H.W.

Tuesday, November 7, 2023 11:58 PM

All the code can be accessed here and executed in google colab or any Jupyter Notebook environment: https://github.com/DragonBoy25830/caltech-cs-156

1)

```python
import numpy as np
```
[1] ✓ 0.1s MagicPython

```python
training_set = []
testing_set = []
```
[2] ✓ 0.0s MagicPython

```python
with open('data_in.txt', 'r') as f:
    for line in f:
        training_set.append([float(x) for x in line.split()])

with open('data_out.txt', 'r') as f:
    for line in f:
        testing_set.append([float(x) for x in line.split()])
```
[3] ✓ 0.0s MagicPython

```python
np_training_set = np.array(training_set)
np_testing_set = np.array(testing_set)
```
[4] ✓ 0.0s MagicPython

```python
def transform_data(transform_index, x1, x2):
    match transform_index:
        case 0:
            return (1)
        case 1:
            return (1, x1)
        case 2:
            return (1, x1, x2)
        case 3:
            return (1, x1, x2, x1**2)
        case 4:
            return (1, x1, x2, x1**2, x2**2)
        case 5:
            return (1, x1, x2, x1**2, x2**2, x1 * x2)
        case 6:
            return (1, x1, x2, x1**2, x2**2, x1 * x2, np.abs(x1 - x2))
        case 7:
            return (1, x1, x2, x1**2, x2**2, x1 * x2, np.abs(x1 - x2), np.abs(x1 + x2))
```

```python
def calc_error(classification, w, z_transform):
    yn = classification
    counter = 0
    for i in range(len(z_transform)):
        z = z_transform[i]
        w_val = np.sign(np.dot(w, z))

        if w_val != yn[i]:
            counter += 1

    return counter / len(classification)
```
[6] ✓ 0.0s

```python
def get_x_values(dataset):
    dataset = np.array(dataset)
    x1 = dataset[:, 0]
    x2 = dataset[:, 1]

    return np.column_stack((x1, x2))
```
[7] ✓ 0.0s

```python
def run_linear_regression_experiment(transform_index, train_num):
    x_values = get_x_values(training_set)
    train_xn = x_values[0:train_num]
    validate_xn = x_values[train_num:]
    train_yn = np_training_set[:, 2][0:train_num]
    validate_yn = np_training_set[:, 2][train_num:]


    test_xn = get_x_values(testing_set)
    test_yn = np_testing_set[:, 2]

    train_xn_transform = [transform_data(transform_index, x1, x2) for (x1, x2) in train_xn]
    validate_xn_transform = [transform_data(transform_index, x1, x2) for (x1, x2) in validate_xn]
    test_xn_transform = [transform_data(transform_index, x1, x2) for (x1, x2) in test_xn]

    w = np.matmul(np.linalg.pinv(train_xn_transform), train_yn)

    E_in = calc_error(train_yn, w, train_xn_transform)
    E_out = calc_error(test_yn, w, test_xn_transform)
    E_val = calc_error(validate_yn, w, validate_xn_transform)

    return E_in, E_out, E_val
```

```python
num_training_points = 25
```
[9] ✓ 0.0s

```python
for k in range(3, 8):
    E_in, E_out, E_val = run_linear_regression_experiment(k, num_training_points)
    print(f"k = {k} --> E_val = {E_val}")
```
[10] ✓ 0.0s

```
k = 3 --> E_val = 0.3
k = 4 --> E_val = 0.5
k = 5 --> E_val = 0.2
k = 6 --> E_val = 0.0
k = 7 --> E_val = 0.1
```

The above code shows that the classification error on the validation set is smallest when $K=6$, so $\boxed{\text{the answer is d.}}$

2)

```python
for k in range(3, 8):
    E_in, E_out, E_val = run_linear_regression_experiment(k, num_training_points)
    print(f"k = {k} --> E_out = {E_out}")
```
[11] ✓ 0.0s

```
k = 3 --> E_out = 0.42
k = 4 --> E_out = 0.416
k = 5 --> E_out = 0.188
k = 6 --> E_out = 0.084
k = 7 --> E_out = 0.072
```

