1.a

Min(S)>= v function is an **anti-monotonicity** and **succinct** constraint.

Set:
S :

Item	Profit	Price
A	-30	30
B	0	5
C	50	25
D	20	20
E	-10	50

v = 25

For min(price)>=25, B and D fail, we can exclude all subsets of B or D.

A, C, E meet the requirement, all sets in S meet the requirement must be a subset belong to them.

1.b

Max(S)<= v function is an **anti-monotonicity** and **succinct** constraint.

Set:
S =

Item	Profit	Price
A	-30	30
B	0	5
C	50	25
D	20	20
E	-10	50

v = 25

For max(price)<=25, A and E fail, we can exclude all subsets of A or E.
B, C, D meet the requirement, all sets in S meet the requirement must be a subset belong to them.

1.c

T

1.d

Anti-monotonic will be more effective if few sets cannot meet the requirement (fail). Since anti-monotonic is fail once fail all, when there are few sets to be pruned, it would be a better solution.

Monotonic will be more effective if few sets can meet the requirement (succeed). Since monotonic is succeed once succeed forever, we can easily find the sets contain element meet requirement and get the right answer.

2.a

Since the minsup for d and e <3, the b's projected database should be:

(_c)(_)f
(_)c(f_)
(_c)f (bc)(f)

2.b

bb	1
bc	3
(bc)	2
bcf	3
bf	3

The frequent pattern should be <bc>, <bcf>and <bf>.

2.c

No candidate sequence needs to be generated and projected (parititioned) databases keep shrinking.

Q3 Decision Tree

In [73]:

```
import math
```

In [97]:

```
def exp_inf(p):
    s = sum(p)
    inf = 0
    for i in p:
        if i == 0:
            inf += 0
        else:
            inf += (float(i)/s)*math.log(float(i)/s,2)
    return abs(inf)
```

In [99]:

```
exp_inf([3,2])
```

Out[99]:

```
0.9709505944546686
```

In [76]:

```
a = [['Sunny','Hot','High','Weak','No'],
['Sunny','Hot','High','Strong','No'],
['Overcast','Hot','High','Weak','Yes'],
['Rain','Mild','High','Weak','Yes'],
['Rain','Cool','Normal','Weak','Yes'],
['Rain','Cool','Normal','Strong','No'],
['Overcast','Cool','Normal','Strong','Yes'],
['Sunny','Mild','High','Weak','No'],
['Sunny','Cool','Normal','Weak','Yes'],
['Rain','Mild','Normal','Weak','Yes'],
['Sunny','Mild','Normal','Strong','Yes'],
['Overcast','Mild','High','Strong','Yes'],
['Overcast','Hot','Normal','Weak','Yes'],
['Rain','Mild','High','Strong','No']]
```

In [77]:

```
sunny = [0,0] overcast = [0,0] rain = [0,0] hot = [0,0] mild = [0,0]
cool = [0,0] high = [0,0] normal = [0,0] strong = [0,0] weak = [0,0]
length = len(a)
```

In [78]:

```
for i in range(len(a)):
    if a[i][0] == 'Sunny':
        if a[i][4] == 'Yes':
            sunny[0] += 1
        else:
            sunny[1] += 1
    elif a[i][0] == 'Overcast':
        if a[i][4] == 'Yes':
            overcast[0] += 1
        else:
            overcast[1] += 1
    elif a[i][0] == 'Rain':
        if a[i][4] == 'Yes':
            rain[0] += 1
        else:
            rain[1] += 1
```

In [79]:

```
for i in range(len(a)):
    if a[i][1] == 'Hot':
        if a[i][4] == 'Yes':
            hot[0] += 1
        else:
            hot[1] += 1
    elif a[i][1] == 'Mild':
        if a[i][4] == 'Yes':
            mild[0] += 1
        else:
            mild[1] += 1
    elif a[i][1] == 'Cool':
        if a[i][4] == 'Yes':
            cool[0] += 1
        else:
            cool[1] += 1
```

