HW7:

(Q5) [D]

for problem 1, we chose the model with $K=6$ based on the code which yielded an out-of-sample classification error of 0.084. for problem 3, we chose the model with $K=6$ based on the code which yielded an out-of-sample classification error of 0.192. Hence the correct answer is [D] bcz it is the closest.

(Q7) With leave-one-out Cross-Validation, we leave one out (x_n, y_n) and calculate h^- using the remaining.

leaving out $(-1,0)$ and fit using $(1,0)$ & $(0,1)$

$g_0^{(n)}(x) = \frac{1}{2}$

$g_1^{(n)}(x) = \frac{x+1}{2+1}$

leaving out $(0,1)$ and fit using $(1,0)$ & $(-1,0)$

$g_0^{(n)}(x) = 0$

$g_1^{(n)}(x) = 0$

leaving out $(1,0)$ and fit using $(-1,0)$ & $(0,1)$

$g_0^{(n)}(x) = \frac{1}{2}$

$g_1^{(n)}(x) = \frac{x-1}{2-1}$

Now summing up squared error for $h_0(x)$:

$(\frac{1}{2}-0)^2 + (0-1)^2 + (\frac{1}{2}-0)^2 = \frac{1}{4} + 1 + \frac{1}{4} = \frac{6}{4} = \boxed{\frac{3}{2}}$

Now summing up squared error for $h_1(x)$.

$\left(\frac{-1-1}{2-1} - 0\right)^2 + (0-1)^2 + \left(\frac{1+1}{2+1} - 0\right)^2 = \frac{4}{(2-1)^2} + 1 + \frac{4}{(2+1)^2}$

Now Se the two errors equate & solve.

$\frac{3}{2} = \frac{4}{(2-1)^2} + \frac{4}{(2+1)^2} + 1 \Rightarrow \frac{1}{2} = \frac{4}{(2-1)^2} + \frac{4}{(2+1)^2}$;

$P = \frac{1}{2}\sqrt{9+4\sqrt{6}}$ hence the correct answer is [C] bcz it's equal to $\sqrt{9+4\sqrt{6}}$

(Q1) [D]

hw7.py

```python
import points
import numpy

in_data = points.get_sample_points()
out_data = points.get_test_points()

def transform(x1, x2, k):
    transformation = [1, x1, x2, x1 ** 2, x2 ** 2, x1 * x2, abs(x1 - x2), abs(x1 + x2)]
    return transformation[:k+1]

def get_weight(k):
    data = []
    scores = []

    for point in in_data:
        data.append(transform(point[0], point[1], k))
        scores.append(point[2])
    data = data[:25]
    scores = scores[:25]
    w = numpy.dot(numpy.linalg.pinv(data), scores)
    return (w, scores, data)

def Eout(k):
    data = []
    scores = []
    for i in range(25, 35):
        data.append(transform(in_data[i][0], in_data[i][1], k))
        scores.append(in_data[i][2])

    summ_error = 0
    w = get_weight(k)[0]
    for d, s in zip(data, scores):
        if (numpy.sign(numpy.dot(d, w)) != s):
            summ_error += 1
    return summ_error/len(scores)

print("k = 3", Eout(3))
print("k = 4", Eout(4))
print("k = 5", Eout(5))
print("k = 6", Eout(6))
print("k = 7", Eout(7))
```

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
k = 3 0.3
k = 4 0.5
k = 5 0.2
k = 6 0.0
k = 7 0.1
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
k = 3 0.3
k = 4 0.5
k = 5 0.2
k = 6 0.0
k = 7 0.1
thierno@Thiernos-MacBook-Pro CS  156a %
```

⇐ According to the code the smallest is when $K=6$ hence the correct answer is [D]

(Q6) [D]

hw7.py > find_bias

```python
import points
import numpy
import random

in_data = points.get_sample_points()
out_data = points.get_test_points()

def transform(x1, x2, k):...

def get_weight(k):...

