Simon Urbanek

KLIMT: **K**lassification - **I**nteractive **M**ethods for **T**rees

Usage and application of KLIMT

Contents

Chapter 1

Usage of KLIMT

1.1 Getting started

KLIMT is a stand-alone application but intended for use in conjunction with R. This chapter gives an overview of the main features and basic introduction to the usage of KLIMT. Processing and preparation of the data in R is not illustrated here, I assume that the user is familiar at least with the basic functionality of R, such as loading of datasets and libraries. Furthermore the unidirectional file interface will be used, because the flexibility of the $\hat{\Omega}$ interface is beyond the the scope of this work. An example for $\hat{\Omega}$ interface can be found in the appendix.

Once in R the quick start to KLIMT with an existing dataset looks like this:

```
source("klimt.r");
d<-read.table("mydataset.txt");
t<-tree(OUT2~.,d);
Klimt(t,d);
```

First line loads the functions necessary for the interface to KLIMT from the file `klimt.r`. As a side effect the Ripley's *tree* library is loaded as well. Second line loads a dataset from external file to the variable `d`. If you already have a dataset or want to use R's internal datasets you can use the corresponding function of R to load the dataset to the variable `d`. In the third line a tree is built using default parameters of the `tree` function. The model is given as `OUT2~.` meaning that `OUT2` is the classification variable and all other

variables should be used to build the classifier. The call does not differ for regression trees, the correct tree is chosen corresponding to the kind of the dependent variable. At this point any additional parameters influencing the tree growth can be applied, such as the choice of the criterion or stopping rules (`mincut`, `minsize`, `nmax`, etc. - see documentation for tree.control and tree). Finally KLIMT is started with the tree `t` and the dataset `d`.

1.2 Variables

When KLIMT is started, two main windows appear: a *variables window* and a *tree window*. Variables window lists all variables of the dataset and is the central place for any actions when the dataset is concerned. In this example the dataset consists of one index variable, four numerical variables `ALTER`, `FIE`, `ZZLQ` and `GRA`, and one categorical variable `OUT2` as illustrated in Fig. 1.1. Beside each variable name red letters denote the type of the variable. `N` stands numerical, `S` for string and `C` for categorical variable. String variable must be categorical, but a numerical variable can be either categorical or continuous. The star symbol (*) appears beside variables that contain missing values. Single click on a variable name selects the variable. Multiple variables can be selected by holding the <Shift> button. In the bottom part of the window various commands can be chosen. Clicking on *Exit* closes the entire KLIMT application. If you generated more trees with the same dataset you can open additional tree files by clicking the *Open tree* button. As long as the newly read tree uses the same dataset[1] it will be linked with all other plots and existing trees. *Hist/Bar* button displays histograms or barcharts for all selected variables - for continuous variables histograms are plotted, for categorical variables barcharts will be shown. If exactly two variables are selected, a scatterplot or parallel dotplot is drawn, depending on the type of each variable.

[1] As the tree file contains variable names only KLIMT will let you load tree of any dataset as long as the variables used in splits and the classes associated with leaves are present in the currently selected dataset. The highlighting and selection will be based on the splitting rules applied to the current dataset.

Figure 1.1: Variables window.

1.3 Working with the tree

1.3.1 Many faces of a tree

The tree window is the main place for all actions related to the tree. The initial look of the tree is shown in Fig. 1.2. Each node is represented by a rectangle with a class name written inside. The size of each rectangle is proportional to the number of cases in the node. Most recent splitting rule is drawn above the node (except for the root node). From the center of each rectangle a straight line is drawn to its children. Every child node is placed on the level directly below the parent.

Various different properties of the visualization can be modified:

- *node size*
  The size of the rectangle representing a node can be either of fixed size or proportional to the number of nodes. Fixed mode usually gives better overview and allows the comparison of proportions within the nodes. Variable size is useful to compare the weight of a branch - branches consisting of mainly small nodes may be of less interest if the categories are of similar size. The node size can be toggled with the `s` key[2].

[2]All properties can also be modified using the View menu.

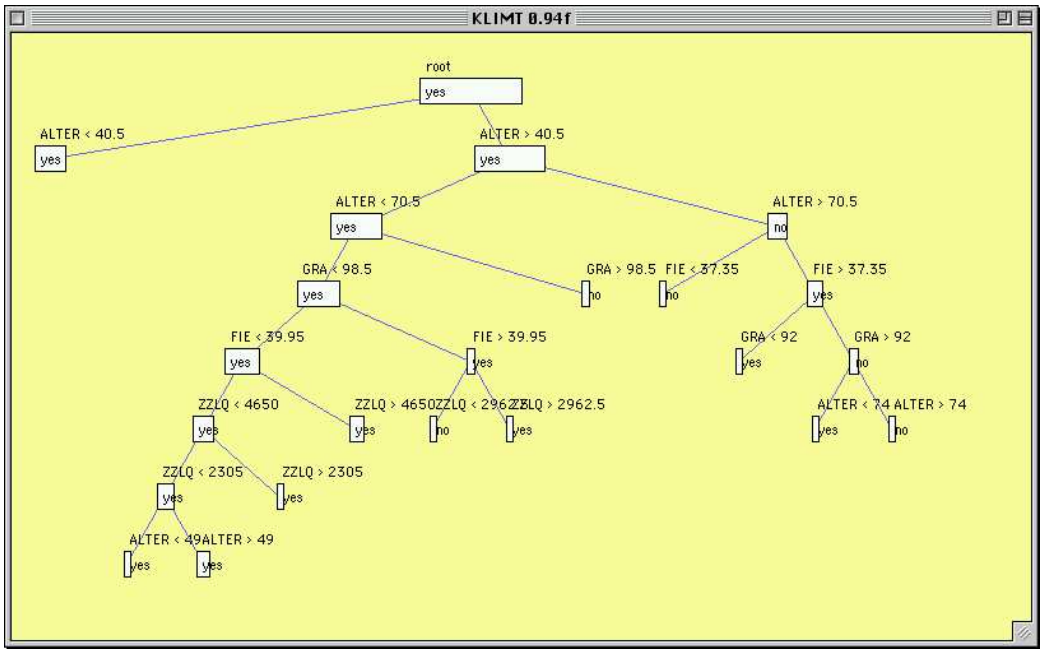

Figure 1.2: Tree window.

- *connecting lines*
  Mostly two different kinds of connecting lines are used. Straight lines as in Fig. 1.2 resemble natural trees but can cause overlapping if nodes are moved downwards. Therefore often the combination of vertical and horizontal lines is used instead, giving the tree a "rods-and-wires" look. In this mode nodes can be dropped down to the lowest level without lines crossing other nodes. The type of connection lines can be toggled with the `c` key.

- *leaves*
  The terminal nodes - leaves - can be either placed below their parents or "dropped" down to a common level - usually the level of the deepest leaf. Such dropping makes visual comparison among all leaves much easier and allows direct identification with a spineplot of leafs as will shown later in section 1.4. The arrangement of final nodes can be toggled with the `f` key.

