CS/CNS/EE 156a

Homework 7

Sung Hoon Choi

(Last name: Choi)

1. Answer: [d]

The classification errors I got were:

k=3 Error: 0.3

k=4 Error: 0.5

k=5 Error: 0.2

k=6 Error: 0 (The smallest)

k=7 Error: 0.1

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 1
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                              # Format: (x1,x2,label)
                              #            ...
                              #        (x1,x2,label)

def nonlinear_trans(x,lineNum,k):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row[0:k+1])
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                             #              ...
                                             #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)


for k in range(3,8):
    print("k = %d ---------------" % k)
```

```
    insample_data = extract_data("in.dta")
    insample_data_np = np.array(insample_data)  # turn it into a numpy array to use numpy library functions
    y = insample_data_np[:, 2]  # extract y-labels from the input data
    transformed_data = nonlinear_trans(insample_data, IN_DTA_NUM,k)
    transformed_data = np.array(transformed_data)
    weights = calculate_weight(transformed_data[:25,:],y[:25])
    calculate_error(weights,transformed_data[25:,:],y[25:],10)
```

---

2. Answer: [e]

The out-of-sample errors I got were:

k = 3 Error: 0.42

k = 4 Error: 0.416

k = 5 Error: 0.188

k = 6 Error: 0.084

k = 7 Error: 0.072 (The smallest)

Please refer to the code below for the derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 2
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                              # Format: (x1,x2,label)
                              #           ...
                              #        (x1,x2,label)

def nonlinear_trans(x,lineNum,k):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row[0:k+1])
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                              #               ...
                                              #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)


for k in range(3,8):
    print("k = %d ---------------" % k)
    insample_data = extract_data("in.dta")
    insample_data_np = np.array(insample_data)  # turn it into a numpy array to use numpy library functions
    y = insample_data_np[:, 2]  # extract y-labels from the input data
    transformed_data = nonlinear_trans(insample_data, IN_DTA_NUM,k)
```

```
    transformed_data = np.array(transformed_data)
    weights = calculate_weight(transformed_data[:25,:],y[:25])

    outsample_data = extract_data("out.dta")
    outsample_data_np = np.array(outsample_data)  # turn it into a numpy array to use numpy library functions
    y = outsample_data_np[:, 2]  # extract y-labels from the input data
    transformed_data = nonlinear_trans(outsample_data, OUT_DTA_NUM, k)
    calculate_error(weights, transformed_data, y, OUT_DTA_NUM)  # Eout - OutSample Error
```

3. Answer: [d]

The errors I got were:

k = 3 Error: 0.28

k = 4 Error: 0.36

k = 5 Error: 0.2

k = 6 Error: 0.08 (The smallest)

k = 7 Error: 0.12

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 3
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                              # Format: (x1,x2,label)
                              #           ...
                              #        (x1,x2,label)

def nonlinear_trans(x,lineNum,k):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row[0:k+1])
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                              #               ...
                                              #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)


for k in range(3,8):
    print("k = %d ---------------" % k)
    insample_data = extract_data("in.dta")
    insample_data_np = np.array(insample_data)  # turn it into a numpy array to use numpy library functions
    y = insample_data_np[:, 2]  # extract y-labels from the input data
```

