UNIVERSIDAD PONTIFICIA ICAI ICADE COMILLAS MADRID

Machine Learning

Chapter 2: Classification II

January 2023

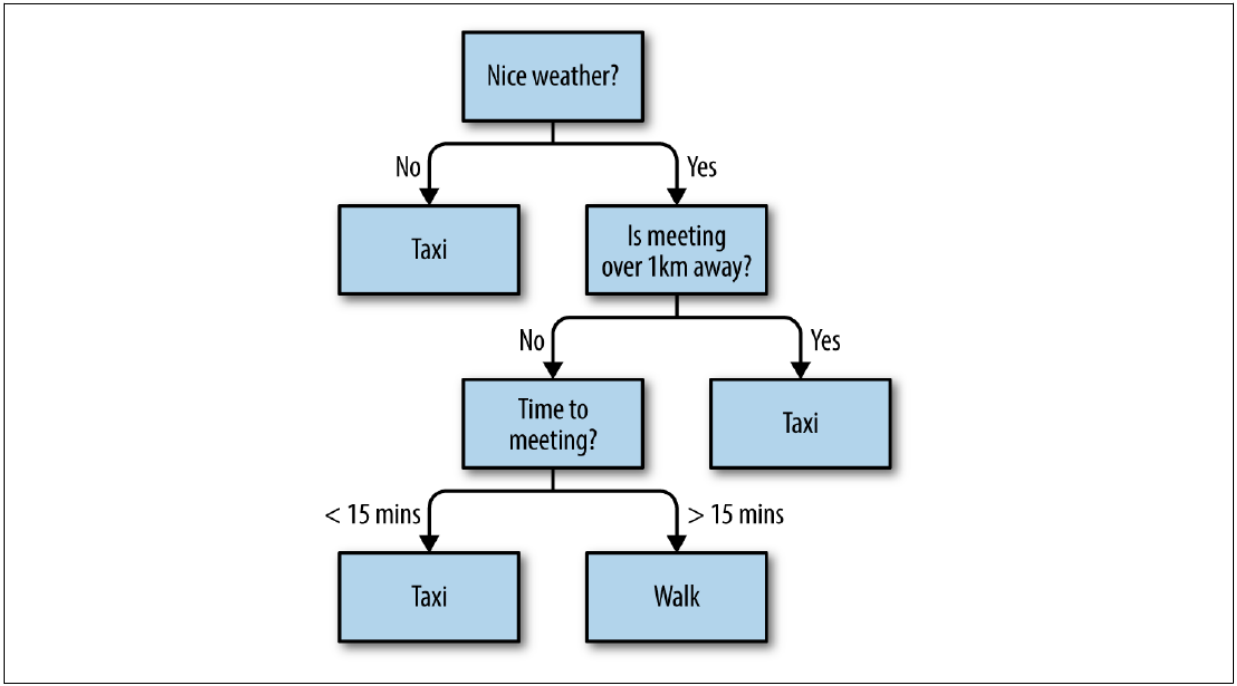
comillas.edu

Contents

1

Decision Trees

Decision Trees
Introduction



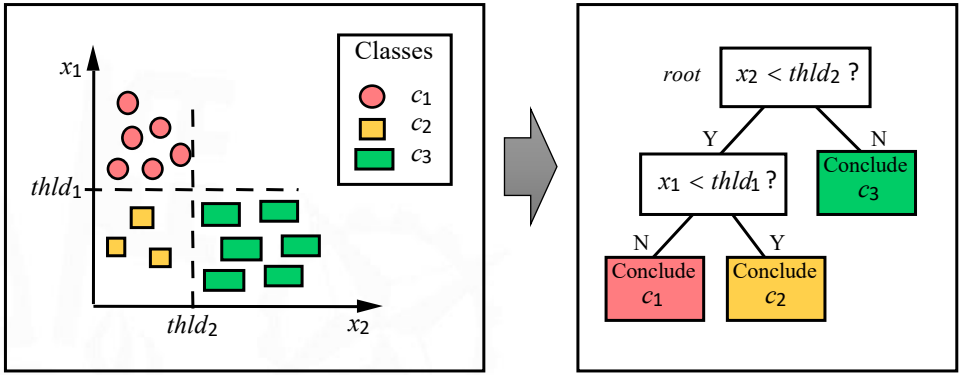
source: (Cook, 2017)

Decision Trees
Introduction

- Main characteristics:
  - Easy-to-understand general representation of a discrete classifier
  - Fast learning algorithms (ID3, Classification and Regression Trees - CART)
  - Built-in feature selection
  - Widely used in solving large classification problems:
    - Credit card risk
    - Medical diagnosis
    - Industrial applications

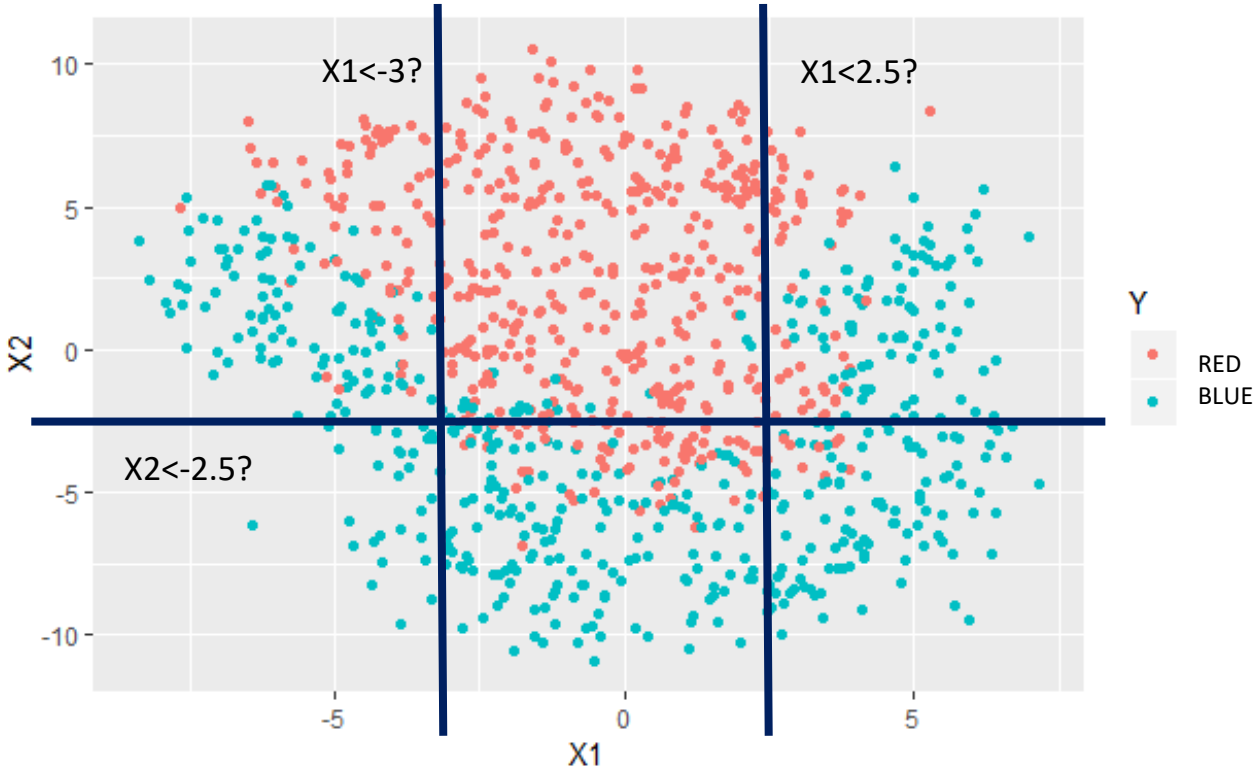
Decision Trees
Introduction

- Example:

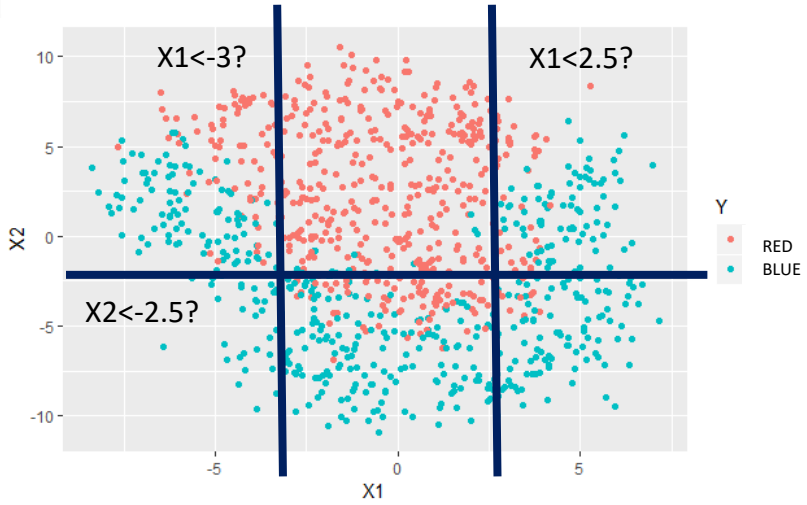


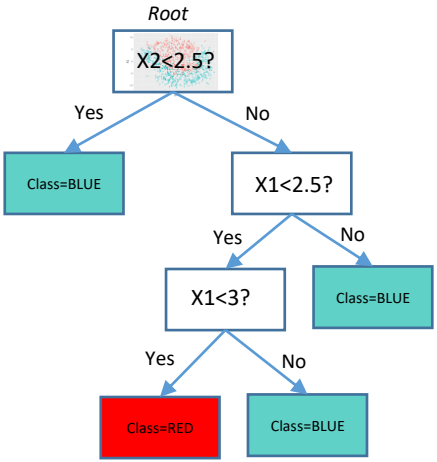
- Types of **nodes**: decision and terminal or *leaf* nodes
- Types of **separators**:
  - For **categorical** input variables: Value of X?
  - For **continuous** input variables: Value of $X <$ threshold?

Decision Trees
Building algorithm



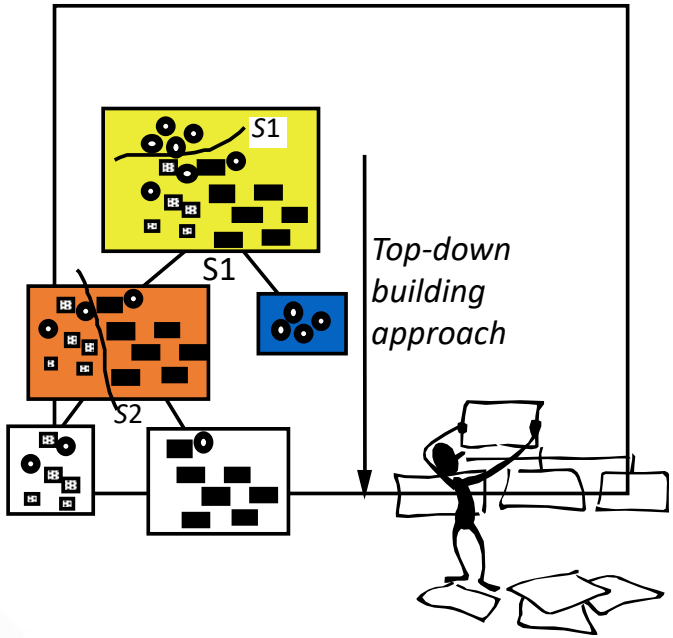
Decision Trees
Building algorithm





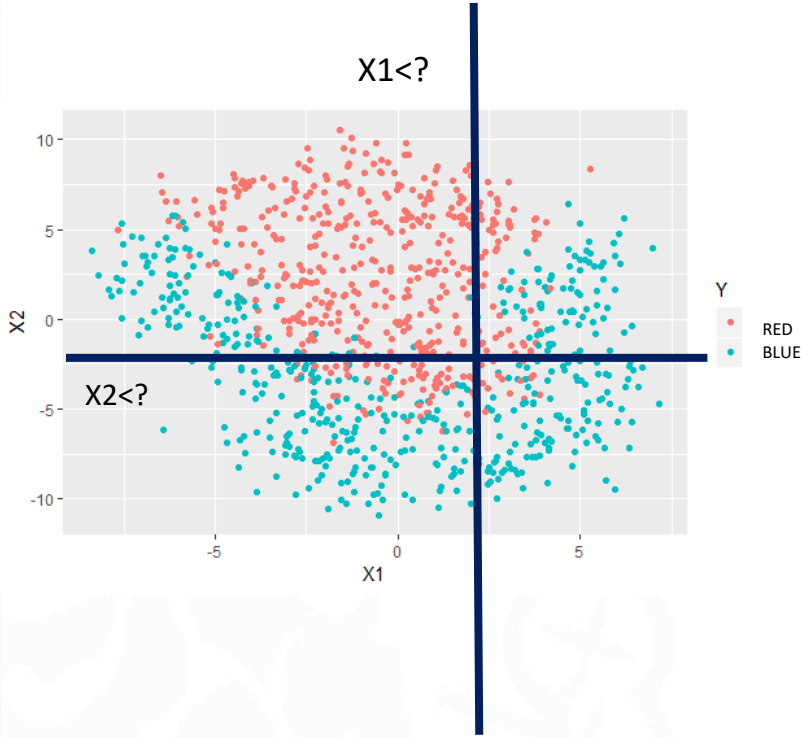
Decision Trees
Building algorithm

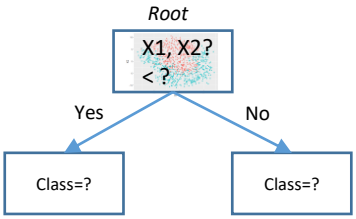
- Main idea:
  - Split the input space recursively by minimizing the impurity of the nodes until the terminal nodes are pure enough.



How do we assess the impurity of a set of samples?

