Predictive Analytics

L. Torgo

ltorgo@fc.up.pt
Faculdade de Ciências / LIAAD-INESC TEC, LA
Universidade do Porto

Jun, 2017

NYU STERN
MS in Business Analytics

What is Prediction?

Definition

- Prediction (forecasting) is the ability to anticipate the future.
- Prediction is possible if we assume that there is some regularity in what we observe, i.e. if the observed events are not random.

Example

Medical Diagnosis: given an historical record containing the symptoms observed in several patients and the respective diagnosis, try to forecast the correct diagnosis for a new patient for which we know the symptoms.

Prediction Models

- Are obtained on the basis of the assumption that there is an unknown mechanism that maps the characteristics of the observations into conclusions/diagnoses. The goal of prediction models is to discover this mechanism.
  - Going back to the medical diagnosis what we want is to know how symptoms influence the diagnosis.
- Have access to a data set with “examples” of this mapping, e.g. this patient had symptoms x, y, z and the conclusion was that he had disease p
- Try to obtain, using the available data, an approximation of the unknown function that maps the observation descriptors into the conclusions, i.e. $Prediction = f(Descriptors)$

“Entities” involved in Predictive Modelling

- Descriptors of the observation:
  set of variables that describe the properties (features, attributes) of the cases in the data set
- Target variable:
  what we want to predict/conclude regards the observations
- The goal is to obtain an approximation of the function $Y = f(X_1, X_{,2}, \cdots, X_p)$, where Y is the target variable and $X_1, X_{,2}, \cdots, X_p$ the variables describing the characteristics of the cases.
- It is assumed that Y is a variable whose values depend on the values of the variables which describe the cases. We just do not know how!
- The goal of the modelling techniques is thus to obtain a good approximation of the unknown function $f()$

How are the Models Used?

Predictive models have two main uses:

1. Prediction
   use the obtained models to make predictions regards the target variable of new cases given their descriptors.
2. Comprehensibility
   use the models to better understand which are the factors that influence the conclusions.

Types of Prediction Problems

- Depending on the type of the target variable (Y) we may be facing two different types of prediction models:
  1. Classification Problems - the target variable Y is nominal
     e.g. medical diagnosis - given the symptoms of a patient try to predict the diagnosis
  2. Regression Problems - the target variable Y is numeric
     e.g. forecast the market value of a certain asset given its characteristics

Examples of Prediction Problems

- Classification
  - A marketing department of a bank stores information on previous telephone contacts with its costumers for selling new products.
  - For each client a series of personal information is stored together with the results of the last contact (if it bought or not the product).
  - The goal of this application is to obtain a predictive model that can forecast whether a client will buy or not a new product before the phone contact takes place.

Case ID	age	job	marital	education	default	balance	housing	loan	y
1	30	unemployed	married	primary	no	1787	no	no	no
2	33	services	married	secondary	no	4789	yes	yes	no
3	35	management	single	tertiary	no	1350	yes	no	no
4	30	management	married	tertiary	no	1476	yes	yes	no
5	59	blue-collar	married	secondary	no	0	yes	no	no
6	35	management	single	tertiary	no	747	no	no	no

- Regression
  - On the previous car insurance data try to forecast the normalized losses of a car based on its characteristics.

Types of Prediction Models

- There are many techniques that can be used to obtain prediction models based on a data set.
- Independently of the pros and cons of each alternative, all have some key characteristics:
  1. They assume a certain functional form for the unknown function $f()$
  2. Given this assumed form the methods try to obtain the best possible model based on:
     1. the given data set
     2. a certain preference criterion that allows comparing the different alternative model variants

Functional Forms of the Models

- There are many variants. Examples include:
  - Mathematical formulae - e.g. linear discriminants
  - Logical approaches - e.g. classification or regression trees, rules
  - Probabilistic approaches - e.g. naive Bayes
  - Other approaches - e.g. neural networks, SVMs, etc.
  - Sets of models (ensembles) - e.g. random forests, adaBoost
- These different approaches entail different compromises in terms of:
  - Assumptions on the unknown form of dependency between the target and the predictors
  - Computational complexity of the search problem
  - Flexibility in terms of being able to approximate different types of functions
  - Interpretability of the resulting model
  - etc.

NYU | STERN
MS in Business Analytics

Which Models or Model Variants to Use?

- This question is often known as the Model Selection problem
- The answer depends on the goals of the final user - i.e. the Preference Biases of the user
- Establishing which are the preference criteria for a given prediction problem allows to compare and select different models or variants of the same model

NYU | STERN
MS in Business Analytics

Evaluation Metrics

Classification Problems

The setting

- Given data set $\{< \mathbf{x}_i, y_i >\}_{i=1}^N$, where $\mathbf{x}_i$ is a feature vector $< x_1, x_2, \cdots, x_p >$ and $y_i \in \mathcal{Y}$ is the value of the nominal variable Y
- There is an unknown function $Y = f(\mathbf{x})$

The approach

- Assume a functional form $h_\theta(\mathbf{x})$ for the unknown function $f()$, where θ are a set of parameters
- Assume a preference criterion over the space of possible parameterizations of $h()$
- Search for the “optimal” $h()$ according to the criterion and the data set

Classification Error

Error Rate

- Given a set of test cases N_{test} we can obtain the predictions for these cases using some classification model.
- The *Error Rate* ($L_{0/1}$) measures the proportion of these predictions that are incorrect.
- In order to calculate the error rate we need to obtain the information on the true class values of the N_{test} cases.

Classification Error

Error Rate

- Given a test set for which we know the true class the error rate can be calculated as follows,

$$L_{0/1} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} I(\hat{h}_\theta(\mathbf{x}_i), y_i)$$

where $I()$ is an indicator function such that $I(x, y) = 0$ if $x = y$ and 1 otherwise; and $\hat{h}_\theta(\mathbf{x}_i)$ is the prediction of the model being evaluated for the test case i that has as true class the value y_i.

Confusion Matrices

- A square $nc \times nc$ matrix, where nc is the number of class values of the problem
- The matrix contains the number of times each pair (ObservedClass,PredictedClass) occurred when testing a classification model on a set of cases

		Pred.		
		c_1	c_2	c_3
Obs.	c_1	n_{c_1,c_1}	n_{c_1,c_2}	n_{c_1,c_3}
	c_2	n_{c_2,c_1}	n_{c_2,c_2}	n_{c_2,c_3}
	c_3	n_{c_3,c_1}	n_{c_3,c_2}	n_{c_3,c_3}

- The error rate can be calculated from the information on this table.

An Example in R

```
trueVals <- c("c1","c1","c2","c1","c3","c1","c2","c3","c2","c3")
preds <- c("c1","c2","c1","c3","c3","c1","c1","c3","c1","c2")
confMatrix <- table(trueVals,preds)
confMatrix

##          preds
## trueVals c1 c2 c3
##       c1  2  1  1
##       c2  3  0  0
##       c3  0  1  2

errorRate <- 1-sum(diag(confMatrix))/sum(confMatrix)
errorRate

## [1] 0.6
```

Cost-Sensitive Applications

- In the error rate one assumes that all errors and correct predictions have the same value
- This may not be adequate for some applications

Cost/benefit Matrices

Table where each entry specifies the cost (negative benefit) or benefit of each type of prediction

		Pred.		
		c_1	c_2	c_3
Obs.	c_1	$B_{1,1}$	$C_{1,2}$	$C_{1,3}$
	c_2	$C_{2,1}$	$B_{2,2}$	$C_{2,3}$
	c_3	$C_{3,1}$	$C_{3,2}$	$B_{3,3}$

- Models are then evaluated by the total balance of their predictions, i.e. the sum of the benefits minus the costs.

An Example in R

```
trueVals <- c("c1","c1","c2","c1","c3","c1","c2","c3","c2","c3")
preds <- c("c1","c2","c1","c3","c3","c1","c1","c3","c1","c2")
confMatrix <- table(trueVals,preds)
costMatrix <- matrix(c(10,-2,-4,-2,30,-3,-5,-6,12),ncol=3)
colnames(costMatrix) <- c("predC1","predC2","predC3")
rownames(costMatrix) <- c("obsC1","obsC2","obsC3")
costMatrix

##       predC1 predC2 predC3
## obsC1     10     -2     -5
## obsC2     -2     30     -6
## obsC3     -4     -3     12

utilityPreds <- sum(confMatrix*costMatrix)
utilityPreds

## [1] 28
```

Predicting a Rare Class

E.g. predicting outliers

- Problems with two classes
- One of the classes is much less frequent and it is also the most relevant

		Preds.	
		Pos	Neg
Obs.	Pos	True Positives (TP)	False Negatives (FN))
	Neg	False Positives (FP)	True Negatives (TN)

Precision and *Recall*

		Preds.	
		P	N
Obs.	P	TP	FN
	N	FP	TN

- *Precision* - proportion of the signals (events) of the model that are correct

$$Prec = \frac{TP}{TP + FP}$$

- *Recall* - proportion of the real events that are captured by the model

$$Rec = \frac{TP}{TP + FN}$$

Precision and *Recall*

Examples

		Preds.	
		P	N
Obs.	P	2	2
	N	1	1

$$Precision = \frac{TP}{TP+FP} = \frac{2}{2+1} = 0.667$$

$$Recall = \frac{TP}{TP+FN} = \frac{2}{2+2} = 0.5$$

$$ErrorRate = \frac{2+1}{2+2+1+1} = 0.5$$

The F-Measure

Combining Precision and Recall into a single measure

- Sometimes it is useful to have a single measure - e.g. optimization within a search procedure
- Maximizing one of them is easy at the cost of the other (it is easy to have 100% recall - always predict “P”).
- What is difficult is to have both of them with high values
- The F-measure is a statistic that is based on the values of precision and recall and allows establishing a trade-off between the two using a user-defined parameter (β),

$$F_\beta = \frac{(\beta^2+1)\cdot Prec \cdot Rec}{\beta^2 \cdot Prec + Rec}$$

where β controls the relative importance of *Prec* and *Rec*. If $\beta = 1$ then *F* is the harmonic mean between *Prec* and *Rec*; When $\beta \to 0$ the weight of *Rec* decreases. When $\beta \to \infty$ the weight of *Prec* decreases.

