CS 156a Problem Set 7

Amitesh Pandey

November 2024

Validation

The code for all 5 of the following problems is appended at the end.

Problem 1

We get 0.0 as the smallest validationt error, obtained with $k = 6$ thus **[d]**.

Problem 2

We get 0.1 as the smallest out error, obtained with $k = 7$, thus **[e]**.

Problem 3

We get 0.08 as the smallest validation, obtained with $k = 6$, thus **[d]**.

Problem 4

We get 0.19 as the smallest out error, obtained with $k = 6$, thus **[d]**.

Problem 5

Our errors with chosen $k = 6$ for out error are $(0.08, 0.19) \approx (0.1, 0.2)$, closest to **[b]**.

Validation Bias

Problem 6

For e_1 and e_2, by definition of uniformly distributed, we have 0.5, 0.5 as the expected values. Now for e, we have $\mathbb{P}[e > z] = \mathbb{P}[e_1, e_2 > z]$, since e_1 and e_2 are independent and uniform, we have $\mathbb{P}[e > z] = (1 - z)^2$, so the CDF of e is $F_z = 1 - (1 - z)^2$. Then the PDF is simply $P_z = \frac{\mathrm{d}}{\mathrm{d}z}(F_z) = \frac{\mathrm{d}}{\mathrm{d}z}(1 - (1 - z)^2) = 2(1 - z)$. For the expected value, we know $\mathbb{E}[e] = \int_0^1 z P_z \mathrm{d}z$, which gets us $\mathbb{E}[e] = 2\int_0^1 z(1 - z)\mathrm{d}z = 1/3$, our solution is $(0.5, 0.5, 0.33)$ closest to **[d]**.

Cross Validation

Problem 7

We coded up a constant and linear model for all given values of ρ and came to the conclusion that up to 2 decimal places, it is option **[c]**, that results in equal error for both models.

PLA vs SVM

The code for all 3 of the following problems is appended at the end

Problem 8

We get that 54.6% of the times, SVM is better than PLA, closest to 60%, option **[c]**.

Problem 9

We get that 74.1% of the times, SVM is better than PLA, closest to 65%, option **[d]**, but barely.

Problem 10

We get that 2.998 is the average vector length over 1000 experiments, closest to 3, so option **[b]**.

Problems 1-5

[3]:
```
import numpy as np

in_sample_data = []
out_sample_data = []
with open('in_dta.txt', 'r') as f:
    for line in f:
        in_sample_data.append(line)

with open('out_dta.txt', 'r') as f:
    for line in f:
        out_sample_data.append(line)
```

[149]:
```
for i in range(len(in_sample_data)):
    act = in_sample_data[i].split(' ')
    act2 = []
    for val in act:
        if val != '':
            act2.append(float(val))
    in_sample_data[i] = act2
in_data = np.asarray(in_sample_data)
```

[150]:
```
for i in range(len(out_sample_data)):
    act = out_sample_data[i].split(' ')
    act2 = []
    for val in act:
        if val != '':
            act2.append(float(val))
    out_sample_data[i] = act2
out_data = np.asarray(out_sample_data)
```

[151]:
```
def non_linear_transform(X, k):
    x_transformed = []
    for x in X:
        x1 = x[0]
        x2 = x[1]
        x_transformed.append([1, x1, x2, x1**2, x2**2, x1*x2, abs(x1 - x2),␣
 ↪abs(x1 + x2)][:k+1])
    return np.asarray(x_transformed)
```

[152]:
```
def linear_regression(X, y):
    X_plus = np.linalg.inv(X.transpose().dot(X)).dot(X.transpose())
    w = X_plus.dot(y)
    return(w)
```

[154]:
```
def get_error(X, w, y):
    correct_pos = []
    ct = 0
    for x in X:
        if np.sign(w.dot(x)) == y[ct]:
            correct_pos.append(ct)
        ct += 1
    err = 1-len(correct_pos)/float(ct)
    return err
```

[155]:
```
X = in_data[:,:2][:25]
y = in_data[:,2][:25]
validation_X = in_data[:,:2][25:]
validation_Y = in_data[:,2][25:]
for k in range(3, 8):
    Z = non_linear_transform(X, k)
    w = linear_regression(Z,y)
    Z_val = non_linear_transform(validation_X, k)
    validation_error = get_error(Z_val,w,validation_Y)
    test_X = out_data[:,:2]
    test_Z = non_linear_transform(test_X, k)
    test_y = out_data[:,2]
    outer = get_error(test_Z, w,test_y)
    print('k = '+ str(k) + ', v_err: ' + str(round(validation_error,2)) + '␣
 ↪o_err: ' + str(round(outer, 2)))
```

```
k = 3, v_err: 0.3 o_err: 0.42
k = 4, v_err: 0.5 o_err: 0.42
k = 5, v_err: 0.2 o_err: 0.19
k = 6, v_err: 0.0 o_err: 0.08
k = 7, v_err: 0.1 o_err: 0.07
```

