Attendance Sheet

Friday, December 1, 2023 1:11 AM

Attendance Requirement Statement for CS 156a Discussion Sessions

Please attach this as the first page of your submitted final exam

Name: Subham Sahoo

I have attended at least 7 of the 18 discussion sessions in class (at least 6 if I submitted the bonus exercise).[1]

Signature: Subham Sahoo

Fall 2023

[1] Requirement reduced from what is stated in the syllabus because of a canceled discussion session.

1) $Q=1 \rightarrow (x_1, x_2)$ 2

$Q=2 \rightarrow (x_1, x_2, x_1x_2, x_1^2, x_2^2)$ 5

$Q=3 \rightarrow (x_1, x_2, x_1x_2, x_1^2, x_2^2, x_1^2x_2, x_1x_2^2, x_1^2, x_2^2)$ 9

$Q=1 \rightarrow 2$

$Q=2 \rightarrow 2+3$

$Q=3 \rightarrow 2+3+4$

dimensionality of $Z = \frac{Q(Q+1)}{2} + Q$

$= \frac{10(10+1)}{2} + 10 = 65$

With a polynomial transform of degree 10, the dimensionality of Z is 65 which is none of the options, so the answer is e.

2) a is not the right answer since all the training will return the same hypothesis. Let h be the hypothesis in $\mathcal{H}$. Over many datasets $D_1, D_2, \cdots, D_K$

$$\bar{g}(x) \approx \frac{1}{K} \sum_{i=1}^{K} g^{(D_i)}(x)$$

$$= \frac{1}{K} \left(g^{D_0}(x) + g^{D_1}(x) + \cdots + g^{D_K}(x) \right)$$

$$= \frac{1}{K} \left(\overbrace{h + h + \cdots + h}^{K \text{ times}} \right) \leftarrow \text{since } \mathcal{H} = h$$

$$= \frac{1}{K} (Kh)$$

$\bar{g}(x) \approx h$ which is in $\mathcal{H}$.

b can't be correct by a very similar logic. All $h \in \mathcal{H}$ are of the form $h(x) = b$. Over many datasets, we can estimate the following expected value.

$$\bar{g}(x) \approx \frac{1}{K} \sum_{i=1}^{K} g^{(D_i)}(x)$$

$$= \frac{1}{K}(b_1 + b_2 + \cdots + b_K)$$

$$\bar{g}(x) \approx \underbrace{\frac{1}{K}(b_1 + b_2 + \cdots + b_K)}$$

This quantity is just another real-valued constant so $\bar{g} \in \mathcal{H}$

---

c is also not right b/c of a very similar argument as b. All $h \in \mathcal{H}$ are of the form $h(x) = ax + b$. Across many datasets, the expected value is

$$\bar{g}(x) \approx \frac{1}{K} \sum_{i=1}^{K} g^{(D_i)}(x)$$

$$= \frac{1}{K}\left[(a_1 x + b_1) + (a_2 x + b_2) + \cdots + (a_K x + b_K)\right]$$

$$= \frac{1}{K}\left[(a_1 + a_2 + \cdots + a_K)x + (b_1 + b_2 + \cdots + b_K)\right]$$

$$= \frac{(a_1 + a_2 + \cdots + a_K)}{K}x + \frac{(b_1 + b_2 + \cdots + b_K)}{K}$$

$$\bar{g}(x) \approx \frac{(a_1 + a_2 + \cdots + a_K)}{K}x + \frac{(b_1 + b_2 + \cdots + b_K)}{K}$$

Since $\bar{g}(x)$ is just another hypothesis of the form $ax+b$, clearly $\bar{g} \in \mathcal{H}$

---

d is the correct answer because of the same argument. All $h \in \mathcal{H}$ are of the form $h(x) = \frac{1}{1+e^{-(a+bx)}}$

WLOG, we take the expected value across 2 datasets.

$$\bar{g}(x) \approx \frac{1}{2} \sum_{i=1}^{2} g^{(D_i)}(x)$$

$$= \frac{1}{2}\left(\frac{1}{1+e^{-(a_1+b_1x)}} + \frac{1}{1+e^{-(a_2+b_2x)}} \right)$$

$$= \frac{1}{2}\left(\frac{1+e^{-(a_2+b_2x)} + 1 + e^{-(a_1+b_1x)}}{(1+e^{-(a_1+b_1x)})(1+e^{-(a_2+b_2x)})} \right)$$

$$\bar{g}(x) \approx \frac{2 + e^{-(a_1+b_1x)} + e^{-(a_2+b_2x)}}{2(1+e^{-(a_1+b_1x)})(1+e^{-(a_2+b_2x)})}$$

