ML HW6

B06303126 Lo Yun Chien

January 2020

Question 1

Correct answer: (b)
By Lecture 212 page 15, we have

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)} (tanh'(s_j^l))$$

Putting l = 2, we have 6 neurons, and k = 1 for there's 1 neuron in layer 3. Thus it counts for 6 * 1 operations. And for l = 1, we have 5 neurons, and k = 6 for there are 5 neurons in layer 2. Thus it counts for 5 * 6 operations.

Sum them up. The answer is 36

Question 2

Correct answer: (d)
First we starts from 1 hidden layer, and then it has 50 neurons. While creating the second layer, consider MR and MC

$$MR = 49 + 3$$
$$MC = 20 + 3$$

Which means, while we transfer 1 neuron to the second layer, we gain 49 + 3 connections and loss 20 + 3 connections. Since $MR > MC$, it's beneficial to do it. Until two layers each are 33, 17 neurons(MR = MC). Now we consider the third layer, transferring 1 neuron from first layer have $MR = 17 + 3$, $MC = 20 + 17$, transferring 1 neuron from second layer to third layer have $MR = 16 + 3$, $MC = 33 + 3$, both not worth it. Since the whole question is convex, then we sure the problem has reached its maximum, which is 1219

Question 3

Correct answer: (d)

$$\frac{\partial err}{\partial s_i^{(L)}} = -\sum_{k=1}^{K} v_k \frac{\frac{\partial q_k}{\partial s_i^{(L)}}}{q_k}$$

For $k \neq i$

$$\frac{\partial q_k}{\partial s_i^{(L)}} = \frac{-e^{s_k}}{(\sum_j e^{s_j})^2} e^{s_i} = -q_k q_i$$

For $k = i$

$$\frac{\partial q_k}{\partial s_i^{(L)}} = \frac{e^{s_i}}{\sum_j e^{s_j}} - (\frac{e^{s_i}}{\sum_j e^{s_j}})^2 = q_i - q_i^2$$

Summarized we get

$$\frac{\partial err}{\partial s_i^{(L)}} = -\sum_{k \neq i} v_k(-q_i) - v_i(1 - q_i) = q_i \sum_k v_k - v_i = q_i - v_i$$

Question 4

Correct answer: (a)
In the first run, $x_i^{(l)}$ are 0 for all i and l, thus $s_i^{(l)}$ are all 0 too. Then $\delta_1^{(2)} = -2$, $\delta_i^{(1)} = 0$ for all i, since it's multiplied the weight. When updating w, only the bias term in the second layer is updated, others since $x_i = 0$ then stay 0. The same induction works for the second and the third update. Therefore $w_{01}^{(1)}$ remain 0.

Question 5

Correct answer: (e)
Since V is a all 2 constant matrix, the hidden layer's value is 2. Running a regression per movie,

$$2 * w_m = r_{nm}$$

the best estimator is the average of r_{nm} among n. Since it has multiplied 2, then weight is half of the average rating.

Question 6

Correct answer: (b)
Derive the partial derivative,

$$\frac{\partial err}{\partial a_m} = -2(r_{nm} - w_m^T v_n - a_m - b_n)$$

Where $\eta = \frac{1}{2}$, and turn it to the negative direction.

$$a_m \longleftarrow a_m + \eta(r_{nm} - w_m^T v_n - a_m - b_n)$$

Thus the correct answer is (b)

Question 7

Correct answer: (d)
We calculate

$$E[y_n \neq G(x_n)] = E[y_n \neq sign(g_1(x_n) + g_2(x_n) + g_3(x_n))] = 0.2$$

It imply when G makes mistakes, at least two of g_i need make mistakes. Due to this probability framework, We define the below algorithm to check $E_{out}(g_i)$ is valid.

For $\max(E_{out}(g_i))$¿0.2 , if second $\max(E_{out}(g_i))$ is bigger than 0.2, then check whether the third one is less than 0.2, if it pass all the trial, it's valid.

if second $\max(E_{out}(g_i))$ is less than 0.2, check whether sum of the second one and the third one is bigger than 0.2, if it pass all the trial, it's valid.

