

Comparing tree-based classifiers for diagnosing metastasis-risk in breast cancer patients

Peter Kaiser[†], Benedikt Brors[‡], Patrick Warnat[‡]

[†]Ruprecht Karls University of Heidelberg, Institute of Pharmacy and Molecular Biotechnology, 69120 Heidelberg, Germany and

[‡]Intelligent Bioinformatics Systems, German Cancer Research Institute, 69120 Heidelberg, Germany

Abstract

Five tree based classifiers were built using gene expression data from 78 breast cancer patients (data taken from van't Veer et al. [2002]¹) in order to compare their classification accuracy. Of these 78 patients, 34 had developed metastases in the five years after diagnosing the tumor and 44 had not. The goal was to find the most accurate classifier for predicting cases which were not used in classifier construction. Estimated prediction errors were calculated by averaging the results of 10 repetitions of a 10-fold cross-validation run. Results show that ensemble methods, whose prediction results from a combination of several classifiers, help to improve classifier accuracy. However, no classifier could be established that predicted classes with sufficiently low error.

1. Introduction

Microarray experiments² are a promising source of information in contemporary cancer research³. They give rise to diagnostic applications that could lead to a targeted drug therapy and thereby to a reduction of unspecific and unnecessary medication of tumor patients. Given the vast amount of data produced by microarray experiments, machine learning algorithms are needed in order to build predictors based on gene expression data. The dataset used in this study contained logarithmic expression ratios of $G=4917$ genes derived from $N=78$ patients, as described in van't Veer *et al.*¹

1.1. Basic definitions and terms

Definition of classification

Classification is a prediction or learning problem in which the response variable i to be predicted assumes one of K unordered values: $i \in \{1, \dots, K\}$. The aim is, given a feature vector, \vec{x} (see below), to predict the class i of a case. In our problem, $K=2$ and class labels are 1="patient is metastase negative after five years" and 2="patient is metastase positive after five years".

Case

We use the word case here to describe the pair $\vec{c} = (\vec{x}, i)$; one case represents one patient.

Feature vector \vec{x} and feature Space χ

Each patient's gene expression profile is a vector $\vec{x} = (x_1, \dots, x_G)$ of feature variables. \vec{x} can be regarded as the coordinate vector of a case. In Figure 1 b, the classes to be separated are represented by \blacklozenge and \blacksquare , respectively. For simplicity, the feature vectors in Figure 1 are reduced to a 2-Dimensional feature space $\chi \in \mathbb{R}^2$.

Classifier

A classifier \hat{C} partitions the feature space χ into K disjoint and exhaustive subsets A_1, \dots, A_K , such that for a case with feature vector $\vec{x} = (x_1, \dots, x_G) \in A_k$, the predicted class is $i \in \{1, \dots, K\}$:

$$\hat{C} : \chi \rightarrow \{1, \dots, K\} \quad (1)$$

Training set

Classifiers are built from past experience, i.e. from cases which are known to belong to a certain class. A training set is usually obtained by sampling n cases without replacement out of the original dataset of N cases. This is schematically shown in Figure 1 a. These n cases comprise the training (learning) set, $L = \{(\bar{x}^1, i^1), \dots, (\bar{x}^n, i^n)\}$, which is used to build the classifier.

Test set

In order to assess the accuracy of the resulting classifier, the smaller set of $N-n$ cases (generally 10% of the original Dataset) is classified using \hat{C} and the predictions are compared to the true class of each case.

Binary decision tree

This study compares classifiers based on binary decision trees. Building a binary decision tree is equal to recursively partitioning the training data into two distinct sets:

