CS 486 A2 Bayesian Network and Decision Networks

Jason Sun (#20387090)

June 15, 2015

Bayes Net

Construction

OC card owner owns computer or smartphone

Fraud current transaction is fraudlent

Trav card holder is currently travelling

FP current transaction is a foreign purchase

IP current purchase is an internet purchase

CRP a computer related purchase was made within the last week

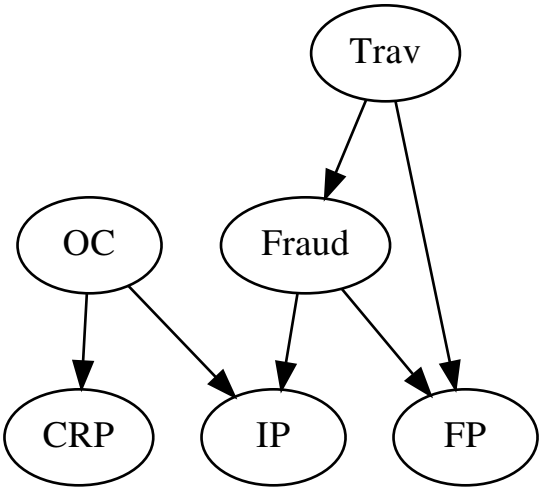

Figure 1: Bayesian Network

Probability interpretation of the financial information given on assignment is as follows.

$$\begin{aligned} P(Fraud|Trav) &= 0.01 \\ P(Fraud|\neg Trav) &= 0.004 \\ P(Trav) &= 0.05 \end{aligned}$$

$$
\begin{aligned}
P(FP|Fraud, \neg Trav) &= 0.10 \\
P(FP|\neg Fraud, \neg Trav) &= 0.01 \\
P(FP|Trav) &= 0.90
\end{aligned}
$$

$$
\begin{aligned}
P(OC) &= 0.75 \\
P(IP|\neg Fraud, OC) &= 0.01 \\
P(IP|Fraud, OC) &= 0.02 \\
P(IP|\neg Fraud, \neg OC) &= 0.001 \\
P(IP|Fraud, \neg OC) &= 0.011 \\
P(CRP|OC) &= 0.10 \\
P(CRP|\neg OC) &= 0.001
\end{aligned}
$$

A CPT can be inferred for each node, from the given assignments.

	$P(OC)$	$P(Trav)$
t	75%	5%
f	$1-75\%$	$1-5\%$

$Fraud$	$Trav$	$P(FP\|Trav)$
t	t	1%
t	f	0.4%
f	t	$1-1\%$
f	f	$1-0.4\%$

FP	$Fraud$	$Trav$	$P(FP\|Fraud, Trav)$
t	t	t	90%
t	t	f	10%
t	f	t	90%
t	f	f	1%
f	t	t	$1-90\%$
f	t	f	$1-10\%$
f	f	t	$1-90\%$
f	f	f	$1-1\%$

IP	OC	$Fraud$	$P(IP\|OC, Fraud)$
t	t	t	2%
t	t	f	1%
t	f	t	1.1%
t	f	f	0.1%
f	t	t	$1-2\%$
f	t	f	$1-1\%$
f	f	t	$1-1.1\%$
f	f	f	$1-0.1\%$

CRP	OC	$P(CRP\|OC)$
t	t	10%
t	f	0.1%
f	t	$1-10\%$
f	f	$1-0.1\%$

Table 1: Conditional Probability Tables

Each CPT can be converted into a `Factor`.

CPT Node	Code
OC	`OC = Factor.new(["OC"], [ 0.75, 1 - 0.75])`
Fraud	`Fraud = Factor.new(["Fraud", "Trav"], [ 0.01, 0.004, 1 - 0.01, 1 - 0.004])`
Trav	`Trav = Factor.new(["Trav"], [ 0.05, 1 - 0.05])`
FP	`FP = Factor.new(["FP", "Fraud", "Trav"], [ 0.90, 0.10, 0.90, 0.01, 1 - 0.90, 1 - 0.10, 1 - 0.90, 1 - 0.01])`
IP	`IP = Factor.new(["IP", "OC", "Fraud"], [ 0.02, 0.01, 0.011, 0.001, 1 - 0.02, 1 - 0.01, 1 - 0.011, 1 - 0.001])`
CRP	`CRP = Factor.new(["CRP", "OC"], [ 0.10, 0.001, 1 - 0.10, 1 - 0.001])`

In general, we can remove any leaf node that is not a query variable or an evidence variable. After its removal, there may be some more leaf nodes, and these too may be irrelevant. Continuing this process, we eventually find that *every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query.*

Note to teaching assistant markers

Providing a prinout of factors computed at each step of the variable elimination results in very long file. I attach a file called `detailed.out` to avoid polluting the pdf. Additionally, you can turn on the flag `DETAILED = true` in `factor.rb` to generate this when testing with `ruby fraud.rb`.