In [80]:

```
for i in range(len(a)):
    if a[i][2] == 'High':
        if a[i][4] == 'Yes':
            high[0] += 1
        else:
            high[1] += 1
    elif a[i][2] == 'Normal':
        if a[i][4] == 'Yes':
            normal[0] += 1
        else:
            normal[1] += 1
```

In [81]:

```
for i in range(len(a)):
    if a[i][3] == 'Strong':
        if a[i][4] == 'Yes':
            strong[0] += 1
        else:
            strong[1] += 1
    elif a[i][3] == 'Weak':
        if a[i][4] == 'Yes':
            weak[0] += 1
        else:
            weak[1] += 1
```

In [82]:

```
Total_yes = strong[0] + weak[0]
Total_no = strong[1] + weak[1]
Total_inf = exp_inf([Total_yes,Total_no])
```

In [83]:

```
Total_inf
```

Out[83]:

```
0.9402859586706309
```

In [85]:

```
def one_inf(one):
    return float(sum(one))/length*exp_inf(one)
```

In [103]:

```
Gain_weather = Total_inf - one_inf(sunny) - one_inf(overcast)- one_inf(rain)
Gain_temp = Total_inf - one_inf(hot) - one_inf(mild)- one_inf(cool)
Gain_hum = Total_inf - one_inf(high) - one_inf(normal)
Gain_wind = Total_inf - one_inf(weak) - one_inf(strong)
```

In [104]:

```
Gain_weather
```

Out[104]:

```
0.2467498197744391
```

In [105]:

```
Gain_temp
```

Out[105]:

```
0.029222565658954647
```

In [106]:

```
Gain_hum
```

Out[106]:

```
0.15183550136234136
```

In [107]:

```
Gain_wind
```

Out[107]:

```
0.04812703040826932
```

Since Gain_weather is highest, it is the root

In [108]:

```
print exp_inf(sunny),exp_inf(overcast),exp_inf(rain)
```

```
0.970950594455 0.0 0.970950594455
```

In [117]:

```
float(overcast[0])/sum(overcast)
```

Out[117]:

```
1.0
```

We can find overcast is 0, there is no need to extend

In [109]:

```
sunny_hot = [0,0]
sunny_mild = [0,0]
sunny_cool = [0,0]
rain_hot = [0,0]
rain_mild = [0,0]
rain_cool = [0,0]
sunny_high = [0,0]
sunny_normal = [0,0]
rain_high = [0,0]
rain_normal = [0,0]
sunny_strong = [0,0]
sunny_weak = [0,0]
rain_strong = [0,0]
rain_weak = [0,0]
```

In [110]:

```
for i in range(len(a)):
    if a[i][0] == 'Sunny':
        if a[i][1] == 'Hot':
            if a[i][4] == 'Yes':
                sunny_hot[0] += 1
            else:
                sunny_hot[1] += 1
        elif a[i][1] == 'Mild':
            if a[i][4] == 'Yes':
                sunny_mild[0] += 1
            else:
                sunny_mild[1] += 1
        elif a[i][1] == 'Cool':
            if a[i][4] == 'Yes':
                sunny_cool[0] += 1
            else:
                sunny_cool[1] += 1
    elif a[i][0] == 'Rain':
        if a[i][1] == 'Hot':
            if a[i][4] == 'Yes':
                rain_hot[0] += 1
            else:
                rain_hot[1] += 1
        elif a[i][1] == 'Mild':
            if a[i][4] == 'Yes':
                rain_mild[0] += 1
            else:
                rain_mild[1] += 1
        elif a[i][1] == 'Cool':
            if a[i][4] == 'Yes':
                rain_cool[0] += 1
            else:
                rain_cool[1] += 1
```

In [111]:

```
for i in range(len(a)):
    if a[i][0] == 'Sunny':
        if a[i][2] == 'High':
            if a[i][4] == 'Yes':
                sunny_high[0] += 1
            else:
                sunny_high[1] += 1
        elif a[i][2] == 'Normal':
            if a[i][4] == 'Yes':
                sunny_normal[0] += 1
            else:
                sunny_normal[1] += 1
    elif a[i][0] == 'Rain':
        if a[i][2] == 'High':
            if a[i][4] == 'Yes':
                rain_high[0] += 1
            else:
                rain_high[1] += 1
        elif a[i][2] == 'Normal':
            if a[i][4] == 'Yes':
                rain_normal[0] += 1
            else:
                rain_normal[1] += 1
```

