Machine Learning Jiecao Chen ID:4746311
HOMEWORK #2

Question 1

a. Given X, take $y = 1$ and W_2 such that $W_2^T X = 0$, for any $0 \leq t \leq 1$:

$$E(tW_1 + (1-t)W_2) = (1 - \hat{y}(tW_1)) \tag{1}$$

$$tE(W_1)+(1-t)E(W_2) = tE(W_1)+(1-t)(1-\frac{1}{1+exp(-0)})^2 = tE(W_1)+(1-t)(\frac{1}{2})^2 \tag{2}$$

now let $W_1^T X = -N$ where N is a very large number, we have

$$(1) \approx 1$$

and

$$(2) \approx t + \frac{1-t}{4}$$

Take $t = 0.5$ gives

$$(1) > (2)$$

This example shows that $E(W)$ is not necessarily a convex function of W

b. Take $y = 1$, we have

$$E(W) = log(1 + exp(-W^T X)) = log(1 + exp(-\sum_{i=1}^{N} w_i x_i)) \tag{3}$$

To show (3) a convex function of W, we can simply show the corresponding Hessian matrix is positive semi-definite:

Denote $k = 1 + exp(-W^T X)$

$$\frac{\partial E(W)}{\partial w_i} = \frac{-x_i exp(-W^T X)}{k} \tag{4}$$

$$H_{ij} = \frac{\partial^2 E(W)}{\partial w_i \partial w_j} = exp(-W^T X)\frac{x_i x_j}{k^2} \tag{5}$$

which means the Hessian matrix can be represented as

$$H = exp(-W^T X)A^T A$$

where

$$A = (w_1 \ w_2 \ \ldots \ w_N)$$

thus H is positive semi-deinite, we proved what we need.

c. This could be directly seen when we note that no matter $y = 1$ or $y = 0$, we have

$$E(W) = log(1 + exp(-W^T X))$$

We can definte the loss function as

$$loss(W) = \sum_{i=1}^{N} log(1 + exp(-W^T X_i))$$

it's a logistics regression problem.

Question 2

a. it's trivial to see that

$$k = \sum_{j=1}^{m} w_j k_j$$

is symmetric

when k_j is symmetric for all $1 \leq j \leq m$.

Given any vector x

$$x^T k x = \sum_{j=1}^{m} w_j x^T k_j x \tag{6}$$

as for any j, $w_j \geq 0$, k_j is positive semi-definite, thus

$$w_j x^T k_j x \geq 0$$

we have

$$(6) \geq 0$$

Which implies k is positive semi-definite, thus is a valid kernel.

b. It's trival to see that K is symmetric let $M^{-1} = diag(e^{x_1^2}, e^{x_2^2}, \ldots, e^{x_n^2})$ then we have

$$(M^{-1})^T K M^{-1} = H = A^T A$$

where $H_{ij} = e^{2x_i x_j}$ and $A = (e^{\sqrt{2}x_1} \; e^{\sqrt{2}x_2} \; \ldots \; e^{\sqrt{2}x_n})$ which implies that $K = M^T A^T A M$, thus K is positive semi-definite. So K is a valid kernel.

Question 3

a. In this question, the "best" attribute is decided by it's information gain. We have following definition for information gain.