The reason behind is trying the overlapping area is big enough, thus the answer is (d)

Question 8

Correct answer: (c)
The same as previous question, we need at least three g_i make mistakes. Since it's all independent, intersection can be calculated as product, which yield

$$\binom{5}{3}(0.6)^2(0.4)^3 + \binom{5}{4}(0.6)^1(0.4)^4 + (0.4)^5$$

The closest answer is (c)

Question 9

Correct answer: (b)
Calculate

$$(1 - \frac{1}{N})^{0.5N} = ((1 - \frac{1}{N})^N)^{0.5} = (\frac{1}{e})^{0.5}$$

The closest answer is (c)

Question 10

Correct answer: (e)
Consider the first component, denote as x_1 and x_1' and assume $x_1 \leq x_1'$. For any cutting point c less than x_1 or bigger than x_1', the decision stump yield the same sign and contribute 1 to ϕ_{ds}. For c between two points, the decision stump contribute -1 to ϕ_{ds}. To sum up, There are $2R-2L-|x_1-x_1'|$ $+1$ $|x_1 - x_1'|$ -1. Thus the first component contribute $2R - 2L - 2|x_1 - x_1'|$. The same induction can be expanded to other components, thus we yield $\phi_{ds}(x, x') = 2d(R - L) - 2\|x - x'\|_1$

Question 11

Correct answer: (a)
In Adaboost, incorrect example multiplies $1 - \epsilon_t$, correct example multiplies ϵ_t for they're sampling possibility. Where $\epsilon_t = 0.05$ and incorrect one is positive example. Thus the answer is $\frac{0.95}{0.05} = 19$

Question 12

Question 13

Correct answer: (c)
While μ_+ or μ_- is tends to 0, the maximum value tends to infinity, therefore, the one who dominant is the one who has the smallest value, therefore is $min(\mu_+, \mu_-)$.

HW6 programming problem 14 ~ 20

This document aims to implement decision tree(CA&R tree) and random forest algorithm

In [1]:

```
import numpy as np
from scipy.stats import mode
import pandas as pd
import time
```

In [2]:

```
train = pd.read_csv("hw6_train.dat.txt", sep=" ", header=None)
test = pd.read_csv("hw6_test.dat.txt", sep=" ", header=None)
```

In [3]:

```
# Gini_index definition

def impurity(y):
    mu_pos = np.count_nonzero(y == 1) / len(y)
    return 1 - mu_pos ** 2 - (1 - mu_pos) ** 2
```

In [4]:

```
# Terminal Criterion
def terminal(X: np.array, y: np.array) -> int:

    # If all the y is the same
    if len(np.unique(y)) == 1:
        return True

    # If all the x is the same
    elif len(np.unique(X, axis=0)) == 1:
        print("All Xn are the same")
        return True
    else:
        return False
```

In [5]:

```
# Branch condition, output decision stump, chosing a cutting point theta and
# using this cut to deduce impurity as quick as possible

def branch_learning(X, y):
    # Combine X and y to df
    df = np.insert(X, X.shape[1], y, axis=1)
    bEin, btheta, feature = 99999, 0, 0
    leftX, rightX, lefty, righty = None, None, None, None

    # For every column
    for i in range(X.shape[1]):
        # The whole df sort by ith column
        ith_df = df[df[:, i].argsort()]
        x_ith = ith_df[:, i]
        # Create thetas as a list with n - 1 length
        theta = [(a + b) / 2 for a, b in zip(x_ith[:-1], x_ith[1:])]

        # For every theta
        for j in range(1, len(y)):
            # Divide dataset to twopart
            y1, y2 = ith_df[:j, -1], ith_df[j:, -1]

            # Calculate the score
            Ein = len(y1) * impurity(y1) + len(y2) * impurity(y2)

            # Find the smallest, and record the relative theta, feature, two groups of data
            if Ein < bEin:
                bEin = Ein
                btheta = theta[j - 1]
                feature = i
                leftX, rightX, lefty, righty = ith_df[:j,
                                                      :-1], ith_df[j:, :-1], y1, y2

    return feature, btheta, leftX, rightX, lefty, righty
```