def Eout(k):...

def Eout_test(k):...

def validation_bias():
    e1 = random.uniform(0,1)
    e2 = random.uniform(0,1)
    e = min(e1, e2)
    return (e1, e2, e)

def find_bias(N):
    e1_av = 0
    e2_av = 0
    e_av = 0
    for i in range(N):
        curr = validation_bias()
        e1_av += curr[0]
        e2_av += curr[1]
        e_av += curr[2]
    return (e1_av/N, e2_av/N, e_av/N)

print(find_bias(1000000))
```

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
(0.4997247888469846, 0.4995020310158439, 0.3330465337656185)
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
(0.5004996685939272, 0.4995699753133675, 0.3330917233766198)
thierno@Thiernos-MacBook-Pro CS  156a %
```

Ln 64, Col 25 Spaces: 4 UTF-8 LF Python 3.10.6 64-bit

According to the above code the correct answer is [d] as it is the closest.

(Q4) [d]

```python
import points
import numpy

in_data = points.get_sample_points()
out_data = points.get_test_points()

def transform(x1, x2, k):
    transformation = [1, x1, x2, x1 ** 2, x2 ** 2, x1 * x2, abs(x1 - x2), abs(x1 + x2)]
    return transformation[:k+1]

def get_weight(k):
    data = []
    scores = []

    for point in in_data:
        data.append(transform(point[0], point[1], k))
        scores.append(point[2])
    data = data[25:]
    scores = scores[25:]
    w = numpy.dot(numpy.linalg.pinv(data), scores)
    return (w, scores, data)

def Eout(k):
    ...

def Eout_test(k):
    data = []
    scores = []
    for point in out_data:
        data.append(transform(point[0], point[1], k))
        scores.append(point[2])

    summ_error = 0
    w = get_weight(k)[0]
    for d,s in zip(data, scores):
        if (numpy.sign(numpy.dot(d, w)) != s ):
            summ_error += 1
    return summ_error/len(scores)

print("k = 3", Eout_test(3))
print("k = 4", Eout_test(4))
print("k = 5", Eout_test(5))
print("k = 6", Eout_test(6))
print("k = 7", Eout_test(7))
```

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
k = 3 0.396
k = 4 0.388
k = 5 0.284
k = 6 0.192
k = 7 0.196
thierno@Thiernos-MacBook-Pro CS  156a %
```

⇐ According to the code the smallest is when k=6 hence the correct answer is [d]

(Q3) [d]

```python
import points
import numpy

in_data = points.get_sample_points()
out_data = points.get_test_points()

def transform(x1, x2, k):
    transformation = [1, x1, x2, x1 ** 2, x2 ** 2, x1 * x2, abs(x1 - x2), abs(x1 + x2)]
    return transformation[:k+1]

def get_weight(k):
    data = []
    scores = []

    for point in in_data:
        data.append(transform(point[0], point[1], k))
        scores.append(point[2])
    data = data[25:]
    scores = scores[25:]
    w = numpy.dot(numpy.linalg.pinv(data), scores)
    return (w, scores, data)

def Eout(k):
    data = []
    scores = []
    for i in range(0, 25):
        data.append(transform(in_data[i][0], in_data[i][1], k))
        scores.append(in_data[i][2])

    summ_error = 0
    w = get_weight(k)[0]
    for d, s in zip(data, scores):
        if (numpy.sign(numpy.dot(d, w)) != s):
            summ_error += 1
    return summ_error/len(scores)
print("k = 3", Eout(3))
print("k = 4", Eout(4))
print("k = 5", Eout(5))
print("k = 6", Eout(6))
print("k = 7", Eout(7))
```

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
k = 3 0.28
k = 4 0.36
k = 5 0.2
k = 6 0.08
k = 7 0.12
thierno@Thiernos-MacBook-Pro CS  156a %
```

⇐ According to the code the smallest is when k=6 hence the correct answer is [d]

