Predictive Analytics

L. Torgo

ltorgo@knoyda.com
KNOYDA, Know Your Data!

Jul, 2019

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.

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

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	Preds. 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 *Symmetric Mean Absolute Percentage Error* (*sMAPE*) is given by,

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

Relative Error Metrics (cont.)

- 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

```
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] 493.991

rmse <- sqrt(mse)
rmse

## [1] 22.22591

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

## [1] 14.35

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

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

## [1] 0.65633

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

## [1] 0.290773

smape <- 1/length(preds) * sum(abs(preds - trueVals) /
                          (abs(preds)+abs(trueVals)))
smape

## [1] 0.5250418

corr <- cor(trueVals,preds)
corr

## [1] 0.6745381
```


Linear Discriminant

The Linear Discriminant

The Idea

Search for linear combinations of the variables that better separate between the objects of the classes

The formalism for two classes - Fisher linear discriminant

- Let $\hat{C}$ the pooled sample covariance matrix

$$\hat{C} = \frac{1}{n_1 + n_2}\left(n_1 \hat{C}_1 + n_2 \hat{C}_2\right)$$

where n_i is the number of training cases per class and $\hat{C}_i$ are the $p \times p$ sample covariance matrices for each class. The sample covariance between two variables is given by

$$Cov(X, Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

The Linear Discriminant (cont.)

The formalism (cont.)

- The following is the score of the separation provided by a p-dimensional vector $\mathbf{w}$,

$$S_{\mathbf{w}} = \frac{\mathbf{w}^T \hat{\mu_1} - \mathbf{w}^T \hat{\mu_2}}{\mathbf{w}^T \hat{C} \mathbf{w}}$$

- Given this score the goal is to find the vector $\mathbf{w}$ that maximizes it. There is a solution for this maximization problem given by,

$$\hat{\mathbf{w}}_{lda} = \hat{C}^{-1}(\hat{\mu_1} - \hat{\mu_2})$$

- Canonical discriminant functions extend the idea for more than two classes

Canonical Discriminant Functions

Example

```
library(MASS)
data(iris)
lda(Species ~ ., iris)

## Call:
## lda(Species ~ ., data = iris)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##     0.3333     0.3333     0.3333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
##
## Coefficients of linear discriminants:
##                    LD1     LD2
## Sepal.Length  0.8294  0.0241
## Sepal.Width   1.5345  2.1645
## Petal.Length -2.2012 -0.9319
## Petal.Width  -2.8105  2.8392
##
## Proportion of trace:
##    LD1    LD2
## 0.9912 0.0088
```


Using LDA for prediction in R

```
sp <- sample(1:150,100)
tr <- iris[sp,]
ts <- iris[-sp,]
l <- lda(Species ~ ., tr)
preds <- predict(l,ts)
(mtrx <- table(preds$class,ts$Species))

##
##                setosa versicolor virginica
##    setosa          16          0         0
##    versicolor       0         18         0
##    virginica        0          1        15

(err <- 1-sum(diag(mtrx))/sum(mtrx))

## [1] 0.02
```

Hands on LDAs - the Vehicle data set

The data set `Vehicle` is available in package **mlbench**. Load it and explore its help page to grab a minimal understanding of the data and then answer the following questions:

1. Obtain a random split of the data into two sub-sets using the proportion 80%-20%.
2. Obtain a linear discriminant using the larger set.
3. Obtain the predictions of the obtained model on the smaller set.
4. Obtain a confusion matrix of the predictions and calculate the respective accuracy.

Multiple Linear Regression

Multiple Linear Regression

- Multiple linear regression is probably the most used statistical method
- It is one of the many possible approaches to the multiple regression problem where given a training data set $\mathbf{D} = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$ we want to obtain an approximation of the unknown regression function $f()$ that maps the predictors values into a target continuous variable value.
- In matrix notation we have $\mathbf{D} = \mathbf{X}|\mathbf{Y}$, where $\mathbf{X}$ is a matrix $n \times p$, and $\mathbf{Y}$ is a matrix $n \times 1$.

Multiple Linear Regression (cont.)

- A regression model $r_D(.)$ can be seen as a function that transforms a vector of values of the predictors, $\mathbf{x}$, into a real number, Y. This model is an approximation of the unknown $f()$ function.
- Regression models assume the following relationship, $y_i = r(\beta, \mathbf{x}_i) + \epsilon_i$, where $r(\beta, \mathbf{x}_i)$ is a regression model with parameters β and ϵ_i are observation errors.
- The goal of a learning method is to obtain the model parameters β that minimize a certain preference criterion.

Multiple Linear Regression (cont.)

- In the case of multiple linear regression the functional form that is assumed is the following:

$$Y = \beta_0 + \beta_1 \cdot X_1 + \cdots + \beta_p \cdot X_p$$

- The goal is to find the vector of parameters β that minimizes the sum of the squared errors $\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 \cdot X_1 + \cdots + \beta_p \cdot X_p))^2$

Multiple Linear Regression

Pros and Cons

- Well-known and over-studied topic with many variants of this simple methodology (e.g. Drapper and Smith, 1981)
- Simple and effective approach when the “linearity” assumption is adequate to the data.
- Form of the model is intuitive - a set of additive effects of each variable towards the prediction
- Computationally very efficient
- Too strong assumptions on the shape of the unknown regression function

Drapper and Smith (1981): Applied Regression Analysis, 2nd edition. Wiley Series in Probability and Mathematical Statistics.

