“데이터 사이언티스트를 위한 DMR-3”
“K Nearest Neighbor, Neural Network, Naive Bayesian, SVM”

Jinseog Kim
Dongguk University
jinseog.kim@gmail.com

2017-04-11

Supervised learning

1. generalized linear models
    1. linear regression
    2. logistic regression
    3. poisson, gamma, ...
2. non-parametric models
    1. decision tree, knn,
3. neural network(1 hidden layer)
4. penalized regression models
    1. ridge - l_2 penalty
    2. lasso - l_1 penalty
    3. elastic net - $l_1 + l_2$ penalty
5. ensembles
    1. bagging
    2. various boosting
    3. random forest
6. deep learning : h2o

Predictive modeling workflow and R methods(functions)

1. fit model `fit <- knn(trainingData, outcome, k = 5)`
2. check or validation models `print`, `plot`, `summary`
3. predict using selected model & test data `predict(fit, newdata)`
4. performance evaluation `ROCR`, 'caret' packages

Naive Bayes 모형

- 입력변수의 조건부 분포가 서로 독립 = 단순 베이즈 가정(naive Bayes assumption)

$$
\begin{aligned}
P(Y = k | X_1 = x_1, \ldots, X_p = x_p) &\propto P(X_1 = x_1, \ldots, X_p = x_p | Y = k) P(Y = k) \\
&\propto P(Y = k) \prod_{j=1}^{p} P(X_j = x_j | Y = k)
\end{aligned}
$$

k-근방 분류

- k-NN은 통계적 모형을 적합하지 않는 메모리 기반의 방법
- x점이 주어지면 x와 가장 거리가 가까운 k개의 훈련자료점의 y값들을 비교
- $N(x)$를 x와 거리가 가장 가까운 k개의 훈련자료들의 집합인 k-근방(k-nearest neighborhood)이라 하면

$$\hat{y}(x) = \arg \max_{l \in \{-1,+1\}} \sum_{x_j \in N(x)} I(y_j = l)$$

- k가 증가하면 편의는 커지고 분산은 감소한다.
- 점근적으로 1-NN의 오분류율은 최적 Bayes 오분류율의 2배를 넘지 않는다고 알려져 있다.

신경망 (Neural networks)

- 다층신경망
  1. 입력층(input layer):
     - 입력변수에 대응되는 노드로 구성, 노드의 수는 입력변수의 개수
  2. 은닉층(hidden layer):
     - 입력층으로부터 전달되는 변수값들의 선형결합을 비선형함수로 처리
     - 출력층 또는 다른 은닉층에 전달하는 역할
  3. 출력층(output layer)
     - 출력변수에 대응되는 노드
     - 분류모형에서는 클래스의 수 만큼의 출력노드가 생성

신경망 (Neural networks)

