CS 267A - Homework 2

Peter Racioppo (103953689)

April 24, 2020

Problem 1

Topics: Logic

Let α, β, and γ be Boolean formulae, and let MC denote the model count of a Boolean formula. Select whether the following is a true or false statement about model counts and provide a brief justification for your choice.

$MC(\alpha) + MC(\beta) + MC(\gamma) - MC(\alpha \wedge \beta \wedge \gamma) = MC(\alpha \vee \beta \vee \gamma)$

FALSE. Let $A = MC(\alpha), B = MC(\beta), C = MC(\gamma)$. Then, $A \vee B \vee C = A \vee (B \vee C) = A + (B \vee C) - A \wedge (B \vee C) - A \wedge B \vee A \wedge C = (A + B + C - B \wedge C) - (A \wedge B + A \wedge C - A \wedge B \wedge A \wedge C) = A + B + C - A \wedge B - A \wedge C - B \wedge C + A \wedge B \wedge C$. That is, $MC(\alpha \vee \beta \vee \gamma) = MC(\alpha) + MC(\beta) + MC(\gamma) - MC(\alpha \wedge \beta) - MC(\alpha \wedge \gamma) - MC(\beta \wedge \gamma) + MC(\alpha \wedge \beta \wedge \gamma)$.

Problem 2

Topics: Probability rules

Suppose that α and β are independent events, that is $Pr(\alpha \wedge \beta) = Pr(\alpha)Pr(\beta)$. Using basic probability rules, show that $Pr(\neg\alpha \wedge \neg\beta) = Pr(\neg\alpha)Pr(\neg\beta)$.

$Pr(\alpha \wedge \beta) = Pr(\alpha)Pr(\beta) = (1 - Pr(\neg\alpha))(1 - Pr(\neg B)) = 1 + Pr(\neg\alpha)Pr(\neg\beta) - Pr(\neg\alpha) - Pr(\neg\beta) = 1 + Pr(\neg\alpha)Pr(\neg\beta) - (1 - Pr(\alpha)) - (1 - Pr(\beta)) = Pr(\neg\alpha)Pr\neg\beta) + Pr(\alpha) + Pr(\beta) - 1$.

$Pr(\neg\alpha \wedge \neg\beta) = 1 - Pr(\alpha \vee \beta) = 1 - Pr(\alpha) - Pr(\beta) + Pr(\alpha \wedge \beta) = 1 - Pr(\alpha) - Pr(\beta) + Pr(\neg\alpha)Pr\neg\beta) + Pr(\alpha) + Pr(\beta) - 1 = Pr(\neg\alpha)Pr(\neg\beta)$. That is, $Pr(\neg\alpha \wedge \neg\beta) = Pr(\neg\alpha)Pr(\neg\beta)$.

Topics: Probability rules

Suppose there are 4 random variables A, B, C, D, and we know that A and B are independent given D. Please evaluate whether each of the following statements must be true, and state how you reach your conclusion:

$Pr(A, D) = Pr(B, D)$
$Pr(A, B) = Pr(A) \cdot Pr(B)$
$Pr(A, B, C, D) = Pr(A|C, D) \cdot Pr(B|C, D) \cdot Pr(C|D) \cdot Pr(D)$
$Pr(A, B, C) = Pr(A|B, C) \cdot Pr(B|C) \cdot Pr(C)$

We know that $Pr(A|D)$ and $Pr(B|D)$ are independent, that is $Pr(A, B|D) = Pr(A|D)Pr(B|D)$, or equivalently, $Pr(A|B, D) = Pr(A|D)$.

Statement 1 is not necessarily true. $Pr(A, D) = Pr(B, D) \iff Pr(A|D)Pr(D) = Pr(B|D)Pr(D) \iff Pr(A|D) = Pr(B|D)$, but this may not be true.

Statement 2 may also be false. This is a statement of independence, not conditional independence.