Clearly, $\bar{g}(x)$ is not of the form $\frac{1}{1+e^{-(a+bx)}}$, so $\bar{g} \notin \mathcal{H}$ and e is wrong

The correct answer is d

3) Let h_1 be a hypothesis that overfit the data relative to another hypothesis h_2. By definition,

$$E_{in}(h_1) < E_{in}(h_2) \qquad E_{out}(h_1) > E_{out}(h_2)$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$E_{in}(h_1) \neq E_{in}(h_2) \qquad E_{out}(h_1) \neq E_{out}(h_2)$$

From the above staments, we see that if there is overfitting, there are in fact at least two hypotheses with different E_{in} & E_{out} values. Thus, a & b are true statements & not the right answer.

Adding the two inequatities, we see that

$$E_{in}(h_1) + E_{out}(h_2) < E_{in}(h_2) + E_{out}(h_1)$$

$$E_{out}(h_2) - E_{in}(h_2) < E_{out}(h_1) - E_{in}(h_1)$$

$$E_{out}(h_2) - E_{in}(h_2) \neq E_{out}(h_1) - E_{in}(h_1)$$

Thus, h_1 & h_2 have different values of $E_{out} - E_{in}$ making c true & not the right answer.

d is false because the overfit measure is determined by comparing the out-of-sample errors for h_1 & h_2 i.e. $E_{out}(h_1) - E_{out}(h_2)$. We do not look at the in-sample-error when determining overfitting.

e is clearly true because we need another hypothesis to compare to when determining the overfit measure.

The correct answer is d.

4) a is false because why not? There's no reason one can't have deterministic noise and stochastic noise.

b & c are false since the definition of deterministic noise the part of f (target function) not captured by $\mathcal{H}$ (hypothesis set) i.e. $f(x) - h^*(x)$. Thus, deterministic noise depends on both the target function & the hypothesis set.

d is true because stochastic noise depends entirely on the input data / probability distribution on X. Regardless of what hypothesis set is used, the noise in the input data doesn't change.

e is false since stochastic noise arises from the probabilistic distribution applied to X and the target distribution. Thus, it does depend on the target distribution.

The correct answer is d.

5) a & e are wrong because there is no regularization term in w_{reg}. Setting $w_{reg} = Cw_{lin}$ is just performing linear regression w/out regularization.

Typically, $w_{reg} = (Z^T Z + \lambda I)^{-1} Z^T y$ where $w^T w$ (equivalent to $w^T I w$) was the regularization parameter. Looking at the formulation of the problem, we see that $w^T \Gamma^T \Gamma w$ is the regularization term. Matching terms in $w^T I w$ & $w^T \Gamma^T \Gamma w$, we can see that $\lambda I = \Gamma^T \Gamma$. This means b can't be true since b suggests a regularization factor of Γ, and since d just scales b, d is incorrect.

c is the only choice which suggests a regularization factor of $\Gamma^T \Gamma$, so the answer is c.

6) a is wrong because the idea of soft margin is to provide a bit more flexibility with the constraint. A hard-order constraint removes this flexibility.

b is right because you can use constraints like one would in linear regression to regularize your hypothesis

c is wrong b/c the VC dimension deals more with the complexity of the hypothesis space as opposed to the model space the constraints deal with.

d is wrong because the soft-order constraints implement a regularization that actually penalizes Ein and helps approximate Eout better i.e., Ein goes up but Eout goes down generally.

e is wrong because b is true.