Regression Problems

The setting

- Given data set $\{< \mathbf{x}_i, y_i >\}_{i=1}^{N}$, where $\mathbf{x}_i$ is a feature vector $< x_1, x_2, \cdots, x_p >$ and $y_i \in \Re$ is the value of the numeric variable Y
- There is an unknown function $Y = f(\mathbf{x})$

The approach

- Assume a functional form $h_\theta(\mathbf{x})$ for the unknown function $f()$, where θ are a set of parameters
- Assume a preference criterion over the space of possible parameterizations of $h()$
- Search for the "optimal" $h()$ according to the criterion and the data set

Measuring Regression Error

Mean Squared Error

- Given a set of test cases N_{test} we can obtain the predictions for these cases using some regression model.
- The *Mean Squared Error* (*MSE*) measures the average squared deviation between the predictions and the true values.
- In order to calculate the value of *MSE* we need to have both the predicitons and the true values of the N_{test} cases.

Measuring Regression Error

Mean Squared Error (cont.)

- If we have such information the *MSE* can be calculated as follows,

$$MSE = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ is the prediction of the model under evaluation for the case i and y_i the respective true target variable value.

- Note that the *MSE* is measured in a unit that is squared of the original variable scale. Because of the this is sometimes common to use the *Root Mean Squared Error* (*RMSE*), defined as $RMSE = \sqrt{MSE}$

Measuring Regression Error

Mean Absolute Error

- The *Mean Absolute Error* (*MAE*) measures the average absolute deviation between the predictions and the true values.
- The value of the *MAE* can be calculated as follows,

$$MAE = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} |\hat{y}_i - y_i|$$

where $\hat{y}_i$ is the prediction of the model under evaluation for the case i and y_i the respective true target variable value.

- Note that the *MAE* is measured in the same unit as the original variable scale.

Relative Error Metrics

- Relative error metrics are unit less which means that their scores can be compared across different domains.
- They are calculated by comparing the scores of the model under evaluation against the scores of some baseline model.
- The relative score is expected to be a value between 0 and 1, with values nearer (or even above) 1 representing performances as bad as the baseline model, which is usually chosen as something too naive.

Relative Error Metrics (cont.)

- The most common baseline model is the constant model consisting of predicting for all test cases the average target variable value calculated in the training data.
- The *Normalized Mean Squared Error* (*NMSE*) is given by,

$$NMSE = \frac{\sum_{i=1}^{N_{test}} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{N_{test}} (\bar{y} - y_i)^2}$$

- The *Normalized Mean Absolute Error* (*NMAE*) is given by,

$$NMAE = \frac{\sum_{i=1}^{N_{test}} |\hat{y}_i - y_i|}{\sum_{i=1}^{N_{test}} |\bar{y} - y_i|}$$

Relative Error Metrics (cont.)

- The *Mean Average Percentage Error* (*MAPE*) is given by,

$$MAPE = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{|\hat{y}_i - y_i|}{y_i}$$

- The *Correlation* between the predictions and the true values ($\rho_{\hat{y},y}$) is given by,

$$\rho_{\hat{y},y} = \frac{\sum_{i=1}^{N_{test}} (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N_{test}} (\hat{y}_i - \bar{\hat{y}})^2 \sum_{i=1}^{N_{test}} (y_i - \bar{y})^2}}$$

An Example in R

```r
trueVals <- c(10.2,-3,5.4,3,-43,21,
              32.4,10.4,-65,23)
preds <-  c(13.1,-6,0.4,-1.3,-30,1.6,
            3.9,16.2,-6,20.4)
mse <- mean((trueVals-preds)^2)
mse

## [1] 494

rmse <- sqrt(mse)
rmse

## [1] 22.23

mae <- mean(abs(trueVals-preds))
mae

## [1] 14.35
```

```r
nmse <- sum((trueVals-preds)^2) /
        sum((trueVals-mean(trueVals))^2)
nmse

## [1] 0.5916

nmae <- sum(abs(trueVals-preds)) /
        sum(abs(trueVals-mean(trueVals)))
nmae

## [1] 0.6563

mape <- mean(abs(trueVals-preds)/trueVals)
mape

## [1] 0.2908

corr <- cor(trueVals,preds)
corr

## [1] 0.6745
```

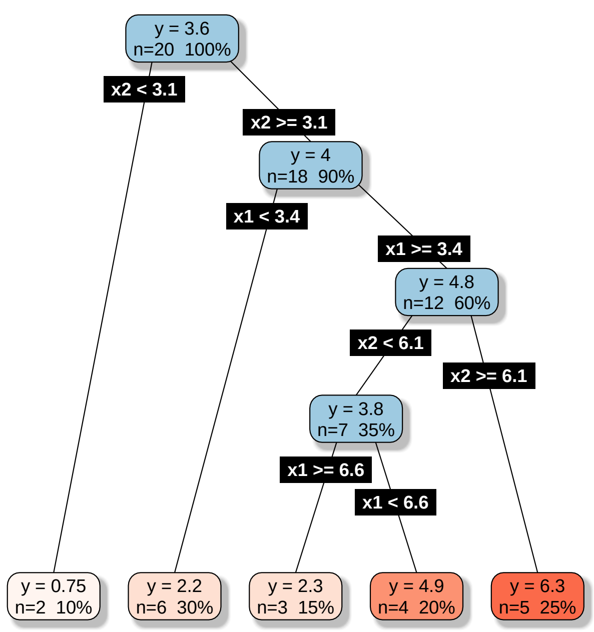
Tree-based Models

Tree-based Models

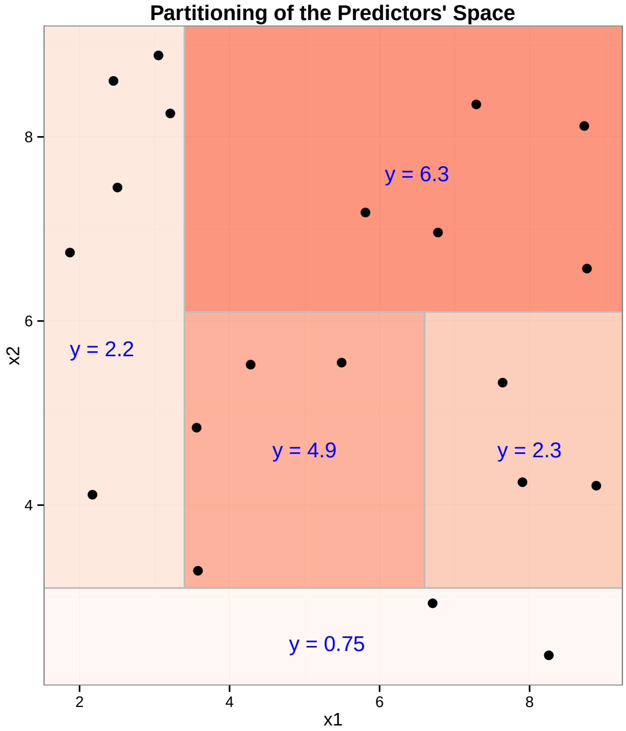
- Tree-based models (both classification and regression trees) are models that provide as result a model based on logical tests on the input variables
- These models can be seen as a partitioning of the input space defined by the input variables
- This partitioning is defined based on carefully chosen logical tests on these variables
- Within each partition all cases are assigned the same prediction (either a class label or a numeric value)
- Tree-based models are known by their (i) computational efficiency; (ii) interpretable models; (iii) embedded variable selection; (iv) embedded handling of unknown variable values and (v) few assumptions on the unknown function being approximated