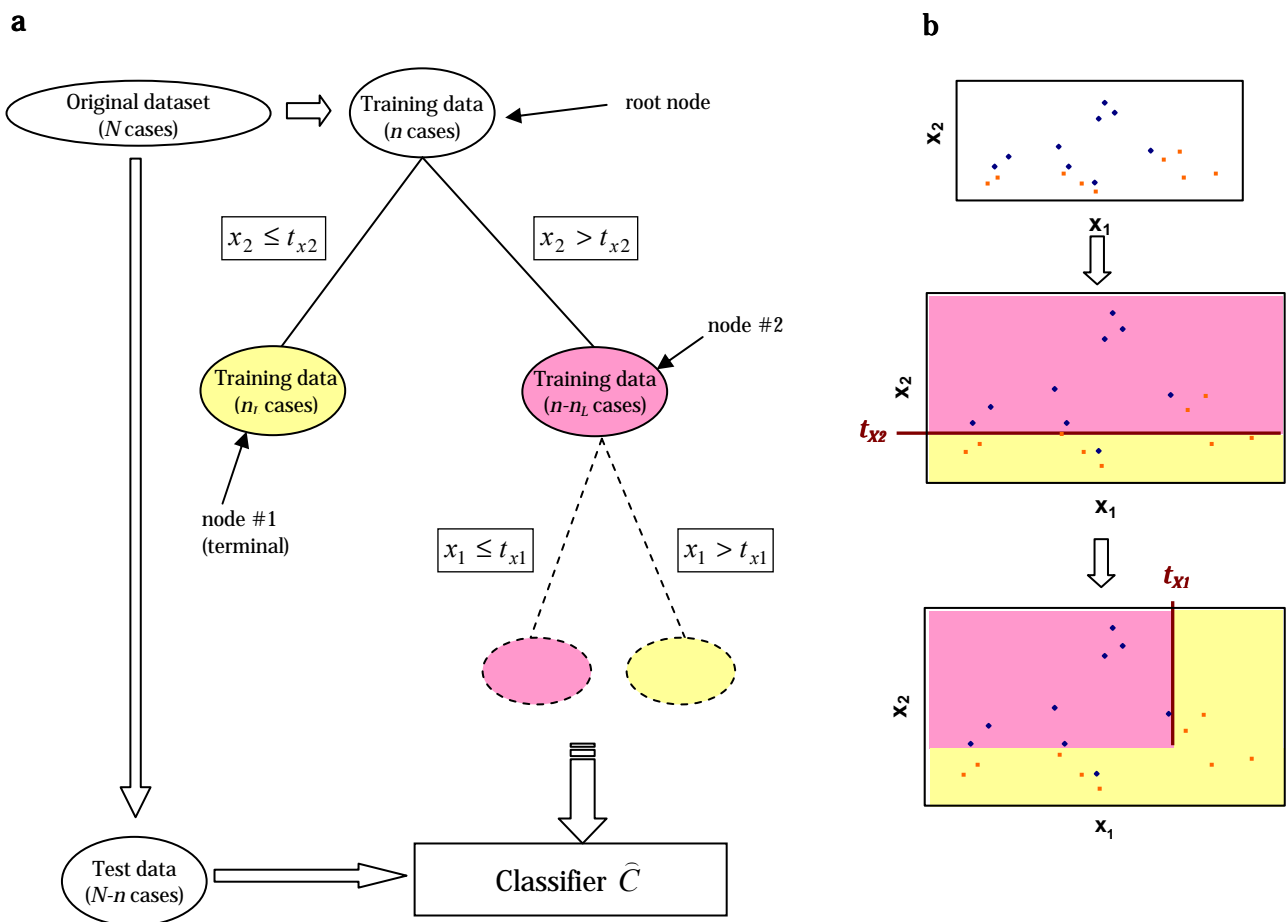


Figure 1: Building a binary decision tree (schematic). **(a)** The original dataset is divided into a large training and a smaller test set. Beginning in the root node, the algorithm tries to partition the training data based only on variable x_1 or x_2 , respectively. A comparison of the optimal split achieved using variable x_1 with threshold value t_{x1} or variable x_2 with threshold value t_{x2} yields x_2 as the optimal split variable with split value t_{x2} . Following this splitting rule (in our example: “Is $x_2 > t_{x2}$?”), the training set is split and distributed into *node #1* and *node #2*. Partitioning proceeds for each node as long as splitting a node decreases its impurity or a minimal number of cases in a terminal node is reached. **(b)** Schematic view of this recursive partitioning problem in the feature space. The decision tree algorithm splits the feature space horizontally and vertically.

In this specific example, gene expression data of approximately 25,000 genes derived from 78 breast cancer patients were analyzed. Thirty-four of these patients had developed metastases in the five years after diagnosis of a primary tumor, 44 remained metastase-free. After initial feature reduction, as described in van't Veer *et al.*⁴, 4917 genes were used to build five different tree-based classifiers with

the aim to predict if a patient develops metastases in the first five years after tumor diagnosis. In the results section, the accuracy of each classifier is documented in order to help to understand the advantages and weaknesses of the methods involved.

2. State of the art

Given the available set of N cases, we want to estimate a prediction error which is close to the error we would observe when classifying a number of previously unseen cases.

The most common way to acquire this estimated prediction error is to calculate the error rate on the test data, i.e. to divide the number of erroneous predictions by the sum over all $N-n$ predicted cases:

$$\frac{\sum_{h=1}^{N-n} I(\hat{C}(\bar{x}^h) \neq \tau(\bar{c}^h))}{N-n} \quad (2)$$

with $\tau(\bar{c}^h)$ being the true class of case \bar{c}^h

$$\text{and } I = \begin{cases} 1 & \text{if } \hat{C}(\bar{x}^h) \neq \tau(\bar{c}^h) \\ 0 & \text{else} \end{cases}$$

This, however, requires a sufficiently large number of test and training cases. While the required several thousand cases are easily available in data mining tasks like consumer analyses etc., microarray studies usually produce no more than one hundred cases. This shortcoming is one of the main obstacles when applying decision tree algorithms to microarray gene expression data.

With several methods being investigated, current research has yet to find a gold standard for this estimation problem (see Dietterich et al. [1997]⁵). A method that has proven to yield reliable estimated prediction errors is 10-fold cross-validation:

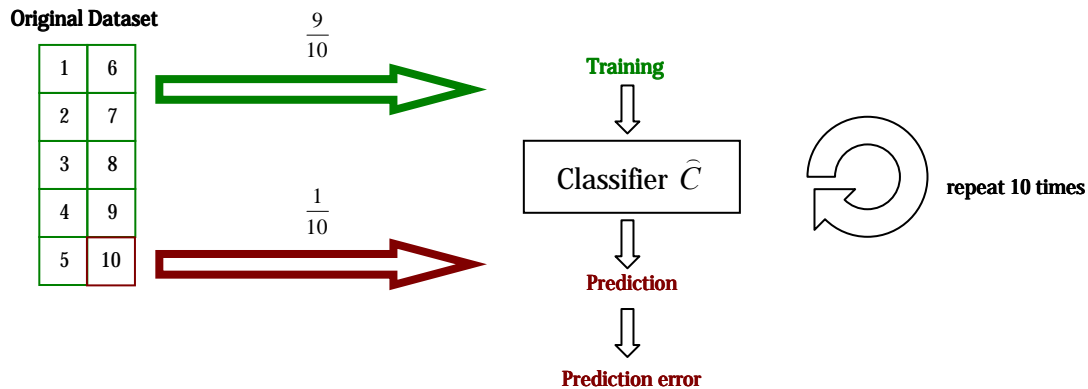


Figure 2: Schematic overview of 10-fold cross-validation. The original dataset is divided into 10 distinct sets. For each partition, the remaining 90% of cases (training data) are used to build a classifier. This classifier is then evaluated by classifying the partition (test data) which was not used when building the tree. A comparison of the predicted classes on the one hand and the true classes on the other yields the prediction error on the test set. This procedure is repeated for all 10 partitions.

- 1) Given a dataset of N cases, randomly divide this dataset into 10 partitions of equal size
- 2) For each of the 10 partitions: build a classifier using the remaining 90% of the total which are not included in the partition
- 3) Compute, for each classifier, the error rate on the test set
- 4) The mean of these error rates is the prediction error rate of a 10-fold cross-validation

As the resulting prediction error rate e_{10cv} might vary between distinct partitioning trials, it is common to reduce variance by repeating this 10-fold cross-validation step 10 times. The average of these errors yields a trustworthy estimated prediction error.

Other methods generally used are leave-one-out cross-validation, where the test data is always one single case, and bootstrapping (see Witten/Frank [1999]⁶).

