Validation

```
import math
import numpy


def read_file(filename):
    file = open(filename)
    data = []
    for line in file:
        data.append(list(map(float, filter(bool, line.split()))))
    return data

def phi(data, k):
    transformed_data = []
    for x1, x2, y in data:
        transformed_data.append(transform(x1, x2, y, k))
    return transformed_data

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

def g(weight, x):
    return numpy.sign(numpy.dot(weight, x))

def split(data, N=25):
    return (data[:N], data[N:])

def w_lin(data):
    Z, y = zip(*data)
    return numpy.dot(numpy.linalg.pinv(Z), y)

def error(weight, data):
    sum_error = 0
    Z, Y = zip(*data)
    for i in range(len(Y)):
        hypothesis = g(weight, Z[i])
        if (hypothesis != Y[i]):
            sum_error += 1
    return sum_error/float(len(Y))

def experiment(k, N=25, reversed=False):
    input_data = read_file("in.txt")
    output_data = read_file("out.txt")
    phi_in = phi(input_data, k)
    phi_out = phi(output_data, k)
    if reversed:
        validation, training = split(phi_in, N)
    else:
        training, validation = split(phi_in, N)
    linear_weight = w_lin(training)

    input_error = error(linear_weight, training)
    validation_error = error(linear_weight, validation)
    output_error = error(linear_weight, phi_out)
    return input_error, validation_error, output_error


print("(In sample, Validation, Out of Sample): ")
for k in range(3, 8):
    print(f'k: {k} -- {experiment(k, N=25, reversed=False)}')
```

```
(In sample, Validation, Out of Sample):
k: 3 -- (0.44, 0.3, 0.42)
k: 4 -- (0.32, 0.5, 0.416)
k: 5 -- (0.08, 0.2, 0.188)
k: 6 -- (0.04, 0.0, 0.084)
k: 7 -- (0.04, 0.1, 0.072)
```

1. d. For k = 6, the classification error is the smallest on the validation set.

2. e. For k = 7, the out of sample classification error is smallest.

```
print("(In sample, Validation, Out of Sample): ")
for k in range(3, 8):
    print(f'k: {k} -- {experiment(k, N=25, reversed=True)}')
```

```
(In sample, Validation, Out of Sample):
k: 3 -- (0.4, 0.28, 0.396)
k: 4 -- (0.3, 0.36, 0.388)
k: 5 -- (0.2, 0.2, 0.284)
k: 6 -- (0.0, 0.08, 0.192)
k: 7 -- (0.0, 0.12, 0.196)
```

3. d. For k = 6, the classification error is the smallest on th validation set.

4. d. For k = 6, the out of sample classification error is the smallest.

5. b. The closest values are 0.1, 0.2

Validation Bias

```
import random
e1 = [random.uniform(0,1) for x in range(10000)]
e2 = [random.uniform(0,1) for x in range(10000)]
e = [min(x,y) for x, y in zip(e1, e2)]

e1_avg = sum(e1)/len(e1)
e2_avg = sum(e2)/len(e2)
e_avg = sum(e)/len(e)

print(f'e1: {e1_avg}, e2: {e2_avg}, e: {e_avg}')
```

```
e1: 0.5023633693791428, e2: 0.49839583595692594, e: 0.33394086689856006
```

6. d. The expected value for two independent random variables, distributed uniformly over [0,1] is 0.5 for both. The minimum of these random variables would be 1/3. 0.4 is closest.

Cross Validation

```
import numpy as np


def small_transform(data, k):
  output = np.zeros((len(data), k + 1))
  for i in range(len(data)):
    x = data[i]
    output[i] = [1, x][:k + 1]
  return output


def lin_reg(X, y):
  inversed = np.linalg.inv(X.transpose().dot(X))
  w = inversed.dot(X.transpose()).dot(y)
  return w


ps = [math.sqrt(math.sqrt(3) + 4), math.sqrt(math.sqrt(3) - 1), math.sqrt(9 + 4 * math.sqrt(6)), math.sqrt(9 - math.sqrt(6))]

for p in ps:
  data = [[-1, 0], [p, 1], [1, 0]]
  h0 = 0
  h1 = 0
  for i in range(3):
    train_data = np.array(data[:i] + data[i+1:])
    test_data = np.array([data[i]])
    X_train, y_train = train_data[:, 0], train_data[:, 1]
    X_test, y_test = test_data[:, 0], test_data[:, 1]
```

```
        X_train_0 = small_transform(X_train, 0)
        X_test_0 = small_transform(X_test, 0)
        w = lin_reg(X_train_0, y_train)
        error0 = (X_test_0.dot(w) - y_test) ** 2
        h0 += error0[0]

        X_train_1 = small_transform(X_train, 1)
        X_test_1 = small_transform(X_test, 1)
        w = lin_reg(X_train_1, y_train)
        error1 = (X_test_1.dot(w) - y_test) ** 2
        h1 += error1[0]

    h0 /= len(data)
    h1 /= len(data)
    print(f"p = {p}:")
    print(f"h0: {h0}")
    print(f"h1: {h1}\n")
```

```
p = 2.3941701709713277:
h0: 0.5
h1: 1.1350433676859402

p = 0.8555996771673521:
h0: 0.5
h1: 64.66494840795316

p = 4.335661307243996:
h0: 0.5
h1: 0.5

p = 2.5593964634688433:
h0: 0.5
h1: 0.9868839293305474
```

