Lecture 9

Tree Based Models, SVM

Assoc. Prof. Dr. Burkay Genç

2024-04-30

Seed used in these slides

```
set.seed(1024)
```

Libraries used in these slides

```
library(rpart)
library(rpart.plot)
library(mlbench)
library(DMwR2)
library(e1071)
```

Tree Based Models

Properties

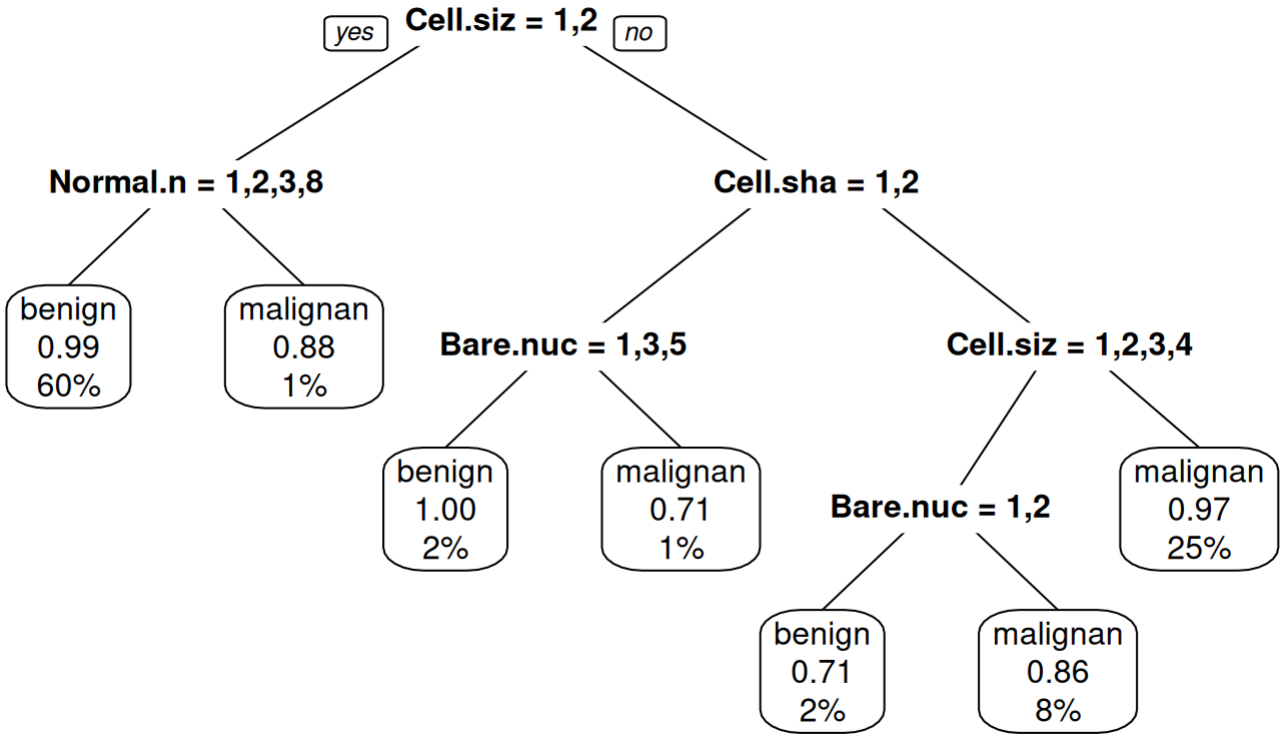
- **Interpretable** results
- Reasonable accuracy
- Applicable for both **classification and regression** tasks
- Works with both **numeric and categorical** variables
- **Can handle NAs**
- No assumption of the shape of the function
- Not top prediction performance
    - Ensembles of trees have much better performance

Shape

- A **hierarchy of logical tests** on variables
  - Is X > 5?
  - Is color = green?
  - Is birthplace in {Ankara, Istanbul, İzmir}?
- Each branch, including the root splits the data at hand into two
  - Decreasing the total error rate
- The leaves contain results / predictions
- The path to a leaf is a conjunction of logical tests

Example

```
##  [1] "Id"              "Cl.thickness"    "Cell.size"       "Cell.shape"
##  [5] "Marg.adhesion"   "Epith.c.size"    "Bare.nuclei"     "Bl.cromatin"
##  [9] "Normal.nucleoli" "Mitoses"         "Class"
```



Algorithm

Algorithm 1 The recursive partitioning algorithm.

