HW9

Yigao Li

April 15, 2018

Dermatology

Data Cleaning

```
set.seed(1)
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
## randomForest 4.6-14
```

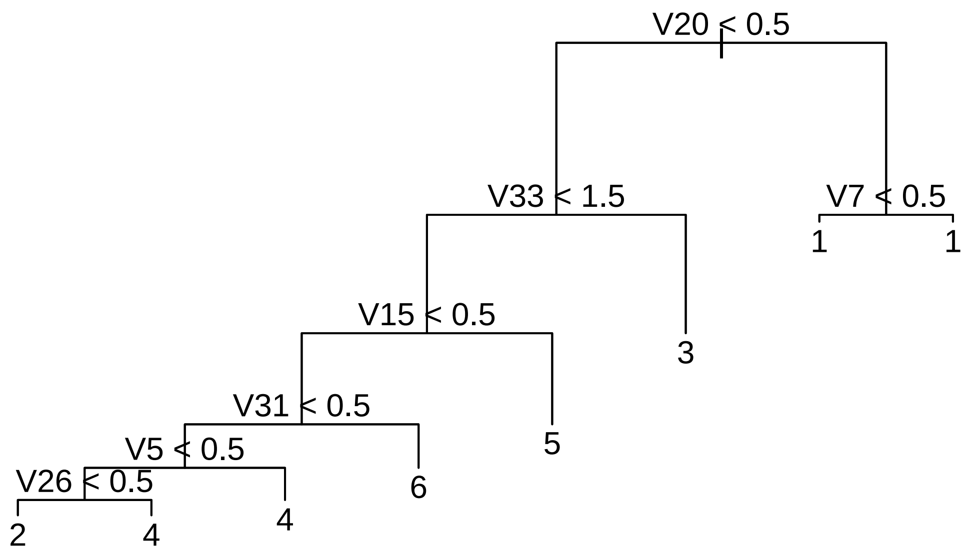
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
derm <- read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/dermatology/dermatology.da
                   sep = ',')
derm$V35 <- as.factor(derm$V35)
derm <- derm[!derm$V34 == '?',]
derm$V34 <- as.numeric(derm$V34)
n <- dim(derm)[1]
train <- sample(n, round(n*0.7))
test <- -train
derm.train <- derm[train,]
derm.test <- derm[test,]
```

This dataset is about erythemato-squamous disease in dermatology. It has 34 attributes, in which 33 are linear and 1 is nominal, and 1 class response. Response has 6 classes. The "Age" attribute has 8 missing values. After removing them, the dataset contains 358 observations, and 70% of them are randomly sampled to be training dataset. The rest of the data is for testing use. The goal is to predict type of disease from other 34 variables, and eventually to find the best few variables to predict skin disease.

Decision Tree

```
derm.tree <- tree(V35 ~ ., data = derm.train)
plot(derm.tree)
text(derm.tree, pretty = 0)
```




```r
tree.pred <- predict(derm.tree, derm.test, type = 'class')
table(derm.test$V35, tree.pred)
```

```
##    tree.pred
##      1  2  3  4  5  6
##   1 36  3  0  0  0  0
##   2  0 21  0  0  0  1
##   3  0  0 22  0  0  0
##   4  0  1  0  8  0  0
##   5  0  0  0  0 10  0
##   6  0  0  0  0  0  5
```

We generate a decision tree considering all other 34 possible variables. The tree uses "*clubbing of the rete ridges*", "*follicular papules*", "*band-like infiltrate*", "*fibrosis of the papillary dermis*", "*fibrosis of the papillary dermis*", "*perifollicular parakeratosis*", "*koebner phenomenon*" and "*disappearance of the granular layer*". Overall, this tree has 5 out of 107 errors when performing decision tree to test data. The test error rate is appoximately 4.6729%.

Bagging

```r
set.seed(5)
derm.bag <- randomForest(V35 ~ ., data = derm.train, mtry = 34, importance = TRUE)
derm.bag
```

```
##
## Call:
```

```
##  randomForest(formula = V35 ~ ., data = derm.train, mtry = 34,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 34
##
##         OOB estimate of  error rate: 3.59%
## Confusion matrix:
##    1  2  3  4  5  6 class.error
## 1 72  0  0  0  0  0  0.00000000
## 2  0 35  1  2  0  0  0.07894737
## 3  0  0 48  0  1  0  0.02040816
## 4  0  2  0 37  0  0  0.05128205
## 5  0  0  0  0 38  0  0.00000000
## 6  2  1  0  0  0 12  0.20000000
```

