CS 156a Set 7

Cason Shepard

January 18, 2023

problem 1

After running the code below, the following values for E_{val} were found:

(a) $k = 3, E_{val} = 0.3$

(b) $k = 4, E_{val} = 0.5$

(c) $k = 5, E_{val} = 0.2$

(d) $k = 6, E_{val} = 0$

(e) $k = 7, E_{val} = 0.1$

The answer is [d]

problem 2

After running the code below, the following values for E_{out} were found:

(a) $k = 3, E_{out} = 0.42$

(b) $k = 4, E_{out} = 0.416$

(c) $k = 5, E_{out} = 0.188$

(d) $k = 6, E_{out} = 0.084$

(e) $k = 7, E_{out} = 0.072$

The answer is [e]

problem 3

After switching the validation and training sets and running the code below, the following values for E_{val} were found:

(a) $k = 3, E_{val} = 0.28$

(b) $k = 4, E_{val} = 0.36$

(c) $k = 5, E_{val} = 0.2$

(d) $k = 6, E_{val} = 0.08$

(e) $k = 7, E_{val} = 0.12$

The answer is [d]

problem 4

After running the code below, the following values for E_{out} were found:

(a) $k = 3, E_{out} = 0.396$

(b) $k = 4, E_{out} = 0.388$

(c) $k = 5, E_{out} = 0.284$

(d) $k = 6, E_{out} = 0.192$

(e) $k = 7, E_{out} = 0.196$

The answer is [d]

problem 5

The out of sample errors for Problems 1 and 3 were 0.072 and 0.192, respectively. These are closest to 0.1 and 0.2, respectively.

The answer is [b]

problem 6

Using the code below, I found experimentally that the average values for e_1 and e_2 were both 0.5, and the average value for $min(e_1, e_2)$ was 0.333. Thus, these are closest to option d.

The answer is [d]

problem 7

In order to calculate the mean squared error for these two models, we will have to calculate the hypothesis function for each leave-one-out case.

$$\textbf{Case 1: } (x,y) = (-1,0),(1,0)$$
$$h_0 = 0, h_1 = 0$$
$$\text{Validation point} = (p,1)$$
$$E^2_{h_0} = 1, E^2_{h_1} = 1$$

$$\textbf{Case 2: } (x,y) = (-1,0),(p,1)$$
$$h_0 = \tfrac{1}{2}, h_1 = \tfrac{1}{p+1}(x+1)$$
$$\text{Validation point} = (1,0)$$
$$E^2_{h_0} = \tfrac{1}{4}, E^2_{h_1} = \tfrac{4}{(p+1)^2}$$

$$\textbf{Case 3: } (x,y) = (p,1),(1,0)$$
$$h_0 = \tfrac{1}{2}, h_1 = \tfrac{1}{p-1}(x-1)$$
$$\text{Validation point} = (-1,0)$$
$$E^2_{h_0} = \tfrac{1}{4}, E^2_{h_1} = \tfrac{4}{(p-1)^2}$$

Now we just need to add up and equate the errors for h_1 and h_0, and solve for p.

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

$$p = \sqrt{9 + 4\sqrt{6}}$$

The answer is [c]

problem 8

After running my algorithm over 1000 trials, using $N = 10$ and 10k out of sample points, g_{svm} was better at approximating the target function 60.4% of the time.

The answer is [c]

problem 9

After running my algorithm over 1000 trials, using $N = 100$ and 10k out of sample points, g_{svm} was better at approximating the target function 61.4% of the time.

The answer is [d]

problem 10

After running my algorithm over 1000 trials, using $N = 100$, g_{svm} had an average of 2.998 support vectors.

The answer is [b]