```
    transformed_data = nonlinear_trans(insample_data, IN_DTA_NUM,k)
    transformed_data = np.array(transformed_data)
    weights = calculate_weight(transformed_data[25:35,:],y[25:35])
    calculate_error(weights, transformed_data[:25, :], y[:25], 25)
```

4. Answer: [d]

The errors I got were:

k = 3 Error: 0.396

k = 4 Error: 0.388

k = 5 Error: 0.284

k = 6 Error: 0.192 (The smallest)

k = 7 Error: 0.196

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 4
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                            # Format: (x1,x2,label)
                            #            ...
                            #        (x1,x2,label)

def nonlinear_trans(x,lineNum,k):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row[0:k+1])
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                            #              ...
                                            #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)


for k in range(3,8):
    print("k = %d ---------------" % k)
    insample_data = extract_data("in.dta")
    insample_data_np = np.array(insample_data)  # turn it into a numpy array to use numpy library functions
    y = insample_data_np[:, 2]  # extract y-labels from the input data
    transformed_data = nonlinear_trans(insample_data, IN_DTA_NUM,k)
    transformed_data = np.array(transformed_data)
    weights = calculate_weight(transformed_data[25:35,:],y[25:35])
```

```
    outsample_data = extract_data("out.dta")
    outsample_data_np = np.array(outsample_data)  # turn it into a numpy array to use numpy library functions
    y = outsample_data_np[:, 2]  # extract y-labels from the input data
    transformed_data = nonlinear_trans(outsample_data, OUT_DTA_NUM, k)
    calculate_error(weights, transformed_data, y, OUT_DTA_NUM)  # Eout - OutSample Error
```

5. Answer: [b]

In problem1, the chosen model was k=6 and its out-of-sample error was 0.084.

In problem3, the chosen model was k=6 and its out-of-sample error was 0.192.

Therefore, the closest values are (0.1, 0.2).

6. Answer: [d]

By generating 100000 pairs of data using Python, I got

E[e1] = 0.499

E[e2] = 0.500

E[e3] = 0.333

Therefore, the closest value is [d].

Please refer to the code below for derivation.

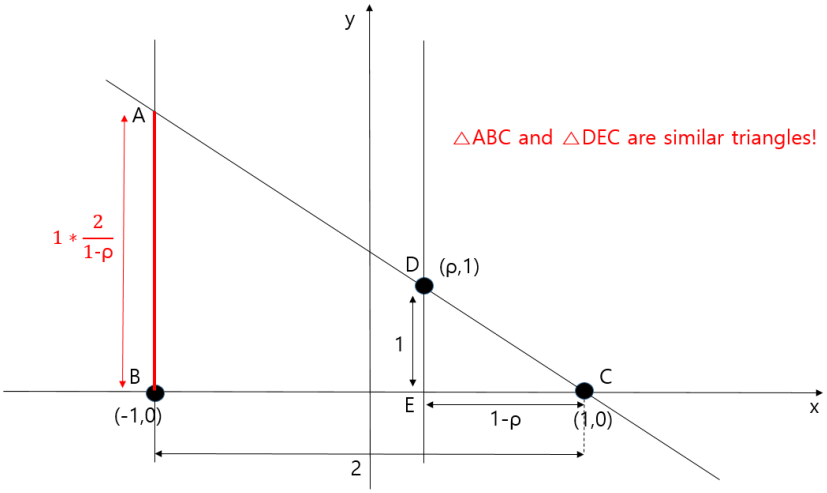
```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 6

import random

DATA_NUM = 100000
total_e1 = 0
total_e2 = 0
total_e = 0
for i in range(0,DATA_NUM):
    e1 = random.uniform(0,1)
    e2 = random.uniform(0,1)
    e = min(e1,e2)
    total_e1 = total_e1 + e1
    total_e2 = total_e2 + e2
    total_e = total_e + e

print("Expected e1: ", total_e1/DATA_NUM)
print("Expected e2: ", total_e2/DATA_NUM)
print("Expected e: ", total_e/DATA_NUM)
```