$\boxed{\text{The correct answer is b.}}$

7

Thursday, November 30, 2023 7:51 PM

The below code was used for all questions 7-10

```
[1]
import numpy as np

[2]
training_features = np.loadtxt("features.train")
testing_features = np.loadtxt("features.test")

[3]
def split_data(one_versus_all, dataset, choice1, choice2):
    to_return = dataset
    if not one_versus_all:
        to_return = []

        for digit, intensity, symmetry in dataset:
            if digit == choice1 or digit == choice2:
                to_return.append([digit, intensity, symmetry])

    to_return = np.array(to_return)
    digits = to_return[:, 0]
    intensity = to_return[:, 1]
    symmetry = to_return[:, 2]

    return digits, intensity, symmetry

[5]
def generate_one_versus_all_yn(digits, one_choice):
    yn = np.ones(len(digits))

    for i, digit in enumerate(digits):
        if digit != one_choice:
            yn[i] = -1

    return np.array(yn)
```

```python
def parse_digits(digits, choice1, choice2):
    result = []

    for digit in digits:
        if digit == choice1 or digit == choice2:
            result.append(digit)

    return digits
```

[6]

```python
def generate_one_versus_one_yn(digits, choice1, choice2):
    parsed_digits = parse_digits(digits, choice1, choice2)
    yn = np.ones(len(parsed_digits))

    for i, digit in enumerate(digits):
        if digit == choice2:
            yn[i] = -1

    return yn
```

[7]

```python
def calc_error(classification, w, z_transform):
    yn = classification
    counter = 0
    for i in range(len(z_transform)):
        z = z_transform[i]
        w_val = np.sign(np.dot(w, z))

        if w_val != yn[i]:
            counter += 1

    return counter / len(classification)
```

[8]

```python
def transform_data(x1, x2):
    return (1, x1, x2)
```

[9]

```python
def calculate_w_reg(Z, ZT, lambda_value, y):
    step_1 = np.matmul(ZT, Z)
    step_2 = step_1 + lambda_value * np.identity(len(step_1))
    step_3 = np.linalg.inv(step_2)
    step_4 = np.matmul(step_3, ZT)
    w_reg = np.matmul(step_4, y)

    return w_reg
```

[10]

```python
def run_regularized_linear_regression(lambda_value, one_versus_all, choice, choice1=-1):
    digits_train, intensities_train, symmetries_train = split_data(one_versus_all, training_features, choice, choice1)
    digits_test, intensities_test, symmetries_test = split_data(one_versus_all, testing_features, choice, choice1)

    if (one_versus_all):
        yn_train = generate_one_versus_all_yn(digits_train, choice)
        yn_test = generate_one_versus_all_yn(digits_test, choice)

    else:
        yn_train = generate_one_versus_one_yn(digits_train, choice, choice1)
        yn_test = generate_one_versus_one_yn(digits_test, choice, choice1)

    xn_train = np.column_stack((intensities_train, symmetries_train))
    xn_train_transform = np.array([transform_data(x1, x2) for (x1, x2) in xn_train])

    xn_test= np.column_stack((intensities_test, symmetries_test))
    xn_test_transform = np.array([transform_data(x1, x2) for (x1, x2) in xn_test])

    w_reg = calculate_w_reg(xn_train_transform, np.transpose(xn_train_transform), lambda_value, yn_train)

    Ein = calc_error(yn_train, w_reg, xn_train_transform)
    Eout = calc_error(yn_test, w_reg, xn_test_transform)

    return Ein, Eout
```

[11]

7)

Q7

```
choices = [5, 6, 7, 8, 9]

for choice in choices:
    Ein, _ = run_regularized_linear_regression(lambda_value=1, one_versus_all=True, choice=choice)
    print(f"E_in for {choice} vs. all: {Ein}")
```

[12]

```
E_in for 5 vs. all: 0.07625840076807022
E_in for 6 vs. all: 0.09107118365107666
E_in for 7 vs. all: 0.08846523110684405
E_in for 8 vs. all: 0.07433822520916199
E_in for 9 vs. all: 0.08832807570977919
```

The above code shows that 8 vs. all produces the lowest Ein. The answer is d

8)

Q8

```
def transform_data(x1, x2):
    return (1, x1, x2, x1 * x2, x1 * x1, x2 * x2)
```

[12] ✓ 0.0s

+ Code + Markd

```
choices = [0, 1, 2, 3, 4]

for choice in choices:
    _, Eout = run_regularized_linear_regression(lambda_value=1, one_versus_all=True, choice=choice)
    print(f"E_out for {choice} vs. all: {Eout}")
```

[22] ✓ 0.1s

```
E_out for 0 vs. all: 0.10662680617837568
E_out for 1 vs. all: 0.02192326856003986
E_out for 2 vs. all: 0.09865470852017937
E_out for 3 vs. all: 0.08271051320378675
E_out for 4 vs. all: 0.09965122072745392
```

The above code shows that 1 vs. all produces the lowest Eout, so the answer is b.