Decision Trees
Building algorithm





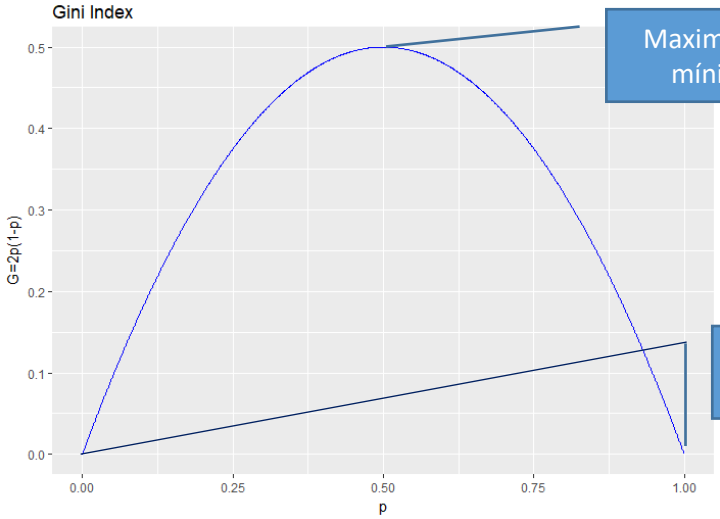
Decision Trees
Building algorithm

- Two main measures of purity:
  - Gini index (CART decision trees)
  - Entropy/Information statistic (C4.5 decision trees)

Decision Trees
Gini Index

- The Gini Index for a given node is defined as:

$$Gini = 1 - \sum_{i=1}^{m} p_i^2 \qquad = 2\,p_1 p_2 \quad for\ \ m = 2$$



Maximum disorder = mínimum purity

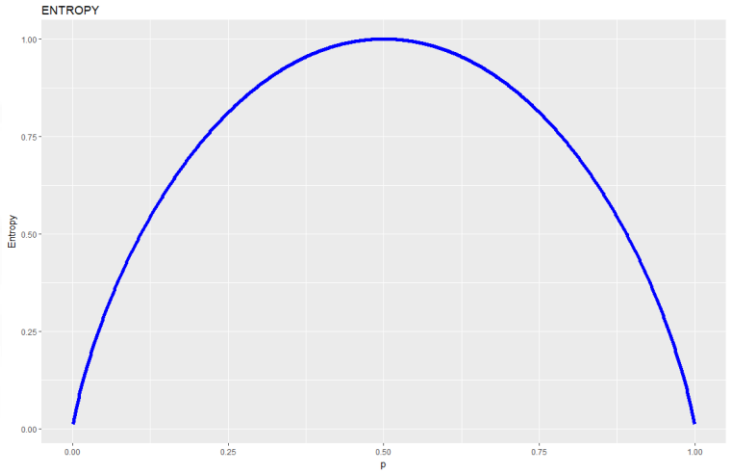
Minimum disorder = maximum purity

Decision Trees
Entropy

- Entropy or information statistic (used in C4.5):

$$Entropy = -\sum_{i=1}^{m} p_i \, log_2(p_i)$$

when $p = 0$ it is customary to have $0 \, log_2(0) = 0$

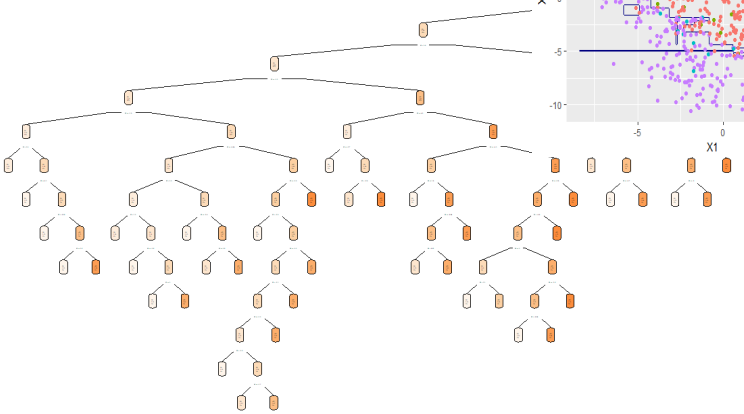


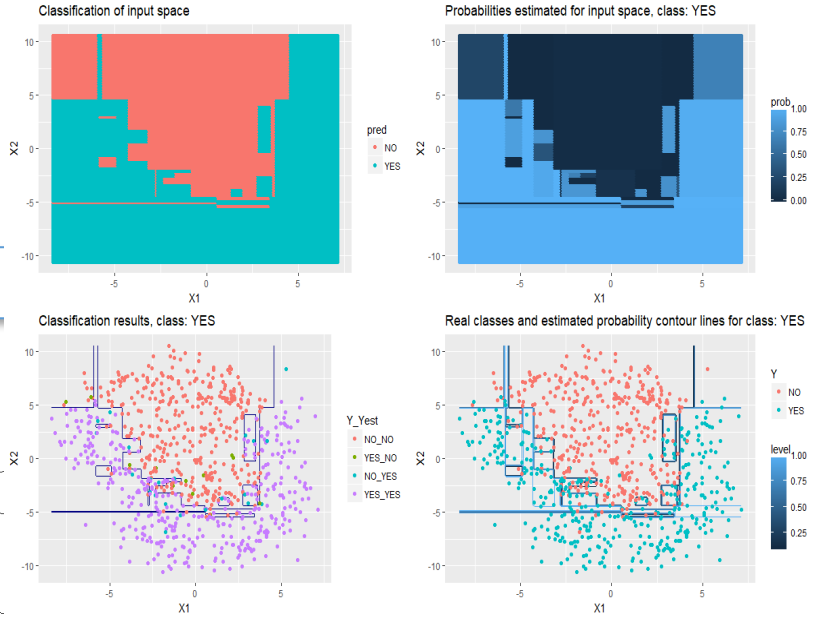
How much impurity/purity are you willing to accept? Regulated by the complexity parameter *cp*:

$Cost_{cp} = Cost + cp\ |T|$

For example: Maximum purity

$cp = 0$





cp: complexity parameter (rpart).

Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. Essentially, the user informs the program that any split which does not improve the fit by cp will likely be pruned off by cross-validation, and that hence the program need not pursue it.

Decision Trees
Building algorithm

- When working with a continuous predictor and a categorical response, the process for finding the optimal split point is given by:
  1. The samples are sorted based on their predictor values.
  2. The split points are then the midpoints between each unique predictor value. If the response is binary, then this process generates a 2×2 contingency table:

	Class 1	Class 2	
$>$ split	n_{11}	n_{12}	n_{+1}
$\leq$ split	n_{21}	n_{22}	n_{+2}
	n_{1+}	n_{2+}	n

Decision Trees
Building algorithm & Gini

	Class 1	Class 2	
$>$ split	n_{11}	n_{12}	n_{+1}
$\leq$ split	n_{21}	n_{22}	n_{+2}
	n_{1+}	n_{2+}	n

- The Gini index prior to the split would be:

$$Gini(\text{prior to split}) = 2\left(\frac{n_{1+}}{n}\right)\left(\frac{n_{2+}}{n}\right)$$

- And the Gini index can be calculated after the split within each of the new nodes and combine them using the proportion of samples in each partition:

$$Gini\ (\text{after split}) = \frac{n_{+1}}{n}\left[2\left(\frac{n_{11}}{n_{+1}}\right)\left(\frac{n_{12}}{n_{+1}}\right)\right] + \frac{n_{+2}}{n}\left[2\left(\frac{n_{21}}{n_{+2}}\right)\left(\frac{n_{22}}{n_{+2}}\right)\right]$$

$$= 2\left[\left(\frac{n_{11}}{n}\right)\left(\frac{n_{12}}{n_{+1}}\right) + \left(\frac{n_{21}}{n}\right)\left(\frac{n_{22}}{n_{+2}}\right)\right]$$

3. Partitioning algorithms evaluate nearly all split points and select the split point value that minimizes the Gini index.
4. The splitting process continues until the stopping criteria is met (such as the minimum number of samples in a node or the maximum tree depth).

Decision Trees
Building algorithm

- Trees that are constructed to have the maximum depth are notorious for over-fitting the training data.
- A more generalizable tree is one that is a pruned version of the initial tree and can be determined by cost-complexity tuning, in which the purity criterion is penalized by a factor of the total number of terminal nodes in the tree.
- After the tree has been pruned, it can be used for prediction. In classification, each terminal node produces a vector of class probabilities based on the training set which is then used as the prediction for a new sample.
- Tree models can also bin categorical predictors. Evaluating purity for each of these new predictors is then simple, since each predictor has exactly one split point.

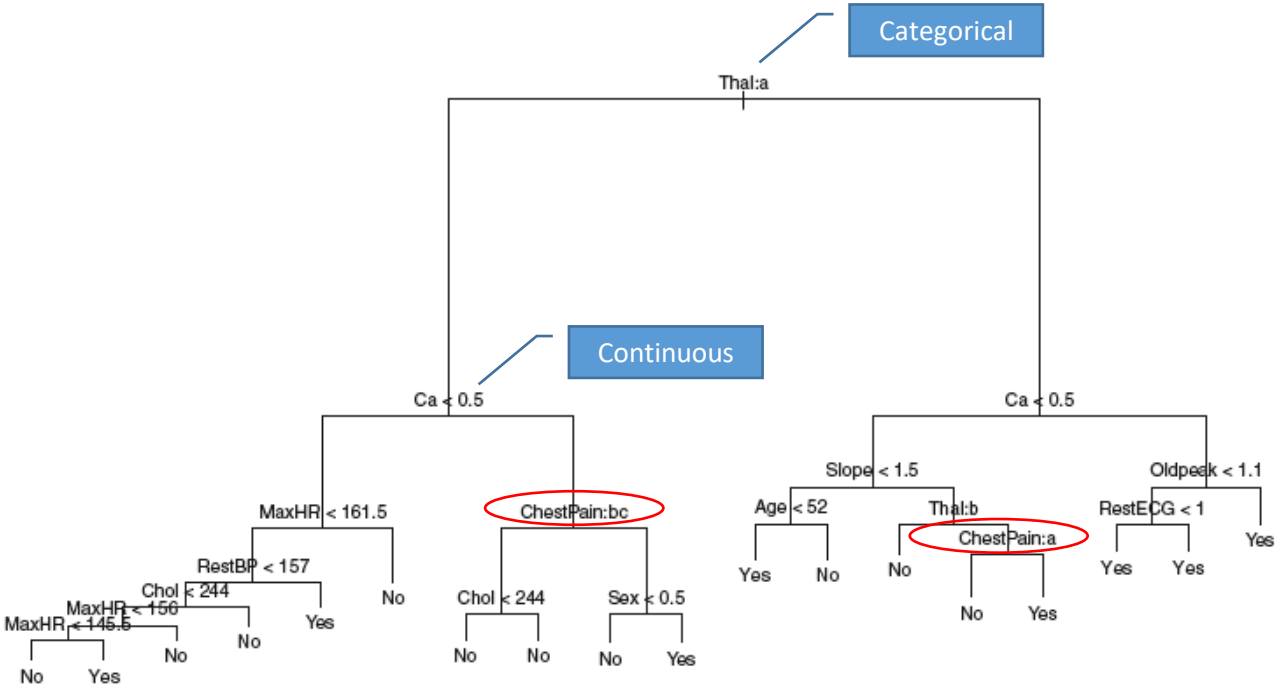
Decision Trees
Building algorithm

- When fitting trees and rule-based models, the practitioner must make a choice regarding the treatment of categorical predictor data:

1. Each categorical predictor can be entered into the model as a single entity so that the model decides how to group or split the values (grouped categories). For a categorical variable X with 3 levels (a,b,c) we consider the grouped categories: a – ab – ac – b – bc – c

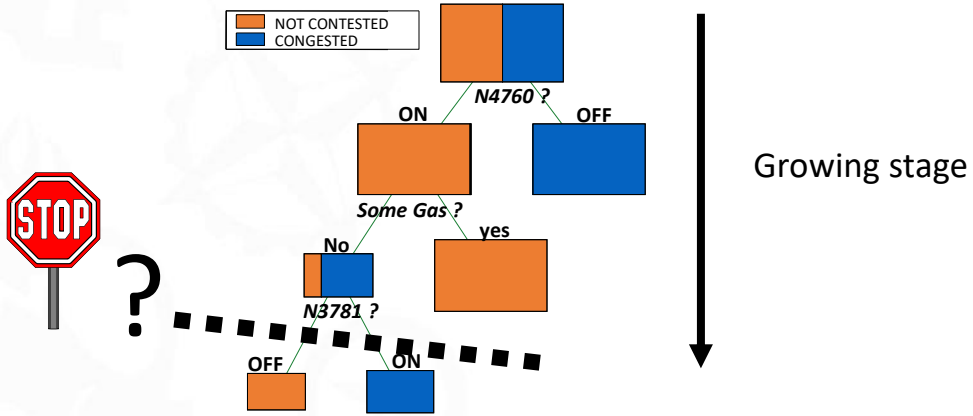
2. Categorical predictors are first decomposed into binary dummy variables. In this way, the resulting dummy variables are considered independently, forcing binary splits for the categories (independent categories). For a categorical variable X with 3 levels (a,b,c) we construct the binary dummy variables: X_a, X_b, X_c and we split each dummy variable indepently: X_a=0/1

Decision Trees
Example: Heart data set



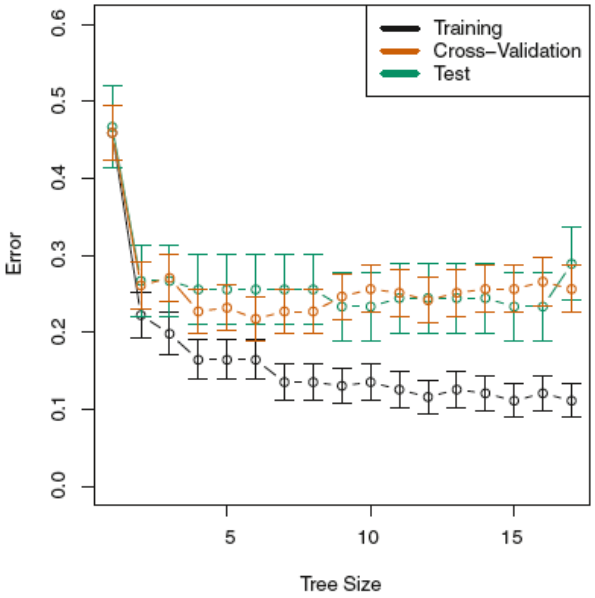
Decision Trees
Stopping criterion

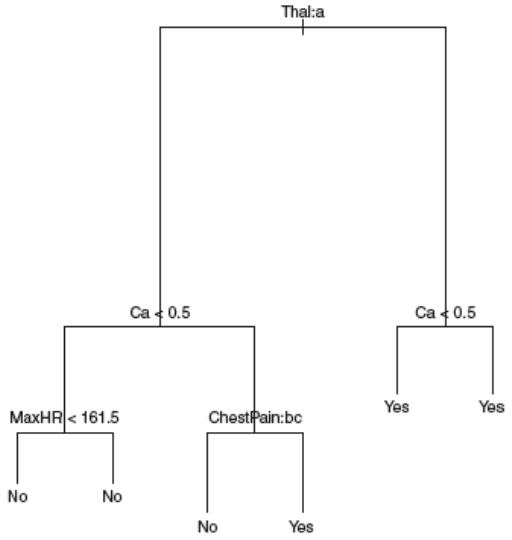
- There exist different stopping criteria (more or less complex)
  - Number of nodes / depth of the tree
  - Minimum number of observations in a node
  - Entropy ⇔ Stop splitting the node *n* if the entropy is small enough



