2016 BIML Tutorial

M1. Bioinformatics & Machine Learning

<Naïve Bayes Classification, HMM>

아주대학교

손경아

Content

- R for bioinformatics
- Bioconductor 소개
- Machine Learning Process
- Naïve Bayes Classification
  - SRBCT Data set
  - Naïve Bayes Classifier
  - Cross Validation
  - Evaluation
- Hidden Markov Model
  - Markov 모델을 이용한 DNA sequence generation
  - Hidden Markov 모델을 이용한 DNA sequence generation
  - Viterbi Algorithm 구현

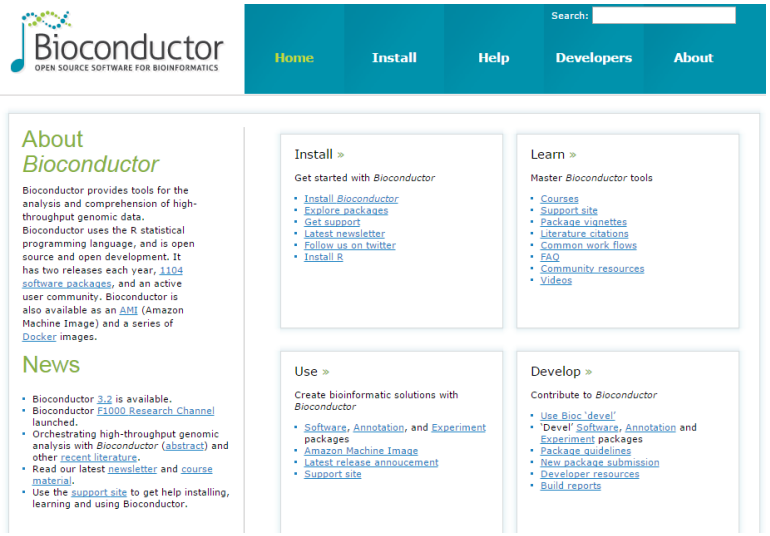
R for Bioinformatics

- Requirements
  - R - https://www.r-project.org/
  - R Studio - https://www.rstudio.com/
  - R을 먼저 설치하고, R Studio를 설치
- R for Bioinformatics
  - 스크립트 언어 기반에 쉽게 이해 가능
  - 다양한 데이터 타입 읽기/쓰기/변환 쉽게 가능
  - 통계적 기법을 적용할 수 있는 다양한 통계적 도구 및 환경 제공
  - 다양한 의학 데이터 라이브러리를 제공하고, 머신러닝 라이브러리 제공
  - 강력한 시각화 라이브러리를 제공

Bioconductor

- Bioconductor
  - 고속 대량 유전체 데이터를 분석하고 이해하기 위한 R 패키지들을 포함하는 오픈소스 프로젝트
  - Microarray에 대한 전처리, 실험 설계, 생명정보학, 데이터, 알고리즘에 대한 다양한 라이브러리 제공
  - http://www.bioconductor.org/packages/release/BiocViews.html#___ExperimentData
- 설치

```
### 라이브러리 설치
source("https://bioconductor.org/biocLite.R")
biocLite()

### 특정 라이브러리 설치
source("https://bioconductor.org/biocLite.R")
biocLite("ABAData")

### 라이브러리 로드
library(ABAData)
```



Machine Learning Process

- Process

- Data Understanding
  - 데이터 수집
  - 데이터의 질적 문제를 정의
- Data Preparation
  - 가공되지 않은 데이터를 모델링에 적합한 데이터 셋으로 바꾸기 위해 필요한 모든 활동
- Modeling
  - 머신러닝 모델링 기술을 선택하고 적용
  - 가장 최적화된 값을 찾기 위해 파라미터를 조정
- Evaluation
  - 만든 모델을 평가하는 단계

Naïve Bayes Classification

- SRBCT Data set
  - SRBCT(Small Round Blue Cell Tumors)는 아이들에게 흔한 악성 종양으로서 약 80%가 20대 이하에서 발병 (Khan et al. 2001)
  - 종양은 4가지 종류
    - NB(neuroblasoma), RMS(rhabdomyosarcoma), BL(Burkitt's lymphoma), EWS(Ewing's family of tumors)
    - 현미경으로 정확히 분류를 할 수가 없음
    - Classification 에서 클래스로서 사용
  - CRBCT cDNA Microarray 데이터를 이용해서 분류
    - 총 2308개의 유전자 데이터
    - Classifier 학습하는데 있어서 Feature 로서 사용
  - Sample 데이터
    - 총 88개 (EWS : 29개, BL : 11개, NB : 18개, RMS : 25개, Non-SRBCT : 5개)
    - 88 x 2308 Matrix