Obtaining Multiple Linear Regression Models in R

```
library(DMwR2)
data(algae)
algae <- algae[-c(62,199),]   # the 2 incomplete samples
clean.algae <- knnImputation(algae) # lm() does not handle NAs!
la1 <- lm(a1 ~ .,clean.algae[,1:12])
la1
```

```
##
## Call:
## lm(formula = a1 ~ ., data = clean.algae[, 1:12])
##
## Coefficients:
##  (Intercept)  seasonspring  seasonsummer  seasonwinter    sizemedium
##    42.942055      3.726978      0.747597      3.692955      3.263728
##    sizesmall      speedlow   speedmedium         mxPH         mnO2
##     9.682140      3.922084      0.246764     -3.589118      1.052636
##            Cl           NO3           NH4          oPO4           PO4
##    -0.040172     -1.511235      0.001634     -0.005435     -0.052241
##          Chla
##    -0.088022
```


Obtaining Multiple Linear Regression Models in R (cont.)

```
summary(la1)
```

```
##
## Call:
## lm(formula = a1 ~ ., data = clean.algae[, 1:12])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.679 -11.893  -2.567   7.410  62.190
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  42.942055  24.010879   1.788  0.07537 .
## seasonspring  3.726978   4.137741   0.901  0.36892
## seasonsummer  0.747597   4.020711   0.186  0.85270
## seasonwinter  3.692955   3.865391   0.955  0.34065
## sizemedium    3.263728   3.802051   0.858  0.39179
## sizesmall     9.682140   4.179971   2.316  0.02166 *
## speedlow      3.922084   4.706315   0.833  0.40573
## speedmedium   0.246764   3.241874   0.076  0.93941
## mxPH         -3.589118   2.703528  -1.328  0.18598
## mnO2          1.052636   0.705018   1.493  0.13715
## Cl           -0.040172   0.033661  -1.193  0.23426
## NO3          -1.511235   0.551339  -2.741  0.00674 **
## NH4           0.001634   0.001003   1.628  0.10516
## oPO4         -0.005435   0.039884  -0.136  0.89177
## PO4          -0.052241   0.030755  -1.699  0.09109 .
## Chla         -0.088022   0.079098  -1.113  0.27265
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The Diagnostic Information of the summary() Call

- Distribution of the residuals (errors) of the model - should have mean zero and should be normally distributed and as small as possible
- Estimate of each coefficient and respective standard error
- Test of the hypothesis that each coefficient is null, i.e. $H0 : \beta_i = 0$
  - Uses a t-test
  - Calculates a t-value as β_i / s_{β_i}
  - Presents a column (Pr(>t)) with the level at which the hypothesis is rejected. A value of 0.0001 would mean that we are 99.99% confident that the coefficient is not null
  - Coefficients for which we can reject the hypothesis are tagged with a symbol

The Diagnostic Information of the summary() Call (cont.)

- We are also given the R^2 coefficients (multiple and adjusted). These coefficients indicate the degree of fit of the model to the data, i.e. the proportion of variance explained by the model. Values near 1 (100%) are better. The adjusted coefficient is more demanding as it takes into account the size of the model
- Finally, there is also a test of the hypothesis that there is no dependence of the target variable on the predictors, i.e. $H0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$. The F-statistic is used with this purpose. R provides the confidence level at which we are sure to reject this hypothesis. A p-level of 0.0001 means that we are 99.99% confident that the hypothesis is not true.

Simplifying the Linear Model

```
final.la1 <- step(la1)
```

```
summary(final.la1)
```

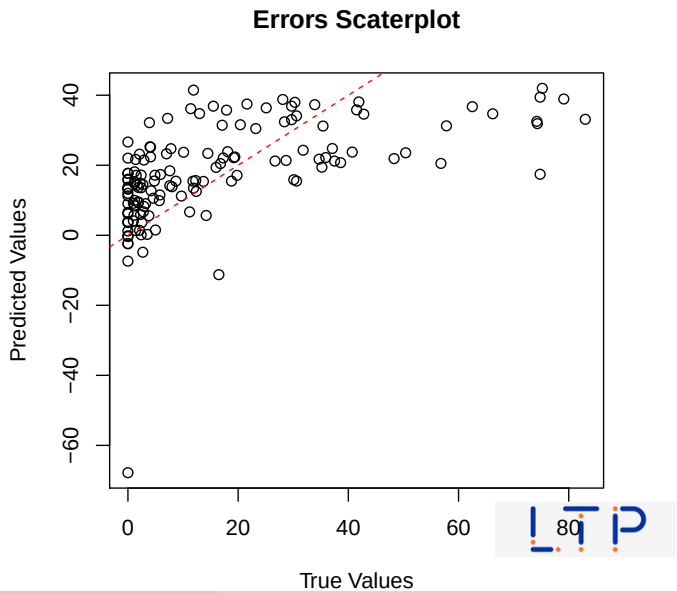
```
##
## Call:
## lm(formula = a1 ~ size + mxPH + Cl + NO3 + PO4, data = clean.algae[,
##     1:12])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.874 -12.732  -3.741   8.424  62.926
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 57.28555   20.96132   2.733  0.00687 **
## sizemedium   2.80050    3.40190   0.823  0.41141
## sizesmall   10.40636    3.82243   2.722  0.00708 **
## mxPH        -3.97076    2.48204  -1.600  0.11130
## Cl          -0.05227    0.03165  -1.651  0.10028
## NO3         -0.89529    0.35148  -2.547  0.01165 *
## PO4         -0.05911    0.01117  -5.291 3.32e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.5 on 191 degrees of freedom
## Multiple R-squared:  0.3527,Adjusted R-squared:  0.3324
## F-statistic: 17.35 on 6 and 191 DF,  p-value: 5.554e-16
```


Using the Models for Prediction

```
clean.test.algae <- knnImputation(test.algae,
    k = 10, distData = clean.algae[, 1:11])
preds <- predict(final.la1,clean.test.algae)
mean((preds-algae.sols$a1)^2)
```

