1 Nonlinear Transforms

Problem 1. *e*
The polynomial transform of order $Q = 10$ consists of all ${x_1}^i {x_2}^j$ where $i + j \leq Q$. Let us consider small examples for $Q = \{1, 2, 3\}$.

For $Q = 1$, we have $(1, x_1, x_2)$. $\mathbb{Z}$ has dimensionality 2.
For $Q = 2$, we have $(1, x_1, x_2, x_1x_2, {x_2}^2, {x_1}^2)$. $\mathbb{Z}$ has dimensionality 5.
For $Q = 3$, we have $(1, x_1, x_2, x_1x_2, {x_1}^2, {x_1}^2, x_1{x_2}^2, {x_1}^2x_2)$. $\mathbb{Z}$ has dimensionality 7.

Therefore, the dimensionality of the $\mathbb{Z}$ space is given by $\mathbb{Z} = (\sum_{i=0}^{Q}(Q - i) + 1) - 1$. When $Q = 10$, the dimension of $\mathbb{Z}$ is 65 which is none of the options.

2 Bias and Variance

Problem 2. *d*
Let us go through all possibilities.

For (a), if H has one hypothesis, then g^D will always be the same. The average hypothesis, $\bar{g}$, will also be the same hypothesis $\in H$. Therefore, this does not work.
For (b), if H is the set of all constant, real-values hypotheses, then each hypothesis, g^D, will be a constant, real value. The average hypothesis,$\bar{g}$, will also be a constant, real value, meaning $\bar{g} \in H$. Therefore, this does not work.
For (c), if H is the linear regression model, then g^D will be a polynomial with real coefficients. The average hypothesis, $\bar{g}$, will also contain real coefficients, and as a result, $\bar{g} \in H$. Thus, this also does not work.
For (d), if H is the logistic regression model, it relies on a sigmoid function that ranges from 0 to 1. When trying to obtain an average hypothesis, $\bar{g}$, it may not be a sigmoid function.

Thus, $\bar{g} \notin H$ always.

3 Overfitting

Problem 3. *d*
Overfitting occurs when a model learns the training data too well, including its noise and outliers, to the point that it performs poorly on new, unseen data. While comparing the values of E_out and E_in can sometimes give insights into overfitting, it's not a foolproof method. There are situations where E_out and E_in might be close or have a small difference, yet the model is still overfitting. To assess overfitting more accurately, we would use techniques such

as cross-validation, where the dataset is split into multiple folds for training and testing, or to monitor the performance of the model on a separate validation set that was not used during training.

Problem 4. ***d***

The statement, Stochastic noise does not depend on the hypothesis, is true. Deterministic noise depends on the complexity of the hypothesis, but stochastic noise is noise related with the data itself.

4 Regularization

Problem 5. ***a***

If we are given that $w_{lin}^T \Gamma^T \Gamma w_{lin} \leq C$*, where* w_{lin} *is the linear regression solution, then the value for* w_{reg} *can just be set to this solution.*

Problem 6. ***b***

From lecture, we know that soft order constraints that regularize polynomials minimizes E_{aug}*. Thus, they can be translated into augmented error.*

5 Regularized Linear Regression

Problem 7. ***d***

```
[62] import numpy as np

     train_file = np.loadtxt('features.train')
     test_file = np.loadtxt('features.test')

     x_train = train_file[:, 1:]
     y_train = train_file[:,0]

     x_test = test_file[:, 1:]
     y_test = test_file[:,0]
```

```
[63] def binary_class(y, n):
       bin_class = -np.ones(len(y))
       bin_class[y == n] = 1
       return bin_class
```

```
[64] def transform_1(x):
       transform = np.ones((x.shape[0],1))
       return np.hstack((transform, x))
```

```
[65] def suedo_regression(x_train, lambd):
       val1 = np.dot(x_train.T, x_train)
       val2 = lambd * np.identity(x_train.shape[1])
       return np.dot(np.linalg.inv(val1 + val2), x_train.T)

     def calc_error(w, x_, y_):
       N = 0
       for i in range(x_.shape[0]):
         N += max(0, np.sign(-np.dot(w.T, x_[i])*y_[i]))
       return N/float(x_.shape[0])
```

```
5 versus all: E_in = 0.07625840076807022
6 versus all: E_in = 0.09107118365107666
7 versus all: E_in = 0.08846523110684405
8 versus all: E_in = 0.07433822520916199
9 versus all: E_in = 0.08832807570977919
```