$Pr(A, B, C, D) = Pr(A, B, C|D)Pr(D) = Pr(A, B|C, D)Pr(C|D)Pr(D)$
$= Pr(A|B, C, D)Pr(B|C, D)Pr(C|D)Pr(D)$. Thus, to show Statement 3, it remains to show that $Pr(A|B, C, D) = Pr(A|C, D)$. But this follows from the conditional independence of A and B wrt D. Thus, Statement 2 is true.

Statement 4 is true by the chain rule. $Pr(A, B, C) = Pr(A, B|C)Pr(C) = Pr(A|B, C)Pr(B|C)Pr(C)$.

Problem 4

Topics: Bayes rule

Suppose there is a rare and terrible disease which occurs with probability 10^{-6}. Doctors have developed a miracle diagnosis technique, which has the following table describing its accuracy:

H) Has Disease	(T) Test Positive	Pr(T—H)
T	T	9/10
T	F	1/10
F	T	1/100
F	F	99/100

Suppose you take the test and it comes back positive.
1. What is the probability that you have the disease?

Problem 4 continued on next page. . .

2. Is this a good test? (i.e., would you be worried if it says you have the disease)

By Bayes' Rule, $P(H|T) = P(T|H)P(H)/P(T)$. $P(H) = 10^{-6}$.
$P(T) = P(T|H)P(H) + P(T|\neg H)P(\neg H) = (9/10)(10^{-6}) + (1/100)(1 - 10^{-6}) \approx 0.01$.
$P(T|H) = 9/10$. Thus, $P(H|T) = (9/10)10^{-6}/0.01 = 9 * 10^{-5}$. Your probability of having the disease after testing positive is not high, so the test is not good in this sense.

Problem 5

Topics: Independence and Discrete Probability
Consider the following partially-defined joint probability distribution on two random variables x and y:
The variables θ_1 and θ_2 are unknown numerical quantities.

x	y	Pr(x,y)
0	0	1/32
0	1	θ_1
1	0	θ_2
1	1	21/32

1. What are the constraints on θ_1 and θ_2 so that this table describes a valid probability distribution?

$\theta_1, \theta_2 > 0$.
$\theta_1 + \theta_2 = 1 - 1/32 - 21/32 = 10/32$.

2. Choose θ_1, θ_2 so that the marginal probability $Pr(x = 0) = 1/8$.

$Pr(x = 0) = Pr(x = 0, y = 0) + Pr(x = 0, y = 1) = 1/32 + \theta_1 = 1/8$.
$\Rightarrow \theta_1 = 1/8 - 1/32 = 3/32$.
$\Rightarrow \theta_2 = 10/32 - \theta_1 = 10/32 - 3/32 = 7/32$.

3. Choose θ_1, θ_2 so that the conditional probability $Pr(x = 0|y = 1) = 1/21$.

$Pr(x = 0|y = 1) = Pr(x = 0, y = 1)Pr(y = 1)$.
$Pr(x = 0, y = 1) = \theta_1$.
$Pr(y = 1) = \theta_1 + 21/32$.
Thus, $Pr(x = 0|y = 1)\theta_1/(\theta_1 + 21/32) = 1/21$.
$\Rightarrow 21\theta_1 = \theta_1 + 21/32$
$\Rightarrow \theta_1 = 21/(20 * 32) = 21/640$.
$\Rightarrow \theta_2 = 10/32 - \theta_1 = (200 - 21)/640 = 379/640$.

4. Choose θ_1, θ_2 so that x and y are independent.

For independence, we must have $Pr(x, y) = Pr(x)Pr(y)$.
$\Rightarrow 21/32 = (\theta_2 + 21/32)(\theta_1 + 21/32)$

Problem 5 continued on next page. . .

$\theta_2 = 10/32 - \theta_1$
$\Rightarrow 21/32 = (10/32 - \theta_1 + 21/32)(\theta_1 + 21/32)$
$\Rightarrow \theta_1^2 - (5\theta_1)/16 + 21/1024 = 0$
$\Rightarrow \theta_1 = 3/32, 7/32$
$\theta_1 = 3/32 \Rightarrow \theta_2 = 10/32 - 3/32 = 7/32.$
$\theta_1 = 7/32 \Rightarrow \theta_2 = 10/32 - 7/32 = 3/32.$
Either of these works.