- *labels*
  Especially for larger trees labels may clutter the tree and therefore KLIMT allows them to be switched on and off by pressing the `l` key.

- *orientation*
  Usually trees are plotted in top-bottom orientation. For deep trees this may cause problems because the screen has usually more room in horizontal than vertical direction. KLIMT allows the tree to be displayed in left-right orientation. Toggling between the two orientations is possible with the `Shift+r` key.

- *criterion*
  KLIMT offers a way to visualize the criterion used during the tree construction. If deviance was used then deviance gain in each inner node is represented by circles and remaining deviance in leaves is represented by rectangles. The size of each symbol is proportional to the corresponding value. This way it is easy to visually detect at a glance pure leaves (no rectangle drawn because deviance is zero), good splits (large circles) or problematic leaves (large rectangles meaning very impure nodes).

At any time you can tell KLIMT to re-arrange the tree using its node placement algorithm by pressing the `r` key. The user is not limited to the

trees designed by KLIMT, he can freely move any node by dragging it with left mouse button. If no modifier key is pressed the entire branch is moved, i.e. the entire subtree with the dragged node as root. If the <Ctrl> button is held down during the dragging, only the selected node is moved. Finally holding the <Shift> key causes the movement to be constrained to one direction.

1.3.2 Tool modes

While working with the tree KLIMT offers four different tool modes. Each mode is identified by different cursor shape.

- *Selection*
  This is the default mode when KLIMT is started and is identified by cursor having the form of an arrow. In this mode you can modify the tree and drag nodes. When you click on a node, all cases of that node are selected as well as the node itself. The case-wise selection will be discussed in detail later in section 1.5. Querying a node is done by holding the <Alt>[3] key during the click. A small query information window will appear with details about the node. Extended query can be obtained by holding both the <Shift> and <Alt> key. You can switch to this mode by pressing the `e` key[4].

- *Node picker*
  This mode allows to select a node without selecting cases in the node. This is useful in conjunction with other plots to identify a node without changing the current case selection. The query is possible in this mode as well. The cursor will have the form of a pointing hand in this mode and the corresponding key to switch to this mode is `n`.

- *Move*
  In this mode dragging the mouse causes the entire working area to be moved in the direction of the dragging. This mode can be temporarily turned on by holding the <Space> key. KLIMT returns to previously used mode as soon as the key is released. Alternatively you can switch to this mode by pressing the `v` key.

[3]The Alt key is on some platforms called the meta-key.
[4]Modes can also be switched via the Tools menu.

- *Zoom*
  Finally KLIMT provides zooming facilities. In this mode, which can be identified by a cursor in form of a magnification glass[5], clicking causes the zoom factor to be doubled, resulting in a zoom-in. If the <Shift> button is held during the click zoom-out is performed instead. The zoom center is always the point where the mouse points while clicking. KLIMT supports logical zoom which means that node symbols are changed to small circles and labels are shortened or left out entirely when zooming out considerably. The zoom mode is activated by pressing the `z` key.

1.3.3 Pruning

Besides the tree visualization properties KLIMT supports manual pruning. After selecting a node with the selection tool or node picker one can go to the menu item Node→Prune or press the `p` key to chop off all nodes below the current node. A small plus sign in a circle beside the node indicates that pruning has been performed. The pruning can be undone by clicking on the plus sign.

The pruned nodes are not visible but still considered when the tree layout is designed or the criterion display is drawn. In order for the nodes to be pruned permanently it is necessary to create a new, pruned tree by pressing the `Shift+n` key. The new tree is still linked to the old one because it uses the same dataset, but layout and visualization is done independently.

1.4 Plots

Visualization of the tree itself does not provide enough insight in the structure and dependencies of the dataset. Trees display the modeled structure but don't reveal much information about single cases. Therefore it is necessary to provide at least some basic plots to be used for the analysis. KLIMT currently supports following plots:

- Histograms
- Barcharts

[5] On some platforms that don't have this cursor type available and don't support custom cursors the cross-hair cursor is used instead.

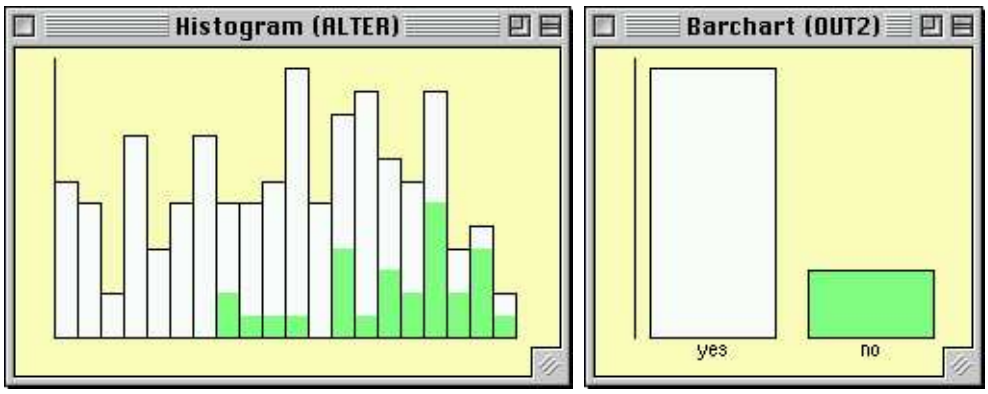

Figure 1.3: A histogram and a barchart.

- Scatterplots
- Treemaps
- Spineplots for leaves

Further plots such as boxplots or regular spineplots as well as special plots in particular tree size vs. misclassification plot are planed for the future.

Histograms and barcharts

Histograms and barcharts are displayed when one or more variables are selected in *Variables* window and the user clicks on the *Hist/Bar* button. For categorical variables barchart and for continuous variables histogram is used. If more variables are selected one plot for each variable is drawn. The resulting plots are shown in Fig. 1.3. Unlike for a tree there is no complex set of tools, only case-wise selection is used. The sequence of the bars in a barchart can be changed by the drag and drop method. The anchor point and bin width of an histogram can be interactively adjusted by dragging the corresponding symbol in left bottom corner. One additional feature of a histogram is the possibility to display the current splitting rule. If a node is selected in a tree and the variable used in the split is the same as the variable of the histogram then the split value is drawn as a red vertical line in the plot.