Naïve Bayes Classification

- Load SRBCT Data set
  - 두 가지 라이브러리를 통해서 데이터 로드 가능
    - Bioconductor에 made4 라이브러리 사용
      - 기존의 데이터에서 가장 영향 있는 유전자 데이터 306개를 사용 (Non-SRBCT 포함)
      - Train (64 x 306), Test (25 x 306) 데이터 구분
    - R 레파지토리에 plsgenomics 라이브러리 사용
      - Non-SRBCT 데이터를 제외한 모든 샘플과 유전자 데이터를 사용 (83 x 2308)
      - EWS : 1 , BL :2 , NB : 3 , RMS :4 으로 인코딩

Bioconductor – made4

```
### 라이브러리 설치
source("https://bioconductor.org/biocLite.R")
biocLite("made4")

### 라이브러리 로드
library(made4)

### 데이터 불러오기
data(khan)
summary(khan)
khan$train
khan$test
khan$train.classes
khan$test.classes
```

R Repository – plsgenomics

```
### 라이브러리 설치
install.packages('plsgenomics')

### 라이브러리 로드
library(plsgenomics)

### 데이터 불러오기
data(SRBCT)
dim(SRBCT$X)
sum(SRBCT$Y==1)
sum(SRBCT$Y==2)
sum(SRBCT$Y==3)
sum(SRBCT$Y==4)
```

Naïve Bayes Classification

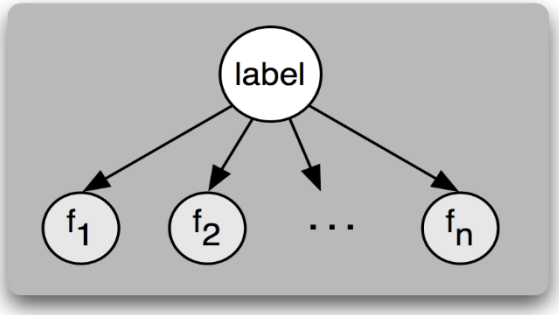
- Naïve Bayes Classification
  - Naïve Bayes Classifier 을 사용하기 위해서 e1071 라이브러리를 사용
    - SVM, Naïve Bayes 등 다양한 머신러닝 알고리즘 제공

```
#machine learning library 설치
install.packages("e1071")

# e1071 라이브러리 불러오기
library(e1071)
```

  - Naïve Bayes Classifier
    - 간단한 Bayesian Network 로서 볼 수 있음
    - 모든 feature 를 독립(independent)이라고 가정

$$P(C_i \mid x_1, x_2, \ldots, x_d) = \frac{P(x_1, x_2, \ldots, x_d \mid C_i) P(C_i)}{P(x)}$$

$$= \frac{\left(P(x_1 \mid C_i) \cdot P(x_2 \mid C_i) \cdot \ldots \cdot P(x_n \mid C_i)\right) P(C_i)}{P(x)}$$



Naïve Bayes Classification

- Naïve Bayes Classifier 사용법
  - Make Model

naiveBayes(x, y, laplace = 0, ...)
- x : A numeric matrix, or a data frame of categorical and/or numeric variables
- y : Class vector
- laplace : Positive double controlling Laplace smoothing

naiveBayes(formula, data, laplace = 0, ...)
- formual : A formula of the form class ~ x1 + x2 + ... Interactions are not allowed
- data : Either a data frame of predictors (categorical and/or numeric) or a contingency table.

