APS – Aprendizagem Estatística

K-NN

Esse método procura classificar um ponto baseado em um número K dos seus vizinhos mais próximos. A classificação é feita a partir do voto da maioria com esses vizinhos. Se o k = 3 e dois dos vizinhos mais próximos são 1 e o terceiro é 0, o ponto sendo classificado seria classificado como 1, devido a esse voto da maioria. É possível achar o número de vizinhos ideais usando validação cruzada ou testando diferentes valores até chegar em um erro total de classificação menor. Se acha o erro total de classificação fazendo a média da diferença entre os valores previsto pelo modelo, treinado usando os dados de treinamento, e as respostas verdadeiras dos dados de teste.

Regressão Logística

Esse modelo também é um de classificação binária. Ele faz uso de uma regressão para fazer essa classificação, calculando a probabilidade do evento sendo investigado acontecer. Com isso, para fazer a classificação final, é preciso escolher um ponto de cutoff, decidida ao analisar a curva ROC, em que as probabilidades calculadas pela regressão são comparadas a esse cutoff para decidir sua classificação final, sendo esse uma decisão entre 0 ou 1 por ser algo binário.

CART

O modelo de CART é feito crescendo árvores de classificação que, usando as outras variáveis, consegue montar um caminho para conseguir classificar os dados relevantes. Esse modelo consegue classificar não binariamente também, com vários nós sendo criados à medida que a árvore vai crescendo. Cada nó terminal acaba criando uma área em que, ao fazer o voto da maioria, acha-se a classificação para aquela área. Cada dado que cair naquela área será classificado de acordo com o voto da maioria presente quando se treinou o modelo. Caso a árvore fique com muitos nós e folhas, é possível podar a árvore, podendo levar a uma diminuição da variância.

Bagging

O Bagging procura reduzir a variância do modelo ao crescer muitas árvores de classificação usando todas as variáveis preditoras, como no modelo de CART acima, e combinar todas para fazer um classificador único. Ao fazer isso, o classificador combinado final fica com uma variância menor do que fazer apenas uma árvore. Ele consegue reduzir essa variância e prever melhor devido a sua utilização de Bootstrap, uma técnica de reamostragem, que cria vários dados de treinamento usados para treinar as árvores a partir dessa reamostragem. Não é preciso podar as árvores também, pois ao deixá-las grandes, estamos deixando o viés baixo e ao agrupá-las com o Bagging, a variância acaba diminuindo também.

Random Forest

Esse modelo também faz uso de árvores de decisão, porém ao invés de usar todas as variáveis preditoras, são selecionadas um número delas aleatoriamente no momento de crescer as árvores. O objetivo disso é tentar diminuir a correlação entre as árvores geradas ao fazer essa seleção das variáveis. O modelo também cresce muitas árvores, assim como o Bagging, porém as árvores são feitas paralelamente e não dependem uma da outra, diminuindo a correlação entre elas mais ainda.

Boosting

O Boosting faz uso de Bagging para gerar muitas árvores com todas as variáveis, mas ao invés de gerar as árvores paralelamente, uma depende da outra. Elas são geradas sequencialmente, baseando-se nas anteriores a fim de reduzir a variância desse jeito. No Boosting, as árvores anteriores aprimoram as futuras e acabam melhorando as previsões por estarem “aprendendo” com as iterações passadas, usando os erros das passadas para chegar em uma árvore com melhores previsões tentando sempre corrigir esses erros dos anteriores. Porém, caso seja escolhido um número muito elevado de árvores, o modelo poderá sofrer de overfit.

SVM

O SVM procura plotar todos os dados em um espaço e classificá-los através de uma reta ou plano que divide os dados maximizando a margem de separação entre as classificações. Se o dado estiver de um lado do plano, ele é classificado de um jeito e vice-versa. Porém, nem sempre os dados são linearmente separáveis, então há casos em que é preciso ser mais flexível com a regra de classificação, sendo o caso mais rígido com uma reta fixa separando os dados. Porém, para conseguir ser mais flexível, é preciso impor uma penalidade de classificar os dados incorretamente a fim de conseguir diminuir o erro de previsão do modelo. Se monta essa reta observando quais são os dados mais próximos da margem de separação.

