SVM 이용 제거율 예측시스템

<SVM의 특징>

\* 장점 : 범주나 수치 예측 문제에 대해 사용할 수 있다.

노이즈 데이터에 영향을 크게 받지 않고 잘 과적합화되지 않는다.

특히 잘 지원되는 일부 SVM 알고리즘 때문에 신경망보다 사용하기 쉽다.

높은 정확도와 높은 프로필로 데이터 마이닝 경쟁에서 우승해 인기를 얻었다.

\* 단점 : 최적의 모델을 찾기 위해 커널과 모델에서 매개변수의 여러 가지 조합 테스트가 필요.

특히 입력 데이터 셋이 예제 개수와 속성의 수가 많다면 훈련이 느릴 수 있다.

해석하기 불가능하지 않지만, 어렵고 복잡한 블랙박스를 만든다.

\* SVM은 이미지 데이터 문제를 해결하는 데 매우 적합하다. 노이즈에 민감하지 않게 복잡한 패턴을 학습할 수 있으며, 높은 정확도로 시각 패턴을 인식할 수도 있다. 뿐만 아니라 SVM의 주요 약점에 대해 이미지 처리는 덜 비판적이다.

\* SVM 학습기가 처리하는 모든 속성은 수치여야 하고, 각 속성은 작은 간격 범위로 돼 있어야 한다.

\* 보통 Machine Learning 모형의 경우 종속변수가 클래스를 나타내는 질적 자료이고, 독립변수는 이 클래스의 특징을 나타내는 변수로 양적 변수와 질적 변수를 포함하고 있는 경우에 독립변수를 이용하여 종속변수의 클래스를 예측하는 방법론이다.

<프로젝트에 적용>

입력 데이터(화상검사로부터 나온 데이터) : 디캡 전후 잔여 OCA 좌표 수, 디캡 전후 잔여 OCA 밀도 수준, 디캡 전후 OCA의 무게

총 80개 데이터 중 training data=64개, test data=16개로 8:2의 비율로 사용

1, 잔여 oca라는 전체 데이터 셋 생성, 그 안의 속성으로 oca의 좌표 수, oca의 무게, oca 밀도 수준 수치적으로 나타냄 (이 부분은 java로 화상검사 데이터를 끌고 옴)

2, 잔여 oca 데이터 셋을 8:2의 비율로 training data와 test data로 나눈다.

samp <- c(sample(1:40,32), sample(41:80,32))

3, e1071 함수 사용할 경우 다음과 같이 training set과 test set을 정해줌

oca.tr <- oca[samp,] oca.te <- oca[-samp,]

(이 때 우리가 구하고자 하는 속성 예측 값, 그것이 좌표 수라면 들어가는 training과 test 데이터의 두번째 화상검사 후 데이터도 그 좌표 수에 해당하는 것만 들어감)

4. 각 세부 항목 별 SVM 모델 형성

model1 <- svm(좌표 수~., data=oca 전체 데이터)

model2 <- svm(무게~., data= oca 전체 데이터)

model3 < svm(밀도 수준~., data= oca 전체 데이터)

5, SVM 모델을 실행시켜보면 각 항목(변수)들에 대해서 SVM 수행되는 것 확인 가능

6, 만든 모델과 test 데이터 셋으로 predict 수행

pred <- predict(model1, oca.te, “probabilities”)

7, 예측 한 결과를 table(pred, oca전체데이터$좌표수) 형태로 표로 나타내어 확인 가능

<R로 실행시키기 – 1) kernlab 함수 이용>

> letters <- read.csv("C:/Users/CHJ/R\_HJ/Machine Learning with R code/mlr-ko/chapter 7/letterdata.csv")

> str(letters)

#str(data) : 데이터 구조, 변수명, 변수 개수, 관찰치 개수, 변수별 상위 데이터셋 미리보기 가능

'data.frame': 20000 obs. of 17 variables:

$ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...

$ xbox : int 2 5 4 7 2 4 4 1 2 11 ...

$ ybox : int 8 12 11 11 1 11 2 1 2 15 ...이하 생략

#데이터 읽고 각 분류의 속성 수와 문자 26개인 것 확인 가능

> letters\_train <- letters[1:16000, ]

> letters\_test <- letters[16001:20000, ]

#training과 test 데이터 설정, 8:2로 사용

> library(kernlab)

> letter\_classifier <- ksvm(letter ~., data=letters\_train, kernel="vanilladot")

Setting default kernel parameters

##모델 만드는 과정 : m<-ksmv(target~predictors, data=mydata, kernel="rbfdot", C=1)일 때

target=모델링 하고자 하는 데이터 프레임, predictors=예측에 사용하는 데이터 프레임의 속성 명시한 R공식, data=target과 predictor 변수가 포함돼 있는 데이터 프레임

kernel="rbfdot"(radial basis), "polydot"(polynomial, "tanhdot"(hyperbolic tangentsigmoid), "vanilladot"(linear) 등의 비선형을 명시, C='soft margin'보다 큰 값은 좁은 여백 만듬

##우리 모델 : m <- ksmv(잔여OCA정보~., data=구하려는 속성 들어간 트레이닝 데이터(좌표 수 training data), kernel=”rbf dot”)로 각 좌표 수, 무게, 밀도 수준으로 3번 돌림

> letter\_classifier

Support Vector Machine object of class "ksvm"

SV type: C-svc (classification)

parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 7037

Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524 -32.7694 -49.9786 -18.1824 이하 생략

Training error : 0.130062

#SVM모델 수행 완료, 얼마나 잘 수행됐는지 알려주지만 테스트 데이터로 성능 살펴볼 필요 있음.

