Machine Learning **(Due: 15/01/20)**

Homework #6: Referred Answers

Instructor: Lin, Hsuan Tien *Name:* Hao-Cheng Lo*, Id:* D08227104

Problem 1:

[b] is correct.

For each $\delta_j^{(2)}$, there are 1 operations, so total is $(1)(6) = 6$. For each $\delta_j^{(1)}$, there are 6 operations, so total is $(5)(6) = 30$. Hence, total is $6 + 30 = 36$.

Problem 2:

[d] is correct.

Here provides the code of my search algorithm. The answer is 1219.

```python
from itertools import *

def calw(li):
    w = 0
    for i in range(len(li)-1):
        if i == len(li)-2:
            w += li[i]*li[i+1]
        else:
            w += li[i]*(li[i+1]-1)
    return w

max_w = 0
for hidden in range(1,26):
    possible = [j for j in combinations_with_replacement(range(2, 51), hidden) if sum(j) ==
    50]
    for each in possible:
        for item in permutations(each):
            temp_w = calw([20]+list(item)+[3])
            if temp_w > max_w:
                max_w = temp_w
print(max_w)

'''
output = 1219
'''
```

Problem 3:

[d] is correct.

Given

(1) $\dfrac{\partial \text{err}}{\partial q_i} = \dfrac{-v_i}{q_i}$

(2) $\dfrac{\partial q_i}{\partial s_j} = \dfrac{\exp(s_i)}{\sum_{l=1}^{K} \exp(s_l)} - (\dfrac{\exp(s_i)}{\sum_{l=1}^{K} \exp(s_l)})^2 = q_i(1-q_i)$, when $i = j$.

(3) $\dfrac{\partial q_i}{\partial s_j} = -\dfrac{\exp(s_i)\exp(s_j)}{(\sum_{l=1}^{K} \exp(s_l))^2} = -q_i(q_j)$, when $i \neq j$.

Hence, $\dfrac{\partial \text{err}}{\partial s_k} = \sum_{i=1}^{K} \dfrac{\partial \text{err}}{\partial q_i}\dfrac{\partial q_i}{\partial s_k} = \dfrac{\partial \text{err}}{\partial q_k}\dfrac{\partial q_k}{\partial s_k} + \sum_{i \neq k} \dfrac{\partial \text{err}}{\partial q_i}\dfrac{\partial q_i}{\partial s_k} = -\dfrac{v_k}{q_k} q_k(1-q_k) + \sum_{i \neq k} \dfrac{-v_i}{q_i} - q_i(q_k)$

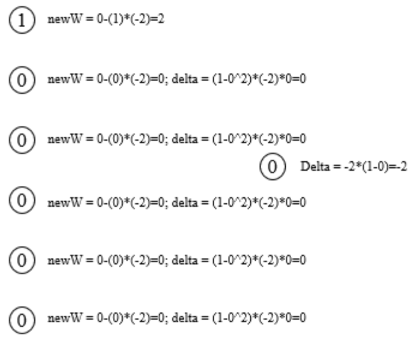
$\rightarrow \dfrac{\partial \text{err}}{\partial s_k} = -v_k(1-q_k) + q_k \sum_{i \neq k} v_i = -v_k(1-q_k) + q_k(1-v_k) = q_k - v_k$

Problem 4:

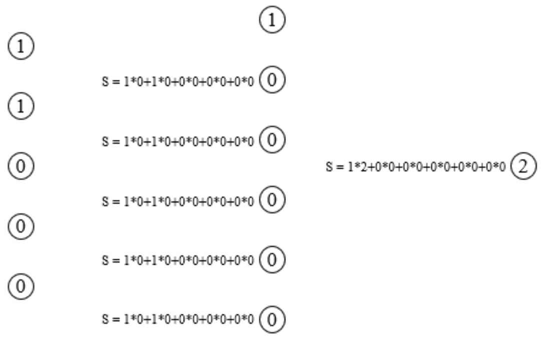
[a] is correct.