```
## [1] 296.0934
```

```
plot(algae.sols$a1,preds,main='Errors Scaterplot',
     ylab='Predicted Values',xlab='True Values')
abline(0,1,col='red',lty=2)
```



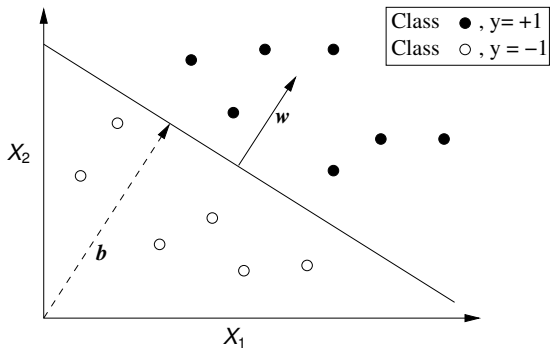
But there are no negative algae frequencies!...

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

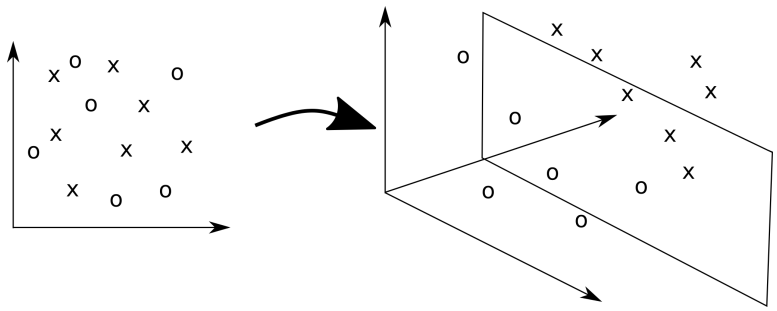


- Obtain a linear separation of the cases (binary classification problems)
- Very simple and effective for linearly separable problems
- Most real-world problems are not linearly separable!

The Basic Idea of SVMs

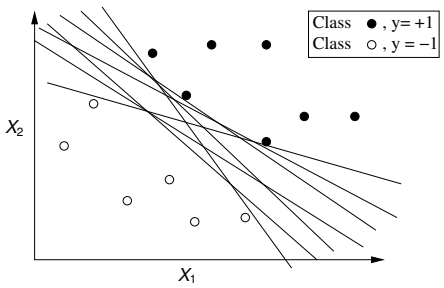
- Map the original data into a new space of variables with very high dimension.
- Use a linear approximation on this new input space.

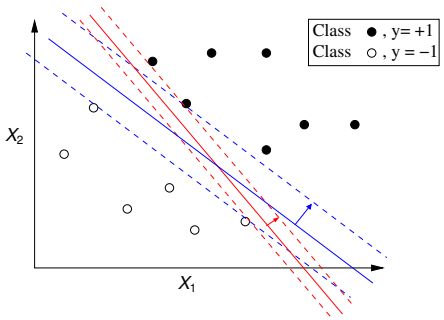
The Idea in a Figure



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

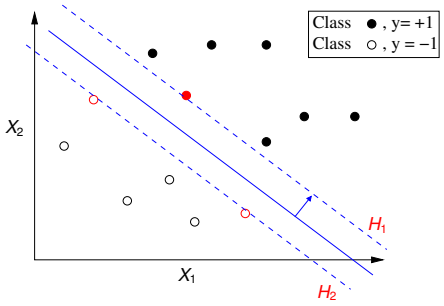
Maximum Margin Hyperplane





- 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,

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

$$\text{Subject to :}$$

$$\alpha_i \geq 0$$

$$\sum_i \alpha_i y_i = 0$$

- 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?

- 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

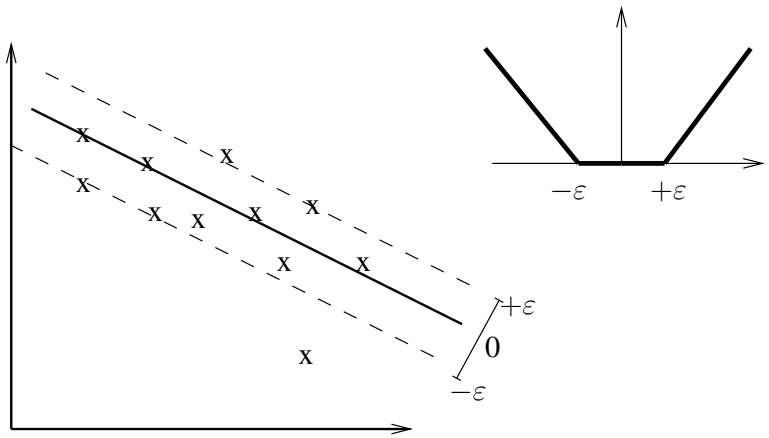
- Vapnik (1995) proposed the notion of ε support vector regression
- The goal in ε-SV Regression is to find a function $f(x)$ that has at most ε deviation from the given training cases
- In other words we do not care about errors smaller than ε

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.)

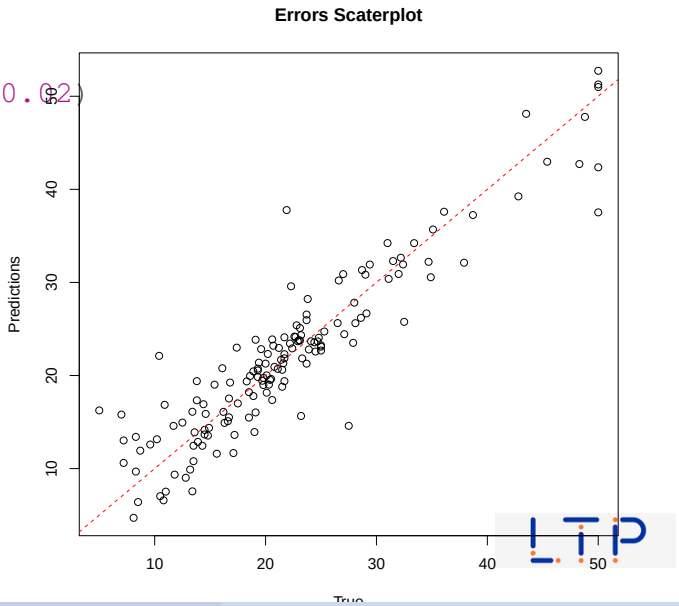
- As within classification we use the kernel trick to map a non-linear problem into a high dimensional space where we solve the same quadratic optimization problem as in the linear case
- In summary, by the use of the $|\xi|_\varepsilon$ loss function we reach a very similar optimization problem to find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

SVMs for regression in R

```
library(e1071)
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)
mean((ts$medv-preds)^2)