An Example of Trees Partitioning

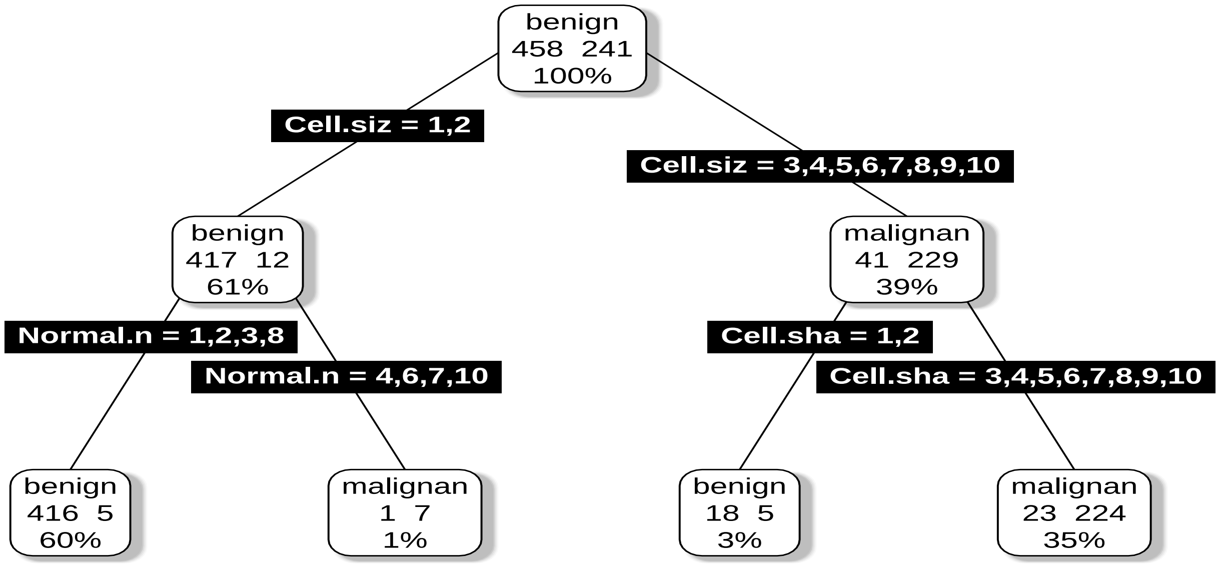
Example of a Regression Tree



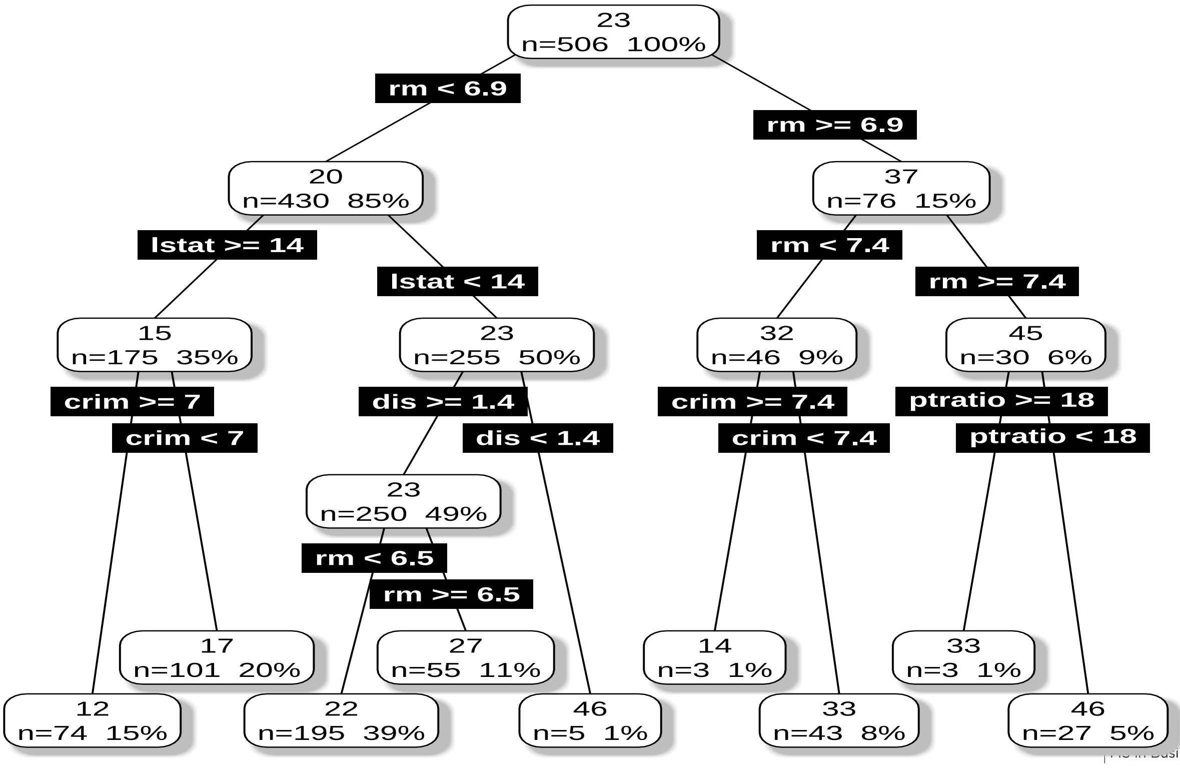
Partitioning of the Predictors' Space



An Example of a Classification Tree



An Example of a Regression Tree



Tree-based Models

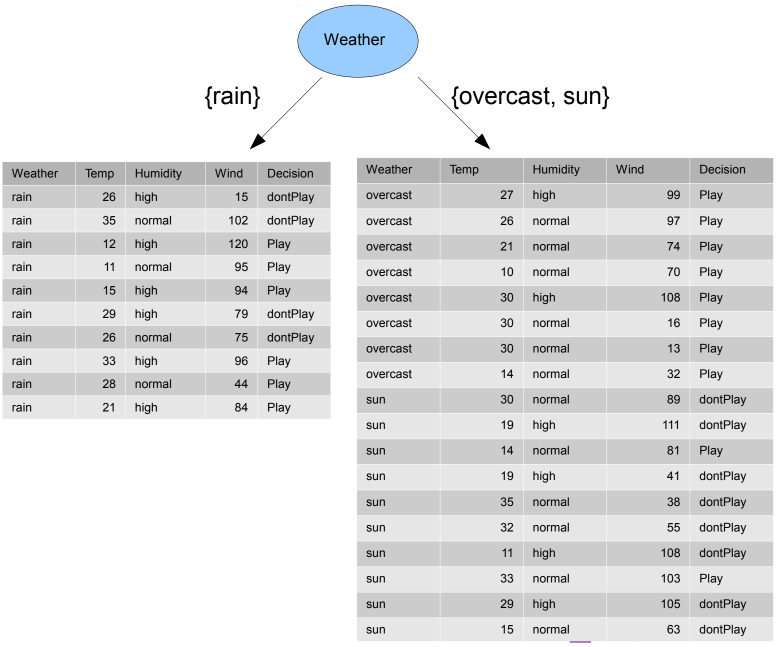
- Most tree-based models are binary trees with logical tests on each node
- Tests on numerical predictors take the form $x_i < \alpha$, with $\alpha \in \Re$
- Tests on nominal predictors take the form $x_j \in \{v_1, \cdots, v_m\}$
- Each path from the top (root) node till a leaf can be seen as a logical condition defining a region of the predictors space.
- All observations “falling” on a leaf will get the same prediction
  - the majority class of the training cases in that leaf for classification trees
  - the average value of the target variable for regression trees
- The prediction for a new test case is easily obtained by following a path from the root till a leaf according to the case predictors values

The Recursive Partitioning Algorithm

```
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

    if <TERMINATION CRITERION> then
        Return a leaf node with the majority class in $D$
    else
        $t \leftarrow$ new tree node
        $t.split \leftarrow$ <FIND THE BEST PREDICTORS TEST>
        $t.leftNode \leftarrow$ RecursivePartitioning($\mathbf{x} \in D : \mathbf{x} \vDash t.split$)
        $t.rightNode \leftarrow$ RecursivePartitioning($\mathbf{x} \in D : \mathbf{x} \nvDash t.split$)
        Return the node $t$
    end if
end function
```

The Recursive Partitioning Algorithm - an example

Weather	Temp.	Humidity	Wind	Decision
rain	26	high	15	dontPlay
rain	35	normal	102	dontPlay
overcast	27	high	99	Play
overcast	26	normal	97	Play
rain	12	high	120	Play
overcast	21	normal	74	Play
sun	30	normal	89	dontPlay
sun	19	high	111	dontPlay
sun	14	normal	81	Play
overcast	10	normal	70	Play
rain	11	normal	95	Play
rain	15	high	94	Play
sun	19	high	41	dontPlay
sun	35	normal	38	dontPlay
rain	29	high	79	dontPlay
rain	26	normal	75	dontPlay
overcast	30	high	108	Play
overcast	30	normal	16	Play
rain	33	high	96	Play
overcast	30	normal	13	Play
sun	32	normal	55	dontPlay
sun	11	high	108	dontPlay
sun	33	normal	103	Play
overcast	14	normal	32	Play
rain	28	normal	44	Play
rain	21	high	84	Play
sun	29	high	105	dontPlay



Weather	Temp	Humidity	Wind	Decision
rain	26	high	15	dontPlay
rain	35	normal	102	dontPlay
rain	12	high	120	Play
rain	11	normal	95	Play
rain	15	high	94	Play
rain	29	high	79	dontPlay
rain	26	normal	75	dontPlay
rain	33	high	96	Play
rain	28	normal	44	Play
rain	21	high	84	Play

Weather	Temp	Humidity	Wind	Decision
overcast	27	high	99	Play
overcast	26	normal	97	Play
overcast	21	normal	74	Play
overcast	10	normal	70	Play
overcast	30	high	108	Play
overcast	30	normal	16	Play
overcast	30	normal	13	Play
overcast	14	normal	32	Play
sun	30	normal	89	dontPlay
sun	19	high	111	dontPlay
sun	14	normal	81	Play
sun	19	high	41	dontPlay
sun	35	normal	38	dontPlay
sun	32	normal	55	dontPlay
sun	11	high	108	dontPlay
sun	33	normal	103	Play
sun	29	high	105	dontPlay
sun	15	normal	63	dontPlay

The Recursive Partitioning Algorithm (cont.)

Key Issues of the RP Algorithm

- When to stop growing the tree - termination criterion
- Which value to put on the leaves
- How to find the best split test

The Recursive Partitioning Algorithm (cont.)

When to Stop?

Too large trees tend to overfit the training data and will perform badly on new data - a question of reliability of error estimates

Which value?

Should be the value that better represents the cases in the leaves

What are the good tests?