First forward pass

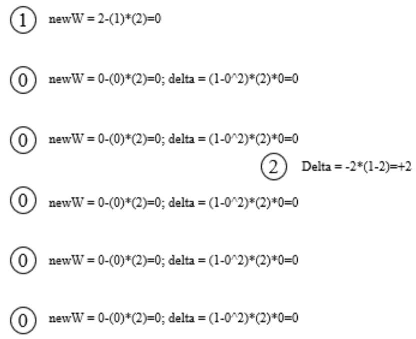
First backward pass:



Second forward pass

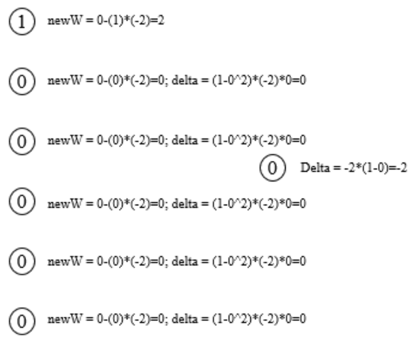


Second backward pass:



Third forward pass

Third backward pass:



Hence, $w^1_{01} = 0$.

Problem 5:

[e] is correct.

$$\frac{\partial \sum_{m=1}^{M} \left(\sum_{n=1}^{D_m} (r_{nm} - 2\mathbf{w}_m)^2 \right)}{\partial \mathbf{w}_m} = -4 \sum_{n=1}^{D_m} (r_{nm} - 2\mathbf{w}_m) = 0 \rightarrow \mathbf{w}_m = \frac{\sum_{n=1}^{D_m} r_{nm}}{\sum_{n=1}^{D_m} 2}.$$

Problem 6:

[b] is correct.

$$a_m \leftarrow a_m - \frac{\eta}{2}\nabla_a \mathrm{err} = a_m - \frac{\eta}{2}(-2)(r_{nm} - \mathbf{w}_m^T\mathbf{v}_n - a_m - b_n) = (1-\eta)a_m + \eta(r_{nm} - \mathbf{w}_m^T\mathbf{v}_n - b_n).$$

Problem 7:

[d] is correct.

The maximum of $E_{\text{out}}(G)$ depends on how many examples are wrongly classified by at least two of gs, note that $E_{\text{out}}(G) = \text{sign}(\frac{1}{3}\sum_{i=1}^{3} g_i)$.

For [a], the maximum of $E_{\text{out}}(G_a)$ is 0.16 (such reach happens when examples wrongly classified in g_{a_2} would be wrongly classified in g_{a_3}). For [b], the maximum of $E_{\text{out}}(G_b)$ is 0.12 (such reach happens when examples wrongly classified in g_{b_1} or g_{b_2} would be wrongly classified in g_{b_3}). For [c], the maximum of $E_{\text{out}}(G_c)$ is 0.10 (such reach happens when examples wrongly classified in g_{c_1} or g_{c_2} would be wrongly classified in g_{c_3}). For [d], the maximum of $E_{\text{out}}(G_d)$ is 0.24 (such reach happens when examples wrongly classified in g_{d_1} or g_{d_2} would be wrongly classified in g_{d_3}). For [e], the maximum of $E_{\text{out}}(G_e)$ is 0.10 (such reach happens when examples wrongly classified in g_{e_1} or g_{e_2} would be wrongly classified in g_{e_3}). Hence, it is possible for $E_{\text{out}}(G_d)$ to be 0.20.

Problem 8:

[c] is correct.

The example wrongly classified in G should be at least wrongly classified in 3 of gs. Hence, the expected $E_{\text{out}}(G) = \sum_{i=3}^{5} \binom{5}{i}(0.4)^i(1-0.4)^{5-i} = 0.317$.

Problem 9:

[b] is correct.

For each sample, the probability that such sample is not sampled is $1 - N^{-1}$. Hence, in $0.5N$ sampling procedures, such sample is never sampled, prob. of which is $\lim_{N\to\infty}(1 - N^{-1})^{0.5N} = e^{-0.5} = 0.6065$.