- Cross-validation is used for selecting the optimal complexity

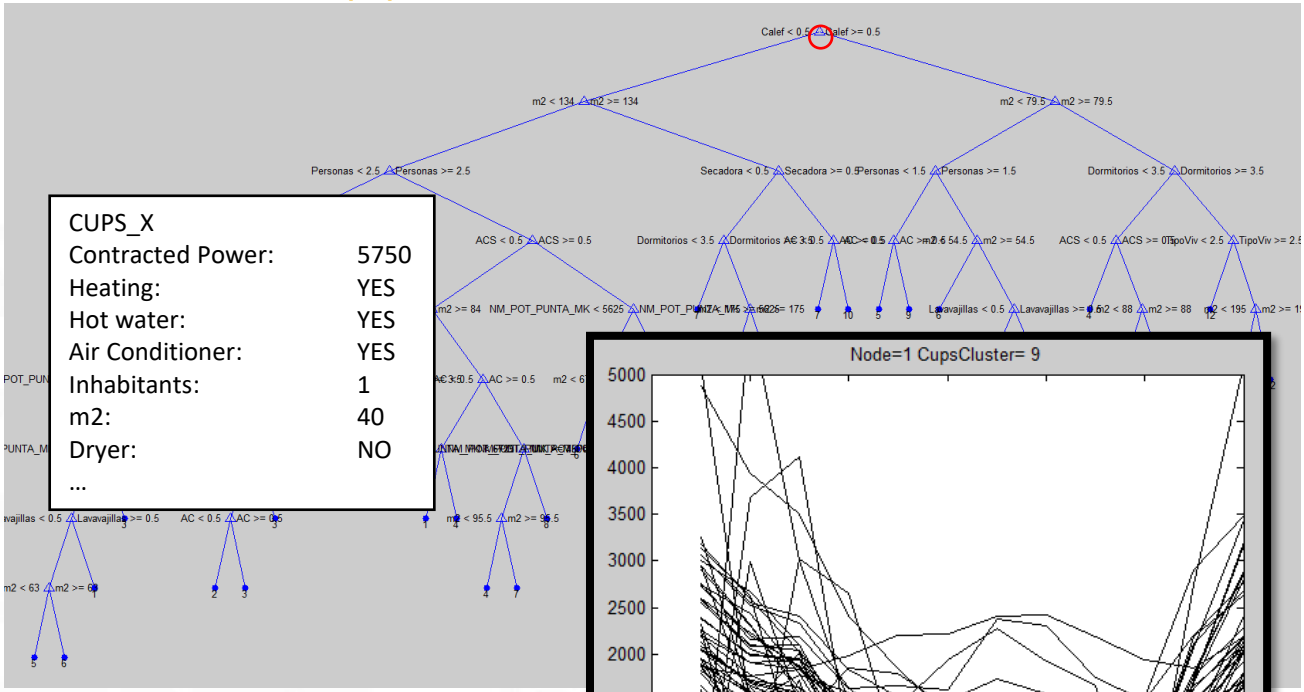
Decision Trees
Example: Heart data set



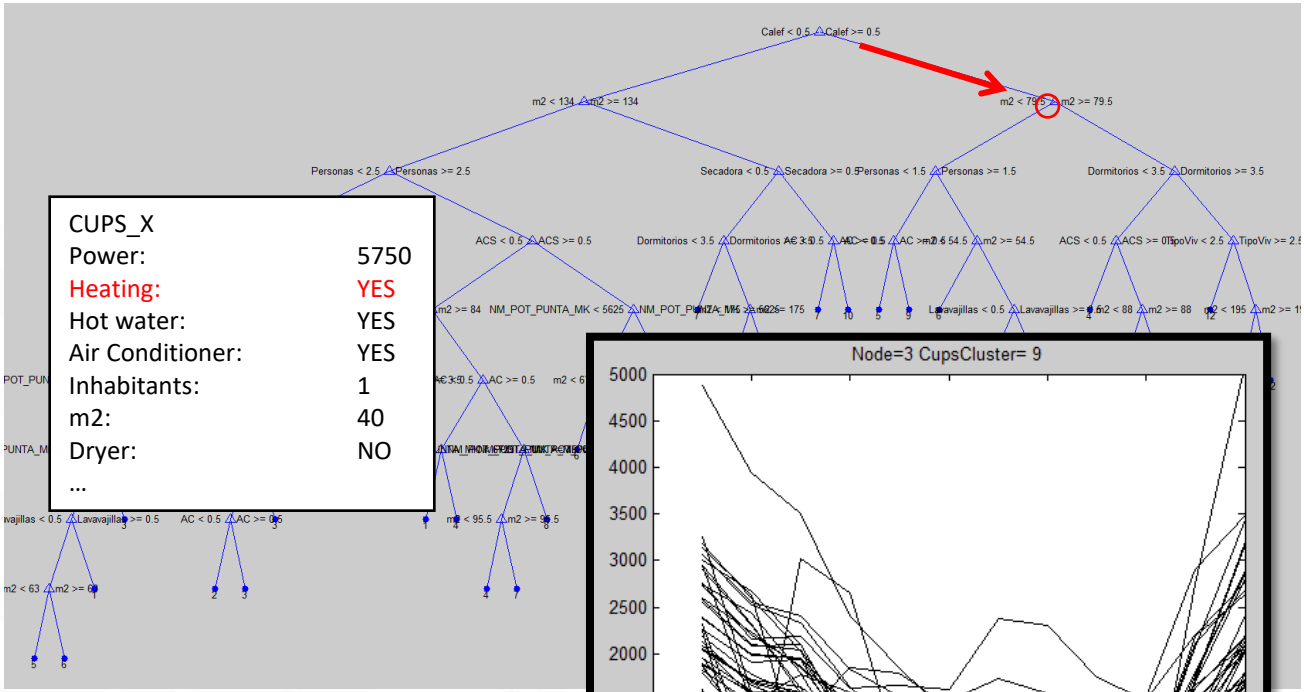


Pruned tree corresponding to the minimal cross-validation error

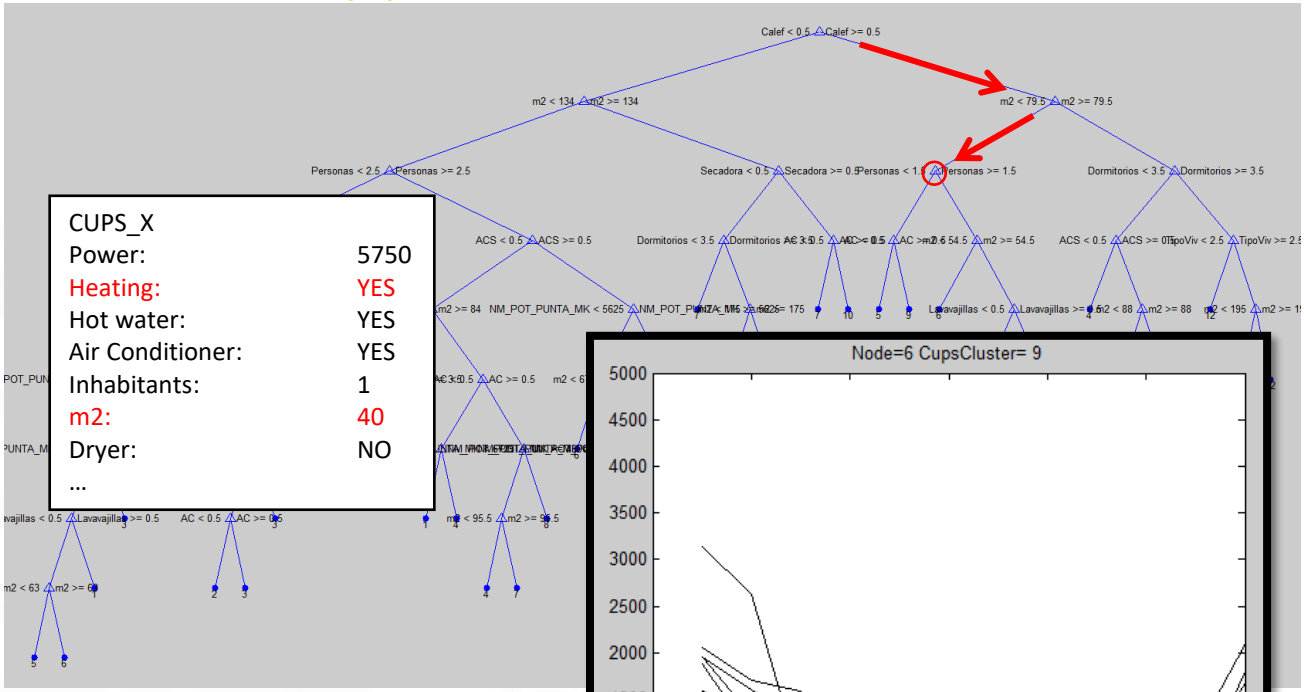
Decision Trees
Industrial application



Decision Trees
Industrial application

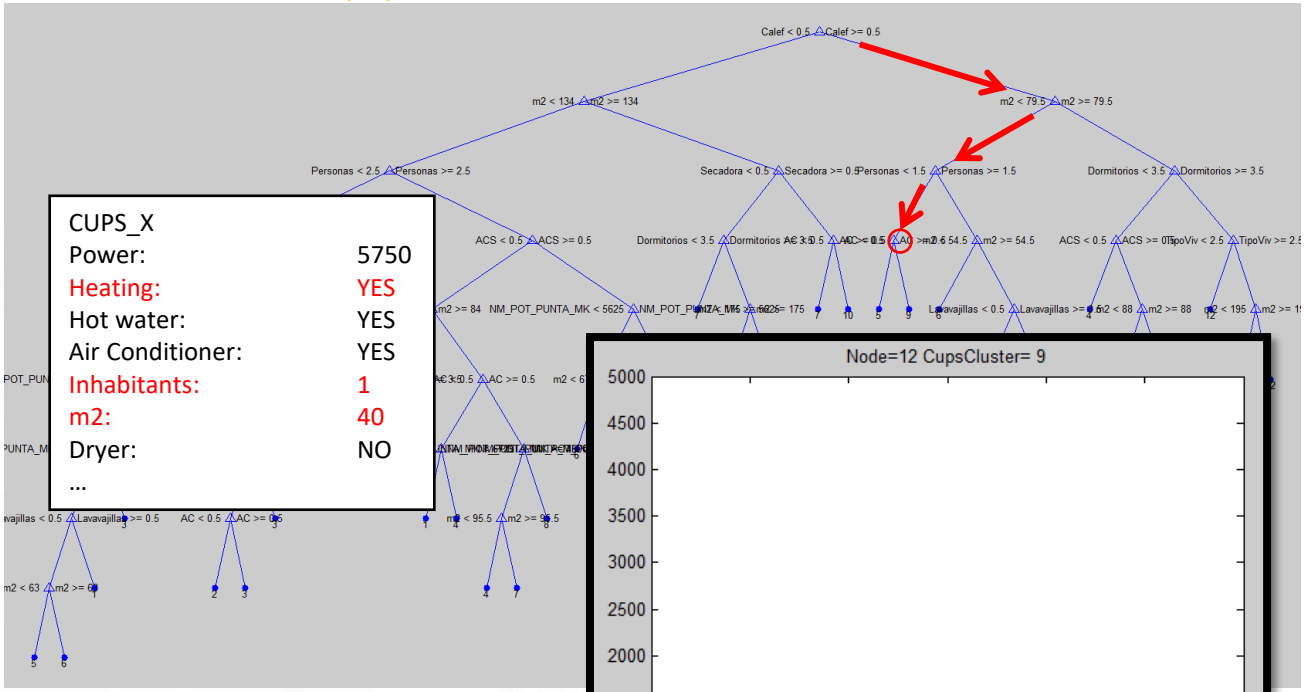


Decision Trees
Industrial application

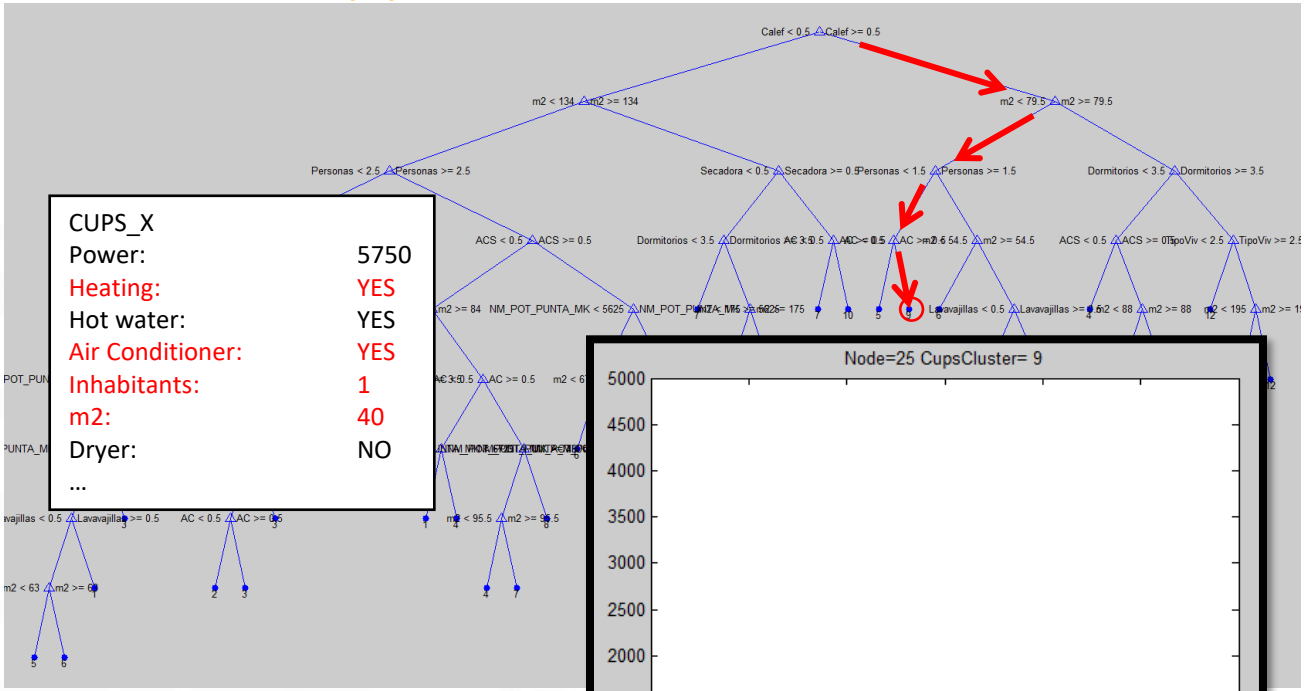


CUPS_X	
Power:	5750
Heating:	YES
Hot water:	YES
Air Conditioner:	YES
Inhabitants:	1
m2:	40
Dryer:	NO
…	

Decision Trees
Industrial application



Decision Trees
Industrial application

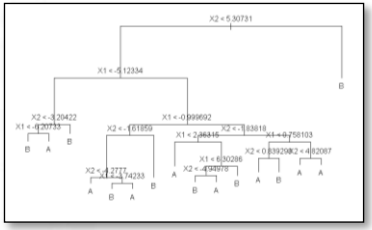


Decision Trees
R implementation

- There exists numerous options in R. For exampe, the function `tree()` from *tree* library.

```
library(tree)
modelFit = tree(Y ~ X1 + X2, fdata_train)
summary(modelFit) #Show information of the model
modelFit          #show model cuts in text
plot(modelFit)    #Plot the splits of the tree
text(modelFit)    #Add the labels to the tree plot
tree.pred= predict(modelFit , fdata_val, type ="class") #predict
```

```
> summary(modelFit) #Show information of the model

Classification tree:
tree(formula = Y ~ X1 + X2, data = fdata_train)
Number of terminal nodes:  16
Residual mean deviance:  0.3265 = 177.6 / 544
Misclassification error rate: 0.05536 = 31 / 560
```




```
> modelFit          #show model cuts in text
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 560 776.300 A ( 0.50000 0.50000 )
   2) X2 < 5.30731 467 631.100 A ( 0.59315 0.40685 )
     4) X1 < -5.12334 87  62.070 B ( 0.11494 0.88506 )
       8) X2 < -3.20422 34  41.190 B ( 0.29412 0.70588 )
        16) X1 < -6.20733 21   8.041 B ( 0.04762 0.95238 ) *
        17) X1 > -6.20733 13  16.050 A ( 0.69231 0.30769 ) *
       9) X2 > -3.20422 53   0.000 B ( 0.00000 1.00000 ) *
     5) X1 > -5.12334 380 462.500 A ( 0.70263 0.29737 )
      10) X1 < -0.999692 106 146.900 A ( 0.50943 0.49057 )
        20) X2 < -1.61859 57  33.880 A ( 0.91228 0.08772 )
          40) X2 < -4.2777 39   0.000 A ( 1.00000 0.00000 ) *
          41) X2 > -4.2777 18  21.270 A ( 0.72222 0.27778 )
            82) X1 < -3.74233 7   8.376 B ( 0.28571 0.71429 ) *
            83) X1 > -3.74233 11   0.000 A ( 1.00000 0.00000 ) *
        21) X2 > -1.61859 49  16.710 B ( 0.04082 0.95918 ) *
      11) X1 > -0.999692 274 290.600 A ( 0.77737 0.22263 )
        22) X2 < -1.83818 120 152.800 A ( 0.66667 0.33333 )
          44) X1 < 2.36315 55   9.996 A ( 0.98182 0.01818 ) *
          45) X1 > 2.36315 65  87.490 B ( 0.40000 0.60000 )
            90) X1 < 6.30286 48  66.210 A ( 0.54167 0.45833 )
             180) X2 < -4.94978 23  26.400 B ( 0.26087 0.73913 ) *
             181) X2 > -4.94978 25  25.020 A ( 0.80000 0.20000 ) *
            91) X1 > 6.30286 17   0.000 B ( 0.00000 1.00000 ) *
        23) X2 > -1.83818 154 122.700 A ( 0.86364 0.13636 )
          46) X1 < 0.758103 23  30.790 B ( 0.39130 0.60870 )
            92) X2 < 0.839298 9   0.000 A ( 1.00000 0.00000 ) *
            93) X2 > 0.839298 14   0.000 B ( 0.00000 1.00000 ) *
          47) X1 > 0.758103 131  54.630 A ( 0.94656 0.05344 )
            94) X2 < 4.82087 122  28.160 A ( 0.97541 0.02459 ) *
            95) X2 > 4.82087 9  12.370 A ( 0.55556 0.44444 ) *
   3) X2 > 5.30731 93  26.510 B ( 0.03226 0.96774 ) *
```