Ln 52, Col 1 Spaces: 4 UTF-8 LF Python 3.10.6 64-bit

(Q2) [c]

```python
import points
import numpy

in_data = points.get_sample_points()
out_data = points.get_test_points()

def transform(x1, x2, k):
    transformation = [1, x1, x2, x1 ** 2, x2 ** 2, x1 * x2, abs(x1 - x2), abs(x1 + x2)]
    return transformation[:k+1]

def get_weight(k):
    data = []
    scores = []

    for point in in_data:
        data.append(transform(point[0], point[1], k))
        scores.append(point[2])
    data = data[25:]
    scores = scores[25:]
    w = numpy.dot(numpy.linalg.pinv(data), scores)
    return (w, scores, data)

def Eout(k):
    ...

def Eout_test(k):
    data = []
    scores = []
    for point in out_data:
        data.append(transform(point[0], point[1], k))
        scores.append(point[2])

    summ_error = 0
    w = get_weight(k)[0]
    for d,s in zip(data, scores):
        if (numpy.sign(numpy.dot(d, w)) != s ):
            summ_error += 1
    return summ_error/len(scores)

print("k = 3", Eout_test(3))
print("k = 4", Eout_test(4))
print("k = 5", Eout_test(5))
print("k = 6", Eout_test(6))
print("k = 7", Eout_test(7))
```

```
ValueError: shapes (4,) and (3,) not aligned: 4 (dim 0) != 3 (dim 0)
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
k = 3 0.42
k = 4 0.416
k = 5 0.188
k = 6 0.084
k = 7 0.072
thierno@Thiernos-MacBook-Pro CS  156a %
```

⇐ According to the code the smallest is when k=7 hence the correct answer is [c]

Ln 47, Col 48 Spaces: 4 UTF-8 LF Python 3.10.6 64-bit

(10) [b]

```python
import numpy
import random
from sklearn import svm
from sklearn.linear_model import Perceptron
def get_slope(x1, x2, y1, y2):
    slope = (y1 - y2) / (x1 - x2)
    y_intercept = y1 - (slope * x1)
    return (slope, y_intercept)
def classify(slope, b, x1, x2) :
    if ((slope * x1) + b) > x2:
        return -1
    else:
        return 1
def Eout(score_0, score_1):
    av = 0
    for s1, s2 in zip(score_0, score_1):
        if (numpy.sign(s1) != numpy.sign(s2)):
            av += 1
    return av/(len(score_0))
def make_points(x, bnNone, slopeNone):
    if (b == None):
        points = [random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1)]
        slope, b = get_slope(points[0], points[1], points[2], points[3])
    data = []
    score = []
    for i in range(x):
        point = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
        data.append(point)
        score.append(classify(slope, b, point[0], point[1]))
    return (data, score, b, slope)
def do_it(runs):
    svm_better = 0
    support_v = 0
    for i in range(runs):
        data, score, b, slope = make_points(100, None, None)
        while (len(set(score)) == 1):
            data, score, b, slope = make_points(100, None, None)
        pla = Perceptron(fit_intercept=False, max_iter=1000)
        pla.fit(data, score)
        classifier = svm.SVC(C=numpy.Infinity, kernel='linear')
        classifier.fit(data, score)
        data_o, score_o, b, slope = make_points(1000, b, slope)
        score_1 = pla.predict(data_o)
        score_2 = classifier.predict(data_o)
        support_v += numpy.sum(classifier.n_support_)
        if (Eout(score_o, score_1) > Eout(score_o, score_2)):
            svm_better += 1
    return(svm_better / runs, support_v/runs)
result = do_it(1000)
print("Number of support Vectors:  ", result[1])
```

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
Number of support Vectors:  2.968
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence according to the above code the correct answer is [b] as 3 is the closest answer for N=100;

