CS156a Final

1. A polynomial transform of order Q applied to X of dimension d = 2 takes the form of

(x1, x2)Q

And is added to the previous tuples. For Q = 1, we get 2 terms:

(x1, x2)

For Q = 2, we get five terms:

(x1, x2, x12, x1x2, x22)

For Q = 3, we get nine terms:

(x1, x2, x12, x1x2, x22, x13, x12x2, x1x22, x23)

We get the trend: 2 + 3 + 4 + 5… since each polynomial is one bigger than the last.

Summing that all up, we get 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 = 65.

The answer is [e].

2. Consider each option:

a. If there is only one hypothesis, then all training sets will result in the same result (thus in H)

b. This gives us g(x) = c, where c is some constant where the average is in H

c. There is nothing wrong with this model either, the average is in H as shown in past homework

d. The logistic regression model could result with an average not in H since the (s) function has some odd properties

The answer is [d].

3. Go through each option:

a. This is true. Overfitting causes the input error to decrease

b. This is true. Overfitting causes the output error to increase

c. This is true. This is a result of parts a and b

d. This is false. Some values for different orders are very close

e. This is true. It is impossible to tell with only one hypothesis

The answer is [d].

4. Go through each option:

a. This is false. Deterministic noise and stochastic noise are independent

b. This is false. If you change H, deterministic noise changes

c. This is false. It does depend on the target function

d. This is true. If you change H, stochastic noise is the same

e. This is false. The target distribution is what causes stochastic noise

The answer is [d].

5. Looking at the difference between wreg and wlin, we consider the equations for the two of them:

wreg = (ZTZ + λI)-1 ZTy

wlin = (ZTZ)-1 ZTy

The only difference is the addition of the lambda term. In this case however, the solution is obvious. The constraint for wreg is wTΓTΓw ≤ C, and it is stated that wlin is the weight that already satisfies that constraint. Thus, wreg = wlin­ through satisfaction of the constraints.

The answer is [a].

6. Consider each option:

a. No, it would become a hard-order constraint

b. Yes, constraints regularize the hypothesis which creates augmented error

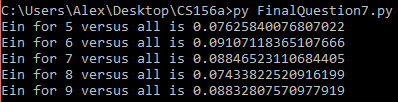
c. No, this is not relevant

d. No, this is usually not the case

The answer is [b].

7. Work is in the code:

1. **import** numpy as np
3. train = open("trainfinal.txt", "r")
5. x = []
6. y = []
7. **for** line **in** train:
8. fields = line.strip().split()
9. **for** l **in** range(int(len(fields)/3)):
10. x.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
11. y.append(float(fields[0 + 3 \* l]))
13. train.close()
15. test = open("testfinal.txt", "r")
17. **for** i **in** [5, 6, 7, 8, 9]:
18. y2 = []
19. x2 = []
20. **for** z **in** range(len(y)):
21. **if** (y[z] == i):
22. y2.append(1)
23. x2.append(x[z])
24. **else**:
25. y2.append(-1)
26. x2.append(x[z])
28. left = np.matmul(np.transpose(x2), x2)
29. summ = np.add(left, np.identity(3))
30. inv = np.linalg.inv(summ)
31. w = np.matmul(inv, np.matmul(np.transpose(x2), y2))
33. wrong = np.sum(np.dot(x2, w) \* y2 < 0)
35. **print**("Ein for " + str(i) + " versus all is " + str(wrong/len(x2)))

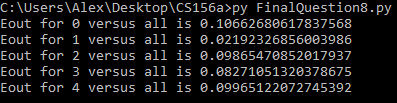


The lowest is 8 versus all, so the answer is [d].

8. Work is in the code:

1. **import** numpy as np
3. **def** transform(x1, x2):
4. **return** [1, x1, x2, x1 \* x2, x1\*\*2, x2\*\*2]