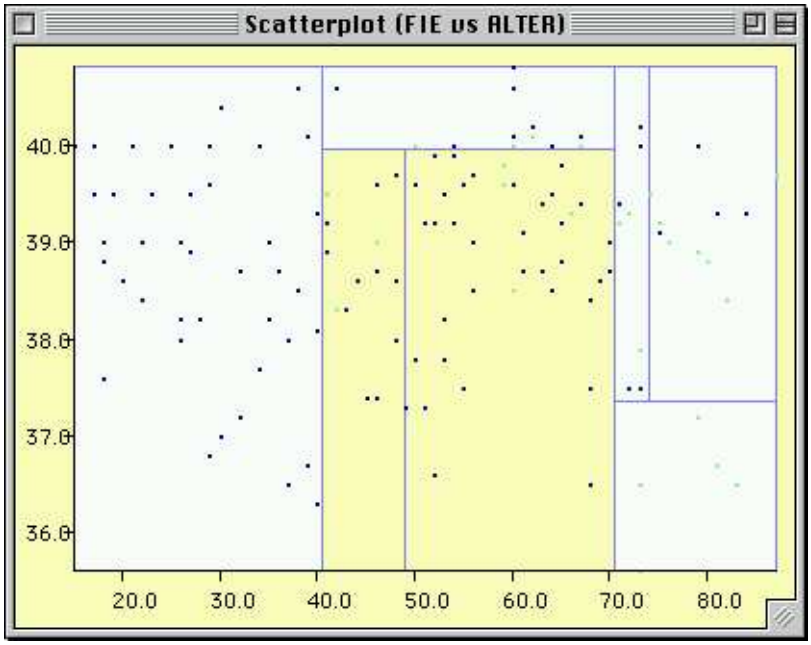

Figure 1.4: An enhanced scatterplot.

Scatterplots

Scatterplots can be displayed in KLIMT by selecting two variables in the *Variable* window and clicking on *Scatter*. If no node is selected in any tree then a regular scatterplot is plotted. Labels on the axes can be toggled by pressing the `l` key similar as in the tree window. The axes can be swapped by pressing the `Shift+r` key which stands for rotate.

The true value of scatterplots in conjunction with trees becomes visible when a node is selected in the tree window. A scatterplot is a two dimensional projection of the data space. A tree can be seen as a rectangular partition of that space. Therefore it is possible to plot that partition in the scatterplot. The result can be seen in Fig. 1.4. The blue lines represent splits in the tree and the yellow filled area corresponds to the currently selected node. If the splits used in the tree contain also other variables than the ones used in the scatterplot, the number of partitions in the scatterplot is smaller than the number of leaves of the tree. This is due to splits in additional dimensions that are not visible in the two dimensional projection. Sometimes plotting

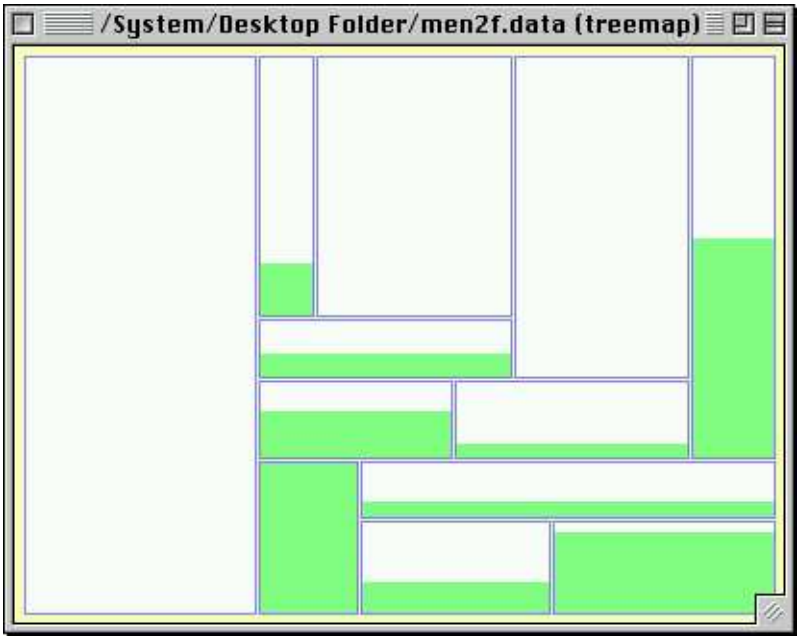

Figure 1.5: A treemap.

several scatterplots of different variable combinations may help to detect the structure and possible weaknesses of the model.

All plots discussed so far are independent of the actual tree and the user can draw any number of such plots. Although they offer some additional features when used in conjunction with a tree, they are still based only on the dataset. The following plots are not based on the dataset alone but on the tree structure and are therefore tightly connected to the tree. Therefore one tree can have only one of these plots associated.

Treemaps and spineplots for leaves

The treemap can be obtained by pressing the `m` key in a tree window or by selecting View→Show treemap in the menu. A typical treemap can be seen in Fig. 1.5. The idea of a treemap as described in **??** is to partition an area proportionally according to a split in each node. Unlike in scatterplots where only splits of the two plotted variables can be seen, in treemaps the area is split in exactly as many partitions as there are leaves. The area of each

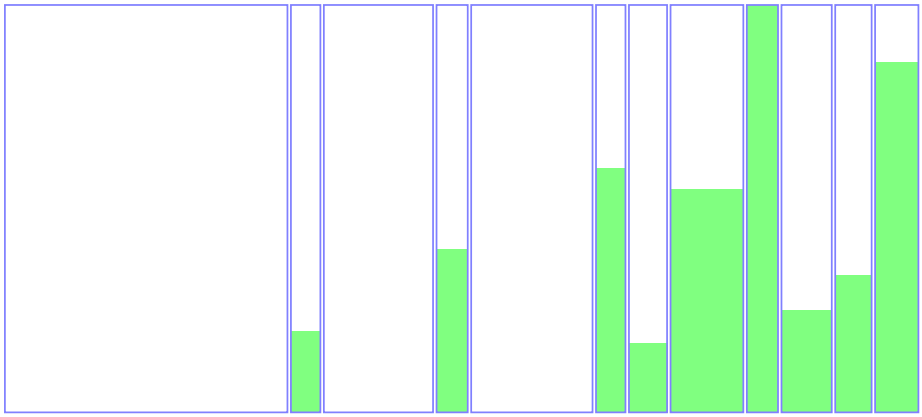

Figure 1.6: A spineplot of leaves.

partition is proportional to the number of cases in the leaf. In a treemap the orientation of the first split can be toggled by pressing the `Shift+r` key which stands for rotate again, just like in scatterplots and tree window.

Technically a spineplot of leaves as shown in Fig. 1.6 is a treemap where the orientation of splits is not alternated. To turn the split alternation on and off the `a` key can be used. In a spineplot the size of each leaf corresponds to the width of the corresponding bar. If the user didn't reorder the leaves in the tree window then the sequence of leaves in the tree window matches exactly the sequence of the leaves in the spineplot.

1.5 Selection

One of basic principles of interactive graphics is the use of linked highlighting. The idea is that selected cases of the dataset are highlighted in every plot of the system. In KLIMT selected cases are displayed in another color than non-selected cases. In previous figures 1.3-1.6 green color denotes the selected cases. The selection is done either by clicking or dragging. In histograms or barcharts clicking selects all cases of a bar, in a tree window it selects all cases of a node and in a scatterplot single case is selected. In scatterplots it is possible to select an entire area of cases by dragging a selection rectangle. When no additional keys are pressed the recently selected cases will be highlighted. Modifier keys change that behavior.