3. Methods

The analysis was based on a reduced dataset of 78 cases and 4917 genes. The comparison involved two single-tree classifiers:

- (i) rpart as an R implementation of Breiman's CART (Classification and regression Tree) algorithm (see Breiman, Olshen, Friedman, Stone [1984]⁷, Therneau, Atkinson [1997]⁸)
- (ii) C5.0, a commercial advancement of Quinlan's C4.5 algorithm (see Quinlan [1993]⁹)

as well as three ensemble methods:

- (iii) bagging, part of the ipred package for R (implementation of Breiman [1996]¹⁰)
- (iv) randomForest, a R implementation of Breiman's random Forests (see Breiman [2001]¹¹)
- (v) Multiple Decision Trees, developed at the ibios Group at DKFZ, Heidelberg (see Schoch C., Kohlmann A. *et al.* [2003]¹²)

As an estimate of prediction error for each classifier, we averaged the error rates of 10 repetitions of a 10-fold cross-validation. The corresponding scripts for all classifiers built in R are attached as supplementary information D.

3.1. Data preparation: Gene filtering

A reduction of the number of genes from approximately 25000 to 4917 was conducted as described in van't Veer *et al.*¹. The filtering was done in R taking the original selection of van't Veer *et al.*¹: "Only genes with more than two-fold regulation and significance of regulation $p < 0.01$ in more than 3 experiments were kept." Here, $p < 0.01$ simply states that there is a chance of less than 1% that a particular gene shows an expression ratio different from unity only due to measurement errors. The corresponding script used for gene filtering is attached as supplementary information C.

3.2. Single Tree Methods

The following is a short overview of the different methods used. Each explanation describes how to build one classifier from a test set of data. For every classifier these steps were repeated 100 times in total.

For reasons of clarity, a short introduction on how rpart builds tree classifiers, as well as rpart's differences to the C5.0 algorithm, are sourced out to supplementary information A and B.

- (i) rpart

The produced trees contained only one split. This proved to give estimated prediction error rates equal to or better than growing more complex trees.

- (ii) Single Decision Tree (SDT) using C5.0

We used Quinlan's commercial C5.0 algorithm, which only has slight differences to Breiman's CART algorithm. "Stumps" were built, as - like for rpart - tree size proved to have little effect on the estimated prediction error.

3.3. Multiple Tree Methods

Tree-based predictors tend to overfit, that means they inadequately adapt themselves to the training data and thus give poor results when predicting test cases. One solution is to smooth the decision boundary by using a combination of predictors, which were built using different samples of the training data:

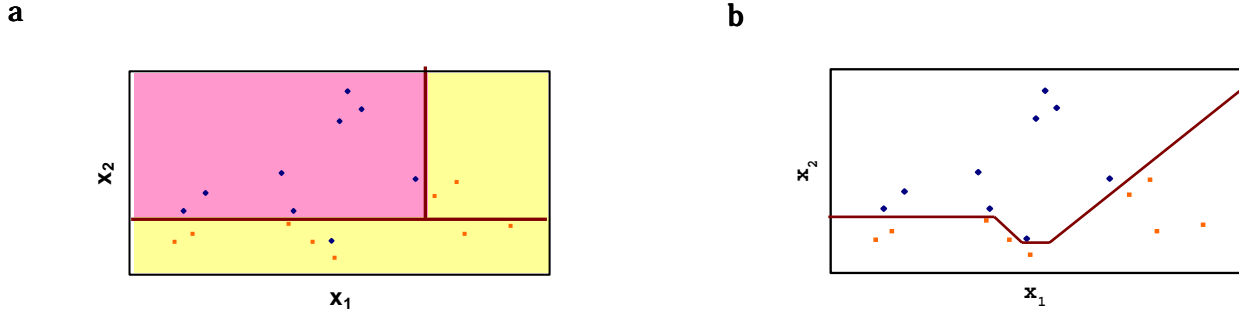


Figure 3: Smoothing the decision boundary of a tree based classifier. (a) While one single tree based classifiers divides the feature space horizontally and vertically (see red line), (b) a combination of trees, such as a randomForest, permits more complex decision boundaries, resulting in a more accurate separation of classes.

As each tree in the ensemble is built using only a sample of the training data, the classifier doesn't "overfit" to the training data. The resulting ensemble predictor thus generally has a lower test set error. In our analysis the ensemble is built using different samples of test cases and – for random Forest and Multiple Decision Trees – different sets of feature variables (genes). The classification resulting from an ensemble predictor is the majority vote over all trees; equality of votes meaning misclassification.

(iii) Bagging

The ipred package¹³ comes with an implementation of Breiman's bagging ("bootstrap aggregating") algorithm (see Breiman [1996]¹⁰). The function `ipredbagg.factor()` was used to build an ensemble of 10 trees, each of them grown to its maximum size. The training data for each of those 10 trees was derived by sampling n cases out of n with replacement. Thus, each tree was built using approximately a fraction of:

$$\begin{aligned}
 \text{fraction of cases used} &= P(\text{case}_c \in \text{sample}) \\
 &= 1 - P(\text{case}_c \text{ not being selected at least once}) \\
 &= 1 - \left(1 - P(\text{case}_c \text{ being selected at a single sampling step})\right)^n \\
 &= 1 - \left(1 - \frac{1}{n}\right)^n \\
 &\approx (1 - e^{-1}) \\
 &\approx 0.632 \quad (3)
 \end{aligned}$$

of the cases contained in the original training data set of n cases.

An optimization of the number of trees used by `ipredbagg.factor()` would have been necessary. However, this was not possible because of memory problems when using the ipred routines. A maximal number of 10 trees could be grown on the computer systems used.

(iv) randomForest

Thorsten Hothorn's latest release of the randomForest package for R¹⁴ was used. Random Forest has proven to be a more stable classifier (Hothorn [2003]¹⁵) than single decision trees. An exhaustive explanation of how random forests are grown as well as a statistical justification can be found in Breiman [2001]¹¹. Building the classifier is summarized as follows:

- 1) grow k trees, each on a bootstrap sample of the training data:
 - a. at each node, randomly select m out of the 4917 features (genes)
 - b. compute the best split among those m features (genes)
 - c. repeat steps a. and b. and grow the tree to its maximum size
- 2) The classifier is acquired by combining the trees; a new case is predicted by a majority vote of all trees.

Random Forests are a simple yet promising expansion of bagging with less computation expenditure but equal or even higher accuracy.

An optimization of the parameter k was realized by an internal 3fold cross-validation on the training data using the function tuneRF(). Further, the optimal number of feature variables (genes) used at each split was computed, using tuneRF() after the optimization of k was done. This resulted in one randomForest classifier with given parameters k and m .

