HW5

October 26, 2023

0.1 CS156A Homework 5

0.2 Wilson Duan

0.2.1 Problem 1.

$0.008 < (0.1)^2(1 - \frac{8+1}{N})$

$0.8 < 1 - \frac{9}{N}$

$\frac{9}{N} < 0.2$

$45 < N$

The smallest answer choice that is greater than 45 is answer choice **c)** 100.

0.2.2 Problem 2.

For the feature vector $(1, x_1^2, x_2^2)$, the linear classification model is described by $\text{sign}(w_0 * 1 + w_1 x_1^2 + w_2 x_2^2)$. According to the hyperbolic decision boundary, we want the model to predict -1 for very large x_1 and very negative x_1. This means that as x_1^2 increases, we want the term inside the sign() to be more negative. Therefore, the coefficient to the term x_1^2 must be negative, so $w_1 < 0$.

We also want to predict 1 for very large x_2 and very negative x_2. This means that as x_2^2 increases, we want the term inside the sign() to be more positive. Therefore, the coefficient to the term x_2^2 must be positive, so $w_2 > 0$.

We also know that w_0 can be adjusted to accomodate these weights, making the hyperbolic decision boundary possible. From the analysis above, the answer choice we arrive at is **d)**.

0.2.3 Problem 3.

The 4th order polynomial transform has a dimension of $d = 14$ (we do not count the 1). The VC dimension of linear models follows the equation $d_{vc} \leq d + 1 = 14 + 1 = 15$. Therefore, the smallest answer choice that is not smaller than d_{vc} is **c)** 15.

0.2.4 Problem 4.

$E(u, v) = (ue^v - 2ve^{-u})^2$

Using the chain rule, $\frac{\partial E(u,v)}{\partial u} = 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u})$

As a result, the answer is **e)**.

0.2.5 Problem 5.

[1]:
```python
import math
import random
import numpy as np
```

[2]:
```python
def partials(u, v):
    partial_u = 2 * (u * np.exp(v) - 2 * v * np.exp(-u)) * \
                (np.exp(v) + 2 * v * np.exp(-u))
    partial_v = 2 * (u * np.exp(v) - 2 * v * np.exp(-u)) * \
                (u * np.exp(v) - 2 * np.exp(-u))
    return partial_u, partial_v

def calculate_error(u, v):
    return (u * math.e ** v - 2 * v * math.e ** -u) ** 2
```

[3]:
```python
error_threshold = 10 ** -14
learning_rate = 0.1
u, v = 1, 1
error = calculate_error(u, v)
iterations = 0
while (error > error_threshold):
    partial_u, partial_v = partials(u, v)
    u -= partial_u * learning_rate
    v -= partial_v * learning_rate
    error = calculate_error(u, v)
    iterations += 1

print("Iterations for error to fall below threshold:", iterations)
```

```
Iterations for error to fall below threshold: 10
```

According to the code above, it takes 10 iterations for the error to fall below the 10^{-14} threshold, so the answer is **d)**.

0.2.6 Problem 6.

[4]:
```python
u, v
```

[4]:
```
(0.04473629039778207, 0.023958714099141746)
```

According to the code above, the answer choices that are closest to the u and v found above arre **e)** (0.045, 0.024).

0.2.7 Problem 7.

[5]:
```python
iterations = 15
u, v = 1, 1
for i in range(iterations):
    partial_u, _ = partials(u, v)
```

```
    u -= partial_u * learning_rate
    _, partial_v = partials(u, v)
    v -= partial_v * learning_rate

    error = calculate_error(u, v)

print("Error after 15 full iterations:", error)
```

```
Error after 15 full iterations: 0.13981379199615324
```

According to the code above, the error after 15 full iterations is roughly 0.1398, which is closest to answer choice **a)**.

0.2.8 Problem 8.

[20]:
```
# Define a set of helper functions
def random_point():
    x = random.random() * 2 - 1
    y = random.random() * 2 - 1
    return (x, y)

def random_line():
    x1, y1 = random_point()
    x2, y2 = random_point()

    slope = (y2 - y1) / (x2 - x1)
    intercept = y1 - slope * x1
    return (slope, intercept)

def evaluate_point(slope, intercept, x, y):
    if (slope * x + intercept > y):
        return -1
    return 1

def create_dataset(n, slope, intercept):
    X = []
    y = []
    for i in range(n):
        a, b = random_point()
        X.append([a, b])
        y.append(evaluate_point(slope, intercept, a, b))
    return np.array(X), np.array(y)

def cross_entropy_error(X, y, w):
    return np.mean(np.log(1 + np.exp(-y * np.dot(X, w))))
```

[23]:
```
N = 100
N_test = 1000
```

```python
runs = 100
learning_rate = 0.01
avg_epochs = 0
avg_error = 0

for i in range(runs):
    # generate train and test datasets
    slope, intercept = random_line()
    X_train, y_train = create_dataset(N, slope, intercept)
    train_set = list(zip(X_train, y_train))
    X_test, y_test = create_dataset(N_test, slope, intercept)

    # initialize
    w = np.zeros(3)
    epochs = 0

    # train model
    while True:
        w_prev = np.copy(w)
        # shuffle data
        random.shuffle(train_set)
        for x, y in train_set:
            x = np.insert(x, 0, 1)
            z = np.dot(w, x)
            gradient = (-y * x) / (1 + np.exp(y * z))
            w -= learning_rate * gradient

        epochs += 1
        if (np.linalg.norm(w - w_prev) < 0.01):
            break


    # test model
    X_test_modified = np.zeros((N_test, 3))
    for j in range(len(X_test)):
        X_test_modified[j] = np.insert(X_test[j], 0, 1)

    error = cross_entropy_error(X_test_modified, y_test, w)
    avg_error += error
    avg_epochs += epochs

avg_error /= runs
avg_epochs /= runs
print("Average E_out:", avg_error)
print("Average epochs to converge:", avg_epochs)
```

```
Average E_out: 0.10379824890640138
Average epochs to converge: 336.44
```

According to the code above, the average E_out is closest to answer choice **d)** 0.100

0.2.9 Problem 9.

According to the code above, the average epochs it takes to converge is closest to answer choice **a)** 350.

0.2.10 Problem 10.

When using SGD, the weights are updated point by point. When a point x is classified properly, we don't want to update the weights, and we want the error to be zero. This makes answer choices a, b, and d incorrect because they update the weights no matter the classification accuracy. When a point is misclassified, we want the error to be $-yw^T x$ in order to simulate a PLA, so the answer is **e)**.