- Prune tree with `prune.misclass()` function.

```
prune. modelFit =prune.misclass(modelFit ,best =15)
```

- Cross-validation method with `cv.tree()` function.

```
cv.modelFit =cv.tree(modelFit ,FUN=prune.misclass)
```

Decision Trees
R implementation – caret

- *caret* has several methods implemented. One is `rpart` from *rpart* package.
- It has one tuning parameter, the complexity parameter *cp.*
- The *params* argument can be used to set the splitting criterion ("gini" or "information")

```
#Training the model
> modelfit = train(fdata_train[,c("X1","X2")], #Input variables
       y = fdata_train$Y, #Output variable
       method = "rpart", #tree
       parms = list(split = "gini"),     #splitting criterion
       preProcess = c("center","scale"), #pre-processing if desired
       tuneGrid = data.frame(cp = 0.01), #complexity parameter
       trControl = ctrl, #Resampling settings
       metric = "ROC") #Summary metrics

#Predictions for new data. Probabilities and classes
> Ypred_knn_prob = predict(modelfit, type="prob" , newdata = fdata_val)
> Ypred_knn_pred = predict(modelfit, type="raw" , newdata = fdata_val)
```

- Plot and text functions can be used.

```
plot(modelfit$finalModel)    #Plot the splits of the tree
text(modelfit$finalModel)    #Add the labels to the tree plot
```

Decision Trees
R implementation – caret II

- `Summary()` function gives information about the fitted nodes.

```
          CP nsplit rel error
1 0.33571429      0 1.0000000
2 0.25357143      1 0.6642857
3 0.04821429      2 0.4107143
4 0.03750000      4 0.3142857
5 0.03214286      6 0.2392857
6 0.01000000      8 0.1750000
```

```
variable importance
X2 X1
51 49
```

```
Node number 1: 560 observations,    complexity param=0.3357143
  predicted class=A  expected loss=0.5  P(node) =1
    class counts:   280   280
   probabilities: 0.500 0.500
  left son=2 (448 obs) right son=3 (112 obs)
  Primary splits:
      X2 < 0.9569922  to the left,  improve=49.30804, (0 missing)
      X1 < -1.168402  to the right, improve=30.54409, (0 missing)
```

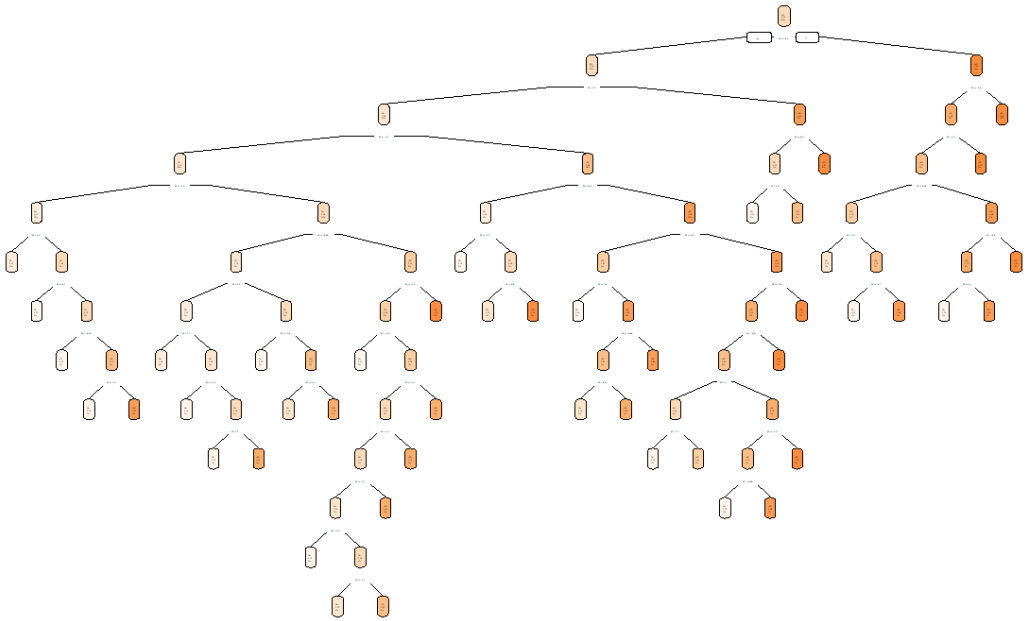
```
Node number 2: 448 observations,    complexity param=0.2535714
  predicted class=A  expected loss=0.3950893  P(node) =0.8
    class counts:   271   177
   probabilities: 0.605 0.395
  left son=4 (329 obs) right son=5 (119 obs)
  Primary splits:
      X1 < -0.8908446 to the right, improve=52.694450, (0 missing)
      X2 < -0.5969633 to the left,  improve= 3.750114, (0 missing)

Node number 3: 112 observations
  predicted class=B  expected loss=0.08035714  P(node) =0.2
    class counts:     9   103
   probabilities: 0.080 0.920

Node number 4: 329 observations,    complexity param=0.04821429
  predicted class=A  expected loss=0.2492401  P(node) =0.5875
    class counts:   247    82
   probabilities: 0.751 0.249
  left son=8 (258 obs) right son=9 (71 obs)
```

- In order to prune the model, select the *cp* level desired:

```
#prunning the final model
#The output is an rpart model!!!
> pruned_rpart = prune(modelfit$finalModel, cp = 0.4)
```

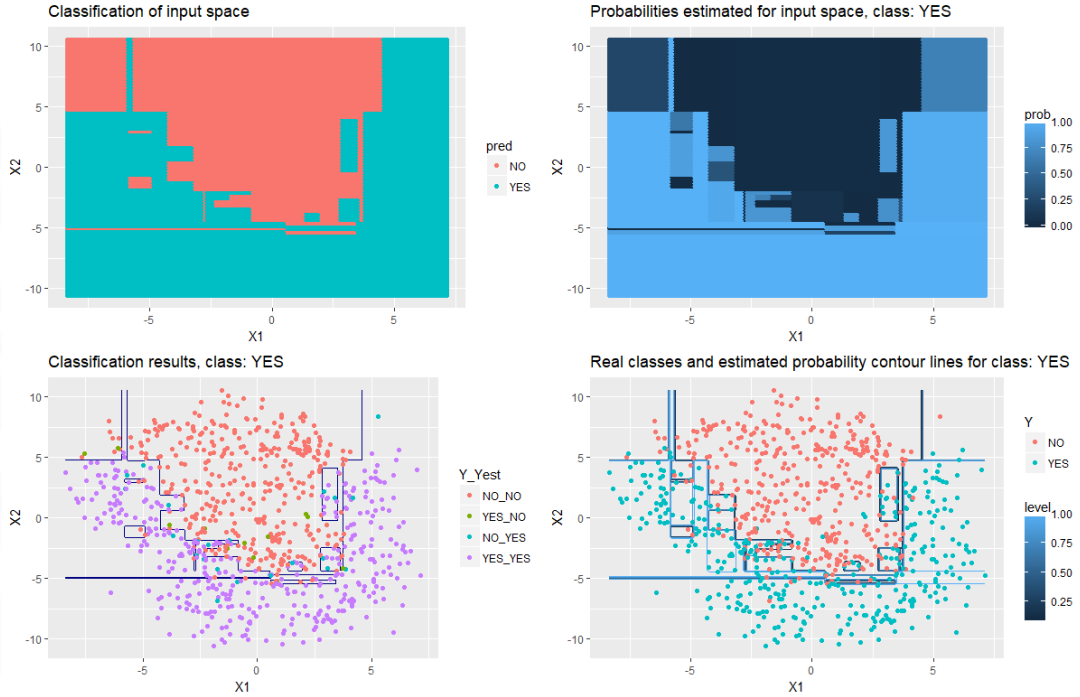
Decision Trees
R implementation

- Tree **cp = 0**



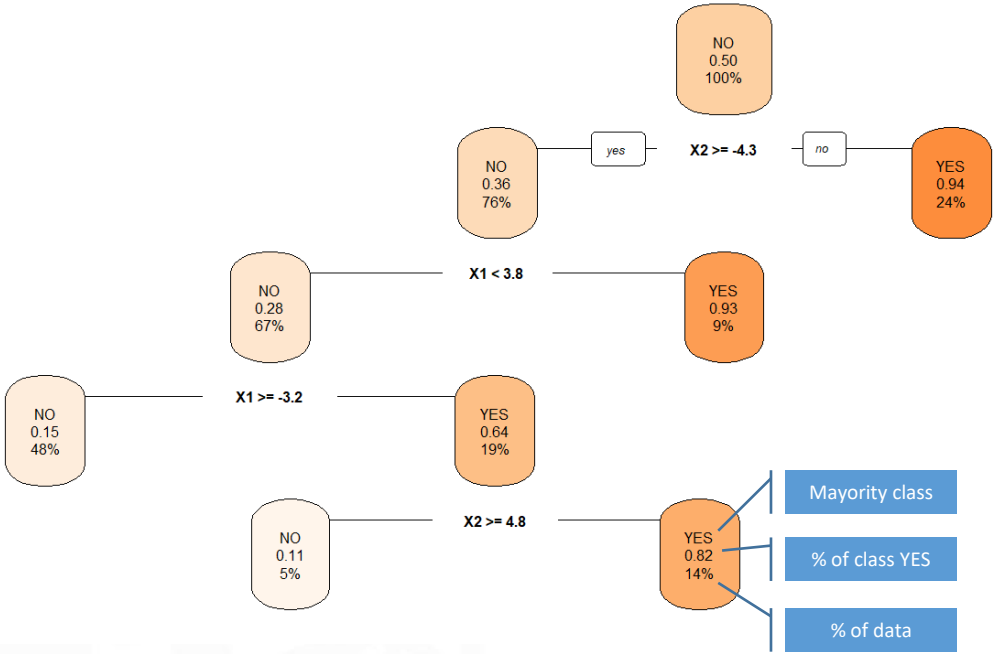
Decision Trees
R implementation

- Tree **cp = 0**



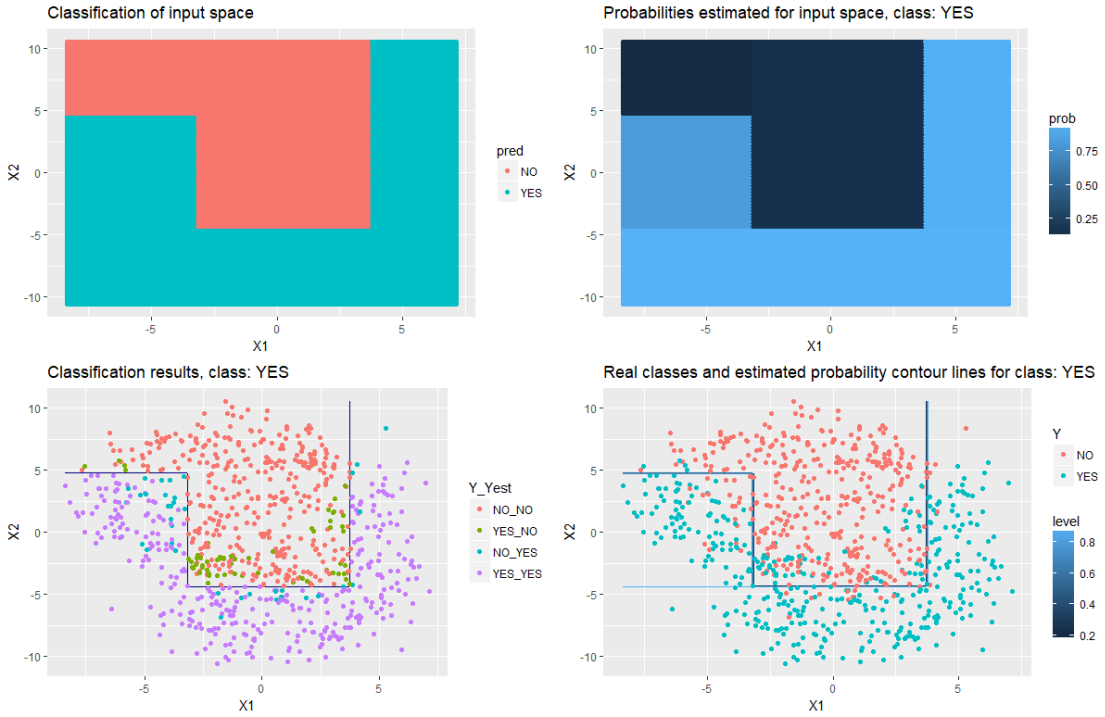
Decision Trees
R implementation

- Tree **cp = 0.025**



Decision Trees
R implementation

- Tree **cp = 0.025**



2

Random Forests

Bagging

- Decision trees suffer from high variance: using different training sets may produce quite different decision trees.

- In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets (e.g. linear regression)

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the bagging variance of a statistical learning method.

Bagging & Random Forests
Bagging

- Given a set of N independent observations $Z_1, \dots, Z_N$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/N $\Rightarrow$ *averaging a set of observations reduces variance*

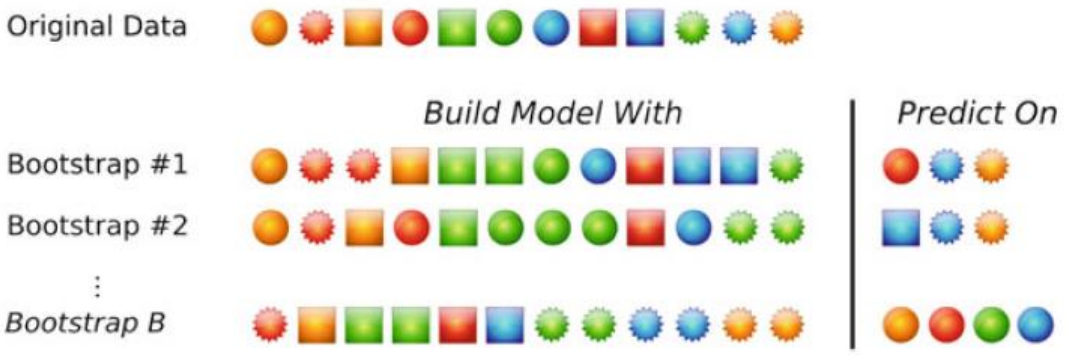
- Bagging ➔ take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions:
  1. We estimate B predictive models $\hat{f}_1(\boldsymbol{x}), \dots, \hat{f}_B(\boldsymbol{x})$ by taking B repeated samples from the (single) training data set (bootstraping)
  2. We average them in order to obtain a single low-variance statistical learning model:

$$\hat{f}_{bag}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\boldsymbol{x}) \text{ for regression}$$

$$\hat{f}_{bag}(\boldsymbol{x}) = majority\_vote(\hat{f}_1(\boldsymbol{x}), \dots, \hat{f}_B(\boldsymbol{x})) \text{ for classification}$$

Bagging & Random Forests
Out-of-Bag Error estimation

- The key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.