(v) Multiple Decision Trees

Using Quinlan's C5.0 algorithm and the iTable tool (developed at DKFZ and phase-IT by Fieres, J., Wiemer, J., Prolendin, A., Warnat, P.), a multiple decision tree classifier, identical to the classifier used by Schoch, C., Kohlmann, A. *et al.* [2003]¹², was constructed. The classifier was built as follows:

- 1) Parameter optimization: optimal number of trees used.
→ Perform a 3-fold cross-validation by executing the following steps:
 - a. split the training data in to three partitions
 - b. use two of the three partitions as training data
 - c. build one single split decision tree
 - d. remove the feature (gene) used for this split from the training data
 - e. determine the split for the next tree using the remaining 4917-1 feature variables (genes)
 - f. repeat steps c. to e. until 50 trees are grown
 - g. predict the class of the third (test) partition by using the votes of trees 1,2,...,50
 - h. perform steps b. to f. for all three partitions
 - i. compute the mean prediction error over those three runs for each number of trees
 - j. the number of trees corresponding to the lowest mean prediction error is k , the optimum number of trees used
- 2) Build an ensemble of k trees using the complete training data set
- 3) Each tree is built as described in steps 1) c. and d.
- 4) The classifier is acquired by combining the trees; a new case is predicted by a majority vote of all trees, equality of votes meaning misclassification.

4. Results

The estimated prediction error for each classifier was computed.

Comparing Tree Based Classifiers: Results Table

Method	estimated prediction error	std. error
MDT	32.26%	1.65%
randomForest	37.31%	1.71%
bagging	38.71%	1.66%
SDT	40.11%	2.06%
rpart	41.90%	1.78%
naive	43.59%	n.A.

Table 1: Estimated prediction errors derived from 10 repeats of a 10-fold cross-validation. "naive" is a rule that classifies all cases as "patient is metastase negative after five years". The estimated prediction error of the classifier "naive" thus corresponds to the fraction of investigated cases, which were of class "patient is metastase negative after five years": $estimated\ prediction\ error(classifier_{naive}) = 34/78 \approx 43.59\%$.

Defining the improvement of a classifier \hat{C} compared to the "naive" classification rule:

$$improvement = \frac{estimated\ prediction\ error(classifier_{naive}) - estimated\ prediction\ error(classifier_{\hat{C}})}{estimated\ prediction\ error(classifier_{naive})}$$

Figure 3 shows a visual comparison of the prediction accuracy of the five classifiers used:

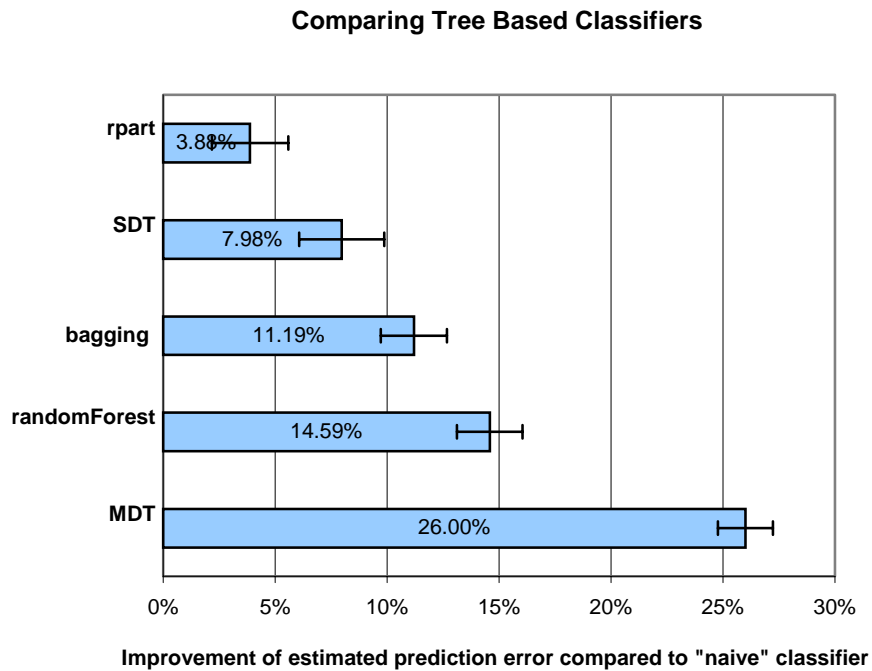


Figure 3: Chart showing the improvement of estimated prediction errors of the classifiers used. Multiple Decision Trees (MDT) outperform other classification algorithms. However, none of the classifiers built so far has a diagnostically relevant low estimated prediction error.

5. Discussion

The Multiple Decision Tree classifier implemented by Benedikt Brors and Patrick Warnat at the German Cancer Research Center (DKFZ), Heidelberg, clearly outperforms the other methods. This may be partially due to a different choice of splitting variables between the CART algorithm - implemented in `rpart` and used in bagging and `randomForest` - and Quinlan's C5.0 algorithm (see supplementary information B).

Furthermore, if a set of feature variables (genes) shows the same improvement of node impurity, CART randomly chooses one of these as a splitting variable, while C5.0 chooses the one that shows a maximum difference in its values between different classes. This was observed by Benedikt Brors and colleagues but has yet to be investigated systematically.

Yet, none of the classifiers was able to separate the two classes with high accuracy. This could have several reasons:

- 1) The data received had too much background noise or contained unprecieved systematic errors in data acquisition and preprocessing
- 2) The classification methods used are not appropriate for solving this explicit problem
- 3) There are more than two subtypes (classes) of breast tumors which differ in their gene expression profile as well as in their tendency to form metastases
- 4) The classification task is non-resolvable because the criterion for dividing the classes (= "patient is metastase negative/positive after five years") is chosen inappropriately

The last problem should be considered in detail. From a rational point of view, one cannot assume that there is a sharp boundary between breast tumors that form metastases within 5 years and those who do not. Because no other phenotypic criterion is used in the ab initio classification of the 78 tumor samples than the formation of metastases, one should suspect some cases to be classified rather deliberately. Therefore, the investigation of the van't Veer dataset can be seen as a primarily exploratory one.