A test is good if it is able to split the cases of sample in such a way that they form partitions that are “purer” than the parent node

Classification vs Regression Trees

- They are both grown using the Recursive Partitioning algorithm
- The main difference lies on the used preference criterion
- This criterion has impact on:
  - The way the best test for each node is selected
  - The way the tree avoids over fitting the training sample
- Classification trees typically use criteria related to error rate (e.g. the Gini index, the Gain ratio, entropy, etc.)
- Regression trees typically use the least squares error criterion

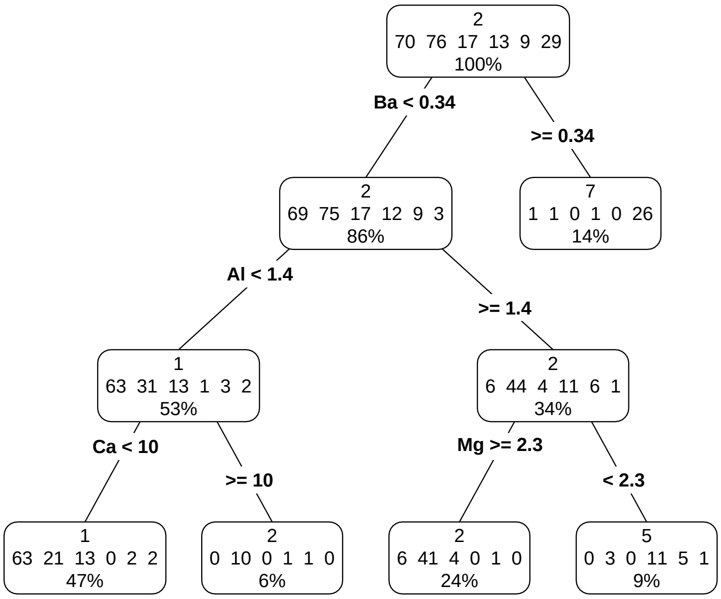
Classification and Regression Trees in R

The package `rpart`

- Package `rpart` implements most of the ideas of the system CART that was described in the book “Classification and Regression Trees” by Breiman and colleagues
- This system is able to obtain classification and regression trees.
- For classification trees it uses the Gini score to grow the trees and it uses Cost-Complexity post-pruning to avoid over fitting
- For regression trees it uses the least squares error criterion and it uses Error-Complexity post-pruning to avoid over fitting
- On package `DMwR` you may find function `rpartXse()` that grows and prunes a tree in a way similar to CART using the above infra-structure

Illustration using a classification task - *Glass*

```
library(DMwR)
library(rpart.plot)
data(Glass,package='mlbench')
ac <- rpartXse(Type  ~ .,Glass)
prp(ac,type=4,extra=101)
```



How to use the trees for Predicting?

```
tr <- Glass[1:200,]
ts <- Glass[201:214,]
ac <- rpartXse(Type  ~ .,tr)
predict(ac,ts)

##     1         2         3          5 6         7
## 201 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 202 0 0.3636364 0.6363636 0.00000000 0 0.0000000
## 203 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 204 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 205 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 206 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 207 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 208 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 209 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 210 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 211 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 212 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 213 0 0.0000000 0.0000000 0.09090909 0 0.9090909
## 214 0 0.0000000 0.0000000 0.09090909 0 0.9090909
```

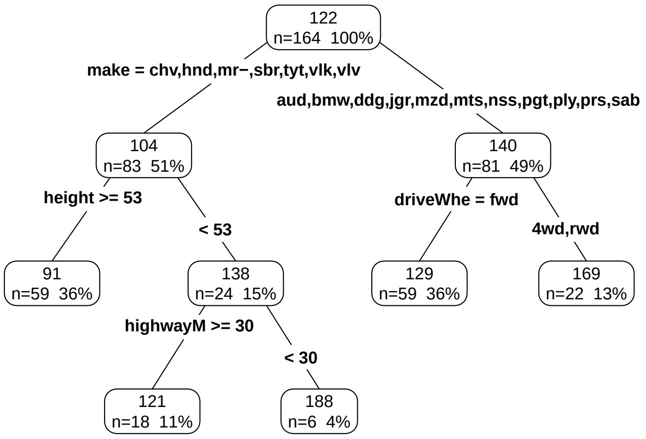
How to use the trees for Predicting? (cont.)

```
predict(ac,ts,type='class')

## 201 202 203 204 205 206 207 208 209 210 211 212 213 214
##   7   3   7   7   7   7   7   7   7   7   7   7   7   7
## Levels: 1 2 3 5 6 7

ps <- predict(ac,ts,type='class')
table(ps,ts$Type)

##
## ps   1  2  3  5  6  7
##   1  0  0  0  0  0  0
##   2  0  0  0  0  0  0
##   3  0  0  0  0  0  1
##   5  0  0  0  0  0  0
##   6  0  0  0  0  0  0
##   7  0  0  0  0  0 13

mc <- table(ps,ts$Type)
err <- 100*(1-sum(diag(mc))/sum(mc))
err

## [1] 7.142857
```

Illustration using a regression task

Forecasting Normalized Losses

```
library(DMwR)
library(rpart.plot)
load('carInsurance.Rdata')
d <- ins[,-1]
ar <- rpartXse(normLoss ~ .,d)
prp(ar,type=4,extra=101)
```



How to use the trees for Predicting?

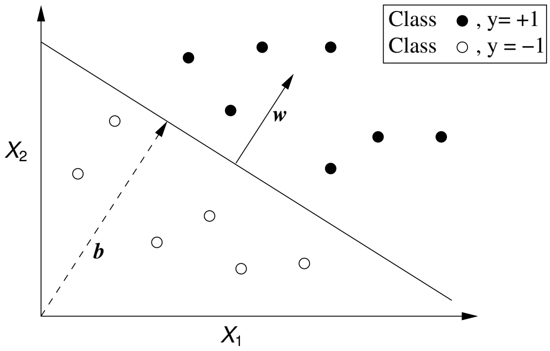
```
tr <- d[1:150,]
ts <- d[151:205,]
arv <- rpartXse(normLoss ~ .,tr)
preds <- predict(arv,ts)
mae <- mean(abs(preds-ts$normLoss),na.rm=T)
mae

## [1] 57.37964

mape <- mean(abs(preds-ts$normLoss)/ts$normLoss,na.rm=T)
mape

## [1] 0.607296
```

Support Vector Machines

A Bit of History...

- SVM's were introduced in 1992 at the COLT-92 conference
- They gave origin to a new class of algorithms named *kernel machines*
- Since then there has been a growing interest on these methods
- More information may be obtained at `www.kernel-machines.org`
- A good reference on SVMs:
  N. Cristianini and J. Shawe-Taylor: An introduction to Support Vector Machines. Cambridge University Press, 2000.
- SVMs have been applied with success in a wide range of areas like: bio-informatics, text mining, hand-written character recognition, etc.

Two Linearly Separable Classes

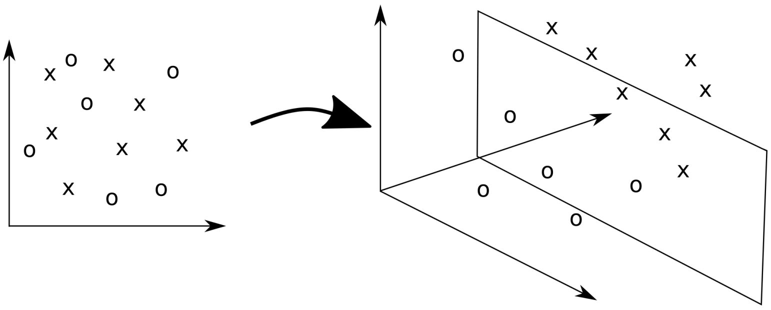


- Obtain a linear separation of the cases (binary classification problems)
- Very simple and effective for linearly separable problems
- Most real-world problems are not linearly separable!

The Basic Idea of SVMs

- Map the original data into a new space of variables with very high dimension.
- Use a linear approximation on this new input space.

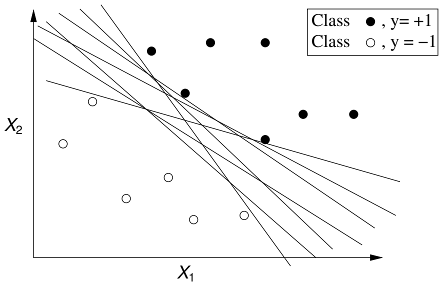
The Idea in a Figure

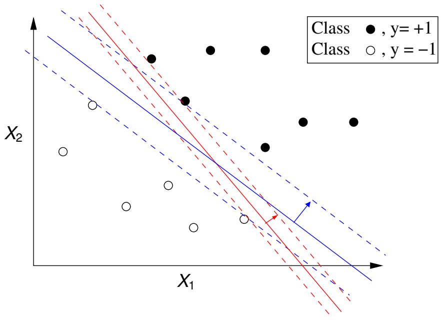


Huh?

Map the original data into a new (higher dimension) coordinates system where the classes are linearly separable

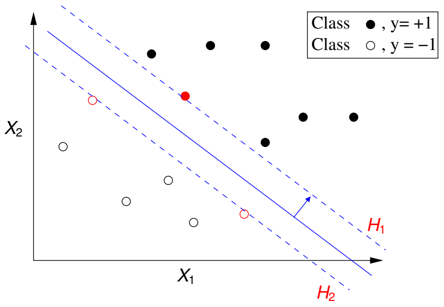
Maximum Margin Hyperplane





- There is an infinite number of hyperplanes separating the two classes!
- Which one should we choose?!
- We want the one that ensures a better classification accuracy on unseen data
- SVMs approach this problem by searching for the maximum margin hyperplane

The Support Vectors



- All cases that fall on the hyperplanes H_1 and H_2 are called the support vectors.
- Removing all other cases would not change the solution!