In [112]:

```
for i in range(len(a)):
    if a[i][0] == 'Sunny':
        if a[i][3] == 'Strong':
            if a[i][4] == 'Yes':
                sunny_strong[0] += 1
            else:
                sunny_strong[1] += 1
        elif a[i][3] == 'Weak':
            if a[i][4] == 'Yes':
                sunny_weak[0] += 1
            else:
                sunny_weak[1] += 1


    elif a[i][0] == 'Rain':
        if a[i][3] == 'Strong':
            if a[i][4] == 'Yes':
                rain_strong[0] += 1
            else:
                rain_strong[1] += 1
        elif a[i][3] == 'Weak':
            if a[i][4] == 'Yes':
                rain_weak[0] += 1
            else:
                rain_weak[1] += 1
```

In [113]:

```
Gain_sunny_temp =one_inf(sunny) - one_inf(sunny_hot) - one_inf(sunny_mild)- one_inf(sunny_c
Gain_sunny_hum = one_inf(sunny)- one_inf(sunny_high) - one_inf(sunny_normal)
Gain_sunny_wind = one_inf(sunny) - one_inf(sunny_weak) - one_inf(sunny_strong)
Gain_rain_temp =one_inf(rain)- one_inf(rain_hot) - one_inf(rain_mild)- one_inf(rain_cool)
Gain_rain_hum =one_inf(rain)- one_inf(rain_high) - one_inf(rain_normal)
Gain_rain_wind =one_inf(rain) - one_inf(rain_weak) - one_inf(rain_strong)
```

In [114]:

```
print Gain_sunny_temp,Gain_sunny_hum,Gain_sunny_wind
```

```
0.203910926591 0.346768069448 0.00713324786499
```

In [115]:

```
print Gain_rain_temp, Gain_rain_hum, Gain_rain_wind
```

```
0.00713324786499 0.00713324786499 0.346768069448
```

In [119]:

```
float(rain_strong[0])/sum(rain_strong)
```

Out[119]:

```
0.0
```

In [120]:

```
float(rain_weak[0])/sum(rain_weak)
```

Out[120]:

```
1.0
```

if we set a prune parameter that all value under 0.01 can be ignored

we can find in sunny condition, humidity is higher than temp

In [122]:

```
sunny_high_hot =[0,0]
sunny_high_mild =[0,0]
sunny_high_cool =[0,0]
sunny_normal_hot =[0,0]
sunny_normal_mild =[0,0]
sunny_normal_cool =[0,0]
```

In [123]:

```
for i in range(len(a)):
    if a[i][0] == 'Sunny':
        if a[i][2] == 'High':
            if a[i][1] == 'Hot':
                if a[i][4] == 'Yes':
                    sunny_high_hot[0] += 1
                else:
                    sunny_high_hot[1] += 1
            elif a[i][1] == 'Mild':
                if a[i][4] == 'Yes':
                    sunny_high_mild[0] += 1
                else:
                    sunny_high_mild[1] += 1
            elif a[i][1] == 'Cool':
                if a[i][4] == 'Yes':
                    sunny_high_cool[0] += 1
                else:
                    sunny_high_cool[1] += 1
        elif a[i][2] == 'Normal':
            if a[i][1] == 'Hot':
                if a[i][4] == 'Yes':
                    sunny_normal_hot[0] += 1
                else:
                    sunny_normal_hot[1] += 1
            elif a[i][1] == 'Mild':
                if a[i][4] == 'Yes':
                    sunny_normal_mild[0] += 1
                else:
                    sunny_normal_mild[1] += 1
            elif a[i][1] == 'Cool':
                if a[i][4] == 'Yes':
                    sunny_normal_cool[0] += 1
                else:
                    sunny_normal_cool[1] += 1
```