[156]:
```
X = in_data[:,:2][25:]
y = in_data[:,2][25:]
validation_X = in_data[:,:2][:25]
validation_Y = in_data[:,2][:25]
for k in range(3, 8):
    Z = non_linear_transform(X, k)
    w = linear_regression(Z,y)
    Z_val = non_linear_transform(validation_X, k)
    validation_error = get_error(Z_val,w,validation_Y)
    test_X = out_data[:,:2]
    test_Z = non_linear_transform(test_X, k)
    test_y = out_data[:,2]
    outer = get_error(test_Z, w,test_y)
    print('k = '+ str(k) + ', v_err: ' + str(round(validation_error,2)) + '␣
 ↪o_err: ' + str(round(outer, 2)))
```

```
k = 3, v_err: 0.28 o_err: 0.4
k = 4, v_err: 0.36 o_err: 0.39
k = 5, v_err: 0.2 o_err: 0.28
k = 6, v_err: 0.08 o_err: 0.19
k = 7, v_err: 0.12 o_err: 0.2
```

Problem 7

[157]:

```python
import math
rho_vals = [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 rho in rho_vals:
    x = np.array([-1, rho, 1])
    y = np.array([0, 1, 0])
    e_const = 0
    e_lin = 0
    for i in range(3):
        xval = x[i]
        yval = y[i]

        xp = np.delete(x, i)
        yp = np.delete(y, i)

        b = np.mean(yp)
        e_const += (yval - b)**2

        xp = np.column_stack((np.ones(xp.shape), xp))
        xval = np.array([1, xval])
        Z = np.dot(np.linalg.inv(np.dot(xp.T, xp)), xp.T)
        w = np.dot(Z, yp)
        e_lin += (yval - np.dot(xval, w))**2

    print('Option ' + ['(a)', '(b)', '(c)', '(d)'][rho_vals.index(rho)] + ', rho␣
 ↪= ' + str(round(rho,2)))
    print('Constant model error: ', round(e_const/3, 2))
    print('Linear model error: ' + str(round(e_lin/3, 2)), end ='\n'+'-'*30␣
 ↪+'\n')
```

```
Option (a), rho = 2.39
Constant model error:  0.5
Linear model error: 1.14
------------------------------
Option (b), rho = 0.86
Constant model error:  0.5
Linear model error: 64.66
------------------------------
Option (c), rho = 4.34
Constant model error:  0.5
Linear model error: 0.5
------------------------------
Option (d), rho = 2.56
Constant model error:  0.5
Linear model error: 0.99
------------------------------
```

Problems 8-10

[26]:

```
#!pip install numpy==1.26.3
from sklearn.linear_model import Perceptron
from sklearn import svm, model_selection

num_experiments = 1000
N = 10

def label_data(X, m, b):
    label = []
    for x in X:
        if x[2] >= (m * x[1] + b):
            label.append(1)
        else:
            label.append(-1)
    return label

def generate_data(N):
    #taking two random points to generate hypothesis
    p1 = [np.random.uniform(-1,1), np.random.uniform(-1,1)]
    p2 = [np.random.uniform(-1,1), np.random.uniform(-1,1)]

    m = (p2[1] - p1[1])/(p2[0] - p1[0])
    b = p1[1] - m * p1[0] #using slope to calculate intercept

    X = np.insert(np.random.uniform(-1, 1, (N, 2)), 0, 1, axis=1)
    #labelling data on the basis of said hypothesis
    y = label_data(X, m, b)
    while len(set(y)) == 1:
```

```
    #making sure not all points fall on one side of the line
        X = np.insert(np.random.uniform(-1, 1, (N, 2)), 0, 1, axis=1)
        y = label_data(X, m, b)

    X_test = np.insert(np.random.uniform(-1, 1, (5*N, 2)), 0, 1, axis=1)
    y_test = get_y(X_test, m, b)

    return (X, y, X_test, y_test)

def PLA(N):
    X, y, X_test, y_test = generate_data(N)
    perceptron_model = Perceptron()
    perceptron_model.fit(X, y)
    return perceptron_model.score(X_test, y_test)

def SVM(N):
    X, y, X_test, y_test = generate_data(N)
    svm_model = svm.SVC(kernel='linear', C=1000000000)
    svm_model.fit(X, y)
    return [len(svm_model.support_vectors_), svm_model.score(X_test, y_test)]

count = 0
for i in range(num_experiments):
    if SVM(N)[1] > PLA(N):
        count += 1
print('With N = 10, SVM is better than PLA ' + str(100*count/num_experiments) +
 ↪'% of the times')

count = 0
N = 100
sv = 0
for i in range(num_experiments):
    svf = SVM(N)
    sv += svf[0]
    if svf[1] > PLA(N):
        count += 1
print('With N = 100, SVM is better than PLA ' + str(100*count/num_experiments) +
 ↪'% of the times')
print('With N = 100, avg support vector count: ' + str(sv/num_experiments))
```

```
With N = 10, SVM is better than PLA 54.6% of the times
With N = 100, SVM is better than PLA 74.1% of the times
With N = 100, avg support vector count: 2.998
```