# Supervised Learning (Classification)

(Sumber: Data Mining dengan R)

## 1. Pendahuluan

*Supervised learning,* disebut juga sebagai analisis klasifikasi, merupakan suatu teknik statistik yang bertujuan untuk mengelompokkan data ke dalam kelas-kelas yang telah memiliki label dengan membangun suatu model yang berdasarkan kepada suatu *data training* serta memprediksi kelas dari suatu data baru. *Data training* adalah satu set data yang telah diketahui klasifikasinya.

Terdapat banyak metode yang termasuk kategori *supervised learning*, namun yang akan dibahas dalam bab ini adalah *k-nearest neighbourhood*, *decision trees*, *random forrest*, *support vector machines*, dan *naive bayes*.

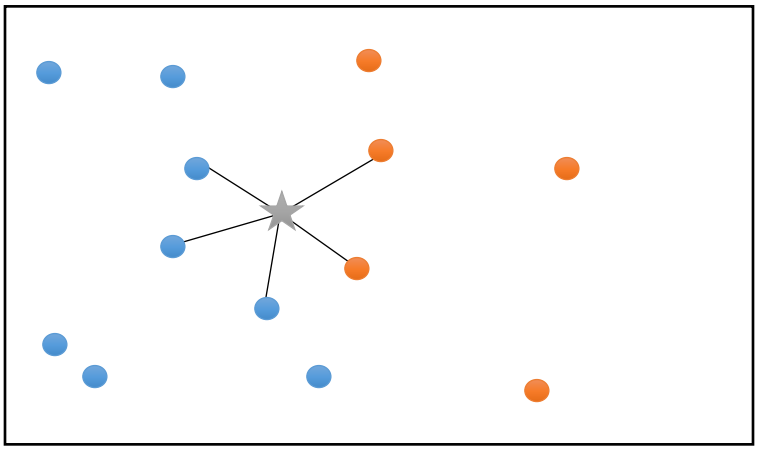
## 2. *K-Nearest Neighbour* (KNN)

KNN merupakan salah satu metode klasifikasi non-parametrik yang prinsip kerjanya mengklasifikasikan suatu objek dalam *test set* berdasarkan mayoritas kelas dari sejumlah *k* tetangga terdekatnya (*neighbour*) dalam *training set*. Nilai *k* merupakan bilangan bulat positif yang ditentukan sendiri oleh pengguna/*user*. Sebagai pertimbangan semakin kecil nilai *k* maka akan semakin rentan terhadap pengaruh *noise/outlier*, sedangkan semakin besar nilai *k* pengaruh *noise/outlier* akan semakin kecil namun akan membuat batasan antar kelas menjadi lebih kabur. Untuk mencari nilai *k* yang optimal dapat digunakan teknik *cross validation*. Nilai *k* yang ganjil dapat digunakan untuk menghindari terjadinya jumlah mayoritas kelas yang sama kuat.

Secara umum, cara kerja dari algoritma KNN untuk adalah sebagai berikut:

1. Tentukan jumlah k tetangga terdekat
2. Hitung jarak satu objek dalam *test set* dengan setiap objek dalam *training set*

3. Ambil sejumlah *k* objek dalam *training set* dengan jarak terkecil terhadap satu objek dalam *test set* tersebut
4. Lihat kelas mayoritas dari sejumlah *k* objek dalam *training set* tersebut. Satu objek dalam *test set* tersebut akan diklasifikasikan sesuai kelas mayoritas.
5. Ulangi langkah 2-4 hingga seluruh objek dalam *test set* telah terklasifikasi.



Gambar 1. Ilustrasi algoritma KNN dengan k=5

Gambar 1 menunjukkan ilustrasi algoritma KNN dengan k=5. Objek baru yang akan dicari kelasnya (dilambangkan dengan tanda bintang) dibandingkan dengan 5 tetangga terdekatnya yaitu 3 objek biru dan 2 objek merah. Karena mayoritas tetangga terdekat adalah objek biru, maka objek baru tersebut diklasifikasikan sebagai kelas objek biru.

Perhitungan jarak/*distance* merupakan hal yang esensial dalam KNN untuk mengetahui objek mana saja dalam *training set* yang menjadi tetangga terdekat. Jenis jarak yang digunakan dalam perhitungan jarak ada bermacam-macam seperti jarak Euclidean, jarak Manhattan, dan jarak Minkowski tergantung dari domain persoalan. Jarak dapat dihitung dengan formula sebagai berikut:

$$d(x_i, x_j) = (|x_{i1} - x_{j1}|^g + |x_{i2} - x_{j2}|^g + \cdots + |x_{ip} - x_{jp}|^g)^{1/g}$$

Dimana:

g = 1, untuk menghitung jarak Manhattan

g = 2, untuk menghitung jarak Euclidean

g = ∞, untuk menghitung jarak Chebychev

xi , xj adalah dua buah data yang akan dihitung jaraknya

p = atribut ke-p dari sebuah objek

KNN merupakan *lazy learner* yang berarti tidak membuat model secara eksplisit. Algoritma ini baru dijalankan ketika ada objek baru yang akan dicari nilai kelasnya. KNN juga merupakan algoritma pembelajaran berbasis contoh (*instance based learning*) dimana data diklasifikasikan berdasarkan contoh yang berlabel.

Karena KNN tidak membuat model secara eksplisit, maka akibatnya setiap kali ada objek baru yang akan diklasifikasikan, objek tersebut harus dibandingkan kembali dengan seluruh objek yang ada dalam *training set*. Hasilnya, waktu eksekusi penggunaan algoritma ini akan berbanding lurus dengan jumlah *training set*-nya sehingga algoritma ini cukup *costly* dari sisi waktu komputasinya. Namun, KNN secara umum memiliki akurasi yang cukup baik, cukup *robust* terhadap *noise/outlier*, efektif digunakan jika jumlah *training set*-nya besar, dan sering digunakan sebagai tolak ukur/pembanding/*benchmark* untuk algoritma klasifikasi lain yang lebih kompleks seperti *Artificial Neural Network* (ANN) dan *Support Vector Machine* (SVM).

**Implementasi KNN dengan R**

Implementasi algoritma KNN pada R dapat dilakukan dengan menggunakan fungsi **knn** dan **kknn**. Fungsi **knn** adalah bagian dari *package class* yang telah tersedia di R, sedangkan fungsi **kknn** adalah bagian dari *package kknn*. Jika package kknn belum tersedia maka dapat diinstall denga perintah `install.packages("kknn")`. Jenis jarak yang digunakan dalam fungsi **knn** adalah jarak Euclidean.

Sintaks fungsi *knn()* adalah sebagai berikut:

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

dan berikut ini adalah penjelasan argumen yang terdapat pada fungsi tersebut:

`train` : *training set*

`test` : *test set.*

`cl` : faktor *true classifications* dari *training set*

`k` : Jumlah tetangga terdekat yang digunakan.

`l` : voting minimum untuk keputusan tertentu, selain itu doubt.

`prob` : Jika bernilai *true*, proporsi vote untuk kelas mayoritas akan dikembalikan sebagai atribut prob.

`use.all` : Menangani kasus sama kuat. Jika bernilai true, semua objek dengan jarak yang sama nilainya dengan k terbesar akan diambil. Jika false, pemilihan secara acak dilakukan terhadap jarak yang sama nilainya dengan k untuk menggunakan tepat k tetangga terdekat.

Sedangkan untuk sintaks fungsi *kknn()* adalah sebagai berikut

```
kknn(formula, train, test, k, distance, kernel)
```

Penjelasan dari argumen pada fungsi tersebut adalah sebagai berikut:

`formula` : formula untuk menentukan variable target

`train` : *training set*

`test` : *test set*

`k` : jumlah tetangga terdekat yang diinginkan

`distance` : jenis jarak yang digunakan

`kernel` : nama kernel/fungsi yang digunakan dengan pilihan triangular, epanechnikov, biweight, triweight, cos, inv, gaussian, rank, optimal, rectangular

Contoh 1 : Implementasi dengan fungsi **knn**

Data *iris* yang ada di dalam R, merupakan data memuat pengukuran panjang dan lebar dari kelopak dan mahkota bunga (sepal length, sepal width, petal length, dan petal width) dari tiga jenis bunga iris, yaitu iris setosa, iris versicolor, dan iris virginica. Dataset *iris3* adalah data yang sama dalam bentuk array 3 dimensi berukuran 50 x 4 x 3.

Untuk melakukan analisis klasifikasi dengan metode knn menggunakan R adalah sebagai berikut:

```r
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```r
### K Nearest Neighbour ## membagi data train dan data test
dt.train <- rbind(iris3[1:25, , 1], iris3[1:25, , 2], iris3[1:25, , 3])
dt.test <- rbind(iris3[26:50, , 1], iris3[26:50, , 2], iris3[26:50, , 3])
clas <- factor(c(rep("s", 25), rep("c", 25), rep("v", 25)))

head(data.frame(dt.train, clas))
```

```
  Sepal.L. Sepal.W. Petal.L. Petal.W. clas
1      5.1      3.5      1.4      0.2    s
2      4.9      3.0      1.4      0.2    s
3      4.7      3.2      1.3      0.2    s
4      4.6      3.1      1.5      0.2    s
5      5.0      3.6      1.4      0.2    s
6      5.4      3.9      1.7      0.4    s
```

```r
library(class)

# LOOCV ##

## Define the clasifier to classify the train dataset
KnnRes <- knn.cv(dt.train, clas, k = 3)

table(KnnRes, clas)
```

```
      clas
KnnRes  c  s  v
     c 24  0  3
     s  0 25  0
     v  1  0 22
```

```r
# Knn with k = 3 Define the clasifier to classify the test dataset
KnnRes2 <- knn(train = dt.train, test = dt.test, cl = clas, k = 3)
KnnRes2
```

```
 [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c
v c
[36] c c c c c c c c c c c c c c c v c c v v v v v v v v v v c v v v v v v
[71] v v v v v
Levels: c s v
```

```r
# Confusion matrix
tab <- table(KnnRes2, clas)
tab
```

```
            clas
KnnRes2  c  s  v
      c 23  0  3
      s  0 25  0
      v  2  0 22
```

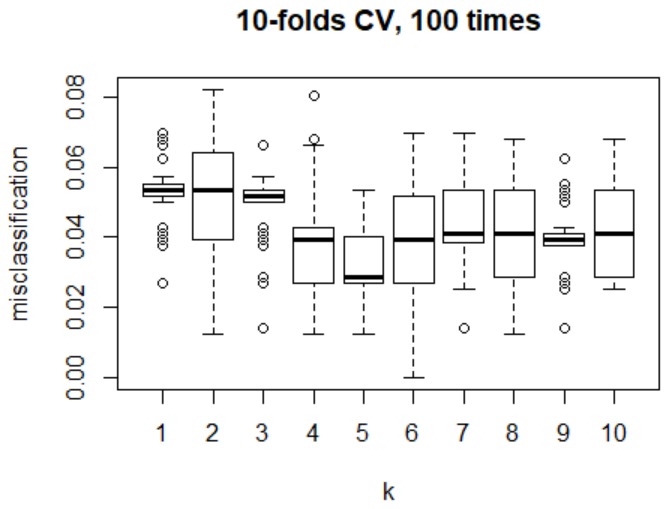
```
      n = sum(tab)
corct = sum(diag(tab))

# Misclassification Error
MisErr <- (n - corct)/n
MisErr
```

```
      [1] 0.06666667
```

```
      # Knn with k = 3
KnnRes3 <- knn(train = dt.train, test = dt.test, cl = clas, k = 2)
tab <- table(KnnRes3, clas)

n = sum(tab)
corct = sum(diag(tab))
MisErr <- (n - corct)/n

# Find The optimal K ## Install package e1071 first
library(e1071)
### 10-folds CV is performed 100 times ###

rep = 100
kfoldcv <- matrix(NA, rep, 10)

set.seed(123)
for (i in 1:rep) {
    kfold <- tune.knn(dt.train, clas, k = 1:10, tunecontrol = tune.control(sa
mpling = "cross",
        cross = 10))
    kfoldcv[i, ] <- summary(kfold)$performances[, 2]
    cat(i)
}
```

```
      1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666768697071727374757677787980818283848586878889909192939495969798991000
```

```
      colnames(kfoldcv) <- 1:10
boxplot(kfoldcv, ylab = "misclassification", xlab = "k", main = "10-folds CV,
100 times")
```