Problem 8. ***b***

```
[67] def transform_2(x):
       x_t = np.ones((x.shape[0],1))
       return np.hstack((x_t, x[:,[0]],x[:,[1]], x[:,[1]]*x[:,[0]], x[:,[0]]**2, x[:,[1]]**2))
```

```
[68] lambd = 1.0
     z_train = transform_2(x_train)
     z_test = transform_2(x_test)
     for n in [0,1,2,3,4]:
       yt_train = binary_class(y_train, n)
       yt_test = binary_class(y_test, n)
       suedo_reg = suedo_regression(z_train, lambd)
       w_reg = np.dot(suedo_reg, yt_train)
       err_test_reg = calc_error(w_reg, z_test, yt_test)
       print(f'{n} versus all: E_in = {err_test_reg}')
```

```
0 versus all: E_in = 0.10662680617837568
1 versus all: E_in = 0.02192326856003986
2 versus all: E_in = 0.09865470852017937
3 versus all: E_in = 0.08271051320378675
4 versus all: E_in = 0.09965122072745392
```

Problem 9. ***e***

As shown in the output, the out of sample Error improves for 5 versus all, but by less than 5%.

```
[69] lambd = 1.0

     xt_train = transform_1(x_train)
     xt_test = transform_1(x_test)

     z_train = transform_2(x_train)
     z_test = transform_2(x_test)

     for n in range(10):
       yt_train = binary_class(y_train, n)
       yt_test = binary_class(y_test, n)

       x_reg = suedo_regression(xt_train, lambd)
       w_reg_x = np.dot(x_reg, yt_train)

       z_reg = suedo_regression(z_train, lambd)
       w_reg_z = np.dot(z_reg, yt_train)

       xtrain_error = calc_error(w_reg_x, xt_train, yt_train)
       ztrain_error = calc_error(w_reg_z, z_train, yt_train)

       xtest_error = calc_error(w_reg_x, xt_test, yt_test)
       ztest_error = calc_error(w_reg_z, z_test, yt_test)

       diff = float(xtest_error - ztest_error)
       print(f'{n} versus all improvement = {diff}')
```

```
    0 versus all improvement = 0.008470353761833582
    1 versus all improvement = 0.0004982561036372679
    2 versus all improvement = 0.0
    3 versus all improvement = 0.0
    4 versus all improvement = 0.0
    5 versus all improvement = 0.0004982561036372679
    6 versus all improvement = 0.0
    7 versus all improvement = 0.0
    8 versus all improvement = 0.0
    9 versus all improvement = 0.0
```

Problem 10. ***a***

Overfitting occurs.

```
[70] zc_train = transform_2(x_train[np.logical_or(y_train==1, y_train==5), :])
     yc_train = binary_class(y_train[np.logical_or(y_train==1, y_train==5)], 1)
     zc_test = transform_2(x_test[np.logical_or(y_test==1, y_test==5), :])
     yc_test = binary_class(y_test[np.logical_or(y_test==1, y_test==5)], 1)

     for lambd in [0.01, 1.0]:
       suedo_reg = suedo_regression(zc_train, lambd)
       w_reg = np.dot(suedo_reg, yc_train)

       err_train_reg = calc_error(w_reg, zc_train, yc_train)
       err_test_reg = calc_error(w_reg, zc_test, yc_test)

       print(f'lambda = {lambd}: E_in: {err_train_reg}, E_out: {err_test_reg}')

     lambda = 0.01: E_in: 0.004484304932735426, E_out: 0.02830188679245283
     lambda = 1.0: E_in: 0.005124919923126201, E_out: 0.025943396226415096
```