But the fact that a clear separation of classes was not possible does not mean that there is no knowledge to extract from the investigations. The genes used as splitting variables can easily be extracted in all of the methods used. This work thus forms a starting point for future examinations of this dataset involving:

- 1) Extraction of a subset of genes that were used as splitting variables
- 2) Comparison of this subset with those 70 marker genes found by van't Veer *et al.*¹
- 3) Database research on the function of the most important genes from this subset
- 4) Building new classifiers that only use the genes extracted in 1)

A large gain in understanding the phenotypic differences between breast tumors forming metastases and those not forming metastases could possibly be derived from database research on these genes. Step no. 4) will hopefully produce even more accurate classifiers. Yet, as these classifiers would be built on a preselection of genes acquired using the available dataset, these classifiers would be biased, of course. Thus one will have to check if the classifiers acquired this way appropriately predict new cases. Opening the door to a further expansion of the methods applied, Thorsten Hothorn¹⁵ showed that the bundling of different predictors - using the functions implemented in the `ipred` package - is feasible and generally improves the predictive power of such a bundled predictor compared to single off-the-shelf approaches. As large the potential as numerous are the caveats to mention: The task of finding the most accurate classifier for a given problem cannot be solved from an absolute point of view. This obvious observation holds true for the entity of classifiers as well as for the subset of decision trees. The question whether the classifier used is the appropriate one has to be posed anew for each classification problem and can only be answered by comparing the accuracy of different approaches applying state of the art standards.

6. Acknowledgements

I thank the members of the Department "Intelligente Bioinformatiksysteme" at the German Cancer Research Center (DKFZ), Heidelberg for their advice and support. I especially thank Benedikt Brors for providing the basic ideas and information on the project and Patrick Warnat for his cooperation and counsel during my work at DKFZ, Heidelberg.

This work is part of the seminar "Mathematical Methods in Bioinformatics" initiated by Moritz Diehl, member of the Institut für Wissenschaftliches Rechnen (IWR) Heidelberg, whom I thank for his initiative and the steady impetus he put in the elaboration of every participant's project.

7. References

-
- ¹ Van't Veer, L *et al.* Gene expression profiling predicts clinical outcome of breast cancer. *Nature* **415** (6871):530-535 (2002)
 - ² Fodor SP, Rava RP, Huang XC, Pease AC, Holmes CP, Adams CL. Multiplexed biochemical assays with biological chip. *Nature* **364**(6437):555-556 (1993)
 - ³ Russo G., Zegar C., Giordano A. Advantages and limitations of microarray technology in human cancer. *Oncogene* **29**; 22(42):6497-6507 (2003)
 - ⁴ Van't Veer, L *et al.* Gene expression profiling predicts clinical outcome of breast cancer. *Nature* **415** (6871):530-535 (2002), *Supplementary Information*
 - ⁵ Dietterich T.G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation* **10**:1895-1923 (1998), preprint available at: <http://www.colorado.edu/ling/courses/Fall2003/7800/dietterich.pdf>
 - ⁶ Eibe F., Witten I. H. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* (1999), Morgan Kaufmann Publishers, **ISBN: 1558605525**
 - ⁷ Breiman L., Olshen, Friedman, Stone *Classification and Regression Trees.* (1984), Kluwer Academic Publishers, **ISBN: 0412048418**
 - ⁸ Therneau T. M., Atkinson E.J. An introduction to recursive partitioning using the rpart routines. (1997), available at: <http://course1.winona.edu/bdeppa/Stat%20425/Tutorials/techrep.ps>
 - ⁹ Quinlan J. R. *C4.5: Programs for machine learning.* (1993), Morgan Kaufmann Publishers, **ISBN: 1558602380**
 - ¹⁰ Breiman L. Bagging predictors. *Machine Learning*, **24**(2):123-140 (1996)
 - ¹¹ Breiman L. Random Forests. *Machine Learning* **45**(1):5-32 (2001)
 - ¹² Schoch C., Kohlmann A. *et al.* Acute myeloid leukemias with reciprocal rearrangements can be distinguished by specific gene expression profiles.
 - ¹³ Hothorn T. The ipred package. (2004), available at <http://lib.stat.cmu.edu/R/CRAN/doc/packages/ipred.pdf>
 - ¹⁴ Hothorn T. The randomForest package. (2004), available at <http://lib.stat.cmu.edu/R/CRAN/doc/packages/randomForest.pdf>
 - ¹⁵ Hothorn T. Bundling Predictors in R. (2003), available at: <http://www.imbe.med.uni-erlangen.de/~hothorn/talks/Rbundtalk.pdf>

A. rpart: Notation, prediction error and building a tree

This description is derived from the explanations of Therneau & Atkinson [1997]⁸. The description is simplified to suit the two class problem which was investigated. For a general overview of tree building algorithms see Breiman, Olshen, Friedman and Stone [1984]⁷ or Quinlan [1993]⁹.

Notation:

$\tau(\bar{c})$	True class of a case \bar{c}
$\pi_1 \pi_2$	Prior probabilities of classes 1 and 2. Equal to the frequencies of the corresponding class in the training set L : $\pi_i = P(\tau(\bar{c}) = i \bar{c} \in L)$ $L = \{(\bar{x}^1, i^1), \dots, (\bar{x}^n, i^n)\}$
A	Node of the tree, represented by the subset of cases of the training set which are in $A \subseteq L$
$\tau(A)$	Class assigned to node A , if A is a leaf node
A_L, A_R	Left and Right daughter node of A
n_i	Number of observations in the sample that are class i
n_A	Number of observations in node A
n_{iA}	Number of observations that are class i and in node A
$P(A)$	Probability of A (for future observations) $= \pi_1 P(\bar{c} \in A \tau(\bar{c}) = 1) + \pi_2 P(\bar{c} \in A \tau(\bar{c}) = 2)$ $\approx \frac{\pi_1 n_{1A}}{n_1} + \frac{\pi_2 n_{2A}}{n_2}$
$p(i A)$	Probability of a case \bar{c} being of class i , given \bar{c} is in A $= P(\tau(\bar{c}) = i \bar{c} \in A)$ $= \frac{\pi_i P(\bar{c} \in A \tau(\bar{c}) = i)}{P(\bar{c} \in A)} = \frac{\frac{\pi_i n_{iA}}{n_i}}{\sum \frac{\pi_i n_{iA}}{n_i}} = \frac{n_{iA}}{n_A}$
$R(A)$	Risk (prediction error) of node A $= \sum_{i=1}^2 p(i A) I(\tau(\bar{c}) \neq \tau(A)) = \frac{\# \text{misclassified}_{\text{node } A}}{\# \text{cases}_{\text{node } A}}$
$R(T)$	Risk (prediction error) of a tree T with k terminal nodes $= \sum_{j=1}^l P(A_j) R(A_j) = \sum_{j=1}^l \left(\frac{\pi_1 n_{1A_j}}{n_1} + \frac{\pi_2 n_{2A_j}}{n_2} \right) \left(\sum_{i=1}^2 p(i A_j) I(\tau(\bar{c}) \neq \tau(A_j)) \right)$ $= \frac{\# \text{missclassified}_{\text{tree}}}{\# \text{cases}_{\text{tree}}}$

