Lecture notes, week2b

(Henry T.H, 25-sep-2015)

# 1. Decision tree

### 1.1. Introduction

+ Tree is very fast. We only have to test on few variables (columns) to arrive at the final output class. The height of the tree determines the speed. We will want to minimize the height of the tree.

+ Dividing the input space by the output class. We always need the output class to build the tree. This is the key difference between decision tree and other indexing tree, in which we don't need any output class.

+ We will divide the space into the homogeneous / pure subsets using one variable.

+ We can use tree to understand the relationships between the input X and the output Y.

### 1.2. Training / growing trees

+ We can use CART, C4.5 or ID3 to develop a tree from a data table.

### 1.3. Training / growing a tree in R language

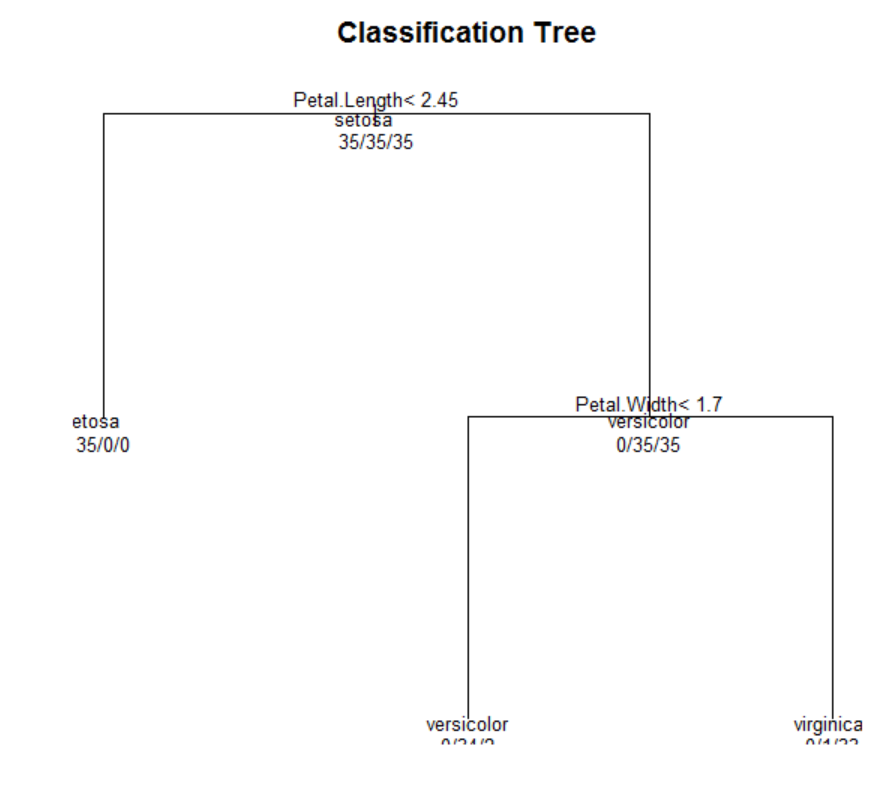
M <- train(Species ~ ., method="**rpart**", data=A)

cat("\n"); print(M$finalModel);

### 1.4. Displaying the tree

dev.new(); plot(M$finalModel, uniform=TRUE, main="Classification Tree");

text(M$finalModel, use.n=TRUE, all=TRUE, cex=.8);



### 1.5. Using a tree for prediction

y1 <- predict(M, newdata=A);

### 1.6. Understanding the tree

|  |  |
| --- | --- |
|  | When we have Pental.Length < 2.45 we can conclude the output is setosa right away.  When Pental.Length > 2.45, we have to consider a little bit more. If Pental.Width < 1.75 then we have versicolor (green). Otherwise we have viginica (blue). |

### 1.7. Tree can be biased

# 2. Bagging

### 2.1. Introduction

+ Bagging is to combine weak classifiers to create a powerful classifier.

|  |  |
| --- | --- |
| y1 = c1(x)  y2 = c2(x)  ..  yk = ck(x) | + We can combine the classifiers by averaging with weights  y = w1\*c1(x) + w2\*c2(x) + .. + wk\*ck(x)  + We can combine the classifiers by voting  y = vote(c1(x), c2(x), .., ck(x) ); |

+ We build the week classifiers from derived (bootstrapped) datasets

### 2.2. Bootstrapping

+ We have to generate random number within the range of 1 to number or rows in the table.

+ In r language, the command x <- runif(1000, 1, 40) is to create the vector of 1000 entries, each value is within 1 and 40.

+ The number will be repeated. If we want them to be unique, we can use x <- unique(x).

+ We can use Dk <- D[x, ] to select one of them.

|  |  |
| --- | --- |
| > data(iris);  > mr <- nrow(iris);  > x <- as.integer( runif(100, 1, mr) );  > print(x); cat("Summary:", min(x), max(x), mr, "\n");  [1] 122 90 25 129 81 146 43 31 89 5 147 48 55 142 61 63 54 145  [19] 134 105 30 21 92 96 32 9 149 65 16 41 107 141 65 66 3 129  [37] 31 145 129 119 6 118 73 87 25 4 21 149 29 118 136 33 20 116  [55] 95 126 119 9 1 81 9 88 52 9 68 92 51 138 60 141 5 69  [73] 81 35 80 126 60 123 3 22 103 90 80 5 56 48 141 21 80 1  [91] 69 20 92 92 143 18 108 47 8 76  Summary: 1 149 150  > D1 <- iris[x, ]; print(head(D1));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  122 5.6 2.8 4.9 2.0 virginica  90 5.5 2.5 4.0 1.3 versicolor  25 4.8 3.4 1.9 0.2 setosa  129 6.4 2.8 5.6 2.1 virginica  81 5.5 2.4 3.8 1.1 versicolor  146 6.7 3.0 5.2 2.3 virginica |  |
| > x <- as.integer( runif(100, 1, mr) );  > print(x); cat("Summary:", min(x), max(x), mr, "\n");  [1] 80 118 37 14 14 117 86 83 101 35 46 12 63 112 118 54 12 55  [19] 40 69 86 85 81 24 53 138 26 2 124 26 115 133 3 39 2 101  [37] 125 10 31 116 64 31 34 55 79 34 23 65 120 104 38 103 130 8  [55] 144 141 118 92 34 71 52 78 56 88 89 126 24 20 74 108 102 119  [73] 79 43 95 121 87 104 75 126 149 97 35 46 11 5 66 10 105 124  [91] 84 139 108 68 97 48 51 133 146 70  Summary: 2 149 150  > D2 <- iris[x, ]; print(head(D2));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  80 5.7 2.6 3.5 1.0 versicolor  118 7.7 3.8 6.7 2.2 virginica  37 5.5 3.5 1.3 0.2 setosa  14 4.3 3.0 1.1 0.1 setosa  14.1 4.3 3.0 1.1 0.1 setosa  117 6.5 3.0 5.5 1.8 virginica  > |  |

### 2.3. Learning the classifiers

+ We should choose them of the same type for code simplification. We can combine different classifiers from different types (svm, linear, logistic) though.

+ One for loop is enough.

|  |  |
| --- | --- |
| trainBag <- function(D, kpar)  {  mr <- nrow(D);  M <- list();    for(k in 1:kpar)  {  x <- unique( as.integer( runif(100, 1, mr) ) );  M[[k]] <- train(Species ~ ., method="rpart", data=D)  }    return(M);  }  predictBag <- function(M, D)  {  mc <- length(M);  mr <- nrow(D);    for(k in 1:mc)  {  D[, paste("y", k, sep="")] <- predict(M[[k]], D);  }    return(D);  } |  |

### 2.4. Averaging the result with Linear combination

+ We just combine the final result with the equation y = w1\*c1(x) + w2\*c2(x) + .. + wk\*ck(x)

+ But we have to learn the linear coefficients (w1, w2, .., wk) with the following setting

|  |  |
| --- | --- |
| y1 = w1\*c1(x1) + w2\*c2(x1) + .. + wk\*ck(x1)  y2 = w1\*c1(x2) + w2\*c2(x2) + .. + wk\*ck(x2)  ...  yn = w1\*c1(xn) + w2\*c2(xn) + .. + wk\*ck(xn) | We need to learn (w1, w2, .., wk) using this setting.  The pairs (xj, yj) is from the training dataset.  We have the (c1, c2, .., ck) learning from (D1, D2, .., Dk) in the previous phase |

### 2.5. Voting the results

+ For categorical data, we have to vote

|  |  |
| --- | --- |
| c1(x) = face  c2(x) = face  c3(x) = face  c4(x) = back  c5(x) = back | count(face) = 3  count(back) = 2  We will conclude the input x is face based on 3 voters and we ignore the other two voters. |

+ Choose the odd number of voter (components) in order to avoid tie votes.

+ We can also write the code for voting

|  |  |
| --- | --- |
| histCat <- function(x)  {  v <- unique(x);  n <- length(v);  h <- 1:n;  for(k in 1:n) h[k] <- sum(x == v[k]);    return(data.frame( val=v, cnt=h ));  }  maxCat <- function(x)  {  h <- histCat(x);  p <- which.max(h$cnt);  return( h$val[p] );  }  v <- c("b", "a", "b", "c", "a", "a");  print( histCat(v) );  print( maxCat(v) ); | val cnt  1 b 2  2 a 3  3 c 1  [1] a  Levels: a b c |

# 3. Random forest (using many trees)

### 3.1. Introduction

+ Random forest is to improve classification result by combining many trees.

+ Unlike bagging, we randomize both rows and columns. When we remove some columns, we force tree-growing algorithms to explore the other columns in the bootstrapped dataset.

+ We combine the result by voting if the output is categorical or averaging if the output is numerical.

### 3.2. Train a random forest in R language

M <- train(z ~ ., method=**"rf"**, data=A)

print(M);

### 3.3. Comparing tree and forest

|  |  |
| --- | --- |
| Tree | Forest |
| -------------testing if it fits  y y1 e1  1 face face 0  2 back back 0  3 back back 0  4 back back 0  5 back back 0  6 back back 0  y  y1 back face  back 473 33  face 3 491  Total error: 36 | -------------testing if it fits  y y1 e1  1 back back 0  2 back back 0  3 back back 0  4 face face 0  5 back back 0  6 back back 0  y  y1 back face  back 513 0  face 0 487  Total error: 0 |
| -------------testing if it handles new input  y y1 e1  1 face face 0  2 back back 0  3 face back 1  4 back back 0  5 face back 1  6 face face 0  y  y1 back face  back 2530 1098  face 0 1372  Total error: 1098 | -------------testing if it handles new input  y y1 e1  1 back back 0  2 back back 0  3 face face 0  4 face face 0  5 face face 0  6 back back 0  y  y1 back face  back 2498 0  face 0 2502  Total error: 0 |

# 4. Boosting

### 4.1. Introduction

+ Boosting method is to boost/improve the performance of weak classifiers by averaging them with learning weights.

+ Unlike bagging and random forest, we don't have to actually remove the rows but we use weights to emphasize on each of them.

+ When we have new component, we will adjust the weights of all elements

### 4.2. Training a boosting model

|  |  |
| --- | --- |
| From the book "Elements of Statistical Learning" | In the beginning we set all the weight wj of sample xj to 1/N, meaning all samples have equal weights.  We learn the weak classifier Gm(x) in the step m using the weights.  Then we compute the error in step m using  (error in step m) = (sum of weights in the wrong case by Gm(x)) / (sum of all weights ) |

### 4.3. Training a boosting model in R language

M <- train(z ~ ., method=**"gbm"**, data=A)

print(M);

### 4.4. Analysis of results

|  |  |  |
| --- | --- | --- |
| Tree | Forest | Boosting |
| -------------testing if it fits  y y1 e1  1 face face 0  2 back back 0  3 back back 0  4 back back 0  5 back back 0  6 back back 0  y  y1 back face  back 473 33  face 3 491  Total error: 36 | -------------testing if it fits  y y1 e1  1 back back 0  2 back back 0  3 back back 0  4 face face 0  5 back back 0  6 back back 0  y  y1 back face  back 513 0  face 0 487  Total error: 0 | -------------testing if it fits  y y1 e1  1 face face 0  2 back back 0  3 face face 0  4 back back 0  5 face face 0  6 face face 0  y  y1 back face  back 506 0  face 0 494  Total error: 0 |
| -------------testing if it handles new input  y y1 e1  1 face face 0  2 back back 0  3 face back 1  4 back back 0  5 face back 1  6 face face 0  y  y1 back face  back 2530 1098  face 0 1372  Total error: 1098 | -------------testing if it handles new input  y y1 e1  1 back back 0  2 back back 0  3 face face 0  4 face face 0  5 face face 0  6 back back 0  y  y1 back face  back 2498 0  face 0 2502  Total error: 0 | -------------testing if it handles new input  y y1 e1  1 face face 0  2 face face 0  3 face face 0  4 face face 0  5 back back 0  6 back back 0  y  y1 back face  back 2510 1  face 0 2489  Total error: 1 |