The Optimal Hyperplane

- SVMs use quadratic optimization algorithms to find the optimal hyperplane that maximizes the margin that separates the cases from the 2 classes
- Namely, these methods are used to find a solution to the following equation,

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

Subject to :

$$\alpha_i \geq 0$$

$$\sum_i \alpha_i y_i = 0$$

- In the found solution, the α_i's > 0 correspond to the support vectors that represent the optimal solution

Recap

- Most real world problems are not linearly separable
- SVMs solve this by “moving” into a extended input space where classes are already linearly separable
- This means the maximum margin hyperplane needs to be found on this new very high dimension space

The Kernel trick

- The solution to the optimization equation involves dot products that are computationally heavy on high-dimensional spaces
- It was demonstrated that the result of these complex calculations is equivalent to the result of applying certain functions (the kernel functions) in the space of the original variables.

The Kernel Trick

Instead of calculating the dot products in a high dimensional space, take advantage of the proof that $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ and simply replace the complex dot products by these simpler and efficient calculations

Summary of the SVMs Method

- As problems are usually non-linear on the original feature space, move into a high-dimension space where linear separability is possible
- Find the optimal separating hyperplane on this new space using quadratic optimization algorithms
- Avoid the heavy computational costs of the dot products using the kernel trick

How to handle more than 2 classes?

It's going to be multiple

- Solve several binary classification tasks
- Essentially find the support vectors that separate each class from all others

The Algorithm

- Given a *m* classes task
- Obtain *m* SVM classifiers, one for each class
- Given a test case assign it to the class whose separating hyperplane is more distant from the test case

Obtaining an SVM in R

The package **e1071**

```
library(e1071)
data(Glass,package='mlbench')
tr <- Glass[1:200,]
ts <- Glass[201:214,]
s <- svm(Type ~ .,tr)
predict(s,ts)
```

```
## 201 202 203 204 205 206 207 208 209 210 211 212 213 214
##   7   2   7   7   7   7   7   2   7   7   7   7   7   7
## Levels: 1 2 3 5 6 7
```

Obtaining an SVM in R (2)

The package **e1071**

```
ps <- predict(s,ts)
table(ps,ts$Type)

##
## ps   1  2  3  5  6  7
##   1  0  0  0  0  0  0
##   2  0  0  0  0  0  2
##   3  0  0  0  0  0  0
##   5  0  0  0  0  0  0
##   6  0  0  0  0  0  0
##   7  0  0  0  0  0 12

mc <- table(ps,ts$Type)
error <- 100*(1-sum(diag(mc))/sum(mc))
error

## [1] 14.28571
```

ε-SV Regression

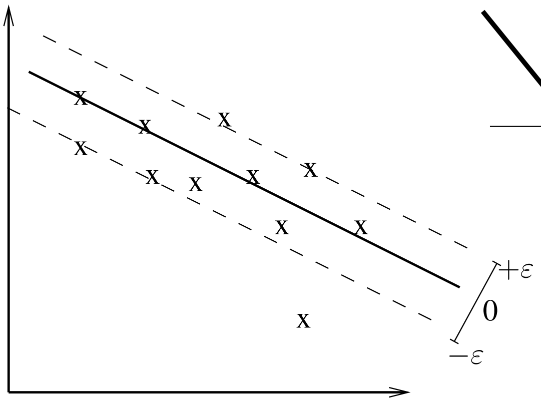
- Vapnik (1995) proposed the notion of ε support vector regression
- The goal in ε-SV Regression is to find a function $f(x)$ that has at most ε deviation from the given training cases
- In other words we do not care about errors smaller than ε

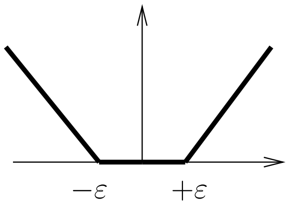
V. Vapnik (1995). The Nature of Statistical Learning Theory. Springer.

ε-SV Regression (cont.)

- ε-SV Regression uses the following error metric,

$$|\xi|_\varepsilon = \begin{cases} 0 & \text{if } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$





ε-SV Regression (cont.)

- The theoretical development of this idea leads to the following optimization problem,

$$\text{Minimize}: \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$\text{Subject to}: \begin{cases} y_i - \mathbf{w}\cdot\mathbf{x} - b & \leq \varepsilon + \xi_i \\ \mathbf{w}\cdot\mathbf{x} + b - y_i & \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases}$$

where C corresponds to the cost to pay for each violation of the error limit ε

ε-SV Regression (cont.)

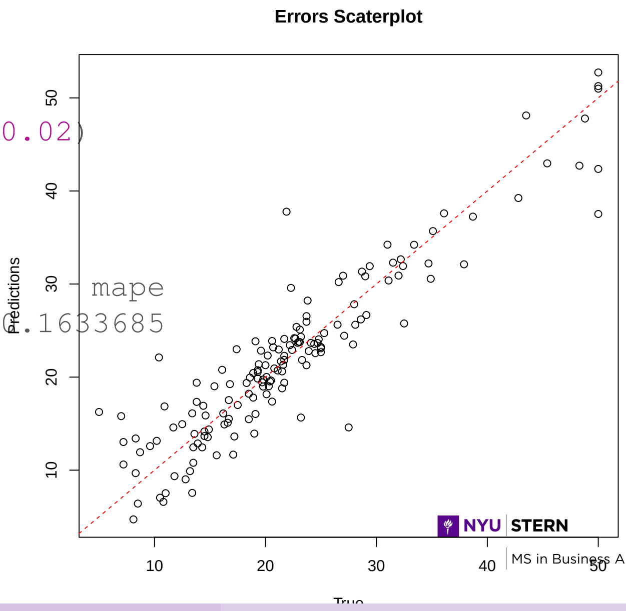
- As within classification we use the kernel trick to map a non-linear problem into a high dimensional space where we solve the same quadratic optimization problem as in the linear case
- In summary, by the use of the $|\xi|_\varepsilon$ loss function we reach a very similar optimization problem to find the support vectors of any non-linear regression problem.

SVMs for regression in R

```r
library(e1071)
library(DMwR)
data(Boston,package='MASS')
set.seed(1234)
sp <- sample(1:nrow(Boston),354)
tr <- Boston[sp,]
ts <- Boston[-sp,]
s <- svm(medv ~ .,tr,cost=10,epsilon=0.02)
preds <- predict(s,ts)
regr.eval(ts$medv,preds)
```

```
##         mae         mse        rmse        mape
##   2.6677663 13.8211063   3.7176749   0.1633685
```

```r
plot(ts$medv,preds,main='Errors Scaterplot',
     ylab='Predictions',xlab='True')
abline(0,1,col='red',lty=2)
```



Model Ensembles and Random Forests

Model Ensembles

What?

- Ensembles are collections of models that are used together to address a certain prediction problem

Why? (Diettrich, 2002)

- For complex problems it is hard to find a model that “explains” all observed data.
- Averaging over a set of models typically leads to significantly better results.

Dietterich, T. G. (2002). Ensemble Learning. In The Handbook of Brain Theory and Neural Networks, Second edition, (M.A. Arbib, Ed.), Cambridge, MA: The MIT Press, 2002. 405-408.

The Bias-Variance Decomposition of Prediction Error

- The prediction error of a model can be split in two main components: the bias and the variance components

- The bias component is the part of the error that is due to the poor ability of the model to fit the seen data
- The variance component has to do with the sensibility of the model to the given training data

The Bias-Variance Decomposition of Prediction Error

- Decreasing the bias by adjusting more to the training sample will most probably lead to a higher variance - the over-fitting phenomenon
- Decreasing the variance by being less sensitive to the given training data will most probably have as consequence a higher bias
- In summary: there is a well-known bias-variance trade-off in learning a prediction model

Ensembles are able to reduce both components of the error

Ranking?

Their approach consist on applying the same algorithm to different samples of the data and use the resulting models in a voting schema to obtain predictions for new cases

Random Forests (Breiman, 2001)

- Random Forests put the ideas of sampling the cases and sampling the predictors, together in a single method
  - Random Forests combine the ideas of bagging together with the idea of random selection of predictors
- Random Forests consist of sets of tree-based models where each tree is obtained from a bootstrap sample of the original data and uses some form of random selection of variables during tree growth

Breiman, L. (2001): "Random Forests". Machine Learning 45 (1): 5—32.