The entropy of a r.v. X with distribution $(p(x_1), p(x_2), \ldots, p(x_n))$

$$H(X) = \sum_{i=1}^{n} -p(x_i) log_2 p(x_i)$$

The conditional entropy

$$H(X|Y) = \sum_{j=1}^{m} p(y_j) H(X|y_j)$$

The information gain due to Y

$$IG(X|Y) = H(X) - H(X|Y)$$

Here is the output when running ***python dstumpIG.py Mushroom.csv***

```
Fold  0  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0150437636761  testData errorRate: 0.012315270936
Fold  6  trainingData errorRate:  0.00724835886214  testData errorRate: 0.0825123152709
Fold  7  trainingData errorRate:  0.011624726477  testData errorRate: 0.0431034482759
Fold  8  trainingData errorRate:  0.0161378555799  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.015590809628  testData errorRate: 0.00738916256158
Train Mean ErrorRate: 0.0147702407002  Test Mean ErrorRate: 0.0147783251232
Train StdVar ErrorRate: 0.00287329344241  Test Mean ErrorRate: 0.0258737951366
========= the decision tree ============
 # 5 Attribute
 |
 -- 1 -> Label: 1
 |
 -- 2 -> Label: -1
 |
 -- 3 -> Label: -1
 |
 -- 4 -> Label: -1
 |
 -- 5 -> Label: 1
 |
 -- 6 -> Label: 1
 |
 -- 7 -> Label: -1
 |
 -- 8 -> Label: -1
 |
 -- 9 -> Label: -1
```

Our **DTree** class will recusively print out the tained tree

b. In this question, we are using Gini index, which is defined as

$$Gini(X) = \sum_{i \neq j} p(i)p(j)$$

Conditional Gini index

$$Gini(X|Y) = \sum_{j} p(y_j) Gini(X|y_j)$$

Gini gain

$$GiniGain(X|Y) = Gini(X) - Gini(X|Y)$$

Here is the output when running ***python dtree2GI.py Mushroom.csv***

```
Fold  0  trainingData errorRate:  0.00656455142232  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.00656455142232  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.00656455142232  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.00656455142232  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.00656455142232  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.00588074398249  testData errorRate: 0.0061576354679
Fold  6  trainingData errorRate:  0.00259846827133  testData errorRate: 0.0357142857143
Fold  7  trainingData errorRate:  0.00574398249453  testData errorRate: 0.0073891625615
Fold  8  trainingData errorRate:  0.00629102844639  testData errorRate: 0.0024630541871
Fold  9  trainingData errorRate:  0.00574398249453  testData errorRate: 0.0073891625615
Train Mean ErrorRate: 0.00590809628009  Test Mean ErrorRate: 0.00591133004926
Train StdVar ErrorRate: 0.0011536704983  Test Mean ErrorRate: 0.0103887175906
========= the decision tree ============
 # 5 Attribute
 |
 -- 1 ->
       # 1 Attribute
       |
       -- 2 -> Label: 1
       |
       -- 3 -> Label: 1
       |
       -- 6 -> Label: 1
 |
 -- 2 ->
       # 1 Attribute
       |
       -- 6 -> Label: -1
 |
 -- 3 ->
       # 1 Attribute
       |
       -- 3 -> Label: -1
       |
       -- 4 -> Label: -1
       |
       -- 6 -> Label: -1
 |
 -- 4 ->
       # 1 Attribute
       |
       -- 3 -> Label: -1
```

```
      |
      -- 4 -> Label: -1
      |
      -- 6 -> Label: -1
|
-- 5 ->
      # 1 Attribute
      |
      -- 2 -> Label: 1
      |
      -- 3 -> Label: 1
      |
      -- 6 -> Label: 1
|
-- 6 ->
      # 19 Attribute
      |
      -- 1 -> Label: 1
      |
      -- 2 -> Label: 1
      |
      -- 3 -> Label: 1
      |
      -- 4 -> Label: 1
      |
      -- 5 -> Label: 1
      |
      -- 6 -> Label: -1
      |
      -- 8 -> Label: 1
      |
      -- 9 -> Label: 1
|
-- 7 ->
      # 1 Attribute
      |
      -- 3 -> Label: -1
      |
      -- 6 -> Label: -1
|
-- 8 ->
      # 1 Attribute
      |
      -- 3 -> Label: -1
      |
```

```
       -- 4 -> Label: -1
       |
       -- 6 -> Label: -1
 |
 -- 9 ->
       # 1 Attribute
       |
       -- 3 -> Label: -1
       |
       -- 4 -> Label: -1
       |
       -- 6 -> Label: -1
```