Dari boxplot di atas, terlihat bahwa hasil terbaik diperoleh pada k = 5. Berikutnya kita uji menggunakan data test dengan k = 5.

```
      KnnRes3 <- knn(dt.test, dt.train, clas, k = 3, prob = TRUE)
tab <- table(KnnRes3, clas)
tab

             clas
KnnRes3  c  s  v
      c 23  0  1
      s  0 25  0
      v  2  0 24

      n = sum(tab)
corct = sum(diag(tab))

MisErr <- (n - corct)/n
MisErr

      [1] 0.04
```

Contoh 2 (Implementasi dengan fungsi **knn**):

Masih dengan dataset yang sama dengan contoh 1, yaitu menggunakan data *Iris*. Training set untuk membangun model dan test set untuk menghitung akurasi prediksi model yang dihasilkan dihasilkan dengan menggunakan random sampling.

```
      Data <- iris

Sample <- sample(1:150, 50)
testing <- Data[Sample, ]
training <- Data[-Sample, ]

dim(Data)
      [1] 150   5
      dim(testing)
      [1] 50  5
      dim(training)
      [1] 100   5
```

Kemudian akan dibangun model berdasarkan training set. Argumen kmax adalah nilai maksimum untuk k yang digunakan oleh model dan nantinya akan dicari k optimal.

```
      library(kknn)
      model <- train.kknn(Species ~ ., data = training, kmax = 9)
model


Call:
train.kknn(formula = Species ~ ., data = training, kmax = 9)

Type of response variable: nominal
Minimal misclassification: 0.05
Best kernel: optimal
Best k: 9
```

Dari output tersebut terlihat bahwa hasil terbaik diberikan pada k = 9, hal ini bisa kita periksa dengan melihat tingkat misklasifikasi untuk setiap k.

```
      model$MISCLASS
        optimal
1    0.09
2    0.09
3    0.09
4    0.09
5    0.08
6    0.07
7    0.07
8    0.07
9    0.05
```

Kemudian dengan menggunakan model yang telah dibangun oleh data training, dapat dilihat juga tingkat akurasi hasil prediksi pada data testing.

```
      prediction <- predict(model, testing[, -5])

# untuk melihat akurasi prediksi kita buat confussion matrix
CM <- table(testing$Species, prediction)
CM

                 prediction
             setosa versicolor virginica
  setosa         14          0         0
  versicolor      0         12         2
  virginica       0          2        20
```

Seperti terlihat di confusion matrix, model yang digunakan cukup baik. Akurasi model dapat dihitung dengan menjumlahkan diagonal dibagi dengan datanya.

```
      accuracy <- (sum(diag(CM)))/sum(CM)
accuracy

      [1] 0.92
```

Secara umum ,model memiliki akurasi yang tinggi, namun hanya akurasi saja tidak cukup untuk menjamin validitas model. Perlu dilihat juga *specificity* dan *sensitivity* dari model terutama ketika kita mempunyai dataset yang *imbalanced*. Analisis *cross-validation* sangat direkomendasikan untuk melihat apakah seberapa baik sebuah model.

### 3. Decision Tree

Konsep dasar dari *decision tree* adalah membuat model aturan (*rule*) berupa *tree*/pohon dari data training yang ada. Kemudian model yang terbentuk dapat digunakan untuk mengklasifikasikan objek yang baru.

Jika *classification tree* atau pohon keputusan digunakan ketika hasil prediksi merupakan keanggotaan dari suatu kelas/kelompok, maka *regression tree* digunakan ketika hasil prediksi merupakan nilai numerik atau angka real. Pohon klasifikasi dan pohon regresi merupakan bagian dari metode *Classification and Regression Tree (CART)* yang dikembangkan oleh Leo Breiman dkk pada tahun 1993. Model yang dihasilkan oleh decision tree ini memiliki kelebihan dalam kesederhanaannya dan kemudahannya untuk dilakukan interpretasi.

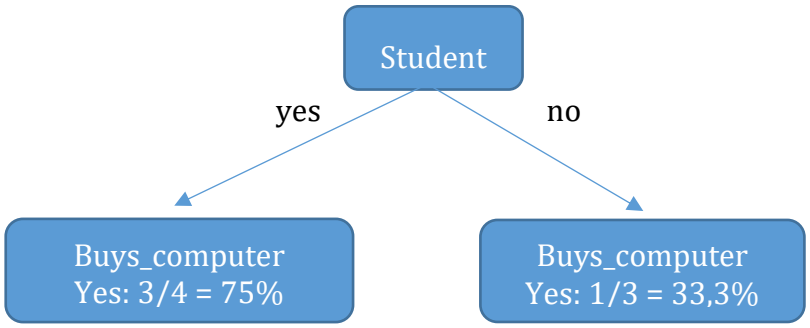
Algoritma *decision tree* secara umum bekerja secara *top-down*, dengan cara memilih atribut yang merupakan *best predictor*/prediktor terbaik sebagai root. Kemudian atribut berikutnya yang merupakan best splitting attribute menjadi cabang dari tree yang terbentuk, demikian seterusnya hingga dataset telah terbagi habis atau atribut-atribut yang ada semuanya telah menjadi cabang dari *tree*. Dalam proses pembentukan decision *tree*, terdapat beberapa metrik untuk menentukan atribut terbaik sebagai *best predictor* seperti *entropy & information gain*, *gain ratio,* dan *gini index*.

Untuk memahami perbedaan ketiga metrik tersebut, akan lebih mudah jika kita langsung melihat contoh. Misal terdapat suatu *training data* tentang pembelian komputer berdasarkan status mahasiswa dan income sebagai berikut:

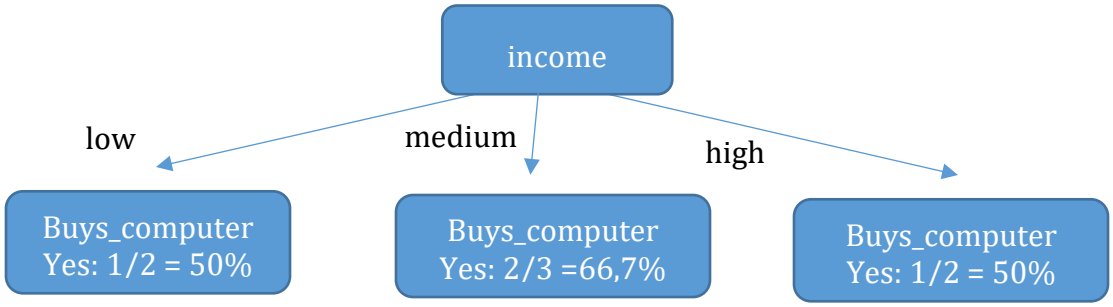
| No | Student | Income | Buys_computer |
|---|---|---|---|
| 1 | No | High | Yes |
| 2 | No | Low | No |
| 3 | No | Medium | No |
| 4 | Yes | Low | Yes |
| 5 | Yes | Medium | Yes |
| 6 | Yes | High | No |
| 7 | Yes | Medium | Yes |

Misal jika kita ingin mengidentifikasi pembelian computer sukses/yes hanya bedasarkan satu atribut saja, maka terdapat dua cara untuk melakukan hal tersebut.

Cara pertama kita menggunakan atribut student:



Cara kedua kita menggunakan atribut income:




Secara sekilas metode pertama lebih bagus dari metode kedua karena metode kedua memberikan hasil yang lebih tercampur dan acak. Algoritma *decision tree* memiliki cara yang kurang lebih sama dalam memilih atribut terbaik sebagai *split node*. Berikut adalah ilustrasi perbedaan metrik yang digunakan untuk memilih atribut dalam proses pembentukan *decision tree*.

***Entropy & Information Gain***

Istilah *entropy* diambil dari ilmu termodinamika yang berarti ukuran keacakan. Konsep *entropy* dikembangkan oleh Claude E. Shannon, yang merupakan pencetus teori informasi, pada tahun 1948. Shannon memperkenalkan konsep *entropy* pada studi statistik dan memberikan formula untuk entropi statistik:

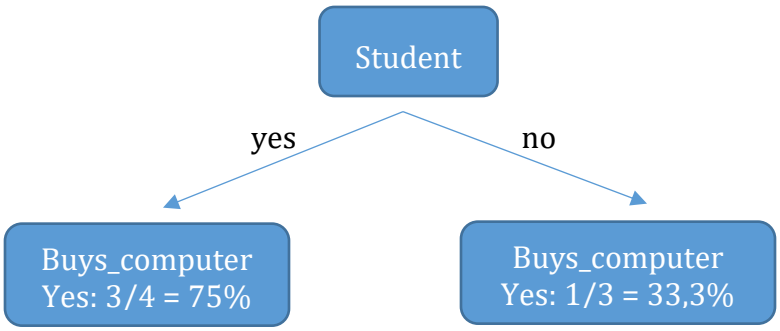
$$Entropy(S) = -\sum_{i=1}^{N} P_j log_2 P_j$$

Dimana Entropy(S) adalah entropy total dari system.

Contoh dalam ilmu peluang, peluang koin yang dilempar dan muncul gambar adalah 0,5. Demikian juga peluang koin yang dilempar dan muncul angka adalah 0,5. Entropi system pelemparan koin menurut formula yang diberikan oleh Shannon adalah Entropy(S) = - (0,5 x $\log_2(0,5)$ + 0,5 x $\log_2(0,5)$) = 1. Nilai entropy system akan maksimal (akan terjadi keacakan maksimal) pada dataset jika hasil yang mungkin terjadi memiliki peluang kemunculan yang sama. Nilai 1 adalah maksimum entropy untuk jumlah kelas 2. Semakin banyak jumlah kelas, maka nilai maksimum entropy juga semakin besar.

Karena nilai entropy akan maksimum jika peluang kemunculan setiap kelas sama, maka algoritma *decision tree* yang menggunakan metrik ini akan memilih atribut yang paling dapat mengurangi entropi maksimum entropy sebagai best predictor.

Contoh algoritma *decision tree* yang mengadopsi metrik ini adalah algoritma *Iterative Dichotomizer 3* (ID3) yang digagas oleh Ross Quinlan pada tahun 1979.