```r
bag.pred <- predict(derm.bag, derm.test)
table(derm.test$V35, bag.pred)
```

```
##    bag.pred
##      1  2  3  4  5  6
##   1 36  3  0  0  0  0
##   2  0 20  0  1  0  1
##   3  0  0 22  0  0  0
##   4  0  1  0  8  0  0
##   5  0  0  0  0 10  0
##   6  0  0  0  0  0  5
```

Here we apply bagging to dermotology dataset. We perform bagging where all 34 predictors are tried at each split. The training error rate is 3.59% and the test error rate is 6 out of 107, approximately 5.6%, which is a little worse than our previous decision tree.

Interesting thing to note that decision tree and bagging both mistakenly predict 3 **psoriasis** cases as **seboreic dermatitis**.

Random Forest

```r
set.seed(8)
derm.rf <- randomForest(V35 ~ ., data = derm.train, mtry = 6, importance = TRUE)
derm.rf
```

```
##
## Call:
##  randomForest(formula = V35 ~ ., data = derm.train, mtry = 6,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 2.39%
## Confusion matrix:
##    1  2  3  4  5  6 class.error
## 1 72  0  0  0  0  0  0.00000000
## 2  0 36  0  2  0  0  0.05263158
## 3  0  0 49  0  0  0  0.00000000
## 4  0  3  0 36  0  0  0.07692308
## 5  0  0  0  0 38  0  0.00000000
```

```
## 6  1  0  0  0  0 14  0.06666667
```

```r
rf.pred <- predict(derm.rf, derm.test)
table(derm.test$V35, rf.pred)
```

```
##    rf.pred
##      1  2  3  4  5  6
##   1 39  0  0  0  0  0
##   2  0 21  0  1  0  0
##   3  0  0 22  0  0  0
##   4  0  0  0  9  0  0
##   5  0  0  0  0 10  0
##   6  0  0  0  0  0  5
```

Finally we apply random forest to our data, comparing to bagging, here we only use $6 \approx \sqrt{34}$ variables at each split. The training error rate is 2.39% and random forest only makes one mistake when applying to testing dataset, and the corresponding error rate is $\approx 0.93458\%$, which is a much better result with respect to decision tree and bagging.

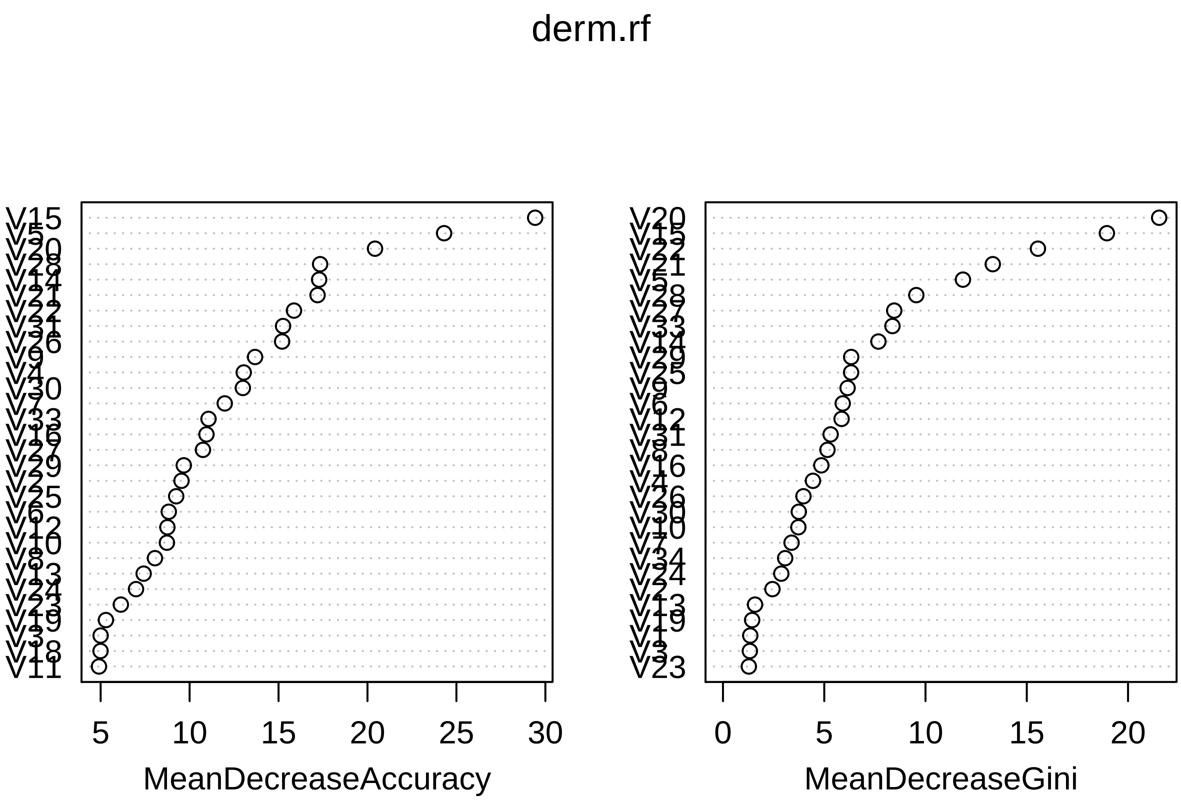
Variable Importance