6 Support Vector Machines

Problem 11. ***c***

Plotting the points, we see that z_2 does not make any difference to the classification. This means that w_2 is 0. Using geometry we find $w_1 = 1$ and b = -0.5.

```
[71] x = np.array([[1,0],[0,1],[0,-1],[-1,0],[0,2],[0,-2],[-2,0]])
     y = np.array([-1, -1, -1, 1, 1, 1, 1])

     def z_transform(x_i):
       return [x_i[1]**2 - 2 * x_i[0] - 1, x_i[0]**2 - 2 * x_i[1] + 1]

     z = np.apply_along_axis(z_transform, 1, x)
```

Problem 12. ***c***

```
[73] import numpy as np
     from qpsolvers import solve_qp

     def get_svms_quadp(x, y):
       y = y.reshape(-1, 1)
       m = x.shape[0]
       K = ((np.dot(x, x.T) + 1) ** 2).astype(float)

       P = (np.outer(y, y) * K).astype(float)
       q = -np.ones((m, 1), dtype=float)
       G = -np.eye(m, dtype=float)
       h = np.zeros((m, 1), dtype=float)
       A = y.T.astype(float)
       b = np.zeros((1, 1), dtype=float)

       alpha = solve_qp(P, q, G, h, A, b, solver="cvxopt")
       support_vector_indices = np.where(alpha > 1e-5)[0]
       support_vectors = z[support_vector_indices]
       return len(support_vector_indices)

     num_svms = get_svms_quadp(x, y)
     print(f'SVMs: {num_svms}')

     SVMs: 5
```

7 Radial Basis Functions

Problem 13. *a*

```
[74] import random
     from sklearn import svm

     def func(x_i):
       return np.sign(x_i[1] - x_i[0] + 0.25*np.sin(np.pi*x_i[0]))

     def points(n):
       x = np.array([[random.uniform(-1,1),random.uniform(-1,1)] for i in range(n)])
       y = []
       for i in x:
         y.append(func(i))
       return x, np.array(y)

     def get_freq():
       clf = svm.SVC(kernel='rbf', coef0=1, C=1e10, gamma=1.5)
       fail_score = 0
       for i in range(1000):
         x,y = points(100)
         clf.fit(x,y)
         E_in = 1.0 - clf.score(x,y)
         if E_in != 0:
           fail_score += 1
       result = fail_score/float(1000)
       return result

     frequency = get_freq()
     print(f'Frequency: {frequency}')

     Frequency: 0.0
```

Problem 14. *e*

```
[75] from sklearn.cluster import KMeans
```

```
[76] def rbf_model(x, center, gamma):
       return np.exp(-gamma * np.sum((x - center)**2))

     def rbf_reg(x, centers, gamma):
       z = np.empty((x.shape[0], centers.shape[0] + 1))
       z[:,0] = np.ones(x.shape[0])
       for i in range(centers.shape[0]):
         z[:, i+1] = np.apply_along_axis(rbf_model, 1, x, centers[i], gamma)
       return z

     def calc_w(z, y):
       m = np.matrix(np.dot(z.T, z))
       b = np.dot(np.dot(m.getI(), z.T), y)
       return b.getA()[0,:]
```

```
[77] def get_freq():
       gamma = 1.5
       K = 9
       n = 100
       ker_Eout = []
       reg_Eout = []
       for i in range(100):
         x, y = points(n)
         x_test, y_test = points(1000)
         clf = svm.SVC(kernel='rbf', coef0=1, C=1e10, gamma=gamma)
         clf.fit(x,y)
         while clf.score(x,y) != 1.0:
           x,y = points(n)
           clf.fit(x,y)

         ker_Eout.append(1.0 - clf.score(x_test,y_test))
         kmeans = KMeans(n_clusters=K, random_state=0, n_init="auto").fit(x)
         kmeans_centers = kmeans.cluster_centers_

         z = rbf_reg(x, kmeans_centers, gamma)
         w = calc_w(z, y)
         z_test = rbf_reg(x_test, kmeans_centers, gamma)
         y_ = np.dot(z_test, w)
         reg_Eout.append(np.sum(y_*y_test < 0) / float(y_test.shape[0]))

       diff = np.array(ker_Eout) - np.array(reg_Eout)
       result = sum(i < 0 for i in diff)/float(len(diff))
       return result

     frequency = get_freq()
     print(f'Frequency: {frequency}')

     Frequency: 0.86
```