7. train = open("trainfinal.txt", "r")
9. x3 = []
10. y = []
11. **for** line **in** train:
12. fields = line.strip().split()
13. **for** l **in** range(int(len(fields)/3)):
14. x3.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
15. y.append(float(fields[0 + 3 \* l]))
17. train.close()
19. test = open("testfinal.txt", "r")
21. xtest2 = []
22. ytest = []
23. **for** line2 **in** test:
24. fields = line2.strip().split()
25. **for** l **in** range(int(len(fields)/3)):
26. xtest2.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
27. ytest.append(float(fields[0 + 3 \* l]))
29. test.close()
31. x = []
32. xtest = []
33. **for** k **in** range(len(x3)):
34. x.append(transform(x3[k][1], x3[k][2]))
36. **for** q **in** range(len(xtest2)):
37. xtest.append(transform(xtest2[q][1], xtest2[q][2]))
39. **for** i **in** range(5):
40. y2 = []
41. x2 = []
42. xtest3 = []
43. ytest3 = []
44. **for** z **in** range(len(y)):
45. **if** (y[z] == i):
46. y2.append(1)
47. x2.append(x[z])
48. **else**:
49. y2.append(-1)
50. x2.append(x[z])
52. **for** l **in** range(len(ytest)):
53. **if** (ytest[l] == i):
54. ytest3.append(1)
55. xtest3.append(xtest[l])
56. **else**:
57. ytest3.append(-1)
58. xtest3.append(xtest[l])
60. left = np.matmul(np.transpose(x2), x2)
61. summ = np.add(left, np.identity(6))
62. inv = np.linalg.inv(summ)
63. w = np.matmul(inv, np.matmul(np.transpose(x2), y2))
65. wrong = np.sum(np.dot(xtest3, w) \* ytest3 < 0)
67. **print**("Eout for " + str(i) + " versus all is " + str(wrong/len(xtest3)))

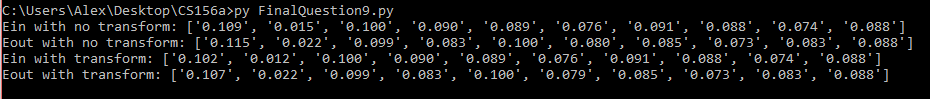


The lowest is 1 versus all, so the answer is [b].

9. The work is in the code:

1. **import** numpy as np
3. **def** transform(x1, x2):
4. **return** [1, x1, x2, x1 \* x2, x1\*\*2, x2\*\*2]

7. train = open("trainfinal.txt", "r")
9. x3 = []
10. y = []
11. **for** line **in** train:
12. fields = line.strip().split()
13. **for** l **in** range(int(len(fields)/3)):
14. x3.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
15. y.append(float(fields[0 + 3 \* l]))
17. train.close()
19. test = open("testfinal.txt", "r")
21. xtest2 = []
22. ytest = []
23. **for** line2 **in** test:
24. fields = line2.strip().split()
25. **for** l **in** range(int(len(fields)/3)):
26. xtest2.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
27. ytest.append(float(fields[0 + 3 \* l]))
29. test.close()
31. x = []
32. xtest = []
33. **for** k **in** range(len(x3)):
34. x.append(transform(x3[k][1], x3[k][2]))
36. **for** q **in** range(len(xtest2)):
37. xtest.append(transform(xtest2[q][1], xtest2[q][2]))
39. Ein = []
40. Eout = []
41. Einz = []
42. Eoutz = []
44. **for** i **in** range(10):
45. y2 = []
46. x2 = []
47. xtest3 = []
48. ytest3 = []
49. z2 = []
50. ztest2 = []
51. **for** z **in** range(len(y)):
52. **if** (y[z] == i):
53. y2.append(1)
54. x2.append(x3[z])
55. z2.append(x[z])
56. **else**:
57. y2.append(-1)
58. x2.append(x3[z])
59. z2.append(x[z])
61. **for** l **in** range(len(ytest)):
62. **if** (ytest[l] == i):
63. ytest3.append(1)
64. xtest3.append(xtest2[l])
65. ztest2.append(xtest[l])
66. **else**:
67. ytest3.append(-1)
68. xtest3.append(xtest2[l])
69. ztest2.append(xtest[l])
71. left = np.matmul(np.transpose(x2), x2)
72. summ = np.add(left, np.identity(3))
73. inv = np.linalg.inv(summ)
74. w = np.matmul(inv, np.matmul(np.transpose(x2), y2))
76. Ein.append(format((np.sum(np.dot(x2, w) \* y2 < 0) / len(x2)), '.3f'))
77. Eout.append(format((np.sum(np.dot(xtest3, w) \* ytest3 < 0) /len(xtest3)), '.3f'))
79. left = np.matmul(np.transpose(z2), z2)
80. summ = np.add(left, np.identity(6))
81. inv = np.linalg.inv(summ)
82. w = np.matmul(inv, np.matmul(np.transpose(z2), y2))
84. Einz.append(format((np.sum(np.dot(z2, w) \* y2 < 0) / len(z2)),'.3f'))
85. Eoutz.append(format((np.sum(np.dot(ztest2, w) \* ytest3 < 0) /len(xtest3)), '.3f'))
87. **print**("Ein with no transform: " + str(Ein))
88. **print**("Eout with no transform: " + str(Eout))
89. **print**("Ein with transform: " + str(Einz))
90. **print**("Eout with transform: " + str(Eoutz))



Go through each option:

a. False, most of the errors are identical between the transform and without it.

b. False, they are the same for most of them.

c. False, it improves the out-of-sample error for some cases.

d. False, they are the same for most of them.