Finding splitting variable and value

Given a set of N cases, let $f(p_{iA})$ be some impurity function and define the impurity of node A as:

$$I(A) = \sum_{i=1}^2 f(p_{iA}) \quad (1)$$

where p_{iA} is the proportion of cases in A that belong to class i . In our classification problem, rpart used the Gini index as an impurity function:

$$f(p) = p(1-p) \quad (2)$$

For each split, rpart chooses the splitting variable (gene) and value resulting in maximum impurity reduction:

$$\Delta I(A) = P(A)I(A) - P(A_L)I(A_L) - P(A_R)I(A_R) \quad (3)$$

This is repeated until no further impurity reduction can be achieved or in case a stopping rule is applied. The stopping rule can either determine a minimum number of cases in a terminal node or can be a threshold number of splits.

B. Difference between C5.0 and rpart in finding the splitting rule

C5.0 uses a slightly different criterion for finding the optimal splitting variable and its corresponding splitting value. The algorithm uses the information gain accomplished by dividing a set of cases at node A . The possible splitting rule for A is symbolized by S :

$$\begin{aligned} \text{gain}(S) &= \text{info}(A) - \text{info}_S(A) \\ &= -\sum_{i=1}^2 \frac{n_{iA}}{n_A} \log_2 \left(\frac{n_{iA}}{n_A} \right) + \sum_{i=1}^2 \frac{n_{iAL}}{n_A} \log_2 \left(\frac{n_{iAL}}{n_{AL}} \right) + \sum_{i=1}^2 \frac{n_{iAR}}{n_A} \log_2 \left(\frac{n_{iAR}}{n_{AR}} \right) \end{aligned} \quad (4)$$

and the potential information generated by dividing the set of cases in node A into two subsets:

$$\text{splitinfo}(S) = -\frac{n_{iAL}}{n_A} \log_2 \left(\frac{n_{iAL}}{n_A} \right) - \frac{n_{iAR}}{n_A} \log_2 \left(\frac{n_{iAR}}{n_A} \right) \quad (5)$$

to define a gain ratio:

$$\text{gain ratio}(S) = \frac{\text{gain}(S)}{\text{splitinfo}(S)} \quad (6)$$

Supplementary Information

As a splitting criterion, the variable and value have to be found which maximize this gain ratio. This ensures that, for each split, the variable and value are assessed which result not only in a maximum information gain, but rather in an equal distribution of cases to the two daughter nodes A_L and A_R .

Another difference between rpart and C5.0 is the determination of the splitting value. Suppose a splitting variable s and the best partitioning of cases into daughter nodes A_L and A_R have been determined:

$$A_L = \{(\bar{x}^1, i^1), \dots, (\bar{x}^b, i^b)\} \quad (7)$$

$$A_R = \{(\bar{x}^{b+1}, i^{b+1}), \dots, (\bar{x}^n, i^n)\} \quad (8)$$

In rpart, the splitting value s_{value} is defined as the midpoint of the two bordering values:

$$s_{\text{value}} = \frac{s(\bar{x}^b) + s(\bar{x}^{b+1})}{2} \quad (9)$$

, with $s(\bar{x})$ being the value of the splitting variable s in a distinct feature vector \bar{x}^b

Whereas in C5.0, the optimal splitting value is determined as the largest value of the splitting variable s in the entire training set that does not exceed the midpoint of the feature values above the split point. This ensures that the highest possible splitting value is used which occurs in the training set:

$$s_{\text{value}} = s(\bar{x}^m) \quad (10)$$

$$\bar{x}^m = \max_{\bar{x}} \left(s(\bar{x}) \leq \frac{s(\bar{x}^{b+1}) + s(\bar{x}^{b+2})}{2} \right)$$

C. R script used for data preparation (kindly provided by Patrick Warnat[‡])

```
## R-script for preprocessing microarray data of the van't Veer Study,  
Nature(2002)  
## Array Data has been retrieved from  
http://www.rii.com/publications/2002/vantveer.htm and saved as csv format.  
## This script extracts the expression ratios for those genes which are  
significantly regulated across the two groups of samples (that is, at least  
a twofold difference and a p-Value of less than 0.01 in more than 3 of 78  
tumors) as described in the publication.  
## The result is written in a tabulator delimited text file.  
  
veer.15yr <- matrix(scan('aDataL5yrReal.csv', sep=',', n = 24481*102),  
24481, 102, byrow = TRUE)  
veer.15yr.ratioSelSequence <- seq(2,101,3)  
veer.15yr.ratioMatrix <-  
veer.15yr[1:(dim(veer.15yr)[1]),veer.15yr.ratioSelSequence]  
veer.15yr.pvalueSelSequence <- seq(3,102,3)  
veer.15yr.pvalueMatrix <-  
veer.15yr[1:(dim(veer.15yr)[1]),veer.15yr.pvalueSelSequence]  
veer.15yr.conditionMatrix <- ((veer.15yr.ratioMatrix > log10(2)) |
```