7. Answer: [c]

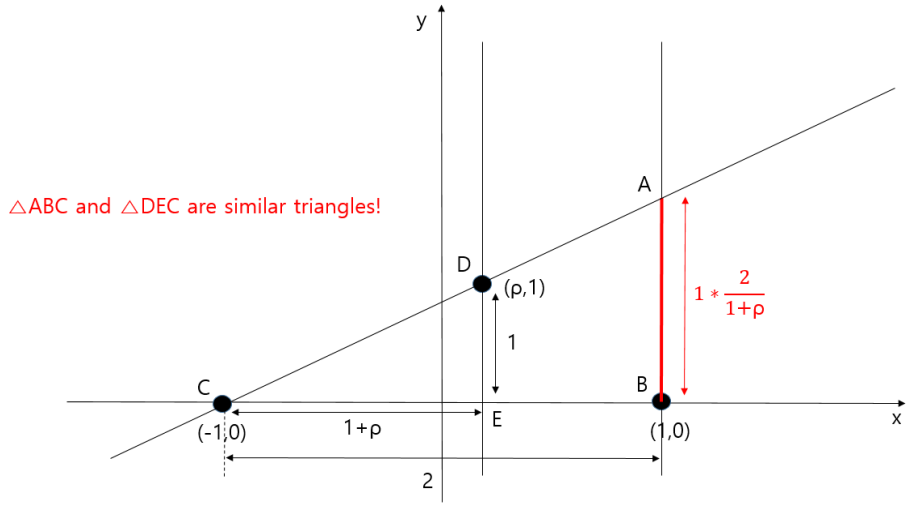
{$h_0(x)=b$}: First, let's leave out $(\rho, 1)$. Then, the equation of the line passing through (-1,0) and (1,0) would be y=0. Thus, the validation square error from y=0 to $(\rho, 1)$ would be 1. ($\because (1-0)^2=1$) Now, let's leave out (-1,0). Then, since we have (1,0) and $(\rho, 1)$, the line must located at the middle of this interval, giving the equation of $y=\frac{1}{2}$. Thus, the validation error from $y=\frac{1}{2}$ to (-1,0) would be $(\frac{1}{2}-0)^2=\frac{1}{4}$. In the same way, if we leave out (1,0) for validation, we get the same validation error of $\frac{1}{4}$ due to symmetry. Thus, the error for {$h_0(x)=b$} is

$$\frac{1}{3}*\left(1+\frac{1}{4}+\frac{1}{4}\right)=\frac{1}{2}$$

{$h_1(x)=ax+b$}: First, let's leave out $(\rho, 1)$. Then, it is exactly the same as what we've done for $h_0(x)=b$ above and we already know that its validation error is 1.($\because (1-0)^2=1$) Now, for the cases of leaving out (-1,0) and (1,0), I drew diagrams below to help explaining the geometry. If we leave out (-1,0) for validation, then we get the following estimation line:



Since △ABC and △DEC are similar triangles, we can find that $\overline{AB}=\frac{2}{1-\rho}$. Since we are using a squared-error measure, the error would be $(\frac{2}{1-\rho})^2=\frac{4}{(1-\rho)^2}$. Similarly, if we leave out (1,0), we get the following:



In this case, $\overline{AB}=\frac{2}{1+\rho}$ and the squared-error measure is $(\frac{2}{1+\rho})^2=\frac{4}{(1+\rho)^2}$.

Thus, the error for {$h_1(x)=ax+b$} is

$$\frac{1}{3}*(1+\frac{4}{(1-\rho)^2}+\frac{4}{(1+\rho)^2})$$

Now that we have the errors for both {$h_0(x)=b$} and { $h_1(x)=ax+b$ }, let's find the ρ that makes them equal.

$$\frac{1}{2}=\frac{1}{3}*(1+\frac{4}{(1-\rho)^2}+\frac{4}{(1+\rho)^2})$$

The solution for this equation is $\rho=\sqrt{9+4\sqrt{6}}$. (Used WolframAlpha)

8. Answer: [c]

I repeated the experiment for 1000 runs, and the simulation indicated that g_{svm} did better than g_{PLA} for 573 times. Thus, the closest answer is [c].