Return : 학습된 Naïve Bayes 모델

  - Predict test data

predict(object, newdata, ...)
- object : 학습된 모델
- newdata : test data set

Return : test data 에 대한 예측 결과

Naïve Bayes Classification

- Bioconductor – made4 데이터 사용

```
### 필요한 라이브러리 불러오기
library(made4)
library(e1071)

### 데이터 로드
data(khan)

### Training, Test data 생성
train_data <- khan$train
train_class <- khan$train.classes
test_data <- khan$test
test_class <- khan$test.classes

### 데이터 전처리
#### made4에서 주는 데이터는 행/열을 바꿔야함
train_data <- t(train_data)
test_data <- t(test_data)

### Naive Bayes Classifier model 구축
naive_bayes_model <- naiveBayes(x=train_data, y=train_class)

### test 데이터 predict
test_result_nb <- predict(naive_bayes_model, test_data)

### predict class 비교
table(test_result_nb, test_class)
```

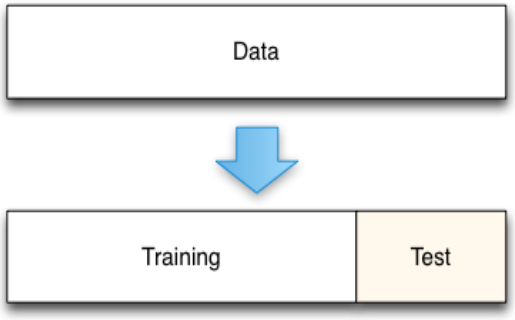
Naïve Bayes Classification

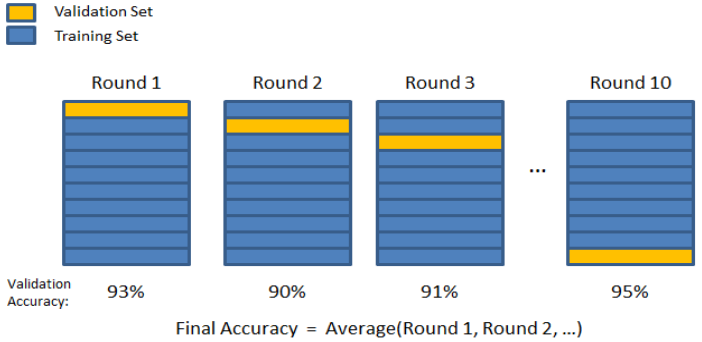
- plsgenomics 데이터 사용

```
### 필요한 라이브러리 불러오기
library(plsgenomics)
library(e1071)

### 데이터 로드
data(SRBCT)

### Training, Test data 생성
train_row <- sample(1:83,60)
train_data <- SRBCT$X[train_row,]
test_data <- SRBCT$X[-train_row,]
train_class <- factor(SRBCT$Y[train_row],levels = c("1","2","3","4"))
test_class <- factor(SRBCT$Y[-train_row],levels = c("1","2","3","4"))

### Naive Bayes Classifier model 구축
naive_bayes_model <- naiveBayes(x=train_data, y=train_class)

### test 데이터 predict
test_result_nb <- predict(naive_bayes_model, test_data)

### predict class 비교
table(test_result_nb, test_class)
```

Naïve Bayes Classification

- Cross Validation
  - 훈련데이터를 사용해서 모델 성능을 평가하고 최적의 모델을 선택
  - Hold Out Method
    - 데이터 셋을 랜덤 하게 2가지 부분 셋으로 나누는 방법
    - Training Set
    - Test Set
  - K-fold Cross validation
    - 데이터 셋을 K개의 부분 셋으로 나누는 방법
    - Hold Out Method 를 K번 반복
    - K개의 부분 셋 중 하나를 Test set으로 사용





Naïve Bayes Classification

- 5-fold Cross validation

```
## Cross Validation
k=5
index <- sample(1:k,nrow(SRBCT$X),replace=TRUE)
folds <- 1:k
myRes=data.frame()

for (i in 1:k) {
   # create training set
   training <- subset(SRBCT$X, index %in% folds[-i])
   # create training set label
   training_class <- factor(subset(SRBCT$Y, index %in% folds[-i]),levels = c("1","2","3","4"))
   # create test set
   test <- subset(SRBCT$X, index %in% c(i))
   # create test set label
   test_class <- factor(subset(SRBCT$Y, index %in% c(i)),levels = c("1","2","3","4"))
   # train model
   naive_bayes_model <- naiveBayes(x=training, y=training_class)
   # run model on test set
   temp <- data.frame(predict(naive_bayes_model, test))
   colnames(temp)="Predicted"
   # create data.frame for results
   results <- data.frame(Predicted=temp, Actual=test_class)
   # append results for each iteration
   myRes <- rbind(myRes, results)
}

table(myRes)
```