Problem 6

Programming Exercise: Implement a sampler
Let $X = \{X_1, X_2, ..., X_n\}$ be a collection of n independent Boolean random variables, with each variable X_i being true with probability p_i, and false with probability $1p_i$. Let Δ be a CNF which uses $\{X_1, X_2, ..., X_n\}$ as its atoms.
Then, we can interpret Δ as a random variable with the following conditional distribution, where x is an assignment to each variable in X:
$Pr(\Delta = true|X = x) = 0 = 1$ if x is a satisfying assignment to Δ, and 0 otherwise (1).
For this problem, you will be computing the marginal probability $Pr(\Delta$ = true), from the joint distribution $Pr(\Delta, X_1, X_2, ..., X_n)$. Intuitively, this is the probability that Δ is satisfied if we randomly draw assignments to the X variables according to specified distributions.

Part A: *Probability of Satisfaction*
Let $X = \{X_1, X_2\}$ be two independent random variables that are each true with probability 1/2.
Let $\Delta = X_1 \wedge X_2$. What is Pr(Δ = true)?
Let $\Delta = X_1 \vee X_2$. What is Pr(Δ = true)?

In the first case, $Pr(\Delta = true) = P(X_1 = true, X_2 = true) = (1/2)(1/2) = 1/4$.

In the second case, $Pr(\Delta = true) = P(X_1 = true, X_2 = true) + P(X_1 = true, X_2 = false) + P(X_1 = false, X_2 = true) = 1/4 + 1/4 + 1/4 = 3/4$.

Part B: *Write a Sampler*
We want to apply Monte-Carlo sampling to our problem of estimating the probability that Δ is satisfied. Implement a function which performs the following task:

Input:
1. A CNF Δ, specified as a list of lists of integers, where a positive integer is a positive literal and a negative integer is a negated literal.
2. Probabilities: A map w from each variable to its probability of being true. You can implement this as a dictionary.
3. n, the number of samples to draw.

Output:
The approximate probability that $Pr(\Delta = true)$ if we randomly draw n samples according to the distribution specified by w.

Monte Carlo estimates of $Pr(\Delta = true)$:

Part 1:
$Pr(\Delta = true) = 0.569, 0.548, 0.592$

Part 2:
$Pr(\Delta = true) = 0.643, 0.652, 0.657$

Draw Sample: A method which draws a random assignment to variables according to a specified weight function w.

Substitution: A method which substitutes the value of each sampled variable into Δ, and computes whether or not Δ is satisfied under this assignment.

In [124]:

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
import random
import math
```

In [82]:
```
# This function takes a Conjuctive Normal Form (CNF) Boolean formula
# and a list of variable truth values, and computes whether the CNF
# is satisfied under this assignment.
def f_Substitute(Delta,Var):
    # Inputs:
    # A CNF Delta
    # A list Var of the True/False values of each variable

    Var_n = Var*2 - 1 # Replace 0s in Var with -1s
    L = np.shape(Delta)[0] # Number of conjunctions in Delta
    # Conj_Vals is a list of truth values for each conjunction
    # in the CNF.
    Conj_Vals = np.zeros(L) # Initialize Conj_Vals to zeros
    # Loop through the list of conjunctions:
    for conj_i in np.arange(L):
        conj = Delta[conj_i] # A given conjunction
        # tf is the True/False value of this conjunction:
        tf = 0 # (Initialize to zero)
        # Loop through the elements in a given conjunction:
        for i in conj:
            # If an i is negative this corresponds to a negation.
            # Thus, we multiply the sign of i by the truth value
            # of the corresponding variable. This variable is
            # indexed by abs(i)-1. If any of the disjunctions
            # are satisfied, the conjunction is true, so we set
            # tf = 1 and break. Otherwise, tf = 0, and the
            # conjunction is false.
            if (np.sign(i)*Var_n[np.absolute(i)-1]) == 1:
                tf = 1
                break
        # Add the truth value for a given conjunction
        # to the list Conj_Vals:
        Conj_Vals[conj_i] = tf

        # Outputs:
        # 'True' if Conj_Vals contains only 1s
        # 'False' otherwise
    return (np.all(Conj_Vals))
```

In [119]:
```
# This function draws a random assignment to variables
# according to a specified weight function w.
def f_Draw_Sample(w):
    # Input: w, a list of probabilities that each variable is true.