The above code shows that the out-of-sample classification error is smallest for $K=7$, so $\boxed{\text{the answer is e}}$

3)

Question 3

```python
def run_linear_regression_experiment_switch(transform_index):
    x_values = get_x_values(training_set)
    train_xn = x_values[-10:]
    validate_xn = x_values[0:25]
    train_yn = np_training_set[:, 2][-10:]
    validate_yn = np_training_set[:, 2][0:25]


    test_xn = get_x_values(testing_set)
    test_yn = np_testing_set[:, 2]

    train_xn_transform = [transform_data(transform_index, x1, x2) for (x1, x2) in train_xn]
    validate_xn_transform = [transform_data(transform_index, x1, x2) for (x1, x2) in validate_xn]
    test_xn_transform = [transform_data(transform_index, x1, x2) for (x1, x2) in test_xn]

    w = np.matmul(np.linalg.pinv(train_xn_transform), train_yn)

    E_in = calc_error(train_yn, w, train_xn_transform)
    E_out = calc_error(test_yn, w, test_xn_transform)
    E_val = calc_error(validate_yn, w, validate_xn_transform)

    return E_in, E_out, E_val
```

[32] ✓ 0.0s

```python
for k in range(3, 8):
    E_in, E_out, E_val = run_linear_regression_experiment(k)
    print(f"k = {k} --> E_val = {E_val}")
```

[33] ✓ 0.0s

```
k = 3 --> E_val = 0.28
k = 4 --> E_val = 0.36
k = 5 --> E_val = 0.2
k = 6 --> E_val = 0.08
k = 7 --> E_val = 0.12
```

The above code shows that the classification error on the validation set is smallest when $K=6$, so $\boxed{\text{the answer is d.}}$

4)

Question 4

```python
for k in range(3, 8):
    E_in, E_out, E_val = run_linear_regression_experiment(k)
    print(f"k = {k} --> E_out = {E_out}")
```

[35] ✓ 0.0s

```
k = 3 --> E_out = 0.396
k = 4 --> E_out = 0.388
k = 5 --> E_out = 0.284
k = 6 --> E_out = 0.192
k = 7 --> E_out = 0.196
```

The above code shows that the out-of-sample classification error is smallest for $K=6$, so $\boxed{\text{the answer is d}}$

5) In problem 1, we chose $K=6$ which has an out-of-sample classification error of 0.084. In problem 2, we chose $K=6$ which has an out-of-sample classification error of 0.192. These values are closest to 0.1 & 0.2, so the answer is b.

6)

Question 6

```
num_trials = 100000
```
[15] ✓ 0.0s

```
def calc_expected_e(trial_num):
    e1_sum = 0
    e2_sum = 0
    e_min_sum = 0

    for _ in range(trial_num):
        e1 = np.random.uniform()
        e2 = np.random.uniform()

        e1_sum += e1
        e2_sum += e2
        e_min_sum += min(e1, e2)

    return e1_sum / trial_num, e2_sum / trial_num, e_min_sum / trial_num
```
[16] ✓ 0.0s

```
calc_expected_e(num_trials)
```
[17] ✓ 0.3s

```
(0.5002927418203492, 0.5000939658012538, 0.3332756901006206)
```

By the law of large numbers, if we perform the same experiment a large number of times, the average of the results from the identical trials should be close to the expected value.

We define an experiment as generating e_1 & e_2 distributed uniformly from $[0,1]$ and e as $\min(e_1, e_2)$. We see from the code above that after 100,000 trials, the average of e_1, e_2, e_c is approximately (0.500, 0.500, 0.333) which is closest to (0.5, 0.5, 0.4). $\boxed{\text{The answer is d.}}$

7) $h_0(x) = b$

- Leave out $(-1,0)$: $h_0(x) = 0.5$

  $\hookrightarrow e_1 = (0.5 - 0)^2 = 0.25$

- Leave out $(p,1)$: $h_0(x) = 0$

  $\hookrightarrow e_2 = (0-1)^2 = 1$

- Leave out $(1,0)$: $h_0(x) = 0.5$

  $\hookrightarrow e_3 = (0.5 - 0)^2 = 0.25$

$$E_{cv_0} = \frac{1}{N}\sum_{i=1}^{N} e_i = \frac{1}{3}(e_1 + e_2 + e_3)$$

$$= \frac{1}{3}(0.25 + 1 + 0.25)$$

$$= \frac{1}{2}$$

$h_1(x) = ax + b$

- Leave out $(-1,0)$: $h_1(x) = \frac{1}{p-1}x - \frac{1}{p-1} = \frac{x-1}{p-1}$