Supplementary Information

```
(veer.l5yr.ratioMatrix < log10(0.5))) & (veer.l5yr.pvalueMatrix < 0.01)
veer.l5yr.GeneIndex <- apply(veer.l5yr.conditionMatrix,1,sum)

veer.g5yr <- matrix(scan('aDataG5yrReal.csv', sep=',', n = 24481*132),
24481, 132, byrow = TRUE)
veer.g5yr.ratioSelSequence <- seq(2,131,3)
veer.g5yr.ratioMatrix <-
veer.g5yr[1:(dim(veer.g5yr)[1]),veer.g5yr.ratioSelSequence]
veer.g5yr.pvalueSelSequence <- seq(3,132,3)
veer.g5yr.pvalueMatrix <-
veer.g5yr[1:(dim(veer.g5yr)[1]),veer.g5yr.pvalueSelSequence]

veer.g5yr.conditionMatrix <- ((veer.g5yr.ratioMatrix > log10(2)) |
(veer.g5yr.ratioMatrix < log10(0.5))) & (veer.g5yr.pvalueMatrix < 0.01)
veer.g5yr.GeneIndex <- apply(veer.g5yr.conditionMatrix,1,sum)

veer.GeneIndex <- veer.l5yr.GeneIndex + veer.g5yr.GeneIndex

veer.GeneNames <- as.matrix(scan('ArrayData_less_than_5yr.csv', what="",
sep=',', skip=2,flush=T,multi.line=F,n=24481))
veer.l5yr.SampleNames <- c(scan('ArrayData_less_than_5yr.csv', what="",
sep=',',nlines=1))
veer.l5yr.sampleSelSequence <- seq(3,102,3)
veer.l5yr.SampleNames <- veer.l5yr.SampleNames[veer.l5yr.sampleSelSequence]

veer.g5yr.SampleNames <- c(scan('ArrayData_greater_than_5yr.csv', what="",
sep=',',nlines=1))
veer.g5yr.sampleSelSequence <- seq(3,132,3)
veer.g5yr.SampleNames <- veer.g5yr.SampleNames[veer.g5yr.sampleSelSequence]

veer.ratioMatrix <- cbind(veer.l5yr.ratioMatrix,veer.g5yr.ratioMatrix)
rownames(veer.ratioMatrix) <- veer.GeneNames
colnames(veer.ratioMatrix) <- c(veer.l5yr.SampleNames,
veer.g5yr.SampleNames)

veer.selectedRatioMatrix <- veer.ratioMatrix[(veer.GeneIndex > 3),]

veer.data <-
cbind(t(veer.selectedRatioMatrix),Target=rep(c("15yr","g5yr"),c(34,44)) )
write.table(veer.data, file = "veerData-m.dat", quote = FALSE, sep = '\t',
eol = "\n", na = "NA", dec = ".", row.names = TRUE,
col.names = TRUE)
```

D. R script for building and comparing the classifiers

```

## R-script for the analysis of microarray data of the van't Veer Study,
Nature 415 (6871):530-535 (2002)
## Original data was retrieved from
http://www.rii.com/publications/2002/vantveer.htm
## Feature reduction to 4917 genes was conducted by Patrick Warnat, ibios
group, DKFZ Heidelberg as described in van't Veer(2002) (see above)
## Training and test data (100 tables each) were extracted from a previous
classification run using the multiple decision tree classifier as described
in Proc Natl Acad Sci U S A. 2002 July 23;99 (15): pages 10008-10013.
## Methods used here: (i) rpart, (ii) bagging, (iii) randomForest
## Results for each classifier are saved as tab delimited .txt files
## A comparison of the Methods is implemented at the end of this script

### loading required packages ###
require(rpart)
require(ipred)
require(randomForest)

### defining general variables ###
# class labels are situated in column$class.col$
class.col<-4919

#Building the formula for rpart using the colname of column$class.col$
Target<-formula(paste(colnames(VD)[class.col],"~.", sep=""))

# The total number of cases is required to compute a minsplit.number for
the rpart run. This minsplit.number guarantees that only stumps are
produced.
total.no.of.cases<-dim(VD)[1]
minsplit.rpart<- total.no.of.cases-round(total.no.of.cases/10)

# Enter the number of cases with the most frequent class label in the
original dataset. This is used for computing the accuracy of a "naïve"
classifier that classifies all cases as being of the most frequent class
encountered in the original dataset.
no.of.most.freq.class<-44
naive.error<-(total.no.of.cases-no.of.most.freq.class)/total.no.of.cases

##### Analysis of the van't Veer data#####

#loading van't Veer dataset, kindly provided by Patrick Warnat
VD<-read.csv("veerData-m.dat", sep="\t", header=TRUE)

#remove NA
VD.mat<-as.matrix(VD[,-c(1,class.col)])
VD.mat[which(is.na(VD.mat))]<-0
VD0<-VD
VD0[,-c(1,class.col)]<-VD.mat

```

Supplementary Information

```
###Single tree method: rpart###
#loop for rpart: 10fold cv (running variable i), repeated 10 times (running
variable j)
# define vector of sample sizes for 10fold cv:
# cv<-kfoldcv(10,dim(VD)[1], nlevel=NULL)

# define index vector s for subset extraction (needed in forthcoming loop):
# s<-c(1:dim(VD)[1])

for (j in 1:10) {
cv<-kfoldcv(10,dim(VD)[1], nlevel=NULL)
s<-c(1:dim(VD)[1])
newprob<-rep(1, times=dim(VD)[1]) #vector of probabilities, reset to
P=1/dim(VD)[1] for each cv loop

for (i in 1:10) {
ex<-sample(s, cv[i], replace=FALSE, prob=newprob)
VD.train<-VD0[-ex,]
VD.test<-VD0[ex,]
newprob[ex]<-0 #making sure that no dataset is used twice
# building classifier
r<-rpart(Target, VD.train, xval=0, minsplit=minsplit.rpart) #only stumps
are produced
# extracting vector of predictions using test data
rpart.predicted<-predict(r, VD.test, type="class")
#concatenating error rates
if (i==1)
rpart.err<-
mean(as.character(rpart.predicted)!=as.character(VD.test[,class.col])) else
rpart.err<-c(rpart.err,
mean(as.character(rpart.predicted)!=as.character(VD.test[,class.col])))
# concatenating predictions
if (i==1)
rpart.predicted.total<-cbind(rpart.predicted, VD.test[,class.col]) else
rpart.predicted.total<-rbind(rpart.predicted.total, cbind(rpart.predicted,
VD.test[,class.col]))
rm(r, VD.train, VD.test)
for (b in 1:10) gc()
}

# save results
write.matrix(rpart.err, file="rpart_error.txt", sep="\t")
write.matrix(rpart.predicted.total, file="rpart_predicted.txt", sep="\t")
```