    # Probabilities must be between 0 and 1:
    if np.min(w) < 0 or np.max(w) > 1:
        print("Invalid probability.")

    N = np.size(w) # The number of variables.
    # Var is a list of the True/False values of each variable:
    Var = np.zeros(N) # Initialize to zeros
    # Loop through the variables:
    for i in np.arange(N):
        # Randomly draw 1 or 0, with probability w[i] and 1 - w[i]:
        Var[i] = np.random.choice(np.array((1,0)),p=np.array((w[i],1-w[i])))

    # Output: Var, a list of the True/False values of each variable.
    return Var
```

In [137]:
```
# This function...
def f_Monte_Carlo_Sampler(Delta,w,n=1000):
    # Inputs:
    # Delta, a CNF
    # w, a list of probabilities that each variable is true.
    # n, the number of samples

    # counter counts the number of
    # times the CNF is satisfied:
    counter = 0 # Initialize to zero
    # Sample n times:
    for i in np.arange(n):
        # Randomly draw truth values for each variable,
        # according to the weight vector w:
        Var = f_Draw_Sample(w)
        # Check whether the CNF is satisfied under this
        # assignment and increment the counter.
        counter += f_Substitute(Delta,Var)

    # Output:
    # counter/n, the sample mean of the
    # number of times the CNF was satisfied.
    return (counter/n)
```

In [141]:
```
# Test:
Delta = [[-1,-2,3],[-1,2,4],[2,3,4],[3,4]]
w = np.array((0.1,0.2,0.5,0.3))

f_Monte_Carlo_Sampler(Delta,w)
```

Out[141]:
```
0.631
```

Part C: Evaluate

For each of the following, use n = 1000 samples to compute Pr(Δ = true), given the probabilities for each atom. You should report 3 runs for each question.

1. $(a \lor b \lor \neg c) \land (b \lor c \lor d \lor \neg e) \land (\neg b \lor \neg d \lor e) \land (\neg a \lor \neg b)$ with Pr(a)=0.3, Pr(b)=0.6, Pr(c)=0.1, Pr(d)=0.8, Pr(e)=0.4

2. (¬a∨c∨d)∧(b∨c∨¬d∨e)∧(¬c∨d∨¬e) with Pr(a) = 0.2, Pr(b) = 0.1, Pr(c) = 0.8, Pr(d) =0.3, Pr(e)=0.5

In [170]:
```
# [a,b,c,d,e] = [1,2,3,4,5]

Delta1 = [[1,2,-3],[2,3,4,-5],[-2,-4,5],[-1,-2]]
w1 = np.array((0.3,0.6,0.1,0.8,0.4))

for i in np.arange(3):
    print(f_Monte_Carlo_Sampler(Delta1,w1))
```

```
0.569
0.548
0.592
```

In [172]:
```
Delta2 = [[-1,3,4],[2,3,-4,5],[-3,4,-5]]
w2 = np.array((0.2,0.1,0.8,0.3,0.5))