Problem 10:

[e] is correct.

$$\begin{aligned}
K_{ds}(x,x') &= (\phi_{ds}(x))^T(\phi_{ds}(x')) \\
&= \sum_{s\in\{+1,-1\}}\sum_{i=1}^{d}\sum_{\theta=2L}^{2R} \operatorname{sign}(x_i-\theta)\operatorname{sign}(x'_i-\theta) \\
&= 2\sum_{i=1}^{d}\Big((R-L)-|x_i-x'_i|\Big) \\
&= 2d(R-L)-2||x-x'||_1
\end{aligned} \tag{0.1}$$

Note that the rationales from the second equal sign to the third equal sign is that geometrically, there are $\frac{1}{2}2|x_i-x'_i|$ numbers of θs s.t. $\operatorname{sign}(x_i-\theta)\operatorname{sign}(x'_i-\theta)=-1$ in all $\frac{1}{2}(2R-2L)$ possible θs.

ref: Support Vector Machinery for Infinite Ensemble Learning

Problem 11:

[a] is correct.

$$\frac{u_+^{(2)}}{u_-^{(2)}} = \frac{u_n^{(1)}\sqrt{\dfrac{1-\epsilon_1}{\epsilon_1}}(\because \text{incorrect})}{u_n^{(1)}/\sqrt{\dfrac{1-\epsilon_1}{\epsilon_1}}(\because \text{correct})} = \frac{1-\epsilon_1}{\epsilon_1} = \frac{95\%}{5\%} = 19.$$

Problem 12:

[d] is correct.

In t, let total $u_n^{(t)}$ of incorrect examples $= x$. Thus, total $u_n^{(t)}$ of correct examples $= U_t - x$. Since total $u_n^{(t+1)}$ of incorrect examples should be equivalent to $u_n^{(t+1)}$ of correct examples in AdaBoost algorithm, we have

$$x\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = (U_t - x)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}. \text{ Hence, } x = \epsilon_t U_t.$$

Accordingly, $U_{t+1} = 2U_t\sqrt{\epsilon_t(1-\epsilon_t)}$. Since $U_1 = 1$, by recursion $U_{t+1} = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}$.

Based on above facts, we have

$$\begin{aligned}
E_{\text{in}}(G_T) &\leq U_{T+1} \\
&= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\
&\leq \prod_{t=1}^{T} 2\sqrt{\epsilon(1-\epsilon)} \\
&\leq \prod_{t=1}^{T} 2(\frac{1}{2})\exp(-2(\frac{1}{2}-\epsilon)^2) \\
&= \exp(\sum_{t=1}^{T} -2(\frac{1}{2}-\epsilon)^2) \\
&= \exp(-2T(\frac{1}{2}-\epsilon)^2)
\end{aligned} \tag{0.2}$$

Problem 13:

[d] is correct.

Let $\mu_+ = x$, $\mu_- = 1 - x$, where $x \in [0, 1]$. The normalized impurity function in the given problem is $f(x) = 2\min(\mu_+, \mu_-) = 2\min(x, 1 - x)$, such that $f(x) = 2x$ where $x \in [0, 0.5]$ and $f(x) = -2x + 2$ where $x \in [0.5, 1]$. The normalized impurity function of [d] is $f_d(x) = (1 - |\mu_+ - \mu_-|)/1 = 1 - |2x - 1|$, such that $f_d(x) = 1 - (-(2x - 1)) = 2x$ where $x \in [0, 0.5]$ and $f_d(x) = 1 - (2x - 1) = -2x + 2$ where $x \in [0.5, 1]$. Hence, $f(x)$ and $f_d(x)$ are equivalent.

Problem 19:

[a] is correct.

Impressive, informative, and succinct introduction of SVM and its underpinning optimization knowledge offers me a panoply understanding of SVM.

Problem 20:

[b] is correct.

Although matrix factorization plays a crucial role in recommendation system etc., lecturer did not put enough stress on its theoretical derivation, classic algorithm, and applications. Hope more advanced topics could be provided in upcoming courses.