1: **function** RECURSIVEPARTITIONING(D)
Input : D, a sample of cases, $\{\langle x_{i,1}, \cdots, x_{i,p}, y_i \rangle\}_{i=1}^{N_{train}}$
Output : t, a tree node

2: **if** <TERMINATION CRITERION> **then**
3: **Return** a leaf node with the <REPRESENTATIVE> of D
4: **else**
5: $t \leftarrow$ new tree node
6: $t.split \leftarrow$ <FIND THE BEST PREDICTORS TEST>
7: $t.leftNode \leftarrow$ RecursivePartitioning($\mathbf{x} \in D : \mathbf{x} \rightarrow t.split$)
8: $t.rightNode \leftarrow$ RecursivePartitioning($\mathbf{x} \in D : \mathbf{x} \not\rightarrow t.split$)
9: **Return** the node t
10: **end if**
11: **end function**

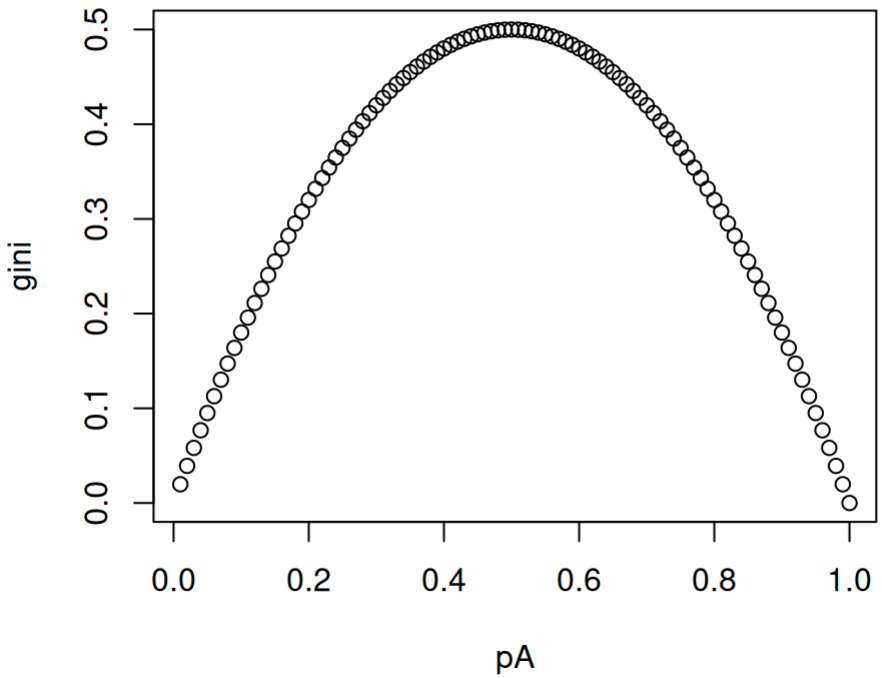
Find Best Split: GINI index

The Gini index of a dataset D, where each example belongs to one of C classes:

$$Gini(D) = 1 - \sum_{i=1}^{C} p_i^2$$

- p_i is the observed frequency of class i.

- Consider a binary case where two classes are A and B



GINI index

If D is split by a logical test s, then

$$Gini_s(D) = \frac{|D_s|}{|D|} Gini(D_s) + \frac{|D_{\neg s}|}{|D|} Gini(D_{\neg s})$$

Then, the reduction in impurity is given by

$$\Delta Gini_s(D) = Gini(D) - Gini_s(D)$$

- Information gain based on entropy is also frequently used

Least Squares

- For regression, LS is frequently used to measure error

$$Err(D) = \frac{1}{|D|} \sum_{\langle x_i, y_i \rangle \in D} (y_i - k_D)^2$$

where k_D is the constant representing value of D.