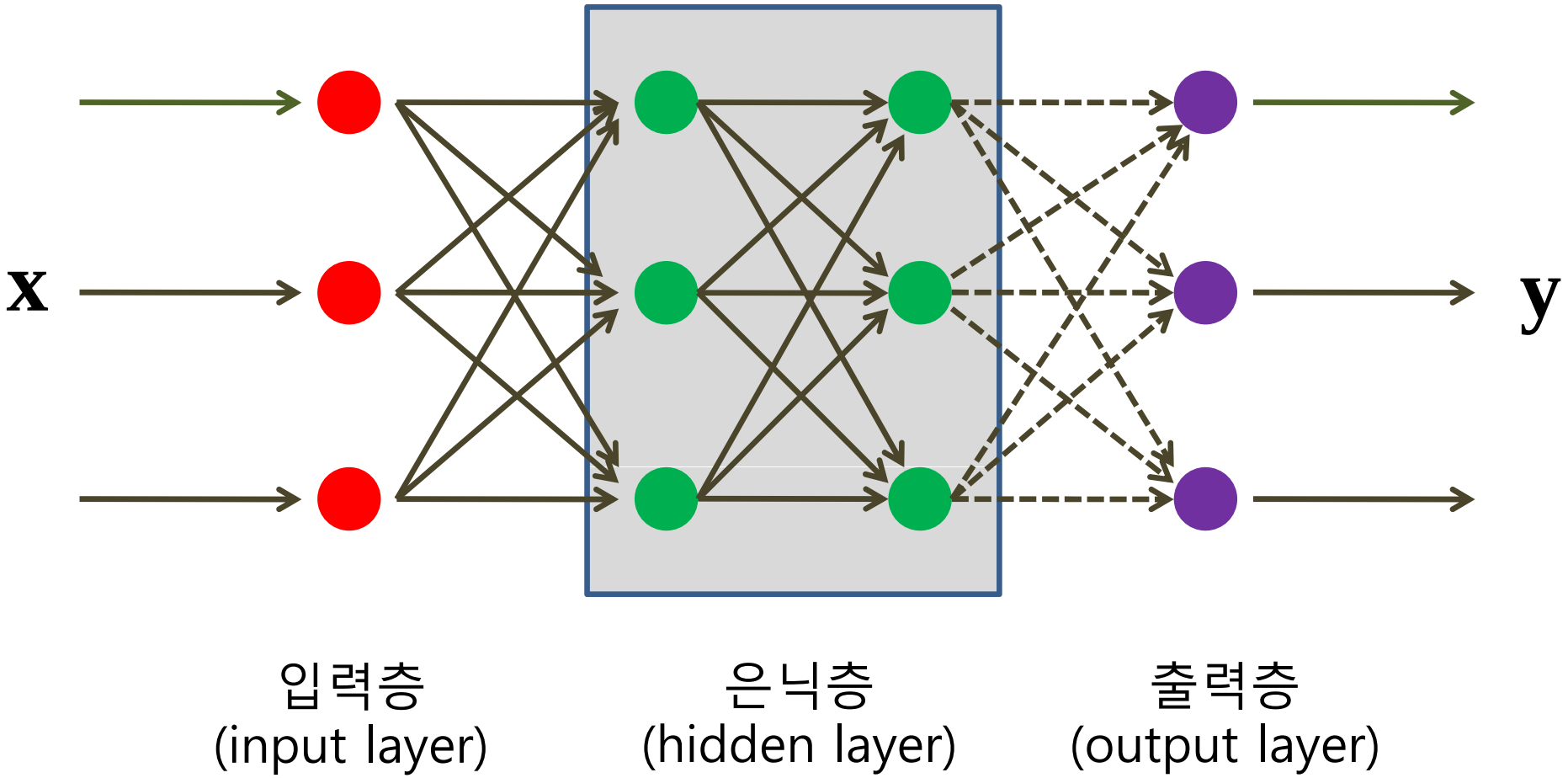


Figure: 다층신경망 모형의 구조

SVM

- Hinge loss: $(1 - y_i f(x_i))_+ \triangleq \max(1 - y_i f(x_i), 0)$
- $\lambda > 0$: tunning parameter
- SVM은 아래의 penalized hinge loss를 최소화하는 방법

$$\min_{f \in \mathcal{H}} \left(\frac{1}{n} \sum_{i=1}^{n} (1 - y_i f(x_i))_+ + \lambda \|f\|^2 \right).$$

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$$

- f는 n개의 훈련자료점에서 계산된 커널의 선형결합

$$\min_{\beta, \beta_0} \left(\frac{1}{n} \sum_{i=1}^{n} (1 - y_i(\beta_0 + \Phi(x_i)^T \beta))_+ + \lambda \|\beta\|^2 \right)$$