9)

Q9

```
Eout_no_transform = []
Eout_transform = []
```

[14] ✓ 0.0s

```
def transform_data(x1, x2):
    return (1, x1, x2)
```

[15] ✓ 0.0s

```
choices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for choice in choices:
    _, Eout = run_regularized_linear_regression(lambda_value=1, one_versus_all=True, choice=choice)
    Eout_no_transform.append(Eout)
    print(f"E_out for {choice} vs. all: {Eout}")
```

[23] ✓ 0.3s

```
E_out for 0 vs. all: 0.10662680617837568
E_out for 1 vs. all: 0.02192326856003986
E_out for 2 vs. all: 0.09865470852017937
E_out for 3 vs. all: 0.08271051320378675
E_out for 4 vs. all: 0.09965122072745392
E_out for 5 vs. all: 0.07922272047832586
E_out for 6 vs. all: 0.08470353761833582
E_out for 7 vs. all: 0.07324364723467862
E_out for 8 vs. all: 0.08271051320378675
E_out for 9 vs. all: 0.08819133034379671
```

```
def transform_data(x1, x2):
    return (1, x1, x2, x1 * x2, x1 * x1, x2 * x2)
```

[17] ✓ 0.0s

```
choices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for choice in choices:
    _, Eout = run_regularized_linear_regression(lambda_value=1, one_versus_all=True, choice=choice)
    Eout_transform.append(Eout)
    print(f"E_out for {choice} vs. all: {Eout}")
```

[24] ✓ 0.3s

```
E_out for 0 vs. all: 0.10662680617837568
E_out for 1 vs. all: 0.02192326856003986
E_out for 2 vs. all: 0.09865470852017937
E_out for 3 vs. all: 0.08271051320378675
E_out for 4 vs. all: 0.09965122072745392
E_out for 5 vs. all: 0.07922272047832586
E_out for 6 vs. all: 0.08470353761833582
E_out for 7 vs. all: 0.07324364723467862
E_out for 8 vs. all: 0.08271051320378675
E_out for 9 vs. all: 0.08819133034379671
```

```python
# show that b, c, and d are wrong
print("Classifier -->\tEout no transform | Eout with transform")
for i, (e1, e2) in enumerate(zip(Eout_no_transform, Eout_transform)):
    if (e1 > e2):
        print(f"{i} vs. all --> {e1} > {e2}")

    else:
        print(f"{i} vs. all --> {e1} ≤ {e2}")
```

[19] ✓ 0.0s MagicPython

```
Classifier -->  Eout no transform | Eout with transform
0 vs. all --> 0.11509715994020926 > 0.10662680617837568
1 vs. all --> 0.02242152466367713 > 0.02192326856003986
2 vs. all --> 0.09865470852017937 ≤ 0.09865470852017937
3 vs. all --> 0.08271051320378675 ≤ 0.08271051320378675
4 vs. all --> 0.09965122072745392 ≤ 0.09965122072745392
5 vs. all --> 0.07972097658196313 > 0.07922272047832586
6 vs. all --> 0.08470353761833582 ≤ 0.08470353761833582
7 vs. all --> 0.07324364723467862 ≤ 0.07324364723467862
8 vs. all --> 0.08271051320378675 ≤ 0.08271051320378675
9 vs. all --> 0.08819133034379671 ≤ 0.08819133034379671
```

By looking at the difference in the Eout between no transformation and the transformation, we see that the overfit measure is always less than or equal to 0, so there is no overfitting occuring and thus a is wrong.

```python
# investigate e
print((1 - (0.07922272047832586 / 0.07972097658196313 )) * 100)
```

[20] ✓ 0.0s MagicPython

```
0.6249999999999978
```

From the values of Eout generated w/out & w/ the transform, we see that b, c, & d are clearly wrong. We also see that the transformation on 5 vs. all improved Eout by 0.6% which is less than 5%, so the answer is e.