- It is shown that $mean(y_i)$ actually minimizes LS.
- If D is split by a logical test s, then

$$Err_s(D) = \frac{|D_s|}{|D|} Err(D_s) + \frac{|D_{\neg s}|}{|D|} Err(D_{\neg s})$$

Then, the reduction in impurity is given by

$$\Delta Err_s(D) = Err(D) - Err_s(D)$$

Termination

- When to stop?
  - Too deep -> over-fitting, variance error
  - Too shallow -> over-simplified, bias error
- Control with parameters
  - leaf size
  - split size
  - depth
  - complexity
- Grow a very large tree, then prune
  - According to some statistical information

Implementation

- Implemented in `rpart` and `party`
  - We will use `rpart`
- Functions
  - `rpart()` and `prune.rpart()`
- Book package contains
  - `rpartXse()` which combines `rpart()` and `prune.rpart()`
  - applies post-prunning with X-SE rule

Formula

- A formula in R is provided in the following form

$$Y \sim X_1 + X_2 + X_3 + X_4 \ldots$$

- This means the value of Y depends on the values of Xs

$$Y \sim .$$

- means Y vs. everything else

Randomicity

- Due to the certain randomized parts of the algorithm, it is possible to obtain slightly different trees between different runs.
- Hence, always use a seed
- `rpart.plot` package allows nice drawings of DTs using `prp`

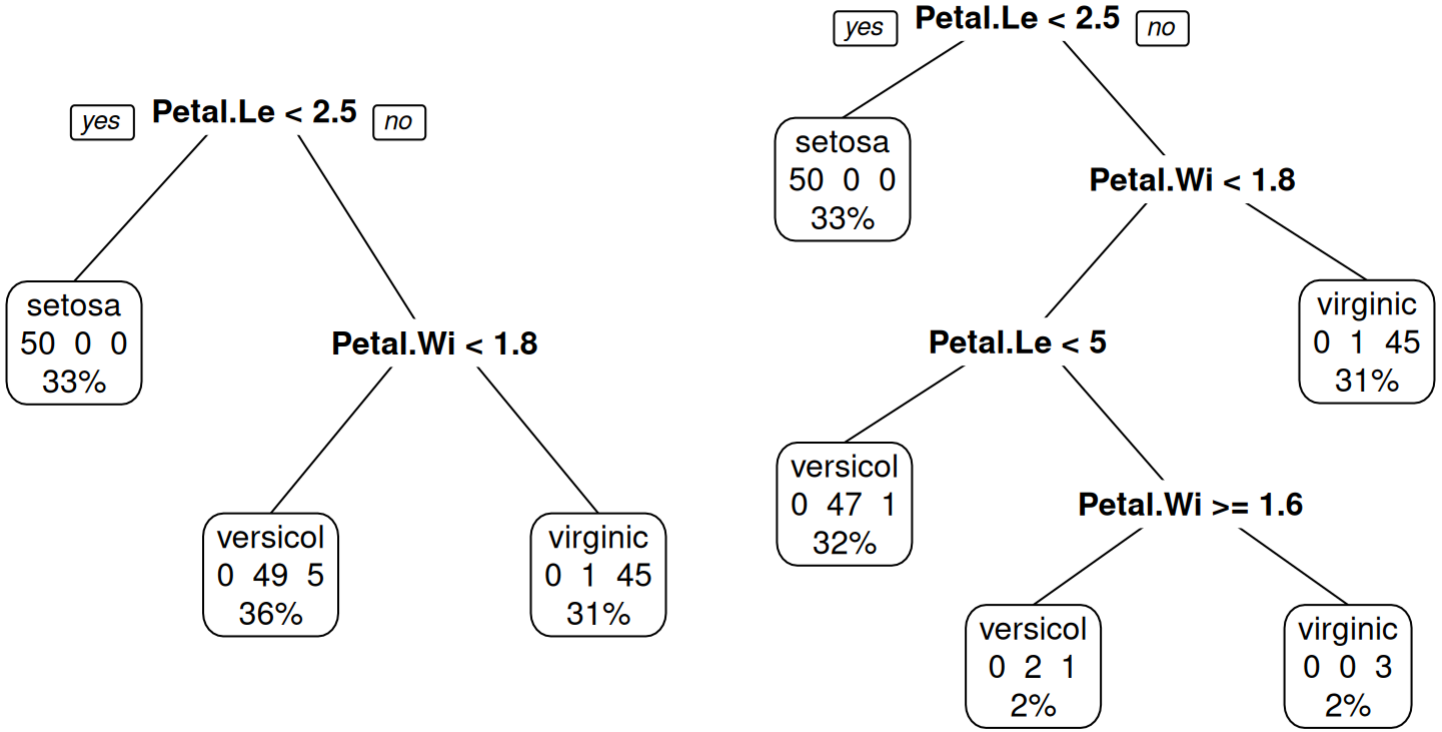
Example

```
data(iris)
ct1 <- rpartXse(Species ~ ., iris, model = TRUE)
ct2 <- rpartXse(Species ~ ., iris, se = 0, model = TRUE)
```