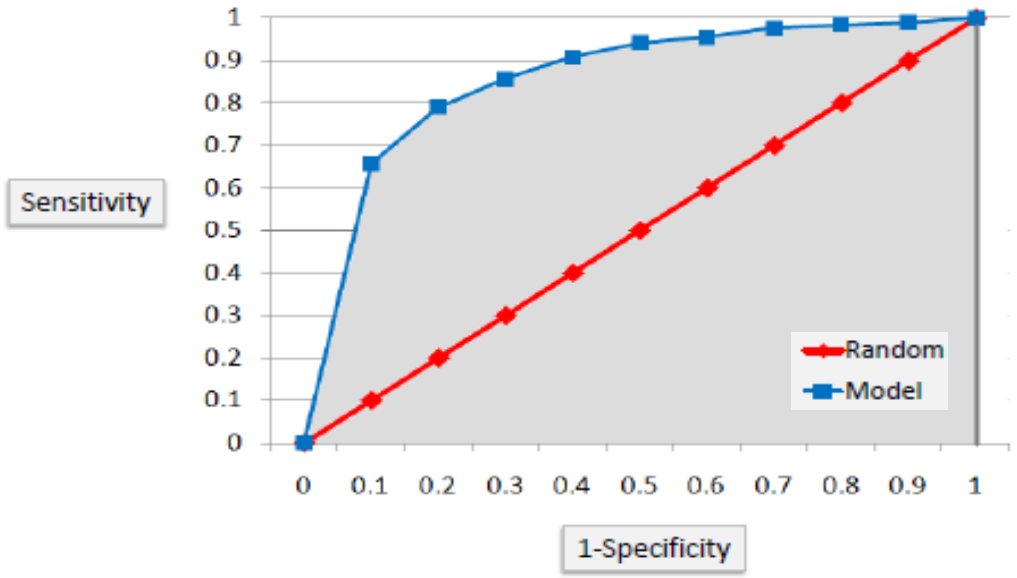
Naïve Bayes Classification

- Evaluation
  - Confusion Matrix
    - Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
    - Recall (Sensitivity)= $\frac{TP}{TP+FN}$
    - Precision = $\frac{TP}{TP+FP}$

Predicted Value (predicted by the test)		Actual Value (as confirmed by experiment)	
		positives	negatives
	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative

Naïve Bayes Classification

- Evaluation
  - ROC(Receiver Operating Curve) Chart
    - Specificity = $\frac{TN}{FP+TN}$
    - Sensitivity : 1인 케이스에 대해 1로 예측한 비율
    - Specificity : 0인 케이스에 대해 1로 잘못 예측한 비율
    - AUC (Area Under the Curve)
      - measure of quality of the classification models



Naïve Bayes Classification

- Evaluation
  - 평가를 쉽게 하기 위해서 caret, pROC 라이브러리를 사용

```
# 평가를 위한 caret, pROC 라이브러리 설치
install.packages( " caret " )
install.packages( " pROC " )

# 라이브러리 사용
library(caret)
library(pROC)
```

  - Confusion Matrix
    - caret 라이브러리에 confusionMatrix 함수 사용
  - ROC(Receiver Operating Curve) Chart
    - pROC 라이브러리에 plot.roc 함수 사용
  - AUC (Area Under the Curve)
    - pROC 라이브러리에 multiclass.roc 함수 사용

Naïve Bayes Classification

- Confusion Matrix

```
### Confusion matrix
confusionMatrix(myRes$Predicted, myRes$Actual)
```

- ROC curve

```
### ROC curve
roc_nb <- plot.roc(as.numeric(myRes$Predicted), as.numeric(myRes$Actual))
lines(roc_nb, col="black")
legend("bottomright", c("Naive Bayes"), fill = c("black"))
```