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 8

from sklearn.svm import SVC   #Used for implementing SVM.
import numpy as np
import random

def gen_target_func():            #generate a target function(f(x)) and return the corresponding vertical coordinate
                                  #input: none
                                  #output: target_function. format: [slope, y_intercept]
    rnd_x1 = np.zeros(2)
    rnd_x2 = np.zeros(2)

    for i in range (0,2):
        rnd_x1[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    for i in range (0,2):
        rnd_x2[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    slope_target_func = (rnd_x2[1]-rnd_x1[1])/(rnd_x2[0]-rnd_x1[0]) # slope = (y2-y1)/(x2-x1)

    y_intercept = rnd_x2[1]-slope_target_func*rnd_x2[0]

    return [slope_target_func, y_intercept]

def Label_data(X_vector, target_f): #return a correct label(1 or -1) by using the input vector and target equation f.
                                  #inputs
                                  # X_vector : input point's coordinate. format: [a, b]
                                  # target_f : target function. format: a
                                  #outputs
                                  # y : correct label for the input vector. format: a (1 or -1)

    if(X_vector[1] > target_f): #if the input's vertical coordinate is above the target function, return 1 label
                                 #if the input's vertical coordinate is below the target function, return -1 label
        return 1
    else:
        return -1

def generate_random_point():   #generate random data point's coordinate
                               #inputs
                               # none
                               #outputs
                               # x: random points. format: [a,b]
    x = np.zeros(2)
    x[0] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    x[1] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    return x

#Initializing the constants which will be used for training and testing
Total_Run = 1000            #Total number of experiment runs.
Test_Data_Num = 1000        #Total number of testing data points
Total_Support_Vector_Num = 0 #Total number of support vectors. Used to find the average number of support vectors.
SVM_better_than_PLA = 0     #Counts the number of cases when SVM did better than PLA.
N=10                        #Number of data points for training.

print("---------------------N=%d----------------------" % N)
for run in range (0,Total_Run):
    clf = SVC(kernel='linear', C=1e12)  #Set C to a very high value for the hard margin.

    target_info = gen_target_func()    #target_info = [slope_target_func, y_intercept]

    x = np.zeros([N,4])      # x - [[1, x1,x2,label(y)],
                             #     [1, x1,x2,label(y)],
                             #     ..
                             #     [1, x1,x2,label(y)]]

    w = np.zeros([3,1])        #initializing w vector
    w = np.squeeze(w)          #remove one dimension for matrix operations
```

```
    # generate N random data points with their correct labels based on the current target function f(x)
    for i in range (0, N):
        x[i,0] = 1                                  #x0 = 1
        x[i,1:3] = generate_random_point()           #random data points coordinate data
        f_x = target_info[0] * x[i,1] + target_info[1] #obtaining the target equation f
        x[i,3] = Label_data(x[i,1:3],f_x)            #using f, obtain the label(y) and append it to the array

    sum_all_one_side  = 0
    for i in range (0, N):
        sum_all_one_side = sum_all_one_side + x[i,3]

    if (sum_all_one_side == N or sum_all_one_side == -N):
        continue;

    #Fit the SVM.
    clf.fit(x[:,0:3],x[:,3])
    #Get the number of support vectors.
    Total_Support_Vector_Num = Total_Support_Vector_Num + len(clf.support_)


    #Go through PLA until the hypothesis converges.
    while (1):
        misclassified_arr = []
        for i in range (0,N):
            g_x = np.dot(w.T, x[i, 0:3])  # g(x) = dot(w,x)
            if(x[i,3] != np.sign(g_x)):
                misclassified_arr.append(i)
        if(len(misclassified_arr) == 0):       #escape the loop when PLA converges.
           break
        w = w + x[random.choice(misclassified_arr), 3] * x[random.choice(misclassified_arr), 0:3]  # w = w + y*X


#Testing begins
    #Generate random points to examine the error rate of g(x)
    test = np.zeros([Test_Data_Num, 4])
    for i in range(0, Test_Data_Num):
        test[i, 0] = 1  # x0 = 1
        test[i, 1:3] = generate_random_point()
        f_x = target_info[0] * test[i, 1] + target_info[1]
        test[i, 3] = Label_data(test[i, 1:3], f_x)  # y (label)

    svm_wrong_counter = 0
    svm_predict = clf.predict(test[:,0:3])
    for i in range(0,Test_Data_Num):
        if (test[i,3] != svm_predict[i]):
            svm_wrong_counter = svm_wrong_counter + 1  #Count the data points misclassified by SVM.