  $\hookrightarrow e_1 = \left(-\frac{2}{p-1} - 0\right) = \frac{4}{(p-1)^2}$

$\hookrightarrow e_1 = \left(-\frac{2}{p-1} - 0 \right)^2 = \frac{4}{(p-1)^2}$

- Leave out $(p, 1)$: $h_1(x) = 0$

$\hookrightarrow e_2 = (0-1)^2 = 1$

- Leave out $(1, 0)$: $h_1(x) = \frac{1}{p+1} x + \frac{1}{p+1} = \frac{x+1}{p+1}$

$\hookrightarrow e_3 = \left(\frac{2}{p+1} - 0 \right)^2 = \frac{4}{(p+1)^2}$

$$E_{cv_1} = \frac{1}{N} \sum_{i=1}^{N} e_i = \frac{1}{3} \left(\frac{4}{(p-1)^2} + 1 + \frac{4}{(p+1)^2} \right)$$

We want $E_{cv_0} = E_{cv_1}$ for some value p.

$$E_{cv_0} = E_{cv_1}$$

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

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

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

$$(p-1)^2(p+1)^2 = 4(p-1)^2 + 4(p+1)^2$$

$$[(p-1)(p+1)]^2 = 8(p^2-2p+1) + 8(p^2+2p+1)$$

$$(p^2-1)^2 = 8p^2 - 16p + 8 + 8p^2 + 16p + 8$$

$$p^4 - 2p^2 + 1 = 16p^2 + 16$$

$$0 = p^4 - 18p^2 - 15$$

$$p^2 = \frac{18 \pm \sqrt{(18)^2 - 4(1)(-15)}}{2}$$

$$p^2 = \frac{18 \pm \sqrt{384}}{2} = \frac{18 \pm \sqrt{64 \cdot 6}}{2} = \frac{18 \pm 8\sqrt{6}}{2}$$

$$p^2 = 9 \pm 4\sqrt{6}$$

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

For $p = \sqrt{9 + 4\sqrt{6}}$, the two models would be tied using leave-one-out cross-validation with squared error measure,

So the answer is C.

8)

```
from sklearn import svm
import numpy as np
import random as rand
```
[1] ✓ 0.7s

```
def classify_point(m, b, x, y):
    expected_value = m * x + b

    if y >= expected_value:
        return 1
    else:
        return -1
```
[2] ✓ 0.0s

```
def generate_target_function():
    x0 = rand.uniform(-1, 1)
    y0 = rand.uniform(-1, 1)
    x1 = rand.uniform(-1, 1)
    y1 = rand.uniform(-1, 1)
```

```python
def generate_target_function():
    x0 = rand.uniform(-1, 1)
    y0 = rand.uniform(-1, 1)
    x1 = rand.uniform(-1, 1)
    y1 = rand.uniform(-1, 1)

    m_f = (y1 - y0) / (x1 - x0)
    b_f = y0 - m_f * x0

    return m_f, b_f
```

[3] ✓ 0.0s

```python
def generate_data(m, b, num_points):
    bias = [1 for _ in range(num_points)]
    x1 = [rand.uniform(-1, 1) for _ in range(num_points)]
    x2 = [rand.uniform(-1, 1) for _ in range(num_points)]

    xn_pre_bias = np.column_stack((x1, x2))
    xn = np.column_stack((bias, xn_pre_bias))
    yn = [classify_point(m, b, x1, x2) for (bias, x1, x2) in xn]

    training_data = np.column_stack((xn, yn))

    return training_data
```

[4] ✓ 0.0s

```python
def update_misclassified_points(m_g, b_g, data):
    xn = data[:, np.array([False, True, True, False])]
    f_classification = data[:, -1]
    g_classfication = [classify_point(m_g, b_g, x, y) for (x, y) in xn]

    new_misclassified_points = []

    for i in range(len(f_classification)):
        if f_classification[i] != g_classfication[i]:
            new_misclassified_points.append(data[i])

    return new_misclassified_points
```