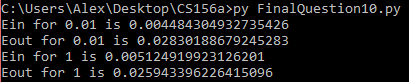
e. True, the difference is 0.08 and 0.079, which is less than 5%

The answer is [e].

10. Work is in the code:

1. **import** numpy as np
3. **def** transform(x1, x2):
4. **return** [1, x1, x2, x1 \* x2, x1\*\*2, x2\*\*2]

7. train = open("trainfinal.txt", "r")
9. x3 = []
10. y = []
11. **for** line **in** train:
12. fields = line.strip().split()
13. **for** l **in** range(int(len(fields)/3)):
14. x3.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
15. y.append(float(fields[0 + 3 \* l]))
17. train.close()
19. test = open("testfinal.txt", "r")
21. xtest2 = []
22. ytest = []
23. **for** line2 **in** test:
24. fields = line2.strip().split()
25. **for** l **in** range(int(len(fields)/3)):
26. xtest2.append([1, float(fields[1 + 3 \* l]), float(fields[2 + 3 \* l])])
27. ytest.append(float(fields[0 + 3 \* l]))
29. test.close()
31. x = []
32. xtest = []
33. **for** k **in** range(len(x3)):
34. x.append(transform(x3[k][1], x3[k][2]))
36. **for** q **in** range(len(xtest2)):
37. xtest.append(transform(xtest2[q][1], xtest2[q][2]))
39. **for** i **in** [0.01, 1]:
40. y2 = []
41. x2 = []
42. xtest3 = []
43. ytest3 = []
44. **for** z **in** range(len(y)):
45. **if** (y[z] == 1):
46. y2.append(1)
47. x2.append(x[z])
48. **elif** (y[z] == 5):
49. y2.append(-1)
50. x2.append(x[z])
52. **for** l **in** range(len(ytest)):
53. **if** (ytest[l] == 1):
54. ytest3.append(1)
55. xtest3.append(xtest[l])
56. **elif** (ytest[l] == 5):
57. ytest3.append(-1)
58. xtest3.append(xtest[l])
60. left = np.matmul(np.transpose(x2), x2)
61. summ = np.add(left,np.multiply(i, np.identity(6)))
62. inv = np.linalg.inv(summ)
63. w = np.matmul(inv, np.matmul(np.transpose(x2), y2))
65. inputerr = np.sum(np.dot(x2, w) \* y2 < 0)
66. **print**("Ein for " + str(i) + " is " + str(inputerr/len(x2)))
67. wrong = np.sum(np.dot(xtest3, w) \* ytest3 < 0)
68. **print**("Eout for " + str(i) + " is " + str(wrong/len(xtest3)))



Consider each option:

a. True, the Ein goes down but the Eout goes up

b. False, they are different

c. False, they are different

d. False, only Ein goes up

e. False, only Eout goes down

The answer is [a].

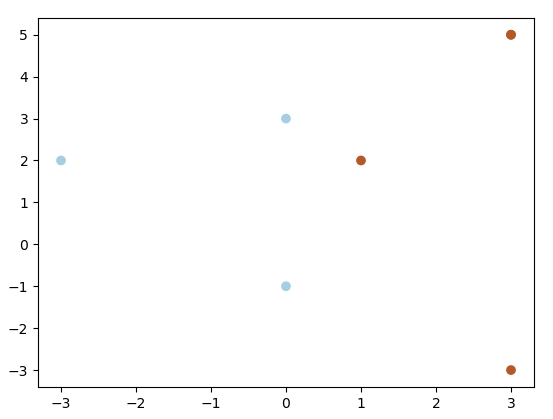
11. Using the transform:

x1 = (-3, 2), y1 = -1 x2 = (0, -1), y2 = -1 x3 = (0, 3), y3 = -1 x4 = (1, 2), y4 = 1 x5 = (3, -3), y5 = 1

x6 = (3, 5), y6 = 1 x7 = (3, 5), y7 = 1

Plotting these points:

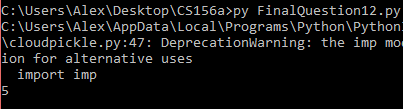
1. **import** matplotlib.pyplot as pl
2. z = [[-3,2],[0,-1],[0,3],[1,2],[3,-3],[3,5],[3,5]]
3. z1 = [i[0] **for** i **in** z]
4. z2 = [k[1] **for** k **in** z]
5. y = [-1,-1,-1,1,1,1,1]
7. pl.scatter(z1,z2,c=y,cmap=pl.cm.Paired); pl.show()



In order to maximize the margin, we need a vertical line halfway between x = 0 and x = 1. This gives us the fact that w2 = 0 and b = - 0.5. In order satisfy the condition, we can see that x1 should be nonzero and the opposite value of b. Thus, the answer is [c].

12. Run SVM with polynomial kernel and degree of 2:

1. **from** sklearn **import** svm
3. x = [[1,0],[0,1],[0,-1],[-1,0],[0,2],[0,-2],[-2,0]]
4. y = [-1,-1,-1,1,1,1,1]
6. clf = svm.SVC(kernel='poly',C=float("inf"), degree=2, gamma=1,coef0=1)
7. clf.fit(x,y)
8. **print**(len(clf.support\_))

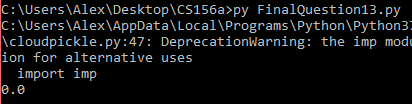


This shows that there are 5 values of that are greater than zero, which makes them a support vector.

The answer is [c].

13. Work is in the code:

1. **from** sklearn **import** svm
2. **import** numpy as np
3. **import** random
5. wrong = 0
7. **for** i **in** range(1000):
8. x = []
9. y = []
10. **for** i **in** range(100):
11. x.append([random.uniform(-1,1), random.uniform(-1,1)])
12. **for** k **in** range(100):
13. y.append(np.sign(x[k][1] - x[k][0] + 0.25 \* np.sin(np.pi \* x[k][0])))
15. clf = svm.SVC(kernel='rbf',C=float("inf"),gamma=1.5)
16. clf.fit(x,y)
17. yin = clf.predict(x)
18. Ein = np.sum(y \* yin < 0) / len(y)
19. **if** Ein != 0:
20. wrong += 1
22. **print**(wrong/1000)

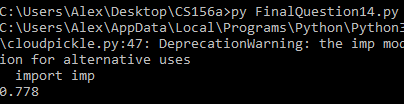


This occurs zero percent of the time, which makes sense. The answer is [a].

14. Work is in the code:

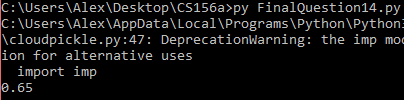
1. **from** sklearn **import** svm
2. **from** sklearn.cluster **import** KMeans
3. **import** numpy as np
4. **import** random
6. wrong = 0
8. **for** i **in** range(1000):
9. x = []
10. y = []
11. **for** i **in** range(100):
12. x.append([random.uniform(-1,1), random.uniform(-1,1)])
13. **for** k **in** range(100):
14. y.append(np.sign(x[k][1] - x[k][0] + 0.25 \* np.sin(np.pi \* x[k][0])))
16. clf = svm.SVC(kernel='rbf',C=float("inf"),gamma=1.5)
17. clf.fit(x,y)
19. x2 = []
20. y2 = []
21. **for** i **in** range(100):
22. x2.append([random.uniform(-1,1), random.uniform(-1,1)])
23. **for** k **in** range(100):
24. y2.append(np.sign(x2[k][1] - x2[k][0] + 0.25 \* np.sin(np.pi \* x2[k][0])))
26. yout = clf.predict(x2)
27. Eout = np.sum(y2 \* yout < 0) / len(y2)
29. kmeans = KMeans(n\_clusters=9).fit(x)
30. centers = kmeans.cluster\_centers\_
32. phi = []
33. **for** i **in** range(len(x)):
34. row = []
35. **for** k **in** range(9):
36. row.append(np.exp(-1.5 \* np.linalg.norm(x[i] - centers[k])\*\*2))
37. phi.append(row)
39. left = np.linalg.inv(np.matmul(np.transpose(phi), phi))
40. right = np.matmul(np.transpose(phi), y)
41. w = np.matmul(left, right)
43. phi = []
44. **for** i **in** range(len(x2)):
45. row = []
46. **for** k **in** range(9):
47. row.append(np.exp(-1.5 \* np.linalg.norm(x2[i] - centers[k])\*\*2))
48. phi.append(row)
50. Eout1 = np.sum(np.matmul(phi, w) \* y2 < 0) / len(y2)
52. **if** (Eout < Eout1):
53. wrong += 1

56. **print**(wrong/1000)



So, it is better 78% of the time. The answer is [e].