In [6]:

```
# Class Tree(root, subtree)
class node():

    def __init__(self):
```

```
        self.feature = None # record branch criteria
        self.threshhold = None # record branch criteria
        self.left = None # record the left subtree address
        self.right = None # record the right subtree address
        self.value = None # If this node is a leaf, value record the base hypothesis

    # Define training process
    def train(self, X: np.array, y: np.array):
        # If terminal criterion met, return the most frequent y
        if terminal(X, y):
            self.value = mode(y)[0][0]

        # If not
        else:
            # learn the branch criterion
            self.feature, self.threshhold, X1, X2, y1, y2 = branch_learning(
                X, y)

            # let left child and right child train new datasets
            self.left, self.right = node(), node()
            self.left.train(X1, y1)
            self.right.train(X2, y2)

    # Define predict process
    def predict(self, X: np.array) -> int:
        # If this node is leaf, return the base algorithm
        if self.value != None:
            return self.value

        # If this node is not leaf, choose whether go left or right
        else:
            if X[self.feature] < self.threshhold:
                return self.left.predict(X)
            else:
                return self.right.predict(X)
```

In [7]:

```
def problem14(train, test):
    print("Problem 14: ")

    # Separate each X, y dataset
    trainX, trainy = train.iloc[:, :-1], train.iloc[:, -1]
    testX, testy = test.iloc[:, :-1], test.iloc[:, -1]

    # Train DecisionTree
    DecisionTree = node()
    before = time.time()

    DecisionTree.train(np.array(trainX),
                       np.array(trainy))

    after = time.time()
    print(f"Training cost {after - before:.2f}")

    before = time.time()

    # Predict DecisionTree
    y_pred = [DecisionTree.predict(x) for x in np.array(testX)]

    after = time.time()
    print(f"Testing cost {after - before:.2f} sec")

    # Calc. Eout
    Eout = sum([y != y_hat for y, y_hat in zip(y_pred, testy)]) / len(y_pred)
    print(f"In pb14, Eout is {Eout:.2f}")

    return

problem14(train, test)
```

```
Problem 14:
Training cost 0.97
Testing cost 0.01 sec
In pb14, Eout is 0.17
```

In [8]:

```
def problem15(train, test, n):
    print("Problem 15: ")
```

```
    # Prepare n length Eoutlist and Forest
    Eoutlist = []
    Forest = []

    before = time.time()
    for _ in range(n):
        # Bagging while N' = 0.5 N
        train_15, test_15 = train.sample(
            int(len(train) / 2), replace=True), test

        # Separate datasets
        trainX, trainy = train_15.iloc[:, :-1], train_15.iloc[:, -1]
        testX, testy = test_15.iloc[:, :-1], test_15.iloc[:, -1]

        # Train DecisionTree
        DecisionTree = node()

        DecisionTree.train(np.array(trainX),
                           np.array(trainy))

        # Add it to the forest
        Forest.append(DecisionTree)

        # Predict y
        y_pred = [DecisionTree.predict(x) for x in np.array(testX)]

        # Calc. Eout
        Eout = sum([y != y_hat for y, y_hat in zip(
            y_pred, testy)]) / len(y_pred)
        Eoutlist.append(Eout)

    after = time.time()
    print(f"Total cost {after - before:.2f} sec")

    avgEout = sum(Eoutlist) / n

    print(f"In pb15, average Eout is {avgEout:.2f}")

    return Forest

Forest = problem15(train, test, 2000)
```

```
Problem 15:
Total cost 690.09 sec
In pb15, average Eout is 0.23
```

