

Janez Demšar

Data Mining

**Material for lectures on Data mining
at Kyoto University, Dept. of Health Informatics
in July 2010**

Contents

INTRODUCTION	4
(FREE) DATA MINING SOFTWARE	5
BASIC DATA MANIPULATION AND PREPARATION	6
SOME TERMINOLOGY	6
DATA PREPARATION	6
FORMATTING THE DATA	7
BASIC DATA INTEGRITY CHECKS	7
MAKING THE DATA USER-FRIENDLY	8
KEEPING SOME THE DATA FOR LATER TESTING	8
MISSING DATA	9
DISCRETE VERSUS NUMERIC DATA	10
FROM DISCRETE TO CONTINUOUS	10
FROM CONTINUOUS TO DISCRETE	11
VARIABLE SELECTION	13
FEATURE CONSTRUCTION AND DIMENSIONALITY REDUCTION	15
SIMPLIFYING VARIABLES	15
CONSTRUCTION OF DERIVED VARIABLES	15
VISUALIZATION	17
HISTOGRAMS	17
SCATTER PLOT	18
PARALLEL COORDINATES	24
MOSAIC PLOT	25
SIEVE DIAGRAM	28
"INTELLIGENT" VISUALIZATIONS	30
DATA MODELING (LEARNING, CLUSTERING...)	31
PREDICTIVE MODELING / LEARNING	32
CLASSIFICATION TREES	32
CLASSIFICATION RULES	34
NAÏVE BAYESIAN CLASSIFIER AND LOGISTIC REGRESSION	36

OTHER METHODS	38
UNSUPERVISED MODELING	38
DISTANCES	38
HIERARCHICAL CLUSTERING	39
K-MEANS CLUSTERING	42
MULTIDIMENSIONAL SCALING	42
VISUALIZATION OF NETWORKS	44
<u>EVALUATION OF MODEL PERFORMANCE</u>	45
SAMPLING THE DATA FOR TESTING	45
MEASURES OF MODEL PERFORMANCE	46
SCALAR MEASURES OF QUALITY	46
RECEIVER-OPERATING CHARACTERISTIC (ROC) CURVE	51
CLASSIFICATION COST	53

Introduction

This material contains the "handouts" given to students for data mining lecture held at the Department of Health Informatics at the University of Kyoto. The informality of the "course", which took five two-hour lectures is reflected in informality of this text, too, as it even includes references to discussions at past lectures and so on.

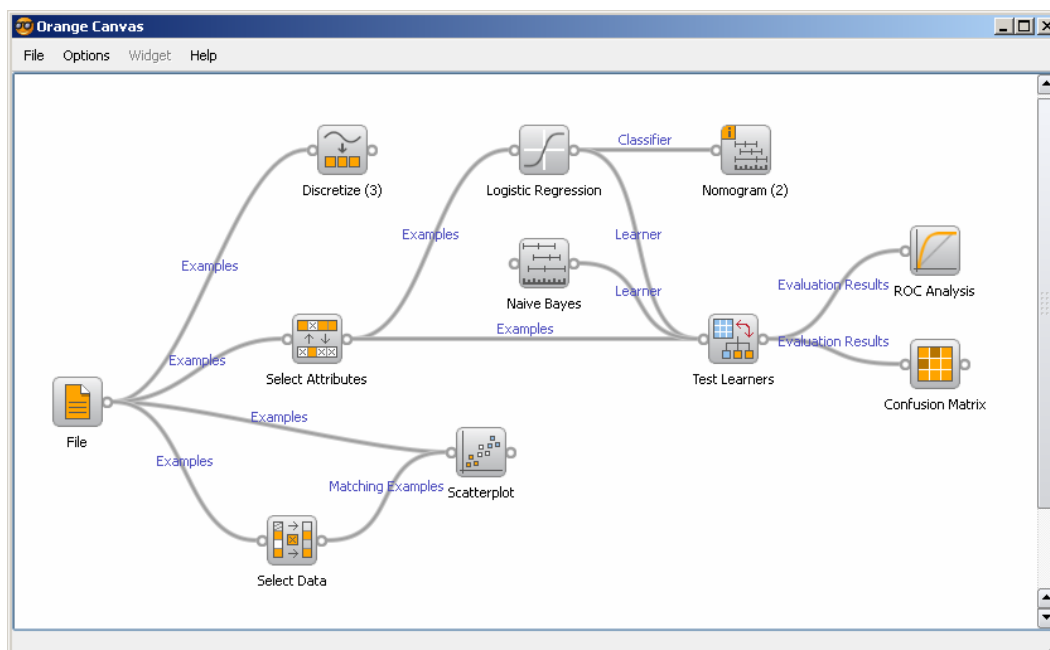
Some more material is available at <http://www.ailab.si/janez/kyoto>.

(Free) Data mining software

Here is a very small selection of free data mining software. Google for more.

- **Orange** (<http://www.ailab.si>) is a free data mining software we are going to use for these lectures.
- **Knime** (<http://www.knime.org/>) is excellent, well designed and nice looking. It has fewer visualization methods than Orange and it's a bit more difficult to work with, but it won't cost you anything to install it and try it out.
- **Weka** (<http://www.cs.waikato.ac.nz/ml/weka/>) is oriented more towards machine learning. While very strong in terms of different kinds of prediction models it can build, it has practically no visualization and it's difficult to use. I don't really recommend it.
- **Tanagra** (<http://eric.univ-lyon2.fr/~ricco/tanagra/>) is smaller package written by a one-man band, but has much more statistics than Orange.

Orange has a different user interface from what you are used to (similar to Knime and, say, SAS Miner and SPSS Modeler). Its central part is **Orange Canvas** onto which we put **widgets**. Each widget has a certain function (loading data, filtering, fitting a certain model, showing some visualization...), and receives and/or gives data from/to other widgets. We say that widgets send **signals** to each other. We set up the data flow by arranging the widgets into schema and **connecting them**. Each widget is represented with an icon which can have "ears" on the left (**input**) and right (**outputs**). Two widgets can only be connected if their signal types match: a widget outputting a data table cannot be connected to a widget which expects something completely different. We will learn more about Orange as we go.



Basic data manipulation and preparation

Some terminology

Throughout the lectures you will have to listen to an unfortunate mixture of terminology from statistics, machine learning and elsewhere. Here is a short dictionary.

Our **data sample** will consist of **data instances** e.g. patient records. In machine learning we prefer to say we have a **data set** and it consists of **examples**.

Each example is described by **features** or **values of variables** (statistics) or **attributes** (machine learning). These features can be **numerical (continuous)** or **discrete (symbolic, ordinal or nominal)**; the latter two terms actually refer to two different things: ordinal values can be ordered, e.g. small, medium, big, while nominal have no specific order, just names, e.g. red, green, blue).

Often there is a certain variable we want to predict. Statisticians call it **dependent variable**, in medicine you will hear the term **outcome** and in machine learning will call it **class** or **class attribute**. Such data is called **classified data** (no relation to CIA or KGB).

Machine learning considers examples to belong to two or more classes, and the task is then to determine the unknown class of new examples (e.g. diagnoses for newly arrived patients). We say that we classify new examples, the method for doing this is called a **classifier**, the problem of finding a good classifier is called **classification problem** ... and so on. The process of constructing the classifier is called **classifier induction** from data or also **learning from data**. The part of data used for learning is often called **learning data** or **training data** (as opposed to **testing data**, which we used for testing).

The "class" can also be continuous. If so, this we shouldn't call it a class (although we often do). The problem is then not a classification problem any more; we call it a **regression problem**.

In statistic terminology, the task is to **predict**, and the statistical term for a classifier is **predictive model**. We do not learn here, we **fit the model to the data**. We sometimes also use the term **modeling**. If the data is classified, the problem is called **supervised modeling**, otherwise it's **unsupervised**.

Since data mining is based on both fields, we will mix the terminology all the time. Get used to it. 😊

Data preparation

This is related to Orange, but similar things also have to be done when using any other data mining software. Moreover, especially the procedures we will talk about in the beginning are also applicable to any means of handling the data, including statistics. They are so obvious, that we typically forget about them and scratch our heads later on, when things begin looking weird.

If all this will seem to you as common sense, this is simply because most of data mining is just systematic application of pure common sense.

Formatting the data

Orange does not handle Excel data and SQL databases (if you don't know what is SQL, don't bother) well yet, so we feed it the data in textual tab-delimited form. The simplest way to prepare the data is to use Excel. Put names of variables into the first row, followed by data instances in each row. Save as tab-delimited, with extension .txt.

This format is somewhat limited with respect to what can you tell Orange about variables. A bit more complicated (but still simple format uses the second and the third row to define the variable types and properties). You can read more about it at <http://www.aillab.si/orange/doc/reference/tabdelimited.htm>.

You can see several examples in c:\Python26\lib\site-packages\orange\doc\datasets, assuming that you use Windows and Python 2.6.

For technical reasons, Orange cannot handle neither values, not variables names and not even file names which include non-ASCII (e.g. kanji) characters. There is little we can do to fix Orange at the moment.

Basic data integrity checks

Check data for errors and inconsistencies. Check for spelling errors. If somebody mistypes *male* as *mole*, we are suddenly going to have three sexes in your data, *females*, *males* and *moles*.

Orange (and many other tools) is case-sensitive: you should not write the same values sometimes with small and sometimes with capital letters. That is, *female* and *Female* are two different species.

Check for any other obviously wrong data, such as mistyped numbers (there are not many 18 kg patients around – did they mean 81?!)

How are unknown values denoted? There are various "conventions": question marks, periods, blank fields, NA, -1, 9, 99... Value 9 seems to be especially popular for denoting unknowns in symbolic values encoded numerically, like gender, e.g. 0 for female, 1 for male and 9 for unknown. If 9's are kept in the data and the software is not specifically told that 9 means an unknown value, it will either think that there are three or even ten sexes (as if life was not complicated enough with two). Or even that sex is a numerical variable ranging from 0 to 9.

You will often encounter data in which unknowns are marked differently for different variables or even for one and the same variable.

Does the same data always mean the same thing? And opposite: can the same thing be encoded by more than one different value? Check for errors in data recording protocol, especially if the data was collected by different people or institutions. In the data I'm working on right now, it would seem that some hospitals have majority of patients classified as Simple trauma, while others have most patients

with Other_Ent. Could this be one and the same thing, but recorded differently in different hospitals due to lax specifications?

A typical classical example – which I happily also encountered while in Nara – is that some institutions would record the patients' height in meters and others in centimeters, giving an interesting bimodal distribution (besides rendering the variable useless for further analysis).

Most such problems can be discovered by observing distributions of variable values.

Such discrepancies should be found and solved in the beginning. If you were given the data by somebody else, you should ask him to examine these problems.

→ **Hands on**

Load the data into Orange and see whether it looks OK. Try to find any mistakes and fix them (use Excel or a similar program for that).



File



Data Table



Attribute Statistics



Distributions

Making the data user-friendly

Data is often encoded by numbers. 0 for female, 1 for male (*wait, or was it the opposite?* you will ask yourself all the time while working with such data). I just got data in which all nominal values had their descriptive names, except gender for some reason. Consider renaming these values, it will make your life more pleasant.

This of course depends upon the software and the methods which you are going to use. We will encounter a simple case quite soon in which we actually want the gender to be encoded by 0 and 1.

Keeping some the data for later testing

An important problem, which was also pointed out by Prof. Nakayama at our first lecture was how to combine the statistical hypothesis testing with data mining. The only correct way I know of – the only one with which you should have no problem thrusting your results and convince the reviewers of your paper of their validity, too – is to use a separate data set for testing your hypotheses and models. Since you typically get all the data which is available in the beginning, you should sacrifice a part of your data for testing. As a rule of a thumb, you would randomly select 30% of data instances, put them into a separate file and leave it alone till just before the end.

If your data has dates (e.g. admittance date), it may make a convincing case if instead of randomly selecting 30%, you take out the last 30% of the data. This would then look like a simulated follow-up study. If you decide for this, you only have to be careful that your data is unbiased. For instance, you are researching a conditions related to diabetes and the last 30% of data was collected in the summer, using

this data as a test set may not be a good idea (since diabetic patients have more problems in summer, as far as I know).

You should split the data at this point. Fixing the problems in the data, which we discussed above, before splitting it, is a good idea, otherwise you will have to repeat the cleaning on the test data separately. But from this point on, we shall do things with the data which already affect your model, so the test data should not be involved in these procedures.

→ **Hands on**

There is a general-purpose widget in Orange to select random samples from the data. You can put the data from this widget into the one for saving data. Try it and prepare a separate data file for testing.



Data Sampler



Save

Missing data

This topic again also concerns statistics, although typical statistical courses show little interest in it. Some data will often be missing. What to do about that?

First, there are different kinds of missing data.

- It can be missing because we don't care about it, since it is irrelevant and was not collected for a particular patient. (*I didn't measure his pulse rate since he seemed perfectly OK.*)
- It can be inapplicable. (*No, my son is not pregnant. Yes, doctor, I'm certain about it.*)
- It may be just unknown.

Especially in the latter case, it may be interesting to know: why? Is its *unknowingness* related to the case? Does it tell us something?

Maybe the data was not collected since its value was known (e.g. pulse rate was not recorded when it was normal). Maybe the missing values can be concluded from the other attributes (blood loss was not recorded when there were no wounds – so we know it is zero 0) or inapplicable (the reason of death is missing, but another attribute tells that the patient survived).

For a good example: in a data set which I once worked with, a number of patients had a missing value for urine color. When I asked the person who gave me the data about the reason, he told me that they were unconscious at the time they were picked up by the ER team.

Knowing why it is missing can help dealing with it. Here are a few things we can do:

- Ask the owner of the data or a domain expert if he can provide either per-instance or general values to **impute** (the technical term for inserting the values for unknowns). If the pulse was not recorded when it was normal, well, we know what the missing value should be.

- Sometimes you may want to replace the missing value with a special value or even a specific variable telling whether the missing value is known or not. For the urine color example, if the data does not contain the variable telling us whether the patient is conscious, we can add the variable Conscious and determine its value based on whether the urine color is known or not. If it is known, the patient is conscious, if not, it is "maybe conscious" (or even "unconscious" if we can know that the data was collected so that the color is known for all conscious patients). Not knowing a value can thus be quite informative.
- Sometimes you can use a model to impute the unknown value. For instance, if the data is on children below 5, and some data on their weights is missing but you know their ages, you can impute the average body mass for the children's age. You can even use sophisticated models for that (that is, instead of predicting what you have to predict for this data, e.g. the diagnosis, you predict values of features based on values of other features).
- Impute constant values, such as "the worst" or "the best" possible, or the most common value (for discrete attributes) or the average/median (for the continuous).
- Remove all instances with missing values. This is the safest route, except that it reduces your data set. You may not like this one if your sample is already small and you have too many missing values.
- Remove the variables with missing values. You won't like this one if the variable is important, obviously. You may combine this with the previous: remove the variables with most missing values and then remove the data instances which are missing values of any of other attributes. This may still be painful, though.
- Keep everything as it is. Most methods, especially in machine learning, can handle missing values without any problem. Be aware, though, that this often implies some kind of implicit imputation, guessing. If you know something about these missing values, it is better to use one of the first solutions described.

→ **Hands on**

There is an imputation widget in Orange. See what it offers.



Impute

Discrete versus numeric data

Most statistical methods require numeric data, and many machine learning methods prefer discrete data. How do we convert back and forth?

From discrete to continuous

This task is really simple for variables which can only take two values, like gender (if we don't have the gender 9). Here you can simply change these values to 0 and 1. So if 0 is female and 1 is male, the variable represents the "masculinity" of the patient. Consider renaming the variable to "male" so you

won't have to guess the meaning of 0 and 1 all the time. Statisticians call such variables **dummy variables, dummies, indicator variables, or indicators**.

If the variable is actually ordered (small, medium, big) you may consider encoding it as 0, 1, 2. Be careful, though: some methods will assume the linearity. Is the difference between small and big really two times that between small and medium?

With general multi-valued variables, there are two options. When your patients can be Asian, Caucasian or Black, you can replace the variable Race with three variables: Asian, Caucasian and Black. For each patient, only one would be 1. (In statistics, if you do, for instance, linear regression on this data, you need to remove the constant term!)

The other option occurs in cases where you can designate one value as "normal". If urine color can be normal, red or green, you would replace it with two attributes, red and green. At most one of them can be 1; if both are zero, that would mean that the color is normal.

→ **Hands on**

There is an imputation widget in Orange. See what it offers.



Continue

From continuous to discrete

This procedure is called **categorization** in statistics, since it splits the data into "categories". For instance, the patient can be split into categories young, middle-age and old. In machine learning this is called **discretization**.

Discretization can be unsupervised or supervised (there go these ugly terms again!). Unsupervised discretization is based on values of variables alone. Most typically we split the data into **quartiles**, that is, we split the data into four approximately equal groups based on the value of the attribute. We could thus have the youngest quarter of the patients, the second youngest, the older and the oldest quarter. This procedure is simple and usually works best. Instead of four we can also use other number of intervals, although we seldom use more than 5 or less than 3 (unless we have a very good reason for doing so).

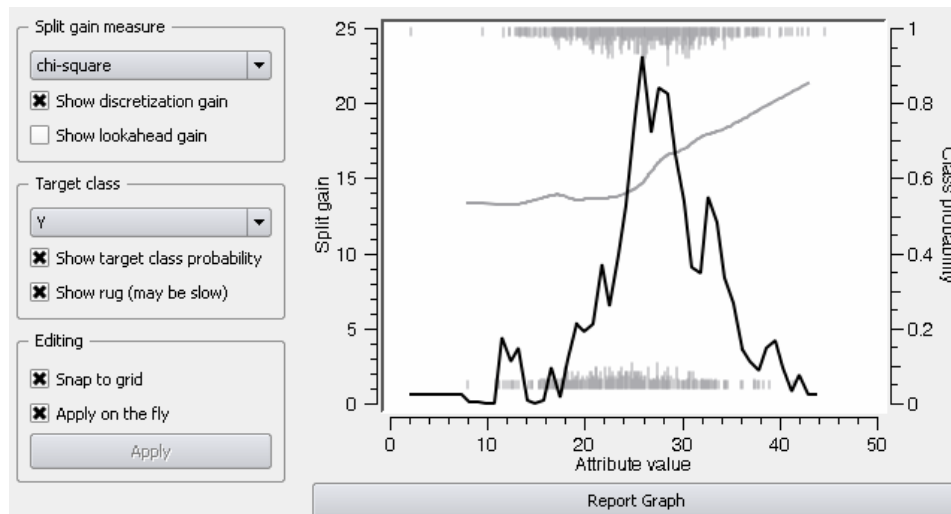
Supervised methods set the **boundaries** (also called **thresholds**) between intervals so that the data is split into groups which differ in class membership. Put simply, say that a certain disease tends to be more common for people older than 70. 70 is thus the natural threshold which splits the patient into two groups, and it does not really matter whether it splits the sample of patients in half or where the quartiles lie.

The supervised methods are based on optimizing a function which defines the **quality of the split**. One can, for instance, compute the **chi-square statistics** to compare the distribution of outcomes (class distributions, as we would say in machine learning) for those above and below a certain threshold.

Different thresholds yield different groups and thus different chi-squares. We can then simply choose the threshold with the highest chi-square.

A popular approach in machine learning is to compute the so called **information gain** of the split, which measures how much information about the instance's class (e.g. disease outcome) do we get from knowing whether the patient is above or below a certain threshold. While related to chi-square and similar measures, one nice thing about this approach is that it can evaluate whether setting the threshold gives more information than it requires. In other words, it includes a criterion which tells us whether to split the interval or not.

With that, we can run the method in an automatic mode which splits the values of the continuous variable into ever smaller sub-intervals until it decides it's enough. Quite often it will not split the variable at all, hence proposing to use "a single interval". This implicitly tells us that the variable is useless (that is, the variable tells us nothing, it has an information gain of 0 for any split) and we can discard it.



You can combine supervised methods with manual fitting of thresholds. The figure above shows a part of Orange's Discretize widget. The graph shows the chi-square statistics (y axis) for the attribute we would get if we split the interval of values of continuous attribute in two at the given threshold (x axis). The gray curve shows the probability of class Y (right-hand y axis) at various values of the attribute. Histograms at the bottom and the top show distribution of attribute's values for both classes.

Finally, there may exist a standard categorization for the feature, so you can forget about the unsupervised and supervised, and use the expert provided intervals instead. For instance, by some Japanese definition BMI between 18.5 and 22.9 is normal, 23.0 to 24.9 is overweight and 25.0 and above is obese. These numbers were decided with some reason, so you can just use them unless you have a particular reason for defining other thresholds.

While discretization would seem to decrease the precision of measurements, the discretized data is usually just as good as the original numerical data. It may even increase the accuracy when the data has

some strong outliers, that is, measurements out of scale. By splitting the data into four intervals, the outliers are simply put into the last interval and do not affect the data as strongly as before. (For those more versed in statistics: this has a similar effect as using non-parametric test instead of a parametric one, for instance Wilcoxon's test instead of t-test. Non-parametric

→ **Hands on**

Load the heart disease data from documentation data sets and observe how the Discretize widget discretizes the data. In particular, observe the Age attribute. Select only women and see if you can find anything interesting.



Discretize Select data

Variable selection

We often need to select a subset of variables and of data instances. Using fewer variables can often result in better predictive models in terms of accuracy and stability.

- Decide which attributes are really important. If you take too many, you are going to have a mess. If you remove too many, you may miss something important.
- As we already mentioned, you can remove attributes with too many missing values. These will be problematic for many data mining procedures, and are not very useful anyway, since they don't give you much data. Also, if they weren't measured when your data was collected (to expensive? difficult? invasive or painful?), think whether those who use your model will be prepared to measure it.
- If you have more attributes with a similar meaning, it is often beneficial to keep only one – unless they are measured separately and there is a lot of noise, so you can use multiple attributes to average out the noise. (You can decide later which one you are keeping.)
- If you are building a predictive model, verify which attributes will be available when the prediction needs to be made. Remove the attributes that are only measured later or those whose values depend upon the prediction (for instance, drugs that are administered depending upon the prediction for a cancer recurrence).

Data mining software can help you in making these decisions.

First, there are simple univariate measures of attribute quality. With two-class problems, you can measure the attribute significance by t-test for continuous and chi-square for discrete attributes. For multi-class problems, you have ANOVA for continuous and chi-square (again) for discrete attributes. In regression problems you probably want to check the univariate regression coefficients (I honestly confess that I haven't done so yet).

Second, you may observe the relations between attributes and relations between attributes and the outcome graphically. Sometimes relations will just pop up, so you want to keep the attribute, and sometimes it will be obvious that there are none, so you want to discard them.

Finally, if you are constructing a supervised model, the features can be chosen during the model induction. This is called **model-based variable selection** or, in machine learning, a **wrapper approach to feature subset selection**. In any case, there are three basic methods. **Backward feature selection** starts with all variables, constructs the model and then checks, one by one, which variable it can exclude without harming (or with the least harm) to predictive accuracy (or another measure of quality) of the model. It removes that variable and repeats the process until any further removing would drastically decrease the model's quality. **Forward feature selection** starts with no variables and adds one at a time – always the one from which the model profits most. It stops when we gain nothing more by further additions. Finally, in **stepwise selection**, one feature can be added or removed at each step.

	Attribute	#	ReliefF	Inf. gain	Gini	log OR (for 'Y')
1	hr_rest	C	0.023	0.007	0.002	NA
2	hr_stress	C	0.017	0.008	0.003	NA
3	age	C	0.011	0.012	0.004	NA
4	sbp_rest	C	0.007	0.005	0.002	NA
5	gender	2	0.000	0.021	0.007	-0.730 (M)
6	smoke	2	0.000	0.003	0.001	-0.288 (Y)
7	bmi	C	-0.014	0.017	0.006	NA

The figure shows a part of Orange's widget for ranking variables according to various measures (we won't go into details about what they are, but we can if you are interested – just ask!).

Note that the general problem with univariate measure (that is, those which only consider a single attribute at a time) is that they cannot assess (or have difficulties with assessing) the importance of attributes which are important in context of other attributes. If a certain disease is common among young males and old females, then neither gender nor age by itself tell us anything, so both variables would seem worthless. Considered together, they tell us a lot.

→ **Hands on**

There are two widgets for selecting a subset of attributes. One is concentrated on estimating the attribute quality (Rank), with the other we can remove/add attributes and change their roles (class attribute, meta attributes...) Especially the latter is very useful when exploring the data, for instance to test different subsets of attributes when building prediction models.

Rank	Select attributes

Feature construction and dimensionality reduction

Simplifying variables

You can often achieve better results by merging several values of a discrete variable into a single one. Sometimes you may also want to simplify the problem by merging the class values. For instance, in non-alcoholic fatty-liver disease which I'm studying right now, the liver condition is classified into one of four brunt stages, yet most (or even all) papers on this topic which construct predictive models converted the problem into binary one, for instance whether the stage is 0, 1 or 2 (Outcome=0), or 3 or 4 (Outcome=1). Reformulated like this, the problem is easier to solve. Besides that, many statistical and machine learning methods only work with binary and not multi-valued outcomes.

There are no widgets for this in Orange. You have to do it in, say, Excel. Orange can help you in deciding which values are "compatible" in a sense. We will learn about this the next time, as we explore visualizations.

Construction of derived variables

Sometimes it is useful to combine several features into one, e.g. presence of neurological, pulmonary and cardiovascular problems into a general "problems". There can be various reasons for doing this, the simplest being a small proportion of patients which each single disease. Some features need to be combined because of interactions, as the case described above: while age and gender considered together may give all the information we need, most statistical and machine learning methods just cannot use them since they consider (and discard) them one at a time. We need to combine the two variables into a new one telling us whether the patient belongs into one of the increased risk groups.

Finally, we should mention – but really only mention – the so-called **curse of dimensionality**. In many areas of science, particularly in genetics, we have more variables than instances. For instance, we measure expressions of 30,000 genes on 100 human cancer tissues. A rule of a thumb (but nothing more than a rule of a thumb) in statistics is that the sample size must be at least 30 times the number of variables, so we certainly cannot run, say linear or logistic regression (neither any machine learning algorithm) on this data.

There are no easy solutions to this problem. For instance, one may apply the *principal component analysis* (or *supervised principal component analysis* or similar procedures), a statistical procedure which can construct a small set of, say, 10 variables computed from the entire set of variables in such a way that these 10 variables carry as much (or all) data from the original variables. The problem is that principal component analysis itself would also require, well, 30 times more data instance than variables (we can bend this rule, but not to the point when the number of variables exceeds the sample size, at least not by that much).

OK, if we cannot keep the data from all variables, can we just discard the majority of variables, since we know that they are not important? Out of 30,000 genes, only a couple, a dozen maybe, is related to the disease we study. We could do this, but how do we determine which genes are related to the disease?

T-tests or information gain or similar for each gene separately wouldn't work: with so many genes and such a small sample, many genes will be randomly related to what we are trying to predict. If we are unlucky (and in this context we are), the randomly related genes will look more related to the disease than those which are actually relevant.

The only way out is to use supplementary data sources to filter the variables. In genetics, one may consider only genes which are already known to be related with the disease and genes which are known to be related to these genes. More advanced methods use variables which represent average expressions (or whatever we measure) of groups of genes, where these groups are defined based on large number of independent experiments, on gene sequence similarities, co-expressions and similar.

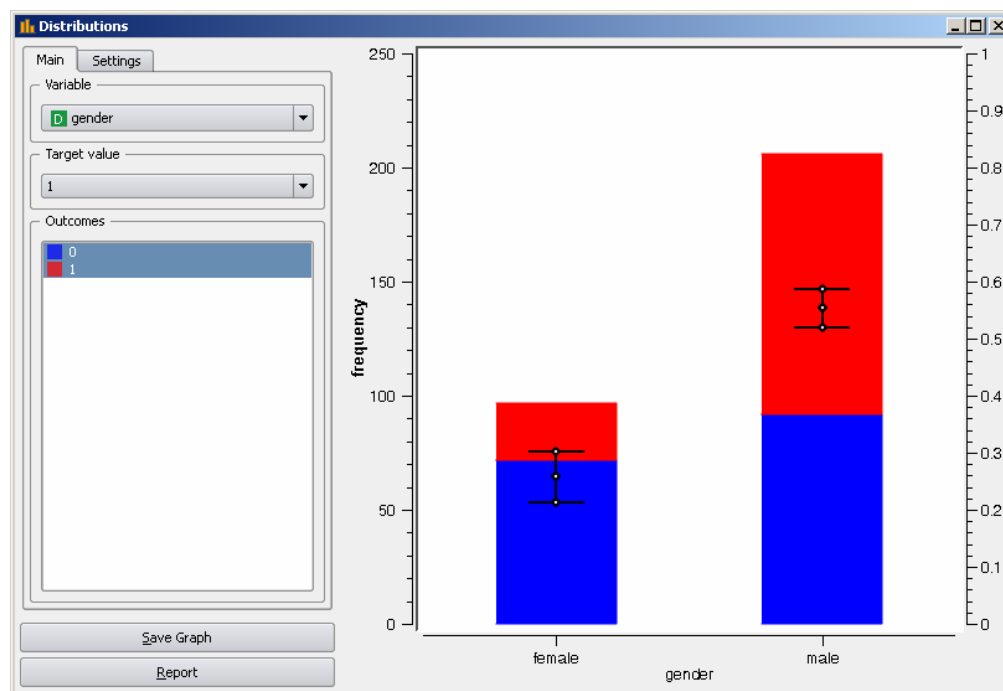
The area of dimensionality reduction is on the frontier of data mining in genetics. The progress in discovering the genetics of cancer and potential cures are not that much in collecting more data but rather in finding ways to split the data we have into irrelevant and relevant.

Visualization

We shall explore some general-purpose visualizations offered by Orange and similar packages.

Histograms

Histogram is a common visualization for representing absolute or relative distributions of discrete or numeric data. In line with its machine learning origins, Orange adds the functionality related to classified data, that is, data in which instances belong to two or more classes. The widget for plotting histograms is called Distributions. The picture below shows the gender distribution on the heart disease data set, which we briefly worked with in the last lecture.



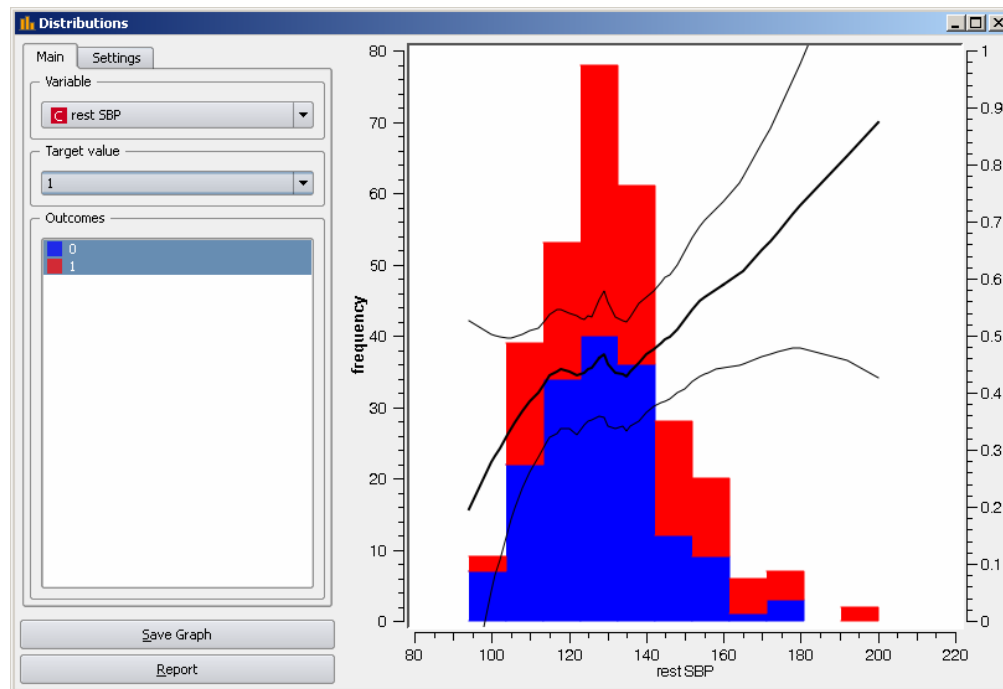
On the left-hand side, the user can choose the variable which distribution (s)he wants to observe. Here we chose the discrete attribute *gender*. The scale on the left side of the plot corresponds to the number of instances with each value of the attribute, which is shown by the heights of bars. Colored regions of the bars correspond to the two classes – blue represents outcome "0" (vessels around the heart are not narrowed), and red parts represent the patients with clogged vessels (outcome "1"). The data contains twice as many men than women. With regard to proportions, males are much more likely to be "red" (have the disease) than females.

To compare such probabilities, we can use the scale on the right, which corresponds to probabilities of the target value (we chose target value "1", see the settings on the left). The circles and lines drawn over the bars show these probabilities: for females it's around 25% (with confidence intervals appx. 20-30%),

and for males it's around 55% (51-59). You can also see these numbers by moving the mouse cursor over parts of the bar.

Note: to see the confidence intervals, you need to click *Settings* (top left), and then check *Show confidence intervals*.

Similar graphs can be drawn for numeric attributes, for instance SBP at rest.



The number of bars can be set in Settings. Probabilities are now shown with the thicker line in the middle, and the thinner lines represent the confidence intervals. You can notice how these spread on the left and right side, where there are fewer data instances available to make the estimation.

→ Hands on exercise

Plot the histogram for the age distribution. Do you notice something interesting with regard to probabilities?

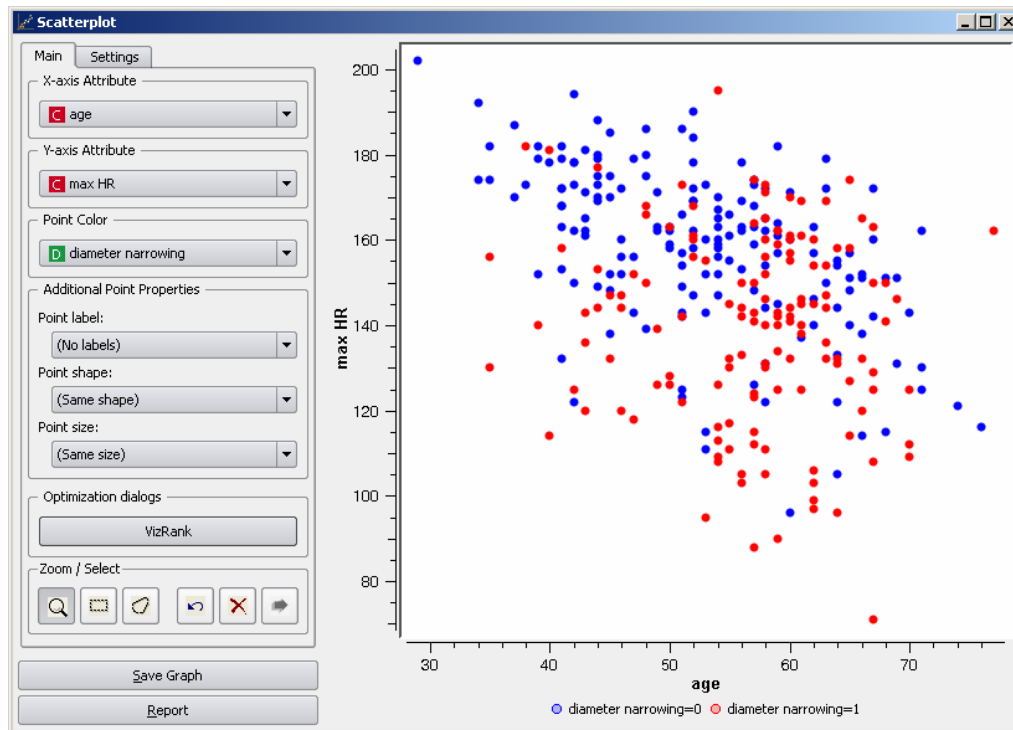
Scatter plot

Scatter plots show individual data instances as symbols whose positions in the plane are defined by values of two attributes. Additional features can be represented by the shape, color and size of symbols.

There are many ways to tweak the scatter plot into even more interesting visualizations. The Gap Minder application, with which we amused ourselves on our first lecture (Japan vs. Slovenia vs. China) is nothing else then a scatter plot with an additional slider to browse through historic data.

Although scatter plot visualizations in data mining software are less fancy than those in specialized applications like Gap Minder, they are still the most useful and versatile visualization we have, in my opinion.

Here is one of my favorites: patients' age versus maximal heart rate during exercise. Colors again tell whether the patient has clogged vessels or not.



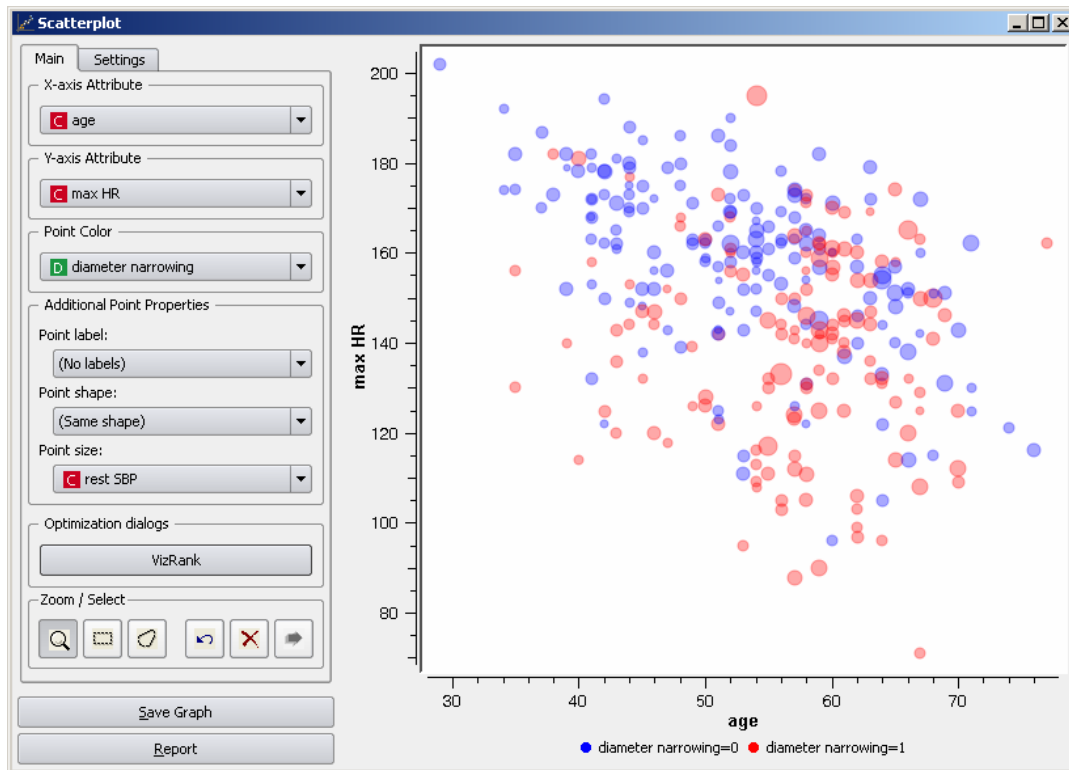
First, disregarding the colors, we see the relation between the two variables (this is what scatter plots are most useful for). Younger patients can reach higher heart rates (the bottom right side of the plot is empty). Among older patients, there are some who can still reach almost the heart rates almost as high as youngsters (although the top-most right area is a bit empty), while some patients' heart rates are rather low. The dots are spread in a kind of triangle, and the age and heart rate are inversely correlated – the older the patient, the lower the heart rate.

Furthermore, the dots right below are almost exclusively red, so these are the people with clogged vessels. Left above, people are young and healthy, while those at the top above can be both.

This is a typical example of how data mining goes: at this point we can start exploring the pattern. Is it true or just coincidental? The triangular shape of the dot cloud seems rather convincing, so let's say we believe it. If so, what is the reason? Can we explain it? A naïve explanation could be that when we age, some people still exercise a lot and their hearts are "as good as new", while some are not this lucky.

This is not necessarily true. When I showed this graph to colleagues with more knowledge in cardiology, they told me that the lower heart rate may be caused by beta blockers, so this can be a kind of confounding factor which is not included in the data. If this is the case, then heart rate tells us (or it is at least related to) whether the patient is taking beta blockers or not.

To verify this we could set the symbol size to denote the rest SBP (to make the picture clearer, I also increased the symbol size and transparency, see the Settings tab). Instead of the rest SBP, we could also plot the ST segment variable. I don't know the data and the problem well enough to be able to say whether this really tells us something or not.

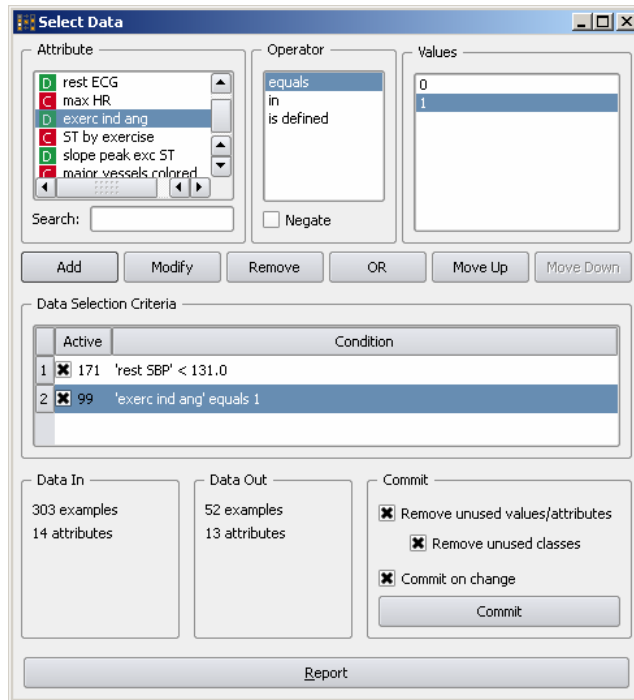
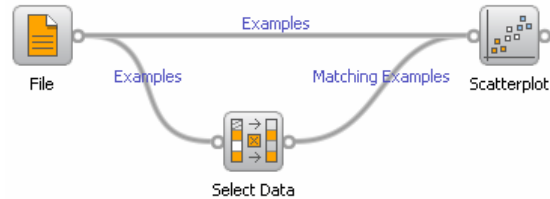


The icons in the bottom let us choose tools for magnifying parts of the graph or for selecting data in the graph (icons with the rectangle and the lasso). When examples are selected, the widget sends them to any widgets connected to its output.

→ **Hands-on exercise**

Draw the scatter plot so that the colors represent the gender instead of the outcome. Does it seem that the patients whose heart rates go down are prevalently of one gender (which)? Can you verify whether this is actually the case? (Hint: select these examples and use the Attribute statistics widget to compare the distribution of genders on entire data set and on the selected sample.)

The widget has many more tricks and uses. For instance, it has multiple outputs. You can give it a subset of examples which come from another widget. It will show the corresponding examples from this subset with filled symbols, while other symbols will be hollow. In the example below we used the Select data widget to select patients with exercise induces angina and rest SBP below 131 and show the group as a subgroup in the Scatter plot. Note that this schema is really simple: many more widgets can output the data, and the subset plotted can be far more interesting than this one.



→ **Hands-on exercise**

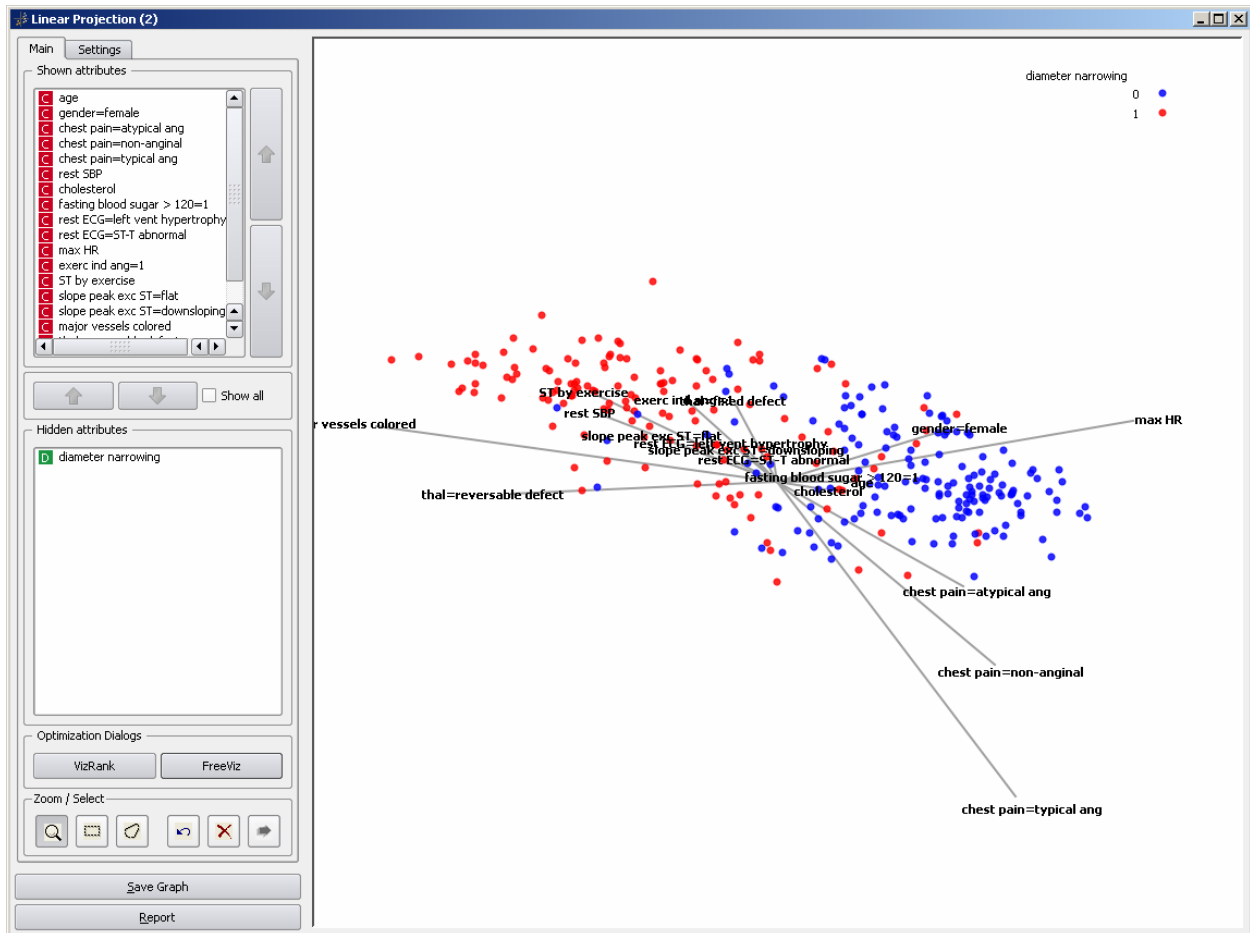
Construct the schema above and see what happens in the Scatter plot.

→ **Hands-on exercise**

Let's explore the difference in genders in some more detail. Remember the interesting finding about the probabilities of having clogged vessels with respect to the age of the patient? Use the Select Data widget to separate women and men and then use two Distribution widgets to plot the probability of disease with respect to age for each gender separately. Can you explain this very interesting finding (although not unexpected, when you think about it) from medical point of view?

There is another widget for plotting scatter plots, in which positions of instances can be defined by more than two attributes. It is called Linear projection. To plot meaningful plots, all data needs to be either continuous or binary. We will feed it through widget Continuize to ensure this.



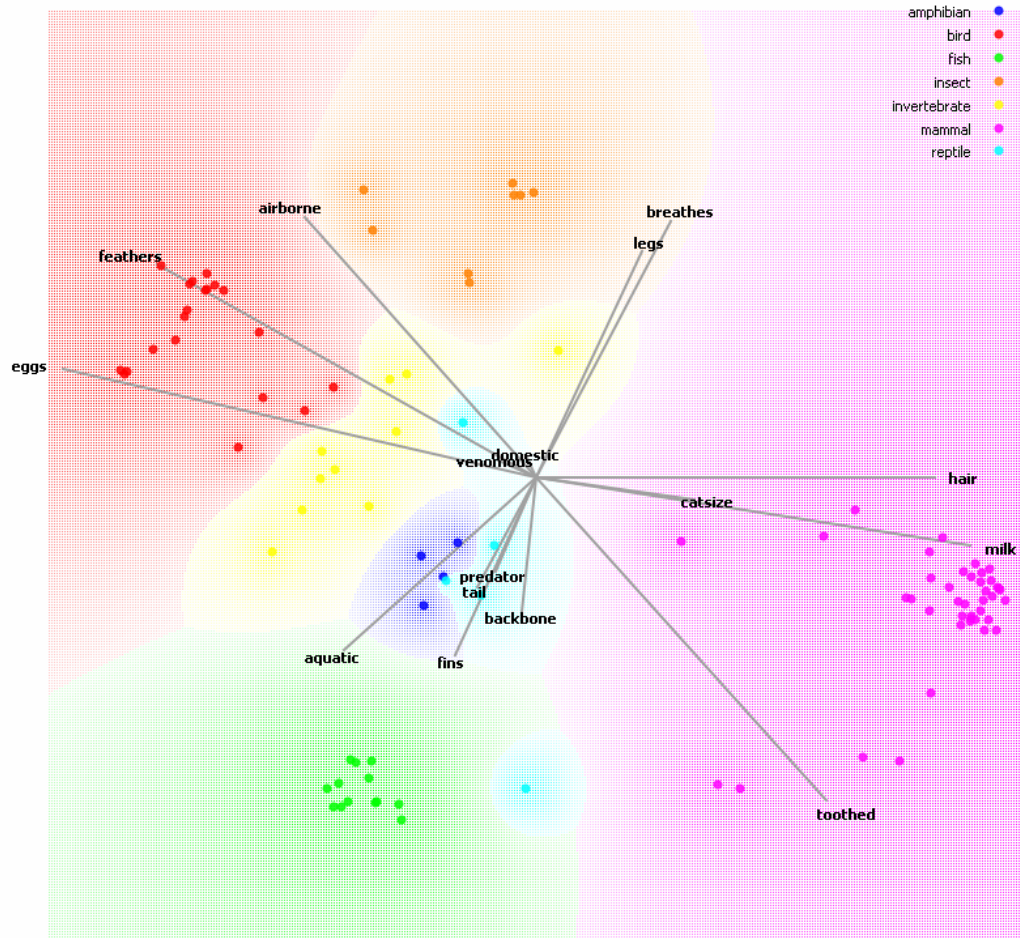


On the left-hand side we can select the variables we want visualized. For the heart disease data, we will select all except the outcome, diameter narrowing. Linear projection does not use perpendicular axis, as the ordinary scatter plot, but can instead use an arbitrary arrangement of axis. The initial arrangement is circular. To get a useful visualization, click the FreeViz button at the bottom. In the dialog that shows up, select Optimize separation. The visualization shows you, basically, which variables are typical for patients which do and do not have narrowed vessels.

Alternatively, in the FreeViz dialog you can choose the tab projection and then Supervised principal component analysis.

For an easier explanation of how to interpret the visualization, here is a visualization of another data set in which the task is to predict the kind of the animal based on its features. We have added some colors by clicking the Show probabilities in the Settings tab.

Each dot represents an animal, with their colors corresponding to different types of animals. Positions are computed from animal's features, e.g. those which are airborne are in the upper left (and those which are airborne and have teeth will be in the middle – tough luck, but there are probably not many such animals which is also the reason why features airborne and toothed are on the opposite sides).



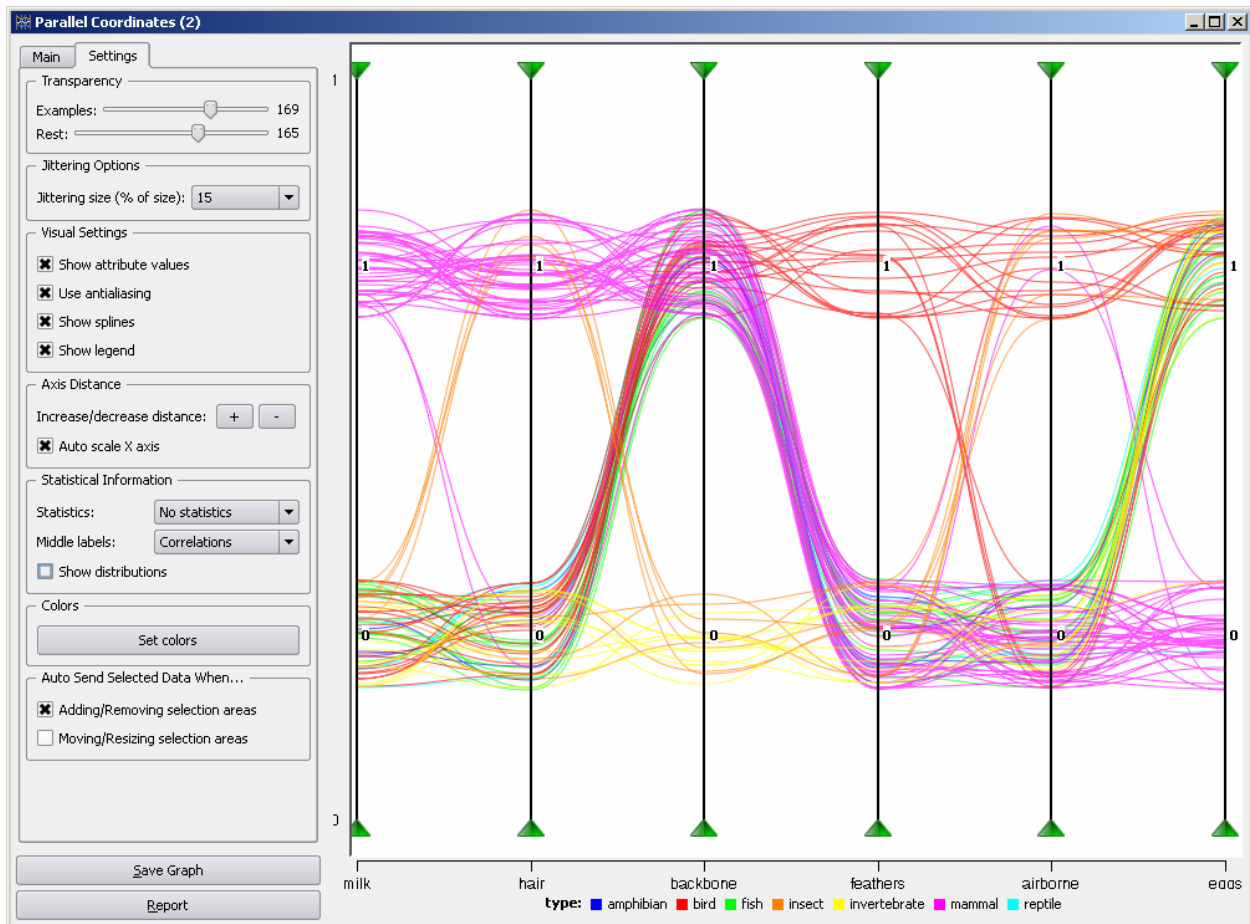
There are quite a few hypotheses (some true and some false) that we can make from this picture. For instance:

- Mammals are hairy and give milk; they are also catsize, although this is not that typical.
- Airborne animals are insects and birds.
- Birds have feathers.
- Having feathers and being airborne are related features.
- Animals either give milk or lay eggs – these two features are on the opposite sides.
- Similarly, some animals are aquatic and have fins, and some have legs and breath.
- Whether the animal is venomous and domestic tell us nothing about its kind.
- Backbone is typical for animals with fins. (?!)

Some of these hypothesis are partially true and the last one is clearly wrong. The problem is, obviously, that all features need to be placed somewhere on two-dimensional plane, so they cannot all have a meaningful place. The widget and the corresponding optimization can be a powerful generator of hypothesis, but all these hypothesis need to be verified against common sense and additional data.

Parallel coordinates

In the parallel coordinates visualization, the axis, which represent the features are put parallel to each other and each data instance is represented with a line which goes through points in the line that correspond to instance's features.

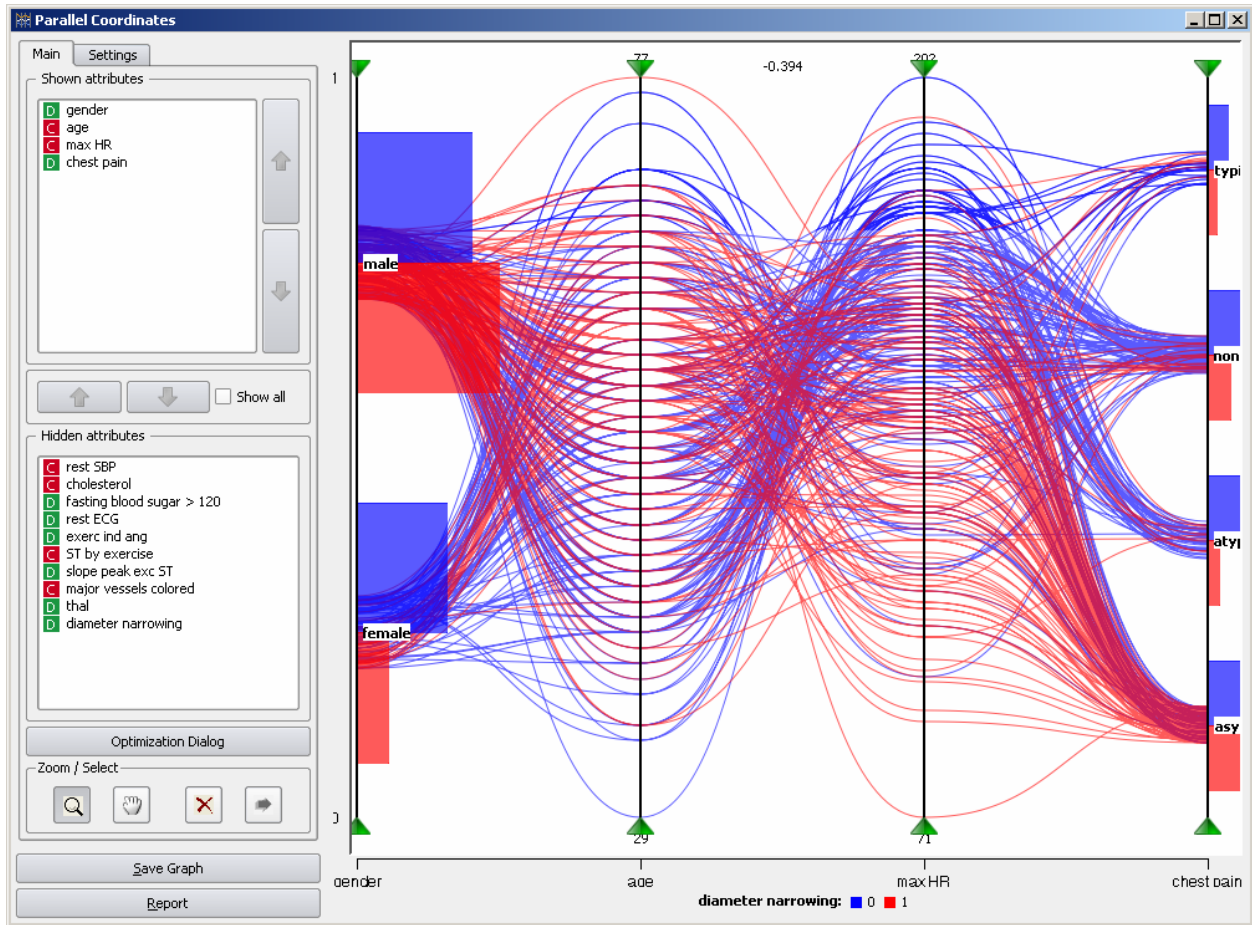


On the zoo data set, each line represents one animal. We see that mammals give milk and have hair (most pink lines go through the top of the two axis representing milk and hair). Feathery animals are exclusively birds (all lines through the top of the *feathers* axis are red). Some birds are airborne and some are not (red lines split between the *feathers* and the *airborne* axis). Fish (green lines) do not give milk, have no hair, have backbone, no feathers, cannot fly, and lay eggs (we can see this by following the green lines).

→ Hands on exercise

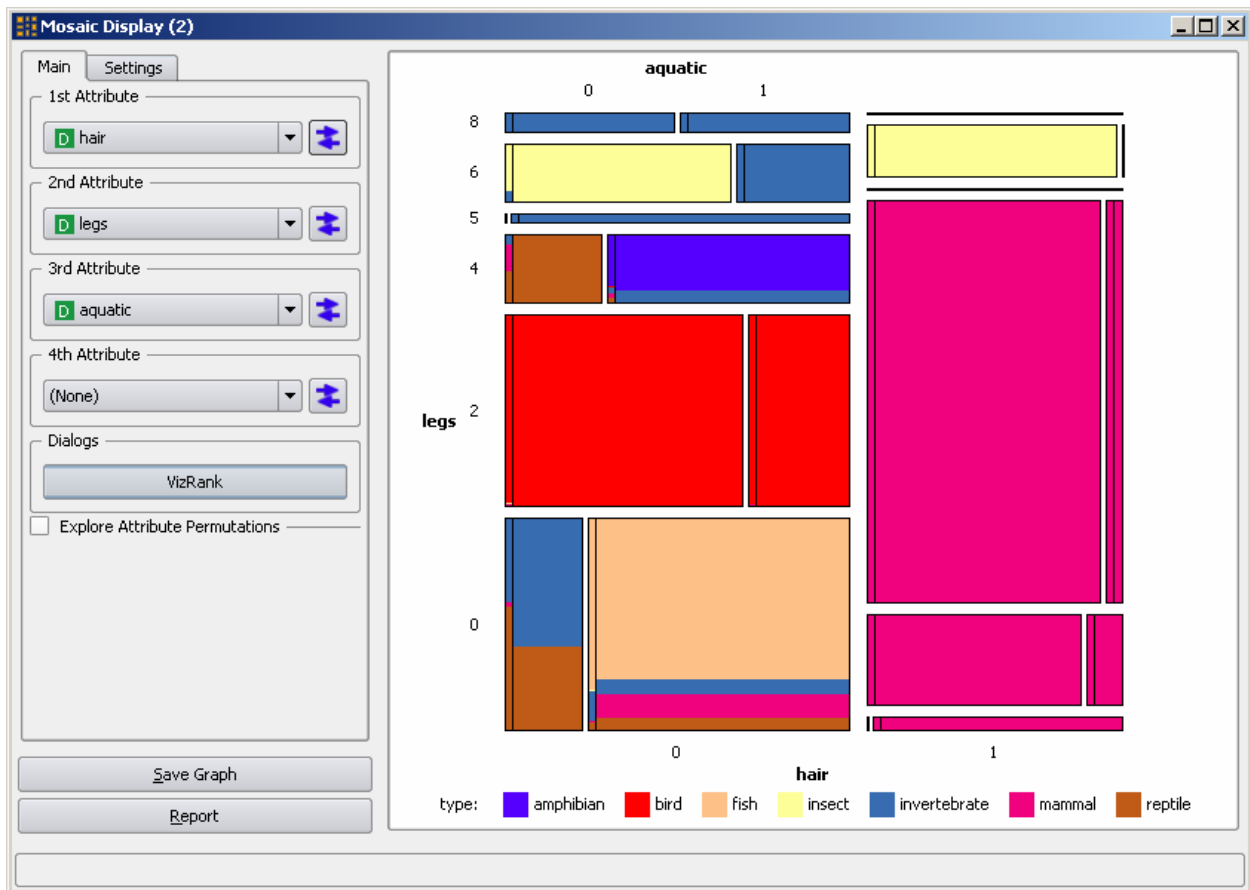
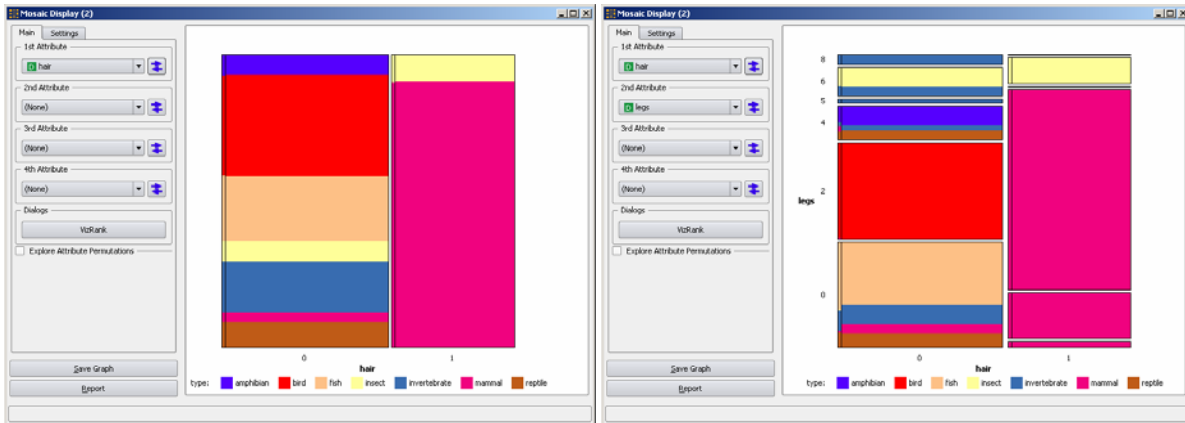
Draw the following graph for heart disease data. Select the same subset of features and arrange them in the same order. Set everything to make the visualization appear as below.

Can you see the same age vs. maximal heart rate relation as we spotted in the scatter plot? Anything else which might seem interesting? Are females in the data really older than males? Is there any relation between the maximal heart rate and the type of the chest pain?



Mosaic plot

In Mosaic plot we split the data set into ever smaller subsets based on values of discrete (or discretized) attributes. We can see the proportions of examples in each group and the class distribution. Figures below show consecutive splits of the zoo data set.



Hairy animals represent almost one half of the data set (the right box in the first picture is almost as wide as the left box). (Note: this data set does not represent an unbiased sample of animal kingdom.) The "hairy" box is violet, therefore most hairy animals are mammals. The yellowish part represents the hairy insects. There is almost no violet in the left box, therefore are almost no non-hairy mammals.

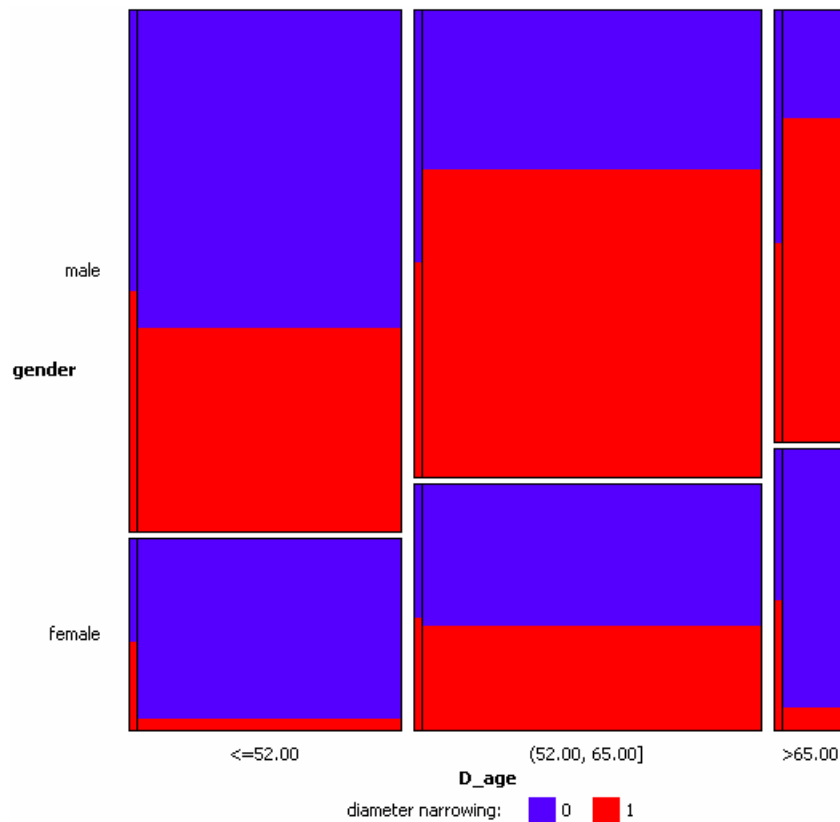
We split each of the two subgroups (non-hairy, hairy) by the number of legs. Most hairy animals have four legs, followed by those with two and those with six. There are only a few with no legs. Four- and two-legged hairy creatures are mammals, and six-legged are insects.

A third of non-hairy animals have no legs. Half of them (in our data set only, not in nature!) are fish, followed by a few invertebrates, mammals and reptiles. One third of non-hairy animals have two legs; all of them are birds. Four-legged non-hairies are amphibians and reptiles, and the six-legged are mostly insects.

For the last split, we have split each group according to whether they are aquatic or not. All mammals without legs are aquatic, while the number of aquatics among two- and four-legged is small (the vertical strips on the right-hand side are thin). There are no hairy aquatic insects.

Majority of non-hairy animals without legs are aquatic: fish, and some mammals and invertebrates. Non-aquatic ones are invertebrates and reptiles. As we discovered before, two-legged non-hairy features are birds (red). One third of them is aquatic and two thirds are not. Four-legged non-hairy animals can be non-aquatic reptiles, while a two-third majority goes to the aquatic amphibians.

Mosaic plots can also show a priori distributions or expected distributions (in a manner similar to those of the chi-square contingency table, that is, the expected number of instances computed from the marginal distributions). The plot below splits the data on heart disease by age group and gender. The small strips on the left side of each box show the prior class distribution (the proportion of patients with and without narrowed vessels on the entire data set).



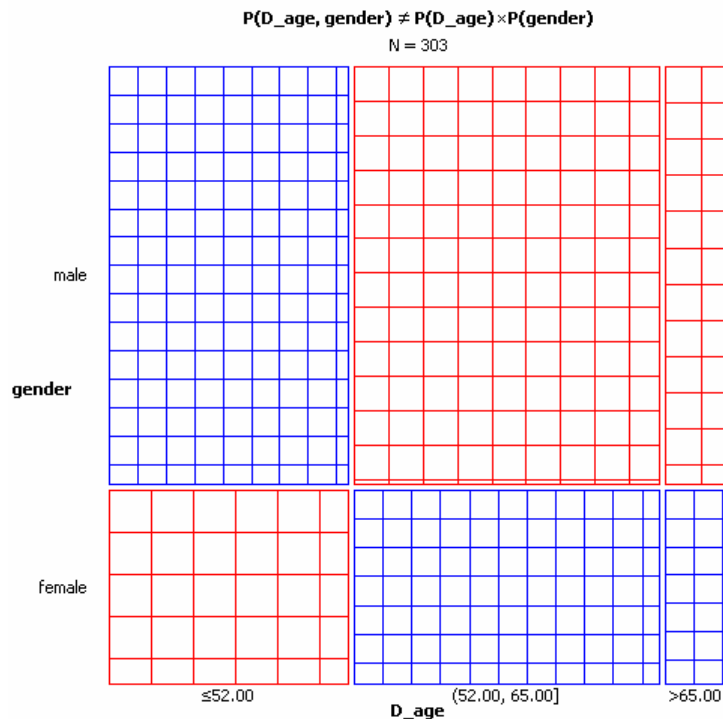
The occurrence of narrowed vessels is very small among young and old females, whereas in the middle age group the proportion is far larger (but still below the total data set average! the problem is so much more common among males!) For males, the proportion simply increases with age.

→ **Hands-on exercise**

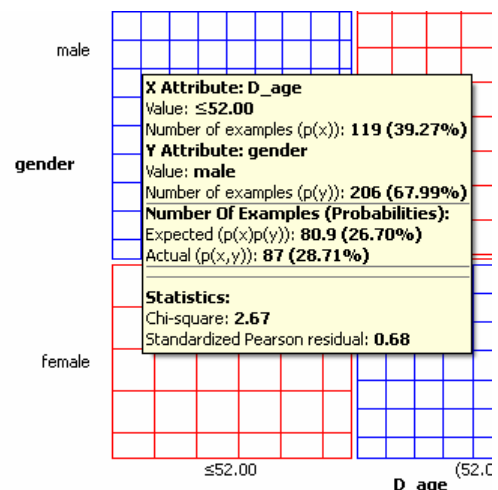
How did we actually plot this? If we feed the data with continuous variables into the Mosaic widget, it will automatically split values of numeric attributes into intervals using one of the built-in discretization methods. Here we manually defined the age groups by using the Discretize widget. Try to do it yourself! (Hint: put Discretize between File and Mosaic. In Discretize, click Explore and set individual discretizations. Select Age and enter the interval thresholds into the Custom 1 box or define the intervals on the graph below – left-clicking adds and right-clicking removes thresholds, and you can also drag thresholds around. Do not forget to click Commit in the widgets, or check Commit automatically.)

Sieve diagram

Sieve diagram, also called parquet diagram, is a visualization of two-way contingency tables, similar to those used in chi-square. The rectangle representing the data set is split horizontally and vertically according to the proportion of values of two discrete attributes.



Note the difference between this and the mosaic plot. In mosaic plot, the data is split into ever smaller subgroups: each age group is split *separately* into groups according to the distribution of genders within that specific age group. The size



of the box in the Mosaic plot thus corresponds to the *observed (actual) number of instances* of each kind. For contrast, the Sieve diagram splits by two attributes independently, so the size of the rectangle reflects the *expected number of instances*. This number can be compared to the observed number: Sieve plot shows the comparison by grid density and color (blue represents a higher than expected number of instances and red represents a lower than expected number). The diagram above suggests that, for instance, younger males are more common than expected according to prior distribution. We can check the number by moving the mouse cursor over the corresponding box.

Given that the younger age group includes 39.27% of data and males represent 67.99%, we would *expect* $39.27\% \times 67.99\% = 26.70\%$ young males. The *actual* number is 28.71%, which is more than expected.

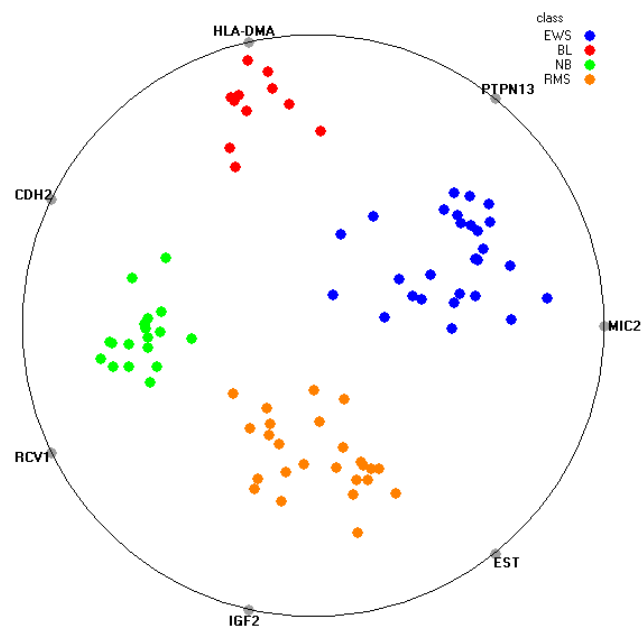
This difference is, however, negligible. Here is a better example. In the figure below, the vertical split follows the distribution of genders (the male part is twice the height of the female part), and the vertical split corresponds to results of the thal test (one half of instances have normal results and most others have reversible defect).

The number of males with normal thal test is 28.57% (86 patients), which is less than the expected 37.56% (113 patients). The number of women with normal thal test is unexpectedly high, 80 instead of the expected 53. And so forth.



"Intelligent" visualizations

There are many different scatter plots we can draw for some data, and the number of different more complex visualizations is even higher. Orange features a set of tools for finding good data visualizations. For instance, this plot (a variation of scatter plot called RadViz), which was found automatically by Orange, uses expressions of 7 genes out of 2308, to distinguish between the four types of SRBCT cancer. In most widgets you will find buttons Optimize or VizRank (Visualization ranking), which will try out huge number of different visualizations and present those which seem most interesting.

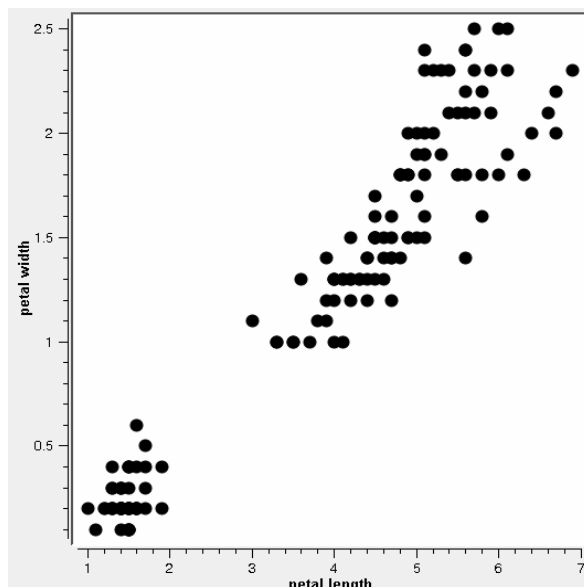


Data Modeling (Learning, Clustering...)

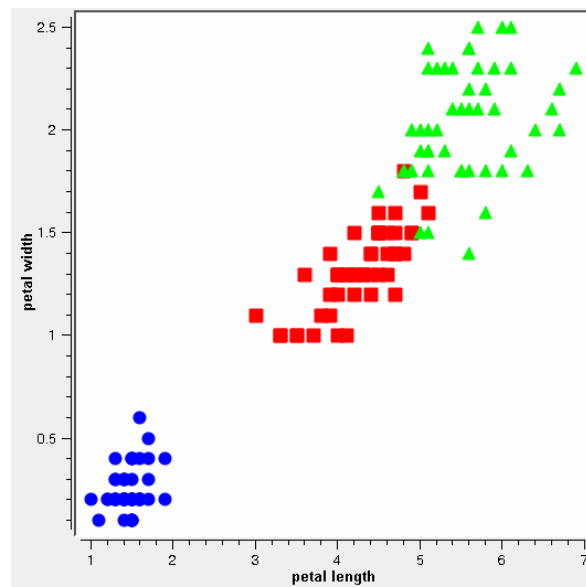
Modeling refers to describing the data in a more abstract form called, obviously, a *model*. The model (usually) does not include the data on individual instances from which it is constructed, but rather describes the general properties entire data set (machine learning often refers to this as *generalization*).

We usually distinguish between *unsupervised* and *supervised* modeling. Roughly, imagine that data instances belong to some kind of groups and our task is to describe these groups. The computer's work can be *supervised* in the sense that the computer is told which instances belong to which group. We called these groups "classes"; we say that we know the class of each example. The computer needs to *learn* how to distinguish between the two groups. Such knowledge is represented by a model that, if in a proper form, can teach us something (new, potentially useful, actionable...) about the data. Moreover, we can use it to determine the group membership of new data instances. For example, if the data represents two groups of people, some are sick and some are not, the resulting model can classify new patients into one of these two groups.

In unsupervised learning, the instances are not split into classes, so the computer is not guided (supervised) in the same sense as above. The task is typically to find a suitable grouping of instances. In statistics, this is called *clustering* (cluster being a synonym for a group). We also often use visual methods which plot data in such a way that groups become obvious to the human observer, who can then manually "draw the borders" between them or use the observed grouping for whatever he needs.



Data for unsupervised learning



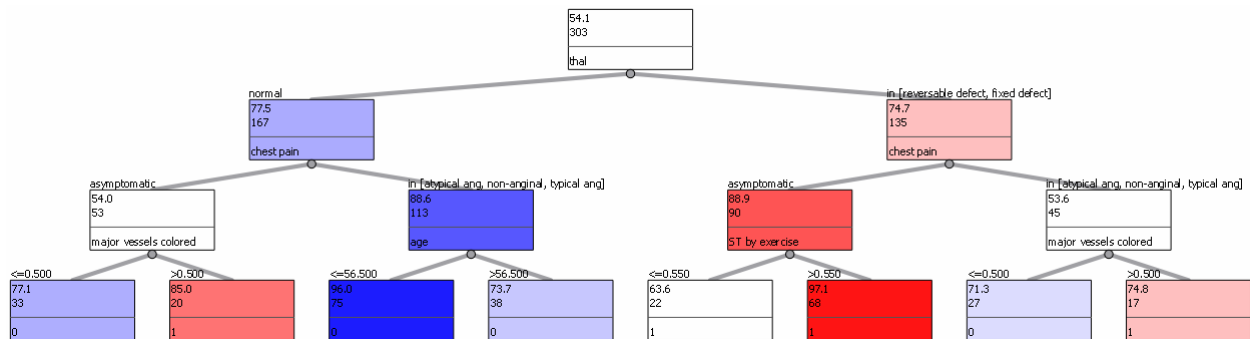
... and for supervised learning

Predictive modeling / Learning

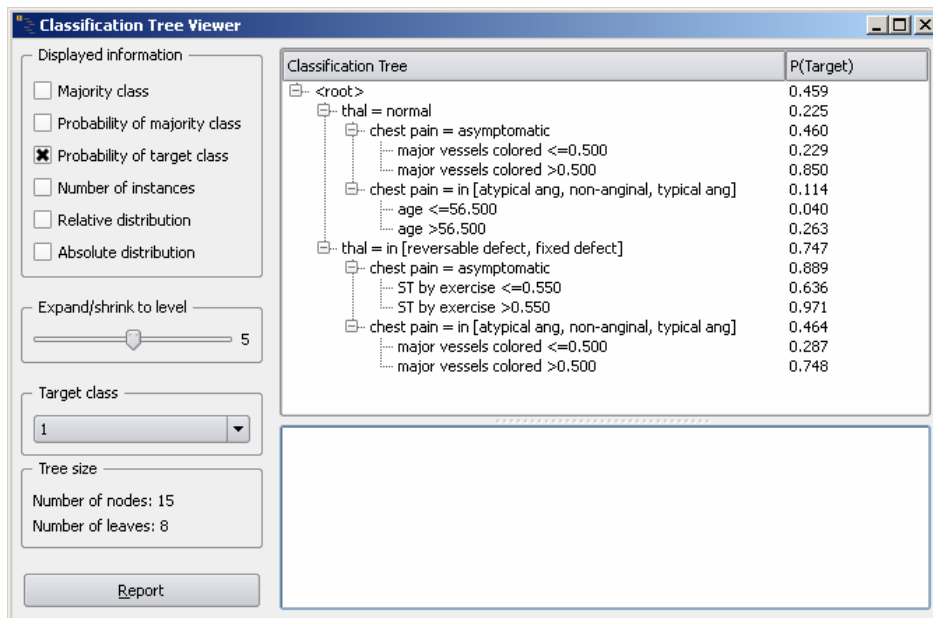
The task of predictive modeling is to use the given data (training examples, learning examples) to construct models which can predict classes or, even better, class probabilities for new, unseen instances. In the context of data mining, we would like these models to provide some insight into the data: the models should be interpretable; they should show patterns in the data. The models should be able to explain their predictions in terms of the features – the patient is probably sick, and the reason why we think so is because he has these and these symptoms.

Classification trees

Induction of *classification trees* (also called *decision trees*) is a popular traditional machine learning method, which produces models as the one on the figure below.

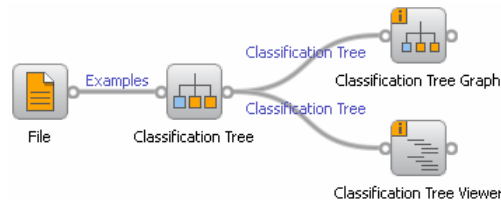


While this is visually pleasing, the tree as presented by the below widget may be easier to read.



We have used three widgets related to trees: **Classification Tree** gets the data and outputs a tree (this is the first widget we have met so far which does not output data instance but something else, a model!). **Classification Tree Viewer** and **Classification Tree Graph** get trees on their inputs and show them. They

also provide output: when you click on tree nodes, they output all training examples which belong to the node. You will find all these widgets in category **Classify**.



Say we get a new patient. The tree suggests that we should first perform the thal test (whatever this means). Let's say that results are normal. For such patients we then ask about the chest pain; suppose that it's typical anginal. In that case, we check the age of the patient. Say he is 45, which is below 56.5. Therefore we predict a 0.04 probability of having the disease.

The algorithm for construction of such trees from data works like this. First, it checks all available variables and finds one which splits the data into subsets which are as distinct as possible – in ideal case, the attribute would perfectly distinguish between the classes, in practice, it tries to get as close as possible. For our data, it decided for thal test. Then it repeats the search for the best attribute in each of the two subgroups. Incidentally, the same feature (the type of chest pain) seems the best attribute in both. The process then continues for all subgroups until certain stopping criteria are met: the number of data instances in a group is too small to split them further, the subgroups are pure enough (ideally, all instances belong to the same class, so there is no need to split them any further), there are no more useful features to split by, the tree is too large *etc.*

Trees are great for data mining, because

- there's no sophisticated mathematics behind them, so they are conceptually easy to understand,
- a tree can be printed out, analyzed and easily used in real-life situations,
- in data mining tools, we can play with them interactively; in Orange, we can connect the tree to other widgets, click the nodes of the tree to observe the subgroups and so on.

They are, however, not so great because

- while they may perform relatively well in terms of proportion of correct predictions, they perform poorly in terms of probability estimates;
- they are "unstable": a small change in the data set, like removing a few instances, can result in a completely different tree. Same for changing the parameters of the learning algorithm. Trees give one perspective of the data, one possible model, not a special, unique, distinguished model;
- trees need to be kept small: computer can easily induce a tree with a few hundred leaves, yet such trees are difficult to understand and to use, and the number of instances in individual leaves is then so small that we cannot "thrust" them. On the other hand, small trees refer to only a small number of attributes, e.g. to only a small number of medical symptoms related to the disease.

The latter point is really important. Trees are good for conceptually simple problems, where a few features are enough to classify the instances. They are a really bad choice for problems in which the prediction must be made by observing and summing up the evidence from a larger number of features. A usefully small tree cannot contain more than, say, ten features. If making the prediction requires observing more variables, forget about the trees.

→ **Hands-on exercise**

Construct a tree like the one above, then plot the same data in a Mosaic plot in which you select the attributes from the root of the tree and the first branches of the tree.

Classification rules

Classification rules are essentially similar to trees, except that instead of having a hierarchy of choices, they are described by lists of conditions, like those below.

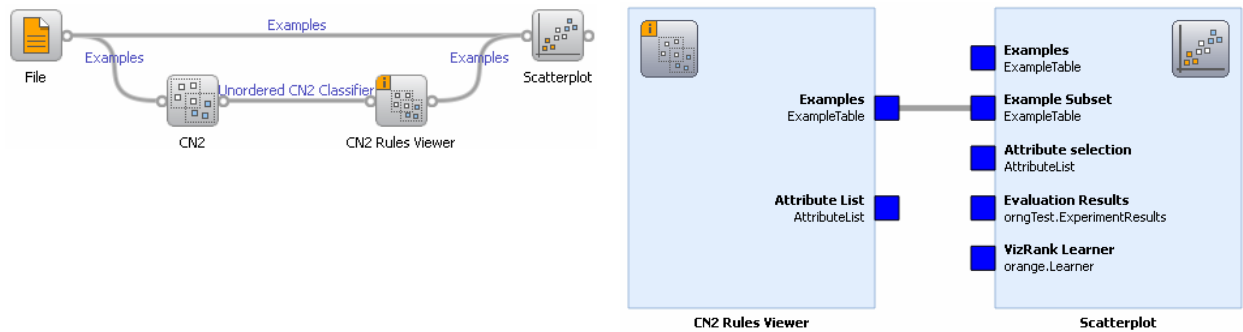
Length	Quality	Coverage	Class	Distribution	Rule
6	0.955	20.0	0	<20.0,0.0>	IF major vessels colored<=0.000 AND exerc ind ang=['0.000'] AND age>50.000 AND cholesterol<=258.000 AND age<=66.000 AND ST by exercise<=2.600 THEN diameter narrowing=0.000
6	0.682	20.0	0	<14.0,6.0>	IF rest ECG=['normal'] AND ST by exercise<=0.800 AND max HR<=163.000 AND exerc ind ang=['0.000'] AND max HR<=162.000 AND rest SBP>100.000 THEN diameter narrowing=0.000
5	0.625	22.0	0	<14.0,8.0>	IF age>63.000 AND ST by exercise<=2.000 AND max HR<=160.000 AND major vessels colored<=2.000 AND rest SBP>100.000 THEN diameter narrowing=0.000
3	0.980	47.0	1	<0.0,47.0>	IF thal=['reversible defect'] AND max HR<=143.000 AND ST by exercise>0.400 THEN diameter narrowing=1.000
4	0.964	26.0	1	<0.0,26.0>	IF chest pain=['asymptomatic'] AND major vessels colored>0.000 AND rest ECG=['left vent hypertrophy'] AND cholesterol<=300.000 THEN diameter narrowing=1.000

The algorithm for construction of such rules starts with an empty rule (no condition, it predicts the same class for all examples). It then adds conditions, which specialize the rule, fitting it to a particular group of instances until it decides that it's enough (stopping criteria are based on the purity of the group, number of instances covered by the rule and some statistical tests). It prints out the rule, removes the data instances covered by the rule and repeats the process with the remaining instances.

→ **Hands-on Exercise**

Construct the below schema. Examples coming from the CN2 Rules Viewer should be connected to Example Subset of the Scatter plot (the simplest way to have this is to first draw the connection between File and Scatterplot and then between the CN2 Rules Viewer and Scatterplot).

Now click on individual rules in the CN2 Rules Viewer and the Scatter plot will mark the training instances covered by the rule.



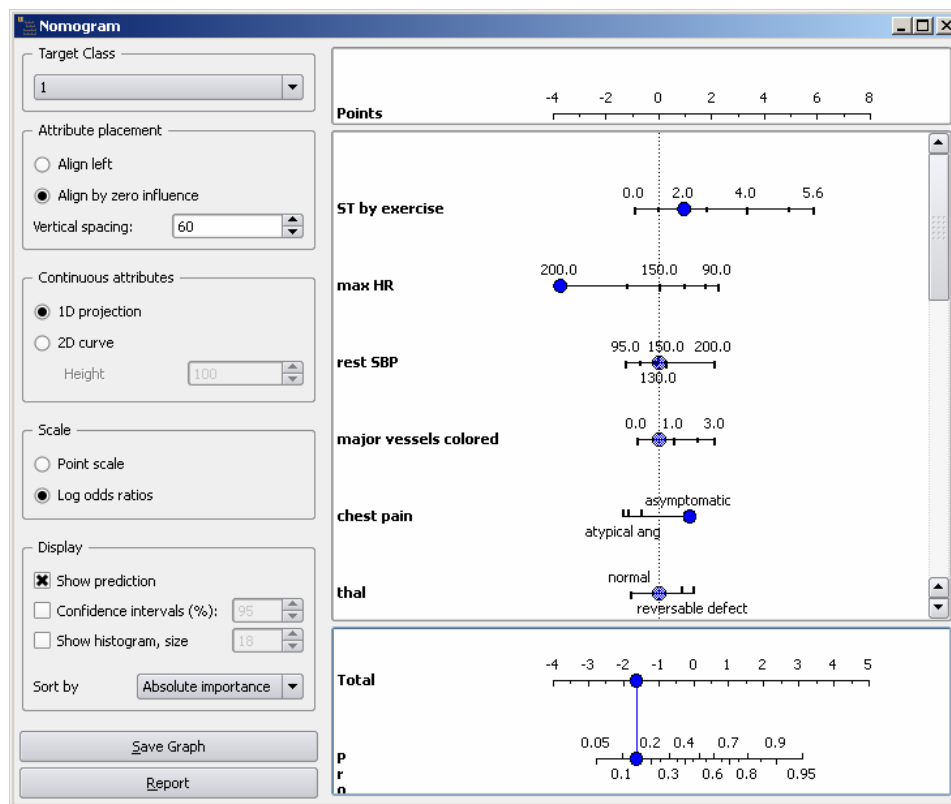
Classification rules have similar properties as the trees, except that their shortcomings are somewhat less acute: they can handle more attributes and are better in predicting probabilities. They may be especially applicable to problems with multiple classes, as in the following examples from our beloved zoo data set.

Length	Quality	Coverage	Class	Distribution	Rule
3	0.714	5.0	amphibian	<4.0,0.0,0.0,0.0,1.0,0.0,0.0>	IF legs=["4.000"] AND milk=["0.000"] AND aquatic=["1.000"] THEN type=amphibian
1	0.955	20.0	bird	<0.0,20.0,0.0,0.0,0.0,0.0,0.0>	IF feathers=["1.000"] THEN type=bird
2	0.933	13.0	fish	<0.0,0.0,13.0,0.0,0.0,0.0,0.0>	IF fins=["1.000"] AND eggs=["1.000"] THEN type=fish
2	0.900	8.0	insect	<0.0,0.0,0.0,8.0,0.0,0.0,0.0>	IF legs=["6.000"] AND aquatic=["0.000"] THEN type=insect
3	0.900	8.0	invertebrate	<0.0,0.0,0.0,0.0,8.0,0.0,0.0>	IF backbone=["0.000"] AND predator=["1.000"] AND airborne=["0.000"] THEN type=invertebrate
2	0.429	5.0	invertebrate	<0.0,0.0,0.0,0.0,2.0,0.0,3.0>	IF legs=["0.000"] AND fins=["0.000"] THEN type=invertebrate
1	0.977	41.0	mammal	<0.0,0.0,0.0,0.0,0.0,41.0,0.0>	IF milk=["1.000"] THEN type=mammal

Naïve Bayesian Classifier and Logistic Regression

Naïve Bayesian classifier computes conditional class probabilities, with feature values as conditions. For instance, it computes the probability for having the disease for females and for males, the probability for having the disease for younger and older patients, for those with non-anginal pain, typical anginal pain and atypical anginal pain, and so on. To predict the overall probability for having the disease, it "sums up" the conditional probabilities corresponding to all data about the patient.

The model can be simplified into summing "points" plotted in a nomogram, like the one shown below. Suppose we have a patient whose ST by exercise equals 2, his heart rate goes up to 200, and he feels asymptomatic chest pain – these mappings can be seen by "projecting" the positions of the points corresponding to the values onto the axis above the nomogram. Altogether, this gives $+1 + -3.75 + 1 = -1.75$ points. The scale below transforms the points into actual probabilities: -1.75 points correspond to around 15% chance of having the disease.



There is a similar model from statistics called *linear regression*. On the outside, they look the same: they can both be visualized using a nomogram. Mathematically, they differ in the way they compute the number of points given for each feature: naïve Bayesian classifier observes each feature separately and logistic regression tries to optimize the whole model at once. We shall not go into mathematics here. We will only describe the practical consequences of the difference in the two approaches.

These models are just great.

- They are very intuitive, easy to explain; they can be printed out and used in the physician's office.
- They give surprisingly accurate predictions.
- The predictions can be easily explained: the strongest argument that the person from our example is healthy is his extremely high heart rate. The other two pieces of data we have, the ST by exercise and the type of chest pain both actually oppose the prediction that he is healthy, yet their influence is smaller than that of the heart rate. The inside view into how the model makes prediction allows the expert to decide whether they make sense or whether they should be overruled and ignored.
- The model can show us which features are more important than others – in general, without a particular given instance. ST by exercise and maximal heart rate are, in general, the two most informative features, since they have the longest axis – they can provide the strongest evidence (the greatest number of points) for or against the disease. The chest pain and thal test are the least important (among those shown in the pictures). A feature can also have zero influence, when all its values are on the vertical line (0 points). A feature can sometimes even speak only in favor or only against the target class; there may be tests which can, if their results come out positive, tell that the patient is sick, while negative result may mean nothing.

What about their shortcomings? For the naïve Bayesian classifier, its shortcomings come from observing (that is, computing the strength) of each feature separately from others.

- In case of diseases whose prediction requires observing two interacting features (for a naïve example, a disease which attacks young men and old women), it cannot "combine" the two features. Such diseases (and, in general, such features) are however rare in practice.
- Another, more common problem are correlated features. Observing each feature separately, naïve Bayesian classifier can count the same evidence twice. Consider a child disease whose probability decreases with child's age; there are no other factors related to the disease. If the data we collected contains children's age and body height, naïve Bayesian classifier will (wrongly?) consider height as an important factor although this is only because of its relation with age. Counting the same piece of evidence twice causes the naïve Bayesian classifier to "overshoot" the probability predictions.

The latter is not much of a problem: the classifier is still pretty accurate. Yet the difference is important for comparison with logistic regression: logistic regression does not have this problem. The class probabilities it predicts are typically very accurate. And the price?

- Logistic regression cannot be used to evaluate importance of individual features. In the childhood disease example from above, logistic regression might say that disease probability is correlated by age alone, and not by the height, which would be correct. On the other hand, it can say that the only important attribute is body height, while age cannot offer any new evidence, which is misleading. Usually it does something in between.

- Logistic regression cannot handle unknown values. All data has to be known to classify a patient. Not so for naïve Bayesian classifier for which we can add just new evidence when we get it.

Both models, especially naïve Bayesian classifier, are good for summing up the evidence from a large number of features. They can learn from a single feature, but they can also sum up small pieces of evidence from thousands of features. For instance, Naïve Bayesian classifiers are often used for spam filtering in which each word from the message (and all other properties of the message) contribute some evidence for the mail being spam or ham.

While we avoided any theory here, there is really a lot of it in behind (unlike classification trees, which are a simple machine learning hack). Logistic regression is a modification of linear regression, which uses logistic function for modeling of probabilities. The "points" assigned to features are actually *log odds ratios*, which are a more practical alternative to conditional probabilities... Naïve Bayesian classifier represents the simplest case of *Bayesian network*, a model which can incorporate probabilistic casual models. Moreover, there is a whole branch of statistics, called *Bayesian statistics*, which represents an alternative to the inferential statistics (testing of hypothesis *etc*) and is growing fast in popularity and usefulness.

Other methods

There are many other similar methods. Here we only mentioned the simple ones, yet, by my experience, naïve Bayesian classifier can easily compete – and lose by only a small difference – with the most powerful and complex machine learning algorithms like, for instance, support vector machines. If you want models which are simple to use and understand, you should logistic regression and naïve Bayesian classifiers. The possible few percent loss in performance will be more than paid of by your ability to explain and, if needed, overrule the predictions it makes.

Unsupervised modeling

Distances

Unsupervised modeling is often based on a notion of *distance* or *difference* between data points. In order to do any grouping based on similarity, we need to somehow define the *similarity* (or its opposite, dissimilarity, difference...).

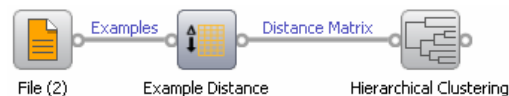
When dealing with ordinary data, the most common definition is based on Euclidean distance. If a and b are two data instances described by values (a_1, a_2, a_3) and (b_1, b_2, b_3) , we can define the difference between them as $(a_1-b_1)^2+(a_2-b_2)^2+(a_3-b_3)^2$ (we can take a square root of that, but it won't change much). We only need to be careful to put all numbers on the same scale – a difference of 0.1 m in body height may mean much more than a difference of 0.1 kg in body weight. The simplest (although not necessarily the best) way of putting the data on the same scale is to normalize all variables by subtracting the average and dividing by standard variance.

For discrete data, we usually say that when two values are different, the difference between them is 1.

Sometimes the distance is not so trivial, especially when the data instances are not described by features/attributes/variables. Say that objects of my study are lecturers at Kyoto University (including occasional speakers, like me) and we would want to cluster them based on their scientific areas. How do we define a meaningful difference between them? This depends, of course, on the purpose of the grouping. Say that I would like to group them by their fields of expertise (and that I don't know the departments at which they actually work). The problem is actually easier than it looks: we have access to public data bases of scientific papers. Say that Dr. Aoki and I have written 3 papers together, while altogether (that is, Dr. Aoki, me or both of us in co-authorship) we wrote 36 papers (I am making this up!). We can then say that the similarity between us is $3/36$ or 0.08. The similarity between Dr. Nakayama and me is then 0 (we have no common papers, so the similarity is zero divided by something). Two professors who wrote all their papers together would have a similarity of 1. To compute the difference, we simply subtract the similarity from 1: the difference between Dr. Aoki and me would be $1 - 0.08 = 0.92$. This definition of similarity is known as the Jaccard index (http://en.wikipedia.org/wiki/Jaccard_index). The purpose of this discussion is not to present the Jaccard index by rather to demonstrate that there may be cases in which the definition of distance/difference/similarity requires a bit of creativity. (See an example on the web page: a "map" of scientific journals from computer science, where the similarity between a pair of journals is based on whether the same people contribute papers to both of them.)

Hierarchical clustering

We will show how this works on animals. All data in the zoo data set is discrete, so the dissimilarity (distance) between two animals equals the number of features in which they differ. The schema in Orange Canvas looks like this:



Example Distance widget gets a set of training instances and computes the distances between them. A couple of definitions are available, but all are attribute-based in the sense that they somehow sum up the differences between values of individual attributes.

Hierarchical clustering of this data is shown on the left picture, and the right picture shows a part of it. (I haven't constructed the data and have no idea why girls were included as animals similar to bears and dolphins.)

The figure is called a *dendrogram*. Probably not much explanation is needed with regard to what it means and how it needs to be read: it merges the animals into ever larger groups or, read from left to right, splits them into ever smaller groups.

Clustering is intended to be informative by itself. If I came from another planet and have no idea about the Earthly animals, this can help me organize them into a meaningful hierarchy.

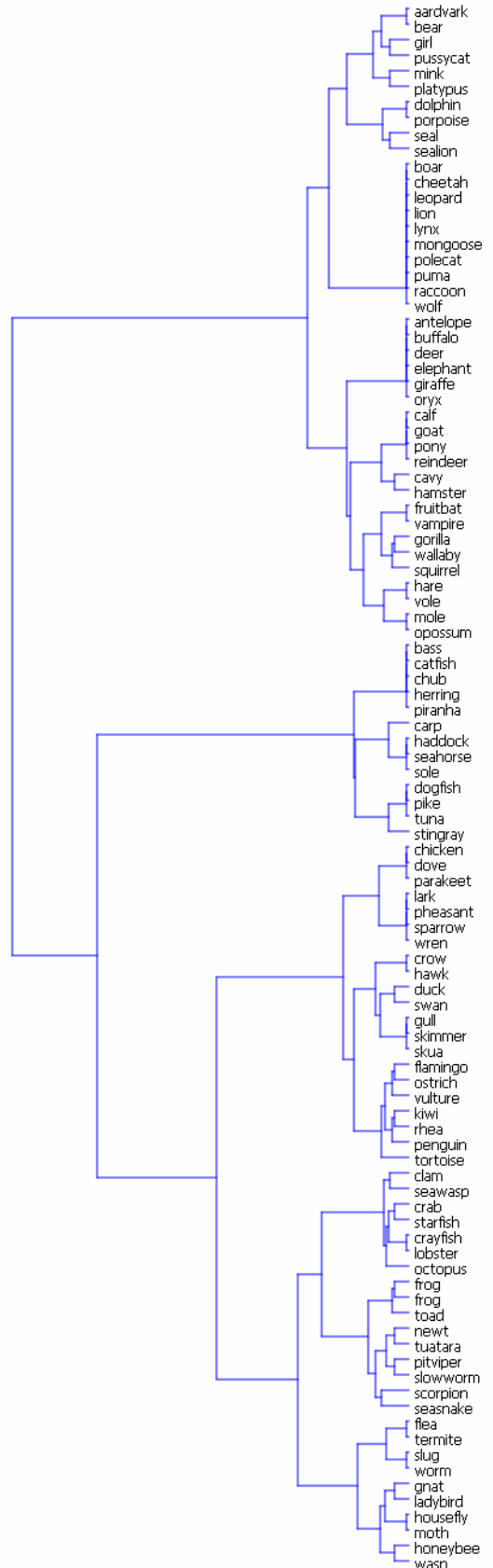
We can play with the clustering widget by drawing a vertical threshold and "cut" the clustering at a certain point, defining a certain number of distinct groups.

For the curious: this is how clustering actually works. The method goes from the bottom up (or, in this visualization, from right to the left). At each step, it merges the two most similar clusters into a common cluster. It starts so that each animal (object, in general) has its own cluster, and finishes the merging when there is only a single cluster left.

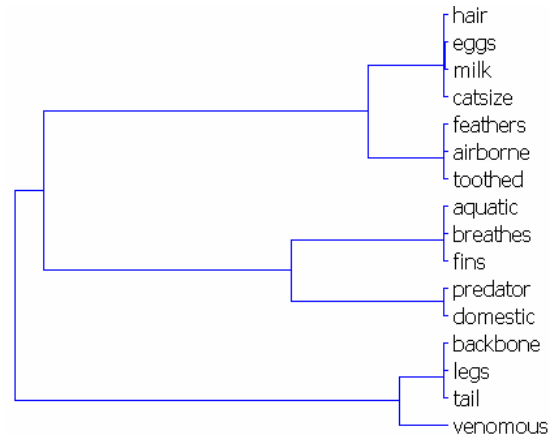
The only detail to work out is how do we define distances between clusters, if we know (that is, have defined, using Euclidean distances or Jaccard index or whatever) the distance between single objects. The first choice is obvious: the distance between two clusters can be the average distance between all objects in one cluster and objects in another. This is called *average linkage*. The distance can be also defined as the distance between the closest objects belonging to the two clusters, the so-called *minimal linkage*. As you can guess, the *maximal linkage* is the distance between the most distant pair of objects belonging to the two clusters. And nobody can guess what the fourth option, *Ward linkage*, is. Neither does it matter: remember that the term *linkage* refers to the way clusters are compared to each other. When constructing the clustering, try all of them and find the one which gives the nicest picture.

The length of horizontal lines in the dendrogram corresponds to the similarity between clusters. The objects on the lowest level are similar to each other, later the distances usually grow. The optimal number of clusters is typically one at which we notice a sharp jump in the distances/line lengths.

We can also turn the data around and compute distances between variables instead of between animals. We can, for instance, compute the Pearson correlation coefficient (which is actually closely related to the Euclidean distance).

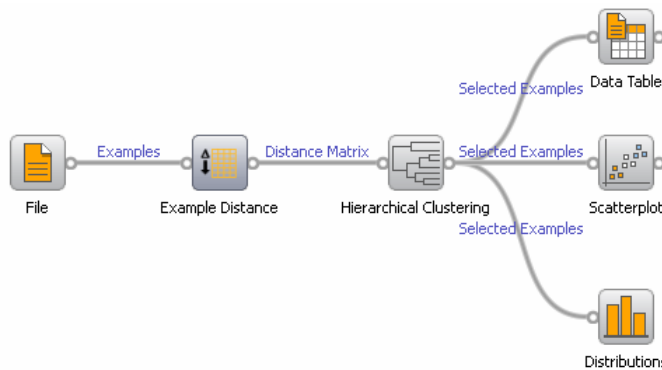


Results may seem a bit strange at first: having hair is closely related to giving milk, that's OK. But having milk and laying eggs?! Being aquatic, having fins and ... breath?! It makes sense, though: laying eggs and giving milk is strongly correlated, though the correlation is in this case *negative*. Knowing that the animal is aquatic tells us something about it having fins, but also something about whether it can breathe or not. We could, of course, define distance differently and get clustering of features which actually *co-occur* (*positive correlation*). As always in unsupervised modeling, it's all a question of how we define the distance.



→ **Hands-on Exercise**

Load the data set called Iris.tab, compute Euclidean distances and construct a hierarchical clustering, and connect the result to the Scatter plot (the middle part of the schema below).



For the clustering widget, see the recommending setting on the right. In particular, check **Show cutoff line** and **Append cluster IDs**, set the place to **Class attribute**.

The best pair of attributes to observe in the Scatter plot is petal length and petal width.

Now you can put a cut-off line into the dendrogram; the dendrogram will be split into discrete clusters shown by colors and the widget will output instances classified according to their cluster membership, that is, the "class" of an instance will correspond to the cluster it belongs to. The Scatter plot will color the instances according to this, cluster-based class instead of the original class. Play with the widget by setting different thresholds and observing the results in the Scatter plot. (If the widget does not output examples uncheck **Commit on change** and check it again, or

Linkage
Average linkage

Annotation
iris
Text font size 12
Line spacing 9

Dendrogram settings
 Limit print depth
Depth 7
 Manually set horizontal size
Size 440

Selection
 Show cutoff line
 Append cluster IDs
Name HC_class
Place Class attribute
 Commit on change
Commit

leave it unchecked and push **Commit** when you change of the threshold; sorry for the bug.)

You can also attach the **Distribution** widget to the clustering. In this case, change the **Place** in **Hierarchical clustering** widget from **Class attribute** to **Attribute**. The ID of the cluster will now be added to instances as an ordinary attribute, not the class. By observing the distribution of iris types (the original class) within each cluster, we can see how well the constructed groups match the classes. In other words, this shows how well can unsupervised learning function in problems for which we could also use supervised learning.

K-Means clustering

K-Means clustering splits the data into k clusters, where the parameter k is set manually. It always uses Euclidean distance (or a similar measure, but let's not go into that). It cannot offer any fancy dendrograms. It simply splits data into k groups and that's it.

The advantage of k-means clustering is that its clusters are often "better defined" (this is purely subjective, intuitive speaking) than those we get by hierarchical clustering. In a sense, if two objects in k-means belong to the same cluster, this "means more" than it two objects are in the same cluster when using hierarchical clustering.

→ Hands-on Exercise

Use the below schema to analyze how well do clusters by the k-Means clustering match the actual classes, like we did with hierarchical clustering before.



Note that the input to k-Means is not a matrix of distances between instances, but instances (examples) themselves.

The K-Means can propose the number of clusters, or we can set it manually. Try setting it to 3 and see how it works. Then let the widget determine the optimal number of clustering itself. If it proposed two, check the scatter plots on the first page of this text. Yes, that's it. See one of the basic problems of clustering (and unsupervised learning in general)?

Multidimensional scaling

Clustering may sometimes seem too rigid: an object belongs to one and only one cluster. Especially in hierarchical clustering (but also in k-means) it can often happen that an object is actually more similar to objects from another cluster than to objects from its cluster. (This seems wrong, but if you think about it, you'll discover it's true and unavoidable.)

the clusters obvious to the user, and this is often enough.

Multidimensional scaling uses an optimization method which can get "stuck" in a certain placement. When using it, you should repeat the layout optimization process for multiple times and select the result which you like the most, or even multiple maps, if they, for instance, show data from different interesting perspectives.

Visualization of networks

Networks are becoming a very popular model in genetics and also elsewhere. They consist of vertices (nodes), which represent objects, and are connected by edges. The network can be constructed in different ways. We can construct it from distances like those in clustering: we can connect two objects if they are sufficiently close to each other. Connection can also represent existence of a relation. For instance, we can connect *pairs of proteins* which interact with each other. Two *scientists* are connected if they co-authored at least one paper. Two *symptoms* can be connected, if they often co-occur (e.g. sore throat and fever). Two *airports* are connected if there is a direct flight between them. Two *hospitals* are connected, if patients are transferred between them.

Analysis of networks is somewhat similar to observing the multidimensional scaling map (moreover, the technique used for optimizing the layout is similar, too). We can find "islands" of mutually interrelated objects (in graph theory, these are called *cliques*) – groups of collaborating scientists, collaborating hospitals, related symptoms...

We can do more than that, though: we can select a bunch of objects based on their properties and then tell the tool we used (say a Network explorer widget in Orange), to select and show only their vicinity, e.g. the network up to, say, their second-degree neighbor (nodes which are at most two edges away from the selected nodes). In genetics, we can select a subset of genes for which it is known that they play a role in a certain hereditary disease and then observe the genes related to the genes related to these genes in hope that some of these may be related to the disease, too.

We can find the *hubs*, of the network: the nodes which have significantly larger number of connections than other nodes and thus hold the network together. In genetics, these may be some general-purpose genes, e.g. genes corresponding to some common proteins. In social networks, these may be the most influential members of the society. In another context, these nodes may represent Al Qaeda leaders...

A small collection of interesting networks is published on the web site.

Evaluation of model performance

Last time, we discussed supervised and unsupervised modeling, which give us various models describing the data. Today we will learn how to measure the quality of the obtained models, either to choose the optimal model or to check the quality of the model to decide whether it is good enough to be used.

In the context of unsupervised modeling, like clustering, the quality of models is difficult to evaluate. In most cases we can only evaluate them subjectively by checking whether they make sense, whether they conform to our expectations and look useful for whatever we want to use them. Supervised learning, however, gives models which make predictions, so their quality can be measured objectively by observing the quality of predictions. In today's lecture we'll learn how to prepare the test data and discuss different measures of quality of models.

Sampling the data for testing

There is a very simple rule: *never test the model on the data which was used for its construction*.

The purpose of assessing the performance of models is to determine how well the model will behave if used in practice. Since the model will be used on new, yet unseen data instances (patients...), it also has to be tested on data which it has not seen before. Otherwise the best model would be simply one which remembers all data and uses that knowledge when making predictions (or, better, "postdictions").

For this purpose, we use resampling techniques in which only a (randomly) chosen subset of data is used for constructing the model and the remaining part is used to compute its quality. This is repeated multiple times and in the end we report the average quality of the model over all trials.

The most common technique used is called **cross-validation**. It splits the data into (usually) ten subsets, called *folds*. In the first step, it reserves the first fold for testing: it uses data from folds 2-10 for constructing the model and then tests it on the first fold. Then it reserves the second fold for testing; folds 1 and 2-9 are used for constructing the model and fold 2 for testing it. And so on; at each step one fold is held out for testing and others are used for learning.

Another sampling method is called **leave-one-out** or **jackknife**. Instead of leaving out one fold (10% of examples, when we do a 10-fold cross validation), it leaves out only one example at a time. It constructs a model from all examples except this one, and tests the prediction on this example. This gives very reliable results, but takes a lot of time: if the data has 1000 instances, it will construct 1000 models.

In **repeated random sampling**, we choose a given number of instances (say 70%) for constructing the model, use the remaining 30% for testing, and repeat this whole procedure a certain number of times.

Besides these three, which are also available in Orange, there exist a number of other resampling methods, which are nicely described here: [http://en.wikipedia.org/wiki/Resampling_\(statistics\)](http://en.wikipedia.org/wiki/Resampling_(statistics)).

A small problem with all these methods is that they give an average performance of constructed models and not of a single model. Say that we decide that the best model for our data is naïve Bayesian classifier, which has an AUC of 0.90 (we will soon see what's AUC, don't worry) – based on an average of the ten-fold cross validation. Having decided, we use the entire data set to construct one single naïve Bayesian model. We however do not know the expected AUC of *this* model. We only know that naïve Bayesian models for this data have average accuracies of 0.90, while this model may be better – or worse.

We usually do not worry about this. First, the final model was constructed from a bit larger data set – from all 10 folds and not just 9 of them. More data usually means better models. Second, most models are quite stable (classification trees excluded!). All ten naïve Bayesian models from cross validation were probably quite similar to each other and to this final model.

There is, though, one bigger problem. Constructing the model usually involves a lot of manual work – everything which we covered in our past lectures. When we get the data, we will use various visualizations and other methods to select the features which we want to use in our models, we may construct new features from existing models, we will fit various thresholds, parameters of learning algorithms (e.g. we will set the desired size of the trees). Although we may have used some resampling technique in the end, the entire data sets was used when preparing for the modeling – which is a part of modeling, too, and should not be done on the data on which the model is later tested.

This is the reason why we stressed in the first and in the second lecture that a part of the data needs to be left aside. As a rule of a thumb, we usually randomly select 30% of data and put it in a separate file. Then we go through everything we (hopefully) learned during these lectures using the remaining 70 % of data. When we decide for the final model, we take the held-out data and use it to test the model. These results are what we report as the true expected performance of our model.

Measures of model performance

There are multiple perspectives from which we can judge models with regard to the quality of their predictions.

Scalar measures of quality

Classification accuracy (CA) is the easiest and the most intuitive measure of quality. It measures the proportion of correct answers the model gives: if it makes the correct prediction for 85 cases out of 100, we say the classification accuracy is 85%.

Let us measure the accuracy of the naïve Bayesian classifier, logistic regression and classification trees on the heart disease data set. The central widget in today's lecture is called Test Learners. On its input it requires a data set (we will provide one from the file widget) and learning algorithms. The schema below

looks a bit unusual: last week, we gave data to learning algorithms, so that they could construct models from it. How do they construct models now, without data?

Widget for naïve Bayesian classifier has two outputs. When given data (as last week), it outputs a model (which we fed to, for instance, the widget showing nomograms). But even when not given any data, it outputs a *learning algorithm*. Learning algorithm is not a model, but a procedure for building the model. So, what Test Learners gets in this schema are not data and three models but data and three *procedures for making models*. Test Learners splits the data in different ways (cross validation or leave-one-out or ...), repeatedly runs the provided procedures on one subset and tests the resulting models on the other.

	Method	CA	Sens	Spec	AUC	Brier
1	Naive Bayes	0.8316	0.7986	0.8598	0.8968	0.2702
2	Classification Tree	0.8218	0.7698	0.8659	0.8396	0.2921
3	Logistic regression	0.8348	0.7914	0.8720	0.8973	0.2531

→ Hands-on exercise

Construct the above schema.

Classification accuracy does not distinguish between the "meanings" of classes. Not all mistakes are equal. For instance, failing to detect a disease can cause a death of a patient, while diagnosing a healthy person as sick is usually less fateful. To distinguish between different types of wrong predictions, we will select one of the classes and call it the *target class* or the *positive class*. In medicine, *positive* will usually mean having a disease, a condition or something like that. Now we can distinguish between four cases of correct and wrong predictions

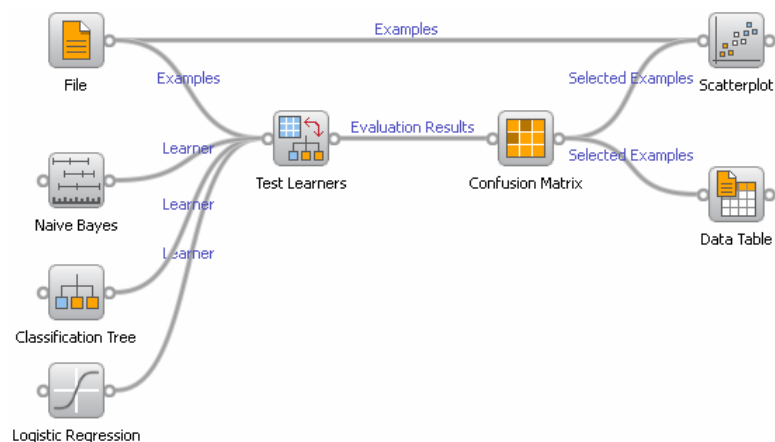
- **true positive (TP)** denotes a data instance (e.g. a patient) for whom the model made a positive prediction and the prediction is correct (so the patient is truly positive)
- **false positive (FP)** denotes instances which were falsely classified as positive (e.g. we predict the patient to have the disease, though he does not)
- **true negative (TN)** instances are those for which the model gave negative prediction and the prediction is correct
- **false negative (FN)** represents instances who are falsely classified as negative (they are sick but the model predicted them to be healthy)

		predicted value	
		positive (P')	negative (N')
actual value	positive (P)	true positive (TP)	false negative (FN)
	negative (N)	false positive (FP)	true negative (TN)

Besides TP, FN, FP and TN, we will also use P and N to denote the number of positive and negative *instances*, and P' and N' for the number of all positive and negative *predictions*. Obviously, $P=TP+FN$, $N=TN+FP$, $P'=TP+FP$ and $N'=TN+FN$. The matrix like this one is a special case of *confusion matrix*. Orange shows in it a widget called Confusion matrix, which we connect to the Test Learners widget.

→ **Hands-on exercise**

To make things interesting, we will also add a Data table widget and a Scatter plot widget, to which we give examples from the File widget and from the Confusion matrix. The latter should represent a subset of data (the easiest way to ensure this if to draw the connection between the File widget and Scatter plot first, and then connect the Confusion matrix to the Scatter Plot). If you now open the Confusion matrix, you can observe what kinds of mistakes the three models made (you can choose between the models on the left-hand side). Clicking the cell in the matrix selects and deselects the corresponding instances. You can therefore observe, say in the Data table and in the Scatter plot, the instances for which a particular model made a particular kind of mistake.



Side note: throughout our lectures we have used Data table and Scatter plot to observe subsets of instances selected by various means, like in the Confusion matrix above. You can of course use other widgets as well. The output of Confusion matrix can be connected to any widget that accepts data sets on its input.

Notice also the different types of signals being sent in this schema. The three modeling widgets output modeling algorithms (Learners) and the Test Learner outputs results of evaluations, while other signals transmit data sets. You cannot connect, say, Data Table directly to Test Learners because Data Table accepts data instances while Test Learners only gives results of evaluation (in a form understood by the Confusion Matrix).

Having defined TP, TN, FP and FN, we can now express classification accuracy in terms of these values. Classification accuracy is the number of correct predictions (that is, true positive and true negative predictions) divided by the number of all predictions made by the model:

$$\text{classification accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

As noted above, in medicine we are interested in different kinds of mistakes and thus define a number of related measures.

A measure which may be more important than classification accuracy is this: what proportion of sick person will the classifier recognize as sick? This is called **sensitivity** of the model. For example, if sensitivity equals 90%, this would mean that the method fails to detect the disease in 10% of cases. Sensitivity is computed as the quotient between the number of patients which the model correctly classified as sick and the number of those who were actually sick:

$$\text{sensitivity} = \frac{TP}{P} = \frac{TP}{TP + FN}$$

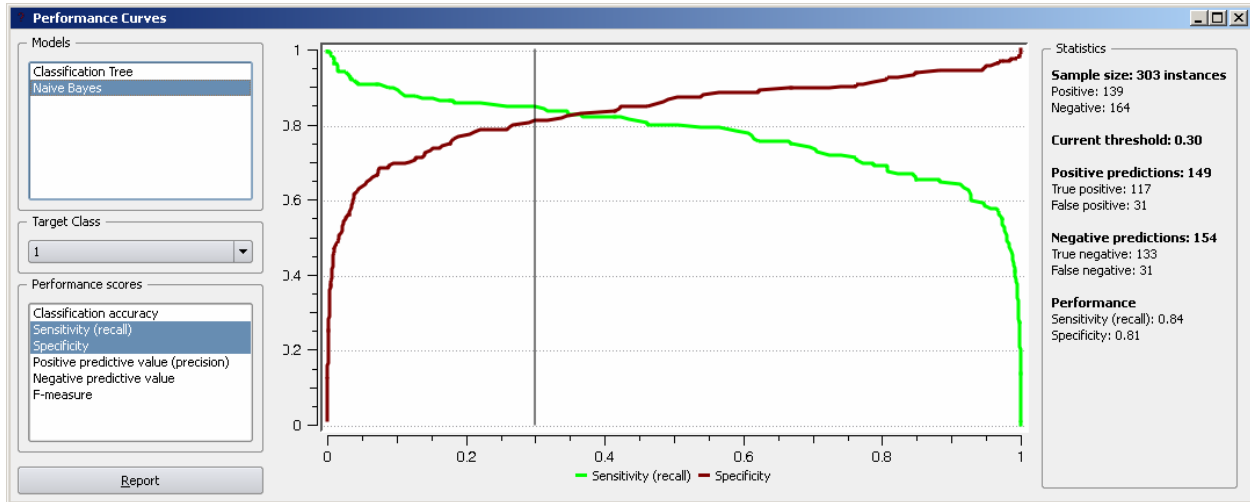
Now suppose that our model has a sensitivity of only 80 % and we think this is too small. We can increase it by lowering the classification threshold: instead of predicting the disease when its probability is higher than 50 %, we can predict the patient to be sick if his probability of having the disease is above 40 % or even 30 %. Is this good? Well... if we decrease the probability threshold to 0 %, that is, simply predict that everybody is sick, we get 100 % specificity. The problem is, obviously, that by lowering the threshold, we will have more and more false positives. This is expressed in another measure.

Specificity is the proportion of negative (healthy) instances which were recognized as negative (healthy).

$$\text{specificity} = \frac{TN}{N} = \frac{TN}{TN + FP}$$

The "paranoid" model with 100 % specificity would have 0 % sensitivity. There is always a trade-off: you can be cautious and have a higher sensitivity at a cost of specificity (miss less sick people at the cost of scaring the healthy) or higher specificity at a cost of sensitivity (less scaring of healthy people at a cost of

not detecting the sick). The following graph shows the sensitivity and specificity at different thresholds for naïve Bayesian classifier on the heart disease data.

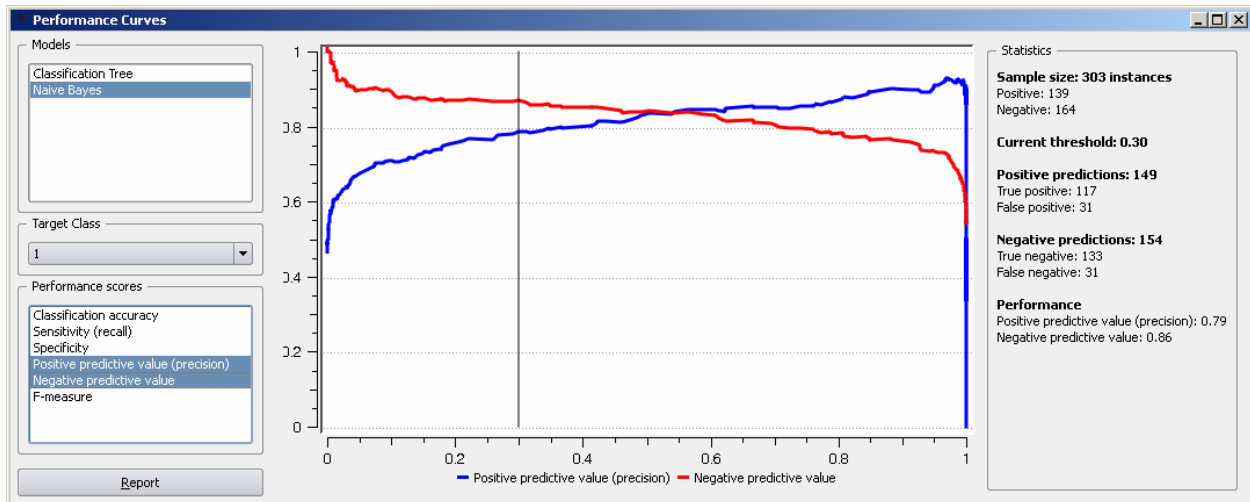


Positive predictive value (PPV) measures the reliability of positive predictions. If the model says that somebody is sick, how sure can we be that the person is really sick? In other words, what is the proportion of the sick among all patients whom the model predicted to be sick? (Note the difference between PPV and sensitivity, for which we would have "among all who are really sick".) The formula for computing this is

$$\text{positive predictive value} = \frac{TP}{P'} = \frac{TP}{TP + FP}$$

Negative predictive value (NPV) is the PPV's counter part: the proportion of negative instances among those whom the model classified as negative.

$$\text{negative predictive value} = \frac{TN}{N'} = \frac{TN}{TN + FN}$$



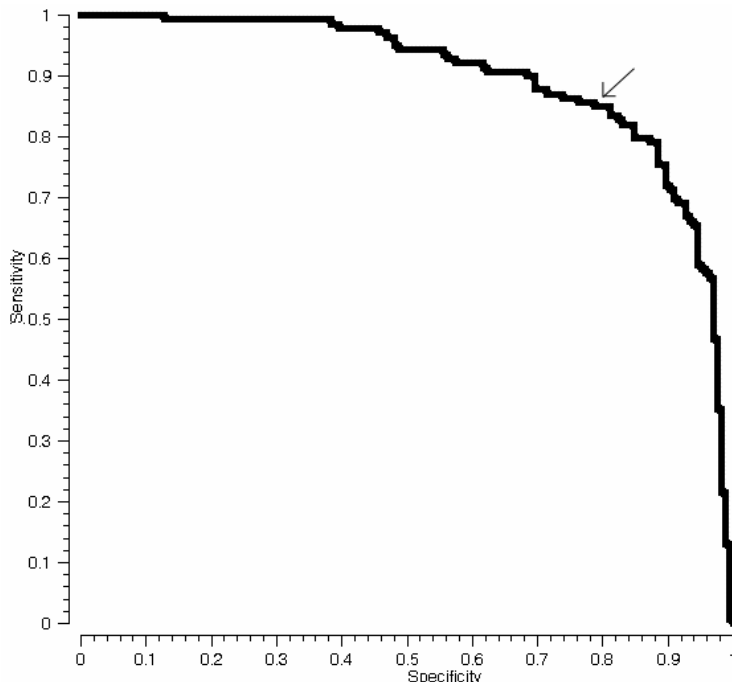
We can obviously turn these four numbers – TP, TN, FP and FN – around in other ways and get some more measures, but for these lectures let us stop here.

To recapitulate, these here the scores we have talked about.

name	meaning
classification accuracy	probability of making the correct prediction
sensitivity	probability that a positive instance is classified as positive (given a sick person, we detect that he is sick)
specificity	probability that a negative instance is classified as negative (given a healthy person, we correctly predict he's healthy)
positive predictive value	probability that the instance classified as positive is indeed positive (if model classifies somebody as sick, what's the chance that he really is)
negative predictive value	probability that the instance classified as negative is indeed negative (if model classifies somebody as healthy, what's the chance that he really is)

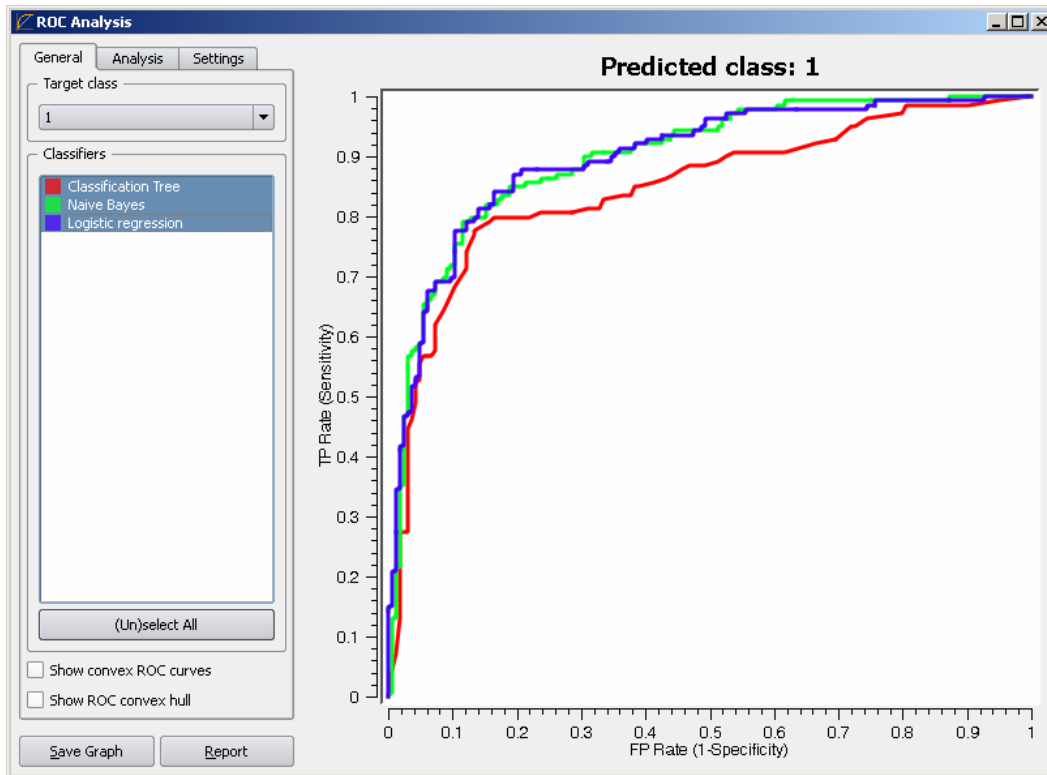
Receiver-Operating Characteristic (ROC) Curve

Let us now return to the relation between specificity and sensitivity. The two curves which show the relation between them at different thresholds are usually plotted as a single curve called **receiver-operating characteristic curve** or, shorter, **ROC Curve**. The curve plots the sensitivity of the model as a function of its specificity. (One can imagine going through the plot we've shown above, from the left to the right; at each probability threshold we read the corresponding specificity and sensitivity and add the point to the ROC curve.)



At a low threshold, we have high sensitivity and low specificity. At the extreme, if we classify all people as sick, we have 100 % sensitivity and 0 % specificity (the left-top-most point). As we increase the threshold, the specificity increases and the sensitivity decreases. For instance, at the point marked with the arrow, the specificity is at around 80 % and sensitivity is 85 %. At the right most-point, at the bottom, we classify all as healthy and have a sensitivity of 0 %, but a specificity of 100 %.

Unfortunately, what we did above is a bit of simplification for easier explanation. Instead of showing the specificity, ROC curves plot 1–specificity, which horizontally flips the above curve.



The widget shows the ROC curves for our three classifiers. To find the specificity of 80%, we look at the point 0.2 (since the graph shows 1-specificity). At that specificity, the classification tree has a sensitivity of around 80% and the logistic regression and naïve Bayesian classifier have a sensitivity of around 85%. Actually, as the curves show, logistic regression and naïve Bayesian models for this data always give us higher sensitivity than classification trees (or the opposite: higher specificity for a chosen sensitivity).

The optimal curve would go straight up from 0, into the upper left corner and then straight right. Such a model would have 100% sensitivity at 100% specificity. And what is the worst (practical) case? Imagine a classifier which knows nothing and always predicts the same probability (say 50%, but any other number would give the same result) of having the disease. The ROC curve for such a model would go diagonally from the lower left to the upper right corner.

ROC curves are the single most common and recognized method for assessing the model quality in medicine. First, they can help us fit the probability threshold which gives the desired sensitivity-to-specificity ratio for the given predictive model. Second, they help us to compare models. Even if a paper in a medical journal which proposes a new predictive model for a certain disease does not show a ROC curve, it will almost certainly give a numerical assessment computed from the ROC curve, the **area under curve** or **AUC**. (This also indicates the importance of the ROC curve: the measure is usually called *area under curve* not *area under ROC curve*. No need to add ROC, we all know which curve is *the* curve.)

As its name tells, AUC measure the area under ROC curve. The ideal curve, which goes straight up and then right, would have an AUC of 1.0, since all the area of the plot is under the curve. The worst possible curve, the diagonal, has an AUC of 0.5. Real-life predictive models are in between: an AUC of 0.7 is so-so, 0.8 is usually publishable and 0.9 is really good.

AUC has also a probabilistic interpretation. Let us say that the predictive model is given two persons, one is sick and the other is not (but we do not know which is which). If the model has to decide which of these two persons is more likely to be sick, AUC measures the probability that the model will get it right. For instance, we have two persons coming to the hospital and only one available bed. We will keep the person for which the model predicts the higher probability of being sick. (Even if the classifier gives us 90% for one person and 88% for the other, we will have to send the second one home – we don't have a bed for him. Similarly, if the first one has a 5% probability of being sick and the second one a 3 % probability, we will still keep the first one, although he is probably OK.) AUC measures the probability that the model will make the correct decision.

(For those more versed in statistics: AUC is equivalent to the U-statistics from Wilcoxon-Mann-Whitney's test, which measures the probability that positive instances are ranked higher than the negative.)

AUC has a number of advantages in comparison with other measures.

For classification accuracy, it is difficult to say what is a *good* classification accuracy. 95% accuracy may be perfect if we want to predict the toss of a coin. On the other hand, imagine a rare disease: 99 % of those who came to the physician do not have it. In such cases, you can reach a 99 % classification accuracy by simply answering No all the time.

You cannot fool the AUC in this way. Classification accuracy (as well as most other measures we got to know today) has little meaning on highly imbalanced data sets, that is, data sets in which examples of one class represent a huge majority. AUC has no problems with that and the rough guide lines – 0.7 is so-so and 0.9 is really good – are not affected by imbalanced data.

While other measures observe models from different perspectives, AUC is all-encompassing in the sense that for a model with a good AUC we can find a threshold that will give good classification accuracy, good sensitivity, good specificity... or a reasonably good combination of all these measures.

Classification cost

Every decision has a certain cost. Let us assume we know the cost of a false positive prediction (say the cost of further tests or unneeded treatments) and the cost of false negative prediction (the cost of not treating somebody who needs a treatment). We will denote the two numbers as c_{FP} and c_{FN} . We can then compute the cost of predictive model's decisions

$$\text{cost} = c_{FP} \frac{FP}{TP + TN + FP + FN} + c_{FN} \frac{FN}{TP + TN + FP + FN} = \frac{1}{TP + TN + FP + FN} (c_{FP}FP + c_{FN}FN)$$

The fractions represent the probabilities that an instance will be a false positive (the number of false positives among all decisions) and that it will be a false negative example. We multiply each probability with the associated cost to get the expected cost of misclassifications.

We want the "cheapest" classifier, so we look for a threshold with an optimal combination of false positives and false negatives. In this way, we are not forced to arbitrarily decide for a certain sensitivity and specificity, as in ROC analysis, since the optimal threshold can be uniquely determined from the cost analysis (assuming we know the costs).

For a more general scenario, we can also associate the cost to correct decisions, so the formula gets two additional terms

$$\text{cost} = \frac{1}{TP + TN + FP + FN} (c_{FP}FP + c_{FN}FN + c_{TP}TP + c_{TN}TN)$$

The costs need not be positive: correct decision can have "negative costs", that is, gains. Or sometimes the correct decisions bear actual cost, such as treating of a patient, which we may sometimes avoid. This may be especially true when the cost is not in terms of money; when other resources are scarce, say during catastrophes like major earthquakes, we might worry more about time, human resources or available drugs, so we can express the cost in terms of these resources being used (or wasted).

To conclude, there exist many numerical measures of quality of models, and it is the task of the field expert to determine which one to use and, if needed, what settings (probability thresholds etc) gave the optimal performance (sensitivity vs. specificity, positive vs. negative predictive value...). One should, however, never ignore the common sense. Even if a model seems to give excellent results, don't use it if it looks unreasonable and based on medically implausible relations (unless you can discover the meaning of these relations – maybe the model uncovered something which was hitherto unknown). Similarly, when deciding between two models with similar performance, pick the one you like more.

You are smarter than your computer.