Question 4

a. AdaBoost using the addictive model, it assumes that the final classfier could be represented as

$$g(x) = sign\big[\sum_{t=1}^{T} \alpha_t G_t(x)\big]$$

it is trying to minimize the following error function

$$E = \sum_{i=1}^{N} exp(-yG_t(x_i))$$

suppose the initial distribution on data X is W, then we have the following update scheme, for $t = 1, 2, \dots, T$:

- try to obtain a G_t in the hypothesis space that minimize $\epsilon_t = Pr_{x \sim W_t}[G_t(x) \neq y]$
- choose $\alpha_t = \frac{1}{2}\big(\frac{1-\epsilon_t}{\epsilon_t}\big)$
- update $W_{t+1} = \frac{W_t(i)exp(-\alpha_t y_i G_t(x_i))}{Z_t}$

where Z_t is a normalization factor.

Following is the input and output:

```
$ python myAdaBoost.py Mushroom.csv 5
T = 5
Fold  0  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0150437636761  testData errorRate: 0.012315270936
Fold  6  trainingData errorRate:  0.00793216630197  testData errorRate: 0.0665024630542
Fold  7  trainingData errorRate:  0.011624726477  testData errorRate: 0.0431034482759
Fold  8  trainingData errorRate:  0.0161378555799  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.011761487965  testData errorRate: 0.0221674876847
Train Mean ErrorRate: 0.0144556892779  Test Mean ErrorRate: 0.0146551724138
```

```
Train StdVar ErrorRate: 0.00283069851513  Test Mean ErrorRate: 0.0218674966678


$ python myAdaBoost.py Mushroom.csv 10
T = 10
Fold  0  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.00929978118162  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0175054704595  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0138129102845  testData errorRate: 0.0135467980296
Fold  6  trainingData errorRate:  0.00793216630197  testData errorRate: 0.0665024630542
Fold  7  trainingData errorRate:  0.00998358862144  testData errorRate: 0.0480295566502
Fold  8  trainingData errorRate:  0.0150437636761  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.0166849015317  testData errorRate: 0.00738916256158
Train Mean ErrorRate: 0.0121444201313  Test Mean ErrorRate: 0.0137931034483
Train StdVar ErrorRate: 0.0031661392124  Test Mean ErrorRate: 0.0225191051623


$ python myAdaBoost.py Mushroom.csv 20
T = 20
Fold  0  trainingData errorRate:  0.0109409190372  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0120350109409  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0120350109409  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0158643326039  testData errorRate: 0.0172413793103
Fold  5  trainingData errorRate:  0.0138129102845  testData errorRate: 0.0135467980296
Fold  6  trainingData errorRate:  0.00615426695842  testData errorRate: 0.0628078817734
Fold  7  trainingData errorRate:  0.00765864332604  testData errorRate: 0.0295566502463
Fold  8  trainingData errorRate:  0.0106673960613  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.00683807439825  testData errorRate: 0.012315270936
Train Mean ErrorRate: 0.0106400437637  Test Mean ErrorRate: 0.0137931034483
Train StdVar ErrorRate: 0.00291324441044  Test Mean ErrorRate: 0.018853227816

$ python myAdaBoost.py Mushroom.csv 40
T = 40
Fold  0  trainingData errorRate:  0.0125820568928  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0103938730853  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0158643326039  testData errorRate: 0.0172413793103
Fold  5  trainingData errorRate:  0.0138129102845  testData errorRate: 0.0135467980296
Fold  6  trainingData errorRate:  0.00533369803063  testData errorRate: 0.0431034482759
Fold  7  trainingData errorRate:  0.0082056892779  testData errorRate: 0.0295566502463
Fold  8  trainingData errorRate:  0.00902625820569  testData errorRate: 0.0024630541871
```

```
Fold  9  trainingData errorRate:  0.00902625820569  testData errorRate: 0.012315270936
Train Mean ErrorRate: 0.0105032822757  Test Mean ErrorRate: 0.0118226600985
Train StdVar ErrorRate: 0.00283581466702  Test Mean ErrorRate: 0.0140437293499
```

b. For LogitBoost, There is no big change comparing with AdaBoost. As pointed out in [1], the only modification is to let the distribution $W_t(i)$ be proportional to

$$\frac{1}{1 + exp(y_i f_{t-1}(x_i))}$$

where $f_t = \sum_{j=1}^{t} \alpha_j G_j$ is the current tained model. The detail of derivation could be found in [2]. But it is a little bit beyond my ability to fully understand it..