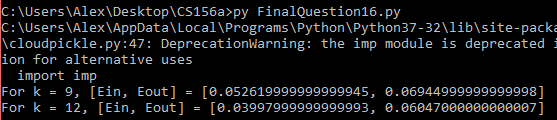
15. Changing the code from 14, but changing the 9’s to 12’s:



So, it is better 65% of the time. The answer is [d].

16. Work is in the code:

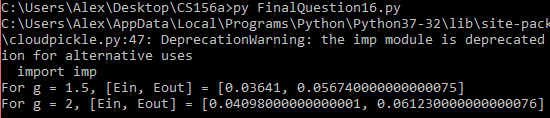
1. **from** sklearn **import** svm
2. **from** sklearn.cluster **import** KMeans
3. **import** numpy as np
4. **import** random
6. ein9 = 0
7. eout9 = 0
8. ein12 = 0
9. eout12 = 0
11. **for** i **in** range(1000):
12. x = []
13. y = []
14. **for** i **in** range(100):
15. x.append([random.uniform(-1,1), random.uniform(-1,1)])
16. **for** k **in** range(100):
17. y.append(np.sign(x[k][1] - x[k][0] + 0.25 \* np.sin(np.pi \* x[k][0])))
19. x2 = []
20. y2 = []
21. **for** i **in** range(100):
22. x2.append([random.uniform(-1,1), random.uniform(-1,1)])
23. **for** k **in** range(100):
24. y2.append(np.sign(x2[k][1] - x2[k][0] + 0.25 \* np.sin(np.pi \* x2[k][0])))
26. **for** k **in** [9, 12]:
27. kmeans = KMeans(n\_clusters=k).fit(x)
28. centers = kmeans.cluster\_centers\_
30. phi = []
31. **for** i **in** range(len(x)):
32. row = []
33. **for** q **in** range(k):
34. row.append(np.exp(-1.5 \* np.linalg.norm(x[i] - centers[q])))
35. phi.append(row)
37. left = np.linalg.inv(np.matmul(np.transpose(phi), phi))
38. right = np.matmul(np.transpose(phi), y)
39. w = np.matmul(left, right)
41. Ein = np.sum(np.matmul(phi, w) \* y < 0) / len(y)
43. phi = []
44. **for** i **in** range(len(x2)):
45. row = []
46. **for** q **in** range(k):
47. row.append(np.exp(-1.5 \* np.linalg.norm(x2[i] - centers[q])))
48. phi.append(row)
50. Eout = np.sum(np.matmul(phi, w) \* y2 < 0) / len(y2)
52. **if** k == 9:
53. ein9 += Ein
54. eout9 += Eout
55. **else**:
56. ein12 += Ein
57. eout12 += Eout
59. **print**("For k = 9, [Ein, Eout] = " + str([ein9/1000, eout9/1000]))
60. **print**("For k = 12, [Ein, Eout] = " + str([ein12/1000, eout12/1000]))



Both go down, so the answer is [d].

17. Changing the code from 16:

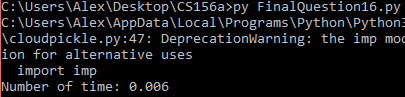
1. **from** sklearn **import** svm
2. **from** sklearn.cluster **import** KMeans
3. **import** numpy as np
4. **import** random
6. ein9 = 0
7. eout9 = 0
8. ein12 = 0
9. eout12 = 0
11. **for** i **in** range(1000):
12. x = []
13. y = []
14. **for** i **in** range(100):
15. x.append([random.uniform(-1,1), random.uniform(-1,1)])
16. **for** k **in** range(100):
17. y.append(np.sign(x[k][1] - x[k][0] + 0.25 \* np.sin(np.pi \* x[k][0])))
19. x2 = []
20. y2 = []
21. **for** i **in** range(100):
22. x2.append([random.uniform(-1,1), random.uniform(-1,1)])
23. **for** k **in** range(100):
24. y2.append(np.sign(x2[k][1] - x2[k][0] + 0.25 \* np.sin(np.pi \* x2[k][0])))
26. **for** g **in** [1.5, 2]:
27. kmeans = KMeans(n\_clusters=9).fit(x)
28. centers = kmeans.cluster\_centers\_
30. phi = []
31. **for** i **in** range(len(x)):
32. row = []
33. **for** q **in** range(9):
34. row.append(np.exp(-g \* np.linalg.norm(x[i] - centers[q])\*\*2))
35. phi.append(row)
37. left = np.linalg.inv(np.matmul(np.transpose(phi), phi))
38. right = np.matmul(np.transpose(phi), y)
39. w = np.matmul(left, right)
41. Ein = np.sum(np.matmul(phi, w) \* y < 0) / len(y)
43. phi = []
44. **for** i **in** range(len(x2)):
45. row = []
46. **for** q **in** range(9):
47. row.append(np.exp(-g \* np.linalg.norm(x2[i] - centers[q])\*\*2))
48. phi.append(row)
50. Eout = np.sum(np.matmul(phi, w) \* y2 < 0) / len(y2)
52. **if** g == 1.5:
53. ein9 += Ein
54. eout9 += Eout
55. **else**:
56. ein12 += Ein
57. eout12 += Eout
59. **print**("For g = 1.5, [Ein, Eout] = " + str([ein9/1000, eout9/1000]))
60. **print**("For g = 2, [Ein, Eout] = " + str([ein12/1000, eout12/1000]))



Both go up, so the answer is [c].

18. Work is in the code:

1. **from** sklearn **import** svm
2. **from** sklearn.cluster **import** KMeans
3. **import** numpy as np
4. **import** random
6. ein9 = 0
8. **for** i **in** range(1000):
9. x = []
10. y = []
11. **for** i **in** range(100):
12. x.append([random.uniform(-1,1), random.uniform(-1,1)])
13. **for** k **in** range(100):
14. y.append(np.sign(x[k][1] - x[k][0] + 0.25 \* np.sin(np.pi \* x[k][0])))
16. x2 = []
17. y2 = []
18. **for** i **in** range(100):
19. x2.append([random.uniform(-1,1), random.uniform(-1,1)])
20. **for** k **in** range(100):
21. y2.append(np.sign(x2[k][1] - x2[k][0] + 0.25 \* np.sin(np.pi \* x2[k][0])))
23. **for** k **in** [9]:
24. kmeans = KMeans(n\_clusters=k).fit(x)
25. centers = kmeans.cluster\_centers\_
27. phi = []
28. **for** i **in** range(len(x)):
29. row = []
30. **for** q **in** range(k):
31. row.append(np.exp(-1.5 \* np.linalg.norm(x[i] - centers[q])))
32. phi.append(row)
34. left = np.linalg.inv(np.matmul(np.transpose(phi), phi))
35. right = np.matmul(np.transpose(phi), y)
36. w = np.matmul(left, right)
38. Ein = np.sum(np.matmul(phi, w) \* y < 0) / len(y)
40. **if** Ein == 0:
41. ein9 += 1
43. **print**("Number of time: " + str(ein9/1000))



This is smaller than 10%, so the answer is [a].

19. Given the function,

P(h = f | D) = is proportional to

Given in the problem, we know that P(h=f) is constant, so the posterior depends on P(D|h = f). However, this is actually just h, since f is constant. Thus, the posterior increases as h goes over [0, 1]. This is a constant rate since we go over [0, 1] at the same rate, so the posterior increases linearly.

The answer is [b].

20. Using the mean-squared error:

E(g) = (y – 0.5(xw1 + xw2))2

E(g) = y2 - y(xw1+xw2) + 0.25(xw1 + xw2)2

E(g) = y2 - yxw1- yxw2+ 0.25(xw1)2 + 0.5xw1xw2 + 0.25(xw2)2

E(g) = 0.5(2.0(y2 - yxw1 - yxw2+ 0.25(xw1)2 + 0.5xw1xw2 + 0.25(xw2)2))

E(g) = 0.5(y2 - 2yxw1 + (xw1)2 + y2 – 2yxw2 + (xw2)2 – 0.5(xw1)2 - 0.5(xw2)2 + xw1xw2)

E(g) = 0.5((y-xw1)2 + (y-xw2)2 – 0.5(xw1 – xw2)2)

And the first two terms are the error of g1 and g2:

E(g) = 0.5(E(g1) + E(g2) – 0.5(xw1 – xw2)2)

Since the first part is the average of the errors, we are subtracting by a negative multiplied by a squared term (which are always positive), so E(g) will always be less that the average. Thus, E(g) cannot be worse than the average. The answer is [c].