- AUC

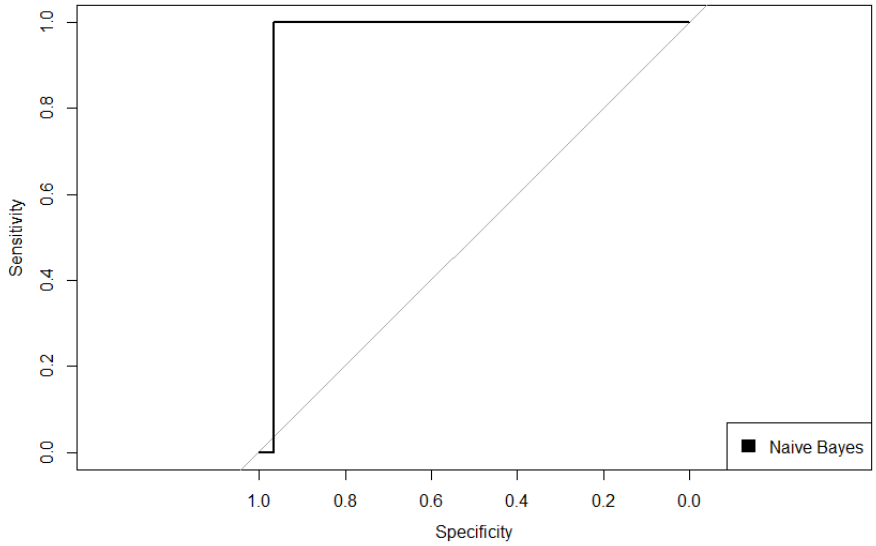
```
### AUC
multiclass.roc(as.numeric(myRes$Predicted), as.numeric(myRes$Actual), percent=TRUE)

Call:
multiclass.roc.default(response = as.numeric(myRes$Predicted),     predictor =
as.numeric(myRes$Actual), percent = TRUE)

Data: as.numeric(myRes$Actual) with 4 levels of as.numeric(myRes$Predicted): 1, 2, 3, 4.
Multi-class area under the curve: 98.53%
```

Naïve Bayes Classification

Confusion Matrix

```
> confusionMatrix(myRes$Predicted,myRes$Actual)
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4
         1 29  0  1  0
         2  0 11  0  0
         3  0  0 15  0
         4  0  0  2 25

Overall Statistics

               Accuracy : 0.9639
                 95% CI : (0.898, 0.9925)
    No Information Rate : 0.3494
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9497
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            1.0000   1.0000   0.8333   1.0000
Specificity            0.9815   1.0000   1.0000   0.9655
Pos Pred Value         0.9667   1.0000   1.0000   0.9259
Neg Pred Value         1.0000   1.0000   0.9559   1.0000
Prevalence             0.3494   0.1325   0.2169   0.3012
Detection Rate         0.3494   0.1325   0.1807   0.3012
Detection Prevalence   0.3614   0.1325   0.1807   0.3253
Balanced Accuracy      0.9907   1.0000   0.9167   0.9828
```

ROC curve

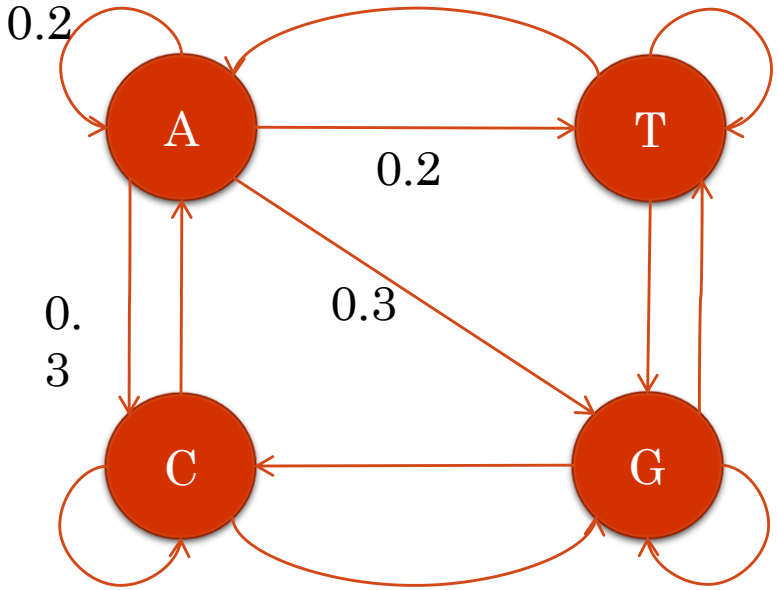


Hidden Markov Model (목차)

- Markov 모델을 이용한 DNA sequence generation
- Hidden Markov 모델을 이용한 DNA sequence generation
- Viterbi Algorithm 구현

