

임헌정

2017년 5월 19일

SK Machine Learning Course

1. Random Forest

1.1. Source : 별도 첨부

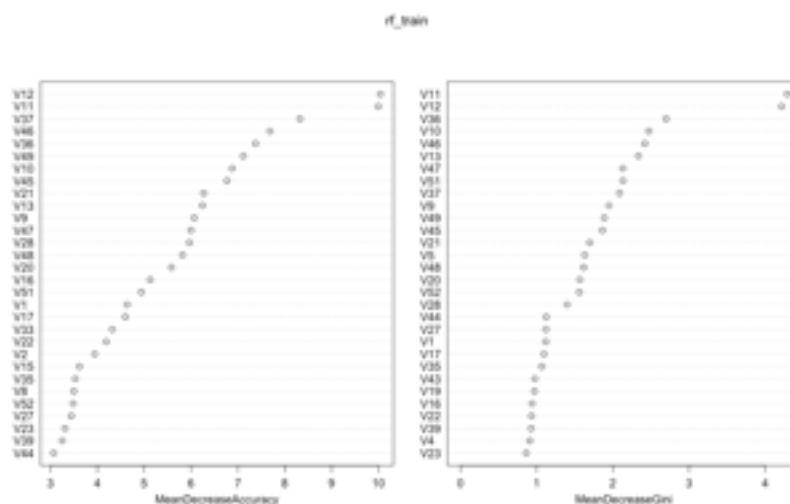
1.2. Training

```
Call:
  randomForest(formula = V61 ~ ., data = train.sonar, mtry = 8,          importance = T)
  Type of random forest: classification
    Number of trees: 500
No. of variables tried at each split: 8

  OOB estimate of  error rate: 13.79%
Confusion matrix:
  M  R class.error
M 57 12  0.1739130
R  8 68  0.1052632
```

1.3. 변수 중요도

1.3.1. V11,V12 변수가 중요한 변수로 알려짐



1.4. Test

1.4.1. 약 83%의 정확성을 보임

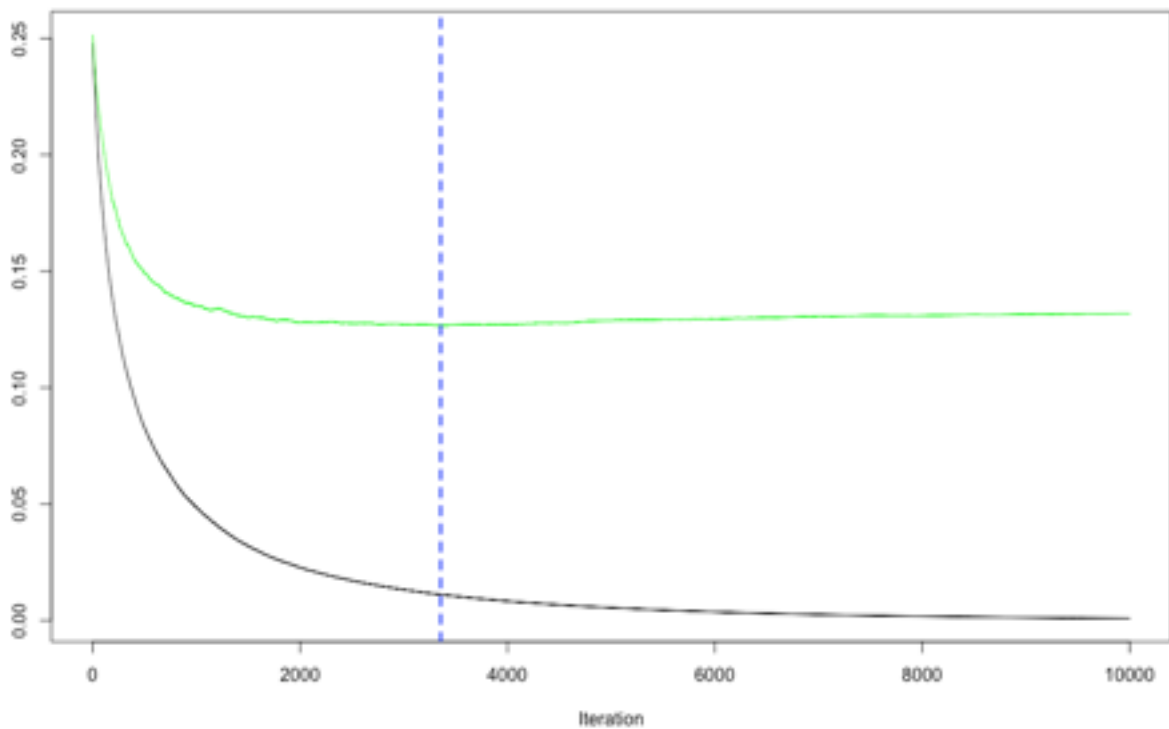
```
> err_rate
      test err
RF (m=8) 0.1746032
> varImpPlot(rf, train)
```