Question 2b Prior Probability

Before we search for previous computer related purchases and before we verify whether it is a foreign and/or internet purchase.

```
====== Question 2b prior: Jon Snow's Credit Company
Pr(Fraud) = {[1]=>0.004300000000000002, [0]=>0.9957}
```

$$P(Fraud) \approxeq 0.43\%$$

Question 2b Posterior Probability

After we verified it is a foreign, but not internet purchase, and card hoder purchased computer related accessories in past week.

```
====== Question 2b posterior: Nerd makes foreign purchase offline
Pr(Fraud | FP, ~IP, CRP) = {[1]=>0.014983808629447266, [0]=>0.9850161913705529}
```

$$P(Fraud|FP, \neg IP, CRP) \approxeq 1.498\%$$

Question 2c Business Trip Knowledge

After we confirm spouse is out of town, changes based on this new piece of information.

```
====== Question 2c: Jon Snow Co. verfies his client is tripping out
Pr(Fraud | FP, ~IP, CRP, Trav) = {[1]=>0.009899992918724067, [0]=>0.9901000070812759}

Changes in fraud probability (rounded 8 decimals):
before 0.01498381 - after 0.00989999 = 0.00508382
After calling we have change of 0.508382%
```

$$P(F|FP, \neg IP, CRP, Trav) - P(F|FP, \neg IP, CRP) \approxeq 0.51\%$$

Question 2d Dishonest Employee

Some dishonest employee wants to steal a credit card, knowing about the Bayes network, and wants to buy over internet.

Prior to making his purchase he should *not travel*, *not buy foreign*, *make a computer related purchase within the last week*, and *don't buy foreign*.

```
====== Question 2d: Jon Snow's traitor

Before infiltration:
Pr(Fraud | IP) = {[1]=>0.009794045938662902, [0]=>0.990205954061337}

After infiltration:
Pr(Fraud | ~Trav, IP, CRP, ~FP) = {[1]=>0.0072597942305469565, [0]=>0.992740205769453}

Changes in fraud probability (rounded 8 decimals):
before 0.00725979 - after 0.00979405 = -0.00253425

After preparing to steal, chance of getting caught -0.253425%
```

The probability of fraud detection reduced (changes) by:

$$P(F|\neg Trav, IP, CRP, \neg FP) - P(F|IP) \approxeq -0.25\%$$

Decision Network

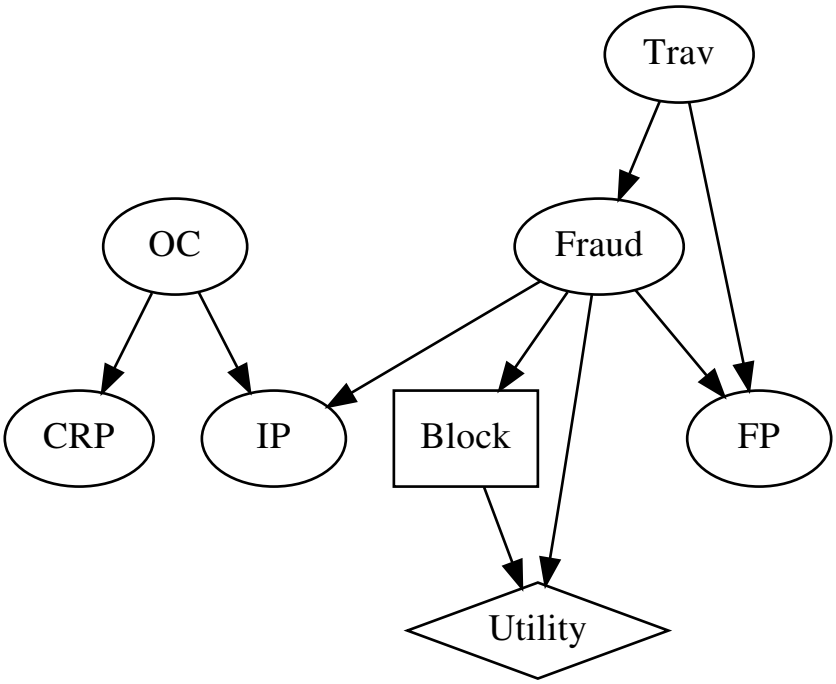

Figure 2: Decision Network

Block	*Fraud*	*Utility*	Comment
t	t	0	No loss from a blocked fraud transaction.
t	f	-10	Blocking a non-fraudulant transaction angers the customer, expected loss of $10
f	t	-1000	A fraudulant transaction that's not blocked means the amount spent is lost, which is -$1000.
f	f	5	Company expects a profit of $5 for transaction of $1000

CPT Node	Code
Utility	`Utility = Factor.new(["Block", "Fraud"], [ 0.0, -10.0, -1000.0, 5.0])`

Question 3b

```
====== Question 3b: Block nerd's $1000 foreign purchase made offline?
Pr(Block | FP, ~IP, CRP) = {[1]=>-9.850161913705529, [0]=>-10.0587276725945}
```

Since 1 is higher ($-9.85 > -10.06$) then we should block.

Question 3c

```
====== Question 3c: Block same transaction knowing he's travelling?
```

```
Pr(Block | FP, ~IP, CRP, Trav) = {[1]=>-9.901000070812758, [0]=>-4.949492883317688}
```

Since 0 is higher ($-4.95 > -9.90$) then we should not block.

Program Output

```
require_relative 'factor'

OC = Factor.new(["OC"],[
    0.75,
    1 - 0.75])
Fraud = Factor.new(["Fraud", "Trav"], [
    0.01,
    0.004,
    1 - 0.01,
    1 - 0.004])
Trav = Factor.new(["Trav"], [
    0.05,
    1 - 0.05])
FP = Factor.new(["FP", "Fraud", "Trav"], [
    0.90,
    0.10,
    0.90,
    0.01,
    1 - 0.90,
    1 - 0.10,
    1 - 0.90,
    1 - 0.01])
IP = Factor.new(["IP", "OC", "Fraud"], [
    0.02,
    0.01,
    0.011,
    0.001,
    1 - 0.02,
    1 - 0.01,
    1 - 0.011,
    1 - 0.001])
CRP = Factor.new(["CRP", "OC"], [
    0.10,
    0.001,
    1 - 0.10,
    1 - 0.001])
Utility = Factor.new(["Block", "Fraud"], [
    0.0,
    -10.0,
    -1000.0,
    5.0])

queryVars = ["Fraud"]
ordering = ["Trav", "FP", "IP", "OC", "CRP"]
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
```

```
# Question 2b Prior
puts "\r\n====== Question 2b prior: Jon Snow's Credit Company"
evidences = {}
result_2b_1 = inference(factors, queryVars, ordering, evidences)
print_summary(result_2b_1, evidences)

# Question 2b Posterior
puts "\r\n====== Question 2b posterior: Nerd makes foreign purchase offline"
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2b_2 = inference(factors, queryVars, ordering, evidences)
print_summary(result_2b_2, evidences)

# Question 2c
puts "\r\n====== Question 2c: Jon Snow Co. verfies his client is tripping out"
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1, "Trav" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2c = inference(factors, queryVars, ordering, evidences)
print_summary(result_2c, evidences)
diff = print_change(result_2b_2, result_2c)
puts "After calling we are #{diff * 100} % more confident"

# Question 2d
puts "\r\n====== Question 2d: Jon Snow's traitor"
evidences = {"IP" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2d_1 = inference(factors, queryVars, ordering, evidences)
puts "Before infiltration:"
print_summary(result_2d_1, evidences)
evidences = {"Trav" => 0, "IP" => 1, "CRP" => 1, "FP" => 0}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2d_2 = inference(factors, queryVars, ordering, evidences)
puts "After infiltration:"
print_summary(result_2d_2, evidences)
diff = print_change(result_2d_2, result_2d_1)
puts "After preparing to steal, chance of getting caught #{diff * 100}%"

# Question 3b
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
fraud = inference(factors, queryVars, ordering, evidences)
result = sumout(multiply(fraud, Utility), "Fraud")
puts "\r\n====== Question 3b: Block nerd's $1000 foreign purchase made offline?"
print_summary(result, evidences)

# Question 3c
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1, "Trav" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
fraud = inference(factors, queryVars, ordering, evidences)
result = sumout(multiply(fraud, Utility), "Fraud")
puts "\r\n====== Question 3c: Block same transaction knowing he's travelling?"
```

```
print_summary(result, evidences)
```

Factor Source Code

```
require 'matrix'
DETAILED = false
# Factors data-structure:
# variable names of the factor are stored in a string, which also indexes
# each possible assignment is used to index into a table of probabilities
# Example:
# Factor.new("CR",[0.8, 0.2,0.2,0.8])
# => #<Factor:
#   @names="CR",
#   @table={[1, 1]=>0.8, [1, 0]=>0.2, [0, 1]=>0.2, [0, 0]=>0.8}>
class Factor
    attr_accessor :names
    attr_accessor :table

    def initialize(varnames, values)
    # values expected to be 2^varnames.size in reverse sorted order (1s to 0s)
        raise MismatchSizeError unless 2 ** varnames.size == values.size
        @names = varnames
        @table = {}
        assignments(Math.log2(values.size)).each_with_index do |a, i|
            @table[a] = values[i]
        end
    end

    def clone
        Factor.new(@names.clone, @table.values.clone)
    end

    private # all methods below this are private
    def assignments(n)
    # n is int, returns 2^n assignments array
        return [[1],[0]] if n == 1
        a = assignments(n-1)
        b1 = a.collect { |e| e + [0] }
        b2 = a.collect { |e| e + [1] }
        (b1 + b2).sort.reverse
    end
end

def restrict(factor, variable, value)
    return factor unless factor.names.include?(variable)
    f = factor.clone
    idx = f.names.index(variable)

    # remove entries not equal to value
    f.table.delete_if { |k, _| k[ idx ] != value }
```

```
    # change key names
    new_table = Hash.new(f.table.size)
    f.table.each do |key, value|
        key.delete_at(idx)
        new_table[key] = value
    end
    f.table = new_table

    # remove variable name
    f.names.slice!(idx)
    return f
end

def multiply(first, second)
    return first.clone if second.nil?
    return second.clone if first.nil?

    def is_desired(k1, mapping, k2)
        # returns true if k1 == k2 given mapping of bits from k1 to k2
        # e.g. k1 = [X, 0, 1] and k2 = [1, 0, X]
        # where k1's index: ["X", "A", "B"] and k2's index: ["B", "A", "D"]
        #                       0,   1,   2                     0,   1,   2
        # mapping would be: {1 => 1, 2 => 0}
        # we want to return true since k1 == k2 given this index
        # more ex:
        # is_desired?([1, 0, 0], [0, 0, 1], {1=>1, 2=>0}) # true
        # is_desired?([0, 0, 0], [0, 0, 1], {1=>1, 2=>0}) # true
        # is_desired?([0, 0, 0], [0, 0, 0], {1=>1, 2=>0}) # true
        # is_desired?([1, 0, 0], [0, 0, 0], {1=>1, 2=>0}) # true
        # is_desired?([1, 1, 0], [0, 0, 0], {1=>1, 2=>0}) # false

        mapping.each do |from, to|
            return false unless k1[from] == k2[to]
        end
        true
    end

    common_names = (first.names & second.names)
    new_names = (first.names | second.names)

    # make mapping of common names from first => second
    mapping = {}
    common_names.each do |name|
        mapping[first.names.index(name)] = second.names.index(name)
    end

    # multiply value1 with value2 if key1 has same entry in key2
    new_values = first.table.collect do |k1, v1|
        second.table.collect do |k2, v2|
            if is_desired(k1, mapping, k2)
                (v1 * v2)
```

```
            end
        end
    end.flatten.compact
    Factor.new(new_names, new_values)
end

def sumout(factor, variable)
    return factor.clone unless factor.names.include?(variable)
    f0 = restrict(factor, variable, 0)
    f1 = restrict(factor, variable, 1)
    name = factor.names.clone
    # merge the two factors
    new_values = f0.table.values.zip(f1.table.values).map {|row| row.inject(:+)}
    name.delete_at(name.index(variable))
    Factor.new(name, new_values)
end

def normalize(factor)
    sum = factor.table.values.inject(:+)
    new_values = factor.table.values.collect {|v| v / sum }
    Factor.new(factor.names, new_values)
end

# inference: computes Pr(queryVars|evidences) by variable elimination
def inference(factors, queryVars, ordering, evidences)
    raise NoCommonNamesError unless factors.is_a?(Array) &&
                                    queryVars.is_a?(Array) &&
                                    ordering.is_a?(Array) &&
                                    evidences.is_a?(Hash)

    # restrict factors w.r.t. evidences
    puts "\t\tInference Step 0 (Unmodified Factors)" if DETAILED
    factors.each { |f| puts f.inspect } if DETAILED
    evidences.each do |var, val|
        factors.each_with_index do |f,i|
            factors[i] = restrict(factors[i], var, val)
        end
    end
    puts "\t\tInference Step 1 (Factors After Restriction)" if DETAILED
    factors.each { |f| puts f.inspect } if DETAILED

    # multiply everything into this factor
    puts "\t\tInference Step 2 (Multiply)" if DETAILED
    prod_factor = nil
    factors.each do |factor|
        prod_factor = multiply(prod_factor, factor)
        puts "Product Factor #{prod_factor.inspect}" if DETAILED
    end

    puts "\t\tInference Step 3 (Sumout)" if DETAILED
    ordering.each do |var|
        prod_factor = sumout(prod_factor, var)
```

```
        puts "Product after sumout #{var}: #{prod_factor.inspect}" if DETAILED
    end

    puts "\t\tInference 4 (Normalized)" if DETAILED
    puts normalize(prod_factor).inspect if DETAILED
    normalize(prod_factor)
end

def print_summary(result, evidences)
    print "Pr(#{result.names.join(", ")}"
    print " | " if evidences.size > 0
    evidences.each_with_index do |e, idx|
        print "~" if e.last == 0
        print "#{e.first}"
        print ", " unless idx == evidences.size - 1
    end
    print ") = #{result.table.inspect}\r\n"
    puts
end

def print_change(f1, f2)
    puts "Changes in fraud probability (rounded 8 decimals):"
    before = f1.table.values.first
    after = f2.table.values.first
    diff = before - after
    puts "before #{before.round(8)} - after #{after.round(8)} = #{diff.round(8)}"
    diff.round(8)
end
```

Fraud Source Code

```
require_relative 'factor'

OC = Factor.new(["OC"],[
    0.75,
    1 - 0.75])
Fraud = Factor.new(["Fraud", "Trav"], [
    0.01,
    0.004,
    1 - 0.01,
    1 - 0.004])
Trav = Factor.new(["Trav"], [
    0.05,
    1 - 0.05])
FP = Factor.new(["FP", "Fraud", "Trav"], [
    0.90,
    0.10,
    0.90,
    0.01,
    1 - 0.90,
```

```
    1 - 0.10,
    1 - 0.90,
    1 - 0.01])
IP = Factor.new(["IP", "OC", "Fraud"], [
    0.02,
    0.01,
    0.011,
    0.001,
    1 - 0.02,
    1 - 0.01,
    1 - 0.011,
    1 - 0.001])
CRP = Factor.new(["CRP", "OC"], [
    0.10,
    0.001,
    1 - 0.10,
    1 - 0.001])
Utility = Factor.new(["Block", "Fraud"], [
    0.0,
    -10.0,
    -1000.0,
    5.0])

queryVars = ["Fraud"]
ordering = ["Trav", "FP", "IP", "OC", "CRP"]
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]

# Question 2b Prior
puts "\r\n====== Question 2b prior: Jon Snow's Credit Company"
evidences = {}
result_2b_1 = inference(factors, queryVars, ordering, evidences)
print_summary(result_2b_1, evidences)

# Question 2b Posterior
puts "\r\n====== Question 2b posterior: Nerd makes foreign purchase offline"
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2b_2 = inference(factors, queryVars, ordering, evidences)
print_summary(result_2b_2, evidences)

# Question 2c
puts "\r\n====== Question 2c: Jon Snow Co. verfies his client is tripping out"
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1, "Trav" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2c = inference(factors, queryVars, ordering, evidences)
print_summary(result_2c, evidences)
diff = print_change(result_2b_2, result_2c)
puts "After calling we are #{diff * 100} % more confident"

# Question 2d
puts "\r\n====== Question 2d: Jon Snow's traitor"
evidences = {"IP" => 1}
```

```
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2d_1 = inference(factors, queryVars, ordering, evidences)
puts "Before infiltration:"
print_summary(result_2d_1, evidences)
evidences = {"Trav" => 0, "IP" => 1, "CRP" => 1, "FP" => 0}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
result_2d_2 = inference(factors, queryVars, ordering, evidences)
puts "After infiltration:"
print_summary(result_2d_2, evidences)
diff = print_change(result_2d_2, result_2d_1)
puts "After preparing to steal, chance of getting caught #{diff * 100}%"

# Question 3b
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
fraud = inference(factors, queryVars, ordering, evidences)
result = sumout(multiply(fraud, Utility), "Fraud")
puts "\r\n====== Question 3b: Block nerd's $1000 foreign purchase made offline?"
print_summary(result, evidences)

# Question 3c
evidences = {"FP" => 1, "IP" => 0, "CRP" => 1, "Trav" => 1}
factors = [Fraud.clone, Trav.clone, FP.clone, IP.clone, CRP.clone, OC.clone]
fraud = inference(factors, queryVars, ordering, evidences)
result = sumout(multiply(fraud, Utility), "Fraud")
puts "\r\n====== Question 3c: Block same transaction knowing he's travelling?"
print_summary(result, evidences)
```