Problem 15. *d*

```
[78] def get_freq():
       gamma = 1.5
       K = 12
       n = 100
       ker_Eout = []
       reg_Eout = []
       for i in range(100):
         x, y = points(n)
         x_test, y_test = points(1000)
         clf = svm.SVC(kernel='rbf', coef0=1, C=1e10, gamma=gamma)
         clf.fit(x,y)
         while clf.score(x,y) != 1.0:
           x,y = points(n)
           clf.fit(x,y)
         ker_Eout.append(1.0 - clf.score(x_test,y_test))
         kmeans = KMeans(n_clusters=K, random_state=0, n_init="auto").fit(x)
         kmeans_centers = kmeans.cluster_centers_

         z = rbf_reg(x, kmeans_centers, gamma)
         w = calc_w(z, y)
         z_test = rbf_reg(x_test, kmeans_centers, gamma)
         y_ = np.dot(z_test, w)
         reg_Eout.append(np.sum(y_*y_test < 0) / float(y_test.shape[0]))

       diff = np.array(ker_Eout) - np.array(reg_Eout)
       result = sum(i < 0 for i in diff)/float(len(diff))
       return result

     frequency = get_freq()
     print(f'Frequency: {frequency}')

     Frequency: 0.84
```

Problem 16. *d*

```
[79] def get_freq():
       gamma = 1.5
       reg_Ein = []
       reg_Eout = []
       Ks = [9, 12]

       for K in Ks:
         for i in range(100):
           x,y = points(100)
           x_test, y_test = points(1000)
           kmeans = KMeans(n_clusters=K, random_state=0, n_init="auto").fit(x)
           kmeans_centers = kmeans.cluster_centers_
           z = rbf_reg(x, kmeans_centers, gamma)
           w = calc_w(z, y)
           y_ = np.dot(z, w)
           reg_Ein.append(np.sum(y_*y < 0) / float(y_test.shape[0]))
           z_test = rbf_reg(x_test, kmeans_centers, gamma)
           y_ = np.dot(z_test, w)
           reg_Eout.append(np.sum(y_*y_test < 0) / float(y_test.shape[0]))

       reg_Ein = np.array(reg_Ein).reshape(2,100).T
       reg_Eout = np.array(reg_Eout).reshape(2,100).T

       return reg_Ein, reg_Eout

     reg_Ein, reg_Eout = get_freq()

     # a. E_in goes down but E_out goes up
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] > 0) & (reg_Eout[:,0]-reg_Eout[:,1] < 0))
     print(f'E_in down, E_out up: {result}')

     # b. E_in goes up but E_out goes down
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] < 0) & (reg_Eout[:,0]-reg_Eout[:,1] > 0))
     print(f'E_in up, E_out down: {result}')

     # c. Both E_in and E_out go up
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] < 0) & (reg_Eout[ :,0]-reg_Eout[:,1] < 0))
     print(f'E_in, E_out up: {result}')

     # d. Both E_in and E_out go down
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] > 0) & (reg_Eout [:,0]-reg_Eout[:,1] > 0))
     print(f'E_in, E_out down: {result}')

     # e. E_in and E_out remain the same
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] == 0) & ( reg_Eout[:,0]-reg_Eout[:,1] == 0))
     print(f'E_in, E_out remain same: {result}')

     E_in down, E_out up: 11
     E_in up, E_out down: 12
     E_in, E_out up: 10
     E_in, E_out down: 50
     E_in, E_out remain same: 0
```