Source: (Kuhn et al., 2013)

Bagging & Random Forests
Random Forests

- Random forests provide an improvement over bagged trees by way of a random small tweak that decorrelates the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples
- When building these decision trees, each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of *n* predictors.
- A fresh sample of *m* predictors is taken at each split, and typically we choose $m = \sqrt{n}$
- Suppose that there is one very strong predictor in the data set. Then in the collection of bagged trees, most or all of the trees will use this predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other.
- Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.

Bagging & Random Forests
Random Forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

   i. Select m variables at random from the p variables.

   ii. Pick the best variable/split-point among the m.

   iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

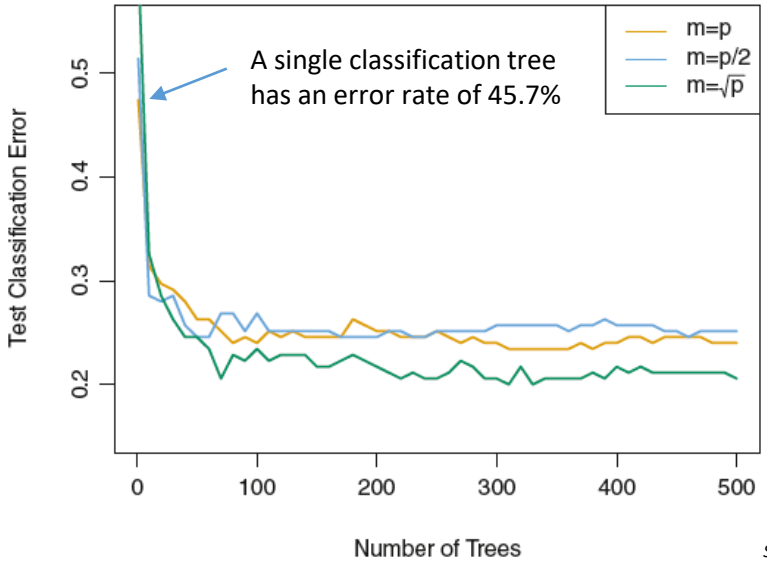
if a random forest is built using ***m = p***, *then this amounts simply to* ***bagging***

Random forests force each split to consider only a subset of the input variables

Source: T. Hastie, R. Tibshirani & J. Friedman (2009). *The Elements of Statistical Learning. Data Mining, Inference and Prediction*. 2nd Ed. Springer.

Bagging & Random Forests
Random Forests

- If a random forest is built using $m = n$, then this amounts simply to bagging.
- Using a small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors.



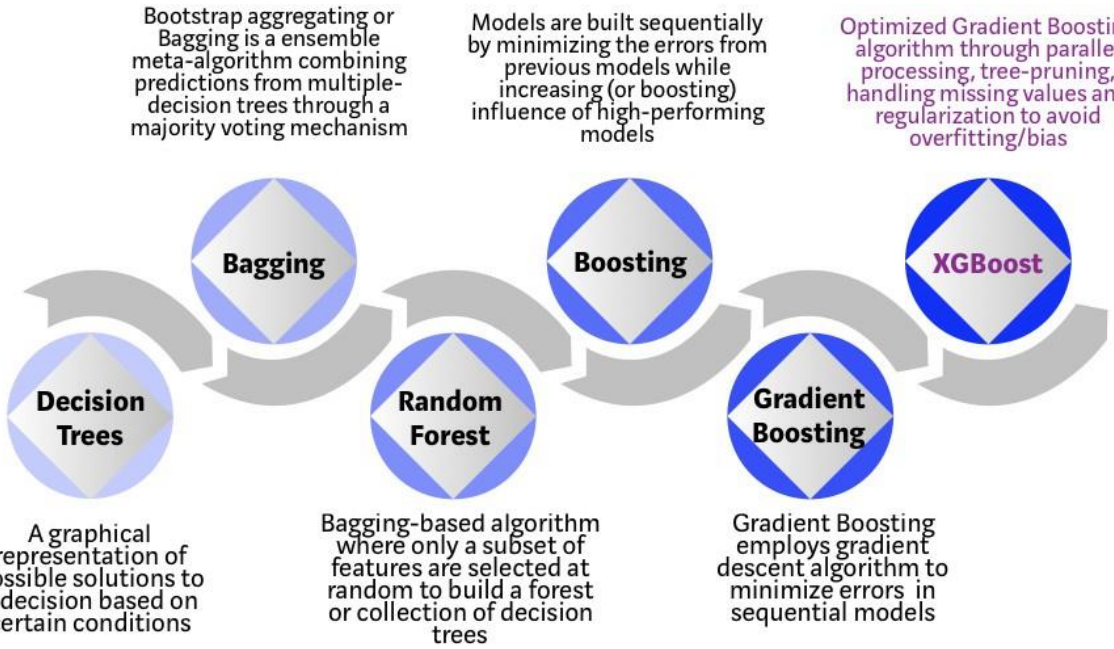
Source: (James et al., 2021)

Random forest
R implementation – caret

- *caret* has several methods implemented. One is `rf` from *randomForest* package.
- It has one tuning parameter *mtry:* the number of variables randomly sampled as candidates at each split.
- The *ntree* argument can be used to specify the number of trees to grow.

```
#Training the model
> modelfit = train(fdata_train, #Input variables
       y = fdata_train$Y, #Output variable
       method = "rf", #Random forest
       ntree = 1000,
       preProcess = c("center","scale"), #pre-processing if desired
       tuneGrid = data.frame(mtry = seq(1,ncol(fdata_train))),
       trControl = ctrl, #Resampling settings
       metric = "ROC") #Summary metrics

#Predictions for new data. Probabilities and classes
> Ypred_rf_prob = predict(modelfit, type="prob" , newdata = fdata_val)
> Ypred_rf_pred = predict(modelfit, type="raw" , newdata = fdata_val)
```

Boosting
From Decision Trees to XGBoost



Source: https://towardsdatascience.com/

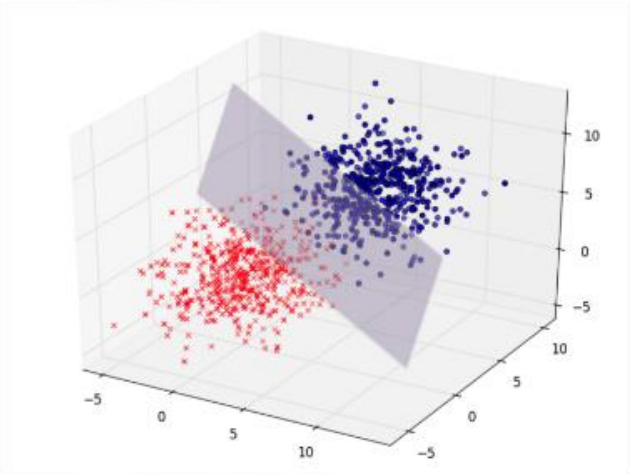
3

Support Vector Machines (SVM)

Support Vector Machines
Introduction

- SVM are a generalization of a simple and intuitive classifier called the *maximal margin classifier.*

- The maximal margin classifier can only solve *linearly separable problems.*



Support Vector Machines
Maximal Margin Classifier

- In a *n*-dimensional space, a ***hyperplane*** is a flat affine subspace of dimension *n-1*, given by:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n = 0$$

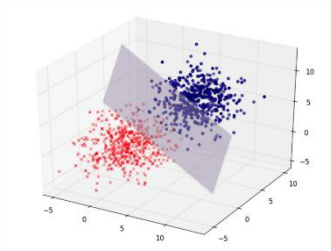
- In two dimensions, a hyperplane is a flat one-dimensional subspace, in other words a ***line***, given by:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

- In three dimensions it is a ***plane***:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 = 0$$

Support Vector Machines
Maximal Margin Classifier



- If ***x*** does not satisfy the equality, rather:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n > 0$$

then ***x*** lies to one side of the hyperplane.

- On the other hand, if:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n < 0$$

then ***x*** lies on the other side of the hyperplane.

- So the hyperplane divides the n-dimensional input space into two halves.

Support Vector Machines
Maximal Margin Classifier

- Now suppose that we have a *N*x*n* matrix of data that consists of *N* observations in a *n*-dimensional input space (the training set).

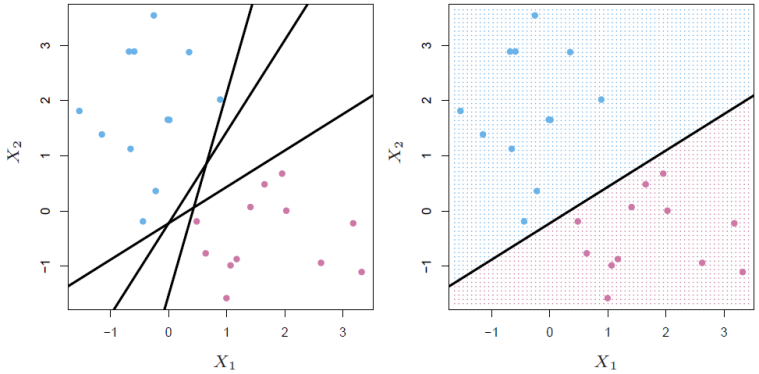
- Suppose that these observations fall into two classes:

$$y_1, y_2, \ldots, y_N \in \{-1, 1\}$$

- We also have a test observation $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)^T$. Our goal is to develop a classifier based on the training data that will correctly classify the test observation.

Support Vector Machines
Maximal Margin Classifier

- Suppose that it is possible to construct a hyperplane that separates the training observations perfectly according to their class labels:



Source: (James et al., 2021)

- A separating hyperplane satisfies for all *i*:

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in} > 0 \text{ if } y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in} < 0 \text{ if } y_i = -1$$

Support Vector Machines
Maximal Margin Classifier

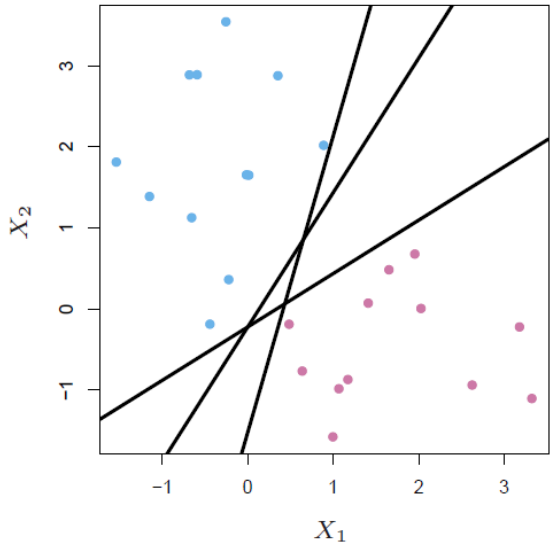
- If a separating hyperplane exists, then we will classify the test observation $\boldsymbol{x}^*$ based on the sign of:

$$f(\boldsymbol{x}^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_n x_n^*$$

- We can also use the magnitude of $f(\boldsymbol{x}^*)$:
  - ✓ If $f(\boldsymbol{x}^*)$ is far from zero, then it means that $\boldsymbol{x}^*$ lies far from the hyperplane, and so we can be confident about our class assignment for $\boldsymbol{x}^*$.
  - ✓ On the other hand, if $f(\boldsymbol{x}^*)$ is close to zero, then $\boldsymbol{x}^*$ is located near the hyperplane, and so we are less certain about the class assignment for $\boldsymbol{x}^*$.

Support Vector Machines
Maximal Margin Classifier

- If our data can be perfectly separated by a hyperplane, then there will exist an infinite number of such hyperplanes:

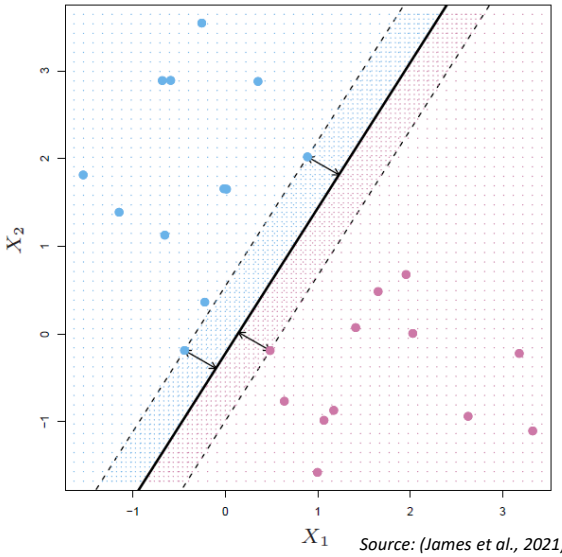


Source: (James et al., 2021)

Support Vector Machines
Maximal Margin Classifier

- A natural choice is the *maximal margin hyperplane* (also known as the *optimal separating hyperplane*), which is the separating hyperplane that is farthest from the training observations.

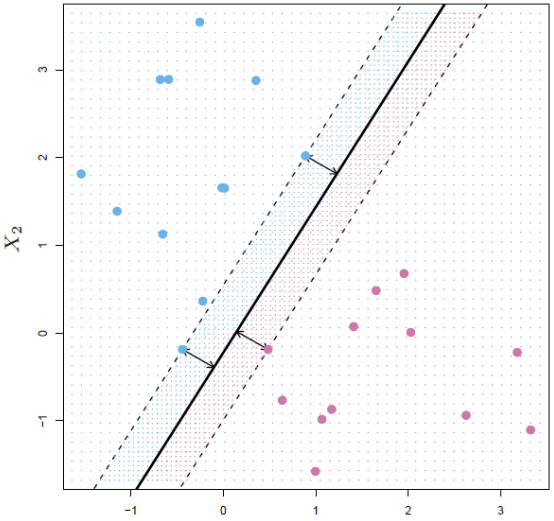
- The *margin* is the minimal (perpendicular) distance from the observations to the hiperplane.
- The *maximal margin hyperplane* is the separating hyperplane for which the margin is largest.



Source: (James et al., 2021)

Support Vector Machines
Maximal Margin Classifier

- In this example we see that 3 training observations are equidistant from the maximal margin hyperplane.
- These 3 observations are known as *support vectors*, since they "support" the maximal margin hyperplane in the sense that if these points were moved slightly then the hyperplane would move as well.
- The maximal margin hiperplane depends directly on the support vectors, but not on the other observations.