2. Boosting

2.1. Source : 별도 첨부

2.2. Training

2.2.1. Best Iteration



2.3. Test

2.3.1. 100% 확률로 분류

```
> pred.gbm
[1] 1.4822471 1.6362988 2.8663180 1.9717670 1.8588258 1.8273315 1.3533269 1.8937540 1.1463959 1.2417211 1.2879686 1.8794538 1.9786242 1.1286568 0.9366835 1.1786331 1.8293688 1.3878583
[19] 0.8818339 1.8384837 0.9782552 1.8274728 0.9235187 0.7931878 1.3782676 1.8258285 1.2813419 1.7483353 1.3362361 1.9751183 1.9989686 1.5146425 1.7541361 1.7988894 1.6258842 1.7514423
[37] 1.8282325 1.9642556 1.8751528 1.9886632 1.7358916 1.9930185 1.9388998 1.4357415 1.6684357 1.4987499 1.8367815 1.8282786 1.4199496 1.8748847 1.8897519 1.8538234 1.6773479 1.9137341
[55] 2.1173929 2.2548861 1.9642576 1.9177887 1.9931386 2.1152813 2.8388233 1.8224754 1.8288715
> sum(isCuts = test.sonar$W6, pred= ifelse(pred.gbm > 0.5, 1, 0))
pred
obs 1
W 28
R 35
> 1- sum(ifelse(ifelse(pred.gbm>0.5, 1, 0)==test.sonar$W6, 1, 0))
[1] 1
```

3. SVM

3.1. Source : 별도 첨부

3.2. Training

```
Call:
svm(formula = V61 ~ ., data = train.sonar)
```

```
Parameters:
  SVM-Type:  C-classification
SVM-Kernel:  radial
    cost:    1
  gamma:    0.01666667
```

```
Number of Support Vectors: 117
```

```
( 56 61 )
```

```
Number of Classes: 2
```

```
Levels:
M R
```

3.3. Test

3.3.1. 약 83%의 정확성을 보임

```
> table(obs = test.sonar$V61, pred=pred.svm)
      pred
obs  M  R
M   19  9
R    2 33
> length(which(pred.svm != test.sonar$V61))/nrow(test.sonar)
[1] 0.1746032
```