Tabela dos Erros

|  |  |
| --- | --- |
|  | **Erro Total de Classificação** |
| **KNN** | 32.10% |
| **Regressão Logística** | 25.82% |
| **CART** | 27.89% |
| **Bagging** | 21.61% |
| **Random Forest** | 18.71% |
| **Boosting** | 19.27% |
| **SVM** | 25.18% |

rm(list = ls())

trn <- read.csv("training.csv", header = TRUE, sep = ";" )

tst <- read.csv("test.csv", header = TRUE, sep = ";")

### KNN

library(FNN)

set.seed(1234)

# Divisão da base de dados em variáveis resposta e explicativa para os dados de teste e treinamento

x\_trn\_knn <- trn[,1:11]

y\_trn\_knn <- trn[,12]

x\_tst\_knn <- tst[,1:11]

y\_tst\_knn <- tst[,12]

# Treinamento do modelo

y\_pred\_knn <- knn(x\_trn\_knn, x\_tst\_knn, y\_trn\_knn, k=2)

# Erro Total de Classificação

mean(y\_pred\_knn != y\_tst\_knn)

### Regressão Logística

library(ISLR)

set.seed(1234)

# Montagem do modelo

modelo\_rl <- glm(quality ~ fixed.acidity + volatile.acidity + citric.acid + residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide + density + pH + sulphates + alcohol, data = trn, family = binomial)

# Previsão da probabilidade

prob\_rl <- predict(modelo\_rl, newdata = tst[, c("fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar", "chlorides", "free.sulfur.dioxide", "total.sulfur.dioxide", "density", "pH", "sulphates", "alcohol")], type = "response")

projec\_rl <- cbind(tst, prob\_rl)

# Achando o ponto de cutoff

library(ROCR)

prev\_rl <- prediction(prob\_rl, projec\_rl$quality)

# Curva ROC

curva\_ROC <- performance(prev\_rl, "tpr", "fpr")

plot(curva\_ROC, col = "red", lwd = 5)

#olhando a curva ROC, foi decidido um ponto de cutoff de 0.6

# Erro Total de Classificação

table(tst$quality, prob\_rl > 0.6)

### CART

set.seed(1234)

library(tree)

# Montagem do modelo

tree\_CART <- tree(quality~., data = trn)

# Treinando o modelo

y\_pred\_CART <- predict(tree\_CART, newdata = tst, type = "class")

# Erro Total de Classificação

table(observed = tst$quality, predicted = y\_pred\_CART)

### Bagging

set.seed(1234)

library(ipred)

library(randomForest)

gbag <- bagging(quality ~ ., data = trn, coob = TRUE)

print(gbag)

### Random Forest

set.seed(1234)

# Montando o modelo

modelo\_rf <- randomForest(quality ~ ., data = trn)

# Treinando o modelo

pr\_rf <- ?predict(modelo\_rf, tst, type = "prob")

y\_pred\_rf <- predict(modelo\_rf,tst)

# Erro Total de Classificação

(test\_error\_rf <- mean(y\_pred\_rf != tst$quality))

table(Predicted = y\_pred\_rf, Observed = tst$quality)

### Boosting

set.seed(1234)

library(fastAdaboost)

# Montando o modelo

modelo\_boost <- adaboost(quality ~ ., trn, nIter = 50)

# Treinando o modelo

y\_pred\_boost <- predict(modelo\_boost, newdata = tst)

# Erro Total de Classificação

y\_pred\_boost$error

### SVM

set.seed(1234)

library(e1071)

# Montagem do modelo

modelo\_svm <- svm(quality ~ ., data = trn, kernel = "linear", scale = FALSE, type = "C-classification")

# Treinando o modelo

y\_pred\_svm <- predict(modelo\_svm, tst)

# Erro total de Classificação

mean(y\_pred\_svm != tst$quality)