Source: (James et al., 2021)

Support Vector Machines
Maximal Margin Classifier

- The maximal margin Hyperplane is the solution to the optimization problem:

$$\max_{\beta_0,\beta_1,\ldots,\beta_n} M$$

Subject to $\sum_{j=1}^{n} \beta_j^2 = 1$

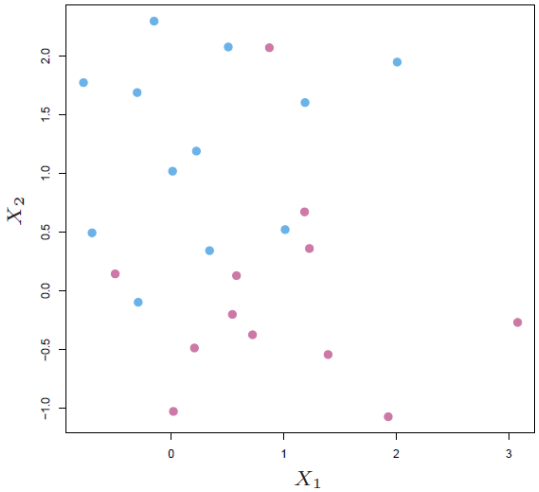
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in}) \geq M \quad \forall i = 1, \ldots, N$$

- The second constraint guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive.
- The first constraint ensures that the perpendicular distance from the i^{th} observation to the hyperplane is given by:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in})$$

Support Vector Machines
Maximal Margin Classifier

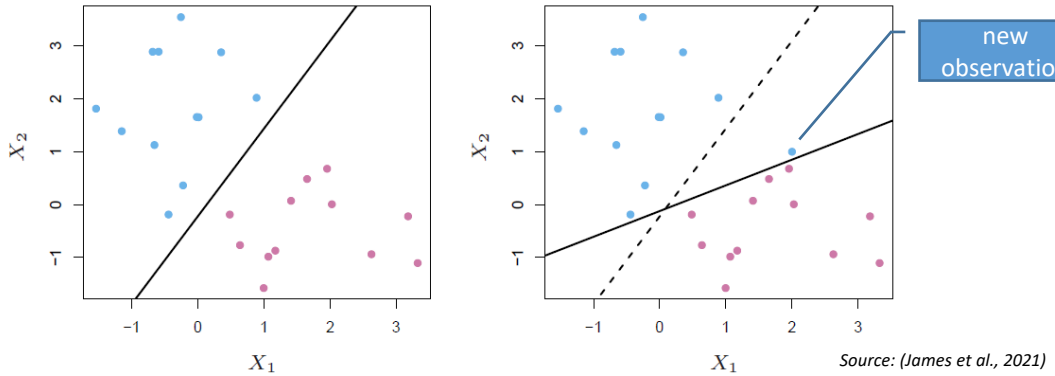
- In many practical cases the training observations can not be perfectly separated by a hyperplane (linearly non-separable problems).
- In these cases, the optimization problem has no solution with $M>0$.
- Example:



Source: (James et al., 2021)

Support Vector Machines
Support Vector Classifiers

- In fact, even if it exists, there are cases in which a classifier based on a separating hyperplane might not be desirable due to its *high sensitivity to individual observations* (→ lack of robustness and overfitting):



Source: (James et al., 2021)

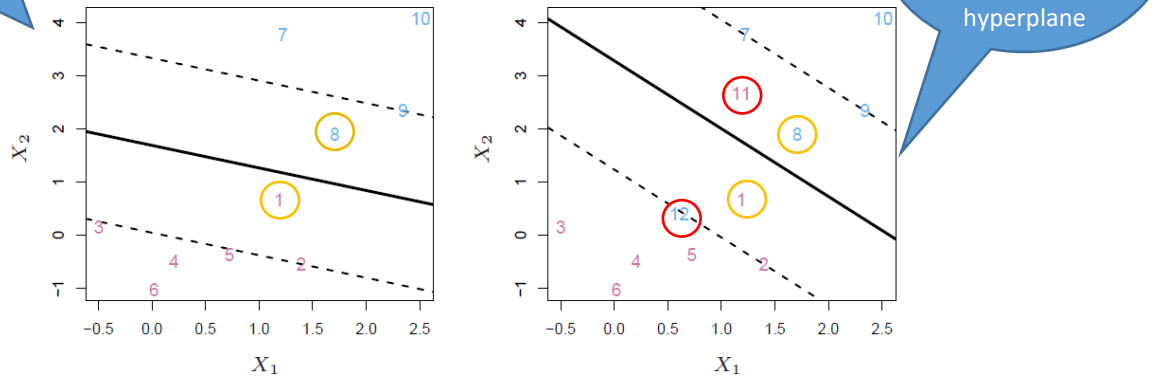
- In theses cases we might consider a classifier based on a hyperplane that does not perfectly separate the two classes, in the interest of greater *robustness* to individual observations and *better classification of most* of the training observations.

Support Vector Machines
Support Vector Classifiers

- The *Support Vector Classifier*, sometimes called *soft margin classifier*, allows a few training observations to be in the incorrect side of the margin, or even on the incorrect side of the hyperplane, in order to do a better job in classifying the remaining data:

Obs in the incorrect side of the margin

No separating hyperplane: obs in the incorrect side of the hyperplane



Source: (James et al., 2021)

Support Vector Machines
Support Vector Classifiers

- The *Support Vector Classifier* classifies a test observation depending on which side of a hyperplane it lies.
- The hyperplane is given by the solution of the optimization problem:

$$\max_{\beta_0,\beta_1,\ldots,\beta_n,\epsilon_1,\ldots,\epsilon_N} \quad M$$

Subject to $\sum_{j=1}^{n} \beta_j^2 = 1$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in}) \geq M(1 - \epsilon_i) \quad \forall i = 1, \ldots, N$$

$$\epsilon_i \geq 0 \quad \forall i = 1, \ldots, N \quad \text{and} \quad \sum_{i=1}^{N} \epsilon_i \leq C$$

where C is a non-negative *tuning parameter*, M is the width of the margin and ϵ_i are *slack variables*.

Support Vector Machines
Support Vector Classifiers

- Once the optimization problem has been solved, we classify a test observation $\boldsymbol{x}^*$ based on the sign of:

$$f(\boldsymbol{x}^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_n x_n^*$$

- The slack variable ϵ_i tells us where the *i*th observation is located:
  - If ϵ_i=0, then the *i*th observation is on the correct side of the margin
  - If ϵ_i>0, then the *i*th observation is on the wrong side of the margin
  - If ϵ_i>1, then the *i*th observation is on the wrong side of the hyperplane

Support Vector Machines
Support Vector Classifiers

- *C* bounds the sum of the ϵ_i's, so it determines the number and severity of the violations to the margin (and to the hyperplane) we will tolerate:
  - If *C*=0, then there is no budget for violations and the problem is reduced to the maximal margin hyperplane (ϵ_i=0 $\forall i$).
  - If *C*>0, then no more tan *C* observations can be on the wrong side of the hyperplane (if an observation is on the wrong side of the hyperplane then ϵ_i>1).

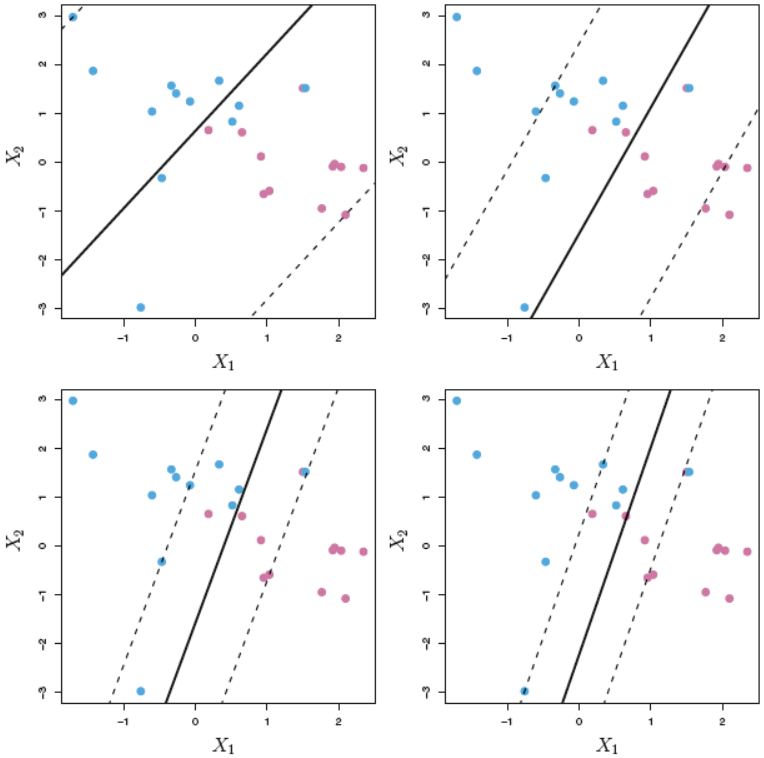
- In practice *C* is treated as a tuning parameter that is usually chosen via cross-validation.
  - When *C* is small, we seek narrow margins that are rarely violated. This amounts to a classifier that is highly fit to the data, which may have low bias but high variance.
  - When *C* is larger, the margin is larger and we allow more violations to it. This amounts to fitting the data less hard and obtaining a classifier that is potentially more biased but with lower variance.

Support Vector Machines
Support Vector Classifiers

- The optimization problem has a very interesting property: only the observations that either lie on the margin or that violate the margin will affect the hyperplane.
- Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as *support vectors*.
- When *C* is large, more observations are involved in determining the hyperplane (there are more support vectors).

Support Vector Machines
Support Vector Classifiers

Large *C*

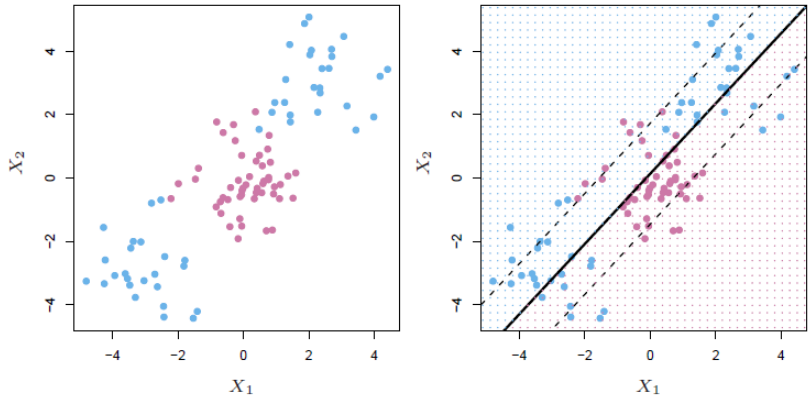


Small *C*

Source: (James et al., 2021)

Support Vector Machines
Classification with non-linear decision boundaries

- In many cases the decision boundary is not linear:



Source: (James et al., 2021)

- Non-linear classification problems can be solved in some cases by enlarging the feature space using functions of the original predictors, such as cuadratic and cubic terms.

Support Vector Machines
Classification with non-linear decision boundaries

- In the case of the support vector classifier we could do the same. For example, if we consider the quadratic terms:

$$\max_{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{n1},\beta_{n2},\epsilon_1,\ldots,\epsilon_N} M$$

Subject to $\sum_{j=1}^{n}\sum_{k=1}^{2}\beta_{jk}^2 = 1$

$$y_i\left(\beta_0 + \sum_{j=1}^{n}\beta_{j1}x_{ij} + \sum_{j=1}^{n}\beta_{j2}x_{ij}^2\right) \geq M(1-\epsilon_i) \quad \forall i = 1,\ldots,N$$

$$\epsilon_i \geq 0 \quad \forall i = 1,\ldots,N \quad \text{and} \quad \sum_{i=1}^{N}\epsilon_i \leq C$$

- We could consider many other functions of the predictors, but we could end with a huge number of features and computations would become unmanageable.

Support Vector Machines
The Support Vector Machine (SVM)

- The SVM is an extension of the support vector classifier, that results from enlarging the feature space in an efficient computational way using kernels.

- The inner product of two observations is given by:

$$\langle \boldsymbol{x}_i, \boldsymbol{x}_{i'} \rangle = \sum_{j=1}^{n} x_{ij} x_{i'j}$$

- It can be shown that the linear support vector classifier can be represented as:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{N} \alpha_i \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle$$

where there are N parameters $\alpha_i, i = 1, .., N$

Support Vector Machines
The Support Vector Machine (SVM)

- In order to evaluate the function $f(\boldsymbol{x})$ we need to compute the inner product between the new point $\boldsymbol{x}$ and each of the training points $\boldsymbol{x}_i$.
- However, it turns out that α_i is nonzero only for the support vectors in the solution. So if $\mathcal{S}$ is the collection of indices of the support vectors:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle$$

- So in representing the linear classifier $f(\boldsymbol{x})$, and in computing its coefficients, all we need are the inner products.

Support Vector Machines
The Support Vector Machine (SVM)

- If we replace the inner product with a generalization of the form:

$$\langle \boldsymbol{x}_i, \boldsymbol{x}_{i'} \rangle \longrightarrow K(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$$

where *K* is some function that we will refer to as a *kernel*.

- The *linear kernel*:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \langle \boldsymbol{x}_i, \boldsymbol{x}_{i'} \rangle = \sum_{j=1}^{n} x_{ij} x_{i'j}$$

quantifies the similarity of a pair of observations using the Pearson correlation.

- The *polynomial kernel* of degree *d*:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \left(1 + \sum_{j=1}^{n} x_{ij} x_{i'j}\right)^d$$

The Support Vector Machine (SVM)

- The *polynomial kernel* of degree *d*:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \left(1 + \sum_{j=1}^{n} x_{ij} x_{i'j}\right)^d$$

  amounts to fitting a support vector classifier in a higher-dimensional space involving polynomials of degree *d*.

- When the support vector classifier is combined with a non-linear kernel, the resulting classifier is known as a Support Vector Machine.

- In this case the function has the form:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i)$$

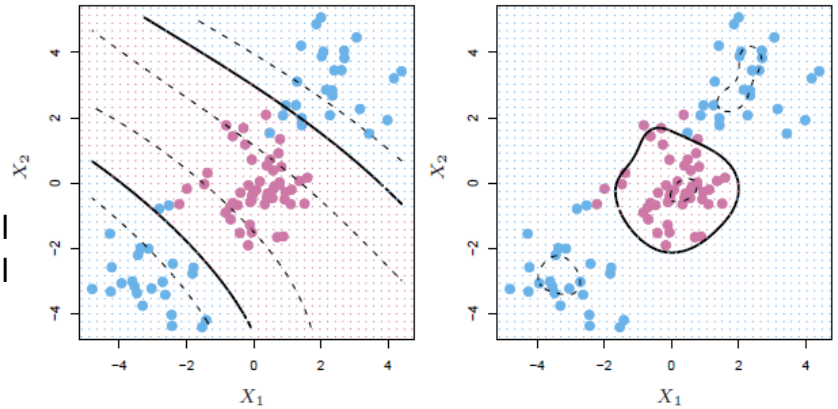
Support Vector Machines
The Support Vector Machine (SVM)

- The *radial kernel*:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = exp\left(-\sigma \sum_{j=1}^{n} \left(x_{ij} - x_{i'j}\right)^2\right)$$

where σ is a positive constant. The radial kernel has a local behavior, as only nearby training observations have an effect on the class label of a test observation.