This algorithm was used to answer Problems 1-5

```
import numpy

for k in [3, 4, 5, 6, 7]:
    in_sample_data = []
    with open('in.dta.txt') as inp:
        for row in inp.readlines():
            row = row [:len(row)-1]
            row = row.split(' ')
            row1 = [x for x in row if x != '']
            row1[0] = float(row1[0])
            row1[1] = float(row1[1])
            if row1[2][0] == '-':
                row1[2] = -1
            else:
                row1[2] = 1
            in_sample_data.append(row1)

    out_sample_data = []
    with open('out.dta.txt') as inp:
        for row in inp.readlines():
            row = row [:len(row)-1]
            row = row.split(' ')
            row1 = [x for x in row if x != '']
            row1[0] = float(row1[0])
            row1[1] = float(row1[1])
            if row1[2][0] == '-':
                row1[2] = -1
            else:
                row1[2] = 1
            out_sample_data.append(row1)

    in_sample_class = []
    out_sample_class = []
    for set_number, data_set in enumerate([in_sample_data, out_sample_data]):
        for idx, point in enumerate(data_set):
            temp = [1, point[0], point[1], pow(point[0],2), pow(point[1],2), point[0]*point[1], abs(poi
            data_set[idx] = temp[0:k+1]
            if set_number == 0:
                in_sample_class.append(point[2])
            else:
                out_sample_class.append(point[2])

    training = in_sample_data[0:25]
    validation = in_sample_data[25:]
    training_class = in_sample_classification[0:25]
    validation_class = in_sample_classification[25:]

    X_points = numpy.array(validation)
    X_dagger = numpy.matmul(numpy.linalg.pinv(numpy.matmul(X_points.T, X_points)), X_points.T)
    w = numpy.matmul(X_dagger, numpy.array(validation_class))

    class_check = []
    for point in out_sample_data:
        if numpy.dot(point, w) > 0:
            class_check.append(1)
        else:
            class_check.append(-1)

    diff = count_diff(out_sample_classification, class_check)
    E_out = diff/len(out_sample_data)

    print(E_out)
```

This algorithm was used to answer Problem 6

```
from random import uniform
from statistics import mean
N = 100000
e_1 = [uniform(0, 1) for x in range(0, N)]
e_2 = [uniform(0,1) for x in range(0, N)]
e_min = []
for x in range(0, N):
    e_min.append(min(e_1[x], e_2[x]))
print(mean(e_1))
print(mean(e_2))
print(mean(e_min))
```

This algorithm was used to answer Problems 8-10

```
from random import uniform
import numpy as np
import random
from sklearn import svm
import math

def count_diff(list_1, list_2):
    count = 0
    for x in range(0, len(list_1)):
        if list_1[x] != list_2[x]:
            count += 1
    return count

def sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return -1
    else:
        return 0

def reclassify(w, class_list, points):
        for i in range(0, len(class_list)):
            class_list[i] = sign(np.dot(points[i], w))

trials = 1000
N = 100
sup_vector_avg = 0
percent = 0

for j in range(0, trials):
    d = 2
    true_class = [0]*N

#PLA PLA PLA PLA PLA
    while true_class == [0]*N or true_class == [1]*N or true_class == [-1]*N:
        x_1 = [uniform(-1, 1) for x in range(0, 2)]
        y_1 = [uniform(-1, 1) for x in range(0,2)]
        f_fit = np.polyfit(x_1, y_1, 1)

        x_n = [uniform(-1, 1) for x in range(0, N)]
        y_n = [uniform(-1, 1) for x in range(0, N)]

        for i in range(0, N):
            if y_n[i] > (y_n[i]*f_fit[0] + f_fit[1]):
                true_class[i] = 1
            else:
                true_class[i] = -1

#WEIGHT CALCULATION
    w = [0,0,0] # g(x)

    classification = [0]*N
    points = [[1, x_n[i], y_n[i]] for i in range(0, N)]

    while true_class != classification:
        wrong = []
        for i in range(0, N):
            if classification[i] != true_class[i]:
                wrong.append(points[i])
        point = random.choice(wrong)
        idx = points.index(point)
        temp_sign = true_class[idx]
        w = [w[0] + temp_sign*point[0], w[1] + temp_sign*point[1], w[2] + temp_sign*point[2]]
        reclassify(w, classification, points)

#ITERATION TRACKING AND SVM
    if j%100 == 0:
        print(j)
```

```
    clf = svm.SVC(kernel="linear", C=math.inf)
    clf.fit(points, true_class)

    sup_vector_avg += len(clf.support_vectors_)/trials

    x_large = [[1, uniform(-1, 1), uniform(-1, 1)] for i in range(0, 1000)]
    y_f = [(x_large[i][1]*f_fit[0] + f_fit[1]) for i in range(len(x_large))]

    correct_class = [sign(x_large[i][2] - y_f[i]) for i in range(0, len(x_large))]

    PLA_error = [sign(np.dot(x_large[i], w)) for i in range(0, 1000)]

    clf_guess = clf.predict(x_large)

    diff_svm = count_diff(clf_guess, correct_class)
    diff_PLA = count_diff(correct_class, PLA_error)

    E_out_PLA = diff_PLA/1000
    E_out_svm = diff_svm/1000

    if (E_out_svm < E_out_PLA):
        percent += 1/trials

print(percent)
print(sup_vector_avg)
```