In [124]:

```
Gain_sunny_high_temp = one_inf(sunny_high)-one_inf(sunny_high_hot) - one_inf(sunny_high_mil
Gain_sunny_normal_temp = one_inf(sunny_normal)-one_inf(sunny_normal_hot) - one_inf(sunny_no
```

In [125]:

```
print Gain_sunny_high_temp,Gain_sunny_normal_temp
```

```
0.0 0.0
```

we need to exclude all 0, so there is no need to extend from wind to temperature

In [130]:

```
float(sunny_normal[0])/sum(sunny_normal)
```

Out[130]:

```
1.0
```

In [131]:

```
float(sunny_high[0])/sum(sunny_high)
```

Out[131]:

```
0.0
```

3.a

the decesion tree should be:

```
    Outlook-sunny    -- Humidity ---high   ----No
                                 ---Normal ----Yes
           -overcast -- Yes
           -rain     -- Wind     ---strong ---- No
                                 ---weak   ---- Yes
```

3.b

for the 1st one, it predict to be Yes but actually No;

for the 2nd one, it predict to be Yes and actually Yes;

for the 3rd one, it predict to be No and actually No;

for the 4th one, it predict to be Yes and actually Yes.

3.b

Confusion matrix:

Actual class/ predicted class	Play(predicted)	Not Play(predicted)
Play(actual)	2	0
Not Play(actual)	1	1

Accuracy rate = (2+1)/4 = 0.75

Precision = 2/ (2+1) = 0.67

Recall = 2/ (2+0) = 1

3.c

In [138]:

```
def splitinf(D):
    inf = 0
    for i in D:
        inf += float(i)/sum(D)*math.log(float(i)/sum(D),2)
    return abs(inf)
```

In [139]:

```
gainratio_weather = float(Gain_weather)/splitinf([sum(sunny),sum(overcast),sum(rain)])
gainratio_temp = float(Gain_temp)/splitinf([sum(hot),sum(mild),sum(cool)])
gainratio_hum = float(Gain_hum)/splitinf([sum(high),sum(normal)])
gainratio_wind = float(Gain_wind)/splitinf([sum(weak),sum(strong)])
```

In [140]:

```
print gainratio_weather,gainratio_temp, gainratio_hum,gainratio_wind
```

```
0.156427562421 0.0187726462224 0.151835501362 0.0488486155115
```

we can find the weather (Outlook) is still the first split when using gain ratio

3.d.1

If the outlook is sunny is true and humidity is high is true, then not play tennis +1;

If the outlook is sunny is true and humidity is high is false, then play tennis+1;

If the outlook is sunny is false, is overcast is true, then play tennis+1;

If the outlook is sunny is false, is overcast is false, and wind is strong is true, then not play tennis+1;

If the outlook is sunny is false, is overcast is false, and wind is strong is false, then play tennis+1.

3.d.2

You can set an n-channel root to n-1 true and anther 1 false question;

4.

P(H|X) = P(X|H) *P(H)/P(X)

4.a

P (play = “yes”) = 9/14 = 0.643

P (play = “no”) = 5/14 = 0.357

4.b

For outlook:

P (O|yes) = 4/9 = 0.44

P (R|yes) = 3/9 = 0.33

P (S|yes) = 2/9 = 0.22

For temperature:

P (Hot|yes) = 2/9 = 0.22

P (Mild|yes) = 4/9 = 0.44

P (Cool|yes) = 3/9 = 0.33

For humidity:

P (High|yes) = 3/9 = 0.33

P (Normal|yes) = 6/9 = 0.67

For wind:

P (Strong|yes) = 3/9 = 0.33

P (Weak|yes) = 6/9 = 0.57

4.C

For outlook:

P (O|no) = 0/5 = 0

P (R|no) = 2/5 = 0.4

P (S|no) = 3/5 = 0.6

For temperature:

P (Hot|no) = 2/5 = 0.4

P (Mild|no) = 2/5 = 0.4

P (Cool|no) = 1/5 = 0.2

For humidity:

P (High|no) = 4/5 = 0.8

P (Normal|no) = 1/5 = 0.2

For wind:

P (Strong|no) = 3/5 = 0.6

P (Weak|no) = 2/5 = 0.4

4.d

For x1 = [overcast, hot, high, strong]:

P (x1|yes) = 0.44*0.22*0.33*0.33 = 0.01

p (x1|no) = 0

P (yes|x1) = 0.01*0.64 = 0.06

P (no|x1) = 0

X1 belongs to yes.