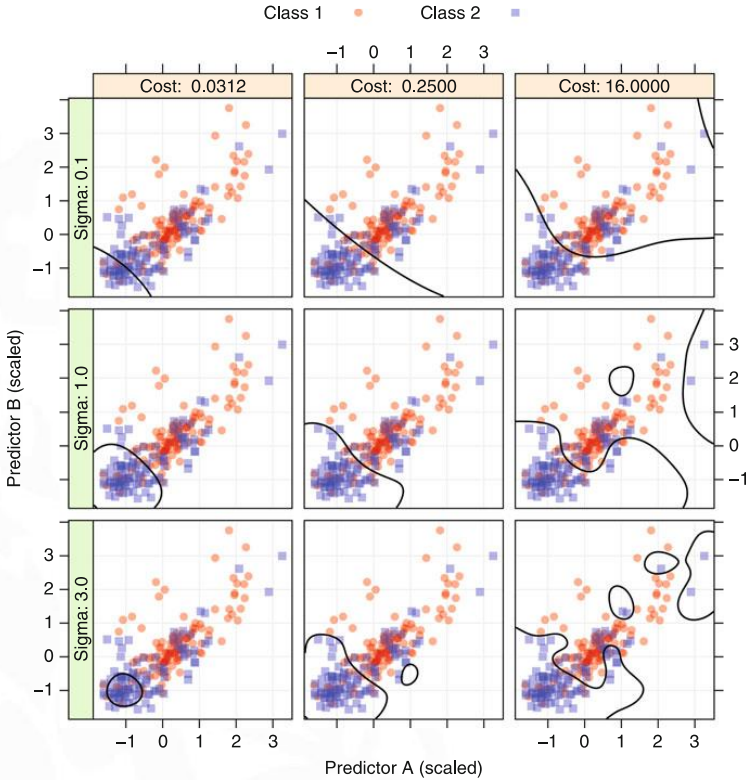
polynomial kernel



radial kernel

Source: (James et al., 2021)

Support Vector Machines
The Support Vector Machine (SVM)



Source: (Kuhn et al., 2013)

Support Vector Machines
SVM with more than two classes

- So far, we have applied SVM to binary classification problems. How can we extend SVMs to *m* classes?

- One-versus-one classification:
  - Construct $\binom{m}{2}$ SVMs, each of which compares a pair of classes
  - We classify a test observation using each of the $\binom{m}{2}$ classifiers
  - Assign the test observation to the class to which it was most frequently assigned.

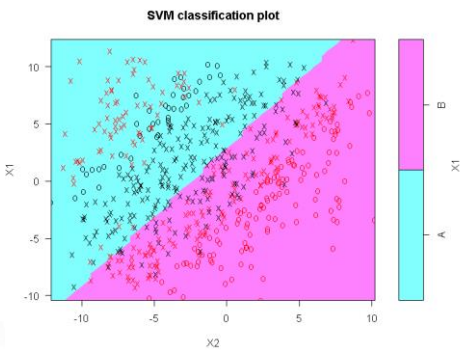
- One-versus-all classification:
  - Construct *m* SVMs, each time comparing one of the *m* classes to the remaining *m-1* classes.
  - Assign the observation to the class for which $f(\boldsymbol{x})$ is largest.

Support Vector Machines
R implementation

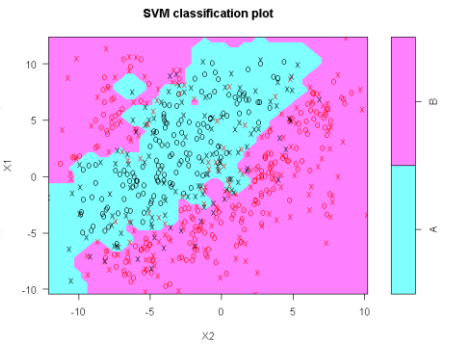
- There exists numerous packages: *e1071*, *kernlab*, *klaR*, and *svmPath*.
- Function `svm()` from *e1071* library.*Kernel* argument specifies the type of SVM.

```
> library (e1071)
> #train model
> svmfit =svm(Y ~ X1 + X2, data=fdata_train , kernel ="linear", cost =10, scale =FALSE )
> plot(svmfit , fdata_train)    #Plots the fit
> summary(svmfit)               #summary of the fit

Call:
svm(formula = Y ~ X1 + X2, data = fdata_train, kernel = "linear", cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5

Number of Support Vectors:  438

 ( 219 219 )


Number of Classes:  2

Levels:
 A B


#Compute predictions
> predict(svmfit,fdata_val)
```

Linear kernel



Radial kernel



Support Vector Machines
R implementation - caret

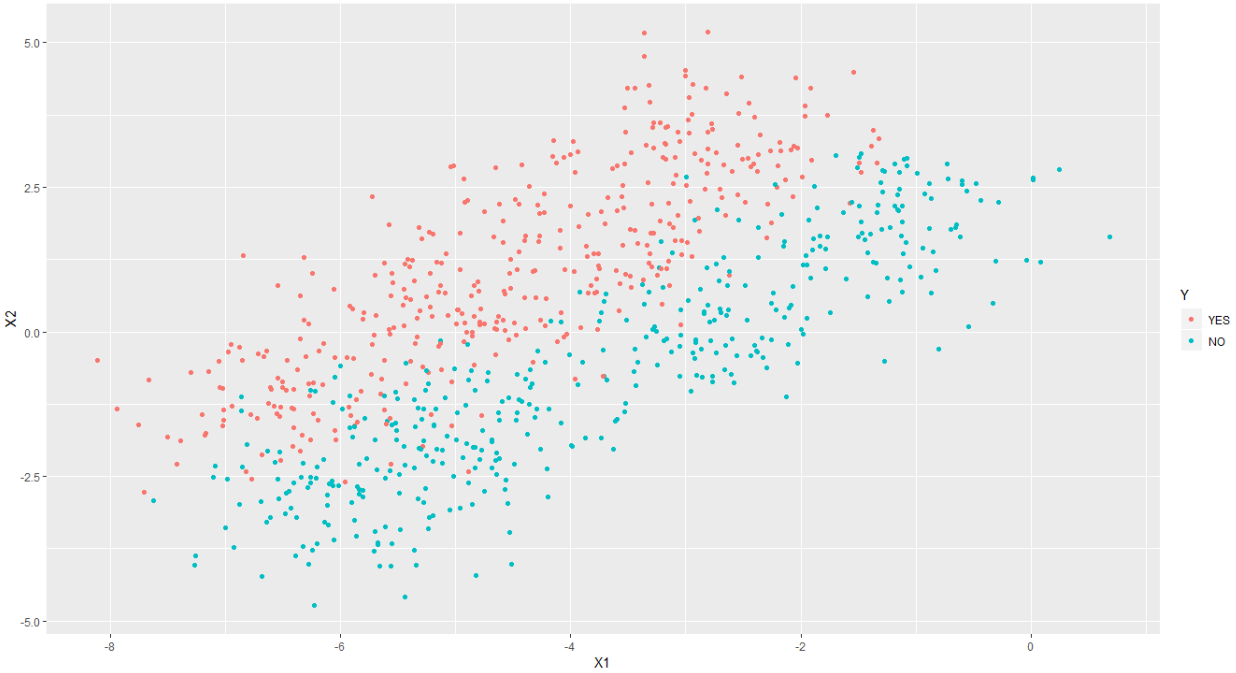
- Methods *svmLinear, svmPoly* or *svmRadial,* from *kernlab* package can be used in *caret* similar to previous models:

```
#Training the model
> modelfit = train(fdata_train[,c("X1","X2")], #Input variables
       y = fdata_train$Y, #Output variable
       method = "svmRadial", #svm model
       preProcess = c("center","scale"), #pre-processing if desired
       tuneGrid = paramsGrid, #parameters
       trControl = ctrl, #Resampling settings
       metric = "ROC") #Summary metrics

#Predictions for new data. Probabilities and classes
> Ypred_prob = predict(modelfit, type="prob" , newdata = fdata_val)
> Ypred_pred = predict(modelfit, type="raw" , newdata = fdata_val)
```

- Tuning parameters:
  - svmLinear: `C` (cost)
  - svmPoly: `degree` (polynomial degree), `scale` and `C` (cost).
  - svmRadial: `sigma` and `C` (cost).

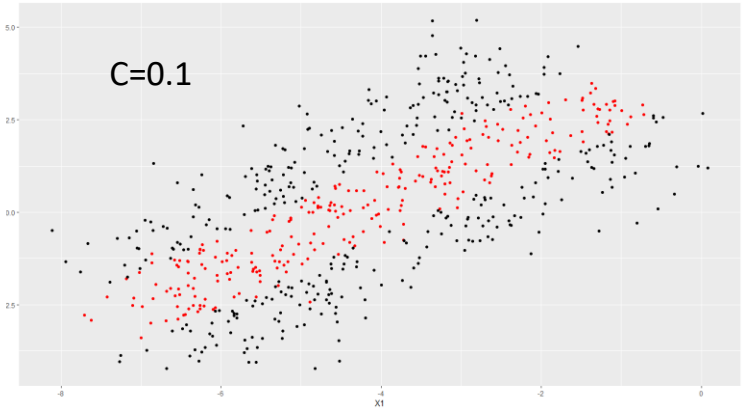
$$\begin{aligned} \text{minimize} \quad & t(w,\xi) = \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(\langle x_i, w\rangle + b) \geq 1 - \xi_i \qquad (i = 1, \ldots, N) \\ & \xi_i \geq 0 \qquad (i = 1, \ldots, N) \end{aligned}$$

Support Vector Machines
Example: ~linearly separable problem

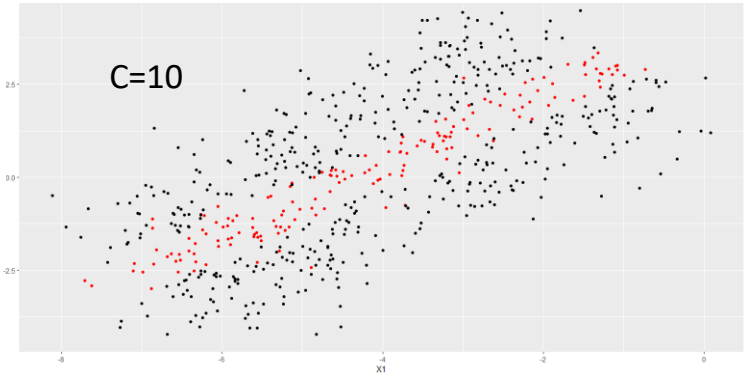


Support Vector Machines
Example: ~linearly separable problem

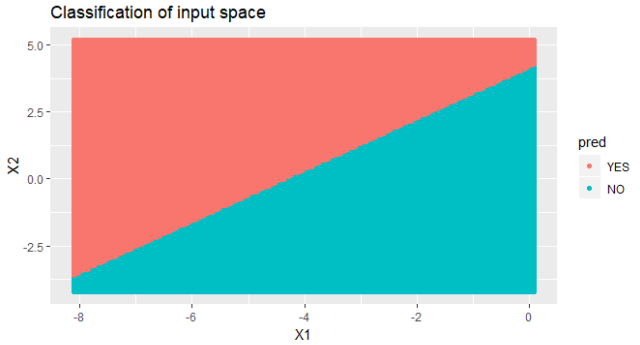
```
SV type: C-svc  (classification)
C = 0.1
Linear (vanilla) kernel function.
Number of Support Vectors : 264

Accuracy  Kappa
0.89375   0.7875
```

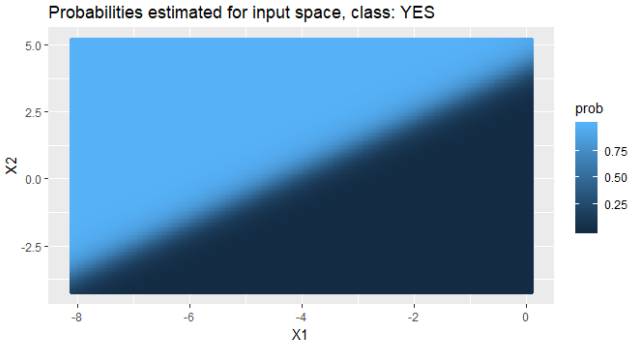


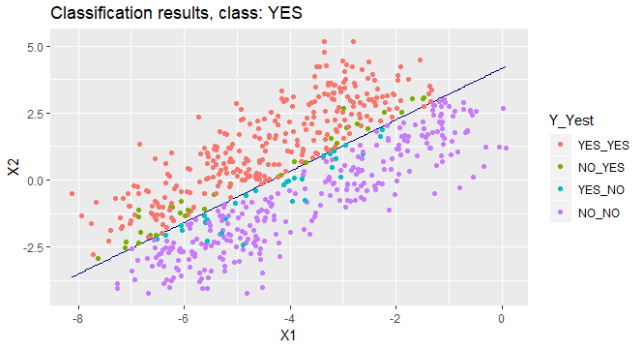

```
SV type: C-svc (classification)
C = 10
Linear (vanilla) kernel function.
Number of Support Vectors : 167

Accuracy   Kappa
0.8921875  0.784375
```