    #Examine the error using the generated test points
    pla_wrong = 0
    for i in range (0,Test_Data_Num):
        g_x = np.dot(w.T, test[i, 0:3])  # g(x) = dot(w,x)
        if (test[i, 3] != np.sign(g_x)):
            pla_wrong = pla_wrong + 1           #Count the data points misclassified by PLA.

    #If SVM did better than PLA, increase the counter.
    if (svm_wrong_counter < pla_wrong):
        SVM_better_than_PLA = SVM_better_than_PLA + 1

print("svm better than pla: %d times " % SVM_better_than_PLA)
print("svn better than pla: %.2f percent" % (SVM_better_than_PLA/Total_Run))
```

9. Answer: [d]

I repeated the experiment for 1000 runs, and the simulation indicated that g_{svm} did better than g_{PLA} for 621 times. Thus, the closest answer is [d].

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 9

from sklearn.svm import SVC   #Used for implementing SVM.
import numpy as np
import random

def gen_target_func():           #generate a target function(f(x)) and return the corresponding vertical coordinate
```

```
                                #input: none
                                #output: target_function. format: [slope, y_intercept]
    rnd_x1 = np.zeros(2)
    rnd_x2 = np.zeros(2)

    for i in range (0,2):
        rnd_x1[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    for i in range (0,2):
        rnd_x2[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    slope_target_func = (rnd_x2[1]-rnd_x1[1])/(rnd_x2[0]-rnd_x1[0]) # slope = (y2-y1)/(x2-x1)

    y_intercept = rnd_x2[1]-slope_target_func*rnd_x2[0]

    return [slope_target_func, y_intercept]

def Label_data(X_vector, target_f): #return a correct label(1 or -1) by using the input vector and target equation f.
                                #inputs
                                # X_vector : input point's coordinate. format: [a, b]
                                # target_f : target function. format: a
                                #outputs
                                # y : correct label for the input vector. format: a (1 or -1)

    if(X_vector[1] > target_f):   #if the input's vertical coordinate is above the target function, return 1 label
                               #if the input's vertical coordinate is below the target function, return -1 label
        return 1
    else:
        return -1

def generate_random_point():     #generate random data point's coordinate
                              #inputs
                              # none
                              #outputs
                              # x: random points. format: [a,b]
    x = np.zeros(2)
    x[0] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    x[1] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    return x

#Initializing the constants which will be used for training and testing
Total_Run = 1000            #Total number of experiment runs.
Test_Data_Num = 1000        #Total number of testing data points
Total_Support_Vector_Num = 0 #Total number of support vectors. Used to find the average number of support vectors.
SVM_better_than_PLA = 0     #Counts the number of cases when SVM did better than PLA.
N=100                       #Number of data points for training.

print("--------------------N=%d----------------------" % N)
for run in range (0,Total_Run):
    clf = SVC(kernel='linear', C=1e12)  #Set C to a very high value for the hard margin.

    target_info = gen_target_func()    #target_info = [slope_target_func, y_intercept]

    x = np.zeros([N,4])        # x - [[1, x1,x2,label(y)],
                               #      [1, x1,x2,label(y)],
                               #      ..
                               #      [1, x1,x2,label(y)]]

    w = np.zeros([3,1])        #initializing w vector
    w = np.squeeze(w)          #remove one dimension for matrix operations

    # generate N random data points with their correct labels based on the current target function f(x)
    for i in range (0, N):
        x[i,0] = 1                                  #x0 = 1
        x[i,1:3] = generate_random_point()           #random data points coordinate data
        f_x = target_info[0] * x[i,1] + target_info[1] #obtaining the target equation f
        x[i,3] = Label_data(x[i,1:3],f_x)            #using f, obtain the label(y) and append it to the array

    sum_all_one_side  = 0
    for i in range (0, N):
        sum_all_one_side = sum_all_one_side + x[i,3]

    if (sum_all_one_side == N or sum_all_one_side == -N):
        continue;

    #Fit the SVM.
```

```
    clf.fit(x[:,0:3],x[:,3])
    #Get the number of support vectors.
    Total_Support_Vector_Num = Total_Support_Vector_Num + len(clf.support_)


    #Go through PLA until the hypothesis converges.
    while (1):
        misclassified_arr = []
        for i in range (0,N):
            g_x = np.dot(w.T, x[i, 0:3])  # g(x) = dot(w,x)
            if(x[i,3] != np.sign(g_x)):
                misclassified_arr.append(i)
        if(len(misclassified_arr) == 0):       #escape the loop when PLA converges.
           break
        w = w + x[random.choice(misclassified_arr), 3] * x[random.choice(misclassified_arr), 0:3]  # w = w + y*X


#Testing begins
    #Generate random points to examine the error rate of g(x)
    test = np.zeros([Test_Data_Num, 4])
    for i in range(0, Test_Data_Num):
        test[i, 0] = 1  # x0 = 1
        test[i, 1:3] = generate_random_point()
        f_x = target_info[0] * test[i, 1] + target_info[1]
        test[i, 3] = Label_data(test[i, 1:3], f_x)  # y (label)

    svm_wrong_counter = 0
    svm_predict = clf.predict(test[:,0:3])
    for i in range(0,Test_Data_Num):
        if (test[i,3] != svm_predict[i]):
            svm_wrong_counter = svm_wrong_counter + 1  #Count the data points misclassified by SVM.

    #Examine the error using the generated test points
    pla_wrong = 0
    for i in range (0,Test_Data_Num):
        g_x = np.dot(w.T, test[i, 0:3])  # g(x) = dot(w,x)
        if (test[i, 3] != np.sign(g_x)):
            pla_wrong = pla_wrong + 1           #Count the data points misclassified by PLA.

    #If SVM did better than PLA, increase the counter.
    if (svm_wrong_counter < pla_wrong):
        SVM_better_than_PLA = SVM_better_than_PLA + 1

print("svm better than pla: %d times " % SVM_better_than_PLA)
print("svn better than pla: %.2f percent" % (SVM_better_than_PLA/Total_Run))
```