Problem 17. *c*

```
[80] K = 9
     reg_Ein = []
     reg_Eout = []
     gammas = [1.5, 2]

     for gamma in gammas:
       for i in range(100):
         x,y = points(100)
         x_test, y_test = points(1000)
         kmeans = KMeans(n_clusters=K, random_state=0, n_init="auto").fit(x)
         kmeans_centers = kmeans.cluster_centers_
         z = rbf_reg(x, kmeans_centers, gamma)
         w = calc_w(z, y)
         y_ = np.dot(z, w)
         reg_Ein.append(np.sum(y_*y < 0) / float(y_test.shape[0]))
         z_test = rbf_reg(x_test, kmeans_centers, gamma)
         y_ = np.dot(z_test, w)
         reg_Eout.append(np.sum(y_*y_test < 0) / float(y_test.shape[0]))

     reg_Ein = np.array(reg_Ein).reshape(2,100).T
     reg_Eout = np.array(reg_Eout).reshape(2,100).T

     # a. E_in goes down but E_out goes up
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] > 0) & (reg_Eout[:,0]-reg_Eout[:,1] < 0))
     print(f'E_in down, E_out up: {result}')

     # b. E_in goes up but E_out goes down
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] < 0) & (reg_Eout[:,0]-reg_Eout[:,1] > 0))
     print(f'E_in up, E_out down: {result}')

     # c. Both E_in and E_out go up
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] < 0) & (reg_Eout[ :,0]-reg_Eout[:,1] < 0))
     print(f'E_in, E_out up: {result}')

     # d. Both E_in and E_out go down
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] > 0) & (reg_Eout [:,0]-reg_Eout[:,1] > 0))
     print(f'E_in, E_out down: {result}')

     # e. E_in and E_out remain the same
     result = np.sum((reg_Ein[:,0]-reg_Ein[:,1] == 0) & ( reg_Eout[:,0]-reg_Eout[:,1] == 0))
     print(f'E_in, E_out remain same: {result}')
```

```
E_in down, E_out up: 24
E_in up, E_out down: 20
E_in, E_out up: 27
E_in, E_out down: 15
E_in, E_out remain same: 1
```

Problem 18. ***a***

```
[81] def get_freq():
       K = 9
       n = 100
       reg_Ein = []
       for i in range(100):
         x, y = points(n)
         kmeans = KMeans(n_clusters=K, random_state=0, n_init="auto").fit(x)
         kmeans_centers = kmeans.cluster_centers_
         z = rbf_reg(x, kmeans_centers, 1.5)
         w = calc_w(z, y)
         y_ = np.dot(z, w)
         reg_Ein.append(np.sum(y_*y < 0) / float(y_test.shape[0]))

       result = sum(E_in == 0 for E_in in reg_Ein)/float(len(reg_Ein))
       return result

     frequency = get_freq()
     print(f'Frequency: {frequency}')
```

```
Frequency: 0.02
```

8 Bayesian Priors

Problem 19. ***b***

We are assuming the prior that $P(h = f)$ is uniform over $f \in [0, 1]$. We also know that the posterior is directly proportional to the likelihood times the prior. The likelihood increases linearly over the interval [0,1] since it is given that f is a constant over this interval. Therefore, we know that the posterior also increases linearly over [0,1].

9 Aggregation

Problem 20. ***c***

If given two learned hypotheses, g_1 and g_2, and construct the average hypothesis g defined by $g(x) = \frac{1}{2}(g_1(x) + g_2(x))$ for all x, it means that both g_1 and g_2 contribute the same amount. The Mean Squared Error takes the sum of the difference squared. It follows that the average of the hypotheses residuals squared will be higher than the square of the average of the residuals. Thus, we know that $E_{out}(g)$ cannot be worse than the average of $E_{out}(g_1)$ and $E_{out}(g_2)$.