Random Forests - the algorithm

- For each of the *k* models
  - Draw a random sample with replacement to obtain the training set
  - Grow a classification or regression tree
    - On each node of the tree choose the best split from a randomly selected subset *m* of the predictors
- The trees are fully grown, i.e. no pruning is carried out

Random Forests in R

The package `randomForest`

```
library(randomForest)
data(Boston,package="MASS")
samp <- sample(1:nrow(Boston),354)
tr <- Boston[samp,]
ts <- Boston[-samp,]
m <- randomForest(medv ~ ., tr)
ps <- predict(m,ts)
mean(abs(ts$medv-ps))

## [1] 2.190258
```

A classification example

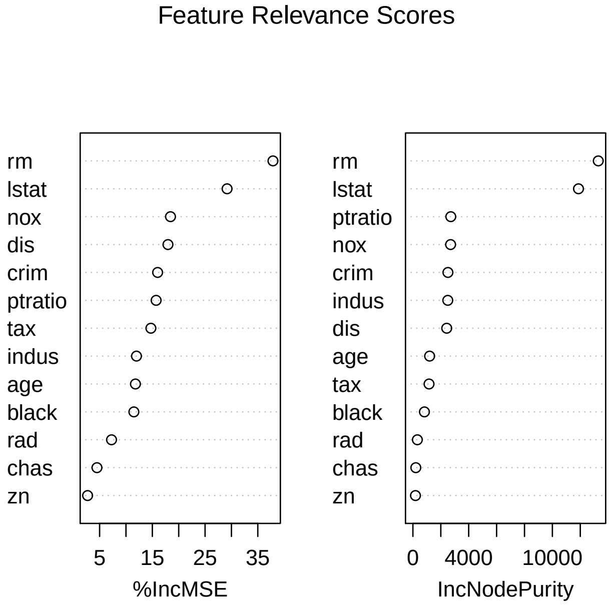
```
data(Glass,package='mlbench')
set.seed(1234)
sp <- sample(1:nrow(Glass),150)
tr <- Glass[sp,]
ts <- Glass[-sp,]
m <- randomForest(Type ~ ., tr,ntree=3000)
ps <- predict(m,ts)
table(ps,ts$Type)

##
## ps   1  2  3  5  6  7
##    1 13  5  3  0  0  1
##    2  2 18  0  3  0  2
##    3  0  0  1  0  0  0
##    5  0  0  0  4  0  0
##    6  0  1  0  0  3  0
##    7  0  0  0  0  0  8

mc <- table(ps,ts$Type)
err <- 100*(1-sum(diag(mc))/sum(mc))
err

## [1] 26.5625
```

Other Uses of Random Forests

Variable Importance

```
data(Boston,package='MASS')
library(randomForest)
m <- randomForest(medv ~ ., Boston,
                  importance=T)
importance(m)
```

```
##            %IncMSE IncNodePurity
## crim     16.001604     2511.3914
## zn        2.719681      184.4274
## indus    11.992644     2501.0269
## chas      4.496731      208.3667
## nox      18.440180     2702.4705
## rm       37.873226    13288.7533
## age      11.793865     1198.7370
## dis      17.957678     2423.8487
## rad       7.259293      320.4829
## tax      14.721102     1157.0856
## ptratio 15.715445     2716.8744
## black    11.498495      826.2531
## lstat    29.172401    11871.6578
```



```
varImpPlot(m,main="Feature Relevance Scores")
```

Evaluation Methodologies and Comparison of Models

Performance Estimation

The setting

- Predictive task: unknown function $Y = f(\mathbf{x})$ that maps the values of a set of predictors into a target variable value (can be a classification or a regression problem)
- A (training) data set $\{< \mathbf{x}_i, y_i >\}_{i=1}^{N}$, with known values of this mapping
- Performance evaluation criterion(a) - metric(s) of predictive performance (e.g. error rate or mean squared error)
- How to obtain a **reliable estimates** of the predictive performance of any solutions we consider to solve the task using the available data set?

NYU | STERN
MS in Business Analytics

Reliability of Estimates

Resubstitution estimates

- Given that we have a data set one possible way to obtain an estimate of the performance of a model is to evaluate it on this data set
- This leads to what is known as a **resubstitution estimate** of the prediction error
- These estimates are unreliable and should not be used as they tend to be over-optimistic!

NYU | STERN
MS in Business Analytics

Reliability of Estimates

Resubstitution estimates (2)

- Why are they unreliable?
  - Models are obtained with the goal of optimizing the selected prediction error statistic on the given data set
  - In this context it is expected that they get good scores!
  - The given data set is just a sample of the unknown distribution of the problem being tackled
  - What we would like is to have the performance of the model on this distribution
  - As this is usually impossible the best we can do is to evaluate the model on **new samples** of this distribution

Goal of Performance Estimation

Main Goal of Performance Estimation

Obtain a **reliable estimate** of the expected prediction error of a model on the unknown data distribution

- In order to be reliable it should be based on evaluation on unseen cases - *a test set*

Goal of Performance Estimation (2)

- Ideally we want to repeat the testing several times
- This way we can collect a series of scores and provide as our estimate the average of these scores, together with the standard error of this estimate
- In summary:
  - calculate the sample mean prediction error on the repetitions as an estimate of the true population mean prediction error
  - complement this sample mean with the standard error of this estimate

Goal of Performance Estimation (3)

- The golden rule of Performance Estimation:

The data used for evaluating (or comparing) any models cannot be seen during model development.

Goal of Performance Estimation (4)

- An experimental methodology should:
  - Allow obtaining several prediction error scores of a model, $E_1, E_2, \cdots, E_k$
  - Such that we can calculate a sample mean prediction error

$$\overline{E} = \frac{1}{k}\sum_{i=1}^{k} E_i$$

  - And also the respective standard error of this estimate

$$SE(\overline{E}) = \frac{s_E}{\sqrt{k}}$$

where s_E is the sample standard deviation of E measured as $\sqrt{\frac{1}{k-1}\sum_{i=1}^{k}(E_i - \overline{E})^2}$

The Holdout Method and Random Subsampling

- The holdout method consists on **randomly dividing the available data sample in two sub-sets** - one used for training the model; and the other for testing/evaluating it
  - A frequently used proportion is 70% for training and 30% for testing

The Holdout Method (2)

- If we have a small data sample there is the danger of either having a too small test set (unreliable estimates as a consequence), or removing too much data from the training set (worse model than what could be obtained with the available data)
- We only get one prediction error score - no average score nor standard error
- If we have a very large data sample this is actually the preferred evaluation method

Random Subsampling

- The Random Subsampling method is a variation of holdout method and it simply consists of repeating the holdout process several times by randomly selecting the train and test partitions
- Has the same problems as the holdout with the exception that we already get several scores and thus can calculate means and standard errors
- If the available data sample is too large the repetitions may be too demanding in computation terms

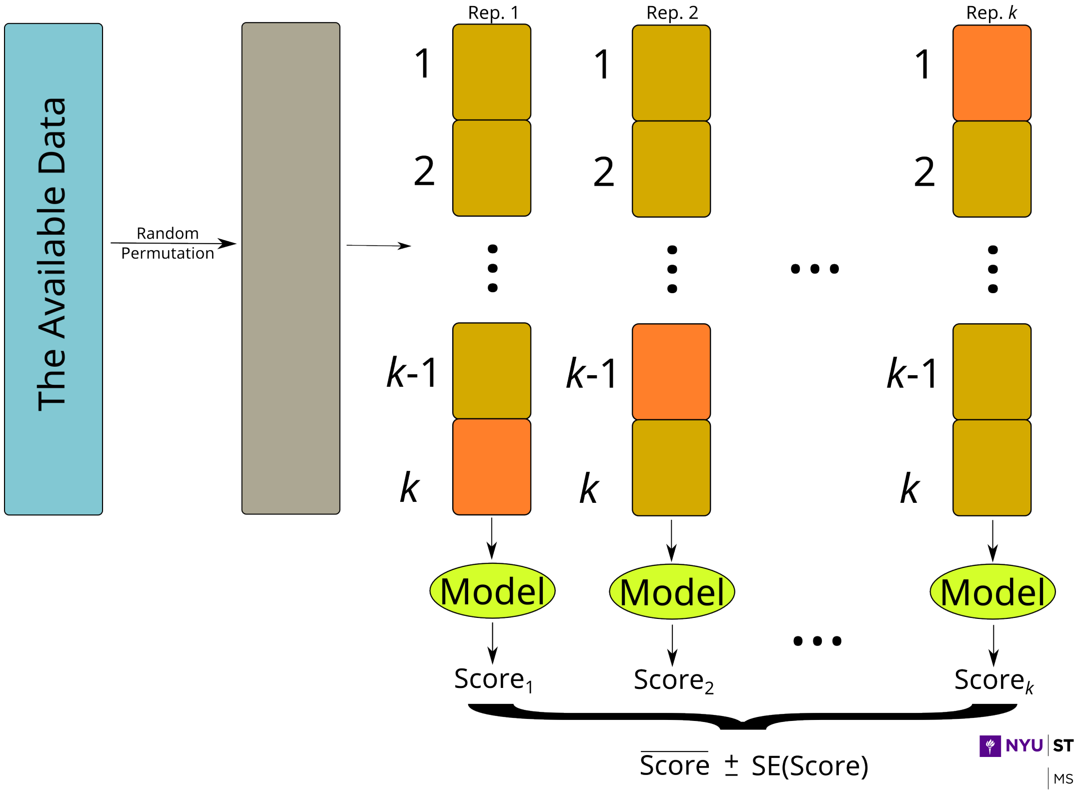
The Holdout method in R

```
library(DMwR)
set.seed(1234)
data(Boston,package='MASS')
## random selection of the holdout
trPerc <- 0.7
sp <- sample(1:nrow(Boston),as.integer(trPerc*nrow(Boston)))
## division in two samples
tr <- Boston[sp,]
ts <- Boston[-sp,]
## obtaining the model and respective predictions on the test set
m <- rpartXse(medv ~.,tr)
p <- predict(m,ts)
## evaluation
regr.eval(ts$medv,p,train.y=tr$medv)

##         mae        mse       rmse       mape       nmse       nmae
##   3.2320067 22.1312980  4.7043913  0.1731599  0.2369631  0.4529675
```

The k-fold Cross Validation Method

- The idea of k-fold Cross Validation (CV) is similar to random subsampling
- It essentially consists of *k* repetitions of training on part of the data and then test on the remaining
- The diference lies on the way the partitions are obtained

The k-fold Cross Validation Method (cont.)