Markov 모델을 이용한 DNA sequence generation

- 샘플 DNA sequence 데이터를 생성하려 한다고 가정하자



$$TransitionMatrix = \begin{pmatrix} 0.2 & 0.3 & 0.3 & 0.2 \\ 0.1 & 0.41 & 0.39 & 0.1 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.5 & 0.17 & 0.17 & 0.17 \end{pmatrix}$$

$e_{i,j}$ = state i 에서 state j 로 갈 확률

*위의 그림에서 C,T,G state에서의 확률은 생략됨

Markov 모델을 이용한 DNA sequence generation

- Transition Matrix 구현
  - 확률 matrix를 row-by-row로 정의

```
nucleotides <- c("A", "C", "G", "T")
# Set the values of the probabilities, where the previous
afterAprobs <- c(0.2, 0.3, 0.3, 0.2)
# Set the values of the probabilities, where the previous
afterCprobs <- c(0.1, 0.41, 0.39, 0.1)
# Set the values of the probabilities, where the previous
afterGprobs <- c(0.25, 0.25, 0.25, 0.25)
# Set the values of the probabilities, where the previous
afterTprobs <- c(0.5, 0.17, 0.17, 0.17)
```

  - 각각의 확률을 하나의 matrix로 병합

```
mergedprobs = c(afterAprobs,afterCprobs,afterGprobs,afterTprobs)
mytransitionmatrix <- matrix(mergedprobs, 4, 4, byrow=TRUE)
rownames(mytransitionmatrix) <- nucleotides
colnames(mytransitionmatrix) <- nucleotides
```

Markov 모델을 이용한 DNA sequence generation

- "sample" function in R
  - 각각의 state에서 다음 state로의 이동을 확률적으로 구하기 위해 사용한다
  - example

```
26  nucleotides = c("A","C","G","T")
27  next_nucleotide = sample(nucleotides, 1, prob=transition_matrix['A',])
```

- DNA sequence 생성
  - 첫번째 nucleotide가 A라고 가정하고 길이 3짜리 DNA sequence를 생성한다

```
26  nucleotides = c("A","C","G","T")
27  DNAseq = "A"
28  next_nucleotide = sample(nucleotides, 1, prob=transition_matrix['A',])
29  DNAseq = c(DNAseq,next_nucleotide)
30
31  prev_nucleotide = next_nucleotide
32
33  next_nucleotide = sample(nucleotides, 1, prob=transition_matrix[prev_nucleotide,])
34  DNAseq = c(DNAseq,next_nucleotide)
35
36  DNAseq
```