For x2 = [sunny, hot, normal, weak]:

P (x2|yes) = 0.22*0.22*0.67*0.67 = 0.022

p (x2|no) = 0.6*0.4*0.2*0.4 = 0.019

P (yes|x2) = 0.022*0.64 = 0.014

P (no|x2) = 0.019*0.36 = 0.007

X2 belongs to yes.

For x3 = [rain, mild, normal, strong]:

P (x3|yes) = 0.33*0.44*0.67*0.33 = 0.032

p (x3|no) = 0.4*0.4*0.2*0.6 = 0.019

P (yes|x3) = 0.032*0.64 = 0.020

P (no|x3) = 0.019*0.36 = 0.007

X3 belongs to yes.

For x4 = [overcast, cool, high, strong]:

P (x4|yes) = 0.44*0.33*0.33*0.33 = 0.015

p (x4|no) = 0

P (yes|x4) = 0.015*0.64 = 0.01

P (no|x4) = 0

X4 belongs to yes.

We can find the result are all “yes” for play tennis.

4.e

Confusion matrix:

Actual class/ predicted class	Play(predicted)	Not Play(predicted)
Play(actual)	2	0
Not Play(actual)	2	0

Precision = 2/ (2+2) = 0.5

Recall = 2/ (2+0) = 1

4.f

Decision Tree: Pros: Easy to understand Cons: Poor accuracy for unseen samples

Naïve Bayes: Pros: Easy to implement Cons: loss of accuracy

Q5 AdaBoost

In [1]:

```
Turples = [[0,-1],[1,-1],[2,-1],[3,1],[4,1],[5,1],[6,1],[7,1],[8,-1],[9,-1]]
```

In [24]:

```
def classfier_M(turple,v):
    error = 0
    if turple[0] < v:
        if turple[1] != -1:
            error = 1
    elif turple[0] > v:
        if turple[1] != 1:
            error = 1
    return error
```

5.a

In [28]:

```
w = [float(1)/10]*10
for v in range(9):
    e = [0]*10
    for i in range(10):
        e[i] = float(w[i])*classfier_M(Turples[i],v)

    print sum(e)
```

```
0.4
0.3
0.2
0.2
0.3
0.4
0.5
0.6
0.6
```

we can find the value is symmetric, and the optimation point is 2 and 3

So the classfier should be :

v = 2 or 3: if x < v, y = -1; if x > v, y = 1

5.b

The weighted error rate for all 10 data are:

In [30]:

```
v = 2
e = [0]*10
for i in range(10):
    e[i] = float(w[i])*classfier_M(Turples[i],v)
    print e[i]
```

```
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.1
0.1
```

5.c

In [33]:

```
w2 = [0]*10
for i in range(10):
    w2[i] = float(w[i])*e[i]/(1-e[i])
```

In [36]:

```
w2_norm = [0]*10
for i in range(10):
    w2_norm[i] = float(w2[i])/sum(w2)
```

In [38]:

```
w2_norm
```

Out[38]:

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5]
```

5.d

In [40]:

```
for v in range(10):
    e = [0]*10
    for i in range(10):
        e[i] = float(w2_norm[i])*classfier_M(Turples[i],v)

    print sum(e)
```

```
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
0.5
0.0
```

we can find the value is symmetric, and the optimation point is 10

So the classfier should be :

v = 10: if x < v, y = -1; if x > v, y = 1

5.e

```
The ensemble h should be:

    v = 2 or 3:
    if x < v, y = -1;
    if x > v, y = 1;

If the classifier answer is wrong, for the set of wrong data:
    v = 10:
    if x < v, y = -1;
    if x > v, y = 1
```