Kembali ke cara pertama yang menggunakan atribut student, kita akan menghitung nilai entropi system yaitu:

- Peluang total kemunculan *buys computer = yes* = 4/7 = 0,57

- Peluang total kemunculan *buys computer = no* = 3/7 = 0,43

- Nilai entropi maksimum *Entropy(S)* = - (0,57 x $\log_2$(0,57) + 0,43 x $\log_2$(0,43)) = 0,985

Untuk Nilai entropi anak kiri:

- Peluang kemunculan *buys_computer = yes* = 3/4 = 0.75

- Peluang kemunculann *buys_computer = no* = 1/4 = 0,25

- Nilai entropi kiri = - (0,75 x $\log_2$ (0,75) + 0,25 x $\log_2$ (0,25)) = 0,811

Untuk Nilai entropi anak kanan:

- Peluang kemunculan *buys_computer = yes* = 1/3 = 0.33

- Peluang kemunculan *buys_computer = no* = 2/3 = 0,67

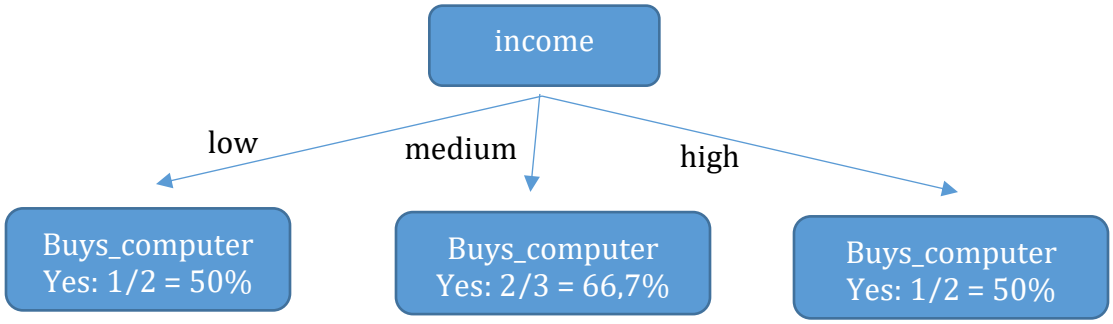
- Nilai entropi kanan = - (0,33 x $\log_2$ (0,33) + 0,67 x $\log_2$ (0,67)) = 0,915

Untuk menentukan apakah atribut tersebut merupakan *best predictor*, maka digunakan konsep *information gain*. Information gain untuk suatu atribut diperoleh dengan mengurangi entropi system dengan rata-rata tertimbang dari entropi atribut. Semakin tinggi nilai information gain, maka semakin baik bagi atribut tersebut untuk menjadi best prediktor. Formula information gain atribut adalah sebagai berikut:

$$Gain(S, A) = Entropy(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} * Entropy(S_i)$$

Menggunakan rumus tersebut, maka untuk nilai information gain dari atribut student, Gain(S, Student) = 0,985 – (4/7 * 0,811 + 3/7 * 0,915) = 0,129

Untuk cara kedua yang menggunakan atribut income:




- Nilai entropi untuk anak kiri adalah - (0,5 x $\log_2$ (0,5) + 0,5 x $\log_2$ (0,5)) = 1

- Nilai entropi untuk anak kanan adalah - (0,5 x $\log_2$ (0,5) + 0,5 x $\log_2$ (0,5)) = 1

- Nilai entropi anak tengah adalah –(0,67 x $\log_2$ (0,67) + 0,33 x $\log_2$ (0,33)) = 0,915

Maka information gain untuk atribut income adalah:

Gain(S, income) = 0,985 – (2/7 * 1 + 3/7 * 0,915 + 2/7 * 1) = 0,02

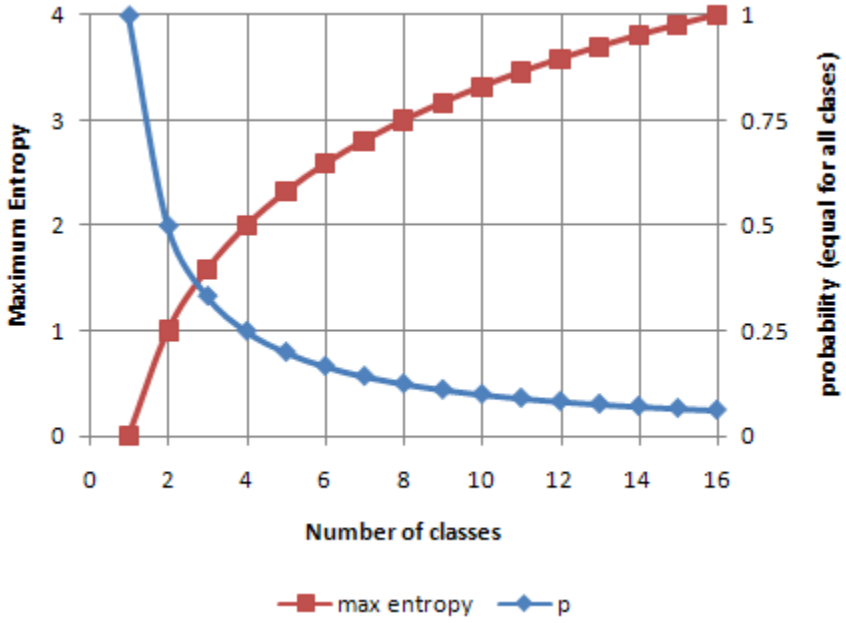
Nilai information gain untuk atribut student lebih besar dari nilai information gain untuk atribut income sehingga atribut student merupakan best predictor attribute sehingga atribut student dapat dipilih menjadi root/akar tree.

Sebagai informasi, jika nilai information gain suatu atribut adalah nol maka artinya atribut tersebut sama sekali tidak dapat mengurangi entropy system. Hasil yang mungkin

muncul pada setiap nilai atribut memiliki peluang yang sama, dengan kata lain atribut tersebut merupakan *worst predictor* dan tidak akan terpilih sebagai suatu cabang dari pohon yang akan dihasilkan.

**Gain Ratio**

Penggunaan entropy system sebagai metric dalam membangun *decision tree* memiliki masalah bahwa information gain ternyata bias terhadap atribut yang memiliki banyak nilai. Semakin banyak nilai dari atribut, maka semakin besar pula nilai dari entropy system. Bias tersebut diilustrasikan pada gambar di bawah ini:




Gambar 2 Ilustrasi perbandingan jumlah kelas dengan entropi.

Sumber: http://people.revoledu.com/kardi/tutorial/DecisionTree/how-to-measure-impurity.htm

Dari gambar tersebut terlihat bahwa untuk variable kelas yang memiliki 2 kemungkinan nilai, maka maksimum entropy adalah 1 (ingat kasus peluang pelemparan koin muncul gambar atau angka). Sedangkan untuk variable kelas yang memiliki 16 kemungkinan nilai, maka nilai maksimum entropy adalah 4.

Untuk mengatasi masalah bias ini maka digunakan konsep *Gain Ratio* yang merupakan normalisasi terhadap nilai *information gain*. Formula umum dari *gain ratio* adalah:

$$gain\ ratio = \frac{information\ gain}{split\ info}$$

Dimana formula dari *Split Info* adalah:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2(\frac{|D_j|}{|D|})$$

Sebagai contoh, untuk cara pertama yang menggunakan atribut student,

*Split Info (Student)* = - ((4/7)*log2(4/7)) + ((3/7)*log2(3/7)) = 0,985

*Gain Ratio (Student)* = 0.129/0.985 = 0.13

Untuk cara kedua yang menggunakan atribut income,

*Split Info (income)* = -((2/7)*log2(2/7) + (3/7)*log2(3/7) + (2/7)*log2(2/7)) = 1,56

*Gain Ratio (income)* = 0,02 / 1,56 = 0,0128

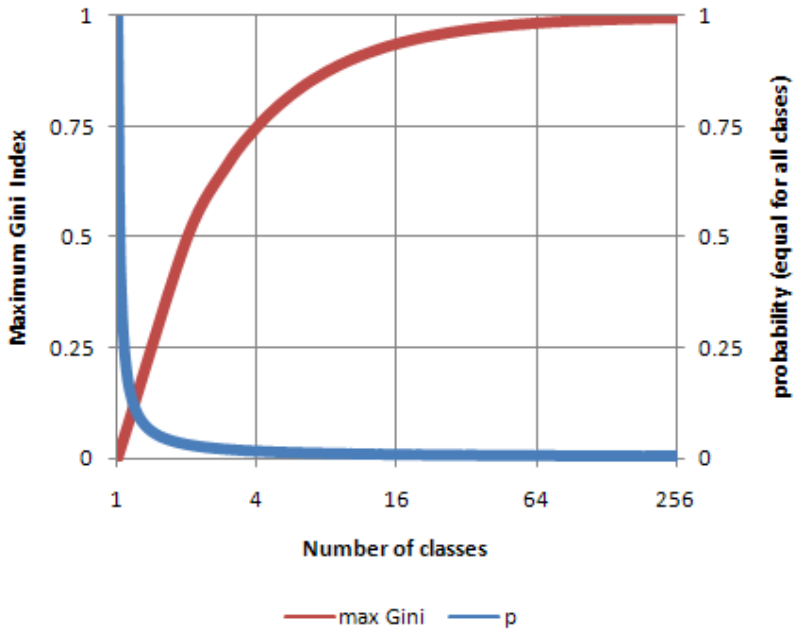
Karena *gain ratio* untuk atribut student lebih besar daripada *gain ratio* untuk atribut income, maka atribut student merupakan predictor yang lebih baik dari atribut income. Konsep gain ratio merupakan pengembangan dari konsep information gain. Contoh algoritma yang menggunakan konsep gain ratio adalah algoritma C4.5 yang juga digagas oleh Ross Quinlan. Algoritma C4.5 merupakan pengembangan dari algoritma ID3.

**Index Gini**

Satu lagi metric yang digunakan untuk membuat decision tree adalah Indeks Gini. Metrik ini digunakan oleh algoritma CART(*Classification and Regression Tree*) yang dikembangkan oleh Leo Breiman. Gini menghitung ketidakmurnian partisi data K dengan formula sebagai berikut:

$$Gini(K) = 1 - \sum_{i=1}^{n} P_i^2$$

Dimana n adalah jumlah kelas, dan $P_i$ adalah kemungkinan observasi dalam K termasuk suatu kelas. Nilai maksimum indeks gini adalah 1.




Gambar 3 Ilustrasi perbandingan jumlah kelas dengan entropi.

Sumber: http://people.revoledu.com/kardi/tutorial/DecisionTree/how-to-measure-impurity.htm

Indeks Gini mengasumsikan binary split untuk setiap atribut dalam S, misal T1 dan T2. Indeks Gini dari K dapat dihitung dengan menggunakan perhitungan:

$$Gini_s(K) = \frac{T_1}{T} Gini\ (T_1) + \frac{T_2}{T} Gini\ (T_2)$$

yang merupakan jumlah tertimbang dari setiap ketidakmurnian pada split node. Pengurangan terhadap ketidakmurnian dihitung dengan rumus:

$$Gini(K) - Gini_s(K)$$

Mirip dengan *information gain* dan *gain ratio*, atribut yang memberikan pengurangan maksimum terhadap ketidakmurnian akan diambil untuk mempartisi data.

Kembali ke contoh cara 1, kita akan menghitung *Gini(K)*:

Gini(K) = $1 - ((4/7)^2 + (3/7)^2) = 0.49$

Untuk cara 1, kita mendapatkan nilai $Gini_s(K)$:

$$Gini_s(K) = \frac{T_1}{T} Gini\ (T_1) + \frac{T_2}{T} Gini\ (T_2)$$

$Gini_s(K) = 4/7 * (1 - ((3/4)^2 + (1/4)^2) + 3/7 * (1 - ((1/3)^2 + (2/3)^2)$

$Gini_s(K)$ = 0.24 + 0.19 = 0.43

Untuk cara 2, kita mendapatkan nilai Gini

= 2/7*(1 – ((1/2)$^2$ + (1/2)$^2$)) + 3/7 *(1 – ((2/3)2 + (1/3)2)) + 2/7*(1 – ((1/2)$^2$ + (1/2)$^2$))

= 0,21

Nilai gini cara 1 lebih besar dari cara 2, artinya atribut student memberikan pengurangan terhadap ketidakmurnian yang lebih besar dari atribut income, yang artinya atribut student lebih baik sebagai predictor dalam tree yang akan dihasilkan daripada atribut income.

Dari ketiga metrik yang digunakan dapat disimpulkan bahwa atribut student lebih baik dari atribut income sebagai best predictor.

**Ilustrasi lengkap pembuatan decision tree menggunakan information gain**

Misal kita memiliki training data sebagai berikut:

| no | outlook | Temp | humidity | windy | play |
|---|---|---|---|---|---|
| 1 | sunny | hot | high | false | no |
| 2 | sunny | hot | high | true | no |
| 3 | overcast | hot | high | false | yes |
| 4 | rainy | mild | high | false | yes |
| 5 | rainy | cool | normal | false | yes |
| 6 | rainy | cool | normal | true | no |
| 7 | overcast | cool | normal | true | yes |
| 8 | sunny | mild | high | false | no |
| 9 | sunny | cool | normal | false | yes |
| 10 | rainy | mild | normal | false | yes |
| 11 | sunny | mild | normal | true | yes |
| 12 | overcast | mild | high | true | yes |
| 13 | overcast | hot | normal | false | yes |

| | | | | | |
|---|---|---|---|---|---|
| 14 | rainy | mild | high | true | no |

Kemudian misal kita akan membuat decision tree lengkap menggunakan algoritma ID3 yang menggunakan metrik information gain.