## [1] 13.82111
```

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



Hands on SVMs

The file `Wine.Rdata` contains 2 data frames with data about the quality of "green" wines: i) `redWine` and ii) `whiteWine`. Each of these data sets has information on a series of wine tasting sessions to "green" wines (both red and white). For each wine sample several physico-chemical properties of the wine sample together with a quality score assigned by a committee of wine experts (variable `quality`).

1. Obtain and SVM for forecasting the quality of the red variant of "green" wines
2. Split the data set in two parts: one with 70% of the samples and the other with the remaining 30%. Obtain an SVM with the first part and apply it to the second. What was the resulting mean absolute error?
3. Using the `round()` function, round the predictions obtained in the previous question to the nearest integer. Calculate the error rate of the resulting integers when compared to the true values

Hands on Linear Regression - the Boston data set

The data set `Boston` is available in package **MASS**. Load it and explore its help page to grab a minimal understanding of the data and then answer the following questions:

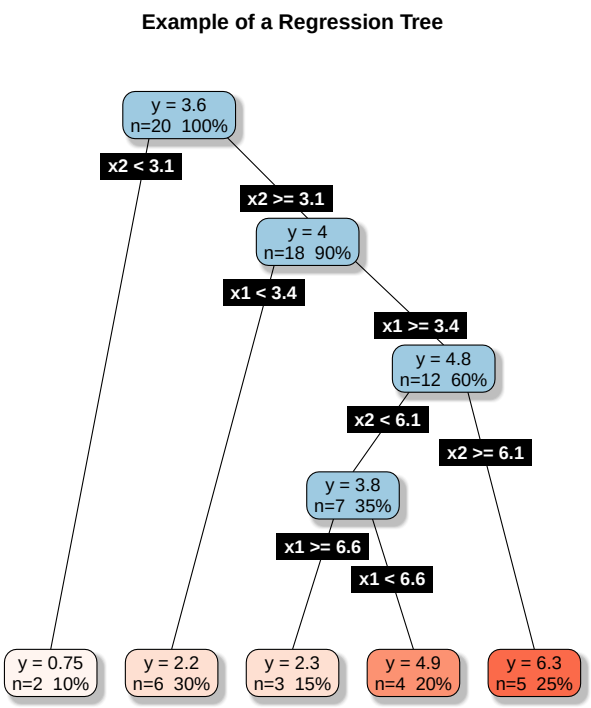
1. Obtain a random split of the data into two sub-sets using the proportion 70%-30%.
2. Obtain a multiple linear regression model using the larger set.
3. Check the diagnostic information provided for the model.
4. Obtain the predictions of the obtained model on the smaller set.
5. Obtain the mean squared error of these predictions and also an error scatter plot.

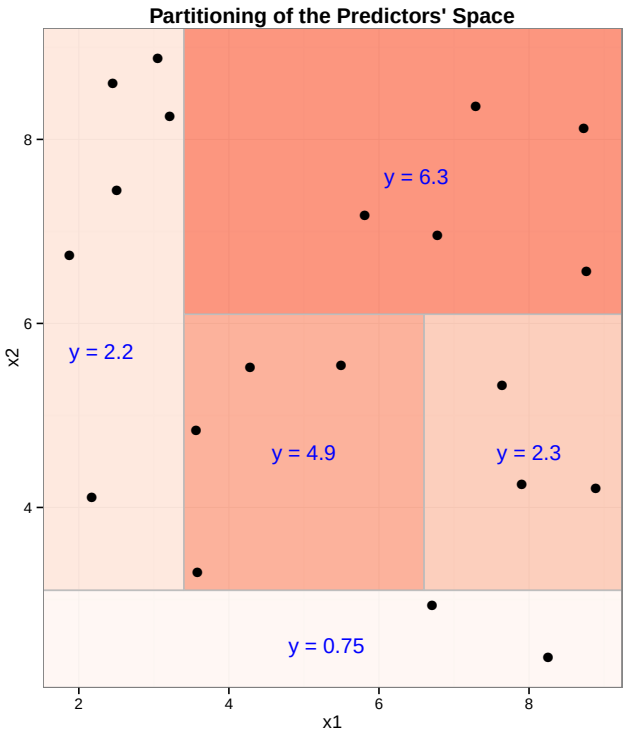
Tree-based Models

Tree-based Models

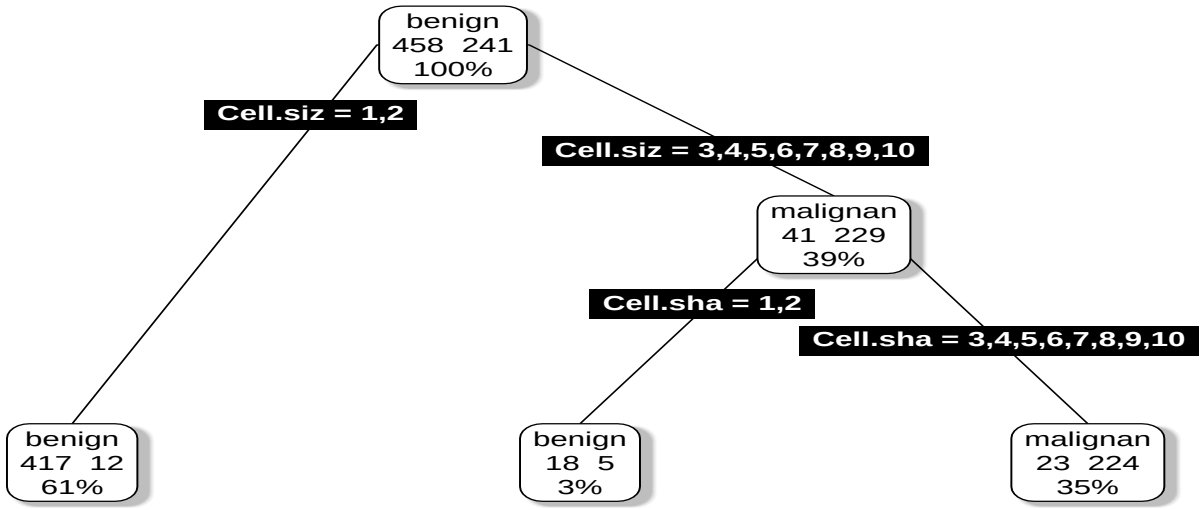
- Tree-based models (both classification and regression trees) are models that provide as result a model based on logical tests on the input variables
- These models can be seen as a partitioning of the input space defined by the input variables
- This partitioning is defined based on carefully chosen 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

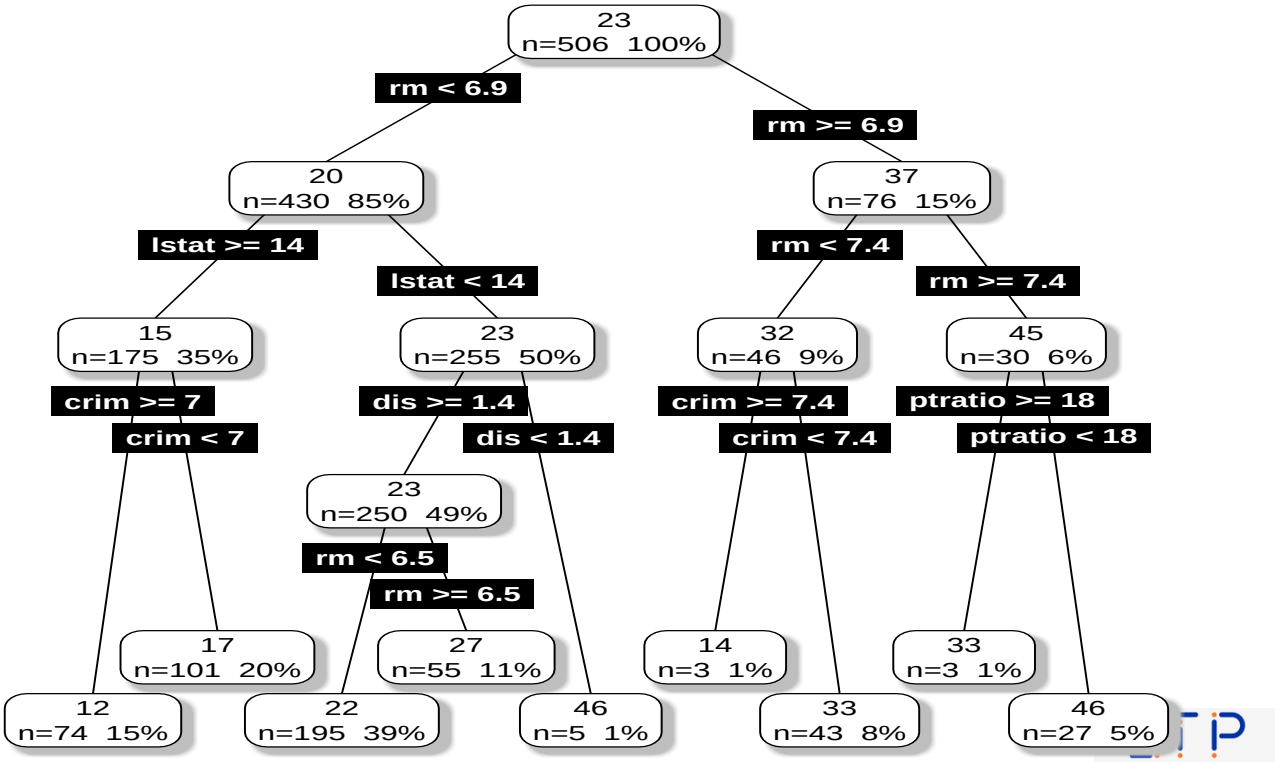




An Example of a Classification Tree



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

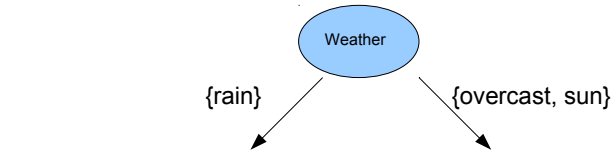
The Recursive Partitioning Algorithm

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
sun	15	normal	63	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

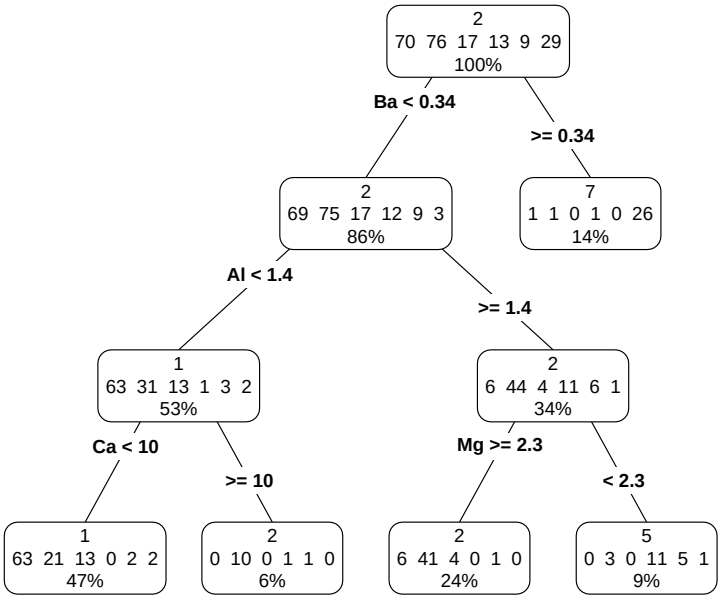
Classification and Regression Trees in R

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 `DMwR2` 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(DMwR2)
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
```