> letter\_predictions <- predict(letter\_classifier, letters\_test)

##예측하기 : p <- predict(m, test, type="response") 일 때 m=ksvm() 함수로 훈련된 모델, test=훈련데이터와 같은 속성을 가진 테스트 데이터를 포함하는 데이터 프레임,

type=예측이 response(예측된 범주)인지 probabilities(예측된 확률, 범주당 하나의 열)인지 명시함. 함수는 type 매개변수에 따라 예측된 범주(또는 확률)의 벡터를 반환함.

#type 매개변수를 명시하지 않으면 기본적으로 response가 사용됨

##우리 모델 : p <- predict(m, 속성 별 트레이닝 데이터)

> head(letter\_predictions)

[1] U N V X N H

Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

#head() 함수 사용해서 상위 6개의 관측치 미리보기 가능

> table(letter\_predictions, letters\_test$letter)

#분류기가 얼마나 잘 수행했는지 살펴보기 위해 테스트 데이터셋에서 실제 문자와 예측된 문자를 비교해야 함, 대각선 값인 144, 121 등은 예측과 실제가 일치한 값 나타냄

#잘못된 결과도 나타내줌, B행 D열의 5는 실제 D지만 B로 잘못 예측한 다섯개를 나타내줌

> agreement <- letter\_predictions == letters\_test$letters

#데이터 프레임 이름$변수명 : 객체를 지정, 어디서 변수를 가져와야 할 지 인식 가능

> table(agreement)

FALSE TRUE

643 3357

#모델의 예측값과 실제 값이 일치하는지 true 또는 false로 반환해줌, table()함수 사용해 4000개 중 3357개를 정확하게 식별했음을 알 수 있음

> prop.table(table(agreement))

agreement

FALSE TRUE

0.16075 0.83925

#이를 비율로서 나타내면 정확도는 약 84%이다.

#단순한 선형 커널 함수가 아닌 복잡한 커널 함수 사용해서 SVM 모델 수립 가능, 가우시안 RBF 커널 사용

> letter\_classifier\_rbf <- ksvm(letter~., data=letters\_train, kernel="rbfdot")

> letter\_predictions\_rbf <- predict(letter\_classifier\_rbf, letters\_test)

> agreement\_rbf <- letter\_predictions\_rbf == letters\_test$letter

> table(agreement\_rbf)

agreement\_rbf

FALSE TRUE

278 3722

> prop.table(table(agreement\_rbf))

agreement\_rbf

FALSE TRUE

0.0695 0.9305

#커널 변경만으로 정확도가 84%에서 93%로 증가,

<R로 실행시키기 – 2) e1071 함수 이용>

> library(e1071)

> data(iris)

#iris는 Species를 Setosa, Versicolor, Virginica 세가지 클래스로 나누어 놓은 데이터

> samp <- c(sample(1:50,25), sample(51:100.25), sample(101:150,25))

> #총 150개 데이터 중 75개 추출해 학습, 나머지 75개는 test 데이터로

> iris.tr <- iris[samp,]

> iris.te <- iris[-samp,]

> model <- svm(Species~., data=iris)

> model

Call:

svm(formula = Species ~ ., data = iris)

Parameters: SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

gamma: 0.25

Number of Support Vectors: 51

#Species 변수에 대해서 SVM 수행, e1071 라이브러리에서 default kernel 값은 radial임, gamma는 기울기와 비슷

#cost는 과적합을 막는 정도를 지정하는 파라미터. 즉 잘못 분류하면 얼마만큼의 비용을 지불할 것인지 결정하는 것

> pred <- predict(model, iris[, -5])

> system.time(pred <- predict(svm\_model1,x))

# 돌아가는 시간 계산 가능

#x <- subset(iris, select=-Species)

#y <- Species 로 준 상태라면 x와 y 변수로 사용 가능

> table(pred, iris$Species)

pred setosa versicolor virginica

setosa 50 0 0

versicolor 0 48 2

virginica 0 2 48

#predict라는 함수에서 predic(m, test, type = "response")

#m=Object of class "svm", created by svm.

#test=An object containing the new input data

#type 옵션은 response, probabilities, votes 중 선택할 수 있다.

#response는 default값으로 반응값이 그대로 반환됨, probabilities는 모델 알고리즘에 따라 확률로 반환된다. votes는 각 분류 가능 범주의 순위를 출력한다.

> svm\_tune <- tune(svm, train.x=x, train.y=y,

kernel="radial", ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)))

> print(svm\_tune)

## Parameter tuning of 'svm':

## - sampling method: 10-fold cross validation

## - best parameters:

## cost gamma

## 1 0.5

## - best performance: 0.05333

#여기서 SVM 모델의 best cost와 gamma 값을 찾아줌 -> cost=1, gamma=0.5

>svm\_model\_after\_tune <- svm(Species ~ ., data=iris, kernel="radial", cost=1, gamma=0.5)

>summary(svm\_model\_after\_tune)

#최적의 cost와 gamma값을 찾았다면 svm모델을 다시 수립하고 run 돌려볼 수 있음

> pred <- predict(svm\_model\_after\_tune,x)

> system.time(predict(svm\_model\_after\_tune,x))

#새롭게 수립된 모델의 prediction 다시 수행해봄

> table(pred,y)

#예측의 결과를 표로 비교가능, y값의 경우 속성 값만 추출한 열 값

> plot(cmdscale(dist(iris[,-5])), col = as.integer(iris[,5]),pch = c("o","+")[1:150 %in% model$index + 1])

#예측한 model을 plot함수로 그려줌, support vector는 + 모양으로, 아닌 것들은 O로 나타냄