Langkah-langkahnya secara umum adalah sebagai berikut:

1. Hitung entropi total dan entropi seluruh atribut
2. Hitung gain untuk setiap atribut
3. Atribut dengan gain tertinggi akan menjadi split node
4. Filter dataset menurut split node dan ulangi langkah 2-3 hingga seluruh atribut telah terpilih atau data sudah habis dibagi.

Untuk mempermudah, training data kita ubah menjadi format berikut:

| **atribut** | | **jml kasus** | **yes** | **no** | **entropy** | **gain** |
|---|---|---|---|---|---|---|
| **total** | | 14 | 10 | 4 | | |
| **outlook** | | | | | | |
| | cloudy | 4 | 4 | 0 | | |
| | rainy | 5 | 4 | 1 | | |
| | sunny | 5 | 2 | 3 | | |
| **temperature** | | | | | | |
| | cool | 4 | 0 | 4 | | |
| | hot | 4 | 2 | 2 | | |
| | mild | 6 | 2 | 4 | | |
| **humidity** | | | | | | |
| | high | 7 | 4 | 3 | | |
| | normal | 7 | 7 | 0 | | |
| **windy** | | | | | | |
| | false | 8 | 2 | 6 | | |
| | true | 6 | 4 | 2 | | |

Entropi total = - (10/14 * $\log_2$(10/14) + 4/14 * $\log_2$(4/14)) = 0,863

**Entropi (outlook)**

Entropy(cloudy) = -(4/4 * $\log_2$(4/4) + 0/4 * $\log_2$(0/4)) = 0

Entropy(rainy) = -(4/5 * $\log_2$(4/5) + 1/5 * $\log_2$(1/5)) = 0,722

Entropy(sunny) = -(2/5 * $\log_2$(2/5) + 3/5 * $\log_2$(3/5)) = 0,97

**Entropi(temperature)**

Entropi(cool) = -(0/4 * $\log_2$(0/4) + 4/4 * $\log_2$(4/4)) = 0

Entropi(hot) = -(2/4 * $\log_2$(2/4) + 2/4 * $\log_2$(2/4)) = 1

Entropi(mild) = -(2/6 * $\log_2$(2/6) + 4/6 * $\log_2$(4/6)) = 0,918

**Entropi(humidity)**

Entropi(high) = -(4/7 * $\log_2$(4/7) + 3/7 * $\log_2$(3/7)) = 0,985

Entropi(normal) = -(7/7 * $\log_2$(7/7) + 0/7 * $\log_2$(0/7)) = 0

**Entropi(windy)**

Entropi(false) = -(2/8 * $\log_2$(2/8) + 6/8 * $\log_2$(6/8)) = 0,811

Entropi(true) = -(4/6 * $\log_2$(4/6) + 2/6 * $\log_2$(2/6)) = 0,918

Kemudian hasil perhitungan dapat kita pindahkan ke table sebelumnya

| **Atribut** | | **jml kasus** | **yes** | **no** | **entropy** | **gain** |
|---|---|---|---|---|---|---|
| **total** | | 14 | 10 | 4 | 0,863 | |
| **outlook** | | | | | | |
| | cloudy | 4 | 4 | 0 | 0 | |
| | rainy | 5 | 4 | 1 | 0,722 | |
| | sunny | 5 | 2 | 3 | 0,970 | |
| **temperature** | | | | | | |
| | cool | 4 | 0 | 4 | 0 | |
| | hot | 4 | 2 | 2 | 1 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | mild | 6 | 2 | 4 | 0,918 | |
| **humidity** | | | | | | |
| | high | 7 | 4 | 3 | 0,985 | |
| | normal | 7 | 7 | 0 | 0 | |
| **windy** | | | | | | |
| | false | 8 | 2 | 6 | 0,811 | |
| | true | 6 | 4 | 2 | 0,918 | |

Kemudian kita hitung information gain untuk setiap atribut:

Gain(total, outlook) = Entropi(Total) - $\sum_{i=1}^{n} \frac{|outlook_i|}{|total|} *$ Entropi(outlook)

Gain(total, outlook) = 0,863 – (4/14*0 + 5/14*0,722 + 5/14*0,970) = 0.259

Gain(total, temperature) = Entropi(Total) - $\sum_{i=1}^{n} \frac{|temperature_i|}{|total|} *$ Entropi(temperature)

Gain(total, temperature) = 0,863 – (4/14*0 + 4/14*1 + 6/14*0,918) = 0.184

Gain(total, humidity) = Entropi(Total) - $\sum_{i=1}^{n} \frac{|humidity_i|}{|total|} *$ Entropi(humidity)

Gain(total, humidity) = 0,863 – (7/14*0,985 + 5/14*0) = 0.370

Gain(total, windy) = Entropi(Total) - $\sum_{i=1}^{n} \frac{|windy_i|}{|total|} *$ Entropi(windy)

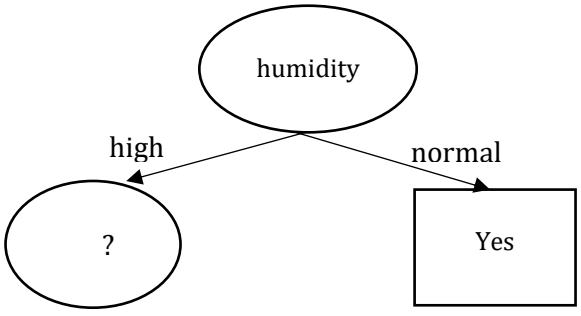
Gain(total, windy) = 0,863 – (8/14*0,811 + 6/14*0,918) = 0.006

Kemudian pindahkan perhitungan ke tabel

| **Atribut** | | **jml kasus** | **yes** | **no** | **entropy** | **gain** |
|---|---|---|---|---|---|---|
| **total** | | 14 | 10 | 4 | 0,863 | |
| **outlook** | | | | | | 0,259 |
| | cloudy | 4 | 4 | 0 | 0 | |
| | rainy | 5 | 4 | 1 | 0,722 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | sunny | 5 | 2 | 3 | 0,970 | |
| **temperature** | | | | | | 0,184 |
| | cool | 4 | 0 | 4 | 0 | |
| | hot | 4 | 2 | 2 | 1 | |
| | mild | 6 | 2 | 4 | 0,918 | |
| **humidity** | | | | | | 0,370 |
| | high | 7 | 4 | 3 | 0,985 | |
| | normal | 7 | 7 | 0 | 0 | |
| **windy** | | | | | | 0,006 |
| | false | 8 | 2 | 6 | 0,811 | |
| | true | 6 | 4 | 2 | 0,918 | |

Dari table di atas terlihat bahwa atribut dengan information gain tertinggi adalah humidity sebesar 0,370. Dengan demikian humidity dapat menjadi root node. Terdapat 2 nilai dari atribut humidity yaitu high dan normal. Dari kedua nilai atribut tersebut, nilai atribut humidity = normal sudah mengklasifikasikan data menjadi satu keputusan yaitu yes, sehingga tidak perlu dilakukan perhitungan lebih lanjut lagi. Tetapi untuk nilai atribut high masih perlu dilakukan perhitungan lebih lanjut.




Setelah terbentuk humidity sebagai root pohon, kita filter dataset, ambil baris yang humidity = high saja. Dari 14 baris yang ada menjadi tinggal 7 baris seperti ditunjukkan pada table di bawah ini:

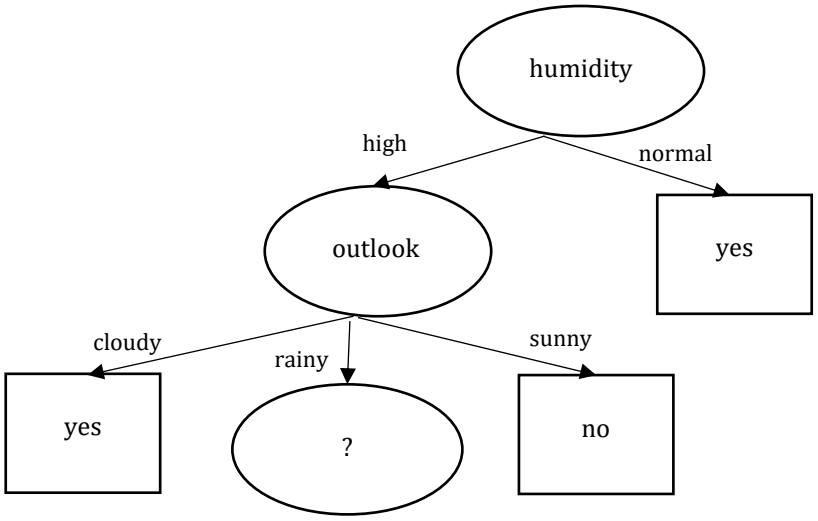
| no | Outlook | Temp | humidity | windy | play |
|---|---|---|---|---|---|
| 1 | Sunny | hot | high | false | no |

| | | | | | |
|---|---|---|---|---|---|
| 2 | Sunny | hot | high | true | no |
| 3 | overcast | hot | high | false | yes |
| 4 | Rainy | mild | high | false | yes |
| 5 | Sunny | mild | high | false | no |
| 6 | overcast | mild | high | true | yes |
| 7 | rainy | mild | high | true | no |

Kemudian kita ubah dalam format table berikut dan hitung entropy dan information gain untuk sisa atribut dengan cara yang sama seperti di atas:

| **atribut** | | **jml kasus** | **yes** | **no** | **entropy** | **gain** |
|---|---|---|---|---|---|---|
| **Humidity high** | | 7 | 3 | 4 | 0,98523 | |
| **outlook** | | | | | | 0,69951 |
| | cloudy | 2 | 2 | 0 | 0 | |
| | rainy | 2 | 1 | 1 | 1 | |
| | sunny | 3 | 0 | 3 | 0 | |
| **temperature** | | | | | | 0,02024 |
| | cool | 0 | 0 | 0 | 0 | |
| | hot | 3 | 1 | 2 | 0,91830 | |
| | mild | 4 | 2 | 2 | 1 | |
| **windy** | | | | | | 0,02024 |
| | false | 4 | 2 | 2 | 1 | |
| | true | 3 | 1 | 2 | 0,91830 | |

Dari table di atas terlihat bahwa atribut berikutnya yang memiliki information gain tertinggi adalah atribut outlook. Maka atribut outlook menjadi node berikutnya dari pohon yang kita buat. Untuk nilai outlook=cloudy sudah mengklasifikasikan data menjadi keputusan play=yes dan untuk nilai outlook=sunny juga sudah mengklasifikasikan data menjadi keputusan play=no. tinggal untuk outlook=rainy saja yang belum.