(9) [d]

```python
def do_it
import numpy
import random
from sklearn import svm
from sklearn.linear_model import Perceptron
def get_slope(x1, x2, y1, y2):
    slope = (y1 - y2) / (x1 - x2)
    y_intercept = y1 - (slope * x1)
    return (slope, y_intercept)
def classify(slope, b, x1, x2) :
    if ((slope * x1) + b) > x2:
        return -1
    else:
        return 1
def Eout(score_0, score_1):
    av = 0
    for s1, s2 in zip(score_0, score_1):
        if (numpy.sign(s1) != numpy.sign(s2)):
            av += 1
    return av/(len(score_0))
def make_points(x, bnNone, slopeNone):
    if (b == None):
        points = [random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1)]
        slope, b = get_slope(points[0], points[1], points[2], points[3])
    data = []
    score = []
    for i in range(x):
        point = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
        data.append(point)
        score.append(classify(slope, b, point[0], point[1]))
    return (data, score, b, slope)
def do_it(runs):
    svm_better = 0
    support_v = 0
    for i in range(runs):
        data, score, b, slope = make_points(100, None, None)
        while (len(set(score)) == 1):
            data, score, b, slope = make_points(100, None, None)
        pla = Perceptron(fit_intercept=False, max_iter=1000)
        pla.fit(data, score)
        classifier = svm.SVC(C=numpy.Infinity, kernel='linear')
        classifier.fit(data, score)
        data_o, score_o, b, slope = make_points(1000, b, slope)
        score_1 = pla.predict(data_o)
        score_2 = classifier.predict(data_o)
        support_v += numpy.sum(classifier.n_support_)
        if (Eout(score_o, score_1) > Eout(score_o, score_2)):
            svm_better += 1
    return(svm_better / runs, support_v/runs)
result = do_it(1000)
print("SVM was better this many times:  ", result[0])
```

```
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
SVM was better this many times:  0.734
thierno@Thiernos-MacBook-Pro CS  156a %
```

Hence by the above code, the correct answer is [d] as 65% is closest.

(8) [c]

```python
import numpy
import random
from sklearn import svm
from sklearn.linear_model import Perceptron
def get_slope(x1, x2, y1, y2):
    slope = (y1 - y2) / (x1 - x2)
    y_intercept = y1 - (slope * x1)
    return (slope, y_intercept)
def classify(slope, b, x1, x2) :
    if ((slope * x1) + b) > x2:
        return -1
    else:
        return 1
def Eout(score_0, score_1):
    av = 0
    for s1, s2 in zip(score_0, score_1):
        if (numpy.sign(s1) != numpy.sign(s2)):
            av += 1
    return av/(len(score_0))
def make_points(x, bnNone, slopeNone):
    if (b == None):
        points = [random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1), random.uniform(-1, 1)]
        slope, b = get_slope(points[0], points[1], points[2], points[3])
    data = []
    score = []
    for i in range(x):
        point = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
        data.append(point)
        score.append(classify(slope, b, point[0], point[1]))
    return (data, score, b, slope)
def do_it(runs):
    svm_better = 0
    support_v = 0
    for i in range(runs):
        data, score, b, slope = make_points(10, None, None)
        while (len(set(score)) == 1):
            data, score, b, slope = make_points(10, None, None)
        pla = Perceptron(fit_intercept=False, max_iter=1000)
        pla.fit(data, score)
        classifier = svm.SVC(C=numpy.Infinity, kernel='linear')
        classifier.fit(data, score)
        data_o, score_o, b, slope = make_points(1000, b, slope)
        score_1 = pla.predict(data_o)
        score_2 = classifier.predict(data_o)
        support_v += numpy.sum(classifier.n_support_)
        if (Eout(score_o, score_1) > Eout(score_o, score_2)):
            svm_better += 1
    return(svm_better / runs, support_v/runs)
result = do_it(1000)
print("SVM was better this many times:  ", result[0])
```

```
SVM was better this many times:  0.636
thierno@Thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/hw7.py"
SVM was better this many times:  0.64
```

Hence according to the above Code, the correct answer is [c] as 65% is the closest.