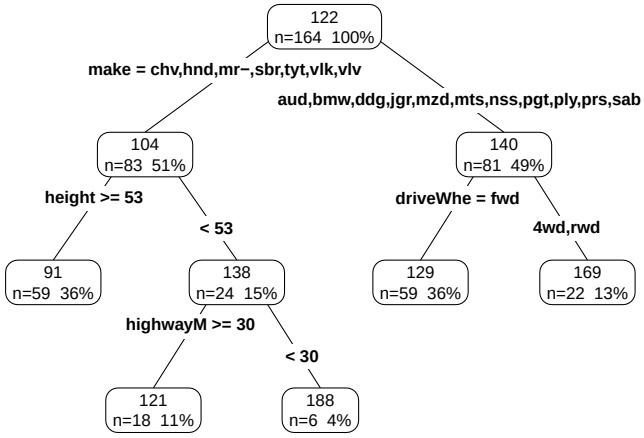
```
mc <- table(ps,ts$Type)
err <- 100*(1-sum(diag(mc))/sum(mc))
err
```

```
## [1] 7.142857
```

Illustration using a regression task

Forecasting Normalized Losses

```
library(DMwR2)
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
```

Hands on Tree-based Models - the Wines data

File `Wine.Rdata` contains two data frames with data on green wine quality: (i) `redWine` and (ii) `whiteWine`. Each of these data sets contains a series of tests with green wines (red and white). For each of these tests the values of several physicochemical variables together with a quality score assigned by wine experts (column `quality`).

1. Build a regression tree for the white wines data set
2. Obtain a graph of the obtained regression tree
3. Apply the tree to the data used to obtain the model and calculate the mean squared error of the predictions
4. Split the data set in two parts: 70% of the tests and the remaining 30%. Using the larger part to obtain a regression tree and apply it to the other part. Calculate again the mean squared error. Compare with the previous scores and comment.

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

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")
```

Hands on Linear Regression and Random Forests

the Algae data set

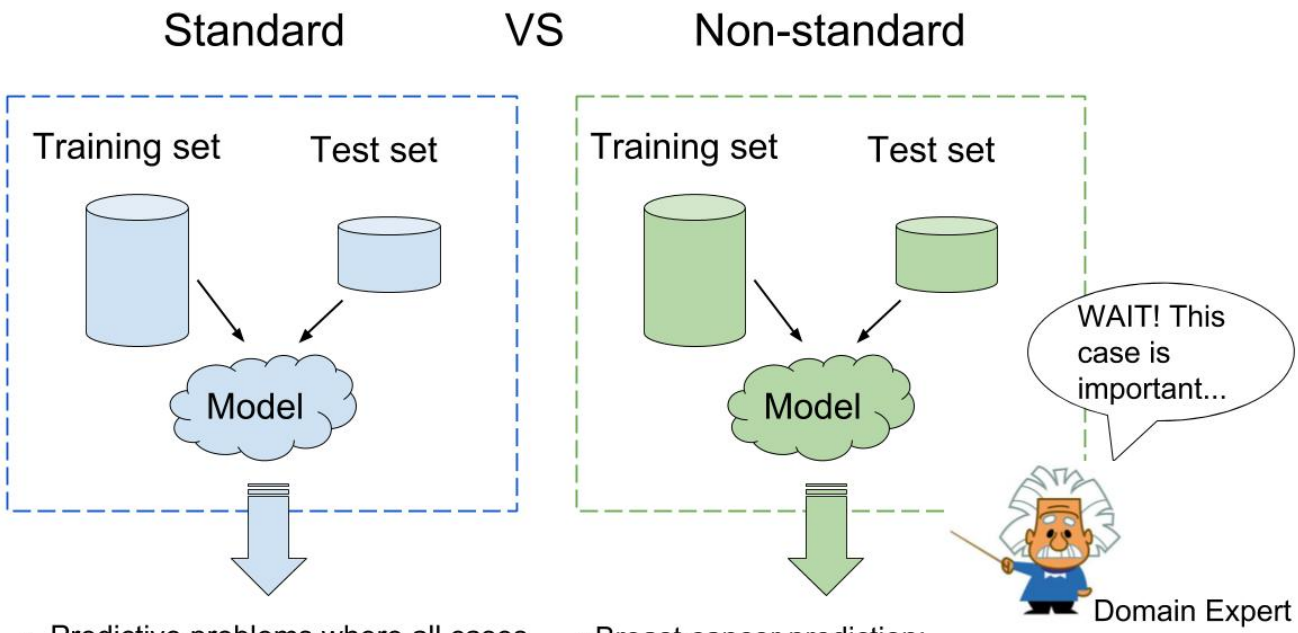
Load in the data set `algae` from package **DMwR2** and answer the following questions:

1. How would you obtain a random forest to forecast the value of alga *a4*
2. Repeat the previous exercise but now using a linear regression model. Try to simplify the model using the `step()` function.
3. Obtain the predictions of the two previous models for the data used to obtain them. Draw a scatterplot comparing these predictions
4. The data frame named `test.algae` contains a test set with some extra 140 water samples for which we want predictions. Use the previous two models to obtain predictions for `a4` on these new samples. Check what happened to the test cases with NA's. Fill-in the NA's on the test set and repeat the experiment.

Handling Imbalanced Distributions

work by Paula Branco, Luis Torgo and Rita Ribeiro

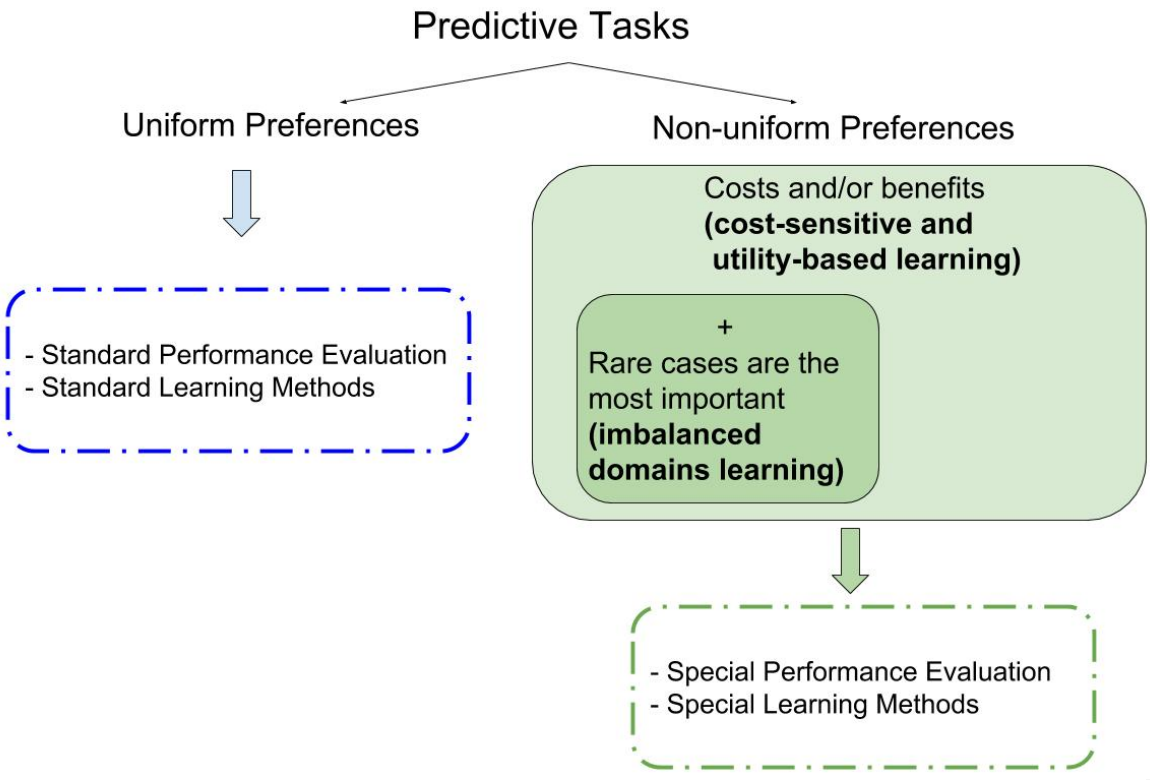
The Problem of Imbalanced Domains



- Predictive problems where all cases have the same importance

- Breast cancer prediction;
- Forecasting floods, extreme weather events;
- Fraud prediction; etc

The Problem of Imbalanced Domains



Utility-based Learning

- Predictive tasks