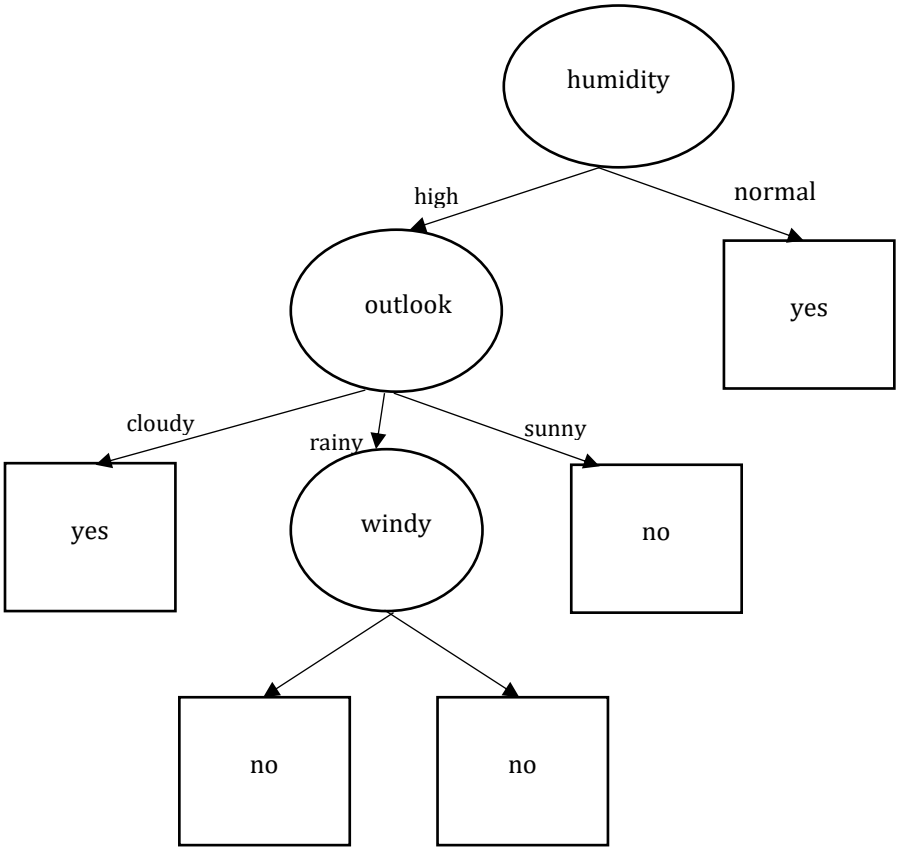

Langkah berikutnya kita filter kembali dataset, ambil untuk baris dimana outlook=rainy

| no | outlook | Temp | humidity | windy | play |
|---|---|---|---|---|---|
| 1 | rainy | mild | high | false | yes |
| 2 | rainy | mild | high | true | no |

Kemudian kita ubah dalam format table berikut dan hitung entropy dan information gain untuk sisa atribut dengan cara yang sama seperti di atas:

| **atribut** | | **jml kasus** | **yes** | **no** | **entropy** | **gain** |
|---|---|---|---|---|---|---|
| **Humidity = high & outlook = rainy** | | 2 | 1 | 1 | 1 | |
| **temperature** | | | | | | 0 |
| | cool | 0 | 0 | 0 | 0 | |
| | hot | 0 | 0 | 0 | 0 | |
| | mild | 2 | 1 | 1 | 1 | |
| **windy** | | | | | | 1 |
| | false | 1 | 1 | 0 | 0 | |
| | true | 1 | 0 | 1 | 0 | |

Dari table di atas terlihat bahwa atribut berikutnya yang memiliki information gain tertinggi adalah atribut windy. Maka atribut windy menjadi node berikutnya dari pohon yang kita buat. Untuk nilai windy=false sudah mengklasifikasikan data menjadi keputusan play=yes dan untuk nilai windy=true juga sudah mengklasifikasikan data menjadi keputusan play=no. karena semua data sudah habis diklasifikasikan, maka pohon keputusan akhir menjadi seperti gambar di bawah ini:




Setelah model pohon keputusan jadi, misal jika ada persoalan jika outlook=sunny, temperature=cool, humidity=high, dan windy=true bagaimanakah keputusannya apakah play = yes atau no?

Menggunakan pohon keputusan di atas, kita cek dulu humidity. Karena humidity = high, kemudian kita cek outlook. Karena outlook=sunny, maka keputusan play=no.

**Kelebihan Decision Tree**

- Tree yang dihasilkan mudah diinterpretasikan dalam artian model pohon yang dihasilkan dapat disajikan dalam bentuk yang mudah dipahami oleh manusia. Algoritma klasifikasi lain seperti Artificial Neural Network (JST), Support Vector Machine (SVM), maupun KNN model yang dihasilkan sulit untuk diinterpretasikan.
- Mampu merangking atribut yang paling baik sebagai predictor
- Setelah model pohon terbentuk, proses komputasi cepat untuk mengklasifikasikan data baru menggunakan model.
- Dapat menangani data numerik dan data kategorik dengan baik. Data numerik akan di-diskrit-kan dalam beberapa algoritma decision tree

**Kekurangan Decision Tree**

- Semakin banyak atribut, semakin kompleks model pohon yang dihasilkan.
- Model pohon yang terlalu kompleks (bercabang banyak) dapat dipangkas/pruned baik menggunakan pre pruning maupun post pruning.
- Cenderung memiliki error rate yang tinggi jika kelas tidak berimbang (imbalanced)

**Implementasi menggunakan R**

Untuk membuat decision tree salah satunya kita bisa menggunakan package party. Jika belum tersedia package party di R maka dapat kita install dengan perintah `install.packages("party")`. Dalam package "party" terdapat fungsi **ctree** yang bisa digunakan untuk membuat dan menganalisis tree.

Sintaks dasar untuk membuat decision tree di R menggunakan fungsi **ctree** adalah:

```
ctree(formula, data)
```

dengan:

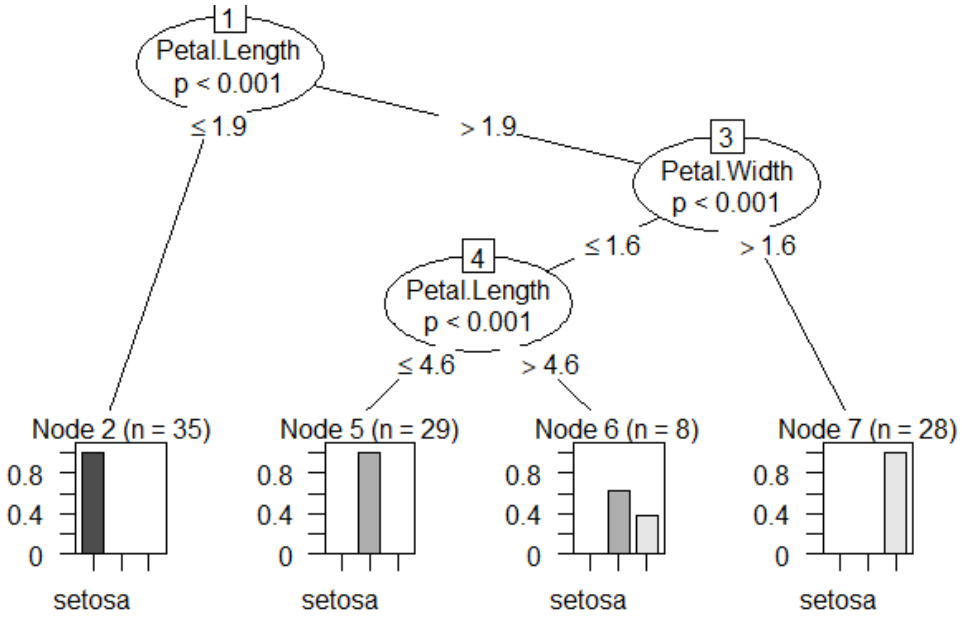
`formula` adalah formula yang menjelaskan variable respon dan variable prediktor.

`data` adalah nama dataset yang digunakan.

Contoh 3 : *decision tree* menggunakan *package* party

Misal kita akan mengunakan dataset iris yang telah tersedia di R dan menggunakan decision tree untuk melakukan klasifikasi jenis bunga iris tersebut.

```
      # Load package party.
library(party)

      Data <- iris

# membagi data training dan data testing
Sample <- sample(1:150, 50)
testing <- Data[Sample, ]
learning <- Data[-Sample, ]

output.tree <- ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Pe
tal.Width,
    data = learning)

# menampilkan model tree yang terbentuk
plot(output.tree)
```




Kemudian dari tree yang dihasilkan oleh data training, kita uji pada data testing sebagai berikut:

```
      # hitung akurasi prediksi
CM <- table(testing[, 5], Prediksi)
CM

                Prediksi
            setosa versicolor virginica
```

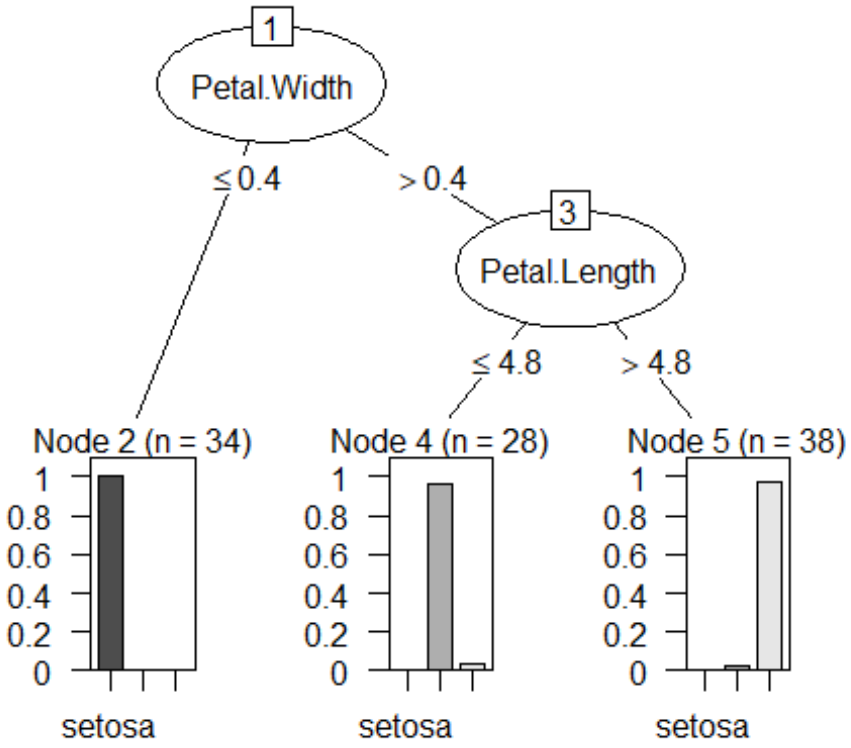
```
  setosa         20          0          0
  versicolor      0          9          1
  virginica       0          4         16

      accuracy <- (sum(diag(CM)))/sum(CM)
accuracy

      [1] 0.9
```

Selain menggunakan *package* party, kita juga bisa menggunakan *package* RWeka. Pada package RWeka terdapat fungsi **J48** yang bisa kita gunakan untuk membuat *decision tree* berbasis *gain ratio*. Jika belum terdapat *package* RWeka maka dapat kita install menggunakan fungsi `install.packages("RWeka")`. Kemudian untuk menampilkan tree hasil **J48** kita juga perlu *package* partykit. Jika belum terdapat *package* partykit maka dapat kita install menggunakan fungsi `install.packages("partykit")`.

Contoh 4 (*decision tree* menggunakan *package* Rweka):

Masih dengan menggunakan data set *iris*, kita akan membuat suatu decision tree untuk mengklasifikasikan sub spesies bunga *Iris*.

```
      library(rJava)
      library(RWeka)
library(partykit)

      Data = iris
Sample <- sample(1:150, 50)
testing <- Data[Sample, ]
learning <- Data[-Sample, ]

output.tree <- J48(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Peta
l.Width, data = learning)

# menampilkan model tree yang terbentuk
plot(output.tree)
```




Kemudian setelah mendapatkan model *classifier* dari data training, berikutnya kita uji dengan data testing berikut.

```
      # pengujian dengan data testing
Prediksi = predict(output.tree, testing)
Prediksi

       [1] setosa     setosa     versicolor setosa     setosa     setosa
 [7] setosa     setosa     setosa     versicolor virginica  versicolor
[13] virginica  setosa     setosa     virginica  versicolor setosa
[19] setosa     versicolor virginica  setosa     setosa     versicolor
[25] virginica  versicolor setosa     setosa     versicolor versicolor
[31] setosa     virginica  virginica  virginica  setosa     versicolor
[37] setosa     virginica  virginica  setosa     setosa     versicolor
[43] virginica  versicolor virginica  versicolor versicolor setosa
[49] virginica  virginica
Levels: setosa versicolor virginica

      #Confusion Matrix
CM <- table(testing[, 5], Prediksi)
CM

                Prediksi
            setosa versicolor virginica
  setosa         22          1         0
```

```
  versicolor      0         12         1
  virginica       0          1        13

      #Akurasi
accuracy <- (sum(diag(CM)))/sum(CM)
accuracy

      [1] 0.94
```

## 4. Naïve Bayes

*Naïve bayes* merupakan metode klasifikasi berbasis probabilita/peluang. Metode ini menghitung sekumpulan probabilitas dengan menjumlahkan frekuensi dan kombinasi nilai dari dataset yang diberikan. Metode ini menggunakan Teorema Bayes dan mengasumsikan semua atribut independen atau tidak saling ketergantungan yang diberikan oleh nilai pada variabel kelas.