7. c. The linear model would use an equation in the form of y = mx + b. The error would be the mean squared error of the equation and the point that was not included. The process is repeated for all three points. The constant model would use a y = b equation where b is calculated as the average of two points. This process is also repeated three times for the three points. After plugging in all answer choices, c was the same for both.

PLA vs SVM

```
import numpy as np
import random as rnd
import matplotlib.pyplot as plt
from sklearn import svm
%matplotlib inline


def line():
    [x1,x2,y1,y2] = [rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0)]
    xA,yA,xB,yB = [rnd.uniform(-1, 1) for i in range(4)]
    w = np.array([x2*y1-y2*x1, y2-y1, x1-x2])
    w_norm = np.array([1, -w[1]/w[2], -w[0]/w[2]])
    return w, w_norm


def pts(n, d, w=None, w_norm=None):
    if w is None:
        w, w_norm = line()
    y = [1]
    while len(set(y)) <= 1:
        d_ = np.random.uniform(-1.0, 1.0,(d,n))
        x_ = np.append(np.ones(n), d_).reshape((d+1,n))
        y = np.sign(np.dot(w.T,x_))
        d_ = np.append(x_, y).reshape((d+2,n))
    return x_, y, w, d_, w_norm


def get_pt(y_, y):
    mc_pts = []
    for i in range(len(y)):
```

```
        if y_[i] != y[i]:
            mc_pts.append(i)
    try:
        index = rnd.choice(mc_pts)
    except IndexError:
        index = 0

    return index, len(mc_pts)

def update(xi, yi_, w_):
    return w_ + yi_ * xi


def pre_process(n, d):
    x_, y, w, d_, w_n = pts(n,d)
    return x_, y, w, d_, w_n


def pla(x_, y):
    w_ = np.zeros(3)
    y_ = np.sign(np.dot(w_.T,x_))

    while np.array_equal(y, y_) != True:
        index, total_mc_pts= get_pt(y_,y)
        w_ = update(x_[:,index], y[index], w_)
        y_ = np.sign(np.dot(w_.T, x_))

    w_n = np.array([1, -w_[1]/w_[2], -w_[0]/w_[2]])

    return i, w_n, w_


pla_disagreement = []
svm_disagreement = []
sv = []
n = 100
d = 2

for i in range(1000):
    x_, y, w, d_, w_n = pre_process(n, d)
    _, w_n_, w_ = pla(x_, y)
    clf = svm.SVC(C=1000000, kernel='linear')
    clf.fit(x_[1:].T, y)

    x_, y, _, _, _ = pts(10000, d, w, w_n)
    y_ = np.sign(np.dot(w_.T,x_))
    zzz, nmc = get_pt(y_, y)

    pla_disagreement.append(nmc)

    y_ = clf.predict(x_[1:].T)
    zzz, nmc = get_pt(y_, y)

    svm_disagreement.append(nmc)

    sv.append(len(clf.support_vectors_))

diff = np.array(svm_disagreement) - np.array(pla_disagreement)
percentage = sum(1 for number in diff if number < 0)/float(len(diff))

print(f'percentage improvement: {percentage}')
```

```
percentage improvement over pla: 0.644
```

8. c

```
from sklearn import svm

pla_disagreement = []
svm_disagreement = []
n = 100
d = 2

for i in range(1000):
    x_, y, w, d_, w_n = pre_process(n, d)
    _, w_n_, w_ = pla(x_, y)
```

```
    clf = svm.SVC(C=1000000, kernel='linear')
    clf.fit(x_[1:].T, y)

    x_, y, _, _, _ = pts(10000, d, w, w_n)
    y_ = np.sign(np.dot(w_.T,x_))
    zzz, nmc = get_pt(y_, y)

    pla_disagreement.append(nmc)

    y_ = clf.predict(x_[1:].T)
    zzz, nmc = get_pt(y_, y)

    svm_disagreement.append(nmc)

diff = np.array(svm_disagreement) - np.array(pla_disagreement)
percentage = sum(1 for number in diff if number < 0)/float(len(diff))

print(f'percentage improvement: {percentage}')
```

```
percentage improvement over PLA: 0.624
```

9. d

```
avg = sum(sv)/float(len(sv))
print(f'average support vectors: {avg}')
```

```
average support vectors: 2.997
```

10. b