To conveniently describe what happens let A denote the set of cases highlighted before the selection, B the set of currently selected cases and C the set of highlighted cases after the selection. Without any key depressed the action equals to $C = B$. Holding the <Shift> button during[6] the selection results in $C = (A \cup B) \setminus (A \cap B)$, in words: the state of selected cases is inverted ($C \cap B = \bar{A} \cap B$), all other cases are untouched ($C \cap \bar{B} = A \cap \bar{B}$). Holding both <Shift> and <Ctrl> buttons results in $C = A \cup B$ hence equals to "adding" new cases. This selection rules apply to all plots.

1.6 Export features

There are two features that are available in all graphics. Pressing `Shift+p` causes KLIMT to produce a *PostScript* file of the current plot[7]. It is not merely a hardcopy snapshot, but fully vectorized output therefore suitable for publication or further processing. This file can be sent directly to a printer.

KLIMT's PoGraSS classes offer also another portable output: the PGS meta format. By pressing the `Shift+x` button for "export" a meta-file output.pgs of the current plot is created. This file can be viewed or converted to any other format which is supported by PoGraSS as described in section **??**.

1.7 Import and missing values

KLIMT supports missing values to that extent that it accepts them and tries to handle them separately when necessary. When building linked highlighting, cases are dropped down the tree as far as they go without the need of evaluation of missing values. This means that the cases of children nodes do not necessarily join to build a parent node. For categorical variables the missing values build an own category denoted `NA`. In continuous variables they have no value (i.e. 0 is not assumed) and therefore they do not influence the min/max functions. Continuous plots currently don't display missing values, but in all other plots (including a tree) they can be selected just like any other values.

[6] More precisely: relevant is the state of keys in the moment of mouse button release.

[7] The file is called output.ps and contains a bounding box which makes it EPS compatible and flexibly reusable.

When importing a dataset KLIMT understands three notations for a missing value: the `NA` string as used by R, `?` as often used on PC platforms and Mac's n/a symbol (ASCII 0xA5). In general KLIMT was made to accept the format produced by `write.table` function of R. Similar ASCII formats such as space delimited values are supported as well. For more obscure formats either R itself or my recently developed DaMaT (Data Management Tool) can be used to obtain a format acceptable for KLIMT.[8] It is also possible to use KLIMT as stand-alone application with a dataset, but without a tree. Trees can be loaded interactively from external files later when necessary.

[8] As KLIMT is intended for use with R, I decided to not integrate DaMaT's functionality in KLIMT, because R provides functions for import and export powerful enough.

Chapter 2

Applications

In previous chapter the various features of KLIMT have been described, but in order to show how exactly they can be used for the exploratory data analysis I want to illustrate the application of KLIMT on two real datasets. The datasets were not chosen explicitely to show the excellence of tree models or KLIMT, but selected at random.

It is hard to present the real value of an interactive software in printed form, because immediate interactivity, the main strength of exploratory methods cannot be expressed. In following sections I will include some snapshots of certain plots, but during the analysis dozens of different views, selections and plots are viewed.

For example opening a few histograms of used variables and clicking on each leaf gives a rough idea about the distribution of the cases for each variable. To do the same with static plots for a tree with ten leaves and four variables it would involve printing of 40 plots. Comparison of 40 printed plots is very hard if possible at all. Interactive methods allow you to use only four plot windows and differences can be very easily compared by alternately clicking on the leaves of interest.

To give at least some idea of the application of KLIMT, following two sections describe the analysis of two real datasets.

2.1 Classification task

The first example is the *meningitis* dataset which was briefly introduced in section **??**. The dataset described here is slightly different from the dataset

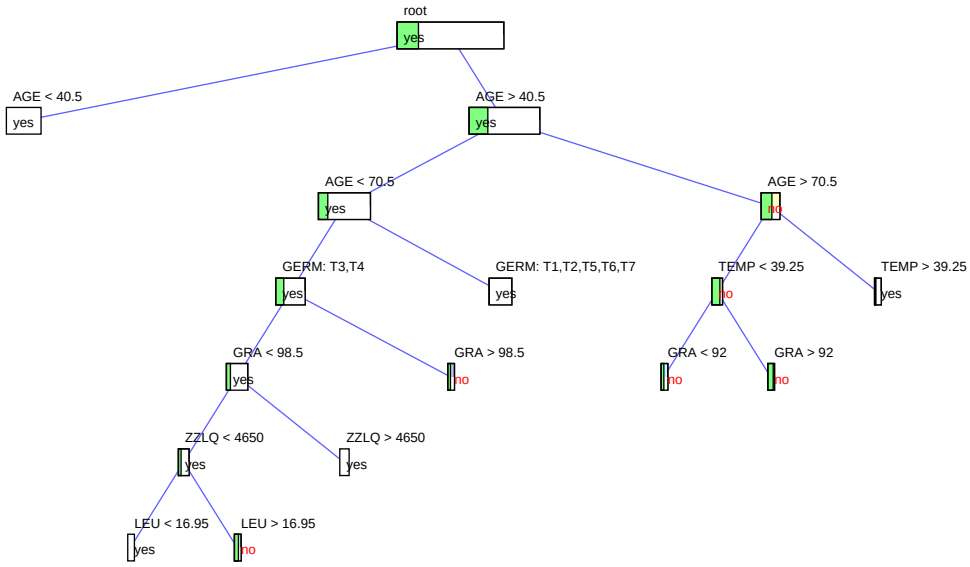

Figure 2.1: Classification tree of *meningitis* dataset using all variables and deviance splitting criterion.

used for the sample tree in section **??** because it contains more variables and more cases (total of 160). The dataset classifies patients treated for meningitis disease in two groups according to the outcome (`OUT`): `no` means that the patient either died or didn't leave the coma, `yes` classifies patients that recovered from the disease. The goal is to find a model that will predict the result of the treatment according to the variables acquired during the first inspection when the disease was detected.

The following variables were used for the building of the classifier: `AGE` and `SEX` are self-explaining, `TEMP` is the body temperature in degrees Celsius, `GERM` is the type of the germ of the meningitis disease, `MEN` denotes the class of the disease and finally `ZZLQ`,`GRA` and `LEU` are values obtained from body liquids. The last three variables as well as `TEMP` contain missing values.

Let's start with building a tree using the variables listed above. The resulting tree is shown in Fig. 2.1. In order to highlight all cases with negative outcome, select `OUT`, click on *Hist/Bar* and select the bar with the `no` label. We see that the tree splits `AGE` very early into three branches, where the leftmost node is even pure because a query displays 0 cases of `no` and 42

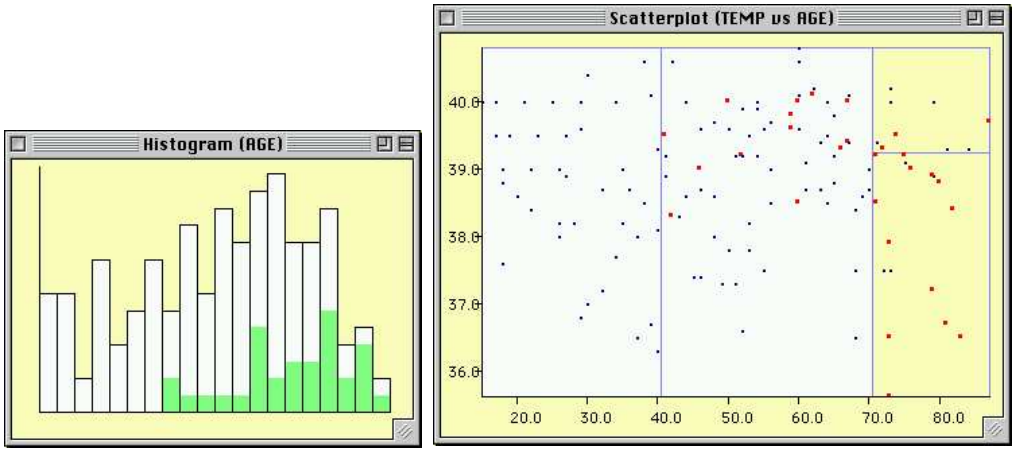

Figure 2.2: Histogram of AGE variable and scatterplot TEMP vs AGE with negative cases selected.

of `yes` class. Histogram of `AGE` in Fig. 2.2 with highlighted negative cases confirms that there were no casualties among younger patients. To compare the negative cases in both remaining nodes of the splits on `AGE` we can switch to the proportional mode. It is clearly visible that the right branch contains relatively more negative cases. A query on the node tells us that the chance of survival for patients older than 70 is roughly 50% and for patients between 40 and 70 years of age it is more than 80%.

Let's consider the right branch, that is patients older than 70 years of age. More information can be obtained from a scatterplot of `TEMP` vs `AGE`, because `TEMP` seems to improve the classification for that group. The plot confirms that old patients with lower body temperature apparently have higher mortality rate. This could be due to the fact that lower temperature means lesser activity of the immunosystem which could indicate that it cannot fight against the disease germs and therefore leading to a bad outcome. Plotting different plots of all other variables to find some explanation with cases in question selected does not yield any results. The further split on `GRA` seems very likely to overfit the data.

The analysis of the middle branch of patients of age between 41 and 70 years gives some interesting results. Patients of that age died only if they were infected by certain meningitis germs. Surprisingly this does not hold true for the older group we analyzed before, in fact they died regardless of

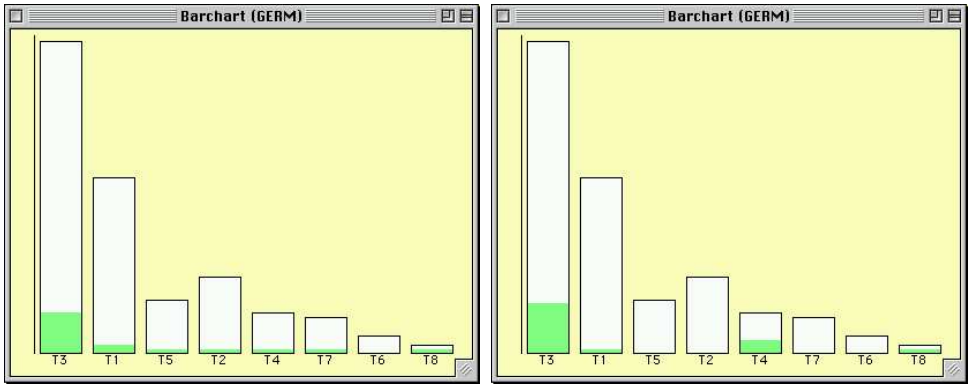

Figure 2.3: Barchart of GERM variable. In left plot shows selected patients older 70 with negative outcome in right plot patients between 41 and 70 with negative outcome.

the germ type as Fig 2.3 shows. This comparison could be done with `GERM` vs `AGE` parallel dotplots and selected `no` class as well. Further splitting uses medical properties and is fairly difficult, because all three variables: `GRA`, `ZZLQ` and `LEU` contain relatively many missing values. Even though the split on `GRA` seems to do well by separating into nodes 4/22 and 6/3 the scatterplot `GRA` vs `AGE` reveals that in practice it's likely to fail because the remaining negative values are widely spread and their further splitting is based only on 4 cases.

Still the results obtained so far are easy to interpret and even possible to explain to a physician. First deciding factor is the age, for older patients the body temperature is the second factor to watch, for younger patients it's the germ type of the disease. The corresponding survival rate estimates are directly visible from the queries of the corresponding nodes.

We have considered only one tree so far. Let's grow another tree and see if it confirms the results concluded from the first model. An option is to generate a tree based on the same dataset but different splitting rule. Fig 2.4 shows a tree grown using the Gini index of impurity without missing values.

KLIMT has the ability to open multiple trees of the same dataset. To load additional tree to KLIMT, save the tree in R in a separate file and use *Open tree..* button in the variables window. All trees are use the same linked highlighting. At first glance we can see that the trees are very similar. First split is on `AGE`, second is `GERM` for younger patients and `TEMP` for older

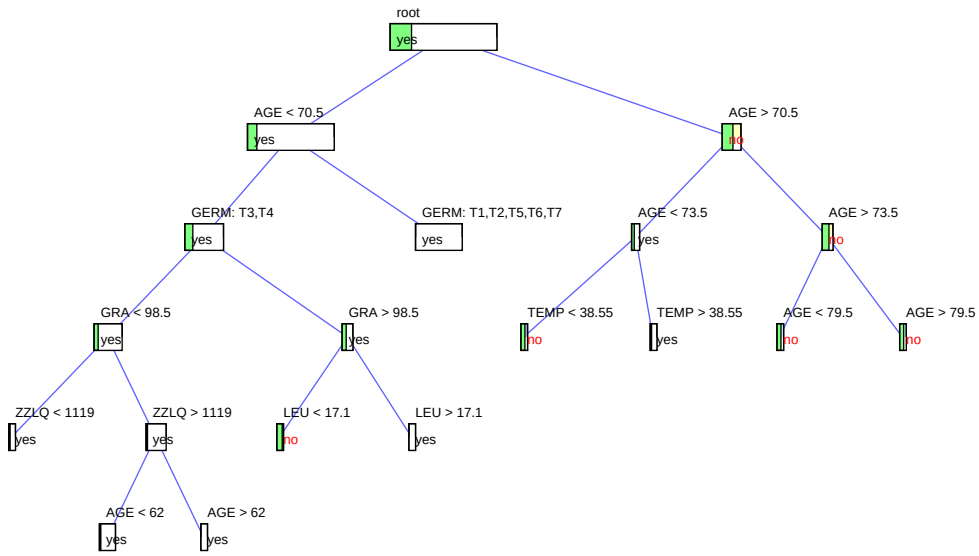

Figure 2.4: Classification tree of *meningitis* dataset using all variables and Gini index as splitting criterion.

patients. One obvious change is that the group of patients of age 40 and younger is not splitted in an own node. Selecting that node in the first tree shows how these cases are classified in the second tree. Obviously they are spread across the entire branch of the tree. Most of them fall in the category of "good" germs, but those not filtered out by the `GERM` rule are hard to catch. This is obviously a weakness of the second classification tree.

Another way of comparing both trees is to use treemaps of both plots as displayed in Fig. 2.5. First tree tends to make more clear classifications, there are only few and smaller nodes with evenly populated leaves. Also the leaves of the second tree always contaminated compared to the leaves of the first tree. It is possible to use the interactive features to see the distribution of cases in leaves of one tree in the other tree.

It would be interesting to use improved classifiers as described in section **??** and visualize the results. This could be done for example by applying colors to cases according to the number of misclassifications. Using already existing interactive features of KLIMT this would help to analyze potential problematic cases. Currently KLIMT does not support this feature, but it is on the list of tasks to be done in next versions.

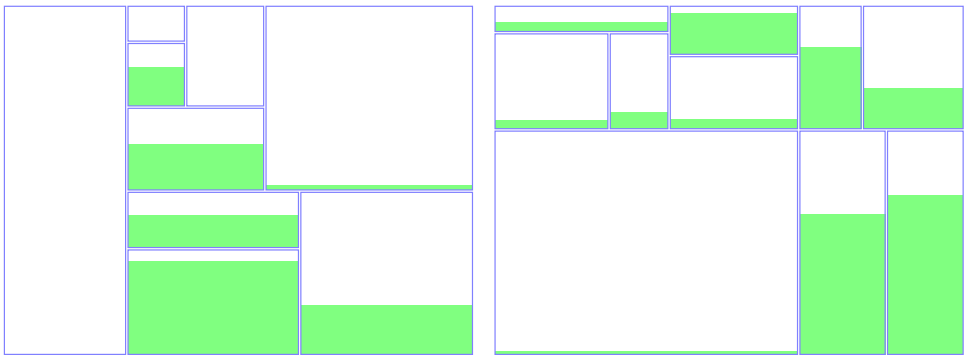

Figure 2.5: Treemaps of both trees from previous figures with negative cases selected.

2.2 Regression task

The second example in this chapter is an example of a regression tree. It is based on the *birth weight* dataset mentioned in section **??**. The variable BirthWt contains the birth weight of a newborn baby. Other variables represent the knowledge accumulated before the birth. They contain basic information about the mother (AgeMom, Race, Smokes and LastWt) as well as boolean variables denoting potential complications (Hypertension, UterineIrrit) or actions taken before the birth (FirstTriVisits, PreLabors). This dataset was used in classification context with birth weights classified in low and high, but now we will try to predict the weight of the newborn with a regression tree. The total number of cases is 189.

The regression tree obtained when using all variables to explain BirthWt is shown in Fig. 2.6. It seems that last weight of the mother plays the most important role, but it is hard to see anything more in the tree. This is also due to the fact that we are interested in the predicted values that are associated with leaves, but they are scattered all over the display. Perhaps it would be more fruitful to display all leaves on the same level. Soon it turns out that displaying all nodes on one level leads to very cluttered display with all labels mixed. Solution to that problem is the *rotate* function which can be used to display the tree in the horizontal orientation. Some information about the quality of each split could be useful, because the tree seems a way too big for that relatively small number of cases. Deviance gain in each node

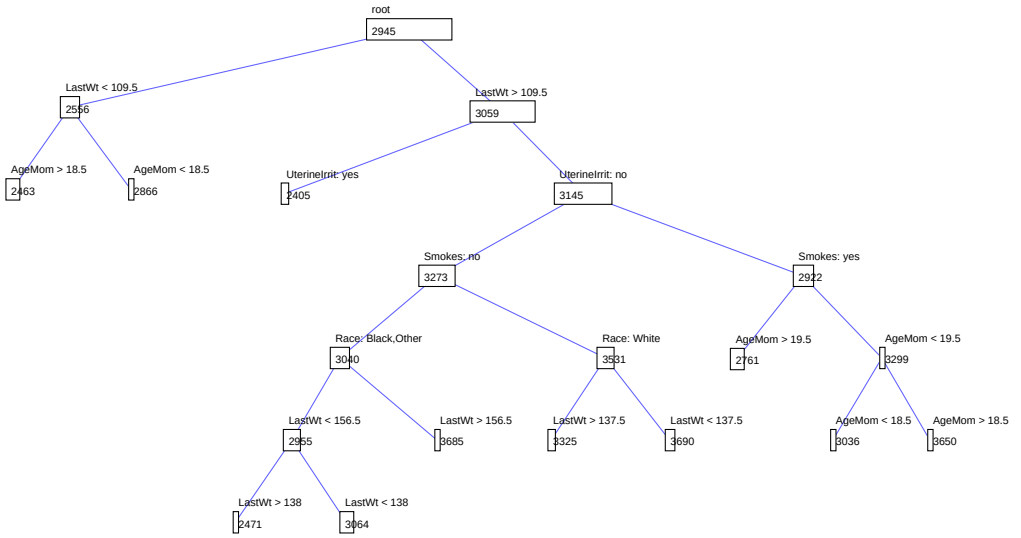

Figure 2.6: Regression tree of *low birth* dataset.

and remaining deviances in leaves can be visualized as well. The resulting tree can be seen in Fig. 2.7.

The red circles beside each inner node represent the deviation gain in the node. The size of each circle is proportional to the gain, that is the difference between the deviance of the parent node and the sum of deviances in children nodes. It is possible to see that the first split, second split on UrineIrrit and fourth on Race have the biggest gain.

The size of each rectangle beside a leaf is proportional to the remaining deviance in the leaf. In this model the remaining deviance mostly proportional to the size of the node, which is a bad sign, because our nodes don't contain tight groups of values of the response variable. Further analysis supports this conjecture. Best way for such analysis is to plot a histogram for BirthWt and highlight node by node. Such histograms for two leaves (2405 left, 3064 right) are given in Fig. 2.8. Although the difference between both response predictions is fairly high (2405 is the lowest prediction, 3064 is approximately in the middle) distributions in both nodes considerably overlap.

In order to prevent the splitting in many small nodes and thus risk overfitting, we can prune the tree. From the deviance gain visualization it seems

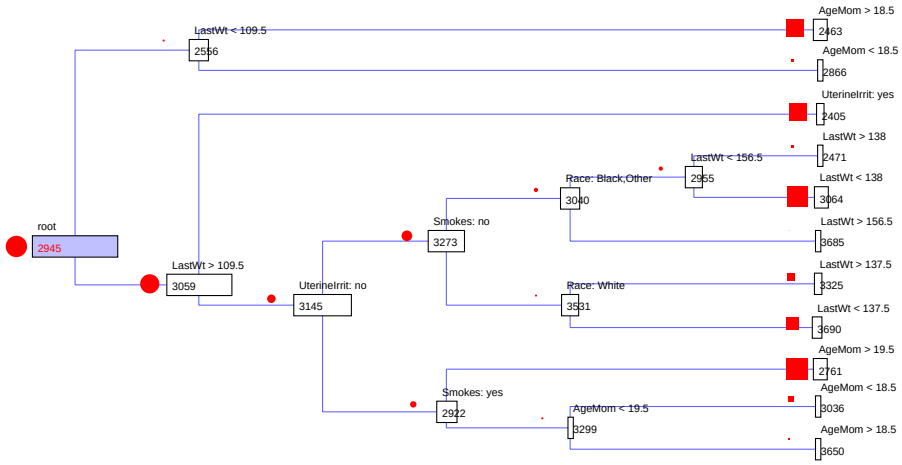

Figure 2.7: The same regression tree but rotated and with deviance gain displayed.

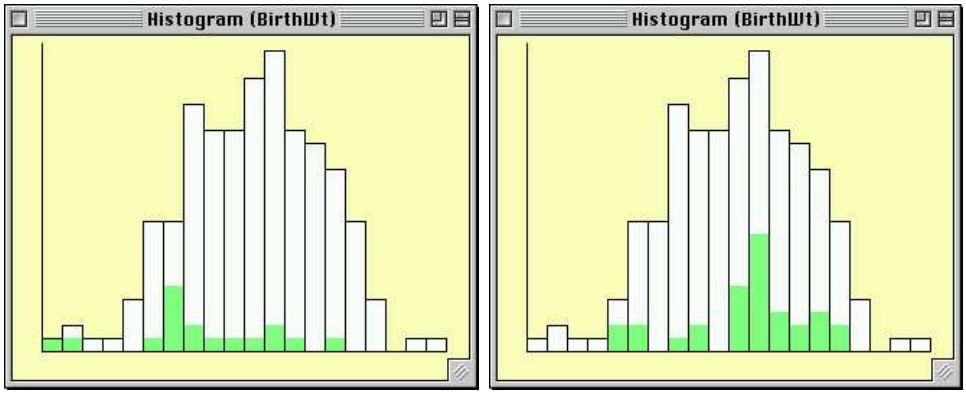

Figure 2.8: Histograms of birth weight with different leaves selected.

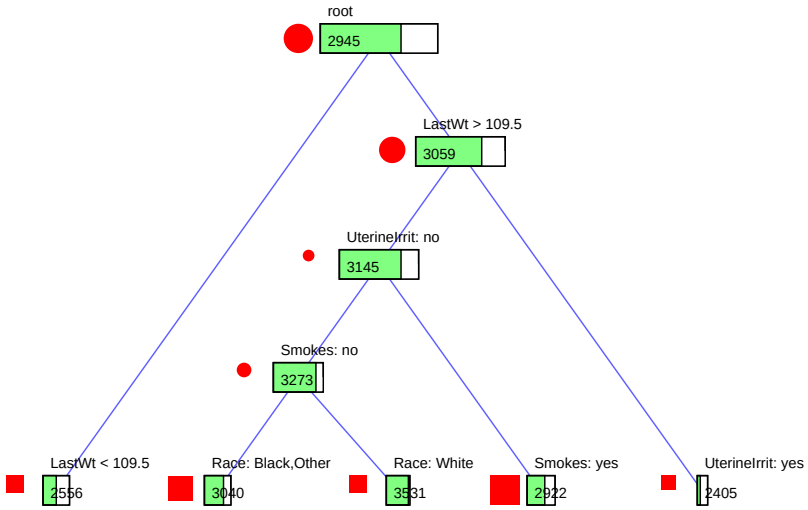

Figure 2.9: Pruned regression tree.

like pruning after the split on Smokes and removing the very poor LastWt split could be a good idea. The resulting tree is shown in Fig. 2.9.

Another approach is to use R's `prune.tree` function to obtain deviance vs size plot as displayed in Fig. 2.10. From this plot it is visible that the best tree will be of size 5. Using the results from this plot R can find the optimal tree of size 5. It is interesting that the intuitive pruning based on gain visualization produces exactly the same tree as this optimalization approach.

The resulting leaves are much better. Despite their size they give quite distinguished predictions. For example a t-test of difference of means for leaves 3040 and 3531 shows highly significant difference ($p = 0.0006$). That test can be either done directly in R or alternatively KLIMT offers the possibility to export selected cases of a variable by pressing `Shift+c` key in the corresponding histogram.

If the above problem is treated as a classification task, the results are totally different. From medical point of view children with lesser weight than 2.5kg are classified as having "low" weight and are to be observed more carefully. Such distinction leads to a classification model. The variables Race, UterrineIrrit and Smokes do not appear in the classification tree at all. Instead PreLabors is used in the root split, followed by LastWt and AgeMom. The corresponding tree is given in Fig. 2.11.

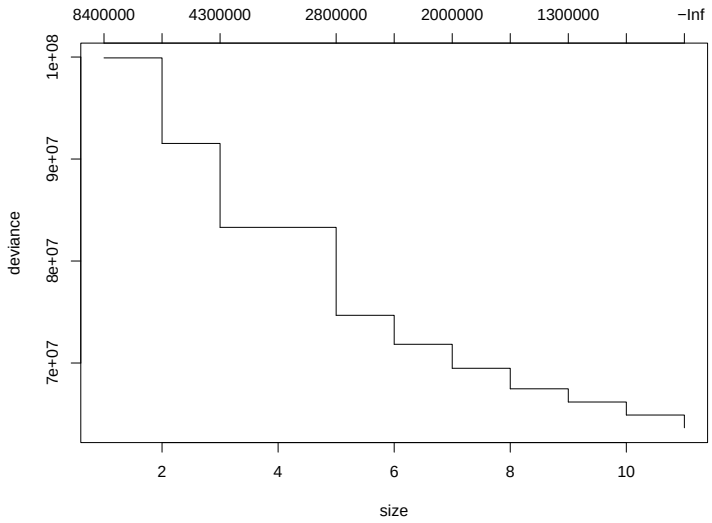

Figure 2.10: Deviance vs size plot for a sequence of pruned trees.

This is not that far from the results obtained when logistic regression is used. PreLabors used for the first split turns out to be only highly significant factor in that model, followed by the significant factors Hypertension and LastWt. Hypertension is not used in the tree model, because there are only 12 cases of hypertension in the dataset and they are distributed 7:5 in low:high thus its contribution in a tree model would be very small.

Obviously the race difference which played an important role in the regression tree (significant difference) is not significant in the classification model. This is mainly because both groups had a mean far higher than 2500, now fall into the same class and therefore are of no interest in the new model.

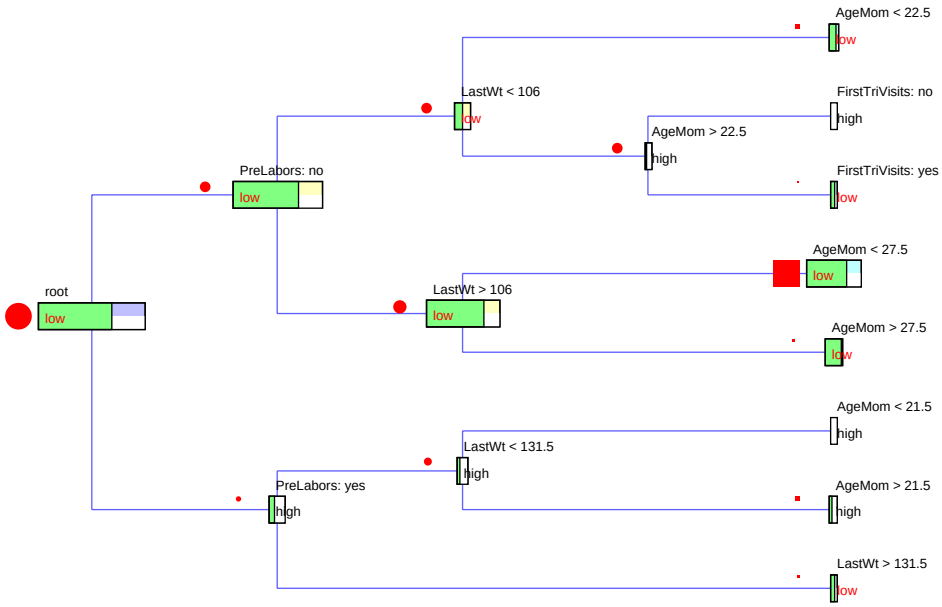

Figure 2.11: Classification tree for the *low birth* dataset. Cases with birth weight $<$ 2,500g are selected.

Appendix A

Shortcuts and Technical Information

A.1 Shortcuts

Tool modes in a tree window:

<e>	sElect cases
<z>	Zoom
<v>	moVe (or hold <space> key for temporary pan mode)
<n>	Node select

Other commands in a tree window:

<c>	toggle type of Connecting lines
<d>	toggle Deviance display
<f>	toggle Final node alignment
<h>	Help
<l>	toggle Labels
<m>	tree Map
<shift><n>	new (pruned) tree
<o>	Open file
<p>	Prune
<q>	Quit
<r>	Re-arrange nodes
<shift><r>	Rotate

<s>	toggle node Size (fixed/proportional)
<+>/<->	change deviance zoom

Commands for a treemap:

<a>	Alternate splits, i.e. toggle between treemap and spline plot view
<r>	Rotate, i.e. change the orientation of the first split

Commands for all plots:

<shift><c>	write all currently selected cases to a file
<shift><p>	Print current plot - create a *PostScript* output file
<shift><x>	eXport current graphics to a PoGraSS metafile

A.2 Classes

The list of main public Java classes is given here along with a short description of each class. These classes can be used in the R interface. More detailed description of all attributes and methods can be obtained from the *JavaDoc* documentation of KLIMT.

Axis
: Handles conversion between data space and the graphical units for displays. It is used by various plots.

BarCanvas
: Implementation of a barchart

Commander
: Interface used by most windows to pass messages, such as user commands.

Common
: This class contains static attributes and methods that can by used by any class. Among others the current version and release numbers are stored here.

DBCanvas
: Extension of `Canvas` to support double-buffering.

Dependent
Interface for classes that want to be notified.

HelpFrame
Help frame for a tree window.

HistCanvas
Implementation of a histogram.

Klimt
Wrapper for KLIMT's functionality. One instance can handle one dataset and several trees. This class is not used in stand-alone KLIMT, but is the basis for the $\hat{\Omega}$ interface between R and KLIMT.

MosaicCanvas
Implementation of treemaps and spineplots of leaves.

MsgDialog
General purposse message dialog with an "OK" button.

Node
Implementation of a general tree structure.

Notifier
A class that provides notification functionality.

PoGraSS
Abstract class defining the PoGraSS interface.

PoGraSSPS, PoGraSSgraphics, PoGraSSmeta
Various implementations of the PoGraSS interface.

SMarker
A class providing the linked highlighting functionality for a dataset.

SNode
Node with statistical information, such as splitting rule etc.

SVar
Storage of a variable.

SVarSet
This class builds a dataset, uses `SVar` for each variable.

ScatterCanvas
Implementation of a scatterplot.

TFrame
Extended `Frame` as used by most windows in KLIMT.

TInfoCanvas
Display of a query result.

TreeCanvas
Implementation of the tree visualization and handling.

VarCanvas
Canvas of the Variables window.

A.3 R-Interface

In order to use the bi-directional R-interface, you must have SJava from the $\hat{\Omega}$ project installed and properly configured, including the corresponding Java runtime environment. The entire code is very experimental and under development so this small section is here for those adventurous to give you a rough idea about the concept. The following example illustrates the use of the interface. It is an example on the lowest level, later often used steps can be grouped to functions that make the work easier for the user. A detailed description of the Java interface can be obtained from the $\hat{\Omega}$ homepage.

```
> library(SJava)
> .JavaInit(list(classPath=c("KLIMT.jar")));
> k<-.JavaConstructor("Klimt","myData")
> t1<-.Java(k,"getTree",0:0)
> .Java(t1,"toString")
[1] "SNode[cond=\"root\",cases=189,F1=9.992E7,Name=\"2945\"]"
> t2<-.Java(k,"addTree","secondTree");
> src<-.Java(t1,"getSource")
> m<-.Java(src,"getMarker")
> for (i in 1:100) .Java(m,"set",i,1:1)
> .Java(m,"NotifyAll")
```

First two lines load the SJava library and initialize the Java interface. `KLIMT.jar` is given as class-path, because we need to use classes of KLIMT. This call assumes that KLIMT's Java-archive is located in the current directory.

In the next step a KLIMT-object is created using `myData` file as source. This equals to calling KLIMT directly from R as described for the file interface, but the main difference is that we obtain an object of the class `Klimt` in R, which is the core for further interactivity between both systems.

To get the first currently loaded tree the `getTree` method of the class `Klimt` can be used. This is done in line 4. What we get is an object of the class `SNode`, i.e. the root node of the tree. Basic information about the node can be obtained using the `toString` method. The call and corresponding output are shown in lines 5 and 6.

An additional tree can be loaded into KLIMT at any time, using the `addTree` method. The tree is displayed immediately in a new window and the currently used linked highlighting is attached the the new tree.

Finally let's demonstrate how R can directly manipulate data objects in KLIMT. We want to select first 100 cases of the dataset associated with the tree t. The `getSource` method delivers the required data source and `getMarker` the corresponding marker that is used to highlight cases. A regular loop in R is used to select all cases with index 1 to 100. Finally we want to notify all plots about the change by calling the `NotifyAll` function. This method can be used to create complex animations. Most parameters of all plots can be changed from within R.

In order to make this interface available to wide variety of users, is it necessary to provide wrapper functions that will hide the underlying $\hat{\Omega}$ commands from the user. Using such wrapper functions, the above example looks as follows.

```
source("klimtOH.r")
k <- Klimt(myTree,myData);
t1 <- Klimt.tree(k,0);
t2 <- Klimt.addTree(k,newTree);
for (i in 1:100) Klimt.setMarkAt(k,i)
Klimt.refresh;
```

The file `klimtOH.r` contains the definitions of the wrapper functions. It also automatically loads the necessary SJava library and initializes the Java

subsystem. All commands are rather self-explaining and correspond to their low-level counterparts.

For latest development check out KLIMT-project homepage:
`http://www.klimt-project.com/`