10. Answer: [b]

I repeated the experiment for 1000 runs, and the average number of support vectors was 2.998. Thus, the answer is [b].

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW7 Problem 10

from sklearn.svm import SVC   #Used for implementing SVM.
import numpy as np
import random

def gen_target_func():            #generate a target function(f(x)) and return the corresponding vertical coordinate
                                  #input: none
                                  #output: target_function. format: [slope, y_intercept]
    rnd_x1 = np.zeros(2)
    rnd_x2 = np.zeros(2)

    for i in range (0,2):
        rnd_x1[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    for i in range (0,2):
        rnd_x2[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    slope_target_func = (rnd_x2[1]-rnd_x1[1])/(rnd_x2[0]-rnd_x1[0]) # slope = (y2-y1)/(x2-x1)

    y_intercept = rnd_x2[1]-slope_target_func*rnd_x2[0]
```

```
    return [slope_target_func, y_intercept]

def Label_data(X_vector, target_f): #return a correct label(1 or -1) by using the input vector and target equation f.
                                #inputs
                                # X_vector : input point's coordinate. format: [a, b]
                                # target_f : target function. format: a
                                #outputs
                                # y : correct label for the input vector. format: a (1 or -1)

    if(X_vector[1] > target_f):    #if the input's vertical coordinate is above the target function, return 1 label
                                #if the input's vertical coordinate is below the target function, return -1 label
        return 1
    else:
        return -1

def generate_random_point():     #generate random data point's coordinate
                               #inputs
                               # none
                               #outputs
                               # x: random points. format: [a,b]
    x = np.zeros(2)
    x[0] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    x[1] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    return x