c. Algorithm for this question was discribed in Question 4.(b). Here is the input and output

```
$ python myLogitBoost.py Mushroom.csv 5
T = 5
Fold  0  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0150437636761  testData errorRate: 0.012315270936
Fold  6  trainingData errorRate:  0.00724835886214  testData errorRate: 0.0825123152709
Fold  7  trainingData errorRate:  0.011624726477  testData errorRate: 0.0431034482759
Fold  8  trainingData errorRate:  0.0161378555799  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.015590809628  testData errorRate: 0.00738916256158
Train Mean ErrorRate: 0.0147702407002  Test Mean ErrorRate: 0.0147783251232
Train StdVar ErrorRate: 0.00287329344241  Test Mean ErrorRate: 0.0258737951366
```

```
$ python myLogitBoost.py Mushroom.csv 10
T = 10
Fold  0  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0150437636761  testData errorRate: 0.012315270936
Fold  6  trainingData errorRate:  0.00724835886214  testData errorRate: 0.0825123152709
Fold  7  trainingData errorRate:  0.011624726477  testData errorRate: 0.0431034482759
Fold  8  trainingData errorRate:  0.0161378555799  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.015590809628  testData errorRate: 0.00738916256158
Train Mean ErrorRate: 0.0147702407002  Test Mean ErrorRate: 0.0147783251232
Train StdVar ErrorRate: 0.00287329344241  Test Mean ErrorRate: 0.0258737951366
```

```
$ python myLogitBoost.py Mushroom.csv 20
T = 20
Fold  0  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0150437636761  testData errorRate: 0.012315270936
Fold  6  trainingData errorRate:  0.00724835886214  testData errorRate: 0.0825123152709
Fold  7  trainingData errorRate:  0.011624726477  testData errorRate: 0.0431034482759
Fold  8  trainingData errorRate:  0.0161378555799  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.015590809628  testData errorRate: 0.00738916256158
Train Mean ErrorRate: 0.0147702407002  Test Mean ErrorRate: 0.0147783251232
Train StdVar ErrorRate: 0.00287329344241  Test Mean ErrorRate: 0.0258737951366
```

```
$ python myLogitBoost.py Mushroom.csv 40
T = 40
Fold  0  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  1  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  2  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  3  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  4  trainingData errorRate:  0.0164113785558  testData errorRate: 0.0
Fold  5  trainingData errorRate:  0.0150437636761  testData errorRate: 0.012315270936
Fold  6  trainingData errorRate:  0.00724835886214  testData errorRate: 0.0825123152709
Fold  7  trainingData errorRate:  0.011624726477  testData errorRate: 0.0431034482759
Fold  8  trainingData errorRate:  0.0161378555799  testData errorRate: 0.00246305418719
Fold  9  trainingData errorRate:  0.015590809628  testData errorRate: 0.00738916256158
Train Mean ErrorRate: 0.0147702407002  Test Mean ErrorRate: 0.0147783251232
Train StdVar ErrorRate: 0.00287329344241  Test Mean ErrorRate: 0.0258737951366
```

References

[1] Schapire, Robert E. *"The boosting approach to machine learning: An overview."* LECTURE NOTES IN STATISTICS-NEW YORK-SPRINGER VERLAG- (2003): 149-172.

[2] Solla, Sara A., Todd K. Leen, and Klaus-Robert Mller. *Advances in neural information processing systems.* The MIT Press, 2000.