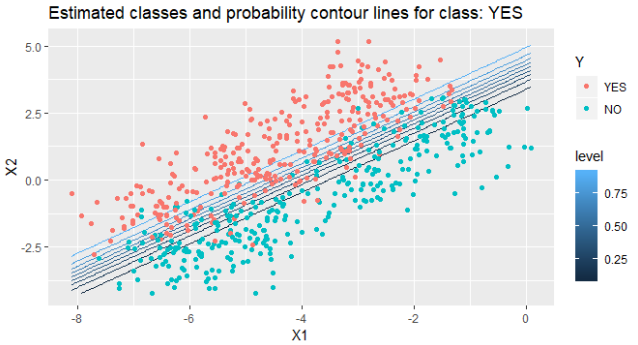


Support Vector Machines
Example: ~linearly separable problem

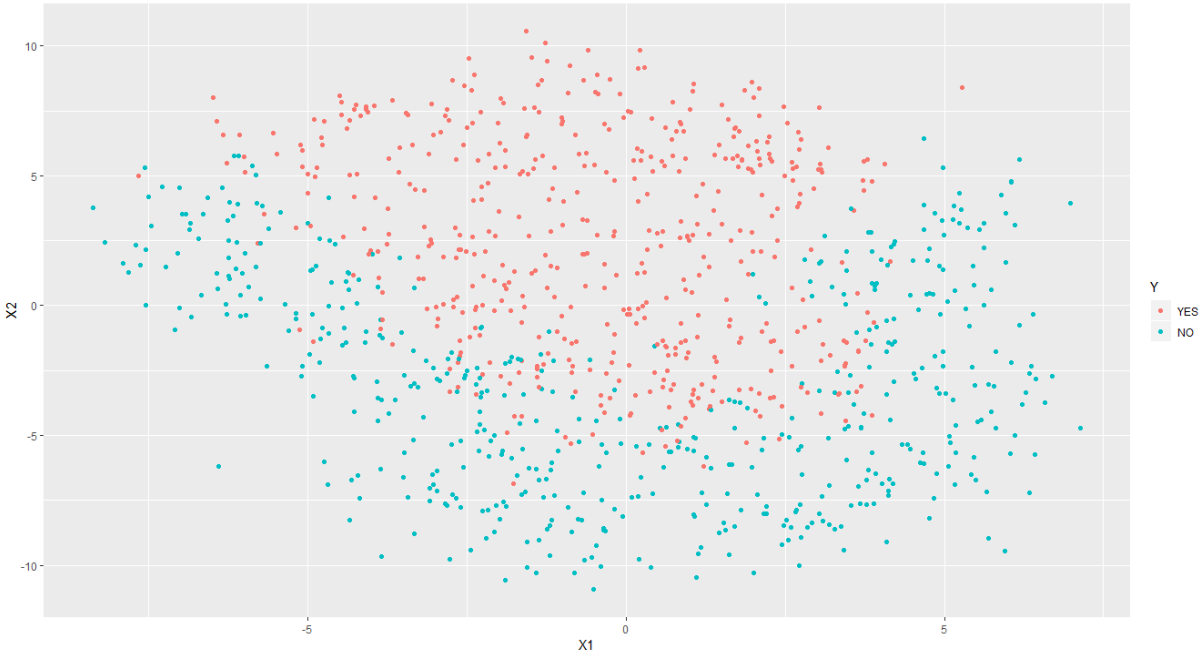






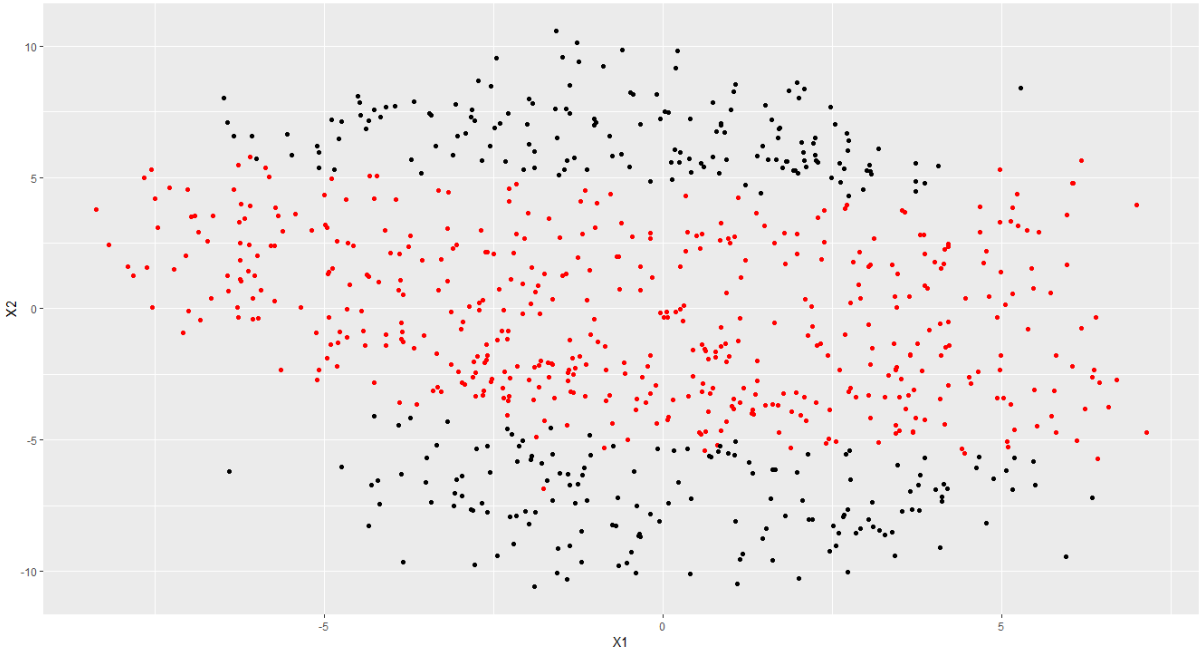


Support Vector Machines
Example: non-linearly separable problem



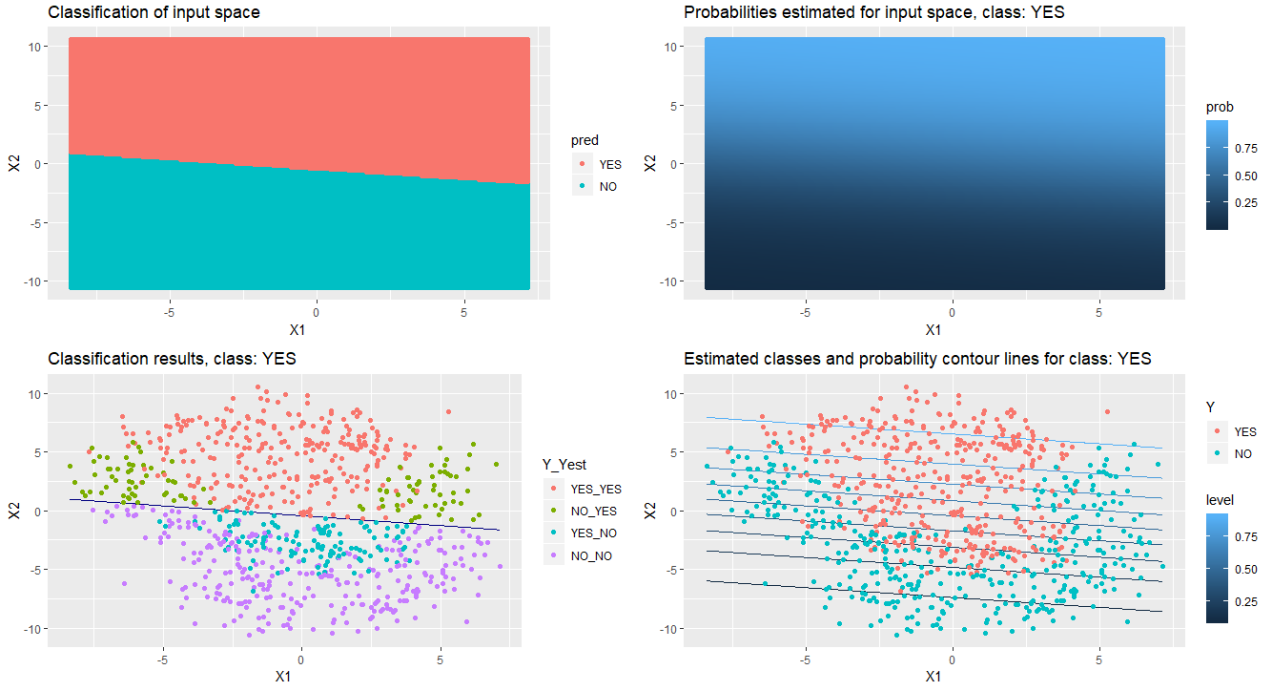
Support Vector Machines
Example: non-linearly separable problem

Linear SVM: Support vectors



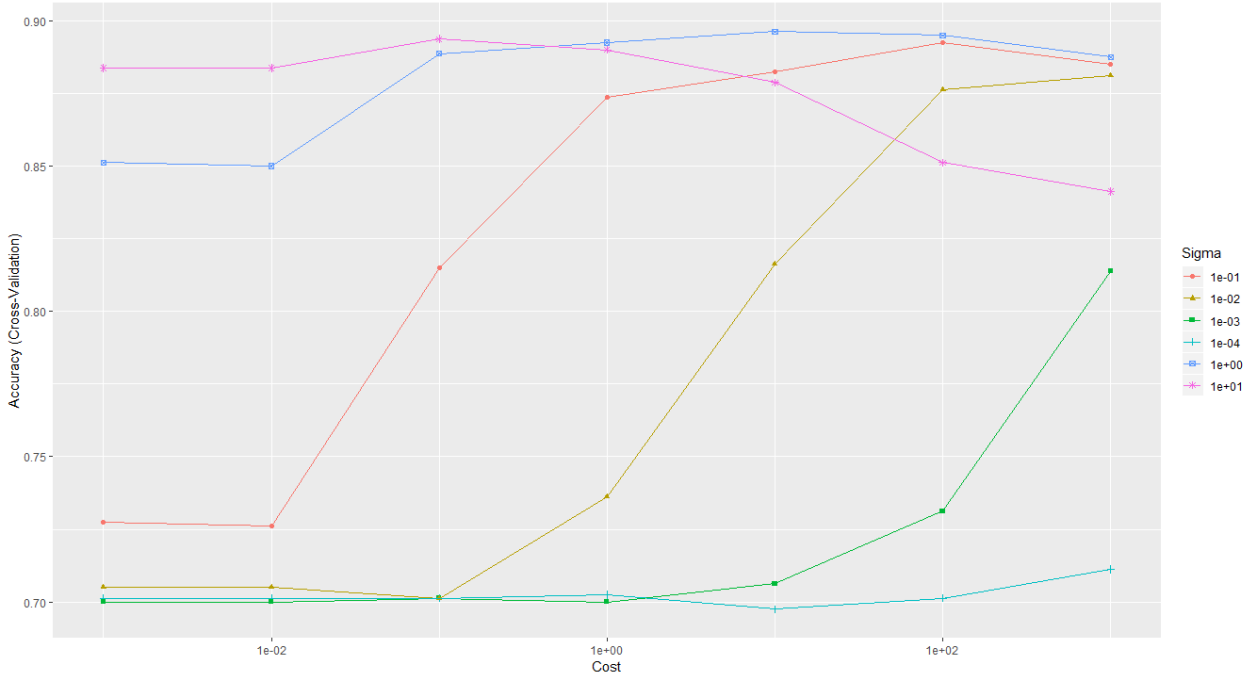
Support Vector Machines
Example: non-linearly separable problem

Linear SVM: decision boundary



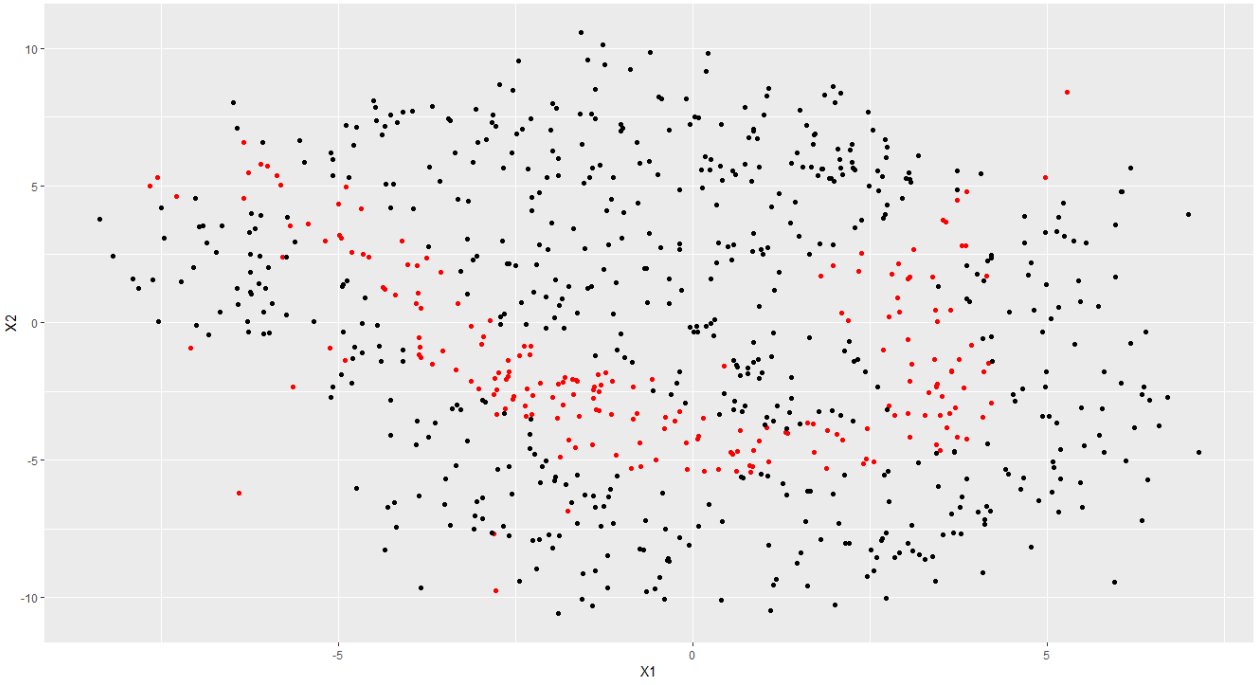
Support Vector Machines
Example: non-linearly separable problem

Radial SVM: Determination of the optimal parameters



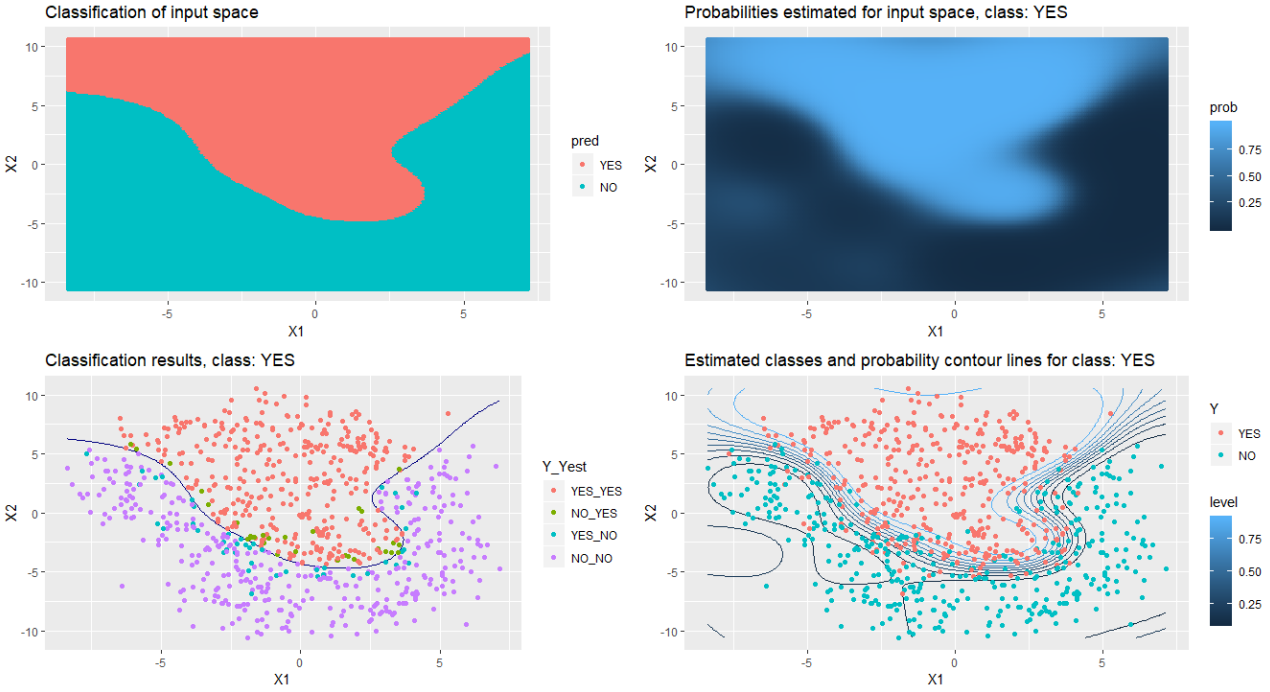
Support Vector Machines
Example: non-linearly separable problem

Radial SVM: Support vectors



Support Vector Machines
Example: non-linearly separable problem

Radial SVM: Decision boundary



4

Bibliography

Bibliography

- C. Bishop (2007). *Pattern Recognition and Machine Learning*. Springer.
- R. Duda, P. Hart & D. Stork (2000). *Pattern Classification*. 2nd Ed. Wiley-Interscience.
- G. James, D. Witten, T. Hastie & R. Tibshirani (2021). An Introduction to Statistical Learning with Applications in R. Second Edition. Springer. See https://www.statlearning.com/)
- M. Kuhn, K. Johnson (2013). *Applied predictive modeling.* Springer.
- I. Narsky, F. Porter (2014). *Statistical Analysis Techniques in Particle Physics*. Wiley.
- D. Cook (2017). *Practical Machine Learning with H2O*. O'Reilly.