for i in np.arange(3):
    print(f_Monte_Carlo_Sampler(Delta2,w2))
```

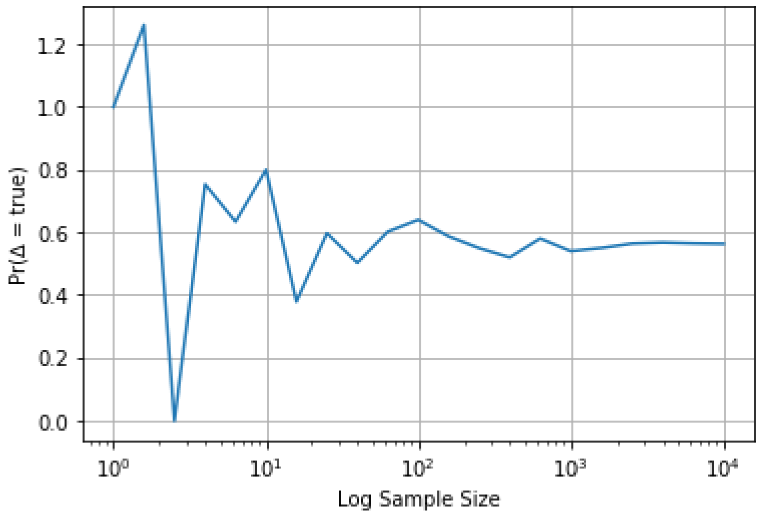
```
0.643
0.652
0.657
```

Plotting how the estimate of Pr(Δ = true) depends on the number of samples:

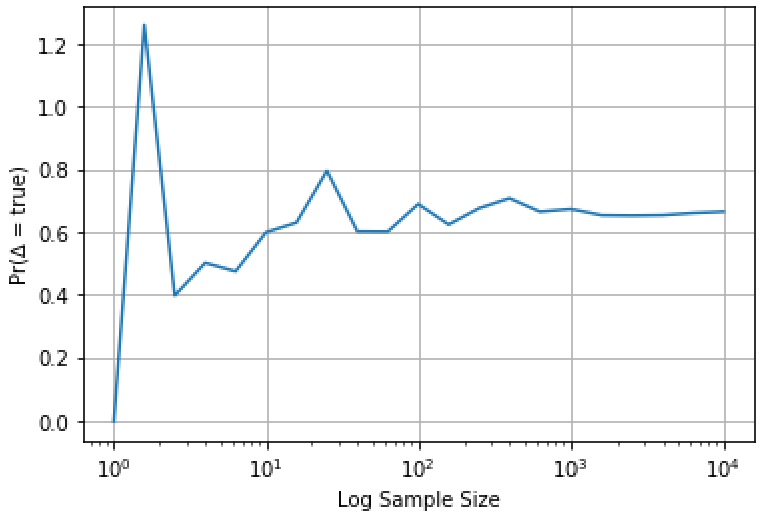
In [173]:
```
# This function computes the Monte Carlo estimate of
# Pr(Δ = true) for a range of sample sizes.
def f_MC_n(Delta,w,order=3):
    # Inputs:
    # Delta, a CNF
    # w, a list of probabilities that each variable is true.
    # order, the max order of the sample size (n <= 10^order)

    ep = np.linspace(0,order,order*5+1) # Vector of exponents
    n_vec = 10**ep # Powers of ten
    # Initialize placeholder for estimated probabilities:
    p_vec = np.zeros(np.size(ep))

    # Loop through the list of sample sizes:
    for i in np.arange(np.size(ep)):
        # Compute the estimate for each sample size:
        p_vec[i] = f_Monte_Carlo_Sampler(Delta,w,n_vec[i])

    # Outputs:
    # n_vec, the sample sizes
    # p_vec, the estimated probabilities
    # for each sample size
    return(n_vec,p_vec)
```

In [174]:
```
# This function plots the estimated
# probabilities vs the sample size
def f_Plot(n_vec,p_vec):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    plt.plot(n_vec,p_vec)
    ax.set_xscale('log')
    plt.xlabel('Log Sample Size')
    plt.ylabel('Pr(Δ = true)')
    plt.grid()
    plt.show()
```

In [176]:
```
# Run for the first CNF:
n_vec,p_vec = f_MC_n(Delta1,w1,order=4)
f_Plot(n_vec,p_vec)
```



In [177]:
```
# Run for the second CNF:
n_vec,p_vec = f_MC_n(Delta2,w2,order=4)
f_Plot(n_vec,p_vec)
```



In [ ]: