Section F: Bayesian Networks

Dataset Preparation:

```
# load data and rename the data columns
mydata <- read.table('house-votes-84.data.txt', sep = ",")
col_name <- c("class","hadicapped","wpcs", "aotbr","pff","esa"
          ,"rgis","astb","atnc","mm","imigration","scc"
          ,"eduspend","srts","crime","dfe","eaasa")
names(mydata) <- col_name

# stratified sampling data: 75% training and 25% testing
library(sampling)
table(mydata$class)
s <- strata(mydata, 'class', size = c(42,67), method = 'srswor')
training <- mydata[-s$ID_unit,]
testing <- mydata[s$ID_unit,]
```

I. Naive Bayes Models

1).

```
library(e1071)
fmla <- class ~
hadicapped+wpcs+aotbr+pff+esa+rgis+astb+atnc+mm+imigration+scc+eduspend+srts+crime+d
fe+eaasa
naivemodel <- naiveBayes(fmla, data = training)
naivemodel
  physician-fee-freeze
Y                 ?         n         y
  democrat   0.030000000 0.910000000 0.060000000
  republican 0.007936508 0.015873016 0.976190476
```

In this physician-fee-freeze CPT, it is interesting that democrat has a totally different attitude from republican.

2).

```
naivepred <- predict(naivemodel, testing)
table(naivepred, testing$class)
naivepred    democrat republican
  democrat        64         7
  republican       3        35
```

Accuracy = 0.9082569

II. Bayesian Network:

1). Using the library bnlearn to construct Bayesian network.

```
library(gRain)
library(bnlearn)
```

Generate empty or random directed acyclic graphs from a given set of nodes.

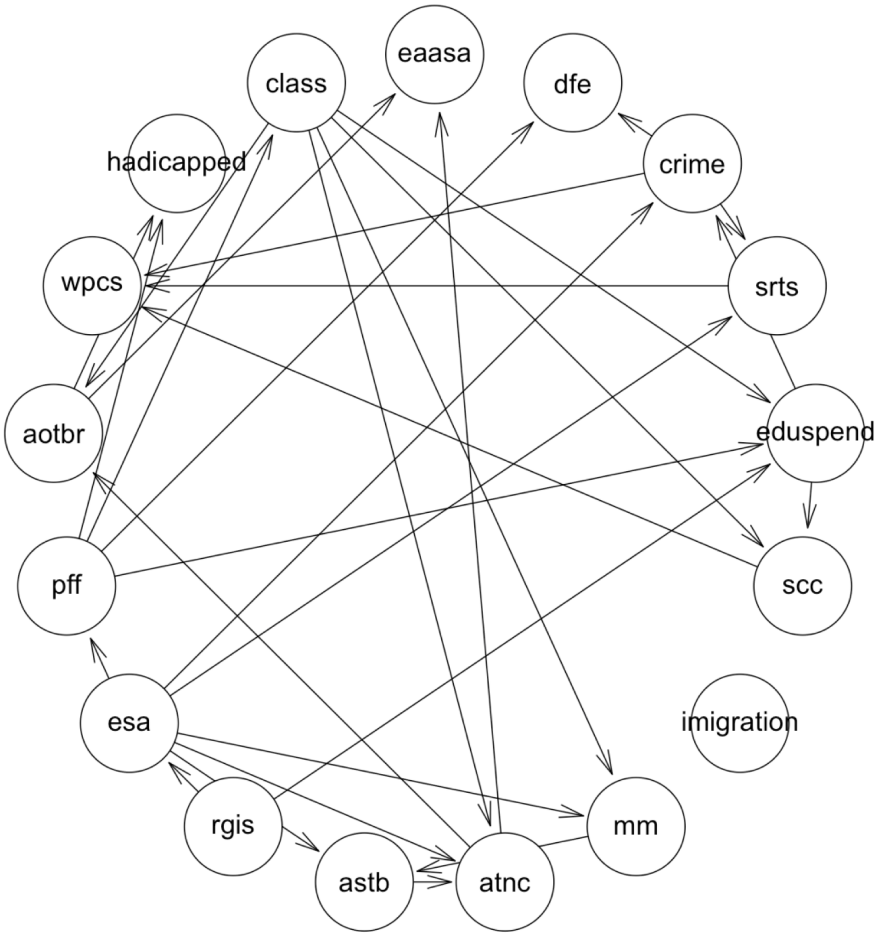
Usage
empty.graph(nodes, num = 1)
random.graph(nodes, num = 1, method = "ordered", ..., debug = FALSE)
Arguments
Nodes: a vector of character strings, the labels of the nodes.
Num: an integer, the number of graphs to be generated.
Method : a character string, the label of a score. Possible values are ordered (full ordering based generation), ic-dag (Ide's and Cozman's Generating Multi-connected DAGs algorithm), melancon (Melancon's and Philippe's Uniform Random Acyclic Digraphs algorithm) and empty (generates empty graphs).

2). Create Bayesian Network
Score-based structure learning algorithms
bnmodel <- tabu(training, score = "k2")
bnmodel
Bayesian network learned via Score-based methods
  model:
 [rgis][imigration][esa|rgis][pff|esa][class|pff][mm|class:esa][eduspend|class:pff:rgis]

[astb|esa:mm][scc|class:eduspend][crime|esa:eduspend][atnc|class:esa:astb][srts|esa:crime]

[dfe|pff:crime][wpcs|scc:srts:crime][aotbr|class:atnc][hadicapped|aotbr:pff][eaasa|aotbr:atnc
]
  nodes: 17
  arcs: 30
    undirected arcs: 0
    directed arcs: 30
  average markov blanket size: 4.59
  average neighbourhood size: 3.53
  average branching factor: 1.76

  learning algorithm: Tabu Search
  score: Cooper & Herskovits' K2
  tests used in the learning procedure: 1376
  optimized: TRUE
plot(bnmodel)



3).

```
library(caret)
pred<-predict(bnmodel,"class", testing)
confusionMatrix(testing$class,pred)
Confusion Matrix and Statistics

          Reference
Prediction   democrat republican
  democrat         64          3
  republican        1         41

               Accuracy : 0.9633
                 95% CI : (0.9087, 0.9899)
    No Information Rate : 0.5963
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9232
 Mcnemar's Test P-Value : 0.6171
```

Sensitivity : 0.9846
Specificity : 0.9318
Pos Pred Value : 0.9552
Neg Pred Value : 0.9762
Prevalence : 0.5963
Detection Rate : 0.5872
Detection Prevalence : 0.6147
Balanced Accuracy : 0.9582

'Positive' Class : democrat

Section G: Observable Markov Models and Hidden Markov Models

I. Exercise 1

```
library(markovchain)

s <- c('a','b','c')
byRow <- TRUE
mcMatrix <- matrix(data = c(0.4,0.3,0.3,0.2,0.6,0.2,0.1,0.1,0.8),byrow = byRow, nrow = 3,
dimnames = list(s,s))
mcTest <- new("markovchain", states = s, byrow = byRow, transitionMatrix = mcMatrix, name =
"mymc")

a = 0;b = 0;c = 0;total=100;
for(i in 1:total){
  char = sample(c('a','b','c'),1, prob = c(0.5, 0.3, 0.2), replace = F)
  if(char == 'a'){
    a = a +1
  }else if(char == 'b'){
    b = b + 1
  }else{
    c = c + 1
  }
  # generate a sequence has 1000 states
  rmarkovchain(n = 1000, object = mcTest, t0 = char)
}
print(a/total);print(b/total);print(c/total)
```

II. Estimation

actual Π = c(0.5, 0.3, 0.2)

estimate Π = c(0.52, 0.28, 0.2)

```
# We can also use the following method to generate a markov chain with 1000 states.
initialState = c(0.5, 0.3, 0.2)
library(HMM)
hmm <- initHMM(c('s1','s2','s3'), c('a','b','c'), c(0.5, 0.3, 0.2), matrix(data =
c(0.4,0.3,0.3,0.2,0.6,0.2,0.1,0.1,0.8),nrow = 3))
```

 # Calculate the A matrix:

```
out <- seq$observation
len <- length(out)
aa=0;ab=0;ac=0;ba=0;bb=0;bc=0;ca=0;cb=0;cc=0;lena=0;lenb=0;lenc=0;
for(i in 1:(len-1)){
  t <- out[i]; t1 <- out[i+1];
  if(t == 'a'){
```

```
   lena = lena + 1
   if(t1 == 'a'){
     aa = aa + 1
   }else if(t1 == 'b'){
     ab = ab + 1
   }else{
     ac = ac + 1
   }
  }else if(t == 'b'){
   lenb = lenb + 1
   if(t1 == 'a'){
     ba = ba + 1
   }else if(t1 == 'b'){
     bb = bb + 1
   }else{
     bc = bc + 1
   }
  }else{
   lenc = lenc + 1
   if(t1 == 'c'){
     ca = ca + 1
   }else if(t1 == 'b'){
     cb = cb + 1
   }else{
     cc = cc + 1
   }
  }
}
matrix(c(aa/lena,ab/lena,ac/lena,ba/lenb,bb/lenb,bc/lenb,ca/lenc,cb/lenc,cc/lenc),nrow=3) A =
matrix(c(aa/len,ab/len,ac/len,ba/len,bb/len,bc/len,ca/len,cb/len,cc/len),nrow=3)
         [,1]      [,2]      [,3]
[1,] 0.3229814 0.3371105 0.3703704
[2,] 0.3913043 0.3456091 0.3209877
[3,] 0.2857143 0.3172805 0.3086420
```