Leave One Out Cross Validation Method (LOOCV)

- Similar idea to k-fold Cross Validation (CV) but in this case on each iteration a single case is left out of the training set
- This means it is essentially equivalent to *n*-fold CV, where *n* is the size of the available data set

The Bootstrap Method

- Train a model on a random sample of size *n* with replacement from the original data set (of size *n*)
  - Sampling with replacement means that after a case is randomly drawn from the data set, it is "put back on the sampling bag"
  - This means that several cases will appear more than once on the training data
  - On average only 63.2% of all cases will be on the training set
- Test the model on the cases that were not used on the training set
- Repeat this process many times (typically around 200)
- The average of the scores on these repetitions is known as the ϵ_0 bootstrap estimate
- The .632 bootstrap estimate is obtained by $.368 \times \epsilon_r + .632 \times \epsilon_0$, where ϵ_r is the resubstitution estimate

NYU | STERN
MS in Business Analytics

Bootstrap in R

```
data(Boston,package='MASS')
nreps <- 200
scores <- vector("numeric",length=nreps)
n <- nrow(Boston)
set.seed(1234)
for(i in 1:nreps) {
   # random sample with replacement
   sp <- sample(n,n,replace=TRUE)
   # data splitting
   tr <- Boston[sp,]
   ts <- Boston[-sp,]
   # model learning and prediction
   m <- lm(medv ~.,tr)
   p <- predict(m,ts)
   # evaluation
   scores[i] <- mean((ts$medv-p)^2)
}
# calculating means and standard errors
summary(scores)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   16.37   21.70   24.20   24.56   26.47   48.82
```

NYU | STERN
MS in Business Analytics

The Infra-Structure of package **performanceEstimation**

- The package **performanceEstimation** provides a set of functions that can be used to carry out comparative experiments of different models on different predictive tasks
- This infra-structure can be applied to any model/task/evaluation metric
- Installation:
  - Official release (from CRAN repositories):

```
install.packages("performanceEstimation")
```

  - Development release (from Github):

```
library(devtools)  # You need to install this package before!
install_github("ltorgo/performanceEstimation",ref="develop")
```

The Infra-Structure of package **performanceEstimation**

- The main function of the package is `performanceEstimation()`
  - It has 3 arguments:
    1. The predictive tasks to use in the comparison
    2. The models to be compared
    3. The estimation task to be carried out
- The function implements a wide range of experimental methodologies including all we have discussed

A Simple Example

- Suppose we want to estimate the mean squared error of regression trees in a certain regression task using cross validation

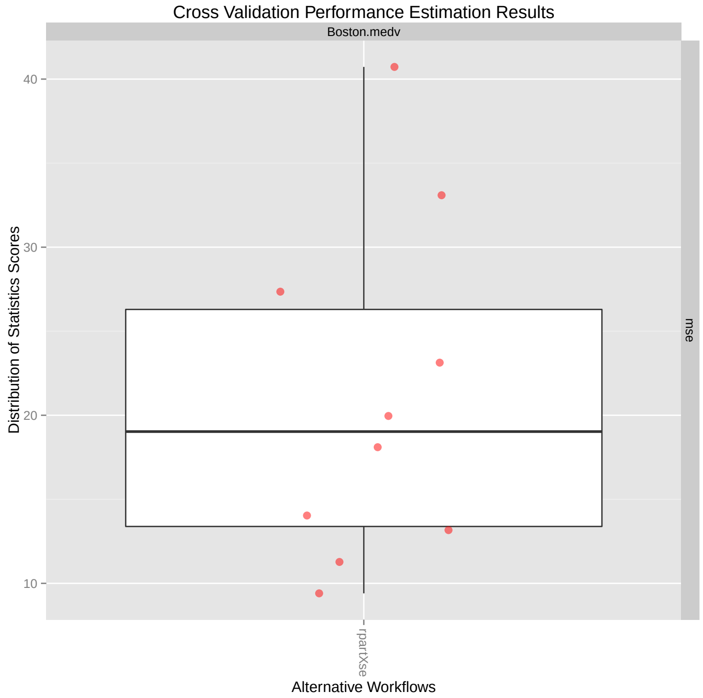
```r
library(performanceEstimation)
library(DMwR)
data(Boston,package='MASS')
res <- performanceEstimation(
    PredTask(medv ~ .,Boston),
    Workflow("standardWF",learner="rpartXse"),
    EstimationTask(metrics="mse",method=CV(nReps=1,nFolds=10)))
```

A Simple Example (2)

```r
summary(res)

##
## == Summary of a  Cross Validation Performance Estimation Experiment ==
##
## Task for estimating  mse  using
##  1 x 10 - Fold Cross Validation
##   Run with seed =  1234
##
## * Predictive Tasks ::  Boston.medv
## * Workflows  ::  rpartXse
##
## -> Task:  Boston.medv
##   *Workflow: rpartXse
##               mse
## avg     21.02393
## std     10.15683
## med     19.02955
## iqr     12.91203
## min      9.40574
## max     40.72403
## invalid  0.00000
```

A Simple Example (3)

```
plot(res)
```



Predictive Tasks

- Objects of class **PredTask** describing a predictive task
  - Classification
  - Regression
  - Time series forecasting
- Created with the constructor with the same name

```
data(iris)
PredTask(Species ~ ., iris)

## Prediction Task Object:
##   Task Name          :: iris.Species
##   Task Type          :: classification
##   Target Feature     :: Species
##   Formula            :: Species ~ .
##   Task Data Source   :: iris

PredTask(Species  ~ ., iris,"IrisDS",copy=TRUE)

## Prediction Task Object:
##   Task Name          :: IrisDS
##   Task Type          :: classification
##   Target Feature     :: Species
##   Formula            :: Species ~ .
##   Task Data Source   :: internal  150x5 data frame.
```

Workflows

- Objects of class **Workflow** describing an approach to a predictive task
  - Standard Workflows
    - Function `standardWF` for classification and regression
    - Function `timeseriesWF` for time series forecasting
  - User-defined Workflows

Standard Workflows for Classification and Regression Tasks

```
library(e1071)
Workflow("standardWF",learner="svm",learner.pars=list(cost=10,gamma=0.1))

## Workflow Object:
##  Workflow ID       ::  svm
##  Workflow Function ::  standardWF
##       Parameter values:
##   learner  ->  svm
##   learner.pars  ->  cost=10 gamma=0.1
```

“standardWF” can be omitted ...

```
Workflow(learner="svm",learner.pars=list(cost=5))

## Workflow Object:
##  Workflow ID       ::  svm
##  Workflow Function ::  standardWF
##       Parameter values:
##   learner  ->  svm
##   learner.pars  ->  cost=5
```

Standard Workflows for Classification and Regression Tasks (cont.)

- Main parameters of the constructor:
  - Learning stage
    - `learner` - which function is used to obtain the model for the training data
    - `learner.pars` - list with the parameter settings to pass to the learner
  - Prediction stage
    - `predictor` - function used to obtain the predictions (defaults to `predict()`)
    - `predictor.pars` - list with the parameter settings to pass to the predictor

Standard Workflows for Classification and Regression Tasks (cont.)

- Main parameters of the constructor (cont.):
  - Data pre-processing
    - `pre` - vector with function names to be applied to the training and test sets before learning
    - `pre.pars` - list with the parameter settings to pass to the functions
  - Predictions post-processing
    - `post` - vector with function names to be applied to the predictions
    - `post.pars` - list with the parameter settings to pass to the functions

Standard Workflows for Classification and Regression Tasks (cont.)

```
data(algae,package="DMwR")
res <- performanceEstimation(
    PredTask(a1 ~ .,algae[,1:12],"A1"),
    Workflow(learner="lm",pre="centralImp",post="onlyPos"),
    EstimationTask("mse",method=CV())       # defaults to 1x10-fold CV
                             )


##
##
## ##### PERFORMANCE ESTIMATION USING  CROSS VALIDATION  #####
##
## ** PREDICTIVE TASK :: A1
##
## ++ MODEL/WORKFLOW :: lm
## Task for estimating  mse  using
##  1 x 10 - Fold Cross Validation
##   Run with seed =  1234
## Repetition  1
## Fold:  1  2  3  4  5  6  7  8  9  10
```

Evaluating Variants of Workflows

Function `workflowVariants()`

```
library(e1071)
data(Boston,package="MASS")
res2 <- performanceEstimation(
    PredTask(medv ~ .,Boston),
    workflowVariants(learner="svm",
                     learner.pars=list(cost=1:5,gamma=c(0.1,0.01))),
    EstimationTask(metrics="mse",method=CV()))
```

Evaluating Variants of Workflows (cont.)

```
summary(res2)
```

```
##
## == Summary of a  Cross Validation Performance Estimation Experiment ==
##
## Task for estimating  mse  using
##  1 x 10 - Fold Cross Validation
##   Run with seed =  1234
##
## * Predictive Tasks ::  Boston.medv
## * Workflows  ::  svm.v1, svm.v2, svm.v3, svm.v4, svm.v5, svm.v6, svm.v7, svm.v8, svm.v9, svm.v10
##
## -> Task:  Boston.medv
##   *Workflow: svm.v1
##                mse
## avg     14.80685
## std     10.15295
## med     12.27015
## iqr     11.87737
## min      5.35198
## max     38.39681
## invalid  0.00000
##
##   *Workflow: svm.v2
##                 mse
## avg     11.995178
## std      7.908371
## med      8.359433
## iqr     11.626306
## min      4.842848
```

```
## invalid  0.000000
##
##   *Workflow: svm.v3
```

Exploring the Results

```
getWorkflow("svm.v1",res2)
```

```
## Workflow Object:
##  Workflow ID       ::  svm.v1
##  Workflow Function ::  standardWF
##       Parameter values:
##   learner.pars  ->  cost=1 gamma=0.1
##   learner  ->  svm
```

