        summ_err_k = (1 - classifier.score(d_test, s_test))

        #RBF
        w, centers = get_weight(d_train, K, gamma, s_train)
        summ_err_r = Eout_RBF(d_test, s_test, K, gamma, centers, w)

        if (summ_err_k < summ_err_r):
            better += 1
    return better/runs

def Reg_Ein_and_Eout(K, gamma, runs):
    summ_Ein = 0
    summ_Eout = 0
    for r in range(runs):
        train_p = create_data(100)
        test_p = create_data(1000)
        d_train = train_p[0]
        s_train = train_p[1]
        d_test = test_p[0]
        s_test = test_p[1]

        w, centers = get_weight(d_train, K, gamma, s_train)
        summ_Eout += Eout_RBF(d_test, s_test, K, gamma, centers, w)
        summ_Ein +=  Eout_RBF(d_train, s_train, K, gamma, centers, w)
    return (summ_Ein/runs, summ_Eout/runs)

def q14(gamma, K, runs):
    av = Kernel_VS_Reg(K, gamma, runs)
    print("Kernel Wins this many times: ", av)

q14(1.5, 9, 1000)
```

```
PROBLEMS 10  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Kernel Wins this many times:  0.811
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Kernel Wins this many times:  0.816
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence according to the following Code, the correct answer is [e]

All helper functions used are the same throughtout all code hence see original definintion

(15) [d]

```
273
274  def q14(gamma, K, runs):
275      av = Kernel_VS_Reg(K, gamma, runs)
276      print("Kernel Wins this many times: ", av)
277
278  def q15():
279      q14(1.5, 12, 1000)
280
281  q15()
282
283  def q16(times, gamma, K, runs):
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Kernel Wins this many times:  0.686
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Kernel Wins this many times:  0.681
thierno@Thiernos-MacBook-Pro CS  156a %
```

Ln 281, Col 6 Spaces: 4

See 914 code for 914() code.

According to the above code, the correct answer is [d].

(16) [d]

```
257
258  def Reg_Ein_and_Eout(K, gamma, runs):
259      summ_Ein = 0
260      summ_Eout = 0
261      for r in range(runs):
262          train_p = create_data(100)
263          test_p = create_data(1000)
264          d_train = train_p[0]
265          s_train = train_p[1]
266          d_test = test_p[0]
267          s_test = test_p[1]
268
269          w, centers = get_weight(d_train, K, gamma, s_train)
270          summ_Eout += Eout_RBF(d_test, s_test, K, gamma, centers, w)
271          summ_Ein +=  Eout_RBF(d_train, s_train, K, gamma, centers, w)
272      return (summ_Ein/runs, summ_Eout/runs)
273
274  def q14(gamma, K, runs):
275
276  def q15():
277
278  def q16(gamma, K, runs):
279      Ein, Eout = Reg_Ein_and_Eout(K, gamma, runs)
280      print("The average Ein for K=",K," is : ", Ein)
281      print("The average Eout for K=",K," is : ", Eout)
282
283  q16(1.5, 9, 1000)
284  q16(1.5, 12, 1000)
285
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
The average Ein for K= 9  is :  0.05534333333333314
The average Eout for K= 9  is :  0.05554177777777781
The average Ein for K= 12  is :  0.042284166666666672
The average Eout for K= 12  is :  0.04218191666666664
thierno@Thiernos-MacBook-Pro CS  156a %
```

Ln 285, Col 1 Spaces: 4

According to the code above, there is a decrease in both on average after 1000 runs hence the correct answer is [d]

All Helper functions used are the same throughout all code hence see original definition

Alternatively

```
285
286  def q16v2(gamma, K1, K2, runs):
287      a = 0
288      b = 0
289      c = 0
290      d = 0
291      e = 0
292      for i in range(runs):
293          Ein1, Eout1 = Reg_Ein_and_Eout(K1, gamma, 1)
294          Ein2, Eout2 = Reg_Ein_and_Eout(K2, gamma, 1)
295          if (Ein2 < Ein1 and Eout2 > Eout1):
296              a += 1
297          if (Ein2 > Ein1 and Eout2 < Eout1):
298              b += 1
299          if (Ein2 > Ein1 and Eout2 > Eout1):
300              c += 1
301          if (Ein2 < Ein1 and Eout2 < Eout1):
302              d += 1
303          if (Ein2 == Ein1 and Eout2 == Eout1):
304              e += 1
305      print("Choice a: ",a)
306      print("Choice b: ",b)
307      print("Choice c: ",c)
308      print("Choice d: ",d)
309      print("Choice e: ",e)
310
311  q16v2(1.5, 9, 12, 1000)
312
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Choice a:  22
Choice b:  40
Choice c:  287
Choice d:  646
Choice e:  0
thierno@Thiernos-MacBook-Pro CS  156a %
```

Ln 307, Col 26 Spaces: 4

Alternatively, with the code above testing each choice, the correct answer is still [d].

Both ways give the same answer.