4. DNN

4.1. Source

4.2. Train

4.2.1. Input & Hidden Layer

4.2.1.1. Activation Function : Relu

4.2.2. Output Layer

4.2.2.1. Activation Function : softmax

4.2.3. Complie

4.2.3.1. Loss Function : use

4.2.3.2. Optimizer : adam

4.2.3.3. metric : accuracy

4.2.4. Fitting

4.2.4.1. Epoch : 100

4.2.4.2. Batch Size : 10

4.3. Test

4.3.1. HiddenLayer(10*10) : 약 82%의 정확성을 보임

4.3.2. HiddenLayer(20*20) : 약 80%의 정확성을 보임

```
Epoch 80/100
145/145 [=====] - 0s - loss: 0.0695 - acc: 0.9834 - val_loss: 0.1625 - val_acc: 0.8895
Epoch 81/100
145/145 [=====] - 0s - loss: 0.0528 - acc: 0.9448 - val_loss: 0.1639 - val_acc: 0.8895
Epoch 82/100
145/145 [=====] - 0s - loss: 0.0688 - acc: 0.9183 - val_loss: 0.1679 - val_acc: 0.7468
Epoch 83/100
145/145 [=====] - 0s - loss: 0.0591 - acc: 0.9172 - val_loss: 0.1599 - val_acc: 0.8895
Epoch 84/100
145/145 [=====] - 0s - loss: 0.0548 - acc: 0.9379 - val_loss: 0.1566 - val_acc: 0.8254
Epoch 85/100
145/145 [=====] - 0s - loss: 0.0534 - acc: 0.9172 - val_loss: 0.1659 - val_acc: 0.8895
Epoch 86/100
145/145 [=====] - 0s - loss: 0.0568 - acc: 0.9379 - val_loss: 0.1581 - val_acc: 0.8895
Epoch 87/100
145/145 [=====] - 0s - loss: 0.0498 - acc: 0.9517 - val_loss: 0.1652 - val_acc: 0.8895
Epoch 88/100
145/145 [=====] - 0s - loss: 0.0468 - acc: 0.9517 - val_loss: 0.1624 - val_acc: 0.8254
Epoch 89/100
145/145 [=====] - 0s - loss: 0.0462 - acc: 0.9448 - val_loss: 0.1681 - val_acc: 0.8895
Epoch 90/100
145/145 [=====] - 0s - loss: 0.0427 - acc: 0.9517 - val_loss: 0.1499 - val_acc: 0.8895
Epoch 91/100
145/145 [=====] - 0s - loss: 0.0435 - acc: 0.9448 - val_loss: 0.1568 - val_acc: 0.7937
Epoch 92/100
145/145 [=====] - 0s - loss: 0.0435 - acc: 0.9517 - val_loss: 0.1568 - val_acc: 0.8254
Epoch 93/100
145/145 [=====] - 0s - loss: 0.0486 - acc: 0.9517 - val_loss: 0.1528 - val_acc: 0.8895
Epoch 94/100
145/145 [=====] - 0s - loss: 0.0589 - acc: 0.9172 - val_loss: 0.1573 - val_acc: 0.8895
Epoch 95/100
145/145 [=====] - 0s - loss: 0.0482 - acc: 0.9586 - val_loss: 0.1662 - val_acc: 0.8895
Epoch 96/100
145/145 [=====] - 0s - loss: 0.0414 - acc: 0.9586 - val_loss: 0.1586 - val_acc: 0.8254
Epoch 97/100
145/145 [=====] - 0s - loss: 0.0384 - acc: 0.9655 - val_loss: 0.1578 - val_acc: 0.8254
Epoch 98/100
145/145 [=====] - 0s - loss: 0.0352 - acc: 0.9655 - val_loss: 0.1541 - val_acc: 0.8895
Epoch 99/100
145/145 [=====] - 0s - loss: 0.0413 - acc: 0.9586 - val_loss: 0.1694 - val_acc: 0.8895
Epoch 100/100
145/145 [=====] - 0s - loss: 0.0391 - acc: 0.9517 - val_loss: 0.1697 - val_acc: 0.8895

Model1
Test score: 0.153561438525
Test accuracy: 0.825396825397

Model2
Test score: 0.169655457952
Test accuracy: 0.809523809524
```

5. LASSO

5.1 Source : 별도 첨부

5.2 Training

5.2.1. best lambda search : `lam = cv.out$lambda.min` `lam = 0.0198393`

5.3 Test

5.3.1 : 약 76%의 정확도를 보임.

```
> table(obs = test.sonar$V61, pred = ifelse(pred_lasso == 1, "M", "R"))
      pred
obs M  R
M 17 11
R   4 31
>
> 1 - sum(ifelse(ifelse(pred_lasso == 1, "R", "M") == test.sonar$V61, 1, 0)) / nrow(test.sonar)
[1] 0.7619048
> |
```

* 종합 결과

* 해당 데이터는 Boosting 알고리즘이 정확도가 가장 높다.

모델	정확도
Random Forest	83%
Boosting	100%
SVM	83%
DNN (10*10)	82%
DNN (20*20)	80%
LASSO	76%