In [ ]:

```
import numpy as np
import math
from random import *
```

PART I

Decision Tree w/ CART algorithm

In [2]:

```
# basic ds = tree structure node...

class NODE:
    def __init__(self, val, idx):
        self.val = val
        self.idx = idx
        self.sign = 0
        self.left = None
        self.right = None
```

In [27]:

```
# CART algorithm

class CART:
    def __init__(self, train_data_path, test_data_path, show = 0):
        # data loading
        dta = np.loadtxt(train_data_path, dtype=np.float, delimiter=" ")
        numrow, numcol = np.shape(dta)

        feature = dta[:,0:numcol-1]
        label = dta[:,numcol-1]
        self.model = self.DoingCART(feature,label)
        if show == 1:
            print("Ein =",self.predCART(self.model, feature, label))
            tdta = np.loadtxt(test_data_path, dtype=np.float, delimiter=" ")
            tnumrow, tnumcol = np.shape(tdta)
            tfeature = tdta[:,0:tnumcol-1]
            tlabel = tdta[:,tnumcol-1]
            print("Eout =",self.predCART(self.model, tfeature, tlabel))

    def ginical(self, label):
        labelsize = np.shape(label)[0]
        if labelsize == 0:
            return 0
        else:
            return 1-(sum(label==1)/float(labelsize))**2-(sum(label==-1)/float(labelsize))**2

    def decisionStump(self, targetfeature, targetlabel):
        '''
        calculation of thetas, only n-1 thetas (not n+1)
        (1) one of theta is inherent in cart_function (base case)
        (2) the other theta is equivilent to -(1): theta x,x,x,x,x == x,x,x,x,x theta
        '''
        thetaArray = np.array([(targetfeature[i] + targetfeature[i + 1]) / 2 for i in range(0, targetfeature.shape[0] - 1)])

        currentBranchCriteria = float("inf")
        targetTheta = 0.0
        for theta in thetaArray:
            #np.where retures idx
            LHS = targetlabel[np.where(targetfeature < theta)]
            RHS = targetlabel[np.where(targetfeature >= theta)]
            b = LHS.shape[0] * self.ginical(LHS) + RHS.shape[0] * self.ginical(RHS)
            if currentBranchCriteria > b:
                currentBranchCriteria = b
                targetTheta = theta
        return currentBranchCriteria, targetTheta

    def branch(self, feature, label):
```

```
        sort_of_each_feature_idx = []
        for i in range(feature.shape[1]):
            sort_of_each_feature_idx.append(np.argsort(feature[:,i]))

        bestBranch = float("inf")
        bestIndex = -1
        bestBranchVal = 0
        for i in range(feature.shape[1]): # for each sorted feature
            targetfeature = feature[sort_of_each_feature_idx[i],i]
            targetlabel = label[sort_of_each_feature_idx[i]]
            tempb,tempval = self.decisionStump(targetfeature,targetlabel)
            if bestBranch > tempb:
                bestBranch = tempb
                bestIndex = i
                bestBranchVal = tempval
        '''
        let's split
        '''
        LX=feature[np.where(feature[:,bestIndex]<bestBranchVal)]
        LY=label[np.where(feature[:,bestIndex]<bestBranchVal)]
        RX=feature[np.where(feature[:,bestIndex]>=bestBranchVal)]
        RY=label[np.where(feature[:,bestIndex]>=bestBranchVal)]

        return LX,LY,RX,RY,bestIndex,bestBranchVal

    def DoingCART(self, feature, label):
        if self.ginical(label) == 0:
            leaf = NODE(-1,-1)
            leaf.sign = label[0] #sign
            #print("leaf")
            return leaf
        LX,LY,RX,RY,bestIndex,bestBranchVal = self.branch(feature,label)
        node = NODE(bestBranchVal,bestIndex) #val, idx
        node.left = self.DoingCART(LX,LY) #l
        node.right = self.DoingCART(RX,RY) #r
        return node

    def predCART_onesample(self, root, onex):
        if root.idx == -1:
            return root.sign
        if onex[root.idx] < root.val:
            return self.predCART_onesample(root.left, onex)
        else:
            return self.predCART_onesample(root.right, onex)

    def predCART(self, root, feature, label):
        count = 0
        for i in range(np.shape(feature)[0]):
```

```
            count += 1 if self.predCART_onesample(root, feature[i]) != label[i] else 0
        return float(count)/np.shape(feature)[0]
```