- Goal: obtain a good approximation h of an unknown function $Y = f(X_1, X_2, \cdots, X_p)$
- Use a training set $D = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$

Utility-based Learning Problem

1. the user assigns a non-uniform importance to the predictive performance of the model h across the target variable domain

Imbalanced Domains Problem

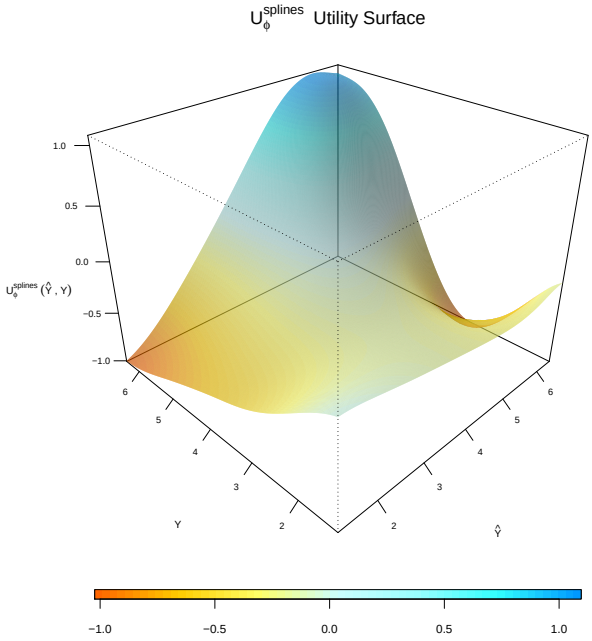
1. the user assigns a non-uniform importance to the predictive performance of the model h across the target variable domain;
2. $|\text{important cases}| << |\text{normal cases}|$

What is a non-uniform importance?

- If we have the **full information** available:
  - cost/utility matrix (classification)
  - utility surfaces (regression)
- If we only have **partial or informal information**:
  - a relevance function
  - estimate information from data distribution

Utility Matrices and Surfaces

		True	
		Fraud	No-Fraud
Predicted	Fraud	1.5	−10
	No-Fraud	−100	0



Relevance Function and Information Estimated from Data

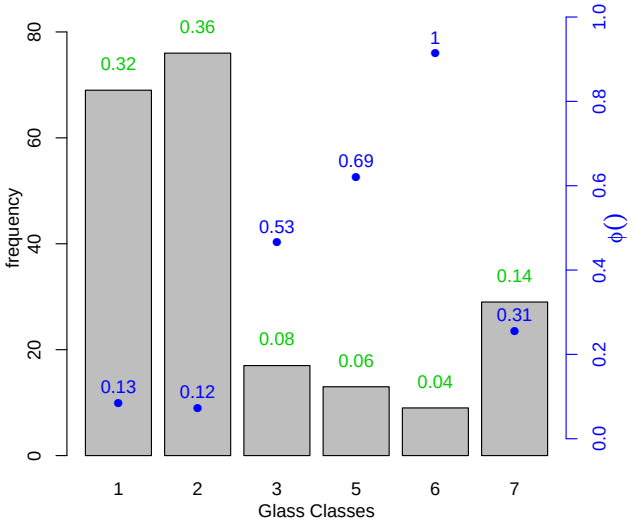


Figure: Classification Example

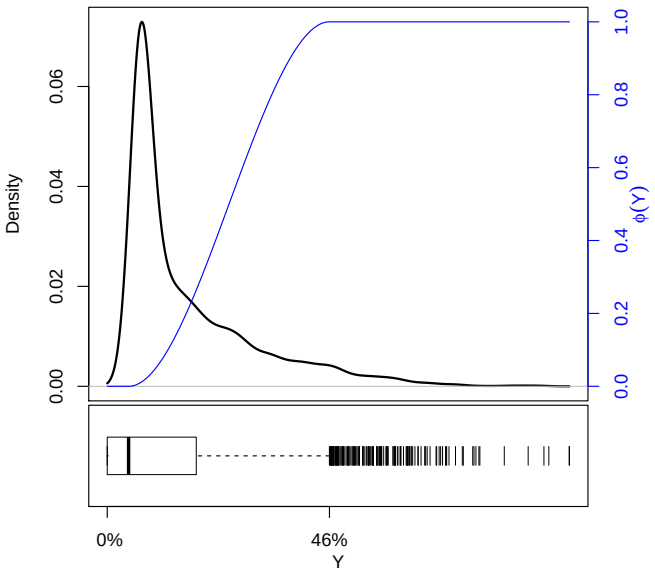


Figure: Regression Example

Main Challenges

Estimation of Utility Information: How can we obtain a utility surface/matrix when only partial or informal information is available?

Modelling Approaches: How can we build models that take into consideration the domain specific preferences?

Performance Assessment Measures: How can we evaluate the performance of the models considering the user preferences?

Utility-based Learning in R - **UBL Package**

- Package that implements most existing methods to handle imbalanced distributions and utilit-based learning approaches
- Available on **CRAN**: `https://CRAN.R-project.org/package=UBL`
- Installation as any R package:

```
install.packages("UBL")
```

- Development versions available on **github**: `https://github.com/paobranco/UBL`

Uses of the package UBL

1. For deriving utility surfaces;
2. For optimizing a utility surface/matrix;
3. For **learning in imbalanced domains**.

Addressing Imbalancved Domains

Two main approaches:

1. Changing the learning algorithms to cope with the imbalance
2. **Changing the original data distribution to facilitate the task of the algorithms** - known as resampling approaches

Addressing Imbalanced Domains Problems with UBL

Classification

Removing or Adding Cases

- Random Under-/Over-sampling
- Tomek Links
- Condensed Nearest Neighbors (CNN)
- One-Sided Selection (OSS)
- Edited Nearest Neighbors (ENN)
- Neighborhood CLeaning rule (NCL)
- Importance Sampling
- Under-sampling with Neighborhood Bias

Generating New Synthetic Cases

- SMOTE
- SMOTE with Neighborhood Bias
- Introduction of Gaussian Noise

Addressing Imbalanced Domains Problems with UBL

Regression

Removing or Adding Cases

- Random Under-sampling
- Random Over-sampling
- Importance Sampling
- Under-sampling with neighborhood Bias

Generating New Synthetic Cases

- SMOTER
- SMOTER with Neighborhood Bias
- Introduction of Gaussian Noise
- SMOGN

A simple (classification) example

```
library(UBL) # Loading our infra-structure
library(e1071) # package containing the svm we will use
data(ImbC) # The synthetic data set we are going to use
summary(ImbC) # Summary of the ImbC data

##        X1                X2          Class
##  Min.   :-13.5843   cat :300   normal:859
##  1st Qu.: -2.6930   dog :400   rare1 : 10
##  Median : -0.1592   fish:300   rare2 :131
##  Mean   : -0.1064
##  3rd Qu.:  2.4633
##  Max.   : 12.7836

table(ImbC$Class)

##
## normal  rare1  rare2
##    859     10    131
```

Obtaining a model with the original data

```
set.seed(123)
samp <- sample(1:nrow(ImbC), nrow(ImbC)*0.7)
train <- ImbC[samp,]
test <- ImbC[-samp,]
model <- svm(Class~., train)
preds <- predict(model,test)
table(preds, test$Class) # confusion matrix

##
## preds    normal rare1 rare2
##   normal    258     5    37
##   rare1       0     0     0
##   rare2       0     0     0
```

The model completely ignored the rare (and more important) classes!

Changing the distribution of the original data

```
# not using the default distance (Eucledian) because of the nominal feature
newtrain <- SmoteClassif(Class~., train, C.perc="balance", dist="HEOM")

## Warning:  SmoteClassif ::  Nr of examples is less or equal to k.
##  Using k = 4 in the nearest neighbours computation in this bump.

# generate a new model with the changed data
newmodel <- svm(Class~., newtrain)
preds <- predict(newmodel,test)
table(preds, test$Class)

##
## preds    normal rare1 rare2
##   normal    109     0     4
##   rare1      10     5     0
##   rare2     139     0    33
```

Trying another method

```
newtrain2 <- RandOverClassif(Class~., train, C.perc="balance")
#generate a new model with the modified data set
newmodel2 <- svm(Class~., newtrain2)
preds <- predict(newmodel2, test)
table(preds, test$Class)

##
## preds    normal rare1 rare2
##   normal    133     1     4
##   rare1       7     4     0
##   rare2     118     0    33
```

A simple (regression) example

```
data(ImbR)
summary(ImbR)

##        X1                X2               Tgt
##  Min.   : 0.3654   Min.   : 0.201   Min.   :10.00
##  1st Qu.: 8.2821   1st Qu.: 8.246   1st Qu.:10.06
##  Median : 9.9811   Median :10.129   Median :10.22
##  Mean   : 9.9418   Mean   :10.078   Mean   :10.98
##  3rd Qu.:11.7202   3rd Qu.:11.903   3rd Qu.:10.72
##  Max.   :19.0565   Max.   :19.474   Max.   :23.17

set.seed(123)
samp <- sample(1:nrow(ImbR), as.integer(0.7*nrow(ImbR)))
trainD <- ImbR[samp,]
testD <- ImbR[-samp,]
```

Obtaining a random forest with the original data

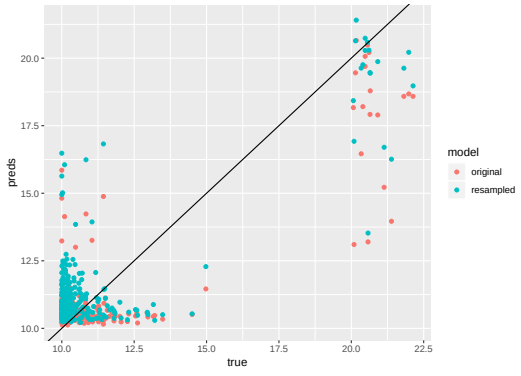
```
library(randomForest)
model <- randomForest(Tgt~., trainD)
preds <- predict(model, testD)
```

Changing the distribution of the original data

```
# using the Introduction of Gaussian Noise with the default parameters
newTrain <- GaussNoiseRegress(Tgt~., trainD)
newModel <-randomForest(Tgt~., newTrain)
newPreds <- predict(newModel, testD)
```


Comparing the results

```
res <- data.frame(true=rep(testD$Tgt,2),preds=c(preds,newPreds),
                  model=c(rep("original",nrow(testD)),rep("resampled",nrow(testD))))
library(ggplot2)
ggplot(res, aes(x=true,y=preds,color=model)) + geom_point() + geom_abline(slope=1,intercept=0)
```



Further Information on Imbalanced Domains

- P. Branco, L. Torgo, R. Ribeiro (2016): *A Survey of Predictive Modeling on Imbalanced Domains*, **ACM Comput. Surv.**, (49), 2-31, 2016
- Check the vignette of package UBL for further examples