10)

```
Ein_0, Eout_0 = run_regularized_linear_regression(lambda_value=0.01, one_versus_all=False, choice=1, choice1=5)
Ein_1, Eout_1 = run_regularized_linear_regression(lambda_value=1, one_versus_all=False, choice=1, choice1=5)

print(f"lambda = 0.01 --> Ein = {Ein_0} | Eout = {Eout_0}")
print(f"lambda = 1.00 --> Ein = {Ein_1} | Eout = {Eout_1}")
```

[21] ✓ 0.0s

```
lambda = 0.01 --> Ein = 0.004484304932735426 | Eout = 0.02830188679245283
lambda = 1.00 --> Ein = 0.005124919923126201 | Eout = 0.025943396226415096
```

The above code shows that b–e are wrong. When going from $\lambda = 1$ to $\lambda = 0.01$, we do observe that Ein goes down while Eout goes up implying overfitting,

So the answer is a

(1)

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC
```

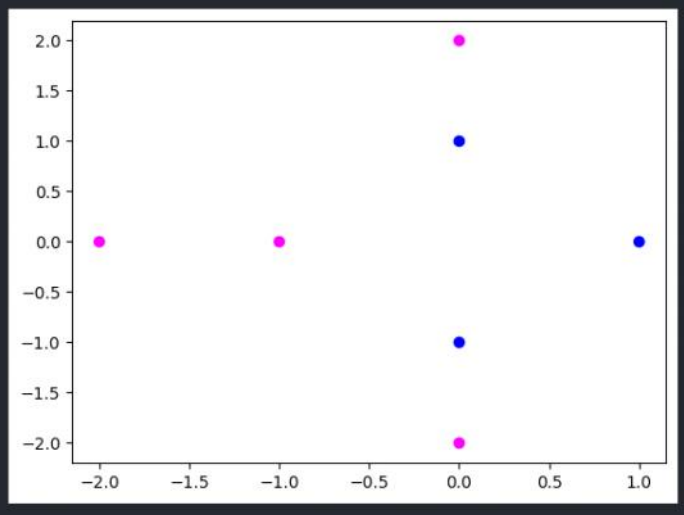
[1]

```
x = np.array([[1, 0], [0, 1], [0, -1], [-1, 0], [0, 2], [0, -2], [-2, 0]])
y = [-1, -1, -1, 1, 1, 1, 1]
c = ['blue' if y < 0 else 'magenta' for y in y]

plt.scatter(x[:, 0], x[:, 1], c=c)
```

[2]

```
<matplotlib.collections.PathCollection at 0x2456bf3c790>
```

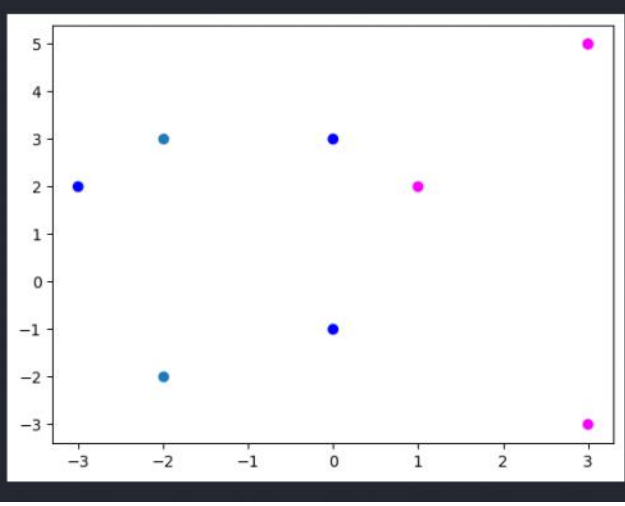



```
z = np.array([(x2 * x2 - 2 * x1 - 1, x1 * x1 - 2 * x2 + 1) for (x1, x2) in x])
c = ['blue' if y < 0 else 'magenta' for y in y]

plt.scatter(z[:, 0], z[:, 1], c=c)
plt.scatter([-2, -2], [-2, 3])
```

[3]

```
<matplotlib.collections.PathCollection at 0x2456e17b1f0>
```



Note in the figure above that the transformation to Z

Note in the figure above that the transformation to Z led to (3,5) occuring twice.

It's easy to see from the above graph that a simple vertical line separates the data and maximizes the margins (by maximizing distance to closest points on $x_1 = 0$ & $x_1 = 1$). Since the separating plane is vertical, w_1 is non-zero & w_2 is zero. This means the

$\boxed{\text{answer must be C.}}$

12)

Q12

```python
def run_svm(C, Q):
    model = SVC(C=C, kernel='poly', degree=Q, gamma=1)
    model.fit(x, y)

    return np.sum(model.n_support_)
```

[10]

```python
possible_c_values = [10 ** i for i in range(-10, 11)]

for C in possible_c_values:
    num_support_vectors = run_svm(C, 2)
    print(f"C: {C} --> {num_support_vectors} support vectors")
```

[22] ✓ 0.0s

```
C: 1e-10 --> 6 support vectors
C: 1e-09 --> 6 support vectors
C: 1e-08 --> 6 support vectors
C: 1e-07 --> 6 support vectors
C: 1e-06 --> 6 support vectors
C: 1e-05 --> 6 support vectors
C: 0.0001 --> 6 support vectors
C: 0.001 --> 7 support vectors
C: 0.01 --> 7 support vectors
C: 0.1 --> 7 support vectors
C: 1 --> 6 support vectors
C: 10 --> 5 support vectors
C: 100 --> 5 support vectors
C: 1000 --> 5 support vectors
C: 10000 --> 5 support vectors
C: 100000 --> 5 support vectors
C: 1000000 --> 5 support vectors
C: 10000000 --> 5 support vectors
C: 100000000 --> 5 support vectors
C: 1000000000 --> 5 support vectors
C: 10000000000 --> 5 support vectors
```

The above code shows that the optimal $\alpha_1, \dots, \alpha_7$ occur when $C \geq 10$ with 5 support vectors. This is in the range 4-5, so the answer is C.

The below code is used for all questions 13-18.

```python
import numpy as np
from sklearn.svm import SVC
from sklearn.cluster import KMeans
import random as rand
```

[1]

```python
num_points = 100
num_iterations = 10000
```

[2]

```python
def generate_training_points(num_points):
    xn = np.array([(1, rand.uniform(-1, 1), rand.uniform(-1, 1)) for _ in range(num_points)])
    yn = np.array([np.sign(x2 - x1 + 0.25 * np.sin(np.pi * x1)) for _, x1, x2 in xn])

    return xn, yn
```

[3]

```python
def calc_error(yn, y_pred):
    # misclassified points are opposite sign so will be -1 when multiplied
    error = yn * y_pred
    return np.count_nonzero(error == -1) / len(yn)
```

[4]

```python
def run_svm(xn, yn, C=1):
    model = SVC(C=C, kernel='rbf', gamma=1.5)
    model.fit(xn, yn)

    Ein = np.sum(model.predict(xn)*yn<0)/(1.*yn.size)

    return Ein, model
```

[5]

13)

Q13

```python
counter = 0

for i in range(num_iterations):
    xn, yn = generate_training_points(num_points)
    Ein, _ = run_svm(xn, yn)

    if (Ein == 0):
        counter += 1

print(f"Rate of data set not being separable by RBF: {100 * counter / num_iterations}%")
```

[6]

```
Rate of data set not being separable by RBF: 4.97%
```

The above code shows that $\leq 5\%$ of the time,

a dataset is not separable by the RBF Kernel.

The answer is a.

14

Thursday, November 30, 2023 7:52 PM

The below code is used for all questions 14-18.

Q14

```
def rbf1(x,mu0,gamma=1.5):
    return( np.exp(-gamma*np.sum((x-mu0)**2)) )
```

[7]

```
def rbf(xn, centers, gamma):
    z = np.zeros((xn.shape[0], 1+centers.shape[0]))  # N x (K+1), extra 1 for intercept
    z[:,0] = np.ones(xn.shape[0])

    for i in range(centers.shape[0]):
        z[:,i+1] = np.apply_along_axis(rbf1, 1, xn, centers[i],gamma)

    return z
```

[8]

```
def calculate_w_reg(Z, ZT, lambda_value, y):
    step_1 = np.matmul(ZT, Z)
    step_2 = step_1 + lambda_value * np.identity(len(step_1))
    step_3 = np.linalg.inv(step_2)
    step_4 = np.matmul(step_3, ZT)
    w_reg = np.matmul(step_4, y)

    return w_reg
```

[9]

```python
def regular_rbf(xn, yn, n_clusters, gamma):
    kmeans = KMeans(n_clusters=n_clusters).fit(xn)

    centers = kmeans.cluster_centers_

    z = rbf(xn, centers, gamma=gamma)

    w = calculate_w_reg(z, z.T, 0, yn)

    y_pred = np.dot(z, w)

    Ein = np.sum(y_pred*yn<0)/(1.*yn.size)

    return Ein, w, centers
```

[38]

```python
def run_experiment(num_clusters, gamma=1.5):
    xn, yn = generate_training_points(num_points)
    xn_test, yn_test = generate_training_points(num_points)

    # regular RBF
    Ein_regular, w, centers = regular_rbf(xn, yn, num_clusters, gamma=gamma)

    z_out = rbf(xn_test, centers, gamma=gamma)

    y_pred = np.dot(z_out, w)

    Eout_regular = np.sum(y_pred*yn_test<0)/(1.*yn_test.size)

    # kernel RBF
    _, model = run_svm(xn, yn, C=1e6)
    y_pred_regular = model.predict(xn_test)
    Ein_kernel = np.sum(y_pred_regular*yn<0)/(1.*yn.size)
    Eout_kernel = np.sum(y_pred_regular*yn_test<0)/(1.*yn_test.size)

    return Ein_regular, Eout_regular, Ein_kernel, Eout_kernel
```

[39]

14)

```python
eout_regular_arr = []
eout_kernel_arr = []

for i in range(100):
    _, Eout_regular, _, Eout_kernel = run_experiment(num_clusters=9)

    eout_regular_arr.append(Eout_regular)
    eout_kernel_arr.append(Eout_kernel)
```

[40]

```python
counter = 0

for (Eout_regular, Eout_kernel) in zip(eout_regular_arr, eout_kernel_arr):
    if (Eout_kernel < Eout_regular):
        counter += 1

print(f"Kernel beating regular: {100 * counter / 100}%")
```

[41]

```
Kernel beating regular: 79.0%
```

The above code shows that the RBF Kernel beats

The above code shows that the RBF Kernel beats regular RBF 79% of the time which is >75%, so the answer is e.

15)

Q15

```
eout_regular_arr = []
eout_kernel_arr = []

for i in range(100):
    _, Eout_regular, _, Eout_kernel = run_experiment(num_clusters=12)

    eout_regular_arr.append(Eout_regular)
    eout_kernel_arr.append(Eout_kernel)
```

[15]

```
counter = 0

for (Eout_regular, Eout_kernel) in zip(eout_regular_arr, eout_kernel_arr):
    if (Eout_kernel < Eout_regular):
        counter += 1

print(f"Kernel beating regular: {100 * counter / 100}%")
```

[16]

```
Kernel beating regular: 61.0%
```

The above code shows that the RBF Kernel beats regular RBF 61% of the time which is between 60% & 90%, so the answer is d.

16)

Q16

```
Ein_9 = []
Ein_12 = []
Eout_9 = []
Eout_12 = []
```

[17]

```
for i in range(100):
    Ein, Eout, _, _ = run_experiment(num_clusters=9)
    Ein_9.append(Ein)
    Eout_9.append(Eout)

    Ein, Eout, _, _ = run_experiment(num_clusters=12)
    Ein_12.append(Ein)
    Eout_12.append(Eout)
```

[18]

```
Ein_down_Eout_up = 0
Ein_up_Eout_down = 0
Ein_up_Eout_up = 0
Ein_down_Eout_down = 0

for ein9, ein12, eout9, eout12 in zip(Ein_9, Ein_12, Eout_9, Eout_12):
    if (ein9 < ein12):
        if eout9 < eout12:
            Ein_up_Eout_up += 1
        else:
            Ein_up_Eout_down += 1

    else:
        if eout9 < eout12:
            Ein_down_Eout_up += 1
        else:
            Ein_down_Eout_down += 1

Ein_down_Eout_up, Ein_up_Eout_down, Ein_up_Eout_up, Ein_down_Eout_down
```

[19]

```
(28, 9, 12, 51)
```

The above code shows that Ein & Eout go down the most often across the runs, so the answer is d.

17)

Q17

```
Ein_15 = []
Ein_2 = []
Eout_15 = []
Eout_2 = []
```

[29]

```
for i in range(100):
    Ein, Eout, _, _ = run_experiment(num_clusters=9, gamma=1.5)
    Ein_15.append(Ein)
    Eout_15.append(Eout)

    Ein, Eout, _, _ = run_experiment(num_clusters=9, gamma=2)
    Ein_2.append(Ein)
    Eout_2.append(Eout)
```

[30]

```
Ein_down_Eout_up = 0
Ein_up_Eout_down = 0
Ein_up_Eout_up = 0
Ein_down_Eout_down = 0

for ein15, ein2, eout15, eout2 in zip(Ein_15, Ein_2, Eout_15, Eout_2):
    if (ein15 < ein2):
        if eout15 < eout2:
            Ein_up_Eout_up += 1
        else:
            Ein_up_Eout_down += 1

    else:
        if eout15 < eout2:
            Ein_down_Eout_up += 1
        else:
            Ein_down_Eout_down += 1

Ein_down_Eout_up, Ein_up_Eout_down, Ein_up_Eout_up, Ein_down_Eout_down
```

[31]

```
(17, 18, 36, 29)
```

The above code shows that Ein & Eout go up the most often across the runs, so the answer is c.

18)

Q18

```python
counter = 0

for i in range(1000):
    Ein, _, _, _ = run_experiment(num_clusters=9, gamma=1.5)

    if (Ein == 0):
        counter += 1

print(f"Percentage of time regular RBF achieves Ein=0: {100 * counter / 1000}%")
```

[23]

```
Percentage of time regular RBF achieves Ein=0: 2.9%
```

The above code shows that regular RBF with the given parameters achieves Ein = 0 2.9% of the time. This is less than 10%, so the answer is a.

19) $$\underbrace{P(h=f \mid D)}_{\text{posterior}} = \frac{P(D \mid h=f)\, P(h=f)}{P(D)} \propto P(D \mid h=f)\, P(h=f)$$

$$\propto \overbrace{P(D \mid h=f)}^{\text{likelihood}}\, P(h=f)$$

If $h=f$, then then the probability of observing a person with a heart attack is just f.

$$P(h=f \mid D) \propto f\, P(h=f)$$

Since $P(h=f)$ is uniform and f is just a constant, the answer seems to be a. My intuition is telling me that it's b though because if we know that a person has had a heart attack, then $P(h=0)=0$ and $P(h=1)=1$. This implies a linear increase from $[0,1]$.

$\boxed{\text{The answer is a or b.}}$

20) Let n be the size of X and g_1 & g_2 be learned hypotheses through linear regression

$$E_{out}(g_1) = \frac{1}{n} \sum_{i=1}^{n} \left(f(x_i) - g_1(x_i) \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(w_1 x_i - y_i \right)^2$$

$$= \frac{1}{n} \| X w_1 - y \|^2$$

$$E_{out}(g_2) = \frac{1}{n} \| X w_2 - y \|^2$$

$$E_{out}(g) = \frac{1}{n} \sum_{i=1}^{n} \left(f(x_i) - g(x_i) \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{2} g_1(x_i) + \frac{1}{2} g_2(x_i) - y_i \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{2} x_i w_1 + \frac{1}{2} x_i w_2 - y_i \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(x_i \left(\frac{w_1 + w_2}{2} \right) - y_i \right)^2$$

$$= \frac{1}{n} \left\| X \left(\frac{w_1 + w_2}{2} \right) - y \right\|^2$$

$$= \frac{1}{n} \| X w_1 + X w_2 - y \|^2$$

$$= \frac{1}{4n} \| Xw_1 + Xw_2 - y \|^2$$

WLOG, assume that $w_2 > w_1$

$$\frac{1}{n} \|Xw_1 - y\|^2 \overset{?}{=} \frac{1}{4n} \| Xw_1 + Xw_2 - y\|^2 \overset{?}{=} \frac{1}{n} \| Xw_2 - y \|^2$$

$$\|Xw_1 - y\|^2 \overset{?}{=} \frac{1}{4} \| Xw_1 + Xw_2 - y\|^2 \overset{?}{=} \| Xw_2 - y \|^2$$

$$\|Xw_1 - y\| \overset{?}{=} \frac{1}{2} \| Xw_1 + Xw_2 - y\| \overset{?}{=} \| Xw_2 - y \|$$

$$2 \|Xw_1 - y\| \overset{?}{=} \| Xw_1 + Xw_2 - y\| \overset{?}{=} 2 \| Xw_2 - y \|$$

$$2 \|Xw_1 - y\| < \| Xw_1 + Xw_2 - y\| < 2 \| Xw_2 - y \|$$

a is wrong b/c we do not have enough information to compare $E_{out}(g)$ to either $E_{out}(g_1)$ or $E_{out}(g_2)$ by itself.

b is wrong by the above inequality, and the above inequality shows that d must be true.

The definition of g suggests that c is the answer.

The answer is c or d.

Test

Friday, December 1, 2023 1:00 PM