R 패키지 및 함수

R패키지 및 함수명	옵션	함수/옵션 설명
e1071::naiveBayes	laplace	Laplace smoothing(default=0)
class::knn	k	근방 관측치 수
nnet::nnet	size	hidden node의 수
	linout	switch for linear output units
	entropy	switch for entropy
	softmax	switch for softmax
	decay	parameter for weight decay(default=0)
	maxit	역전파 알고리즘의 반복 횟수를 지정하며 기본값은 100임
neuralnet::neuralnet	hidden	
	err.fct	error function("sse"|"ce"=cross entroy)
	act.fct	activation function("logistic"|"tanh")
	linear.ouout	switch for linear output units
	algorithm	learning algorithm("backprop"|rprop+|...)
e1071::svm	kernel	"linear"|"polynomial"|"radial basis"
	gamma	parameter for kernel function
	type	"C-classification"|"one-classification"
		"nu-classification"
	cost	C constraint(regularized param: defaut=1)
	cross	K-fold CV(default=0)
dataset::iris		Edgar Anderson's Iris Data
ElemStatLearn::spam		spam mail data

Naive Bayes 예제

```
library(e1071)
model <- naiveBayes(Species ~ ., data = iris)
# predict
predict(model, iris[1:5, -5], type = "raw")
```

```
##      setosa   versicolor    virginica
## [1,]      1 2.981309e-18 2.152373e-25
## [2,]      1 3.169312e-17 6.938030e-25
## [3,]      1 2.367113e-18 7.240956e-26
## [4,]      1 3.069606e-17 8.690636e-25
## [5,]      1 1.017337e-18 8.885794e-26
```

```
pred <- predict(model, iris[,-5])
# Confusion Matrix
table(pred, iris$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         47         3
##   virginica       0          3        47
```

```
# using laplace smoothing:
model2 <- naiveBayes(Species ~ ., data = iris, laplace = 3)
```

k NN 예제

```
library(class)
set.seed(1)
y <- iris[,5]
# Partition indice into training data & test data
tr.idx <- sample(length(y), 75)
# prediction via knn
pred <- knn(train=iris[tr.idx, -5], test=iris[-tr.idx, -5], cl=y[tr.idx], k = 3)
# confusion matrix
table(pred, y[-tr.idx])
```

```
##
## pred          setosa versicolor virginica
##   setosa          24          0         0
##   versicolor       0         22         3
##   virginica        0          0        26
```

neural network 예제: nnet

```
library(nnet)
ir1 <- nnet(Species~., data=iris[tr.idx,], size = 2, decay = 5e-4)
```

```
## # weights:  19
## initial  value 98.070522
## iter  10 value 49.984837
## iter  20 value 33.254723
## iter  30 value 4.667817
## iter  40 value 3.787903
## iter  50 value 3.701295
## iter  60 value 3.672516
## iter  70 value 3.670662
## iter  80 value 3.666205
## iter  90 value 3.664552
## iter 100 value 3.662954
## final  value 3.662954
## stopped after 100 iterations
```

```
summary(ir1)
```

```
## a 4-2-3 network with 19 weights
## options were - softmax modelling  decay=5e-04
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1
##   4.82   0.17   1.73  -1.43  -2.47
```