```r
importance(derm.rf)
```

```
##              1          2           3          4           5          6
## V1   1.1212352  2.5810701  0.07432983  3.4247063  0.56611974  2.0265386
## V2   2.5966143  4.1230171  0.45030524  7.5267071  3.87182327  3.3030289
## V3   4.3786258  1.4539432  0.86838174  1.5142744  0.20792153  1.0817478
## V4   1.7774080  6.5223841  1.78637818 10.8169912  5.11576322  5.4041109
## V5   1.9320871 20.3399767 -0.24667056 22.1401297 11.03839398  9.9501973
## V6   5.6058381  7.2201896  8.04763073  7.7345735  7.59898153  5.0663081
## V7   3.0823792  6.4561625  1.67838656  6.9542008  0.18918319 11.8006920
## V8   4.6348953  6.8256980  6.97980528  7.4676550  6.80266281  5.2582367
## V9   6.7781432  7.9275264  3.91986426 10.3805723  9.46061061 11.0098053
## V10  6.1242103 -0.4315894  2.29147777  7.8622167  7.10332926 -0.3328987
## V11  0.1496289  1.4669249  1.28959194  3.6413765  3.01073774  3.8266497
## V12  6.0187695  7.2479854  8.11216013  7.9238172  6.62884893  5.7464442
## V13  2.9965471  5.0699774 -1.00100150  3.5102417  2.99284732  3.1751972
## V14  4.7373822 12.9309548  4.14038112 12.2396044 10.49835827  6.2075543
## V15 10.2077146 14.7311157  4.27305554 14.8482763 31.27834739  9.8541113
## V16  5.3236949  9.0542419  1.98287543  5.6584495  5.27792839  3.6047153
## V17 -1.1528252 -3.2455858  1.40606443  5.8908118  1.77204991 -0.9019504
## V18  1.0286997  5.4924750  0.84330005  0.4897803 -1.59625380  3.2250154
## V19  3.8265373  1.2015946  1.36144413  2.4329604  2.57943667 -0.7332947
## V20 20.8957022 14.0088634 10.86277079 14.2723516 15.46420416  1.8626257
## V21 13.0917696  5.1626250  7.14774406 17.7396384 12.63024980 10.9859163
## V22 15.0141086  7.7711180  8.53324740 11.8007163 10.53324312 10.3136442
## V23  3.8136043 -1.7425071  1.39687941  5.5855788  4.86853589  3.0996530
## V24  3.8946713  5.8782793  3.74825036  4.0345282  6.28286904  3.9869826
## V25  6.1368189  7.9261179  8.70343359  8.5201252  6.89973494  6.4438753
## V26  2.5432957 15.3031585  0.92680159  9.0822545  7.31245830  4.4142856
## V27  6.8837175  9.1947183 10.68014619  9.0054636  8.14987542  5.8481315
## V28 11.6799375 13.2150866 -0.71853319  9.6520179 11.13253927  0.5332509
## V29  5.8297399  7.5380078  8.92114520  7.2536225  7.44067291  4.7531954
## V30  6.1773822  6.1560685  1.41325714  5.3477228  2.24617538 12.5694339
```

```
## V31  5.3944146  6.7796388  1.85892432  6.3131841  6.10068033 16.1794985
## V32 -1.1048485  0.9387415  3.28824269 -1.5581816  2.32455981  0.6393903
## V33  6.0737396  6.1982695 10.66560735 10.5191150  8.56318480  6.8798691
## V34 -0.8827829  3.8295257  1.19966628 -2.4635880 -0.09507169  4.3901708
##     MeanDecreaseAccuracy MeanDecreaseGini
## V1              4.275822        1.3475518
## V2              9.544768        2.4426311
## V3              4.998245        1.3270855
## V4             13.043517        4.4422160
## V5             24.307111       11.8485800
## V6              8.829261        5.9130011
## V7             11.975872        3.3862222
## V8              8.052295        5.1588477
## V9             13.679742        6.1535511
## V10             8.724909        3.7226030
## V11             4.905307        0.6787429
## V12             8.752492        5.8567893
## V13             7.417686        1.5826805
## V14            17.282584        7.6745736
## V15            29.425731       18.9553905
## V16            10.948459        4.8562643
## V17             3.125845        1.1816913
## V18             4.994041        1.1419023
## V19             5.295073        1.4397998
## V20            20.423773       21.5355459
## V21            17.193330       13.3210688
## V22            15.868048       15.5510609
## V23             6.134047        1.2764218
## V24             6.987553        2.8793551
## V25             9.246090        6.3262858
## V26            15.200751        3.9734746
## V27            10.750444        8.4556734
## V28            17.336754        9.5453431
## V29             9.674469        6.3295862
## V30            12.994270        3.7445560
## V31            15.253843        5.3153930
## V32             1.531842        1.2152127
## V33            11.062806        8.3668309
## V34             2.596535        3.0696892
```

```r
varImpPlot(derm.rf)
```



According to results from `importance()` and `varImpPlot()` applying to our random forest model, the 5 most important variables to predict skin diseases are *koebner phenomenon*, *fibrosis of the papillary dermis*, *clubbing of the rete ridges*, *elongation of the rete ridges* and *spongiosis*.

Conclusion

In conclusion, random forest is the best method for skin diseases classification. According to our testing results, it has only less than 1% error. After exploring more in depth, we find out that *koebner phenomenon*, *fibrosis of the papillary dermis*, *clubbing of the rete ridges*, *elongation of the rete ridges* and *spongiosis* are the 5 most important factors that help to determine the type of skin disease.

Classification Problem

```r
load("ex0408.rData")
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.4.4
```

```
## Loading required package: survival
```

```
## Loading required package: lattice
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

Random Forest

```r
set.seed(7)
mytree<- randomForest(z ~ ., data = mydf.train, mtry = 3, ntree = 400)
mytree
```

```
##
## Call:
##  randomForest(formula = z ~ ., data = mydf.train, mtry = 3, ntree = 400)
##                Type of random forest: classification
##                      Number of trees: 400
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 25.47%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE  1381 1607   0.5378179
## TRUE    940 6072   0.1340559
```

```r
mytree.train.pred <- predict(mytree, mydf.train)
table(mydf.test$z, mytree.train.pred)
```

```
##        mytree.train.pred
##         FALSE TRUE
##   FALSE   844 2137
##   TRUE   2144 4875
```

```r
mytree.test.pred <- predict(mytree, mydf.test)
table(mydf.test$z, mytree.test.pred)
```

```
##        mytree.test.pred
##         FALSE TRUE
##   FALSE  1445 1536
##   TRUE    896 6123
```

The training error rate is 42.81% and the test error rate is 24.32% calculated from the confusion matrices. These error rates hugely vary.

Boosting

```r
mydf.train$z <- as.numeric(mydf.train$z)-1
set.seed(7)
mydepth = 2
mysteps = 300
myshrink = .1
myboost = gbm(z ~ ., data = mydf.train, distribution="bernoulli", n.trees = mysteps,
              interaction.depth = mydepth, shrinkage = myshrink)
myboost.train.prob <- predict(myboost, mydf.train, n.trees = mysteps, type = "response")
myboost.train.pred <- rep(FALSE, length(myboost.train.prob))
myboost.train.pred[myboost.train.prob > 0.5] <- TRUE
table(mydf.train$z, myboost.train.pred)
```

```
##    myboost.train.pred
##     FALSE TRUE
##   0  1550 1438
##   1   803 6209
```

```
myboost.test.prob <- predict(myboost, mydf.test, n.trees = mysteps, type = "response")
myboost.test.pred <- rep(FALSE, length(myboost.test.prob))
myboost.test.pred[myboost.test.prob > 0.5] <- TRUE
table(mydf.test$z, myboost.test.pred)
```

```
##        myboost.test.pred
##         FALSE TRUE
##   FALSE  1502 1479
##   TRUE    928 6091
```

Boosting training error rate is 22.41% and test error rate is 24.07%. Comparing to random forest, training and test error are close to each other.

The reason that boosting performs better on training dataset is due to the data. According to dataset background, the `z` response column is determined by whether other numbers divide the first number. Random forest grows a full decision tree with more tree levels. However, it is impossible to split classifications based on numeric values. For example, 128 divides 8, but both 127 and 129 cannot divide 8. In this case, boosting can do better due to weak classifiers and its high bias. Boosting increases accuracy by reducing bias.