(17) [C]

```
311
312  def q17(gamma, K, runs):
313      Ein, Eout = Reg_Ein_and_Eout(K, gamma, runs)
314      print("The average Ein for gamma=",gamma," is : ", Ein)
315      print("The average Eout for gamma=",gamma," is : ", Eout)
316
317  q17(1.5, 9, 1000)
318  q17(2.0, 9, 1000)
319
```

```
PROBLEMS 10  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

Choice e:  0
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
The average Ein for gamma= 1.5  is :  0.05730444444444456
The average Eout for gamma= 1.5  is :  0.05738666666666664
The average Ein for gamma= 2.0  is :  0.05833000000000005
The average Eout for gamma= 2.0  is :  0.05828966666666668
thierno@Thiernos-MacBook-Pro CS  156a %
```

According to the code above, both increase on average after 100 runs hence the correct answer is [C]

Alternatively

Both ways give the same answer.

```
278
279  def q17(gamma1, gamma2, K, runs):
280      a = 0
281      b = 0
282      c = 0
283      d = 0
284      e = 0
285      for i in range(runs):
286          Ein1, Eout1 = Reg_Ein_and_Eout(K, gamma1)
287          Ein2, Eout2 = Reg_Ein_and_Eout(K, gamma2)
288          if (Ein2 < Ein1 and Eout2 > Eout1):
289              a += 1
290          if (Ein2 > Ein1 and Eout2 < Eout1):
291              b += 1
292          if (Ein2 > Ein1 and Eout2 > Eout1):
293              c += 1
294          if (Ein2 < Ein1 and Eout2 < Eout1):
295              d += 1
296          if (Ein2 == Ein1 and Eout2 == Eout1):
297              e += 1
298
299      print("Choice a: ",a)
300      print("Choice b: ",b)
301      print("Choice c: ",c)
302      print("Choice d: ",d)
303      print("Choice e: ",e)
304
305  q17(1.5, 2, 9, 100)
306
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
Choice a:  4
Choice b:  6
Choice c:  54
Choice d:  36
Choice e:  0
thierno@Thiernos-MacBook-Pro CS  156a %
```

With comparing all choices, choice [C] still wins hence [C] is the correct answer

All Helper functions used are the same throughout all code hence see original definition

(18) [a]

```
342
343  def zero_Ein(gamma, K, runs):
344      Ein_zero = 0
345      for r in range(runs):
346          train_p = create_data(100)
347          d_train = train_p[0]
348          s_train = train_p[1]
349
350          w, centers = get_weight(d_train, K, gamma, s_train)
351          if Eout_RBF(d_train, s_train, K, gamma, centers, w) == 0:
352              Ein_zero += 1
353      return Ein_zero/runs
354
355  def q18(gamma, K, runs):
356      av_Ein_zero = zero_Ein(gamma, K, runs)
357      print("The average times Ein is zero is: ", av_Ein_zero)
358
359  q18(1.5, 9, 1000)
```

```
PROBLEMS 10  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
The average times Ein is zero is:  0.0
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/fin.py"
The average times Ein is zero is:  0.0
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence according to the above code, the correct answer is [a].

(19) [b] From the problem we assume that the prior, $P(h=f)$, is uniform hence our posterior is determined by our dataset entirely. From lecture 14 slide 7 we know:

$$P(h=f|D) = \frac{P(D|h=f)\,P(h=f)}{P(D)} \propto P(D|h=f)\,P(h=1)$$

Now as our prior is uniform over $h \in [0,1]$. Now as our posterior is dependant on our dataset then we know for our case it's increasing linearly over $[0,1]$ because the probability of our data, which is a single point with a heart attach, given $h=f$ is increasing linearly as $h=f$ approaches 1, the 100% chance of a heart attack, as that is our only data point, hence the correct answer is b.

(20) [c]

Note: For aggregate hypothesis each hypothesis contributes equally hence.

[a] False because for the case where g_2 is constantly an extremely poor hypothesis relative to g_1 then their average $E_{out}(g)$ would be greater than $E_{out}(g_1)$.

[b] False because if we let $g_{min} = \min(g_1, g_2)$ hence making the other g_{max} then like the logic in [a] g_{max} will be a poor hypothesis relative to g_{min} hence their average $E_{out}(g)$ would be greater than $E_{out}(g_{min})$.

[c] We know that each hypothesis g_1 & g_2 contribute equally to our aggregate hypothesis, g, hence $E_{out}(g)$ cannot be worse than the average $E_{out}(g_1)$ & $E_{out}(g_2)$ hence the correct answer is C.

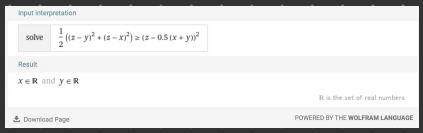
Alternatively

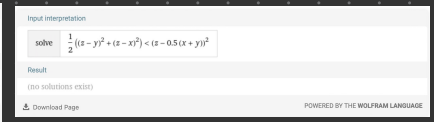
we know from lecture 4 that squared error is $E_{out}(g) = E\left[(g(x)-f(x))^2\right]$ hence the average in this case would be:

$$E\left[\frac{(g_1-f)^2 + (g_2-f)^2}{2}\right]$$

for $E_{out}(g_1)$ & $E_{out}(g_2)$ and for $E_{out}(g) = E\left[\left(\frac{(g_1+g_2)}{2} - f\right)^2\right]$

Now plugging these averages into Mathematica (with $f=2$) we see that there is no such solution where g would have a higher out of sample error than the average E_{out} of g_1 & g_2 hence solidifying that the correct answer is C.





[d] False because Consider a g_1 that overestimates the target function and a g_2 that underestimates the target function hence their average $E_{out}(g)$ will be lower than both $E_{out}(g_1)$ & $E_{out}(g_2)$.

[e] False because C is correct.