Aturan/Teorema Bayes adalah sebagai berikut:

$$P(H|X) = \frac{P(X|H) \cdot P(H)}{P(X)}$$

dimana:

$X$ : Data dengan kelas yang belum diketahui

$H$ : Hipotesis data merupakan suatu kelas spesifik

$P(H|X)$ : Probabilitas hipotesis $H$ berdasar kondisi $X$ (probabilitas posterior)

$P(H)$ : Probabilitas hipotesis $H$ (probabilitas prior)

$P(X|H)$ : Probabilitas $X$ berdasarkan kondisi pada hipotesis $H$

$P(X)$ : Probabilitas $X$

Terdapat dua asumsi pada metode **Naive Bayes** :

- semua atribut adalah prioritas/sama pentingnya
- semua atribut bersifat independen secara statistik (nilai satu atribut tidak terkait dengan nilai atribut lain)

Asumsi ini kebanyakan tidak selalu benar, namun dalam praktiknya walaupun asumsi tidak terpenuhi, metode ini tetap memberikan hasil yang baik. Karena dalam **Naive Bayes** diasumsikan bahwa semua atribut saling independen, maka persamaannya dapat berubah menjadi:

$$P(C|F_1 \dots F_n) = \frac{P(C) \cdot P(F_1 \dots F_n|C)}{P(F_1 \dots F_n)}$$

dimana:

$C$ : Kelas

$F_1 \dots F_n$ : Karakteristik petunjuk yang dibutuhkan untuk melakukan klasifikasi

$P(C|F1 \dots Fn)$ : Probabilitas kelas C ketika kondisi karakteristik $F_1$ ... $F_n$

$P(C)$ : Probabilitas kelas C (probabilitas prior)

$P(F1 \dots Fn|C)$ : Probabilitas karakteristik $F_1$ ... $F_n$ ketika kondisi kelas C

$P(F1 \dots Fn)$ : Probabilitas karakteristik $F_1$ ... $F_n$

Contoh 5 : Ilustrasi Perhitungan Sederhana

Misal terdapat suatu *training set* sebagai berikut:

| Outlook | Temp. | Humidity | Windy | Play |
|---|---|---|---|---|
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| overcast | hot | high | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| overcast | cool | normal | true | yes |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| rainy | mild | normal | false | yes |
| sunny | mild | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | true | no |

Kemudian dari *trainingset* tersebut akan digunakan untuk menjawab persoalan: Menggunakan metode naïve bayes, tentukan kelas *play* (yes atau no?) jika diketahui *outlook=sunny, temperature = cool, humidity = high*, dan *windy = true*.

Pertama, hitung dahulu semua kejadian dalam tabel frekuensi sebagai berikut:

| | Play | |
|---|---|---|
| Outlook | yes | no |
| sunny | 2 | 3 |
| overcast | 4 | 0 |
| rainy | 3 | 2 |
| TOTAL | 9 | 5 |

| Outlook | Temp. | Humidity | Windy | Play |
|---|---|---|---|---|
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| overcast | hot | high | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| overcast | cool | normal | true | yes |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| rainy | mild | normal | false | yes |
| sunny | mild | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | true | no |

| | Play | | | Play | | | Play | | | Play | | | Play |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Outlook | yes | no | Temp. | yes | no | Humid. | yes | no | Windy | yes | no | | |
| sunny | 2 | 3 | hot | 2 | 2 | high | 3 | 4 | false | 6 | 2 | yes | 9 |
| overcast | 4 | 0 | mild | 4 | 2 | normal | 6 | 1 | true | 3 | 3 | no | 5 |
| rainy | 3 | 2 | cool | 3 | 1 | | | | | | | | |
| TOTAL | 9 | 5 | TOTAL | 9 | 5 | TOTAL | 9 | 5 | TOTAL | 9 | 5 | TOTAL | 14 |

Kemudian ubah setiap frekuensi menjadi bentuk probabilitas/peluang

| | Play | | | Play | | | Play | | | Play | | | Play |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Outlook | yes | no | Temp. | yes | no | Humid. | yes | no | Windy | yes | no | | |
| sunny | 2 | 3 | hot | 2 | 2 | high | 3 | 4 | false | 6 | 2 | yes | 9 |
| overcast | 4 | 0 | mild | 4 | 2 | normal | 6 | 1 | true | 3 | 3 | no | 5 |
| rainy | 3 | 2 | cool | 3 | 1 | | | | | | | | |
| TOTAL | 9 | 5 | TOTAL | 9 | 5 | TOTAL | 9 | 5 | TOTAL | 9 | 5 | TOTAL | 14 |

| | Play | | | Play | | | Play | | | Play | | | Play |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Outlook | yes | no | Temp. | yes | no | Humid. | yes | no | Windy | yes | no | | |
| sunny | 0.22 | 0.60 | hot | 0.22 | 0.40 | high | 0.33 | 0.80 | false | 0.67 | 0.40 | yes | 0.64 |
| overcast | 0.44 | 0.00 | mild | 0.44 | 0.40 | normal | 0.67 | 0.20 | true | 0.33 | 0.60 | no | 0.36 |
| rainy | 0.33 | 0.40 | cool | 0.33 | 0.20 | | | | | | | | |

2 kejadian dari **Outlook = rainy** dan **Play = no**

5 kejadian dari **Play = no**

Kemudian sesuai persoalan, hitung *likelihood play=yes* ketika *outlook=sunny, temperature = cool, humidity = high,* dan *windy = true*

| | Play | | | Play | | | Play | | | Play | | | Play |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Outlook | yes | no | Temp. | yes | no | Humid. | yes | no | Windy | yes | no | | |
| sunny | 0.22 | 0.60 | hot | 0.22 | 0.40 | high | 0.33 | 0.80 | false | 0.67 | 0.40 | yes | 0.64 |
| overcast | 0.44 | 0.00 | mild | 0.44 | 0.40 | normal | 0.67 | 0.20 | true | 0.33 | 0.60 | no | 0.36 |
| rainy | 0.33 | 0.40 | cool | 0.33 | 0.20 | | | | | | | | |

*Likelihood play=yes* dihitung dengan mengalikan semua atribut *outlook sunny=yes, temperature cool=yes, humidity high = yes, windy true = yes, dan play = yes*.

*Likelihood yes given attributes* = 0.22 x 0.33 x 0.33 x 0.33 x 0.64 = 0.0053

Dengan cara yang sama, hitung *likelihood play=no* ketika *outlook=sunny, temperature = cool, humidity = high,* dan *windy = true*

| | Play | | | Play | | | Play | | | Play | | | Play |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Outlook | yes | no | Temp. | yes | no | Humid. | yes | no | Windy | yes | no | | |
| sunny | 0.22 | 0.60 | hot | 0.22 | 0.40 | high | 0.33 | 0.80 | false | 0.67 | 0.40 | yes | 0.64 |
| overcast | 0.44 | 0.00 | mild | 0.44 | 0.40 | normal | 0.67 | 0.20 | true | 0.33 | 0.60 | no | 0.36 |
| rainy | 0.33 | 0.40 | cool | 0.33 | 0.20 | | | | | | | | |

*Likelihood play=no* dihitung dengan mengalikan semua atribut *outlook sunny = no, temperature cool = no, humidity high = no, windy true = no,* dan *play = no*

*Likelihood no given attributes* = 0,60 x 0,20 x 0,80 x 0,60 x 0,36 = 0,0206

Kemudian masukkan perhitungan tersebut ke dalam rumus:

$$P(C|F_1 \dots F_n) = \frac{P(C) \cdot \frac{P(F1 \cap C)}{P(C)} \cdot \dots \cdot \frac{P(Fn \cap C)}{P(C)}}{P(F_1 \dots F_n)}$$

Peluang untuk *Play=yes*: $\frac{0.0053}{0.0053 + 0.0206}$ = 20,5%

Peluang untuk *Play=no* : $\frac{0.0206}{0.0053 + 0.0206}$ = 79,5%

Karena Peluang untuk *play = no* lebih besar dari peluang untuk *play = yes*, maka untuk persoalan *outlook=sunny, temperature = cool, humidity = high, dan windy = true, play = no*.

**Implementasi dengan R**

Implementasi algoritma naïve bayes pada R dapat dilakukan dengan menggunakan fungsi *naiveBayes()* dari package e1071. Jika pada R yang digunakan belum tersedia package ini, maka package ini perlu diinstall terlebih dahulu dengan menggunakan fungsi:

```
install.packages("e1071")
```

Sintaks fungsi *naiveBayes()* adalah sebagai berikut:

```
naiveBayes(formula, training_data)
```

Keterangan:

- formula, formula untuk menentukan target variable.

- training_data, obyek data training.

Fungsi di atas digunakan untuk membuat model. Sedangkan untuk melakukan prediksi

digunakan fungsi *predict().* Berikut ini adalah sintaks fungsi *predict():*

```
predict(model, testing_data)
```

Keterangan:

- `model` adalah obyek output dari fungsi *naiveBayes().*

- `testing_data` adalah obyek *test set*.

Contoh 6 : Implementasi dengan R

Dari kasus pada contoh 5, berikut ini contoh pengerjaan dengan menggunakan R:

```
library(e1071)
# input data training
data.training = as.data.frame(rbind(
```

```
      c("Sunny", "Hot", "High", "False", "No"),
      c("Sunny", "Hot", "High", "True", "No"),
      c("Overcast", "Hot", "High", "False", "Yes"),
      c("Rainy", "Mild", "High", "False", "Yes"),
      c("Rainy", "Cool", "Normal", "False", "Yes"),
      c("Rainy", "Cool", "Normal", "True", "No"),
      c("Overcast", "Cool", "Normal", "True", "Yes"),
      c("Sunny", "Mild", "High", "False", "No"),
      c("Sunny", "Cool", "Normal", "False", "Yes"),
      c("Rainy", "Mild", "Normal", "False", "Yes"),
      c("Sunny", "Mild", "Normal", "True", "Yes"),
      c("Overcast", "Mild", "High", "True", "Yes"),
      c("Overcast", "Hot", "Normal", "False", "Yes"),
      c("Rainy", "Mild", "High", "True", "No")))
# memberi nama kolom
names(data.training)[1] = "OUTLOOK"
names(data.training)[2] = "TEMP"
names(data.training)[3] = "HUMIDITY"
names(data.training)[4] = "WINDY"
names(data.training)[5] = "PLAY"

# input data testing
data.test = as.data.frame(cbind("Sunny", "Cool", "High", "True"))
names(data.test)[1] = "OUTLOOK"
names(data.test)[2] = "TEMP"
names(data.test)[3] = "HUMIDITY"
names(data.test)[4] = "WINDY"
# membuat model
model <- naiveBayes(PLAY ~ ., data = data.training)
print(model)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
       No       Yes
0.3571429 0.6428571

Conditional probabilities:
     OUTLOOK
Y      Overcast     Rainy     Sunny
  No  0.0000000 0.4000000 0.6000000
  Yes 0.4444444 0.3333333 0.2222222

     TEMP
Y          Cool       Hot      Mild
  No  0.2000000 0.4000000 0.4000000
```

```
  Yes 0.3333333 0.2222222 0.4444444

     HUMIDITY
Y          High    Normal
  No  0.8000000 0.2000000
  Yes 0.3333333 0.6666667

     WINDY
Y         False      True
  No  0.4000000 0.6000000
  Yes 0.6666667 0.3333333
```

```
# melakukan klasifikasi data testing (prediksi)
predict_result <- predict(model, data.test)
print(predict_result)
```

```
[1] No
Levels: No Yes
```

### 5. *Predicition evaluation*, *cross validation*, dan *ROC curve*

Untuk menilai seberapa baik classifier yang dihasilkan, kita dapat membagi dataset yang tersedia menjadi dua yaitu training set dan test set. Training set digunakan untuk membuat model dan test set digunakan untuk menilai seberapa baik classifier yang dihasilkan.