- `se=0` is a less agressive prunning

Example

```
par(mfrow=c(1,2))
prp(ct1, type = 0, extra = 101)
prp(ct2, type = 0, extra = 101)
```



Example

```
samp <- sample(1:nrow(iris), 120)
tr_set <- iris[samp, ]
tst_set <- iris[-samp, ]
model <- rpartXse(Species ~ ., tr_set, se = 0.5)
predicted <- predict(model, tst_set, type = "class")
head(predicted)
```

```
##     12     15     35     37     40     43
## setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
table(tst_set$Species, predicted)
```

```
##             predicted
##              setosa versicolor virginica
##   setosa          8          0         0
##   versicolor      0         10         1
##   virginica       0          0        11
```

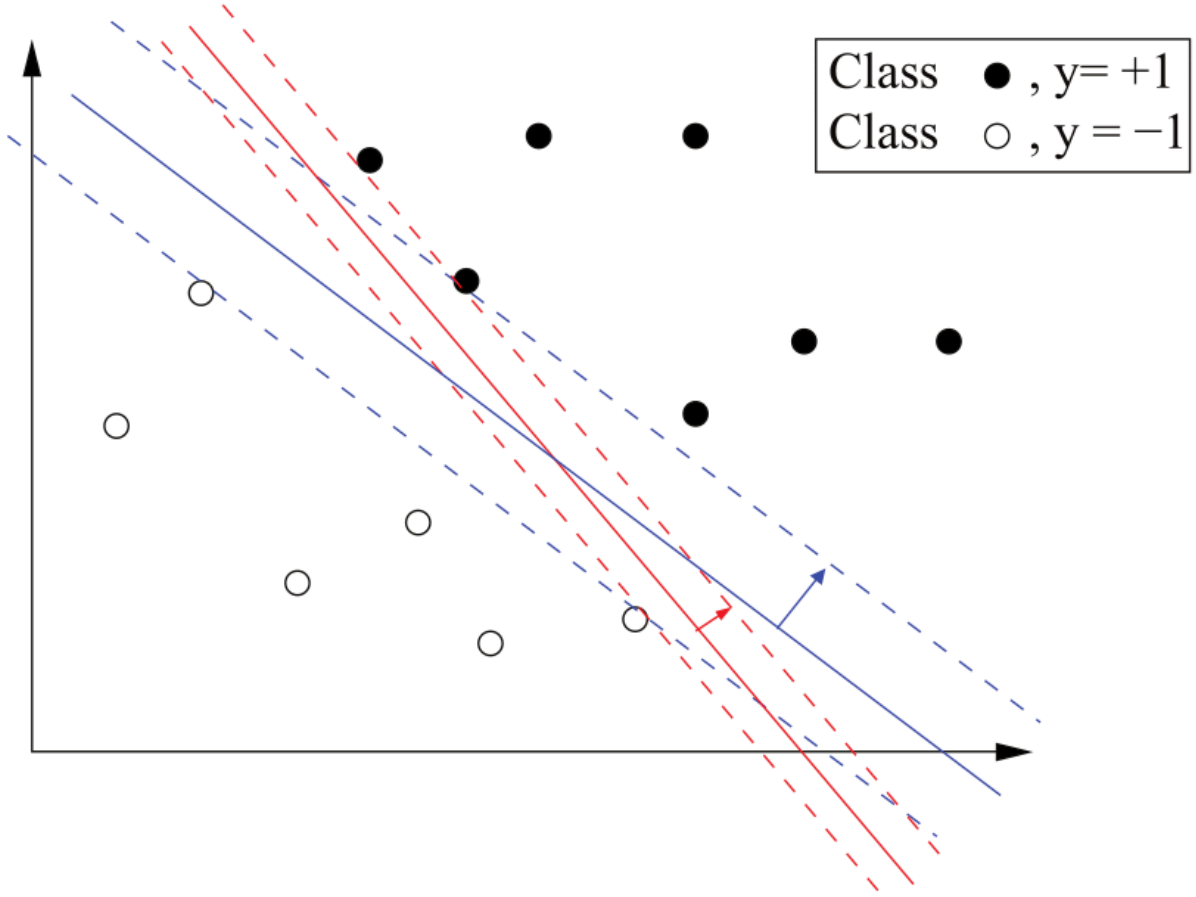
```
errorRate <- sum(predicted != tst_set$Species) / nrow(tst_set)
errorRate
```

```
## [1] 0.03333333
```