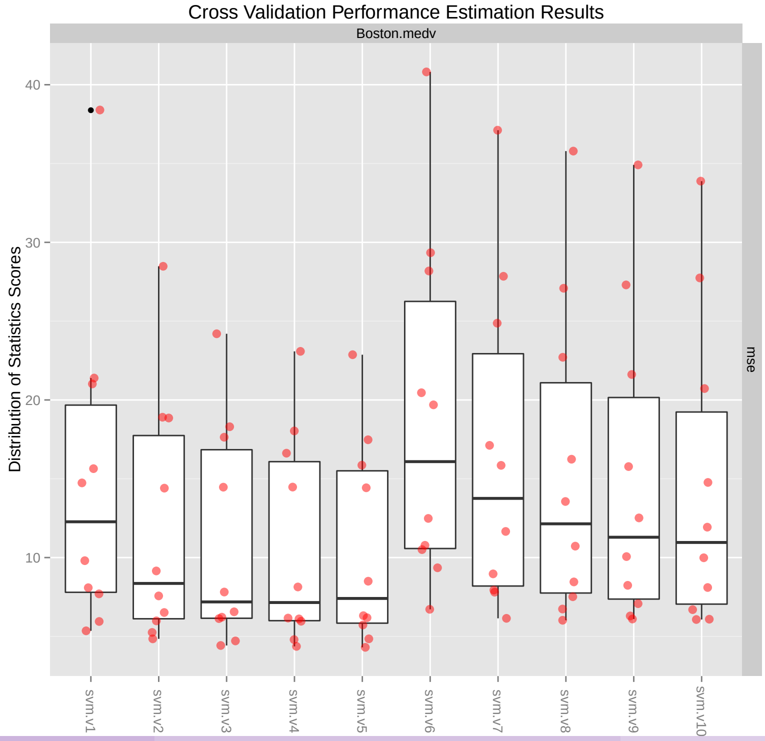
```
topPerformers(res2)
```

```
## $Boston.medv
##     Workflow Estimate
## mse   svm.v5    10.65
```

```
##   *Workflow: svm.v7
##                 mse
```

Visualizing the Results

```
plot(res2)
```



Estimation Tasks

- Objects of class **EstimationTask** describing the estimation task
  - Main parameters of the constructor
    - **metrics** - vector with names of performance metrics
    - **method** - object of class **EstimationMethod** describing the method used to obtain the estimates

```
EstimationTask(metrics=c("F","rec","prec"),method=Bootstrap(nReps=100))

## Task for estimating  F,rec,prec  using
## 100  repetitions of  e0  Bootstrap experiment
##   Run with seed =  1234
```

Performance Metrics

- Many classification and regression metrics are available
  - Check the help page of functions `classificationMetrics` and `regressionMetrics`
- User can provide a function that implements any other metric she/he wishes to use
  - Parameters **evaluator** and **evaluator.pars** of the `EstimationTask` constructor

Comparing Different Algorithms on the Same Task

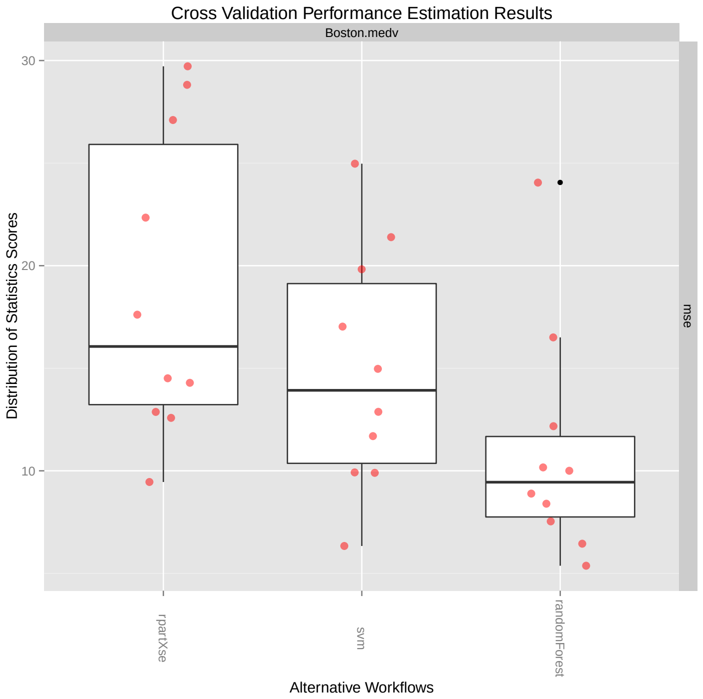
```r
library(randomForest)
library(e1071)
res3 <- performanceEstimation(
    PredTask(medv ~ ., Boston),
    workflowVariants("standardWF",
             learner=c("rpartXse","svm","randomForest")),
    EstimationTask(metrics="mse",method=CV(nReps=2,nFolds=5)))
```

Some auxiliary functions

```
rankWorkflows(res3,3)

## $Boston.medv
## $Boston.medv$mse
##        Workflow Estimate
## 1 randomForest 10.95412
## 2          svm 14.89183
## 3     rpartXse 18.92990
```

The Results

```
plot(res3)
```

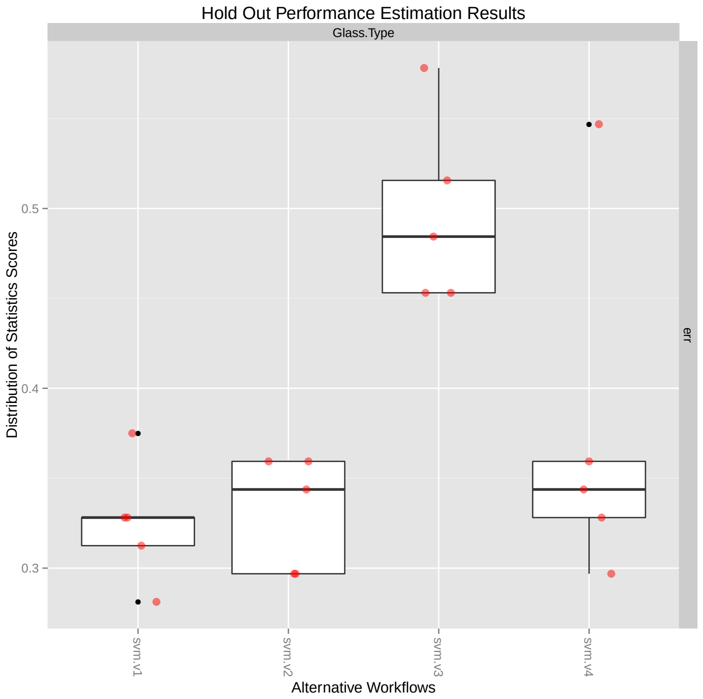


An example using Holdout and a classification task

```
data(Glass,package='mlbench')
res4 <- performanceEstimation(
    PredTask(Type ~ ., Glass),
    workflowVariants(learner="svm",   # You may omit "standardWF" !
                     learner.pars=list(cost=c(1,10),
                                       gamma=c(0.1,0.01))),
    EstimationTask(metrics="err",method=Holdout(nReps=5,hldSz=0.3)))
```

The Results

```
plot(res4)
```

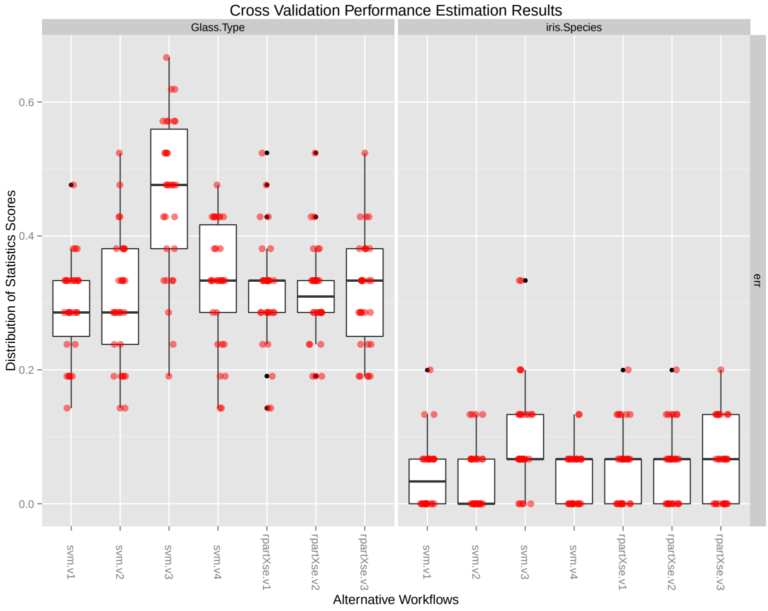


An example involving more than one task

```
data(Glass,package='mlbench')
data(iris)
res5 <- performanceEstimation(
    c(PredTask(Type ~ ., Glass),PredTask(Species ~.,iris)),
    c(workflowVariants(learner="svm",
                       learner.pars=list(cost=c(1,10),
                                         gamma=c(0.1,0.01))),
      workflowVariants(learner="rpartXse",
                       learner.pars=list(se=c(0,0.5,1)),
                       predictor.pars=list(type="class"))),
    EstimationTask(metrics="err",method=CV(nReps=3)))
```

The Results

```
plot(res5)
```



The Results (2)

```
topPerformers(res5)

## $Glass.Type
##     Workflow Estimate
## err   svm.v1    0.294
##
## $iris.Species
##     Workflow Estimate
## err   svm.v2     0.04

topPerformer(res5,"err","Glass.Type")

## Workflow Object:
##  Workflow ID       ::  svm.v1
##  Workflow Function ::  standardWF
##       Parameter values:
##   learner.pars  ->  cost=1 gamma=0.1
##   learner  ->  svm
```