Jika label suatu baris dalam test set sama dengan hasil klasifikasi oleh model yang dihasilkan, maka hal ini disebut sebagai correct classification. Jika label suatu baris dalam test set berbeda dengan hasil klasifikasi oleh model yang dihasilkan, maka hal ini disebut dengan misclassification.

Semakin banyak jumlah correct classification, semakin tinggi akurasi dari sebuah classifier, dan sebaliknya semakin banyak jumlah misclassification, semakin rendah akurasi dari sebuah classifier. Akurasi hanyalah salah satu metric yang menentukan bagus/tidaknya sebuah classifier. Metric yang lainnya akan dijelaskan pada pembahasan berikutnya.

Untuk membagi dataset sebagai training set dan test set, terdapat beberapa metode yang dapat digunakan seperti hold out method, cross validation, dan bootstrap.

#### Hold out method

Pada hold out method, data dibagi secara random menjadi training set dan test set. Missal 2/3 bagian menjadi training set untuk membangun model, sedangkan 1/3 bagian digunakan untuk mengukur akurasi model. Metode holdout dapat juga divariasikan menggunakan random sampling dimana proses holdout diulang sebanyak k kali. Akurasi yang digunakan adalah akurasi rata-rata dari seluruh k proses.

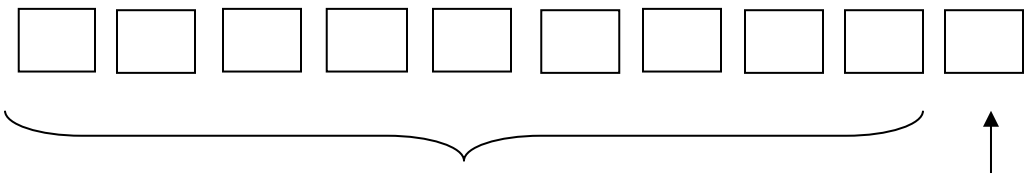
**Cross validation**

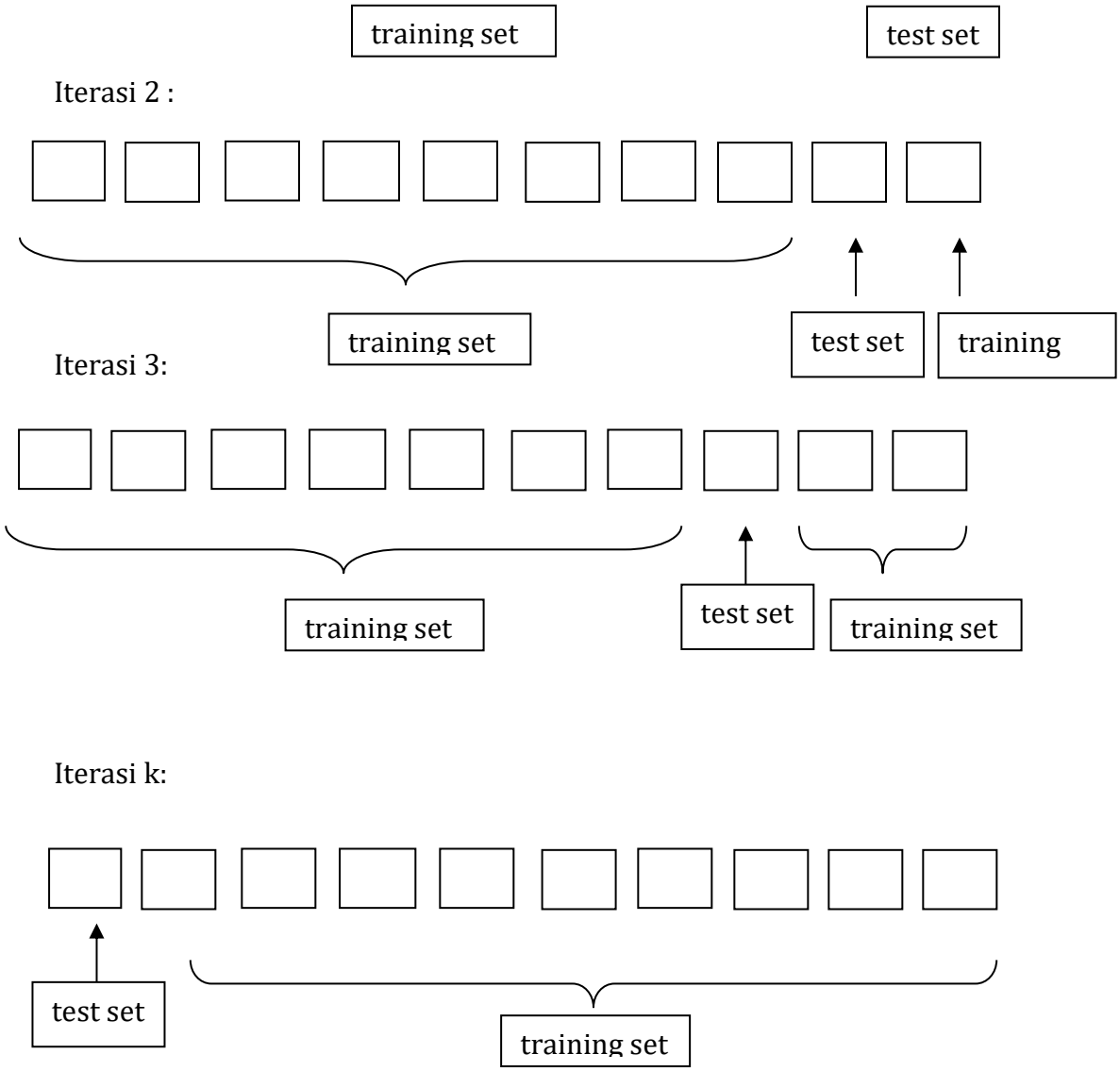
– ***K-fold cross validation***

*K-fold-cross validation* merupakan metode untuk melakukan pengujian dan perbaikan (validasi) terhadap suatu *classifier* yang sedang dibangun. Saat melakukan training untuk membuat suatu model *classifier*, umumnya data dibagi atas *training set* sebanyak 80% dan *test set* sebanyak 20%. Namun muncul pertanyaan apakah 80% data yang digunakan sudah merupakan training set terbaik dan apakah 20% data yang dijadikan validation set sudah merupakan data yang hasil pengujiannya dapat mewakili akurasi dari *classifier* tersebut? Untuk menjawab pertanyaan ini digunakan *k-fold-cross validation*, yaitu suatu metode dimana data dibagi secara random menjadi beberapa (*k*) bagian kemudian dilakukan training terhadap *classifier* dengan menggunakan beberapa bagian dan tes dilakukan dengan bagian yang lain.

Jumlah *k* yang biasa digunakan biasanya adalah 10 (10-*fold-cross validation*). Pada 10-*fold-cross validation* dataset dibagi menjadi 10 bagian ($S_1, S_2, S_3, \ldots, S_{10}$). Pada iterasi pertama dilakukan pelatihan dengan menggunakan $S_1$ sampai dengan $S_9$, classifier hasil pelatihan ini kemudian diuji dengan menggunakan $S_{10}$. Pada iterasi ke-2 pelatihan dilakukan dengan menggunakan $S_1$ sampai dengan $S_8$ ditambah dengan $S_{10}$, classifier hasil pelatihan kemudian diuji dengan menggunakan $S_9$. Secara singkat metode ini digembarkan pada Gambar berikut:

Iterasi 1 :






Akurasi akhir adalah akurasi rata-rata dari sejumlah k proses

– **Leave-one-out cross validation**

Mirip seperti k-fold-cross validation, leave-one-out cross validation membangun model dari seluruh dataset dikurangi 1 baris untuk test set. Misal untuk dataset berjumlah 100 baris, maka leave-one-out cross validation akan membuat model dari 99 baris dan sisanya 1 baris digunakan sebagai test set. Iterasi akan diulang hingga seluruh baris telah mendapat giliran sebagai test set. Jadi iterasi akan berlangsung sebanyak jumlah baris pada

dataset. Leave-one-out cross validation biasanya dilakukan pada dataset yang jumlah barisnya sedikit karena banyaknya jumlah iterasi. Akurasi adalah berapa persen seluruh iterasi terjadi correct classification.

- **Stratified cross-validation**

Pada stratified cross-validation, pembentukan fold dilakukan secara stratifikasi sehingga distribusi kelas pada setiap fold kurang lebih sama dengan dataset total.

**Bootstrap**

Pada metode Bootsrap, dilakukan sampling with replacement terhadap training set dimana setiap kali sebuauh baris terpilih sebagai sampel, maka baris tersebut dapat terpilih kembali pada proses sampling berikutnya dan ditambahkan pada training set. Metode bootstrap bekerja baik pada dataset yang jumlah barisnya sedikit. Salah satu metode bootstrap yang umum dilakukan adalah .632 bootstrap dengan ccara sebagai berikut:

1. Pada suatu dataset yang berisi sejumlah d baris dilakukan sampling dengan pengembalian sebanyak d kali, yang menghasilkan training set sejumlah d baris.
2. Data yang tidak terambil sebagai training set dijadikan sebagain test set. Hasilnya sekitar 63,2% dataset awal akan masuk di bootstrap, dan sisanya 36,8% menjadi test set. (karena $(1 - 1/d)^d \approx e^{-1} = 0{,}368$).
3. Ulangi prosedur sampling sebanyak k kali, akurasi dari model adalah:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0{,}632 * Acc(M_i)_{test_{set}} + 0{,}368 * Acc(M_i)_{train\_set})$$

**Confusion Matrix**

Salah satu alat bantu untuk menilai seberapa baik sebuah classifier adalah confusion matrix. Tabel confusion matrix dihasilkan dari aplikasi model pada test set. Dari confusion matriks dapat diturunkan berbagai metric evaluasi classifier seperti akurasi, specificity,

sensitivity, dan lain-lain. Table confusion matriks merupakan table binary, dimana kelas dibagi menjadi 2 label kelas saja. Jika terdapat multiclass, misalnya 3 kelas (A,B,C) maka confusion matrix dibagi 3 tabel menjadi antara kelas A dan bukan kelas A, antara kelas B dan bukan kelas B,dan antara kelas C dan bukan kelas C. Berikut adalah format umum dari confusion matrix:

| Actual Class\Predicted Class | C1 | ¬C1 |
|---|---|---|
| C1 | True Positives (TP) | False Negatives(FN) |
| ¬C1 | False Positive (FP) | True Negative(TN) |

Actual Class adalah kelas yang sebenarnya pada test set. Predicted class adalah kelas hasil prediksi dari model yang dihasilkan oleh classifier. True Positive (TP) adalah jumlah baris berlabel kelas C1 pada test set yang benar diklasifikasikan sebagai kelas C1 oleh classifier. False Negative (FN) adalah jumlah baris berlabel C1 pada tes tes namun diklasifikasikan sebagai kelas bukan C1 oleh classifier. False Positive (FP) adalah jumlah baris berlabel kelas bukan C1 pada test set, namun diklasifikasikan sebagai kelas C1 oleh classifier. True Negative (TN) adalah jumlah baris berlabel kelas bukan C1 pada test set dan benar diklasifikasikan sebagai kelas bukan C1 oleh classifier.