Support Vector Machines

Support Vector Machines

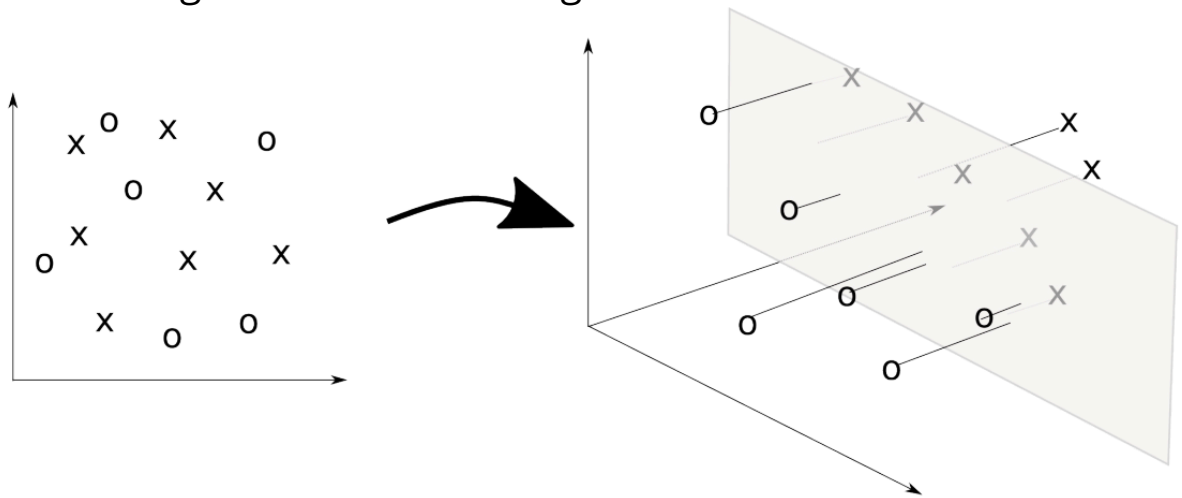
.