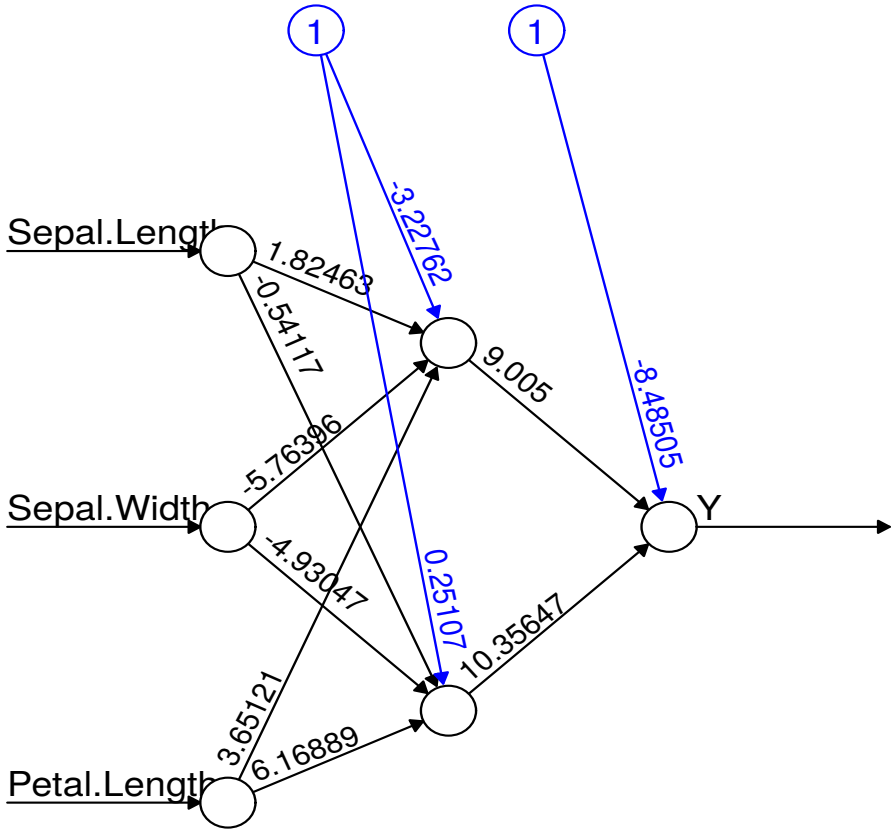
neural network 예제: neuralnet

- Not multiclass problem
- Model formula에 '.'사용시 오류발생

```
library(neuralnet)
iris2 <- iris[1:100,-5]
iris2$Y <- as.integer(iris$Species[1:100])-1
nn <- neuralnet( Y~Sepal.Length+Sepal.Width+Petal.Length, data=iris2,
                 hidden=2, err.fct="ce",
 linear.output=FALSE)
# basic result
nn
```

```
## $call
## neuralnet(formula = Y ~ Sepal.Length + Sepal.Width + Petal.Length,
##     data = iris2, hidden = 2, err.fct = "ce", linear.output = FALSE)
##
## $response
##     Y
## 1   0
## 2   0
## 3   0
## 4   0
## 5   0
## 6   0
```

Visualize NN: neuralnet



Error: 0.011453 Steps: 98

Support Vector Machine : Spam Data

```
data(spam, package = "ElemStatLearn")
```

```
str(spam)
### 'data.frame':   4601 obs. of  58 variables:
###  $ A.1 : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
###  $ A.2 : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
### ...
###  $ A.57: int  278 1028 2259 191 191 54 112 49 1257 749 ...
###  $ spam: Factor w/ 2 levels "email","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

Support Vector Machine : parmeter tunning & fitting

```r
# sampling index of train data
tr.idx <- sample(nrow(spam), 0.7*nrow(spam))
# tunning parameters
# tuning
obj <- tune(svm, spam ~ ., data = spam[tr.idx,],
            ranges = list(cost = c(0.25, 0.35)))
summary(obj)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.35
##
## - best performance: 0.08074534161
##
## - Detailed performance results:
##   cost          error    dispersion
## 1 0.25 0.08322981366 0.01093591109
## 2 0.35 0.08074534161 0.01003660426
```

Support Vector Machine : parmeter tunning & fitting

```r
# fit svm model
model <- svm(spam ~., data = spam[tr.idx,], cost=obj$best.parameters$cost)
summary(model) # coefs
```

```
##
## Call:
## svm(formula = spam ~ ., data = spam[tr.idx, ], cost = obj$best.parameters$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.35
##       gamma:  0.01754385965
##
## Number of Support Vectors:  1142
##
##  ( 588 554 )
##
##
## Number of Classes:  2
##
## Levels:
##  email spam
```

SVM : prediction & performance evaluation

```
pred <- predict(model, newdata=spam[-tr.idx,])
table(spam$spam[-tr.idx], pred)
```

```
##         pred
##          email spam
##    email   823   34
##    spam     69  455
```