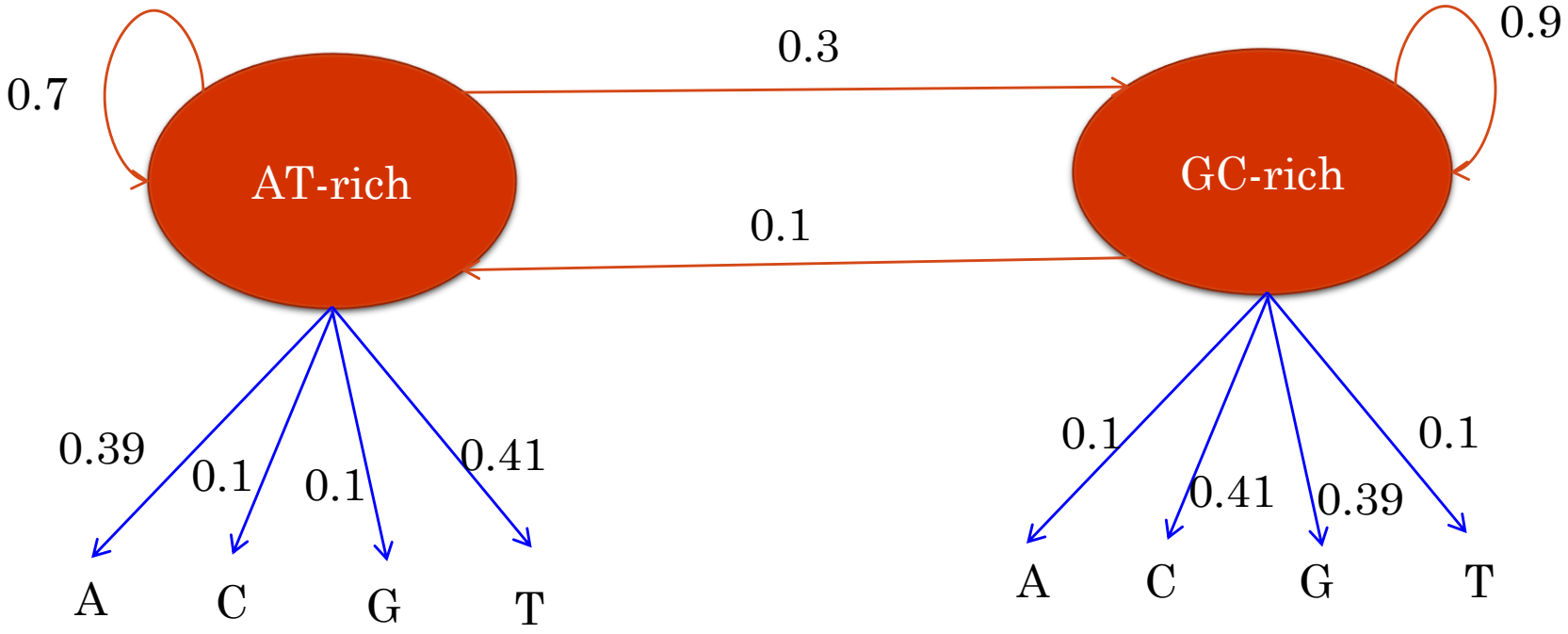
Markov 모델을 이용한 DNA sequence generation

- Function Definition

```
26  generatemarkovseq <- function(transitionmatrix, initial_nucleotide, seqlength)
27  {
28    nucleotides <- c("A", "C", "G", "T")
29    mysequence <- character()
30    firstnucleotide = initial_nucleotide
31
32    mysequence[1] <- firstnucleotide
33    for (i in 2:seqlength)
34    {
35      prevnucleotide <- mysequence[i-1]
36      probabilities <- transitionmatrix[prevnucleotide,]
37      nucleotide <- sample(nucleotides, 1, prob=probabilities)
38      mysequence[i] <- nucleotide
39    }
40    return(mysequence)
41  }
```

Hidden Markov 모델을 이용한 DNA sequence generation

- AT-rich와 CG-rich, 두개의 hidden state가 있다고 가정한다
- 주어진 Transition matrix와 emission matrix로 DNA sequence를 생성



Hidden Markov 모델을 이용한 DNA sequence generation

- Building transition and emission matrix

$$TransitionMatrix = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}$$

$$EmissionMatrix = \begin{pmatrix} 0.39 & 0.1 & 0.1 & 0.41 \\ 0.1 & 0.41 & 0.39 & 0.1 \end{pmatrix}$$

```
48  states <- c("AT-rich", "GC-rich")
49  ATrichprobs <- c(0.7, 0.3)
50  GCrichprobs <- c(0.1, 0.9)
51  thetransitionmatrix <- matrix(c(ATrichprobs, GCrichprobs), 2, 2, byrow = TRUE)
52  rownames(thetransitionmatrix) <- states
53  colnames(thetransitionmatrix) <- states
54  thetransitionmatrix
55
56  nucleotides <- c("A", "C", "G", "T")
57  ATrichstateprobs <- c(0.39, 0.1, 0.1, 0.41)
58  GCrichstateprobs <- c(0.1, 0.41, 0.39, 0.1)
59  theemissionmatrix <- matrix(c(ATrichstateprobs, GCrichstateprobs), 2, 4, byrow = TRUE)
60  rownames(theemissionmatrix) <- states
61  colnames(theemissionmatrix) <- nucleotides
62  theemissionmatrix
```

Hidden Markov 모델을 이용한 DNA sequence generation

- 처음 state가 AT-rich라고 가정하고 길이 3짜리 DNA sequence를 생성해보자

```
65  nucleotides = c("A","C","G","T")
66  states = c("AT-rich","GC-rich")
67  first_state = 'AT-rich'
68
69  probabilities = theemissionmatrix[first_state,]
70  seq = sample(nucleotides,1,prob=probabilities)
71
72  prev_state = first_state
73  next_state = sample(states,1,prob=thetransitionmatrix[prev_state,])
74  probabilities = theemissionmatrix[next_state,]
75  new_nucleotide = sample(nucleotides,1,prob=probabilities)
76  seq = c(seq,new_nucleotide)
77
78  prev_state = next_state
79  next_state = sample(states,1,prob=thetransitionmatrix[prev_state,])
80  probabilities = theemissionmatrix[next_state,]
81  new_nucleotide = sample(nucleotides,1,prob=probabilities)
82  seq = c(seq,new_nucleotide)
```