Supplementary Information

```
### Ensemble method No.1: bagging ###
##10fold cv for bagging

for (j in 1:10) {
cv<-kfoldcv(10,dim(VD)[1], nlevel=NULL)
s<-c(1:dim(VD)[1])
newprob<-rep(1, times=dim(VD)[1]) #vector of probabilities, reset to
P=1/dim(VD)[1] for each cv loop

for (i in 1:10) {
ex<-sample(s, cv[i], replace=FALSE, prob=newprob)
VD.train<-VD0[-ex,]
VD.test<-VD0[ex,]
newprob[ex]<-0 #making sure that no dataset is used twice
bagg<-ipredbagg.factor(VD.train[,class.col], VD.train[,-c(1,class.col)],
control=rpart.control(xval=0), coob=FALSE, comb=NULL, nbagg=10,
keepX=FALSE)
for (h in 1:3) {
if (h==1 && i==1 && j==1) bagg.features<-bagg$mtree[[h]]$btree$splits else
bagg.features<-rbind(bagg.features, bagg$mtree[[h]]$btree$splits)
}
bagg.predicted<-as.character(predict(bagg, VD.test[, -c(1,class.col)],
type="class", aggregation="majority"))
if (i==1 && j==1) bagg.features.total<-bagg.features else
bagg.features.total<-rbind (bagg.features.total, bagg.features) #extracting
the features used in bagging
if (i==1 && j==1) bagg.err<-
mean(bagg.predicted!=as.character(VD.test[,class.col])) else bagg.err<-
c(bagg.err, mean(bagg.predicted!=as.character(VD.test[,class.col])))
rm(bagg,bagg.predicted, VD.train, VD.test)
for (b in 1:10) gc()
}
}

# save results
write.matrix(bagg.err, file="bagg_error.txt", sep="\t")
write.matrix(cbind (row.names(bagg.features.total), bagg.features.total),
file="bagg_features.txt", sep="\t") #saving the features used in bagging

### Ensemble method No.2: randomForest ###
## 10fold cv for randomForest

for (j in 1:10) {
cv<-kfoldcv(10,dim(VD)[1], nlevel=NULL)
s<-c(1:dim(VD)[1])
newprob<-rep(1, times=dim(VD)[1]) #vector of probabilities, reset to
P=1/dim(VD)[1] for each cv loop

for (i in 1:10) {
ex<-sample(s, cv[i], replace=FALSE, prob=newprob)
VD.train<-VD0[-ex,]
VD.test<-VD0[ex,]
newprob[ex]<-0 #making sure that no dataset is used twice
cv.para<-kfoldcv(3, dim(VD.train)[1], nlevel=NULL) #3fold cv for parameter
optimization
s.para<-c(1:dim(VD.train)[1])
```

Supplementary Information

```
newprob.para<-rep(1,times=dim(VD.train)[1])
for (h in 1:3) {
ex.para<-sample(s.para,cv.para[h], replace=FALSE, prob=newprob.para)
VD.train.para<-VD.train[-ex.para,]
VD.test.para<-VD.train[ex.para,]
newprob.para[ex.para]<-0 #making sure that no dataset is used twice
rF.para<-tunerRF(VD.train.para[,-c(1,class.col)], VD.train.para[,class.col],
trace=FALSE, xtest=VD.test.para[,-c(1,class.col)],
ytest=VD.test.para[,class.col], plot=FALSE, doBest=TRUE)
if (h==1) rF.para.err<-rF.para$test$err.rate else rF.para.err<-
cbind(rF.para.err, rF.para$test$err.rate)
rm(rF.para,VD.train.para, VD.test.para)
}
rF.para.err.mean<-apply(rF.para.err, 1, mean)
ntree.opt<-which.min(rF.para.err.mean)
rF<-tunerRF(VD.train[,-c(1,class.col)], VD.train[,class.col], trace=FALSE,
plot=FALSE, importance=TRUE, xtest=VD.test[,-c(1,class.col)],
ytest=VD.test[,class.col], doBest=TRUE,ntree=ntree.opt)
if (i==1 && j==1) rF.conf<-rF$test$confusion else rF.conf<-
as.data.frame(rbind(rF.conf, rF$test$confusion))
if (i==1 && j==1)
rF.err.rate<-rF$test$err.rate[ntree.opt] else rF.err.rate<-c(rF.err.rate,
rF$test$err.rate[ntree.opt])
if (i==1 && j==1) rF.err.rate.1to500trees<-rF$test$err.rate else
if (i==1 && j==1) rF.importance<-rF$importance else rF.importance<-
as.data.frame(cbind(rF.importance, rF$importance))
if (i==1 && j==1) rF.para.err.mean.save<-rF.para.err.mean else
rF.para.err.mean.save<-as.data.frame(rbind(rF.para.err.mean.save,
rF.para.err.mean))
}
}

#save results
write.matrix(rF.conf, file="rF_confusion.txt", sep="\t")
write.matrix(rF.err.rate, file="rF_err_rate.txt", sep="\t")
write.matrix(rF.importance, file="rF_importance.txt", sep="\t")
```

Supplementary Information

```
### compute error rates ###
#loading results from rpart run
err.rpart<-read.table("rpart_error.txt", sep="\t", header=TRUE)

#extract ERRORS from randomForest classifiers
#loading results from randomForest classifier (ntree=500)
rF.conf<-read.table("rF_confusion.txt", sep="\t", header=TRUE)

#extracting number of false predictions
false.g5yr<-sum(rF.conf[seq(2,200, by=2),1])
false.l5yr<-sum(rF.conf[seq(1,199, by=2),2])

#calculate error rate
total.cases<-sum(c(rF.conf[,1], rF.conf[,2]))
err.rF<-(false.l5yr+false.g5yr)/total.cases

#loading results from bagging
bagg.err<-read.table(file="bagg_error.txt", sep="\t")

#loading results from mineit
err.SDT<-read.table("resultsSDT/CrVal_1.out", sep="\t")
err.MDT<-read.table("resultsMDT/CrVal_1.out", sep="\t")

#build dataframe
RES<-as.data.frame(c(mean(err.SDT),mean(err.MDT),mean(err.rpart), err.rF,
mean(bagg.err), naive.error))
rownames(RES)<-c("SDT", "MDT", "rpart", "randomForest", "bagging", "naiv")
colnames(RES)<-c("Error rate derived from 10 repeats of a 10fold
CV", "stderr")

#plot results
x<-as.factor(rownames(RES))
y<-as.vector(RES[,1])
plot(x,y,main=colnames(RES), xlab="Method", ylab="Prediction error")

#print results
postscript("results_classifier_comparison.ps")
plot(x,y,main=colnames(RES), xlab="Method", ylab="Prediction error")
dev.off()
```