Linearly separable sets

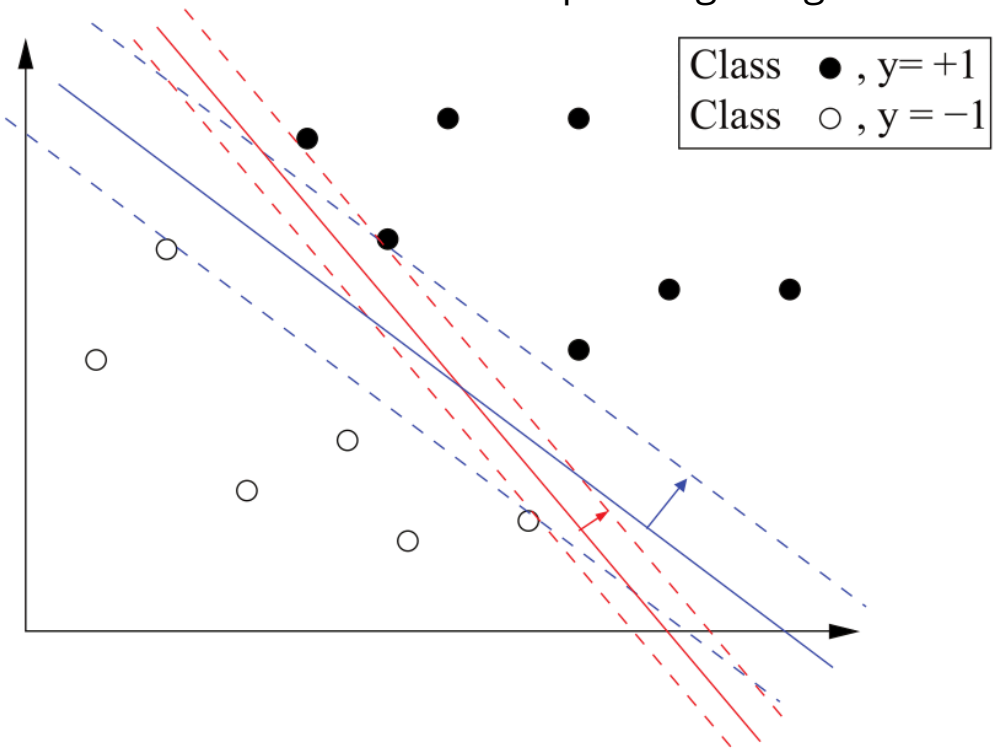
Support Vector Machines

- Linearly non-separable sets
  - Lift to a higher dimension using a non-linear function



Support Vector Machines

- Questions
  - Which function to use?
  - Which hyperplane to choose?
    - The one that maximizes the separating margin



Support Vector Machines

- Choosing the optimal hyperplane
  - Involves linear algebra and quadratic optimization
    - Lagrangian relaxation
    - Dual problem
    - Karush-Kuhn-Tucker conditions
  - Core operation is computing the dot product of two points (vectors)
    - Which can be very expensive after dimension expansion
  - We need to do this faster

Kernel Trick

- Kernel trick
  - Consider two points $x : \langle x_1, x_2 \rangle$ and $z : \langle z_1, z_2 \rangle$
  - Let $\phi(x)$ be a nonlinear mapping of x to a higher dimension
  - We want to compute $\phi(x) \cdot \phi(z)$
  - Consider the following kernel function: $K(x_i, x_j) = (x_i \cdot x_j)^2$
  - Then

$$K(x, z) = (\langle x_1, x_2 \rangle \cdot \langle z_1, z_2 \rangle)^2$$

$$= (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2$$

$$= \langle x_1^2, x_2^2, \sqrt{2} x_1 x_2 \rangle \cdot \langle z_1^2, z_2^2, \sqrt{2} z_1 z_2 \rangle$$

  - So, for $\phi(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2, \sqrt{2} x_1 x_2 \rangle$ we have

$$K(x, z) = \phi(x) \cdot \phi(z)$$

Kernel Function Families

- This means, if we find these Kernel functions then we can use them for mapping our data to higher dimensions much faster.
- Indeed there are many such kernel function families
  - Gaussian kernel

$$K(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$$

  - Polynomial

$$K(x_i, x_j) = (x_i \cdot x_j)^d$$

  - Radial kernel

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$$

Support Vector Machines

- How does SVM handle non-binary classification?
  - By solving multiple binary classification problems
- Regression?
  - ϵ-SV approach finds an optimal hyperplane where each data point lies within ϵ distance of the hyperplane.

Implementation

- Implemented in packages `e1071` and `kernlab`.
  - They are quite similar. `kernlab` may be more flexible. `e1071` is simpler.

Example

```
data(iris)
rndSample <- sample(1:nrow(iris), 100)
tr <- iris[rndSample, ]
ts <- iris[-rndSample, ]
s <- svm(Species ~ ., tr)
ps <- predict(s, ts)
(cm <- table(ps, ts$Species))
```

```
##
## ps           setosa versicolor virginica
##   setosa         24          0         0
##   versicolor      0         14         0
##   virginica       0          1        11
```

Example

```
s2 <- svm(Species ~ ., tr, cost=10, kernel="polynomial", degree=3)
ps2 <- predict(s2, ts)
(cm2 <- table(ps2, ts$Species))
```

```
##
## ps2          setosa versicolor virginica
##   setosa         24          0         0
##   versicolor      0         15         3
##   virginica       0          0         8
```