[6] ✓ 0.0s

```python
def PLA(w, m_g, b_g, training_data):
    iterations = [w]

    misclassified_points = update_misclassified_points(m_g, b_g, training_data)

    while len(misclassified_points) != 0:
        new_w = iterations[-1]
        m_g = -1 * new_w[1] / new_w[2]
        b_g = -1 * new_w[0] / new_w[2]

        random_index = np.random.randint(0, len(misclassified_points))
        misclassified_point = misclassified_points[random_index]

        w = np.array(new_w) + misclassified_point[-1] * np.array(misclassified_point[0])
        iterations.append(new_w)

        misclassified_points = update_misclassified_points(m_g, b_g, training_data)

    return iterations
```

[7] ✓ 0.0s

```python
pla_weights = []
svm_weights = []
```

[8] ✓ 0.0s

```python
def calc_error(w, f_classification):
    xn = data[:, np.array([True, True, True, False])]
    f_classification = data[:, -1]
    g_classfication = [np.sign(np.dot(w, x)) for x in xn]

    counter = 0
    for i in range(len(f_classification)):
        if f_classification[i] != g_classfication[i]:
            counter += 1

    return counter / len(xn)
```

[10] ✓ 0.0s

```python
for i in range(1000):
    m, b = generate_target_function()
    data = generate_data(m, b, 10)

    while len(np.unique(data[:, -1])) == 1:
        data = generate_data(m, b, 10)

    X = data[:, 0:3]
    y = data[:, 3]
    clflin = linear_model.Perceptron()
    clflin.fit(X, y)
    pla_weights.append(calc_error(clflin.coef_, y))

    clf = svm.SVC(kernel='linear', C=1e10)
    clf.fit(X, y)
    svm_weights.append(calc_error(clf.coef_, y))
```

[11] ✓ 0.6s

```python
counter = 0

for pla_weight, svm_weight in zip(pla_weights, svm_weights):
    if pla_weight < svm_weight:
        counter+= 1

counter / 1000
```

[12] ✓ 0.0s

```
0.622
```

The above code shows that g_{svm} is better than g_{PLA} about 62.2% of the time which is approximately 60%, so the answer is C.

9)

```python
pla_weights = []
svm_weights = []
num_svm_vectors = []
```

[13] ✓ 0.0s

```python
for i in range(1000):
    m, b = generate_target_function()
    data = generate_data(m, b, 10)

    while len(np.unique(data[:, -1])) == 1:
        data = generate_data(m, b, 100)

    X = data[:, 0:3]
    y = data[:, 3]
    clflin = linear_model.Perceptron()
    clflin.fit(X, y)
```

9)

```
pla_weights = []
svm_weights = []
num_svm_vectors = []
```
[13] ✓ 0.0s

```
for i in range(1000):
    m, b = generate_target_function()
    data = generate_data(m, b, 10)

    while len(np.unique(data[:, -1])) == 1:
        data = generate_data(m, b, 100)

    X = data[:, 0:3]
    y = data[:, 3]
    clflin = linear_model.Perceptron()
    clflin.fit(X, y)
    pla_weights.append(calc_error(clflin.coef_, y))

    clf = svm.SVC(kernel='linear', C=1e10)
    clf.fit(X, y)
    svm_weights.append(calc_error(clf.coef_, y))
    num_svm_vectors.append(len(clf.support_vectors_))
```
[14] ✓ 0.6s

```
counter = 0

for pla_weight, svm_weight in zip(pla_weights, svm_weights):
    if pla_weight < svm_weight:
        counter+= 1

counter / 1000
```
[15] ✓ 0.0s

··· 0.625

The above code shows that g_{svm} is better than g_{PLA} about 62.5% of the time which is approximately 65%, so the answer is d.

10)

```
np.mean(num_svm_vectors)
```
[16] ✓ 0.0s

··· 2.875

The above code shows the average number of support vectors of g_{svm} is 2.875 which is closest to 3, so the answer is b.