In [9]:

```
def problem16(train, Forest):
    print("Problem 16: ")
    # G is the blending function
    G = []
    before = time.time()

    # Separate dataset
    trainX, trainy = train.iloc[:, :-1], train.iloc[:, -1]
    for tree in Forest:
        y_pred = [tree.predict(x) for x in np.array(trainX)]
        G.append(y_pred)

    # If sign(g_1 + g_2 ... g_n) == 1: then G predict 1
    G = np.array(G)
    y_pred_G = [1 if i == 1 else -
                1 for i in np.apply_along_axis(sum, 0, G) > 0]

    # Calc. Ein
    Ein = sum([y != y_hat for y, y_hat in zip(
        y_pred_G, trainy)]) / len(y_pred_G)

    after = time.time()
    print(f"Total cost {after - before:.2f} sec")
    print(f"In pb16, Ein is {Ein:.2f}")

problem16(train, Forest)
```

```
Problem 16:
Total cost 8.47 sec
In pb16, Ein is 0.01
```

In [10]:

```
def problem17(test, Forest):
    print("Problem 17: ")
    # G is the blending function
    G = []
    testX, testy = test.iloc[:, :-1], test.iloc[:, -1]

    # Using every tree to predict each x and return an N * 2000 matrix
    for tree in Forest:
        y_pred = [tree.predict(x) for x in np.array(testX)]
        G.append(y_pred)

    # Aggregate along column axis, if sum > 0, then return 1 else -1
    G = np.array(G)
    y_pred_G = [1 if i == 1 else -
                1 for i in np.apply_along_axis(sum, 0, G) > 0]

    # Calc. Eout
    Eout = sum([y != y_hat for y, y_hat in zip(
        y_pred_G, testy)]) / len(y_pred_G)

    print(f"In pb17, Eout is {Eout:.2f}")

problem17(test, Forest)
```

```
Problem 17:
In pb17, Eout is 0.15
```

In [11]:

```
def problem18(train, n):
    print("Problem 18: ")
    # Forest will save all trees, index_dict record: index_dict[train.index] = set(Which tree has used i
    Forest = []
    index_dict = {}

    # Validation error
    Eval = 0

    # Initialize index_dict as empty set for all n
    for i in range(len(train)):
        index_dict[i] = set()

    # Run n times
    for i in range(n):
        # Separate dataset
        train_18 = train.sample(int(len(train) / 2), replace=True)
        trainX, trainy = train_18.iloc[:, :-1], train_18.iloc[:, -1]


        # Record whether xn has been used for tree i
        used_index = set(train_18.index)
        for chosen in used_index:
            index_dict[chosen].add(i)

        # Train DecisionTree
        DecisionTree = node()

        DecisionTree.train(np.array(trainX),
                           np.array(trainy))

        # Add it to the Forest
        Forest.append(DecisionTree)

    # CompleteSet: A set from 0 to n integers
    completeSet = set([i for i in range(n)])

    # for every xn
    for i in range(len(train)):
        # For xn, find trees don't pick it, is the complement of index_dict[n]
        G_minus = completeSet.difference(index_dict[i])
        y_pred = 0

        # If xn has been used in every tree, return constant hypothesis -1
        if len(G_minus) == 0:
            y_pred = -1

        # else, create G- predict G-(xn)
```

```
        else:
            y_pred_list = []
            for j in G_minus:
                y_pred_list.append(Forest[j].predict(np.array(train.iloc[i, :-1])))

            y_pred = 1 if sum(y_pred_list) > 0 else -1

        # Calc. Eval
        if y_pred != train.iloc[:, -1][i]:
            Eval += 1

    Eval /= len(train)
    print(f"Eval is {Eval:.2f}")
```

```
problem18(train, 2000)
```

```
Problem 18:
Eval is 0.07
```

Problem 19: (d) aggregation models: AdaBoost and Gradient Boosting

老師在上課講的例子很生動，演算法邏輯也符合人性，況且實用性又高，透過alpha和un 的相對關係，又可以知道機器在哪個地方學得不好。

我認為我在這個演算法上收益良多，是老師講解得足夠清楚的功勞

Problem 20: (b) matrix factorization

我認為他很靠近neuron network，事實上好像也是這樣，況且即使了解他的規則，要如何加入新xn和ym (新成員和新電影)，以及中間的neuron該放幾個，怎麼validation，都不是很明朗

感覺是最不清楚的一個部分

In [67]:

```
print("end of document")
```

```
end of document
```