#Initializing the constants which will be used for training and testing
Total_Run = 1000            #Total number of experiment runs.
Test_Data_Num = 1000        #Total number of testing data points
Total_Support_Vector_Num = 0 #Total number of support vectors. Used to find the average number of support vectors.
SVM_better_than_PLA = 0     #Counts the number of cases when SVM did better than PLA.
N=100                      #Number of data points for training.

print("---------------------N=%d----------------------" % N)
for run in range (0,Total_Run):
    clf = SVC(kernel='linear', C=1e12)  #Set C to a very high value for the hard margin.

    target_info = gen_target_func()    #target_info = [slope_target_func, y_intercept]

    x = np.zeros([N,4])        # x - [[1, x1,x2,label(y)],
                               #      [1, x1,x2,label(y)],
                               #      ..
                               #      [1, x1,x2,label(y)]]

    w = np.zeros([3,1])        #initializing w vector
    w = np.squeeze(w)          #remove one dimension for matrix operations

    # generate N random data points with their correct labels based on the current target function f(x)
    for i in range (0, N):
        x[i,0] = 1                                  #x0 = 1
        x[i,1:3] = generate_random_point()           #random data points coordinate data
        f_x = target_info[0] * x[i,1] + target_info[1] #obtaining the target equation f
        x[i,3] = Label_data(x[i,1:3],f_x)            #using f, obtain the label(y) and append it to the array

    sum_all_one_side  = 0
    for i in range (0, N):
        sum_all_one_side = sum_all_one_side + x[i,3]

    if (sum_all_one_side == N or sum_all_one_side == -N):
        continue;

    #Fit the SVM.
    clf.fit(x[:,0:3],x[:,3])
    #Get the number of support vectors.
    Total_Support_Vector_Num = Total_Support_Vector_Num + len(clf.support_)


    #Go through PLA until the hypothesis converges.
    while (1):
        misclassified_arr = []
        for i in range (0,N):
            g_x = np.dot(w.T, x[i, 0:3])  # g(x) = dot(w,x)
            if(x[i,3] != np.sign(g_x)):
                misclassified_arr.append(i)
        if(len(misclassified_arr) == 0):       #escape the loop when PLA converges.
           break
        w = w + x[random.choice(misclassified_arr), 3] * x[random.choice(misclassified_arr), 0:3]  # w = w + y*X
```

```
#Testing begins
    #Generate random points to examine the error rate of g(x)
    test = np.zeros([Test_Data_Num, 4])
    for i in range(0, Test_Data_Num):
        test[i, 0] = 1  # x0 = 1
        test[i, 1:3] = generate_random_point()
        f_x = target_info[0] * test[i, 1] + target_info[1]
        test[i, 3] = Label_data(test[i, 1:3], f_x)  # y (label)

    svm_wrong_counter = 0
    svm_predict = clf.predict(test[:,0:3])
    for i in range(0,Test_Data_Num):
        if (test[i,3] != svm_predict[i]):
            svm_wrong_counter = svm_wrong_counter + 1  #Count the data points misclassified by SVM.

    #Examine the error using the generated test points
    pla_wrong = 0
    for i in range (0,Test_Data_Num):
        g_x = np.dot(w.T, test[i, 0:3])  # g(x) = dot(w,x)
        if (test[i, 3] != np.sign(g_x)):
            pla_wrong = pla_wrong + 1           #Count the data points misclassified by PLA.

    #If SVM did better than PLA, increase the counter.
    if (svm_wrong_counter < pla_wrong):
        SVM_better_than_PLA = SVM_better_than_PLA + 1

print("svm better than pla: %d times " % SVM_better_than_PLA)
print("svn better than pla: %.2f percent" % (SVM_better_than_PLA/Total_Run))
print('Average Number of Support Vectors: ', Total_Support_Vector_Num/Total_Run) #For Problem 10
```