Confusion matriks dapat memiliki baris dan kolom tambahan untuk menjumlahkan total sehingga tabel confusion matrix menjadi sebagai berikut:

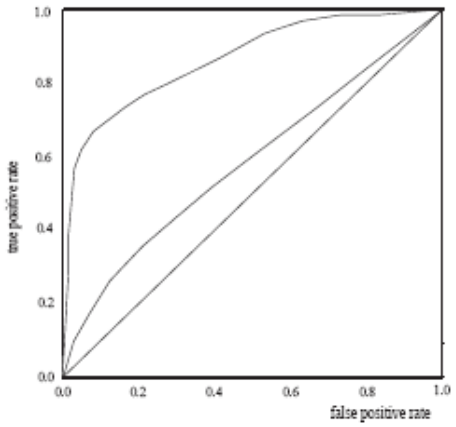
| A\P | C | ¬C | |
|---|---|---|---|
| C | **TP** | **FN** | **P** |
| ¬C | **FP** | **TN** | **N** |

| | P' | N' | All |
|---|---|---|---|

- Akurasi adalah persentase baris test set yang diklasifikasikan dengan benar. Accuracy = (TP+TN)/All
- Error rate adalah 1 – accuracy, atau *error rate = (FP + FN)/All*
- Sensitivity: True Positive recognition rate. *Sensitivity = TP/P*
- Specificity: True Negative recognition rate. *Specificity = TN/N*
- Precision: exactness – berapa% baris yang dilabel positif oleh classifier memang benar positif. *Precision = TP/ (TP+FP)*
- Recall: completeness – berapa % baris yang berlabel positif oleh classifier dilabel positif. *Recall = TP/ (TP+FN)*. Nilai sempurna dari precision dan recall adalah 1.
- F Measure (F1 atau F-score): rata-rata harmonis dari precision dan recall. *F = (2 * precision * recall) / (precision + recall)*
- Fβ : ukuran tertimbang dari precision dan recall: memberikan weight sebanyak β kali untuk recall terhadap precision $F\beta = ((1 + \beta^2) * precision * recall)/(\beta^2 * precision * recall)$

**ROC Curve**

ROC( Receiver Operating Characteristics) Curves atau kurva ROC digunakan sebagai perbandingan visual untuk model-model klasifikasi. Penggunaan ROC curve berawal dari tori deteksi  sinyal yang menunjukkan trade-off antara true positive rate dan false positive rate.




Sumbu vertical mewakili true positive rate dan sumbu horizontal mewakili false positive rate dan terdapat pula garis diagonal. ROC curve mengurutkan baris pada test set dari positif ke negative. Baris-baris yang kelihatannya termasuk kelas positif berada pada bagian atas. Luas area di bawah ROC curve adalah ukuran akurasi model. Model klasifikasi dengan akurasi sempurna akan memiliki luas area 1,0. Semakin mendekat kurva ke garis diagonal, semakin tidak akurat model klasifikasi.

Akurasi adalah metric paling umum yang digunakan untuk menilai seberapa bagus sebuah classifier. Namun jika hanya menggunakan akurasi saja, maka penilaian terhadap seberapa bagusnya classifier dapat menjadi menyesatkan. Sebagai contoh terdapat sebuah confusion matrix untuk classifier cancer sebagai berikut:

| Actual Class \ Predicted class | cancer = yes | cancer = no | Total | Recognition(%) |
|---|---|---|---|---|
| cancer = yes | **90** | **210** | 300 | 30.00 (*sensitivity)* |
| cancer = no | **140** | **9560** | 9700 | 98.56 (*specificity)* |
| Total | 230 | 9770 | 10000 | 96.40 (*accuracy*) |

Classifier di atas memang memiliki akurasi yang tinggi yaitu 96,40%. Ukuran specificity juga tinggi sebesar 98,56%. Namun classifier tersebut memiliki sensitivity yang rendah (30%). Nilai precision juga rendah yaitu 90/230 = 39,13% dan nilai recall juga

rendah yaitu 90/300 = 30%. Untuk kasus dengan data yang tidak seimbang seperti di atas dimana kejadian cancer=yes sangat jarang dibandingkan cancer=no, maka kita memerlukan metric yang lain selain accuracy. Nilai sensitivity 30% menandakan bahwa untuk 300 kejadian cancer yang seharusnya diberikan label yes,classifier ini hanya mampu mengklasifikasi benar sebanyak 90 kasus saja. Sensitifity merupakan metric yang juga penting untuk kasus imbalanced data dimana kita ingin kelas yang minoritas juga diklasifikasi dengan tingkat kebenaran yang tinggi.

**Implementasi dengan R**

Package **caret** di R merupakan salah satu package yang menyediakan berbagai metode untuk mengestimasi akurasi dari algoritma *machine learning*. Metode-metode tersebut antara lain *data split*, *k-fold cross validation*, *leave one-out cross validation*, dan bootstrap.

Sedangkan untuk membuat plot ROC, juga terdapat beberapa package yang bisa membuat plot ROC tersebut, antara lain package **plotROC**, **pROC**, dan package **verification**. Namun dalam contoh berikut ini, kita akan menggunakan package **pROC**.

Contoh 7.

Kita akan menggunakan data set german credit data yang diambil dari URL *https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)* dengan mengurangi jumlah atributnya. Dataset german credit merupakan data yang berisi penilaian kualitas kredit dari nasabah suatu bank, apakah baik (*good*) atau buruk (*bad*). Berikut ini kita akan melakukan *cross-validation* dengan menggunakan metode *k-fold*, *leave-one-out*, dan *bootstrap*.

```
# import data
german <- read.csv("german_credit.csv", header = TRUE)
head(german)
  Duration Amount Installment Residence Age ExistCard Maintenance
1        6   1169           4         4  67         2           1
2       48   5951           2         2  22         1           1
3       12   2096           2         3  49         1           2
4       42   7882           2         4  45         1           2
```

```
5       24   4870           3         4  53         2           2
6       36   9055           2         4  35         1           2
  Telephone ForeignWorker Class
1         0             1  Good
2         1             1   Bad
3         1             1  Good
4         1             1  Good
5         1             1   Bad
6         0             1  Good
```

```r
# jadikan variabel kategorik sebagai faktor
german$Telephone <- as.factor(german$Telephone)
german$ForeignWorker <- as.factor(german$ForeignWorker)
german$Class <- as.factor(german$Class)

# load library yang diperlukan
library(caret)
library(klaR)

# K-FOLD, K = 10
# definisikan training control
train_control <- trainControl(method = "cv", number = 10)
# train model
model <- train(Class ~ ., data = german, trControl = train_control,
                  method =    "rf")
# tampilkan hasil
print(model)
```

```
Random Forest

1000 samples
   9 predictor
   2 classes: 'Bad', 'Good'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
Resampling results across tuning parameters:

  mtry  Accuracy  Kappa
  2     0.727     0.1990964
  5     0.717     0.2414068
  9     0.714     0.2345496

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

```r
# LEAVE-ONE-OUT
# definisikan training control
train_control <- trainControl(method = "LOOCV")
# train model
model <- train(Class ~ ., data = german, trControl = train_control,
                  method = "rf")
# tampilkan hasil
```

```
print(model)
Random Forest

1000 samples
   9 predictor
   2 classes: 'Bad', 'Good'

No pre-processing
Resampling: Leave-One-Out Cross-Validation
Summary of sample sizes: 999, 999, 999, 999, 999, 999, ...
Resampling results across tuning parameters:

  mtry  Accuracy  Kappa
  2     0.734     0.2231308
  5     0.713     0.2209555
  9     0.708     0.2133621

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.

# BOOTSTRAP
# definisikan training control
train_control <- trainControl(method = "boot", number = 100)
# train model
model <- train(Class ~ ., data = german, trControl = train_control,
               method = "rf")
# tampilkan hasil
print(model)
Random Forest

1000 samples
   9 predictor
   2 classes: 'Bad', 'Good'

No pre-processing
Resampling: Bootstrapped (100 reps)
Summary of sample sizes: 1000, 1000, 1000, 1000, 1000, 1000, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  2     0.7164191  0.1758122
  5     0.7005435  0.1989379
  9     0.6915628  0.1887830

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

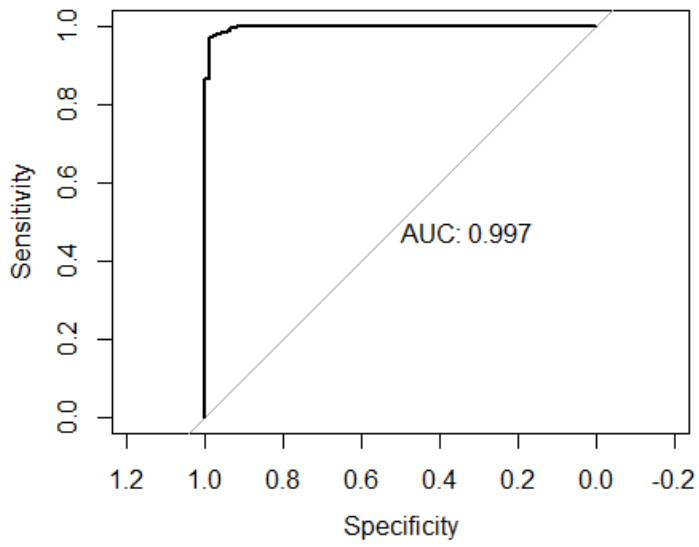
Dengan menggunakan ketiga metode *cross-validation* di atas, terlihat bahwa akurasi dari model tidak terlalu tinggi, tidak sampai 75%.

Berikut ini adalah contoh untuk membuat ROC pada data *german credit*, pada metode *cross-validation* data split.

```r
# PLOT ROC data train sebanyak 75%
inTrain <- createDataPartition(y = german$Class, p = 0.75, list = FALSE)
train <- german[inTrain, ]
test <- german[-inTrain, ]

# buat model random forest.
output.forest <- train(Class ~ ., data = german, method = "rf")

# prediksi probability data testing
prediksi <- predict(output.forest, test, type = "prob")

# load library
library(pROC)
```

```
Type 'citation("pROC")' for a citation.


Attaching package: 'pROC'

The following objects are masked from 'package:stats':

    cov, smooth, var
```

```r
german.roc <- roc(test$Class, prediksi$Bad)
plot.roc(german.roc, print.auc = TRUE)
```



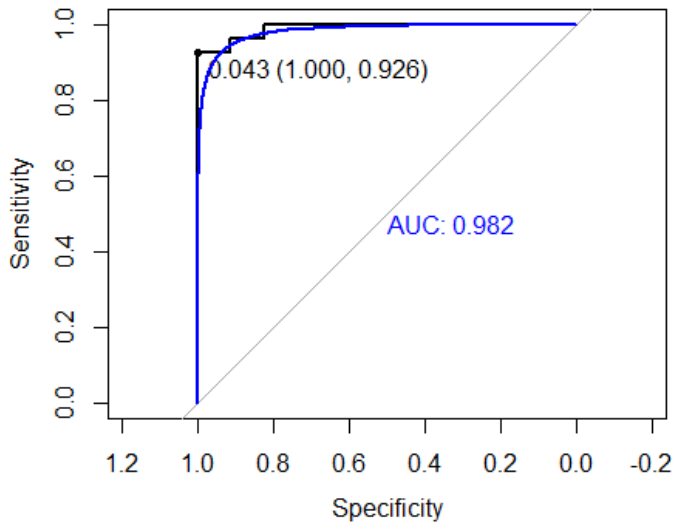

```r
# to get threshold and accuracy
result.coords <- coords(german.roc, "best", best.method = "closest.topleft",
    ret = c("threshold", "accuracy"))
print(result.coords)
```

```
threshold  accuracy
    0.304     0.976
```

Contoh 8

Pada contoh ini kita akan membuat plot ROC dengan menggunakan data set iris.

```
library(caret)
library(pROC)

data(iris)

# kelas setosa tidak dilibatkan dalam contoh ini karena ROC hanya untuk
# kelas dengan 2 level
iris <- iris[iris$Species == "virginica" | iris$Species == "versicolor", ]
iris$Species <- factor(iris$Species)

# split data train dan data test
samples <- sample(NROW(iris), NROW(iris) * 0.5)
data.train <- iris[samples, ]
data.test <- iris[-samples, ]

# buat model dengan random forrest
forest.model <- train(Species ~ ., data.train, method = "rf")

# prediksi data test
result.predicted.prob <- predict(forest.model, data.test, type = "prob")  # P
rediction

result.roc <- roc(data.test$Species, result.predicted.prob$versicolor)  # Dra
w ROC curve.
plot(result.roc, print.thres = "best", print.thres.best.method = "closest.top
left")
# tambahkan garis smoothed ROC
plot.roc(smooth(result.roc), add = TRUE, col = "blue", print.auc = TRUE)
```



```r
# to get threshold and accuracy
result.coords <- coords(result.roc, "best", best.method = "closest.topleft",
    ret = c("threshold", "accuracy"))
print(result.coords)
```

```
threshold  accuracy
    0.043     0.960
```