Hidden Markov 모델을 이용한 DNA sequence generation

- Function Definition

```
86  generatehmmseq <- function(transitionmatrix, emissionmatrix, initial_state, seqlength)
87  {
88    nucleotides = c("A", "C", "G", "T")
89    states = c("AT-rich", "GC-rich")
90    mysequence = character()
91    mystates = character()
92
93    firststate = initial_state
94    probabilities = emissionmatrix[firststate,]
95    firstnucleotide = sample(nucleotides, 1, prob=probabilities)
96    mysequence[1] = firstnucleotide
97    mystates[1] = firststate
```

Hidden Markov 모델을 이용한 DNA sequence generation

- Function Definition

```
  for (i in 2:seqlength)
  {
    prevstate    <- mystates[i-1]
    stateprobs   <- transitionmatrix[prevstate,]
    state        <- sample(states, 1, prob=stateprobs)
    probabilities <- emissionmatrix[state,]
    nucleotide   <- sample(nucleotides, 1, prob=probabilities)
    mysequence[i] <- nucleotide
    mystates[i]  <- state
  }

  for (i in 1:length(mysequence))
  {
    nucleotide   <- mysequence[i]
    state        <- mystates[i]
    print(paste("Position", i, ", State", state, ", Nucleotide = ", nucleotide))
  }
}
```

HMM: Viterbi Algorithm 구현

- Input : DNA sequence, transition matrix, and emission matrix
- Goal : to find the most probable sequence of hidden states
- Example : what are the most probable sequences of hidden states?
  - 1. ATATTTATATAAATAATT
  - 2. CGGCCGGCGCGCGCGCG
  - 3. AAGCGTGGTGCTACGGC

HMM: Viterbi Algorithm 구현

- 아래와 같이 transition matrix 와 emission matrix 주어지고, DNA sequence “AG” 의 most probable hidden state sequence 를 찾아보자

$$TransitionMatrix = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}$$

$$EmissionMatrix = \begin{pmatrix} 0.39 & 0.1 & 0.1 & 0.41 \\ 0.1 & 0.41 & 0.39 & 0.1 \end{pmatrix}$$

- 처음 state를 AC-rich라 가정하고 시작한다.
- 주어진 DNA sequence 의 처음 nucleotide 는 A이므로, (AR-rich, A) 에서 다음 nucleotide G가 나오기 위한

HMM: Viterbi Algorithm 구현

- Function Definition

```
161  makeViterbimat <- function(sequence, transitionmatrix, emissionmatrix)
162  {
163    v <- matrix(NA, nrow = length(sequence), ncol = dim(transitionmatrix)[1])
164    colnames(v) = c('AT-rich','GC-rich')
165
166    v[1,'AT-rich'] = 1
167    v[1,'GC-rich'] = 0
168
169    for (i in 2:length(sequence)) # For each position in the DNA sequence:
170    {
171      statelprobnucleotidei <- emissionmatrix[1,sequence[i]]
172      v[i,1] <-  statelprobnucleotidei * max(v[(i-1),] * transitionmatrix[,1])
173
174      statelprobnucleotidei = emissionmatrix[2,sequence[i]]
175      v[i,2] <-  statelprobnucleotidei * max(v[(i-1),] * transitionmatrix[,2])
176
177    }
178    return(v)
179  }
```

HMM: Viterbi Algorithm 구현

- Function Definition

```
123  viterbi <- function(sequence, transitionmatrix, emissionmatrix)
124  {
125    # Get the names of the states in the HMM:
126    states <- rownames(theemissionmatrix)
127
128    # Make the Viterbi matrix v:
129    v <- makeViterbimat(sequence, transitionmatrix, emissionmatrix)
130
131    for( i in 1:(dim(v))[1]){
132      print(v[i,])
133    }
134  }
```