III. Hidden Markov Chain

1). Consider the following sequence of observables: PPPPCCPPPCCCPCCCCCPPPCPCP

```
observations <- c('P','P','P','P','C','C','P','P','P','C','C','C','P','C','C','C','C','C','P','P','P','C','P','C','P')
# Based on the sequence, I have calculated the transition probability matrix:

transProbs <- matrix(c(0.5833333,0.4166667,0.5833333,0.4166667),nrow=2)
hmm = initHMM(c("P","C"), c("P","C"), transProbs=transProbs)
logForwardProbabilities = forward(hmm,observations)
print(exp(logForwardProbabilities))
```

index
states 1 2 3 4 5 6 7 8 9 10
P -1.386294 -2.079442 -2.772589 -3.465736 -4.158883 -4.85203 -5.545177 -6.238325 -6.931472 -7.624619
C -1.386294 -2.079442 -2.772589 -3.465736 -4.158883 -4.85203 -5.545177 -6.238325 -6.931472 -7.624619
index
states 11 12 13 14 15 16 17 18 19 20 21
P -8.317766 -9.010913 -9.704061 -10.39721 -11.09035 -11.7835 -12.47665 -13.1698 -13.86294 -14.55609 -15.24924
C -8.317766 -9.010913 -9.704061 -10.39721 -11.09035 -11.7835 -12.47665 -13.1698 -13.86294 -14.55609 -15.24924
index
states 22 23 24 25
P -15.94239 -16.63553 -17.32868 -18.02183
C -15.94239 -16.63553 -17.32868 -18.02183

The probability is : 2.271025e-10
2). The most likely sequence of hidden states that generated this sequence:
R R B R B R B R R R R B R R R B R B R R B B B A
For every time, I will choose the largest probability of the three states.

3) backword method:
logBackwardProbabilities = backward(hmm,observations)
print(exp(logBackwardProbabilities))
states 1 2 3 4 5 6 7 8
A 6.610869e-10 1.168936e-09 2.692346e-09 3.196795e-09 9.828811e-09 4.284252e-08 7.464368e-08 1.771660e-07
B 1.060257e-09 2.188288e-09 3.559745e-09 9.671368e-09 3.958630e-08 6.815535e-08 1.423638e-07 2.247160e-07
R 4.830322e-10 8.476577e-10 1.984193e-09 2.309491e-09 7.005132e-09 3.131494e-08 5.407408e-08 1.307712e-07
index
states 9 10 11 12 13 14 15 16
A 1.910486e-07 6.730404e-07 2.021062e-06 9.283651e-06 1.027490e-05 3.490442e-05 1.166625e-04 0.0004081061
B 6.458821e-07 1.998752e-06 8.538251e-06 1.190552e-05 3.371536e-05 1.130238e-04 3.921154e-04 0.0012328861
R 1.374090e-07 4.866048e-07 1.437824e-06 6.849733e-06 7.399446e-06 2.515014e-05 8.392835e-05 0.0002948495
index
states 17 18 19 20 21 22 23 24 25
A 0.0012524783 0.005481979 0.009279696 0.02332593 0.02050704 0.095014 0.0957 0.37 1
B 0.0050632677 0.008585062 0.018350623 0.02729288 0.08731724 0.117182 0.3497 0.86 1

R 0.0008925287 0.004009754 0.006709086 0.01726661 0.01458452 0.070204 0.0686 0.27 1
[f(A,10) b(A,10)] / p(observations) = (1.929852e-04*6.730404e-07)/ 2.271025e-10 = 0.5719305
[f(B,10) b(B,10)] / p(observations) = (3.417891e-05*1.998752e-06)/ 2.271025e-10 = 0.300812
[f(R,10) b(R,10)] / p(observations) = (3.605734e-04*4.866048e-07)/ 2.271025e-10 = 0.7725884

R has the largest probability, then R is the most likely hidden state that generated the "C" in position 10 of the sequence.

4). learn the transition probabilities and the emission probabilities
Use the HMM to generate a sequence of observables of length 2000.

```
hmm <- initHMM(States=states, Symbols=emis, startProbs=initProbs, transProbs=transProbs,
emissionProbs=emissionProbs)
simHMM <- simHMM(hmm, 2000)
```

For an initial Hidden Markov Model (HMM) and a given sequence of observations, the Baum-Welch algorithm infers optimal parameters to the HMM. Since the Baum-Welch algorithm is a variant of the Expectation-Maximisation algorithm, the algorithm converges to a local solution which might not be the global optimum.

```
bw <- baumWelch(hmm,simHMM$observation,100)

$startProbs
  A   B   R
0.6 0.3 0.1

$transProbs
    to
from        A         B         R
   A 0.2302310 0.6427725 0.1269966
   B 0.2986122 0.3057461 0.3956417
   R 0.1789913 0.6197112 0.2012975

$emissionProbs
      symbols
states        P         C
     A 0.3577717 0.6422283
     B 0.8563439 0.1436561
     R 0.3122474 0.6877526
```