Problem 14

my answer = [c]

In [30]:
```
DT = CART("hw6_train.txt","hw6_test.txt",1)
```

```
Ein = 0.0
Eout = 0.166
```

Validate *My Answer* with package

In [61]:
```
from sklearn.tree import DecisionTreeClassifier

# training data
dta = np.loadtxt("hw6_train.txt", dtype=np.float, delimiter=" ")
numrow, numcol = np.shape(dta)
feature = dta[:,0:numcol-1]
label = dta[:,numcol-1]

#testing data
tdta = np.loadtxt("hw6_test.txt", dtype=np.float, delimiter=" ")
tnumrow, tnumcol = np.shape(tdta)
tfeature = tdta[:,0:tnumcol-1]
tlabel = tdta[:,tnumcol-1]

# Fit regression model
regr_1 = DecisionTreeClassifier(max_depth=2000)
regr_1.fit(feature, label)

# Predict
y_1 = regr_1.predict(tfeature)
print("Eout =", sum(y_1!=tlabel)/1000)
```

```
Eout = 0.176
```

PART II

Random Forest

In [48]:
```
# revised CART for random forest
def ginical(label):
    labelsize = np.shape(label)[0]
    if labelsize == 0:
        return 0
    else:
        return 1-(sum(label==1)/float(labelsize))**2-(sum(label==-1)/float(labelsize))**2

def decisionStump(targetfeature, targetlabel):
    '''
    calculation of thetas, only n-1 thetas (not n+1)
    (1) one of theta is inherent in cart_function (base case)
    (2) the other theta is equivilent to -(1): theta x,x,x,x,x == x,x,x,x,x theta
    '''
    thetaArray = np.array([(targetfeature[i] + targetfeature[i + 1]) / 2 for i in range(0, targetfeature.shape[0] - 1)])

    currentBranchCriteria = float("inf")
    targetTheta = 0.0
    for theta in thetaArray:
        #np.where retures idx
        LHS = targetlabel[np.where(targetfeature < theta)]
        RHS = targetlabel[np.where(targetfeature >= theta)]
        b = LHS.shape[0] * ginical(LHS) + RHS.shape[0] * ginical(RHS)
        if currentBranchCriteria > b:
            currentBranchCriteria = b
            targetTheta = theta
    return currentBranchCriteria, targetTheta

def branch(feature, label):
    sort_of_each_feature_idx = []
    for i in range(feature.shape[1]):
        sort_of_each_feature_idx.append(np.argsort(feature[:,i]))

    bestBranch = float("inf")
    bestIndex = -1
    bestBranchVal = 0
    for i in range(feature.shape[1]): # for each sorted feature
        targetfeature = feature[sort_of_each_feature_idx[i],i]
        targetlabel = label[sort_of_each_feature_idx[i]]
        tempb,tempval = decisionStump(targetfeature,targetlabel)
        if bestBranch > tempb:
            bestBranch = tempb
            bestIndex = i
            bestBranchVal = tempval
    '''
    let's split
    '''
```

```
    LX=feature[np.where(feature[:,bestIndex]<bestBranchVal)]
    LY=label[np.where(feature[:,bestIndex]<bestBranchVal)]
    RX=feature[np.where(feature[:,bestIndex]>=bestBranchVal)]
    RY=label[np.where(feature[:,bestIndex]>=bestBranchVal)]

    return LX,LY,RX,RY,bestIndex,bestBranchVal

def DoingCART(feature, label):
    if ginical(label) == 0:
        leaf = NODE(-1,-1)
        leaf.sign = label[0] #sign
        #print("leaf")
        return leaf
    LX,LY,RX,RY,bestIndex,bestBranchVal = branch(feature,label)
    node = NODE(bestBranchVal,bestIndex) #val, idx
    node.left = DoingCART(LX,LY) #l
    node.right = DoingCART(RX,RY) #r
    return node

def predCART_onesample(root, onex):
    if root.idx == -1:
        return root.sign
    if onex[root.idx] < root.val:
        return predCART_onesample(root.left, onex)
    else:
        return predCART_onesample(root.right, onex)

def predCART(root, feature, label):
    count = 0
    for i in range(np.shape(feature)[0]):
        count += 1 if predCART_onesample(root, feature[i]) != label[i] else 0
    return float(count)/np.shape(feature)[0]
```

In [49]:
```
# bootstrapmatrix row:T col:idx

bootstrapMatrix = []
for i in range(2000):
    bootstrapMatrix.append(choices(range(0,1000),k=500))
```

In [67]:
```
from operator import itemgetter
# training data
dta = np.loadtxt("hw6_train.txt", dtype=np.float, delimiter=" ")
numrow, numcol = np.shape(dta)
trainfeature = dta[:,0:numcol-1]
trainlabel = dta[:,numcol-1]

#testing data
tdta = np.loadtxt("hw6_test.txt", dtype=np.float, delimiter=" ")
tnumrow, tnumcol = np.shape(tdta)
testfeature = tdta[:,0:tnumcol-1]
testlabel = tdta[:,tnumcol-1]
```

Problem 15

my answer = [d]

In [69]:
```
def RF(T):
    error = 0
    forest = []
    for i in range(T):
        x = np.array(list(itemgetter(*bootstrapMatrix[i])(trainfeature)))
        y = np.array(list(itemgetter(*bootstrapMatrix[i])(trainlabel)))
        model = DoingCART(x,y)
        error += predCART(model,testfeature,testlabel)
        forest.append(model)
    return error/float(T), forest
```

In [70]:
```
error, forest = RF(2000)
print(error)
```

```
0.2341425000000003
```

In [ ]:

```
'''def RandomForest(T):
    error = 0
    forest = []
    for i in range(T):
        x = np.array(list(itemgetter(*bootstrapMatrix[i])(trainfeature)))
        y = np.array(list(itemgetter(*bootstrapMatrix[i])(trainlabel)))
        model = DecisionTreeClassifier(max_depth=2000)
        model.fit(x, y)

        # Predict
        y_1 = model.predict(testfeature)
        error += sum(y_1!=testlabel)/1000
        forest.append(model)
    return error/float(T), forest

err, forest = RandomForest(2000)'''
```

Problem 16

my answer = [a]

In [77]:

```
def RF_predict(Forest, feature):
    pos=0
    neg=0
    for tree in Forest:
        predict_label = tree.predict([feature])
        if predict_label[0] == 1:
            pos += 1
        else:
            neg += 1
    return (1 if pos > neg else -1)

def cal_RF_error(Forest, feature, label):
    m = np.shape(feature)[0]
    error = 0
    for i in range(m):
        predict_label = RF_predict(Forest, feature[i])
        error += (1 if predict_label!=label[i] else 0)
    return float(error)/m

print(cal_RF_error(forest, trainfeature, trainlabel))
```

```
0.016
```

Problem 17

my answer = [d]

In [78]:
```
print(cal_RF_error(forest, testfeature, testlabel))
```

```
0.16
```

Problem 18

my answer = [b]

In [81]:
```
def RF_predict_OOB(Forest, feature, idx):
    pos=0
    neg=0
    for key, tree in enumerate(Forest):
        if idx not in bootstrapMatrix[key]:
            predict_label = tree.predict([feature])
            if predict_label[0] == 1:
                pos += 1
            else:
                neg += 1
    return (1 if pos > neg else -1)

def cal_RF_error_OOB(Forest, feature, label):
    m = np.shape(feature)[0]
    error = 0
    for i in range(m):
        predict_label = RF_predict_OOB(Forest, feature[i], i)
        error += (1 if predict_label!=label[i] else 0)
    return float(error)/m

print(cal_RF_error_OOB(forest, trainfeature, trainlabel))
```

```
0.075
```

In [ ]: