Supervised statistical learnning

4 luglio 2024

Indice

Capitolo 1

Introduction

Important remark 1 (Consigli esame). Written test lasting 70 minutes consisting in 5-7 questions, both multiple choice and open, some of which to be solved in R. the final grade is out of thirty.orale per il +/- 3
The aim is to assess the student's ability to use the learned definitions, concepts and properties and to solve exercises.
During the written exam, students can only use the cheat sheet that is provided on virtuale.unibo.it, containing references to R packages and functions. No book/notes. Don't memorize. look cheatsheet. look help page.

1.1 Introduction to statistical learning

Definition 1.1.1 (Statistical learning)**.** A vast set of tools for understanding data of two types:

1. *unsupervised statistical learning* is used to describe the associations and patterns among a set of input measures (which are treated on the same level of importance);

2. *supervised statistical learning* (focus on the course) involves building a model for predicting an important/selected response variable based on one or more inputs

Important remark 2 (Supervised learning typical scenario). We have:

1. a **training dataset**, in which we observe both the **outcome** measure (*quantitative* in case of a *regression problem*, *qualitative* for a *classification* one) and **features** measurements that will be used for the prediction;

2. we build a a learner, which will enable us to predict the outcome for new unseen objects.

3. a **good learner** is one that accurately predicts the outcome on new observation

Important remark 3 (Truth bombs). Some remarks:

- there is no **single best method** for all the problems:

  - different *datasets* have different features and what is best for one problem is not best for another
  - different *tasks* (eg prediction/inference) are better tackled using different tools

  The user should search which is the best type of learner for the problem at hand; to do that must know what are the assumpion behind and search for the best

- **simple vs complex methods**: simple doesnt mean it works worst there is a *bias and variance tradeoff*, where more complex low bias solutions involve more variance in the results, which highly depends on the sample used

- to **get a good learner**: we want to find a learner that find a *balance in the tradeoff* between low bias and variance to minimize MSE in the test set and to do it tipically one *compare different methods/parametrization* developed in the training set on a new validation.

1.2 Estimating f and the bias variance tradeoff

Important remark 4 (General framework). We:

- observe a response Y and p different predictors, $X_1, \dots, X_p$
- assume that there is some relationship between Y and $\mathbf{X} = (X_1, \dots, X_p)$, which can be written in the very general form as

$$Y = f(\mathbf{X}) + \varepsilon$$

  where

  - f is some fixed but unknown function/relation between Y and $X_1, \dots, X_p$ in the population, and represents the systematic information that $\mathbf{X}$ provides about Y;
  - ε is a random error term (since relation in the population is not perfect), which is independent of $\mathbf{X}$ and has mean zero (generic, not necessary gaussian)

Important remark 5 (Notation). Here

1. x_{ij} will be the value of the j-th predictor (or input) for observation i, where $i = 1, \dots, n$ and $j = 1, \dots, p$;
2. y_i will be the response variable for the i-th observation.

So the *training data* consist of $\{(x_1, y_1), \dots, (x_n, y_n)\}$ where $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$

1.2.1 Reason for estimating f

Important remark 6 (Reasons to estimate f). Two main reasons:

1. **prediction**: in many situations, data on a set of inputs $\mathbf{X}$ are available, but the *output* Y *cannot be easily obtained* (es future/difficult/costly). In this setting, since the error term averages to zero, we can build a model to predict Y using

$$\hat{Y} = \hat{f}(\mathbf{X})$$

   where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y.
   In this setting, $\hat{f}$ is often treated as a *black box*, (one is not typically concerned with the exact form of $\hat{f}$, provided that it yields accurate predictions for Y)

2. **inference**: we can estimate f to study the relation between X and Y that is to understand how Y changes as a function of $X_1, \dots, X_p$.
   Here $\hat{f}$ cannot be treated as a black box, because we need to know its exact form

1.2.2 Methods for estimating f

Remark 1 (Aim and methods of estimating f). The proper methods used to estimate f may depend on our ultimate/main goal:

- if we want *inference*, for example, linear models allow for relatively simple and interpretable results, but may not yield as accurate predictions as some other approaches;
- for *prediction* otoh, some of the highly non-linear approaches can potentially provide quite accurate predictions for Y, but this comes at the expense of a less interpretable model for which inference is more challenging.

Important remark 7 (When to prefer simpler/restrictive methods). Why choose to use a more restrictive method instead of a very flexible approach?
Several reason for preferring a more restrictive model:

1. *low data* available: sometimes we can afford a complex model
2. if we are interested in *inference*, restrictive models (eg the linear model may) are *more interpretable* (eg flexible complex approaches can lead to such complicated estimates of f that it is difficult to understand how any individual predictor is associated with the response).
3. if we are interested in *prediction* not always a complex/flexible model is the best choice since decrease the bias but exposes to *overfitting* and higher variance in the estimation (thus going with bad performance on not-training sample since in the training phase we followed too much the training sample noise)

Remark 2. In the following we highlight the two main methods of estimation: parametric and non-parametric methods

1.2.2.1 Parametric methods

Definition 1.2.1 (Parametric method)**.** The approach described is **parametric** when make explicit assumptions about the functional form of f and reduces the problem of estimating f down to estimating a set of parameters.

Important remark 8 (Parametric howto). Parametric methods involves a two-step model-based approach:

1. First, we make an assumption about the functional form, or shape, of f . For example, that f is linear in X:

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p.$$

2. once assumed that f is linear, rather than estimating an entirely arbitrary p-dimensional function $f(X)$, one only needs to estimate the $p + 1$ coefficients $\beta_0, \beta_1, \ldots, \beta_p$. We need a procedure that uses the training data to fit or train the model (eg ordinary least squares, maximum likelihood, etc)

Important remark 9 (Pros/cons). Parametric estimation is

- much **simpler** in terms of *number of parameters*, *estimation method* and *interpretation*;
- **less demanding** in terms of data
- being less complex in terms of number of estimated parameters, results, variance is lower and results are generally **more stable** (to new samples);

However

- it's *less general*: being based on an assumption, we may "choose" a model which is very different from f (so our estimate and prediction will be poor/biased)

1.2.2.2 Nonparametric methods

Definition 1.2.2. Nonparametric methods do not make explicit assumptions about the functional form of f.

Important remark 10 (Nonparametric howto). They seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly.

Important remark 11 (Pros/cons). Nonparametric methods are:

- *more general*: by avoiding the assumption of a particular functional form for f they have the potential to accurately fit a wider range of possible shapes for f, so they avoid the danger of a resulting model that does not fit the data well;
- *more demanding*: a very large number of observations (far more than is typically needed for a parametric approach) is required in order to obtain an accurate estimate for f.

1.2.3 Assessing Model Accuracy

Remark 3. It is important to decide for any given dataset which method produces the best results; there are two type error which can be computed

Definition 1.2.3 (MSE (training))**.** Is computed (and actually minimized during model building) in training dataset as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \hat{f}(x_i)\Big)^2 \tag{1.1}$$

where (y_i, x_i) are observation of the training set, $\hat{f}(x_i)$ is the prediction that $\hat{f}$ gives for the i-th observation.

Remark 4. It will be small if the predicted responses are very close to the true responses, and will be large if for some of the observations, the predicted and true responses differ substantially.

Important remark 12. We are more interested in knowing whether $\hat{f}(x_0)$ is approximately equal to y_0 , where (x_0, y_0) is *a previously unseen test observation* not used to train the statistical learning method.

Definition 1.2.4 (test MSE)**.** It's defined as the previous MSE but evaluated on the test dataset

$$Ave\Big((y_0 - \hat{f}(x_0))^2\Big) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_0 - \hat{f}(x_0)\Big)^2 \tag{1.2}$$

which is defined as average squared prediction error for these test observations (x_0, y_0).

Important remark 13 (Choosing the method with best test MSE). We want to choose the method/model that gives the **lowest test MSE**: we avoid choosing on training one because

- we would have no guarantee that such method will also have the lowest test MSE;
- this strategy could lead us to more complex models (with low train MSE) which perform bad (high test MSE) on new observation due to *overfitting*.

Important remark 14 (Overfitting: what is, errors). It occurs when a too complex model is fit, and pick up some patterns that are just caused by random chance/sample rather than by true properties of the unknown function f.
When this occurs train MSE will be low but test MSE will be very large because the supposed patterns that the method found in the training data simply don't exist in the test data.

Important remark 15 (Train/test MSE: which is smaller). Regardless of whether or not overfitting has occurred, almost always the training MSE is smaller than the test one since most statistical methods seek to minimize the training MSE (eg OLS).

1.2.4 The bias-variance tradeoff

Proposition 1.2.1. *The expected test MSE* [1]*:*

$$\mathbb{E}\left[\left(y_0 - \hat{f}(x_0)\right)^2\right] = \underbrace{\text{Bias}\left(\hat{f}(x_0)\right)^2 + \text{Var}\left[\hat{f}(x_0)\right]}_{reducible} + \underbrace{\text{Var}\left[\varepsilon\right]}_{irreducible} \tag{1.3}$$

where y_0 is the random variable representing one point we want to predict in the test set, $x_0 = (x_{01}, \ldots, x_{0p})$ is its set of p predictors/covariates in the test set, $\hat{f}$ is the predictor we trained in a separate set of data to best approximate f, $\hat{f}(x_0)$ our prediction for y_0.

Dimostrazione. Being $y_0 = f(x_0) + \varepsilon$ we have that

$$\mathbb{E}\left[(y_0 - \hat{f}(x_0))^2\right] = \mathbb{E}\left[(f(x_0) + \varepsilon - \hat{f}(x_0))^2\right]$$

Now $f(x_0)$ is the true model, it's deterministic, so $\mathbb{E}\left[f(x_0)\right] = f(x_0)$; furthermore we ease the notation a bit ($f(x_0) \to f$, $\hat{f}(x_0) \to \hat{f}$) and credo $\hat{f}, f, \varepsilon$ are independent. Assume for a moment that both $\hat{f}$ and x_0 are fixed. Thus we can rewrite:

$$\begin{aligned}
&\mathbb{E}\left[(f + \varepsilon + \hat{f})^2\right] \\
&= \mathbb{E}\left[\left(f + \varepsilon - \hat{f} + \mathbb{E}\left[\hat{f}\right] - \mathbb{E}\left[\hat{f}\right]\right)^2\right] \\
&= \mathbb{E}\left[\left(\left(f - \mathbb{E}\left[\hat{f}\right]\right) + \varepsilon + \left(\mathbb{E}\left[\hat{f}\right] - f\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)^2 + \varepsilon^2 + \left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)^2 + 2\left(f - \mathbb{E}\left[\hat{f}\right]\right)\varepsilon + 2\left(f - \mathbb{E}\left[\hat{f}\right]\right)\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right) + 2\varepsilon\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)\right. \\
&= \underbrace{\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)^2\right]}_{\text{Bias}\left(\hat{f}\right)^2} + \underbrace{\mathbb{E}\left[\varepsilon^2\right]}_{\text{Var}[\varepsilon]} + \underbrace{\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)^2\right]}_{\text{Var}\left[\hat{f}\right]} + \ldots \\
&\quad \ldots + 2\,\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)\right]\underbrace{\mathbb{E}\left[\varepsilon\right]}_{=0} + 2\,\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)\right]\underbrace{\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)\right]}_{=0} + 2\underbrace{\mathbb{E}\left[\varepsilon\right]}_{=0}\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)\right] \\
&= \underbrace{\text{Bias}\left(\hat{f}\right)^2 + \text{Var}\left[\hat{f}\right]}_{\text{reducible}} + \underbrace{\text{Var}\left[\varepsilon\right]}_{\text{irreducible}}
\end{aligned}$$

In the last step:

- all the double products disappears for different reasons: one is that $\mathbb{E}\left[\varepsilon\right] = 0$, the other is since the property of the mean (the average of the differences from the mean is always zero)
- the first remained term is the bias of the estimator $\hat{f}$, which is the first part of the reducible error

[1] The average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0)

- the last remaining term is the variance of the estimator $\hat{f}$

$$\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right]-\hat{f}\right)^2\right]=\mathbb{E}\left[\left(\hat{f}-\mathbb{E}\left[\hat{f}\right]\right)^2\right]=\mathrm{Var}\left[\hat{f}\right]$$

  and constitutes the second part of the reducible error;

- finally we have that $\mathbb{E}\left[\varepsilon^2\right]=\mathrm{Var}\left[\varepsilon\right]$ since $\mathbb{E}\left[\varepsilon\right]=0$, and $\mathrm{Var}\left[\varepsilon\right]$ is the irreducible error.

□

Important remark 16 (Commento alle componenti). In 1.3 the components of the test error are:

1. the **reducible error**: $\hat{f}$ will not be a perfect estimate for f, and this inaccuracy will introduce some error. This error is reducible because we can potentially improve the accuracy of $\hat{f}$ by using the most appropriate statistical learning technique to estimate f.
   The reducible error is composed by
   (a) the squared **bias** of the estimator refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model
   Generally, more flexible/complex methods result in less bias
   (b) the **variance** of the estimator refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. Ideally the estimate for f should not vary too much between training sets.
   In general, more flexible/complex statistical methods have higher variance
2. the **irreducible error**: error due to ε (it is its variance). The prediction $y_0=f(x_0)+\varepsilon$ is a function of ε and thus its variability will also affects the accuracy of our predictions. This error is irreducible, because no matter how well we estimate f, we cannot reduce the error introduced by ε (containing unmeasured variables that are useful in predicting Y).
   The expected test MSE can never lie below $\mathrm{Var}\left[\varepsilon\right]$ the irreducible error.

Important remark 17 (The bias-variance tradeoff). In the reducible error part we have the bias-variance tradeoff:

- the more flexible a model is the less is the bias (we're approximating better) but chances are our estimator is very variable (approximating to much at data)
- simpler methods are more biased but more stable

Remark 5 (optimizing complexity). In order to optimize the test error, we need to increase flexibility at an optimal level:

- as we increase the flexibility of a class of methods, the bias tends to initially decrease faster than the variance increases. Consequently, the expected test MSE declines;
- however, at some point increasing flexibility has little impact on the bias but starts to significantly increase the variance. When this happens the test MSE increases.

1.3 Evaluating MSE

Important remark 18. To select the best method we need to calculate test MSE and select the learning method with the smallest one:

- if **we have a separate test dataset** we can evaluate the test MSE there;
- if **no separate test dataset** is available a number of techniques can be used using the available training data:
  - some *methods adjust the training error rate* in order to estimate the test error rate (e.g. Mallow's C_p);
  - we consider a class of methods that estimate the test error rate by *holding out a subset of the training observations* from the fitting process, and then applying the statistical learning method to those held out observations: these are **validation set approach** and **resampling methods**

1.3.1 Validation set approach (VSA)

Definition 1.3.1 (Validation set approach)**.** We randomly divide the available set of observations into two parts, a *training set* and a *validation set*;

- the model is fit on the training set,
- the fitted model is used to predict the outcome in the validation set;
- the resulting validation set error rate provides an estimate of the test error rate.

Important remark 19 (Pros/cons). This approach is conceptually simple and easy to implement. But:

1. only *a subset of the observations* are used to fit the model; since statistical methods tend to perform worse when trained on fewer observations, the validation set error rate may tend to overestimate the test error rate we'd obtained using the entire training data set;
2. the estimate of the test error rate can be highly variable depending on the splitting performed (which observations are in the training set and which in the validation set).

Important remark 20 (Three way splitting for data-rich model selections). If we are in *data-rich* situation and there is a *learner selection* problem, the best approach would be to randomly divide the dataset into *three* parts:

- a *training* set (eg 50%): used to fit the competing models;
- a *validation* set (eg 25%): used for model selection by comparing prediction errors;
- a *test* set (eg 25%): used to obtain a final error estimate for the chosen model in a fresh dataset.

Otherwise, splitting in two developing on the *training* and using the *test* set both for choosing the best model and keeping the same error as final error estimate, would make the chosen model final test error estimate *underestimating the true test error*, sometimes substantially. So we estimate the error in a pristine dataset (the test).

1.3.2 Resampling method

Definition 1.3.2 (Resampling methods)**.** Involve repeatedly drawing samples from a training set, fitting a model of interest on each sample and test it on the remaining observation.

Remark 6. These methods can be computationally expensive but permits to assess a method/parametrization test MSE and compare different solutions. Cross validation is the main method: two main types

1.3.2.1 Leave-one-out cross validation (LOOCV)

Definition 1.3.3 (LOOCV)**.** For each observation of the training set, say $i = 1$,

- we split the sample considering the single observation (x_1, y_1) composes the *validation set*; the remaining observations (x_2, y_2), ..., (x_n, y_n) make up the *training set*;
- the model trained on the training set is used to predict $\hat{y}_1$ in the validation set;
- the MSE for the single observation is calculated according to

  $$MSE_1 = (y_1 - \hat{y}_1)^2$$

  now even if MSE_1 is unbiased for the test error, it is highly variable (based upon a single observation, (x_1, y_1)).

After performing the procedure for the n observation we'll have n errors estimates: the LOOCV estimate for the test MSE is their average

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Important remark 21 (Pros/cons vs validation set approach). Compared to VSA, LOOCV:

- is a *non-random*/deterministic procedure (performed multiple times will always yield the same results);
- *doesnt' overestimate the test error rate* (VSA should be performed with 3-way splitting);
- has *less bias using almost all the dataset* during training estimation ($n-1$ observations);

- has *correlated prediction* for different units (being developed on similar training set). Their mean (and thus the MSE error estimate) will have more *variability* (variance of the sum/mean has the covariance part as well);
- can be *time consuming* to implement if n is large.
  In case of *linear regression*, a *shortcut* (needing only a single model fit) is calculating the error this way:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2$$

  where the i-th residual is divided by $1 - h_i$ (before being exponentiated), where and h_i is the leverage statistic which can be found in the i-th place of the diagonal of the hat matrix $\mathbf{H}$.
  Remember hat matrix is the one that premultiplied to the outcome returns the prediction ($\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$):

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T,$$

1.3.2.2 k-fold cross-validation

Definition 1.3.4 (k-fold CV)**.** We randomly divide the observations into k (typically 5 or 10) non overlapping groups/folds of approximately equal size. Then for each fold i:

- the fold i is treated as a validation set;
- the method is fit on the remaining $k - 1$ folds;
- the MSE_i is then computed on the observations in the held-out fold.

The k-fold CV estimate is computed averaging the k MSE_i:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

Important remark 22. LOOCV is a special case of k-fold CV in which $k = n$.

Important remark 23 (k-fold vs LOOCV). Compared to LOOCV, k-fold with $k < n$:

- is *quicker* (less cycles/estimates);
- *often gives more accurate estimates* of the test error (sum up of more bias but less variance):
  - on one hand *estimates have more bias* (being the training set smaller):
    - LOOCV will give approximately unbiased estimates of the test error (each training set contains $n - 1$ observations, almost as the full data set);
    - k-fold will give to an intermediate level of bias (between LOOCV and and validation set), since each training set contains $(k - 1)n/k$ observations (fewer than in the LOOCV, but more than validation set approach).

  - otoh *prediction are less correlated*, since overlapping of the training sets is lower:
    * with LOOCV, we average the outputs of n fitted models, each of which is trained on an almost identical set of observations (therefore these outputs are highly positively correlated with each other);
    * with k-fold CV, we are averaging the outputs of k fitted models (that are somewhat less correlated with each other, since the overlap between the training sets in each model is smaller)

    Thus since the mean of less correlated quantities has lower variance, the *test error estimate from k-fold CV tends to have lower variance* than LOOCV.

- is *less applicable if the sample size is small*: here the only method is leave one out, especially with *classification* problems where all the output variable classes must appear in the training set;

1.3.2.3 CV error for classification

Important remark 24 (Evaluating error in CV for classification). When Y is qualitative the procedure is the same, we only change the metric used for evaluating error. Instead of MSE we use the expected number of misclassified observations.

Example 1.3.1. For instance, in the classification setting, the LOOCV error rate takes the form:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

where

$$I(y_i \neq \hat{y}_i) = \begin{cases} 1, & \text{if } y_i \neq \hat{y}_i \\ 0, & \text{if } y_i = \hat{y}_i \end{cases}$$

The k-fold CV and validation set error rates are defined analogously.

1.3.3 CV for model selection the right way

Remark 7. Consider a problem (say classification) with a large number of predictors, (eg genomic or proteomic applications) and we need to choose among them.

Remark 8 (The wrong way). A wrong typical approach is:

1. find a subset of good predictors with strong univariate correlation with the outcome in the whole sample;

2. use this subset of predictors in cross-validation to estimate eventual tuning parameters and the prediction error.

This is wrong since units was already used: leaving samples out after the variables have been selected does not correctly mimic the application of the classifier to a completely independent test set, since these predictors have already seen the left out samples.
Thus estimate of the test error will be biased (selecting the best variable at first will falsely low the error estimate in CV because we already optimized).

Important remark 25 (The right way). To do model selection using CV:

1. divide the samples into K cross-validation folds (groups).
2. for each fold $k = 1, 2, \ldots, K$:
   (a) find a subset of good predictors with fairly strong univariate correlation with the outcome (using all of the samples except those in fold k);
   (b) using just this subset of predictors, build a multivariate model, using again all of the samples except those in fold k;
   (c) use the model to predict the outcomee for the samples in fold k and calculate
3. calculate the mean CV error estimates by taking the mean from all the K folds

Capitolo 2

Classification

2.1 Classifier evaluation evaluation

Remark 9. Suppose that we have estimated f on the training observations (x_t, y_t) where the outcome y_t is *qualitative*. Now if we have *test* observation $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where the outcome $y_1, \ldots, y_n$.

2.1.1 General test error rate

To quantify accuracy of our estimate $\hat{f}$ is the error rate calculated in the test set:

$$Ave(I(y_i \neq \hat{y}_i)) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \tag{2.1}$$

where

- where $\hat{y}_i$ is the predicted class label that results from applying the classifier to the test observation with predictor x_i;
- $I(y_i \neq \hat{y}_i)$ is an indicator variable that equals 1 if $y_i \neq \hat{y}_i$ and zero if $y_i = \hat{y}_i$.

2.1.2 Other measures in binary classification

Important remark 26 (Confusion matrix). In a dichotomic case we can observe the **confusion matrix** in table 2.1 where:

- TP (true positives): positive units correctly labeled by the classifier;

	Predicted Yes	Predicted No	Tot
Actual Yes	TP	FN	P
Actual No	FP	TN	N
Total	P'	N'	P + N

Tabella 2.1: Confusion matrix

- TN (true negatives): negative units correctly labeled;
- FP (false positives): negative units incorrectly labeled as positive;
- FN (false negatives): positive units mislabeled as negative.

Several indexes can be defined using the quantities involved in the confusion matrix.

Definition 2.1.1 (Accuracy)**.** Percentage of test set units that are correctly classified by the classifier:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} \tag{2.2}$$

Definition 2.1.2 (Error rate)**.** Percentage of units that are wrongly classified by the classifier and coincides with what presented in the previous section.

$$\text{error rate} = \frac{\text{FP} + \text{FN}}{\text{P} + \text{N}} = 1 - \text{accuracy} \tag{2.3}$$

Definition 2.1.3 (Sensitivity/recall)**.** It's the true positive rate:

$$\text{sensitivity} = \frac{\text{TP}}{\text{P}} \tag{2.4}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}} \tag{2.5}$$

Definition 2.1.4 (Specificity)**.** It's true negative rate:

$$\text{sensitivity} = \frac{\text{TN}}{\text{N}} \tag{2.6}$$

Definition 2.1.5 (Precision (VPP))**.** Percentage of units labeled as positive by the classifier actually are:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Definition 2.1.6 (F measure (another overall measure))**.** An overall indicator which is the harmonic mean of precision and recall:

$$\text{F} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Important remark 27 (Class imbalance problem). Within a binary classification context, and especially in case of unbalanced groups it's important to consider not only overall measures (such as accuracy/error rate); infact when one class (tipically the main class of interest, eg positive class) is rare:

- looking statistics "overall" may hide a bad performance ...
- optimizing a classifier "overall" (many classifier minimize the overall error rate) would tend to privilege the classification on the most represented group and may result in a bad performance ...

in the class of interest.

Important remark 28 (Solutions). Thus one can

- present *couple of statistics* which focus on components separately such as sensitivity + specificity or precision (VPP) + recall (sensitivity)
- adopt classifier less sensitive to the imbalance (eg put larger *weights to smaller class*)
- *resample* from the minority class to increas its number (problem is that there's no much added variability, only rebalancing)
- *generate artificial data* for the minority class (eg generate points on the segment connecting two dots of the minority class, or using a kerne density function centered on each point of the minority class

2.2 Bayes classifier

Remark 10. Name come from the fact that it simply applies the bayes theorem to choose the population/group of classification.

Definition 2.2.1. The Bayes classifier calculates for each class/group h the posterior/conditional probabilities of class belonging $\mathbb{P}(Y = h|X = x_0)$ according to Bayes theorem:

$$\mathbb{P}(Y = h|X = x_0) = \frac{\pi_h f_h(x_0)}{\sum_{\ell=1}^{K} \pi_\ell f_\ell(x_0)}$$

where:

- π_h is the a priori probability of belonging to population h (eg percentuale nella popolazione)
- $f_h(x_0)$ is the probability of observing x_0 assuming it comes from population h (also known as likelihood);
- $\mathbb{P}(Y = h|X = x_0)$ is the posterior probability of belonging to population h given we have observed x_0

Then it assigns each observation to the most likely class given its predictor values x_0, that is it assign to the class h for which the conditional probability $\mathbb{P}(Y = h|X = x_0)$ is largest

Important remark 29 (Properties). The Bayes classifier:

- is the **gold standard classifier**: it's possible to show that the Bayes classifier produces the lowest possible test error rate (called *the Bayes error rate*) basically the unreducible error;
- is a **theoretical only classifier** that can't be used in practice (unless we are in a simulation scenario); problem in applying it is that with real data the information we need are usually unknown (i guess the likelihoods).
- is thus **useful as benchmark** for comparing with new methods in research

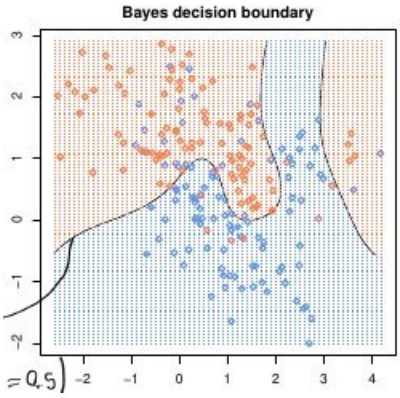
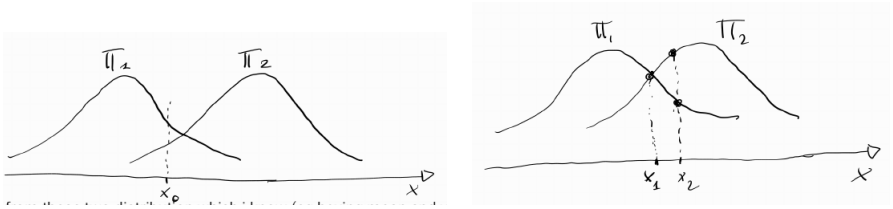


Figura 2.1: Bayes decision boundary.



(a) (b)

Figura 2.2: Bayes univariate

Example 2.2.1. In a two-class problem, say class 1 or class 2, the Bayes classifier corresponds to predicting class one if $\mathbb{P}(Y = 1|X = x_0) > 0.5$, and class two otherwise.

Remark 11. In case of rare population (π_h small) to assign to the class h there must be a convincing effect of the likelihood, otherwise the numerator will be small.

Definition 2.2.2 (Bayes error rate)**.** For a single given value $X = x_0$ the probability to correctly classify is $\max_h \mathbb{P}(Y = h|X = x_0)$ so the probability of a miss (the error rate at $X = x_0$) will be

$$1 - \max_h \mathbb{P}(Y = h|X = x_0)$$

Overall, the **Bayes error rate** is given by the complement to 1 of the average probability to hit the prevision/classification (for all the possible values of X)

$$1 - \mathbb{E}\left[\max_h \mathbb{P}(Y = h|X)\right],$$

Example 2.2.2. If the two population perfectly overlap, the error rate will be 0.5, while if the two population are perfectly separed the error will be 0.

Example 2.2.3 (Univariate example)**.** Suppose that we are in the univariate case (so we can visualize the stuff easily) and suppose our population are gaussian (can be anything). We have data coming from two gaussian population Π_1 and Π_2 (figure 2.2 a)). I can generate data from these two distribution which I know (eg having mean and variance). Thing is that these two population are not perfectly separated (there's an interval where data from both of two population could come).
In order to choose from which population a single x_1 value comes from i can apply the bayes theorem. Eg with respect to the first population i can calculate easily the probability that given its value, it comes from population 1:

$$\mathbb{P}(\Pi_1|X = x_0) = \frac{\mathbb{P}(\Pi_1) \cdot \mathbb{P}(X = X_0|\Pi_1)}{\mathbb{P}(\Pi_1) \cdot \mathbb{P}(X = X_0|\Pi_1) + \mathbb{P}(\Pi_2) \cdot \mathbb{P}(X = X_0|\Pi_2)}$$

same thing can be done with $\mathbb{P}(\Pi_2|X = x_0)$. Finally if the posterior probability

$$\mathbb{P}(\Pi_1|X = x_0) > \mathbb{P}(\Pi_2|X = x_0)$$

I choose to classify the x_0 in the first population.
This can be simplified considering the fact that all the decision depends on the numerator (denominator is common) and the fact that if Π_1 and Π_2 have the same prior $\mathbb{P}(\Pi_1) = \mathbb{P}(\Pi_2) = 0.5$ (a typical scenario) i can make the choice based only on likelihood (that are the density of the two normals in the same point. The rule will simply become

$$\mathbb{P}(X = x_0|\Pi_1) > \mathbb{P}(X = x_0|\Pi_2)$$

Despite being gold standard for classification (as can be shown) even the Bayes classifier make mistakes, since there are section of the real line with considerable overlap between the two distributions.
Eg in figure 2.2 (b), in x_1 the error rate will be approximately 0.5, having the two distribution the same height. Similarly in x_2 we will classify the point as coming from Π_2 (having higher likelihood), but it could be the case of a point coming from population 1 nonetheless. This is what the unreducible error is for the bayes classifier, we can avoid this error dued to overlapping of populations.
From the research point of view, this simulation procedure can be used to have an estimate of the lower bound of a classifier error (to be compared with the error of the model we propose for benchmarking). In other words a new classifier can (and should) be compared in simulation with a bayes classifier to have an idea of how good is it.

Example 2.2.4 (Simulated bivariate data: Bayes decision boundary)**.** In figure 2.1 a simulated data set in a two-dimensional space consisting of predictors X_1 and X_2 where the group is represented by the color of the dot (Y blue or orange). We can generate multivariate normal data with the `mvtnorm` package btw so it should be something like:

```
library(mvtnorm)
red <- rmvnorm(100, mu1, sigma1)
blue <- rmvnorm(100, mu2, sigma2)
# densities can be then obtained with dmvnorm function
```

For each value of X_1 and X_2 , there 3 is a different probability of the response being orange or blue: since this is simulated data, we know how the data were generated and we can calculate the conditional probabilities for each value of X_1 and X_2.
In the graph:

- the orange shaded region reflects the set of points for which $\mathbb{P}(Y = \text{orange}|X) > 0.5$: an observation that falls on the orange region of the boundary will be assigned to the orange class
- the blue shaded region indicates the set of points for which $\mathbb{P}(Y = \text{orange}|X) < 0.5$; an observation on the blue side of the boundary will be assigned to the blue class.
- the black line is called the **Bayes decision boundary** and in the boundary the posterior are 0.5 for each the two groups: $\mathbb{P}(Y = \text{orange}|X) = \mathbb{P}(Y = \text{blue}|X) = 0.5$

For the simulated data, the *Bayes error* rate is 0.1304: it is greater than zero, because the classes overlap in the true population so $max_h \mathbb{P}(Y = h|X = x_0) < 1$ for some values of x_0.

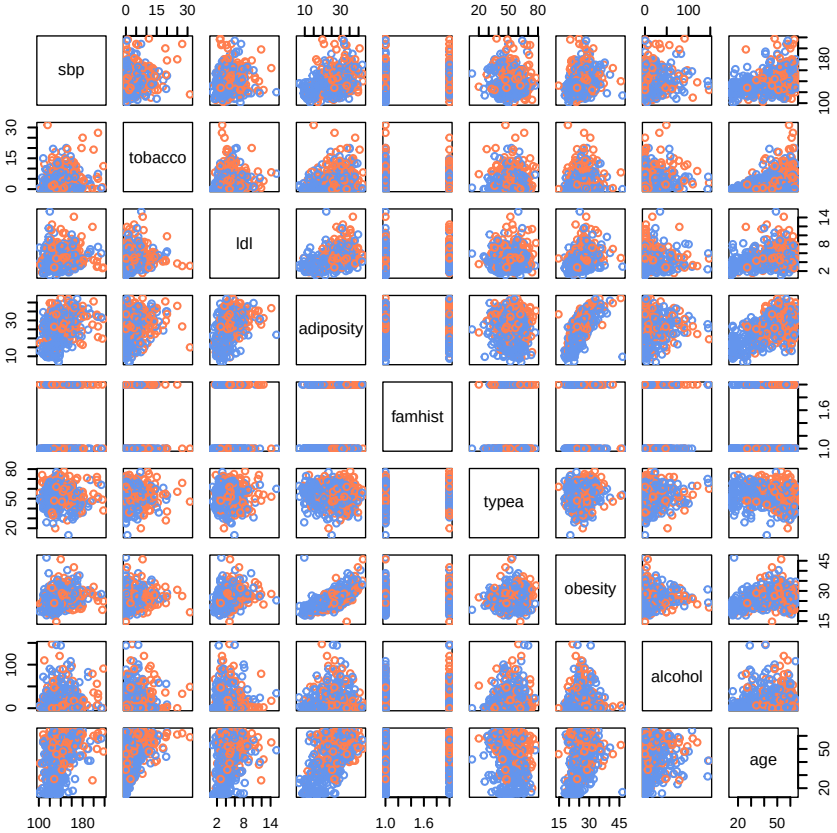
2.3 Classification dataset: `SAheart`

Example 2.3.1 (Classification problem: south african heart disease data)**.** We'll consider a subset of the Coronary Risk-Factor Study (CORIS) baseline survey, carried out in three rural areas of the Western Cape, South Africa:

- the dataset `SAheart` is included in the `ElemStatLearn` package
- the response variable (`chd`) is the presence or absence of myocardial infarction (MI) at the time of the survey
- there are 160 cases in our data set, and a sample of 302 controls (so we're in the unbalanced setting).
- the aim of the study was to study the risk factors
- the dataset contains 462 observations on the following 9 covariantes other than the outcome `chd`:
  - `sbp`: systolic blood pressure
  - `tobacco`: cumulative tobacco (kg)
  - `ldl`: low density lipoprotein cholesterol
  - `adiposity`: a numeric vector
  - `famhist`: family history of heart disease, a factor with levels Absent/Present
  - `typea`: type-A behavior, a measure of psychosocial stress, as measured by the self-administered Bortner Scale.
  - `obesity`: a numeric vector
  - `alcohol`: current alcohol consumption

– age: age at onset

```
library(lbdatasets)
head(SAheart)

##   sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
## 1 160   12.00 5.73     23.11 Present    49   25.30   97.20  52   1
## 2 144    0.01 4.41     28.61  Absent    55   28.87    2.06  63   1
## 3 118    0.08 3.48     32.28 Present    52   29.14    3.81  46   0
## 4 170    7.50 6.41     38.03 Present    51   31.99   24.26  58   1
## 5 134   13.60 3.50     27.78 Present    60   25.99   57.34  49   1
## 6 132    6.20 6.47     36.21 Present    62   30.77   14.14  45   0

summary(SAheart)

##       sbp           tobacco             ldl           adiposity
##  Min.   :101.0   Min.   : 0.0000   Min.   : 0.980   Min.   : 6.74
##  1st Qu.:124.0   1st Qu.: 0.0525   1st Qu.: 3.283   1st Qu.:19.77
##  Median :134.0   Median : 2.0000   Median : 4.340   Median :26.11
##  Mean   :138.3   Mean   : 3.6356   Mean   : 4.740   Mean   :25.41
##  3rd Qu.:148.0   3rd Qu.: 5.5000   3rd Qu.: 5.790   3rd Qu.:31.23
##  Max.   :218.0   Max.   :31.2000   Max.   :15.330   Max.   :42.49
##     famhist        typea          obesity         alcohol            age
##  Absent :270   Min.   :13.0   Min.   :14.70   Min.   :  0.00   Min.   :15.00
##  Present:192   1st Qu.:47.0   1st Qu.:22.98   1st Qu.:  0.51   1st Qu.:31.00
##                Median :53.0   Median :25.80   Median :  7.51   Median :45.00
##                Mean   :53.1   Mean   :26.04   Mean   : 17.04   Mean   :42.82
##                3rd Qu.:60.0   3rd Qu.:28.50   3rd Qu.: 23.89   3rd Qu.:55.00
##                Max.   :78.0   Max.   :46.58   Max.   :147.19   Max.   :64.00
##       chd
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3463
##  3rd Qu.:1.0000
##  Max.   :1.0000

## pairs plot to visualize data
## exclude the last column (response)
## stratify col by response
pairs(SAheart[,-ncol(SAheart)],
      col = ifelse(SAheart$chd==1, "coral", "cornflowerblue"),
      lwd = 1.5) # set circles thick
```



2.4 Logistic regression

2.4.1 The model

Important remark 30 (Why not linear regression?). Considering a dichotomic outcome Y, if we model the relationship between X and Y, using a linear regression model with the outcome variable directly to represent these probabilities

$$p(X) = \beta_0 + \beta_1 X$$

some problems occur:

- assumption on conditional normality is not respected
- prediction of the estimated model could fall outside the 0-1 interval on extremes of the independent variable

Important remark 31 (Logistic regression and function). To avoid these problems logistic regression *models the probability* that Y *belongs to a particular*

category given the covariates, $\mathbb{P}(Y = 1|X) = p(X)$, using a function that forces outputs between 0 and 1, the *logistic function*:

$$\mathbb{P}(Y = 1|X) = p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Important remark 32 (Prediction, classification). Once estimated the coefficients we can obtain a predicted probability (that will range between 0 and 1) and thus classify class = 1 for any individual for whom the probability is larger than (say) 0.5, or 0 otherwise.

Definition 2.4.1 (Odd of event)**.** After a bit of manipulation, we find quantity the *odds of event*

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0+\beta_1 X}$$

which can take values between 0 (close to 0 indicates low probability of event) and ∞ (resp very high)

Definition 2.4.2 (Log-odd of event)**.** By taking the logarithm of both sides, we arrive at the *log-odds* or *logit* of event:

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X$$

Remark 12. Thus the logistic regression model has a *logit that is linear in* X: increasing X by one unit changes the log-odds by β_1, or equivalently it multiplies the odds by e^{β_1}.

Important remark 33 (β_j interpretation). Because the relationship between $p(X)$ and X is not a straight line, β_1 does not correspond to the change in $p(X)$ associated with a one-unit increase in X; the amount that $p(X)$ changes due to a one-unit change in X will depend on the current value of X. Regardless of that:

- if β_1 is positive then increasing X will be associated with increasing $p(X)$;
- if β_1 is negative then increasing X will be associated with decreasing $p(X)$.

2.4.2 Coefficient estimation

There's no close formula for the estimators, so coefficients are estimated via maximum likelihood.

The idea is to seek estimates for β_0 and β_1 such that the predicted probability $\hat{\pi}(x_i)$ of chd for each individual corresponds as closely as possible to the individual's observed chd status. In other words, we try to find β_0 and β_1 such that plugging these estimates into the model for $p(X)$ yields:

- a number close to one for all individuals who got chd,
- and a number close to zero for all individuals who did not.

More formally, the likelihood function in the binary case is the following. Being units independent, the likelihood is the products of the probabilities that can be compactly rewritten as product of bernoulli

$$\mathrm{L}\left(\beta_0, \beta_1\right) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to maximize this likelihood function.

Log likelihood of logistic regression can be written as:

$$\begin{aligned}
\ell(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \left[y_i \log p(x_i;\boldsymbol{\beta}) + (1-y_i)\log(1-p(x_i;\boldsymbol{\beta}))\right] \\
&= \sum_{i=1}^{n} \left[y_i \log p(x_i;\boldsymbol{\beta}) + \log(1-p(x_i;\boldsymbol{\beta})) - y_i \log(1-p(x_i;\boldsymbol{\beta}))\right] \\
&= \sum_{i=1}^{n} \left[y_i \log\left(\frac{p(x_i;\boldsymbol{\beta})}{1-p(x_i;\boldsymbol{\beta})}\right) + \log(1-p(x_i;\boldsymbol{\beta}))\right]
\end{aligned}$$

Now we have by the previous development that $\log\left(\frac{p(x_i;\boldsymbol{\beta})}{1-p(x_i;\boldsymbol{\beta})}\right) = \beta_0 + \beta_1 x_i$. Given furthermore that $p(x_i;\boldsymbol{\beta}) = \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}$ we have

$$\log(1-p(x_i;\boldsymbol{\beta})) = \log\left(\frac{1+e^{\beta_0+\beta_1 x_i} - e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}\right) = -\log\left(1+e^{\beta_0+\beta_1 x_i}\right)$$

And thus finally the likelihood is:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i(\beta_0+\beta_1 x_i) - \log(1+e^{\beta_0+\beta_1 x_i})\right]$$

To maximize the log-likelihood, we set its partial derivative with respect to β_0, β_1 to zero. Taking partial derivatives we have

$$\begin{aligned}
\frac{\partial \ell(\boldsymbol{\beta})}{\partial \beta_0} &= \sum_{i=1}^{n}\left[y_i - \frac{1}{1+e^{\beta_0+\beta_1 x_i}}\cdot e^{\beta_0+\beta_1 x_i}\cdot 1\right] = \sum_{i=1}^{n}\left[y_i - p(x_i;\boldsymbol{\beta})\right] \\
\frac{\partial \ell(\boldsymbol{\beta})}{\partial \beta_1} &= \sum_{i=1}^{n}\left[y_i x_i - \frac{1}{1+e^{\beta_0+\beta_1 x_i}}\cdot e^{\beta_0+\beta_1 x_i}\cdot x_i\right] = \sum_{i=1}^{n}\left[x_i(y_i - p(x_i;\boldsymbol{\beta}))\right]
\end{aligned}$$

which are 2 equations nonlinear in β; these equation are set equal to zero and solved via newton raphson.

Thus, in general, assuming that the vector of inputs x_i includes the constant term 1 to accommodate the intercept, we can write:

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \beta_i} = \sum_{i=1}^{n} x_i(y_i - p(x_i,\beta)) = 0$$

A fun fact in all the process is that setting the first equation above equal to 0 makes (having the first component of x_i is 1):

$$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} p(x_i;\boldsymbol{\beta})$$

so the intercept is a quantity that serves to adjust other estimates in order to have the sum of predicted probability matching with the sum of successes in our sample.

2.4.3 Multiple logistic regression

We now consider the problem of predicting a binary response using multiple predictors; we can generalize the model as follows:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = X\beta$$

where $X = (X_1, \ldots, X_p)$ are p predictors. The equation can be rewritten as

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}$$

with $\beta_0, \ldots, \beta_p$ estimated again via maximum likelihood.

2.4.4 Example

Example 2.4.1. For the SAheart data, the estimated coefficients of the logistic regression model that predicts the probability of chd using tobacco:

```
library(lbdatasets)
## univariate tobacco
## ------------------
mod <- glm(chd ~ tobacco, data = SAheart, family = binomial)
summary(mod)$coef

##               Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -1.1894300 0.13899530 -8.557340 1.155018e-17
## tobacco      0.1452696 0.02476472  5.865991 4.464574e-09
```

We have that:

- a one-unit increase in tobacco is associated with an increase in the log odds of chd by 0.1453, > 0, so there is a positive association between tobacco and chd
- z-statistic associated with β_1 is equal to $\beta_1/SE(\beta_1)$, and so a large (absolute) value of the z-statistic indicates evidence against $H_0 : \beta_1 = 0 \implies p(X) = \frac{e^{\beta_0}}{1+e^{\beta_0}}$
- $\hat{\beta}_0$ is typically not of interest; its main purpose is to adjust the average fitted probabilities to the proportion of ones in the data.
- once the coefficients have been estimated, it is easy to compute the predicted probability of chd for any given tobacco consumption. eg for an individual with a tobacco consumption of 1.5 the predicted probability of having a coronary heart disease is

$$\hat{p}(x) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}} = \frac{e^{-1.1894+0.1453\cdot 1.5}}{1+e^{-1.1894+0.1453\cdot 1.5}} = 0.2746.$$

while in contrast, for an individual with a tobacco of 0.001 is lower, and about 23.34%.

```
predict(mod, newdata = data.frame("tobacco" = c(1.5, 0.001)), type = 'response')

##         1         2
## 0.2745765 0.2333869
```

- one can use qualitative predictors with the logistic regression model. eg with the qualitative variable famhist (Present or Absent)

```
## univariate famhist
## ------------------
mod <- glm(chd ~ famhist, data = SAheart, family = binomial)
summary(mod)$coef

##                  Estimate Std. Error   z value     Pr(>|z|)
## (Intercept)    -1.168993  0.1431060 -8.168720 3.116788e-16
## famhistPresent  1.168993  0.2032552  5.751357 8.853003e-09
```

The coefficient associated is positive with statistically significant p-value: subjects with a family history of heart disease tend to have higher chd probabilities than those that do not have:

$$\hat{\mathbb{P}}(\text{chd} = 1|\text{famhist} = \text{Present}) = \frac{e^{-1.1690+1.1690\times 1}}{1+e^{-1.1690+1.1690\times 1}} = 0.5$$
$$\hat{\mathbb{P}}(\text{chd} = 1|\text{famhist} = \text{Absent}) = \frac{e^{-1.1690+1.1690\times 0}}{1+e^{-1.1690+1.1690\times 0}} = 0.237$$

```
predict(mod, newdata = data.frame("famhist" = c("Present", "Absent")), type = 'r

##        1        2
## 0.500000 0.237037
```

Going with the **multiple logistic regression**, there are some surprises in this table of coefficients, which must be interpreted with caution.

```
## Multiple logistic regression
## ----------------------------
out.lr <- glm(chd ~ ., data = SAheart, family = binomial) # . means all the remaining
summary(out.lr)

##
## Call:
## glm(formula = chd ~ ., family = binomial, data = SAheart)
##
## Coefficients:
```

```
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -6.1507209  1.3082600  -4.701 2.58e-06 ***
## sbp             0.0065040  0.0057304   1.135 0.256374
## tobacco         0.0793764  0.0266028   2.984 0.002847 **
## ldl             0.1739239  0.0596617   2.915 0.003555 **
## adiposity       0.0185866  0.0292894   0.635 0.525700
## famhistPresent  0.9253704  0.2278940   4.061 4.90e-05 ***
## typea           0.0395950  0.0123202   3.214 0.001310 **
## obesity        -0.0629099  0.0442477  -1.422 0.155095
## alcohol         0.0001217  0.0044832   0.027 0.978350
## age             0.0452253  0.0121298   3.728 0.000193 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 472.14  on 452  degrees of freedom
## AIC: 492.14
##
## Number of Fisher Scoring iterations: 5
```

- Systolic blood pressure (sbp) is not significant! Nor is obesity, and its sign is negative. This results from the correlation between the set of predictors. On their own, they are both significant, and with positive sign. In the presence of many other correlated variables, they are no longer needed (and can even get a negative sign).

- At this stage the analyst might do some **model selection**; find a subset of the variables that are sufficient for explaining their joint effect on the prevalence of chd:

  - one way is to drop the least significant coefficient, and refit the model. This is done repeatedly until no further terms can be dropped from the model.
  - alternatively (more time) refit each of the models with one variable removed, and then perform an analysis of deviance to decide which variable to exclude. The residual deviance of a fitted model is minus twice its log-likelihood, and the deviance between two models is the difference of their individual residual deviances.

  Doing the first way

```
## Reduced model with all the significant at the first step
## ------------------------------------------------------
out.lr.red <- glm(chd ~ tobacco + ldl + famhist + typea + age,
                  data = SAheart, family = "binomial")
summary(out.lr.red)
```

```
##
## Call:
## glm(formula = chd ~ tobacco + ldl + famhist + typea + age, family = "binomial"
##     data = SAheart)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -6.44644    0.92087  -7.000 2.55e-12 ***
## tobacco         0.08038    0.02588   3.106  0.00190 **
## ldl             0.16199    0.05497   2.947  0.00321 **
## famhistPresent  0.90818    0.22576   4.023 5.75e-05 ***
## typea           0.03712    0.01217   3.051  0.00228 **
## age             0.05046    0.01021   4.944 7.65e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 475.69  on 456  degrees of freedom
## AIC: 487.69
##
## Number of Fisher Scoring iterations: 5

## residual deviance is now larger because we removed explicatory
## variables (like RSS)

## comparison between the models
coef(out.lr); coef(out.lr.red) # coefficient don't change too much

##    (Intercept)            sbp        tobacco            ldl      adiposity
##  -6.1507208650   0.0065040171   0.0793764457   0.1739238981   0.0185865682
## famhistPresent          typea        obesity        alcohol            age
##   0.9253704194   0.0395950250  -0.0629098693   0.0001216624   0.0452253496
##    (Intercept)        tobacco            ldl famhistPresent          typea
##    -6.44644451     0.08037533     0.16199164     0.90817526     0.03711521
##            age
##     0.05046038

AIC(out.lr); AIC(out.lr.red) # AIC is better in the restricted model

## [1] 492.14
## [1] 487.6856
```

After selection (tobacco is measured in total lifetime usage in kilograms; thus) an increase of 1kg in lifetime tobacco usage accounts for an increase in the odds of coronary heart disease of $\exp(0.081) = 1.084$ or 8.4%.

Finally we estimate the **test error** of the logistic regression (the last model) for

the classification of patients with chd via k-fold cross-validation (with $k = 5$).

```
## 5-fold CV
## ---------
set.seed(1234)
k <- 5 # nfolds
n <- nrow(SAheart)
folds <- sample(1:k, n, replace = TRUE)
cv_err <- rep(NA, k) # contains cross validation error
yhat <- rep(NA, n) # predicted value for each observation
## select only
db <- SAheart[c("chd","tobacco","ldl","famhist","typea","age")]
table(folds)  # here numerosity of the folds are not same

## folds
##   1   2   3   4   5
##  88  85 105  90  94

for (i in 1:k){ # go through the different folds
    ## split data
    train_id   <- folds != i
    test_id    <- folds == i
    db_train <- db[train_id, ]
    db_test  <- db[test_id, ]
    ## estimate the model using training data
    fold_mod <- glm(chd ~ ., data = db_train, family = binomial)
    ## predict class on test data and store it
    yhat[test_id] <- pred <- as.integer(predict(fold_mod, newdata = db_test, type = "response"))
    ## estimate fold error
    cv_err[i] <- mean(db_test$chd != pred)
}

## fold and mean error
cv_err

## [1] 0.3068182 0.2941176 0.3333333 0.2111111 0.2340426

mean(cv_err)

## [1] 0.2758846

## confusion matrix and stats
addmargins(table('pred' = yhat, 'chd' = SAheart$ chd))

##      chd
## pred   0   1 Sum
##   0  247  73 320
##   1   55  87 142
##   Sum 302 160 462

(error = mean(yhat != SAheart$chd))
```

```
## [1] 0.2770563

(sens = 87/160)

## [1] 0.54375

(precision = 87/145)

## [1] 0.6
```

Sensitivity and precision are not that great.

2.5 LDA

Important remark 34 (Benefit vs logistic). We have:

- when the classes are well-separated, the parameter estimates for the logistic regression model are unstable (this happen because any sigmoid can fit data well when they are very different);
- LDA is more stable if n is small and the distribution of the predictors X is approximately normal in each of the classes;
- LDA is easily extendable to and popular when we have *more than two response classes.*

2.5.1 Fisher method with 2 groups

Definition 2.5.1 (lda)**.** We:

- have two population (Π_0 and Π_1)
- want to classify based on the characteristic $\mathbf{x}$ in two population Π_1 and Π_0: if the posterior probability $f(\Pi_1|\mathbf{x})$ is above a certain treshold we classify the unit as coming from Π_1, otherwise from Π_0;
- estimating $f(\Pi_1|\mathbf{x})$ can be difficult especially for the multivariate likelihood (and consider Fisher acted in 1920');
- however we need not be overly concerned with estimating $f(\Pi_1|\mathbf{x})$: if we believe that $f(\Pi_1|\mathbf{x})$ increases monotonically (or approximately monotonically) in some direction $\mathbf{w}$ in the $\mathbf{x}$ space, we can compare $\mathbf{x}'\mathbf{w}$ with a treshold to choose the classified group (especially the mean of the two groups in the new direction/projection)
- so I project my point through a different space by using $\mathbf{w}$ in $\mathbf{x}'\mathbf{w}$ (i project $\mathbf{x}$ along the direction $\mathbf{w}$) where the two population are as far/distinct as possible. Then I check to which population my unit is closer to.
- how to find $\mathbf{w}$:

– we want to choose it in a way that $\mathbf{x}'\mathbf{w}$ separate most the two groups in the final variable: if groups are summarized by the their means we want the difference of them to be as high as possible

$$\mathbf{w}'(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_0) = \mathbf{w}'\overline{\mathbf{x}}_1 - \mathbf{w}'\overline{\mathbf{x}}_0$$

eg the difference above to be maximized

– furthermore we want to consider the variability and standardizing it: the only assumption that Fisher made is that the two population have the same variance (in the original paper nothing been said about gaussian).
Supposing the two population have equal variance/covariance matrices (*only assumption is homoskedasticity*) the estimated common within-class variance in the direction $\mathbf{w}$ is $\mathbf{w}'\mathbf{S}\mathbf{w}$, where $\mathbf{S}$ is the estimated within class variance-covariance matrix (the sample estimation of Σ).
This leads to the (squared) distance measure:

$$D(\mathbf{w}) = \frac{(\mathbf{w}'\overline{\mathbf{x}}_1 - \mathbf{w}'\overline{\mathbf{x}}_0)^2}{\mathbf{w}'\mathbf{S}\mathbf{w}}$$

$D(\mathbf{w})$ gives the distance or separability between the two classes in the direction $\mathbf{w}$. To find the best direction we need to find that $\mathbf{w}$ which maximises $D(\mathbf{w})$ and this can be shown to be proportional

$$\mathbf{w} \propto \mathbf{S}^{-1}(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_0)$$

- once applied the transformation a new object $\mathbf{x}$ will be classified according to its position on this continuum, so if nearer to class 1 (its mean i suppose) will classified as 1 or 0 viceversa.

Important remark 35 (Optimality for elliptical distribution). Consider that

- actually no distributional assumptions have been made in the above derivation, the method is applicable if the two population are homoschedastic (with common variance covariance matrix Σ);

- the distributions were summarised in terms of their first and second-order moments: if the underlying distributions could be completely described by those moments (elliptical, the most important special case being the multivariate normal distribution) we have *optimality*: the rule that we find as result is actually approximating the bayes error done by the bayes classifier (the gold standard).

Example 2.5.1 (MVN case)**.** Suppose that the classes have multivariate normal distributions

$$\begin{aligned} \Pi_0 &\sim \text{MVN}(\mu_0, \Sigma), \quad \text{with prior probability } \pi_0 \\ \Pi_1 &\sim \text{MVN}(\mu_1, \Sigma), \quad \text{with prior probability } \pi_1 \end{aligned}$$

Then the posteriors likelihood ratio (the thing we use to classify 1 if the statistic calculated is > 1) would be (denominators elides themself):

$$\frac{f(1|\mathbf{x})}{f(0|\mathbf{x})} = \frac{\pi_1 f(\mathbf{x}|1)}{\pi_0 f(\mathbf{x}|0)} = \frac{\pi_1 \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right)}{\pi_0 \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_0)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_0)\right)}$$
$$= \frac{\pi_1}{\pi_0} \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right)}{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_0)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_0)\right)}$$

To decide we can compare this ratio with a threshold: if the costs of the two types of misclassification are equal this threshold will be 1.
If we take logs to simplify (and thus the threshold change to 0) we have:

$$\log\frac{f(1|\mathbf{x})}{f(0|\mathbf{x})} = \log\frac{\pi_1}{\pi_0} - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_0)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_0)$$
$$= \log\frac{\pi_1}{\pi_0} - \frac{1}{2}\left[\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} - \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_1\mathbf{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_1\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 - \mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0 + \boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\mathbf{x} - \boldsymbol{\mu}_0^T\mathbf{\Sigma}\right.$$
$$= \log\frac{\pi_1}{\pi_0} - \frac{1}{2}\left[-2\mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + 2\mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0\right]$$
$$= \log\frac{\pi_1}{\pi_0} - \frac{1}{2}\left[-2\mathbf{x}^T\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0) + \boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0\right]$$
$$= \log\frac{\pi_1}{\pi_0} + \mathbf{x}^T\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0) - \frac{1}{2}\boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0$$

At the end the decision depends on :

- the priors π_0, π_1
- the rescaled (for the overall variance) vectors means of the two population (terms $-\frac{1}{2}\boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1$ and $+\frac{1}{2}\boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0$)
- finally/especially $\mathbf{x}^T\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)$ that is the difference between the two population means by the variance. This thing is very similar to

  $$D(\mathbf{w}) = \frac{(\mathbf{w}^T\overline{\mathbf{x}}_1 - \mathbf{w}^T\overline{\mathbf{x}}_0)^2}{\mathbf{w}^T\mathbf{S}\mathbf{w}}$$

  obtained before (before introducing normality hypothesis) and being $\mathbf{S}$ the sample estimate of $\mathbf{\Sigma}$.
  So by following two different approaches (Fisher distribution agnostic for $D(\mathbf{w})$, and Bayes rules using gaussian distribution) we find something similar.
  By focusing on this last term, the final classification rule will be equivalent to compare $\mathbf{x}'\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)$ with a threshold (depending on the $\boldsymbol{\mu}_k$, Σ, the priors and the costs)

Important remark 36 (Final remarks). To summarize:

- Fisher's linear discriminant method is *optimal for elliptical distributions*, such as the multivariate normal, with equal covariance matrices (but it does not 'assume' multivariate normal distributions).

- on the other hand, the method *may perform well even if the distributions are not elliptical*. For example, there is evidence to suggest that it does well even for multivariate binary data when the true optimum decision surface is linear.

- *Overfitting* may play a role in the practical cases the higher the ratio of parameters p to number of samples n is;

- *changing treshold*: if there are too few positive predicted the researcher could diminish the treshold to mark positive (however this could worsen prediction on the negative (more false positive) and overall error rate)

- since Fisher's method is based on second-degree terms in the **x**s, one might expect a relationship of some kind to regression, which minimises a sum of squares criterion. In fact the relationship is very close.

2.5.2 Relationship with linear regression

Let's consider again two classes, with the class membership of each point being described by a variable y coded as 0 or 1: by using OLS, it is possible to formulate a class membership prediction rule, by classifying new case as class 1 if the prediction from the regression $\hat{y}$ greater than the threshold and 0 otherwise. Considering a standard linear model where $\mathbf{X}$ is mean centered, $y = 0$ if $i \in \Pi_0$ (n_0 are the number of units of Π_0), $y = 1$ if $i \in \Pi_1$ (n_1 are the number of units of Π_1), and obtain the estimator

$$\begin{aligned}
&\mathbf{y} = \mathbf{X}\boldsymbol{\beta} \\
&\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\
&(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \widehat{\boldsymbol{\beta}}
\end{aligned}$$

In this setup we have:

$$\mathbf{X}^T\mathbf{y} = n_1(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}) = \frac{n_1 n_0}{n_0 + n_1}(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_0)$$

Supposing that the dataset is ordered having the variable $\mathbf{y}$ with all the n_0 zeros at first and then n_1 ones at the end we have:

$$\mathbf{X} = \begin{bmatrix} (x_{11} - \overline{\mathbf{x}}_1) & (x_{12} - \overline{\mathbf{x}}_2) & \dots & (x_{1p} - \overline{x}_p) \\ (x_{21} - \overline{\mathbf{x}}_1) & (x_{22} - \overline{\mathbf{x}}_2) & \dots & (x_{2p} - \overline{x}_p) \\ \dots & \dots & \dots & \dots \\ (x_{n1} - \overline{\mathbf{x}}_1) & (x_{n2} - \overline{\mathbf{x}}_2) & \dots & (x_{np} - \overline{x}_p) \end{bmatrix}$$

$$\mathbf{X}^T\mathbf{y} = \begin{bmatrix} (x_{11} - \overline{x}_1) & (x_{21} - \overline{x}_1) & \dots & (x_{n1} - \overline{x}_1) \\ (x_{12} - \overline{x}_2) & (x_{22} - \overline{x}_2) & \dots & (x_{n2} - \overline{x}_2) \\ \dots & \dots & \dots & \dots \\ (x_{1p} - \overline{x}_p) & (x_{2p} - \overline{x}_p) & \dots & (x_{np} - \overline{x}_p) \end{bmatrix} \begin{bmatrix} 0 \\ \dots \\ 0 \\ 1 \\ \dots \\ 1 \end{bmatrix} = \begin{bmatrix} \textcircled{1} \\ \textcircled{2} \\ \dots \\ \textcircled{p} \end{bmatrix}$$

The last vector's first element is calculated as:

$$\begin{aligned}
\textcircled{1} &= \sum_{i=1}^{n} y_i(x_{i1} - \overline{x}_1) \overset{(1)}{=} \sum_{i=1}^{n_0} y_i(x_{i1} - \overline{x}_1) + \sum_{i=n_0+1}^{n_0+n_1} y_i(x_{i1} - \overline{x}_1) \\
&\overset{(2)}{=} 0 + \left(\sum_{i=n_0+1}^{n_0+n_1} x_{i1} \right) - n_1\overline{x}_1 \overset{(3)}{=} n_1\overline{x}_{(1)1} - n_1\overline{x}_1 \\
&= n_1(\overline{x}_{(1)1} - \overline{x}_1)
\end{aligned}$$

where

- in (1) we splitted the sum in the 0 and 1 components parts;
- in (2) we noted that all the y from the first set are 0 so is the sum, while from the second set are actually all 1 and so splitted the remaining sum;
- in (3) we denoted $\overline{x}_{(1)1}$ as the mean of variable 1 in population 1.

Similarly for ②, $\overline{x}_{(1)2}$ means the mean of variable 2 in population 1.

$$\begin{aligned}
\textcircled{2} &= \sum_{i=1}^{n} y_i(x_{i2} - \overline{x}_2) = \sum_{i=1}^{n_0} y_i(x_{i2} - \overline{x}_2) + \sum_{i=n_0+1}^{n_0+n_1} y_i(x_{i2} - \overline{x}_2) \\
&= 0 + \left(\sum_{i=n_0+1}^{n_0+n_1} x_{i2} \right) - n_1\overline{x}_2 = n_1\overline{x}_{(1)2} - n_1\overline{x}_2 \\
&= n_1(\overline{x}_{(1)2} - \overline{x}_2)
\end{aligned}$$

and the same happens for ⓟ.
So finally we can rewrite

$$\begin{aligned}
\mathbf{X}^T\mathbf{y} &= \begin{bmatrix} n_1(\overline{x}_{(1)1} - \overline{x}_1) \\ n_1(\overline{x}_{(1)2} - \overline{x}_2) \\ \cdots \\ n_1(\overline{x}_{(1)p} - \overline{x}_p) \end{bmatrix} \overset{(1)}{=} n_1 \left[\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}} \right] \overset{(2)}{=} n_1 \left[\overline{\mathbf{x}}_{(1)} - \frac{\overline{\mathbf{x}}_{(1)} \cdot n_1 + \overline{\mathbf{x}}_{(0)} \cdot n_0 +}{n_1 + n_0} \right] \\
&= n_1 \left[\frac{n_1\overline{\mathbf{x}}_1 + n_0\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_1 n_1 - \overline{\mathbf{x}}_{(0)} n_0}{n_1 + n_0} \right] = n_1 \left[\frac{n_0(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})}{n_1 + n_0} \right] \\
&= \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})
\end{aligned}$$

where

- in (1) we substitute with the vector of means of population 1 $\overline{\mathbf{x}}_{(1)}$ and the vector of overall means ($\overline{\mathbf{x}}$)
- in (2) we splitted the overall mean vector using the two populations mean vector and group numbers

So $\mathbf{X}^T\mathbf{y}$ can be rewritten as the difference between the two groups means vectors weighted by a ratio coming from their numerosity.
Now let's see how $\mathbf{X}^T\mathbf{X}$ looks like. Since $\mathbf{X}^T\mathbf{X}$ is the overall covariance matrix

we can decompose it in within and the between sum of squares (covariance matrix) using the standard decomposition in within sum of square

$$(\mathbf{X}^T\mathbf{X}) = (n_0 + n_1 - 2)\mathbf{S} + \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})^T$$

where

- $\mathbf{S}$ is the sample covariance matrix.
- $(n_0 + n_1 - 2)\mathbf{S}$ is the within covariance matrix
- $\frac{n_1 n_0}{n_1+n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})$ is the between covariance matrix

Finally substituting all the results above we have:

$$(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

$$\left[(n_0 + n_1 - 2)\mathbf{S} + \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})^T\right]\boldsymbol{\beta} = \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})$$

By setting/calling $\alpha = (\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})^T\boldsymbol{\beta}$

$$\begin{aligned}
(n_0 + n_1 - 2)\mathbf{S}\boldsymbol{\beta} + \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})\alpha &= \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)}) \\
(n_0 + n_1 - 2)\mathbf{S}\boldsymbol{\beta} &= \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)}) - \frac{n_1 n_0}{n_1 + n_0}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})\alpha \\
(n_0 + n_1 - 2)\mathbf{S}\boldsymbol{\beta} &= \frac{n_1 n_0}{n_1 + n_0}\left[\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)}\right][1 - \alpha]
\end{aligned}$$

and finally

$$\boldsymbol{\beta} = \frac{1 - \alpha}{n_0 + n_1 - 2}\mathbf{S}^{-1}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})\frac{n_1 n_0}{n_1 + n_0}$$

Important remark 37. To recap we can see that

$$\boldsymbol{\beta} \propto \mathbf{S}^{-1}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})$$

that is the coefficients $\boldsymbol{\beta}$ are proportional to the term $\mathbf{S}^{-1}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})$ we found in the classifier (before assuming gaussianity).
So there are connections between all the three; for this reason the classification they produce will be somewhat related/correlated (not the same).

2.5.3 More than two classes

The method for two classes can be easily extended to more than 2

- if we assume multivariate normal distribution: in a bayesian way, we assign a new point $\mathbf{x}$ to the class j which has the largest value of

$$\begin{aligned}
f(j|\mathbf{x}) &= \frac{\pi_j \cdot f(\mathbf{x}|j)}{\sum_j \pi_j \cdot f(\mathbf{x}|j)} \\
&\propto \pi_j \cdot f(\mathbf{x}|j) \\
&= \pi_j \frac{1}{(2\pi)^{(p/2)}\,|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu_j)^T\Sigma^{-1}(\mathbf{x} - \mu_j)\right]
\end{aligned}$$

Again log transforming this is equivalent to assigning to the class which has the largest value of

$$\log(\pi_j) + \mathbf{x}^T \Sigma^{-1} \mu_j - \frac{1}{2} \mu_j^T \Sigma^{-1} \mu_j$$

- otherwise, without using bayes and multivariate normal assumption the vector $\mathbf{w}$ is determined maximizing the ratio

$$\lambda = \frac{\mathbf{w}^T \mathbf{B} \mathbf{w}}{\mathbf{w}^T \mathbf{S} \mathbf{w}}$$

where:

  - $\mathbf{B}$ is the between variance/covariance matrix,

$$\mathbf{B} = \frac{1}{C-1} \left[\sum_{j=1}^{C} (\overline{\mathbf{x}}_j - \overline{\mathbf{x}})(\overline{\mathbf{x}}_j - \overline{\mathbf{x}})' \right]$$

  - $\mathbf{S}$ is the within group variance/covariance

$$\mathbf{S} = \frac{1}{\sum_j n_j - C} \left[\sum_{j=1}^{C} \sum_{i=1}^{n_j} (\mathbf{x}_{ij} - \overline{\boldsymbol{x}}_j)(\mathbf{x}_{ij} - \overline{\boldsymbol{x}}_j)' \right]$$

That is we are trying to find that direction $\mathbf{w}$ which maximises the separation between the sample means, standardised for the within-class relations between the variables.
Differentiating λ with respect to $\mathbf{w}$ and equating to zero yields

$$\mathbf{B}\mathbf{w} - \lambda \mathbf{S}\mathbf{w} = 0.$$

This two-sided eigenvalue equation has solutions for values of λ satisfying $|\mathbf{B} - \lambda \mathbf{S}| = 0$. That is has solutions for eigenvalues of $\mathbf{S}^{-1}\mathbf{B}$.
The number of distinct eigenvalues is equal to the smaller of $C-1$ and p. The eigenvector $\mathbf{w}_1$ corresponding to the largest of these eigenvalues (λ_1) is the direction which leads to maximum separation, as measured by λ, between the groups. This direction is the first discriminant function or first canonical variate. When there are only two groups, as in the previous section, this direction is the only discriminant function.
Subsequent eigenvectors $\mathbf{w}_k$ correspond to maxima of λ subject to constraints $\mathbf{w}_k^T \mathbf{S} \mathbf{w}_h = 0$ for $h = 1, \ldots, k-1$, so that $\mathbf{w}_k^T \mathbf{X}$ and $\mathbf{w}_h^T \mathbf{X}$ are uncorrelated for $h \neq k$.
If we make the $\mathbf{w}_k$ unique by adding the constraint $\mathbf{w}_k^T \mathbf{S} \mathbf{w} \mathbf{k} = 1$ then $\mathbf{W}\mathbf{S}\mathbf{W}^T = \mathbf{I}$ where $\mathbf{w}_k^T$ is the k-th row of $\mathbf{W}$. Given that the canonical variates define the directions in which the groups are best separated, in the sense described above, it is natural to plot the data using these variates as axes. In particular, the two-dimensional space spanned by the first two variates is often used.
When the objective of the analysis is interpretation of the canonical variates, instead of classification, the canonical variates are often adjusted to have zero overall sample mean and unit standard deviation.

2.5.4 Example

Example 2.5.2. Exercise

1. Divide the dataset into training and validation sets. Perform Linear Discriminant Analysis to predict variable chd and compute the test error estimate on the validation set.

```
## LDA
library(lbdatasets)

### Training + Validation sets ####
n <- nrow(SAheart)
## select train indexes and test
set.seed(1234)
train <- sample(1:n, ceiling(n/2), replace = FALSE)
test  <- - train
db_train <- SAheart[train,]
db_test <- SAheart[test,]

out.lda <- MASS::lda(chd ~ ., data=db_train)
out.lda

## Call:
## lda(chd ~ ., data = db_train)
##
## Prior probabilities of groups:
##         0         1
## 0.6753247 0.3246753
##
## Group means:
##        sbp  tobacco      ldl adiposity famhistPresent    typea  obesity
## 0 135.2885 2.661538 4.244295  23.94718      0.3141026 52.60256 25.52397
## 1 143.1467 5.216800 5.897467  27.99227      0.5866667 55.26667 26.38547
##    alcohol      age
## 0 16.40724 39.11538
## 1 18.76013 48.81333
##
## Coefficients of linear discriminants:
##                          LD1
## sbp              0.013358178
## tobacco          0.074236616
## ldl              0.253832043
## adiposity        0.051541163
## famhistPresent   0.650938356
## typea            0.037621116
## obesity         -0.136416561
## alcohol         -0.002074936
## age              0.012492279

## the first line says the prior probabilities eg
```

```
## mean(SAheart[train, "chd"])
## it returns the vector of coefficients to combine our data in a
## single measure that is differentiated the most

## prediction on test dataset
pred.lda <- predict(out.lda, newdata = db_test)

## confusion matrix and error
table(Actual = db_test$chd, LDA = pred.lda$class)

##        LDA
## Actual   0   1
##      0 131  15
##      1  46  39

mean(db_test$chd != pred.lda$class)

## [1] 0.2640693
```

The LDA output indicates that $\hat{\pi}_1 = 0.675$ and $\hat{\pi}_2 = 0.325$; in other words, 67.5% of the training observations correspond to patient with negative `chd`. It also provides the group means; these are the average of each predictor within each class, and are used by LDA as estimates of μ_k.
The coefficients of linear discriminants output provides the linear combination of the predictors that are used to form the LDA decision rule.
The `predict` function returns a list with three elements. The first element, `class`, contains LDA's predictions about the presence of `chd`. The second element, `posterior`, is a matrix whose kth column contains the posterior probability that the corresponding observation belongs to the k-th class. Finally, x contains the linear discriminants, described earlier.

2. How does it compare with the logistic regression?

```
## estimate in train
out.lr <- glm(chd ~ ., data = db_train, family=binomial)
## summary(out.lr)

# prediction and error in test
yhat.lr <- as.integer(predict(out.lr, newdata = db_test, type="response") > 0.5)
table(Actual=db_test$chd, LR=yhat.lr)

##        LR
## Actual   0   1
##      0 131  15
##      1  47  38

mean(db_test$chd != yhat.lr)

## [1] 0.2683983
```

The LDA and logistic regression predictions are almost identical.

2.6 Naive Bayes Classifier

2.6.1 The classifier

Remark 13 (Reason and area of application). In order to apply the Bayes classifier (gold standard classifier) we would need having the multivariate densities for each group we want to classify; to estimate them in a multivariate setting is not easy (especially if we don't assume parametric multivariate distributions, eg we use nonparametric methods)

Definition 2.6.1 (Naive Bayes)**.** The naive Bayes model assumes that given/within a class $Y = h$, the features X_k are independent:

$$f_h = \prod_{j=1}^{p} f_{hj}(X_j)$$

Therefore:

- multivariate density in each group is just a product of univariate ones (which are simpler to estimate);
- this estimation of univariate density f_{hj} (the variable j in group $Y = h$) can be tackled
  - supposed gaussian (so we estimate its parameter only)
  - estimating all these densities nonparametrically (by histograms for discrete variables and quantitative ones or kernel density estimation for quantitatives);
- after having the densities and reconstructed the multivariate density we apply the bayes classifier idea as usual, with a priori probability as well.

Important remark 38 (Assumption and benefits). The assumption of independence of feature within class is generally not true, however:

- it does simplify the estimation dramatically: naive Bayes is especially appropriate/applied when the dimension p of the feature space is high, making density estimation unattractive.
- the classifiers often outperform far more sophisticated alternatives (not assuming it).

For these reasons Naive Bayes classifier is a simple but powerful technique that has remained popular over the years.

Remark 14. In the following we briefly see two methods for non parametric density estimation, histogram and kernel density.

2.6.2 Density estimation with histogram

Remark 15. This is the most widely used nonparametric approach

Definition 2.6.2 (Histogram)**.** We partition the range of possible values into cells of equal length and plot for each one a bar with heigth proportional to the number of observations falling in it.

Important remark 39 (Classifying with histograms). A new point with measurement x would be classified by comparing the heights of the histograms at x for each class.

Important remark 40 (Cons). We have that:

- if the samples for each class had only a few cases (heights of the bars take only a few values), there would be high probability that the estimated probabilities $\hat{f}(x|j)$ (proportional to the heights of the bars) to be equal;
- there are discontinuities in probability estimates at cell boundary (unnatural with continuous variables anyway);
- it is necessary to **decide widths/number/location** of the cells (no assumptions on the shape, like parametric stuff, but a lot of stuff to decide)

Important remark 41 (Width of cells). Width of the cell tune the bias/variance tradeoff:

- large cells (more bias less variance): different x values may correspond to very different probabilities, so putting al togheter the histogram estimate may be very biased for some parts of the cell. cells will tend to have more observations falling within them, so the corresponding probability estimates will have small variance.
- small cells (less bias more variance): few observations will lie within it, so the variance of the probability estimate for that cell will be large; but small cells, covering only a small range of x, have the advantage of small bias.

To choose 'the best' cell width one would needs to adopt some overall performance criterion (such as mean squared error summed over cells) and choose the cell width and location to minimise this.

Remark 16 (Multivariate histograms-based classificator and curse of dimensionality). Aside from naive bayes (which handle different variables separatedly) a multivariate histogram based classificator would incur in the *curse of dimensionality*.
Eg if there are 10 variables, each variable separately divided into 10 cells there would be 10^{10} cells in the multivariate space (most of them empty). Thus for new object classifications may be based on comparing probability estimates of zero with probability estimates of zero

2.6.3 Density estimation with Kernel method

Important remark 42. Kernel density estimation is a generalisation of the histogram approach which overcomes histogram cons; remains the problem of how *wide* the "cells" should be but various estimates have been suggested.

- the aim of the kernel method is to estimate the probability density function in the point x, $f(x)$ (obtaining $\hat{f}(x)$), from a sample of n points randomly drawn from f: the greater the proportion of the sample points lying near x, the higher should be $\hat{f}(x)$;

- if we take an interval/bandwidth h centered on x (bandwidth is the width of a moving window on the real line centered on point of interest which determines which points are considered) we could estimate the probability $\hat{f}(x)$ by the proportion of the n points of the sample which fall in this interval: each point in the bandwidth contributes $1/n$, the remaining 0.
  Problem is that whenever a point enter/leaves the interval $[x-h, x+h]$ there will be a discontinuous jump of $\pm\frac{1}{n}$ (non-smooth similarly to histograms);

- non-smoothness problem can be overcome by replacing the crude weigths (0 or $1/n$) by a smoother set of weights, obtained using a smoother/*kernel function*, which decay as the distance between the center/point of interest x and point considered increases.
  If $X_1, X_2 \ldots, X_n$ denotes a sample of size n from a random variable with density f, density estimate in the center x will be:

$$\hat{f}x = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

  where

  - the *kernel function* K is usually a unimodal/symmetric probability density function centered on 0. Points X_i near the center x ($x - X_i$ small in absolute value) will have heigher weight
  - $h > 0$ is the *smoothing/bandwidth* parameter as above;

  So the value of the kernel estimate at the point x is simply the average of the n kernel ordinates at that point.
  Some decision are still needed:

  - *which kernel to adopt*: there are several, but the impact is not so much:
    * *gaussian*: in this case the parameter h is the standard deviation and the function is the standard normal density function;
    * *Epanechnikov*:

$$K(x) = \frac{3(1-x^2)}{4} \quad -1 \leq x < 1$$

    * *triangle* kernel

$$K(x) = 1 - |x| \quad -1 \leq x < 1$$

  - the second more important decision is the *width/spread of the kernel*: h is similar to choosing the width of the histogram cells in terms of bias/variance tradeoff:

  - h too small (overfitting): estimate will be highly irregular, tending towards the unsmoothed empirical distribution as $h \to 0$.
  - h too large (oversmoothing): estimate will be too smooth, tending towards a constant value as $h \to \infty$.

  There are several approaches (complicated look at the slides).

- *Inheritance*: the kernel density estimator inherits the analytic properties of the function K, so if we adopt a proper density kernel (such as the gaussian) the estimation resulting will be a proper density.
  For example, if K is itself taken to be a unimodal probability density function then $\hat{f}(x)$ will itself be a density ($\hat{f}(x) > 0$, $\int \hat{f}(x)\, \mathrm{d}x = 1$)

2.6.4 Exercise

Example 2.6.1. Use 5-fold CV to estimate the test error of the Naive Bayes classifier, with density estimated with both Gaussian and nonparametric kernel density.

```
library(lbdatasets)
library(klaR)

## Caricamento del pacchetto richiesto:  MASS
##
## Caricamento pacchetto:  'MASS'
## Il seguente oggetto è mascherato da 'package:lbdatasets':
##
##     anorexia

n <- nrow(SAheart)
k <- 5
set.seed(1234)
folds <- sample(1:k, n, replace = TRUE)
## cv error for the two methods (1 for fold)
cv_err_g <- cv_err_k <- rep(NA, k)
## predictions for the two methods (1 for unit)
yhat_g  <- yhat_k  <- rep(NA, n)

## preprocess data (extract response, remove categorical variables)
## the function needs x and y separed (non è vero si possono tenere
## assieme e usare la formula ma ho poco tempo per mettere tutto a
## posto
x <- SAheart[,-c(5,10)]
y <- SAheart[,10]

## prima di fare andare il ciclo per intero assegnare i = 1 e
## testare il codice
for (i in 1:k){

    # splitting (keep x and y separed due to klar::NaiveBayes
```

```
    train <- folds != i
    test  <- folds == i
    x_train <- x[train,]
    x_test  <- x[test,]
    y_train <- y[train]
    y_test  <- y[test]

    ## Naive Bayes with kernel density (usekernel = TRUE)
    mod_k <- NaiveBayes(x = x_train,
                        grouping = as.factor(y_train),
                        usekernel = TRUE)
    pred_k <- predict(mod_k, newdata = x_test)$class # class to extract the predicted label
    cv_err_k[i] <- mean(y_test != pred_k)
    yhat_k[test] <- pred_k

    ## Naive Bayes with Gaussian density (usekernel = FALSE)
    mod_g <- NaiveBayes(x = x_train,
                        grouping = as.factor(y_train),
                        usekernel = FALSE)
    pred_g <- predict(mod_g, newdata = x_test)$class
    cv_err_g[i] <- mean(y_test != pred_g)
    yhat_g[test] <- pred_g
}

## confusion matrix and mean cv errors
addmargins(table(Actual=y, NB.k=yhat_k))

##       NB.k
## Actual   1   2 Sum
##    0   223  79 302
##    1    79  81 160
##    Sum 302 160 462

mean(cv_err_k)

## [1] 0.3422304

addmargins(table(Actual=y, NB.g=yhat_g))

##       NB.g
## Actual   1   2 Sum
##    0   228  74 302
##    1    67  93 160
##    Sum 295 167 462

mean(cv_err_g)

## [1] 0.3054338
```

Here the gaussian returns the smallest error

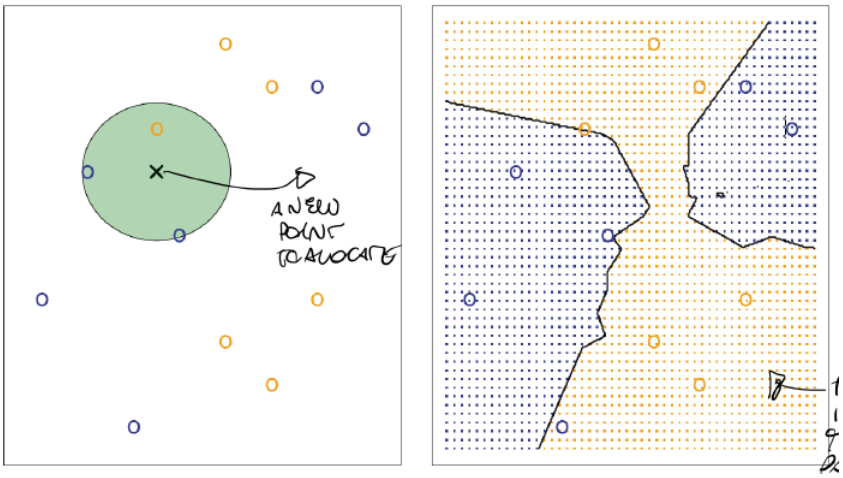


Figura 2.3: knn1

2.7 k-NN Classifier

2.7.1 The model

Definition 2.7.1 (k-nearest neighbors)**.** Given a positive integer k and a test observation x_0, the k-Nearest Neighbor classifier (k-NN):

1. first identifies $\mathcal{N}_0$, the k neighbors points in the training data that are closest to x_0;
2. estimates the conditional probability for class h as the fraction of points in $\mathcal{N}_0$ whose response values equal h:
$$\mathbb{P}(Y = h|X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = h)$$
3. applies Bayes rule classifying the test observation x_0 to the class with the largest probability (ties are randomly resolved).

Remark 17. The k-NN approach, using $k = 3$, is illustrated in figure 2.3 situation where our sample consist f six blue and six orange observations (left panel), while the partitioning produced is represented on the right panel

Important remark 43 (Choices). we need to choose k training points, the number of neighbor using a **distance**

- *distance*: assuming the features are real-valued, we use *Euclidean distance* $d(i) = \left\|x_{(i)} - x_0\right\|$ but we first *standardize* each of the features to have mean 0 and variance 1 (since features measured in different units would be impactful on distance)
- choice of k: has a drastic effect on the obtained classifier (see fig 2.4)
  - when $k = 1$ (left), the decision-area is very flexible/rough and finds patterns in the data that don't correspond to the Bayes decision boundary: it's a classifier with low bias but very high variance.

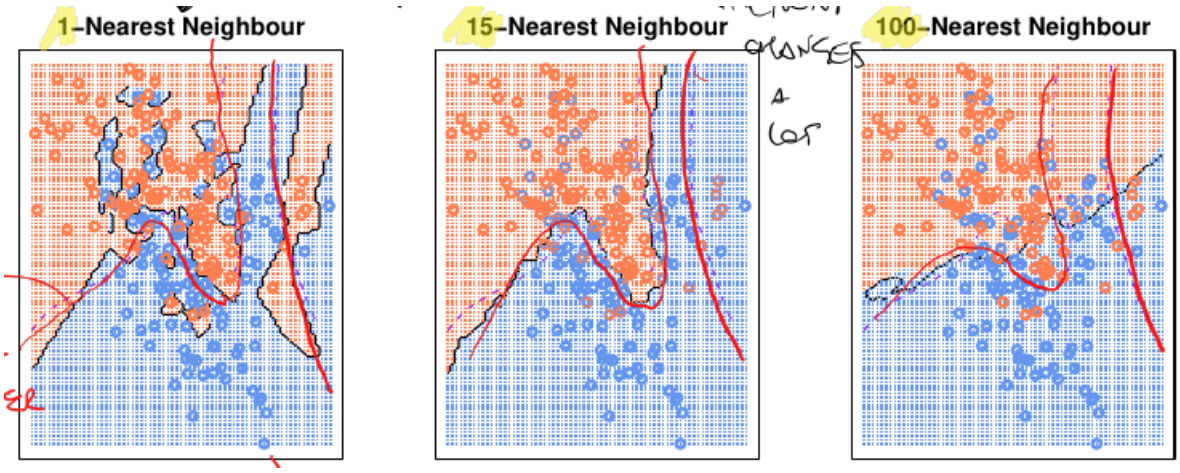


Figura 2.4: Various choices for k with the bayes optimal classifier in red (same in the three images) and the area color representing the classification done with knn

- as k grows (eg 15 or 100), method becomes less flexible and produces a smooth boundary that is close to linear (lower variance but high-bias classifier).

The choice of k can be done via test by calculating the error on test set or via crossvalidation.

Remark 18 (Final remarks). In general:

- it's a memory based classifier and require no model to be fit;
- if number of variables is greater than number of observation, $p > n$, distances we find are not reliable;
- despite being simple k-NN can often produce classifiers surprisingly close to the optimal Bayes classifier

2.7.2 Exercise

1. Divide the dataset into training and validation sets.
2. Use 5-fold CV to choose the best number of neighbors among (1, 3, 5, 11, 15, 25, 45, 105).
3. Use the validation set to estimate the test error.

```
library(lbdatasets)
library(class)

## setup global data
x <- SAheart[, -c(5,10)] # remove famhist (dichotomic) and chd (outcome)
```

```
y <- SAheart$chd
n <- nrow(SAheart)

## 1) splitting in training/development set and validation set
## ----------------------------------------------
set.seed(1234)
# use devel per evitare confusione con sotto nella cv
devel <- sample(1:n, size = ceiling(n/2), replace = FALSE)
valid <- -devel
## development dataset used in cross validation (splitted in training
## and test)
x_devel <- x[devel,]
y_devel <- y[devel]
## validation dataset used for error estimation
x_valid <- x[valid,]
y_valid <- y[valid]

## X standardization
## training set
x_devel_std <- scale(x_devel, center = TRUE, scale = TRUE)
## validation set based on training infos
devel_means <- colMeans(x_devel)
devel_sds   <- apply(x_devel, 2, sd)
x_valid_std <- data.frame(
    Map(function(x, m, s) (x-m)/s,
        as.list(x_valid),
        as.list(devel_means),
        as.list(devel_sds))
)

## 2) choosing optimal k
## ---------------------
## here we use only the train dataset via crossvalidation.
## We don't touch the validation dataset which will
## be used at the end
set.seed(1234)
n_devel <- nrow(x_devel_std)
folds <- sample(1:5, n_devel, replace = TRUE)# cv folds labels
k <- c(1, 3, 5, 11, 15, 25, 45, 105) # n of neighbors we want to consider
cv_err <- matrix(NA, nrow = 5,      # error for the different
                 ncol = length(k), # folds and different k we consider
                 dimnames = list(NULL, paste0("k=",k)))
for (i in 1:5){ # cycle on each fold
    train <- folds != i
    test  <- folds == i
    x_train <- x_devel_std[train,]
    y_train <- y_devel[train]
    x_test <- x_devel_std[test,]
    y_test <- y_devel[test]
```

```
    for (j in 1:length(k)){ # cycle on each k
        yhat <- knn(train = x_train, # x used for train
                    cl = y_train,    # y used for traininig
                    test = x_test,   # x used for prediction of y
                    k = k[j])
        cv_err[i,j] <- mean(yhat != y_test)
    }
}
cv_err # all errors for all folds for all k
```

```
##            k=1       k=3       k=5      k=11      k=15      k=25      k=45
## [1,] 0.2727273 0.3181818 0.2727273 0.2500000 0.2727273 0.2727273 0.2500000
## [2,] 0.5238095 0.4285714 0.4523810 0.3809524 0.3809524 0.3571429 0.3333333
## [3,] 0.3518519 0.4074074 0.3888889 0.3148148 0.2962963 0.2777778 0.3703704
## [4,] 0.4130435 0.3260870 0.3913043 0.3478261 0.3695652 0.3695652 0.3695652
## [5,] 0.3333333 0.4444444 0.3333333 0.2666667 0.2888889 0.2888889 0.2888889
##          k=105
## [1,] 0.2727273
## [2,] 0.2857143
## [3,] 0.3888889
## [4,] 0.3913043
## [5,] 0.2666667
```

```
colMeans(cv_err) # mean error for each k
```

```
##       k=1       k=3       k=5      k=11      k=15      k=25      k=45     k=105
## 0.3789531 0.3849384 0.3677270 0.3120520 0.3216860 0.3132204 0.3224316 0.3210603
```

```
names(which.min(colMeans(cv_err))) # best k using 11-fold cv
```

```
## [1] "k=11"
```

```
## 3) estimate of the error
## ------------------------
## now we've choosen the optimal k, let's use it to do the prediction
## on the standardized validation set and obtaining the error estimate.

## in general if we'standardize the train/development set before applying the
## estimation procedure, then when we go to the validation/test set to compute
## error applying the estimate to obtain the predicted value we need
## to standardize the test group X as well (think of a model i guess,
## otherwise there's discrepancies between beta units and variable
## unit in the prediction)

# prediction on the validation set
yhat_valid <- knn(train = x_devel_std,  # train: we use all the (standardized) train data
                  cl = y_devel,         # class: all the y from the train data
                  test = x_valid_std, # test: standardized x on the validation set
                  k = 11) # optimal k found before

## confusion matrix and error on the validation set
addmargins(table('pred' = yhat_valid, 'actual' = y_valid))
```

```
##      actual
## pred    0   1 Sum
##   0   122  57 179
##   1    24  28  52
##   Sum 146  85 231
```

```
mean(yhat_valid != y_valid)
```

```
## [1] 0.3506494
```

Capitolo 3

Dimension reduction procedures

Important remark 44. In the regression setting with p regressor and n observation,

- when the relationship between the response and the predictors is close to linear the least squares estimates will have low bias
- if $n \gg p$ the estimates tend to have low variance.

Problems occurs when p is large:

- if $p > n$, there is no longer a unique least squares coefficient estimate ($\mathbf{X}^T\mathbf{X}$ is not singular): the variance is infinite so the method cannot be used at all;
- if $p \cong n$ there can be a lot of variability in the fit: a small change in the training data can cause a large change in the least squares coefficient estimates.
  As consequences we have overfitting and consequently poor predictions on test units.

There are three possible solution, focus of this chapter:

- **classical model selection**: identifying a subset of relevant covariates and use only them;
- **regularization**: fit the model on all the p predictor but *penalize/shrink estimates* toward 0 by using ridge (which only shrink toward 0) or lasso (which can actually set to 0); compared to OLS this can introduce a bit bias but reduces the variability of estimates;
- **dimension reduction**: create $M < p$ new variables obtained by computing M different linear combinations of the starting p variables and use them as predictors in the regression model (clearly $n \gg M$)

3.1 Regression dataset

Example 3.1.1 (Prostate cancer)**.** The data for this example come from a study by Stamey et al. (1989) that examined the correlation between the level of prostate specific antigen (PSA) and a number of clinical measures, in 97 men who were about to receive a radical prostatectomy. The dataset is from the `ElemStatLearn` package, no longer available on cran.

```
library(lbdatasets)
summary(prostate)
```

```
##      lcavol           lweight           age             lbph              svi
##  Min.   :-1.3471   Min.   :2.375   Min.   :41.00   Min.   :-1.3863   Min.   :0.000
##  1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00   1st Qu.:-1.3863   1st Qu.:0.000
##  Median : 1.4469   Median :3.623   Median :65.00   Median : 0.3001   Median :0.000
##  Mean   : 1.3500   Mean   :3.629   Mean   :63.87   Mean   : 0.1004   Mean   :0.216
##  3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00   3rd Qu.: 1.5581   3rd Qu.:0.000
##  Max.   : 3.8210   Max.   :4.780   Max.   :79.00   Max.   : 2.3263   Max.   :1.000
##       lcp              gleason          pgg45            lpsa
##  Min.   :-1.3863   Min.   :6.000   Min.   :  0.00   Min.   :-0.4308
##  1st Qu.:-1.3863   1st Qu.:6.000   1st Qu.:  0.00   1st Qu.: 1.7317
##  Median :-0.7985   Median :7.000   Median : 15.00   Median : 2.5915
##  Mean   :-0.1794   Mean   :6.753   Mean   : 24.38   Mean   : 2.4784
##  3rd Qu.: 1.1787   3rd Qu.:7.000   3rd Qu.: 40.00   3rd Qu.: 3.0564
##  Max.   : 2.9042   Max.   :9.000   Max.   :100.00   Max.   : 5.5829
```

The goal is to predict the log of PSA (`lpsa`) from a number of measurements including:

1. log cancer volume (`lcavol`),
2. log prostate weight `lweight`,
3. `age`,
4. log of benign prostatic hyperplasia amount (`lbph`),
5. seminal vesicle invasion (`svi`),
6. log of capsular penetration (`lcp`),
7. Gleason score (`gleason`),
8. percent of Gleason scores 4 or 5 (`pgg45`).

This is a supervised learning problem, known as a *regression problem*, because the outcome measurement is *quantitative*.

3.2 Model selection

3.2.1 Best subset selection

Remark 19. We fit a separate least squares regression for each possible combination of the p predictors in order to identify the best (using common criteria).

Definition 3.2.1 (Best subset selection algorithm)**.** The steps are:

1. Start with the null model $\mathcal{M}_0$
2. for each number of predictors $k = 1, 2, \ldots, p$:
   - fit all $\binom{p}{k}$ models that contain exactly k predictors
   - pick the best (eg smallest RSS or equivalently largest R^2) and save it as $\mathcal{M}_k$
3. compare the model selecting the best from $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using metrics such as crossvalidated prediction error, C_p (AIC), BIC or adjusted R^2

Important remark 45 (Pros/cons). Pros:

- is simple
- explore all the possible options/models, not only nested ones

Cons:

- *computational limitations*: there are 2^p models that involve subset of p predictors so the number grows rapidly as p increases;
- *overoptimization*: the larger the search space, the higher the chance of finding models that look good on the training data even though they might not have any predictive power on test data. It's a multiple testing-like problem.

3.2.2 Stepwise selection

Important remark 46 (Costrainted vs unconstrainted search). Best subset is an *unconstrainted optimization*: in stepwise approaches (forward or backward) we optimize *given* the results obtained in the previous steps.

Definition 3.2.2 (Forward stepwise selection)**.** The steps are:

1. start from the null model $\mathcal{M}_0$
2. for the number of starting variables $k = 0, \ldots, p-1$:
   - consider all $p - k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor
   - choose the best among these $p - k$ models (lower RSS or higher R^2) saving it as $\mathcal{M}_{k+1}$
3. compare the model selecting the best from $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using metrics such as crossvalidated prediction error, C_p (AIC), BIC or adjusted R^2

Important remark 47 (Pros/cons). Pros:

- *less models to be estimated* (compared to best subset): involves fitting one null model, along with $p - k$ models in the k-th iteration, for $k = 0, \ldots, p-1$, which overall are

$$1 + \sum_{k=0}^{p-1} (p - k) = 1 + p(p+1)/2$$

models. eg if $p = 20$ best subset selection requires fitting 1,048,576 models, whereas forward stepwise selection requires fitting only 211 models.

- *can be used even when* $n < p$ but stopping at $\mathcal{M}_{n-1}$ otherwise estimate will not be unique

Cons:

- all models investigated are *nested*: tends to do well in practice but there's no guarantee to find the best possible model out of all 2^p models containing subsets of the p predictors.
  Eg if the best one model parameter contains X_1 and the best overall X_2 and X_3 this latter will not be choosen because it doesn't have X_1

Definition 3.2.3 (Backward stepwise selection)**.** The steps are:

1. start from the full model $\mathcal{M}_p$ which contains all the p predictors
2. for the number of starting variables $k = p, p-1, \ldots, 1$:
   - consider all k models that contains all but one predictors in $\mathcal{M}_k$ (for a total of $k-1$ predictors)
   - choose the best among these k models (lower RSS or higher R^2) saving it as $\mathcal{M}_{k-1}$
3. compare the model selecting the best from $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using metrics such as crossvalidated prediction error, C_p (AIC), BIC or adjusted R^2

Important remark 48 (Pros/cons). Pros:

- investigates *less number of models*, same number as forward: $1+p(p+1)/2$

Cons:

- model investigated are *nested* so again there's no guarantee to yield the best model containing a subset of the p predictors;
- differently from forward, backward *requires* $n > p$ otherwise the procedure can't start

Definition 3.2.4 (Hybrid approaches)**.** Start from the null model, variables are added to the model sequentially, in analogy to forward selection. However, after adding each new variable, the method may also remove any variables that no longer provide an improvement in the model fit.

Important remark 49 (Pros/cons). This approach mimics the best subset selection while retaining the computational advantages of forward and backward stepwise selection

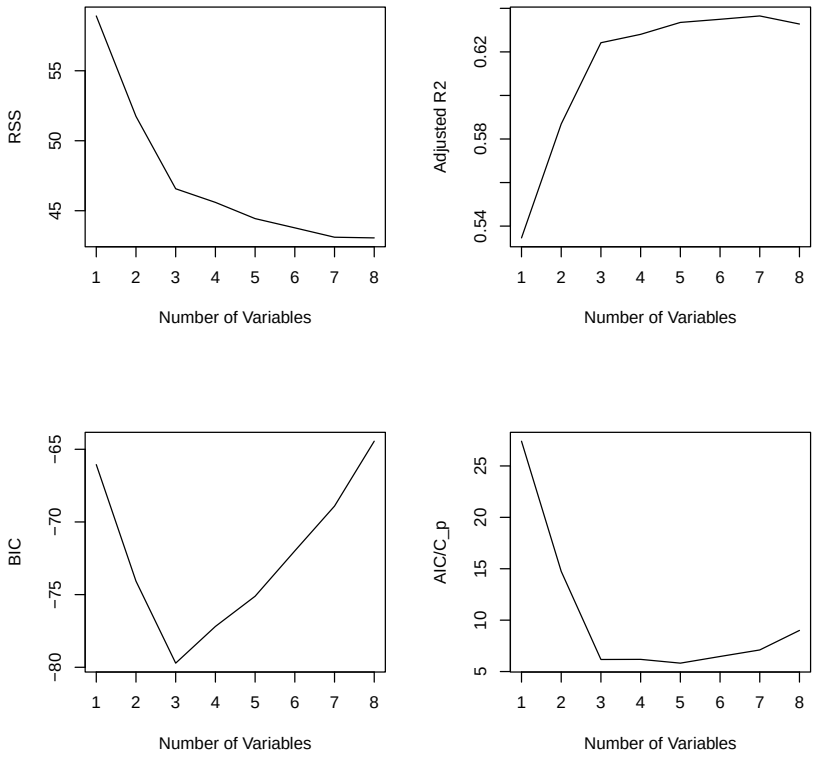
3.2.3 Exercise

Example 3.2.1. Using the prostate dataset:

1. perform model selection via best subset, forward selection, backward elimination and hybrid methods.

```
## Model Selection
library(leaps)

### Best subset selection
### --------------------
## The regsubsets() function (from the leaps library) performs
## best subset selection (works only on linear models) by
## identifying the best model that contains a given number of
## predictors, where best is quantified using RSS.  The syntax is
## the same as for lm(). The summary() command outputs the best
## set of variables for each model size.

out.bs <- regsubsets(lpsa ~ ., data=prostate) # method="exhaustive" is by default best sub
(sum.out.bs <- summary(out.bs))

## Subset selection object
## Call: eval(parse_only(code), envir = envir)
## 8 Variables  (and intercept)
##         Forced in Forced out
## lcavol      FALSE      FALSE
## lweight     FALSE      FALSE
## age         FALSE      FALSE
## lbph        FALSE      FALSE
## svi         FALSE      FALSE
## lcp         FALSE      FALSE
## gleason     FALSE      FALSE
## pgg45       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     " "
## 5  ( 1 ) "*"    "*"     "*" "*"  "*" " " " "     " "
## 6  ( 1 ) "*"    "*"     "*" "*"  "*" " " " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"

## An asterisk indicates that a given variable is included in the
## corresponding model. For instance, this output indicates that
## the best two-variable model contains only lcavol and
## lweight. Above models are nested but it does not need to be so.

## By default, regsubsets() only reports results up to
## the best eight-variable model. But the nvmax option can be used
## in order to return as many variables as are desired.

sum.out.bs$rsq
```

```
## [1] 0.5394320 0.5955040 0.6359499 0.6435561 0.6526150 0.6577801 0.6630054 0.66

## we see that the R2 statistic increases from 53.9%, when only
## one variable is included in the model, to almost 70%, when all
## variables are included. As expected, the R2 statistic increases
## monotonically as more variables are included.


## The optimum model under this criterion is a compromise
## influenced by the sample size, the effect sizes of the
## different predictors, and the degree of collinearity between
## them.  We can examine these to try to select the best overall
## model.

par(mfrow=c(2,2))
plot(sum.out.bs$rss,xlab = "Number of Variables", ylab="RSS",type="l")
plot(sum.out.bs$adjr2,xlab = "Number of Variables", ylab="Adjusted R2",type="l")
plot(sum.out.bs$bic,xlab = "Number of Variables", ylab="BIC",type="l")
plot(sum.out.bs$cp,xlab = "Number of Variables", ylab="AIC/C_p",type="l")
```



```
## we see that rss always go down with number of predictors (eg
## rss for 1 should be the rss for model with only lcavol
## adjusted R^2 and bic don't always  agree on a common chosen model:
## choosing the best model
which.max(sum.out.bs$adjr2)

## [1] 7

which.min(sum.out.bs$bic)

## [1] 3

which.min(sum.out.bs$cp)

## [1] 5

## The methods doesnt always agree on a common solution: if we
## look at adjusted R^2 we choose the model with 7 variables, at
## BIC with 3 (BIC is much parsimonious).

## finally can use the coef() function to see the coefficient
## estimates associated with the 7-predictor model.
print(coef(out.bs, 7), digits = 3)

## (Intercept)      lcavol     lweight         age        lbph         svi         lcp
##     0.49415     0.56955     0.61442    -0.02091     0.09735     0.75240    -0.10496

## Stepwise forward
## ----------------
out.fs <- regsubsets(lpsa ~ ., data = prostate, method = "forward")
summary(out.fs)

## Subset selection object
## Call: eval(parse_only(code), envir = envir)
## 8 Variables  (and intercept)
##         Forced in Forced out
## lcavol      FALSE      FALSE
## lweight     FALSE      FALSE
## age         FALSE      FALSE
## lbph        FALSE      FALSE
## svi         FALSE      FALSE
## lcp         FALSE      FALSE
## gleason     FALSE      FALSE
## pgg45       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " " "     " "
```

```
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " " "     " "
## 5  ( 1 ) "*"    "*"     "*" "*"  "*" " " " " "     " "
## 6  ( 1 ) "*"    "*"     "*" "*"  "*" " " " " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*" "     "*"

## Stepwise backward
## -----------------
out.be <- regsubsets(lpsa ~ ., data = prostate, method = "backward")
summary(out.be)

## Subset selection object
## Call: eval(parse_only(code), envir = envir)
## 8 Variables  (and intercept)
##         Forced in Forced out
## lcavol      FALSE      FALSE
## lweight     FALSE      FALSE
## age         FALSE      FALSE
## lbph        FALSE      FALSE
## svi         FALSE      FALSE
## lcp         FALSE      FALSE
## gleason     FALSE      FALSE
## pgg45       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: backward
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " " "     " "
## 5  ( 1 ) "*"    "*"     "*" "*"  "*" " " " " "     " "
## 6  ( 1 ) "*"    "*"     "*" "*"  "*" " " " " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*" "     "*"

### Stepwise hybrid
## ----------------
out.ha <- regsubsets(lpsa ~ ., data = prostate, method = "seqrep")
summary(out.ha)

## Subset selection object
## Call: eval(parse_only(code), envir = envir)
## 8 Variables  (and intercept)
##         Forced in Forced out
## lcavol      FALSE      FALSE
## lweight     FALSE      FALSE
## age         FALSE      FALSE
```

```
## lbph        FALSE      FALSE
## svi         FALSE      FALSE
## lcp         FALSE      FALSE
## gleason     FALSE      FALSE
## pgg45       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: 'sequential replacement'
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " " "     " "
## 5  ( 1 ) "*"    "*"     "*" "*"  "*" " " " " "     " "
## 6  ( 1 ) "*"    "*"     "*" "*"  "*" " " " " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     " "
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"
```

For this data, the best one-variable through eight-variable models are each identical for best subset, forward selection and backward selection (this doesn't need to be the case). The result is slightly different for the hybrid version.

2. choose the best model via Cross-Validation. Estimate the test error.

```
## ----------------------------
## Choose the best model via CV
## ----------------------------

## there's no predict for regsubset so we provide one
predict.regsubsets <- function(object, newdata, id, ...){
    ## extract the formula from the regsubset model/object,
    form <- as.formula(object$call[[2]])
    ## rearrange data matrix to return a design matrix (eg create
    ## dummys, the intercept of 1, given the formula and the
    ## data)
    mat <- model.matrix(form, newdata)
    ## extract the estimated coefficents of the model with id parameters(id is the number
    ## predictors, eg)
    ##  > coef(out.bs, id = 1)
    ## (Intercept)      lcavol
    ##      1.5073      0.7193
    coefi <- coef(object, id=id)
    ## to obtain the prediction multiply the data matrix for the
    ## regression coefficients, selecting the proper data
    mat[, names(coefi)] %*% coefi
}

## folds
k <- 5
p <- ncol(prostate) - 1
```

```r
set.seed(1234)
folds <- sample(1:k, nrow(prostate), replace = TRUE)
## matrix that for each fold of cross validation K and foreach
## number of parameters contains the prediction error
cv.errors <- matrix(NA, k, p, dimnames = list(NULL, paste0("M=",1:p)))

## cv main loop. now we write a for loop that performs
## cross-validation. In the jth fold, the elements of folds that equal
## j are in the test set, and the remainder are in the training
## set. We make our predictions for each model size (using our new
## predict() method), compute the test errors on the appropriate
## subset, and store them in the appropriate slot in the matrix
## cv.errors.

## again before making run all the loop set i = 1 and j = 1 and
## test the code
for (i in 1:k){
    train <- folds!=i
    test  <- !train
    db_train <- prostate[train, ]
    db_test <- prostate[test,]
    ## divide in train and validation set
    ## apply regsubset to data matrix X excludign
    mod <- regsubsets(lpsa ~ ., data = db_train)
    ## now calculate the error for each number of predictors
    for (j in 1:p){
        pred <- predict(mod, newdata = db_test, id=j)
        cv.errors[i,j] <- mean((db_test$lpsa - pred)^2)
    }
}

## Error in parse_only(code):  non trovo la funzione "parse_only"

## This has given us a 5 x 8 matrix, of which the (i, j)-th element
## corresponds to the test MSE for the ith cross-validation fold
## for the best j-variable model.

cv.errors # all the MSEs, now choose using the mean for each column

##      M=1 M=2 M=3 M=4 M=5 M=6 M=7 M=8
## [1,]  NA  NA  NA  NA  NA  NA  NA  NA
## [2,]  NA  NA  NA  NA  NA  NA  NA  NA
## [3,]  NA  NA  NA  NA  NA  NA  NA  NA
## [4,]  NA  NA  NA  NA  NA  NA  NA  NA
## [5,]  NA  NA  NA  NA  NA  NA  NA  NA

colMeans(cv.errors) # the smallest prediction error is obtained with 3 predictor

## M=1 M=2 M=3 M=4 M=5 M=6 M=7 M=8
##  NA  NA  NA  NA  NA  NA  NA  NA
```

```
plot(colMeans(cv.errors), type ='b') # b for plot point and lines

## Warning in min(x):  nessun argomento non-mancante al minimo;
si restituisce Inf
## Warning in max(x):  nessun argomento non-mancante al massimo;
si restituisce -Inf
## Error in plot.window(...):  i valori di 'ylim' devono essere
finiti
```

```
## We see that cross-validation selects a three-variable model. We
## now perform best subset selection on the full data set in order to
## obtain the three-variable model.
reg.best <- regsubsets(lpsa ~ ., data=prostate)
coef(reg.best, 3)

## (Intercept)      lcavol     lweight         svi
##  -0.7771566   0.5258519   0.6617699   0.6656666
```

3.3 Shrinkage/penalization

Remark 20 (Approach). We fit a model containing all p predictors but shrinking the estimates toward 0. By doing that the method increase the bias but reduces the variance of the estimates, ending often in a fit improvement and test error reduction. The two main methods are ridge and lasso.

3.3.1 Ridge

Definition 3.3.1 (OLS fitting)**.** In the standard least square fitting the estimates for $\beta_0, \ldots, \beta_p$ are obtained minimizing

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

Definition 3.3.2 (Ridge regression fitting)**.** The coefficients are estimated by minimizing a slightly different quantity:

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

where $\lambda \geq 0$ is a tuning parameter. In this minimization we have:

- the *RSS*: by making RSS small, ridge regression seeks as always coefficient estimates that fit the data well;
- the *shrinkage penalty* $\lambda \sum_{j=1}^{p} \beta_j^2$ composed of a norm of the vector coefficients and a tuning parameter. This term is small when $\beta_1, \ldots, \beta_p$ are close to zero, and so in the minimization has the effect of shrinking the estimates of β_j towards zero.

Important remark 50 (The parameter λ). The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates:

- when $\lambda = 0$ the penalty term has no effect, and ridge regression will produce the least squares estimates;
- when $\lambda \to \infty$ the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero.

In figure 3.1 the coefficients of the prostate dataset as λ increases.

Important remark 51 (Scaling is needed). The ridge regression (and the lasso as well) coefficient estimates can change substantially when multiplying a given predictor by a constant, because the modulus of the betas are directly considered in the minimization process.
In order to put all the different variable/scales on the same level, it is best to apply ridge regression after standardizing the predictors and making them have a standard deviation of one. As a result the final fit will not depend on the scale on which the predictors are measured.

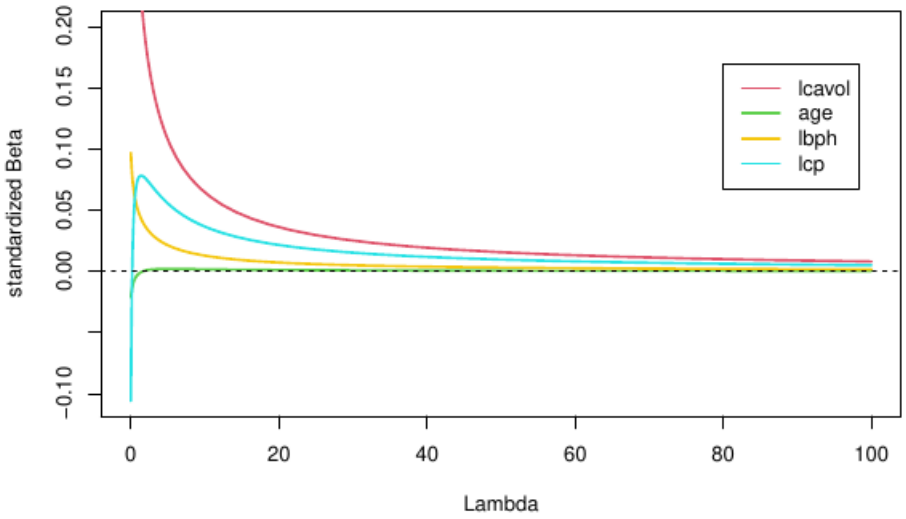


Figura 3.1: Ridge coefficients.

Important remark 52 (Pros/cons). Pros:

- permits estimation if $p \geq n$;
- works well even if the least squares estimates have high variance ($p \approxeq n$): as λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance (increased bias, btw);
- *computationally speaking* is fairly quick/doesn't imply major penalizations on estimation time: once fixed λ it's a standard minimization process. It has substantial computational advantages over best subset selection (compute one model for each λ, instead of 2^p models)

Cons:

- will *include all p predictors* in the final model: ridge *doesn't do variable selection* (the penalty $\lambda \sum_j \beta_j^2$ will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero unless $\lambda = \infty$); this could be a problem in interpretation if the number of variables p is quite large. For this issue we have lasso.

3.3.2 LASSO

Definition 3.3.3. The coefficients are estimated by minimizing a slightly different quantity:

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

Important remark 53. The lasso:

- is *similar to ridge*: $\lambda \geq 0$ is again the tuning penalization parameter to be choosen in CV, while the major difference is the *penalization term* where the β_j^2 term (ℓ_2 penalty) is substituted by $|\beta_j|$ (ℓ_1 penalty)[1];
- the penalization change forces some of the coefficient estimates to be exactly 0 when the tuning parameter λ is sufficiently large, so lasso *performs variable selection*

3.3.3 Final remarks

Important remark 54 (Comparing ridge and lasso). For:

- *interpretation/inference* lasso has a major advantage that produces simpler models involving only a subset of the predictors.
- *prediction*: there's no default better method between the two: cross-validation must be used to determine best approach on a particular dataset. In general
  - lasso generally performs better when a relatively small number of predictors have substantial coefficients, and the remaining predictors have low/near zero coefficients;
  - ridge will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.

Important remark 55 (Choosing λ). Ridge/Lasso will produce a different set of coefficient estimates for each value of λ: to implement them fully, we need select an optimal value for the tuning parameter λ.
We tipically do the estimates for several λ and then choose the best λ by crossvalidation:

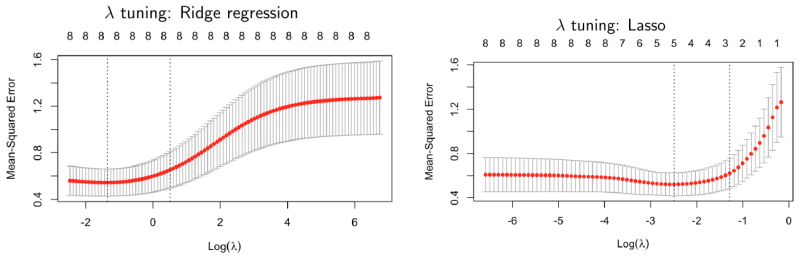
1. we choose a grid of λ values, do estimates, and compute the cross-validation error for each value of λ;
2. we then select the tuning parameter value for which the cross-validation error is smallest;
3. finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter.

In figure 3.2 the CV mean square error for different values of lambda for both ridge and lasso; in alto il numero di variabili con coefficienti non nulli (si vede che le covariate non diminuiscono con ridge mentre lo fanno con lasso).

3.3.4 Exercise

We will use the `glmnet` package in order to perform ridge regression and the lasso. The main function in this package is `glmnet`, which can be used to fit ridge regression models, lasso models, and more. We must pass in an `x` matrix as well as a `y` vector (we do not use the common formula syntax).

[1] Names come frome the fact that the ℓ_2 norm of a vector is $\sum_{i=1}^{n} x_i^2$, while ℓ_1 is $\sum_{i=1}^{n} |x_i|$.



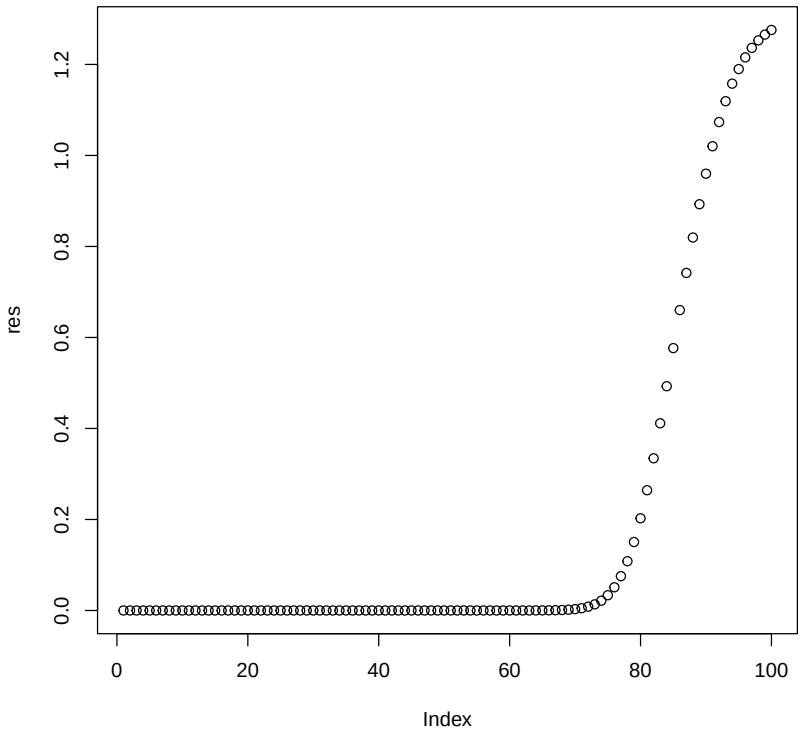
(a) (b)

Figura 3.2: Ridge and lasso tuning

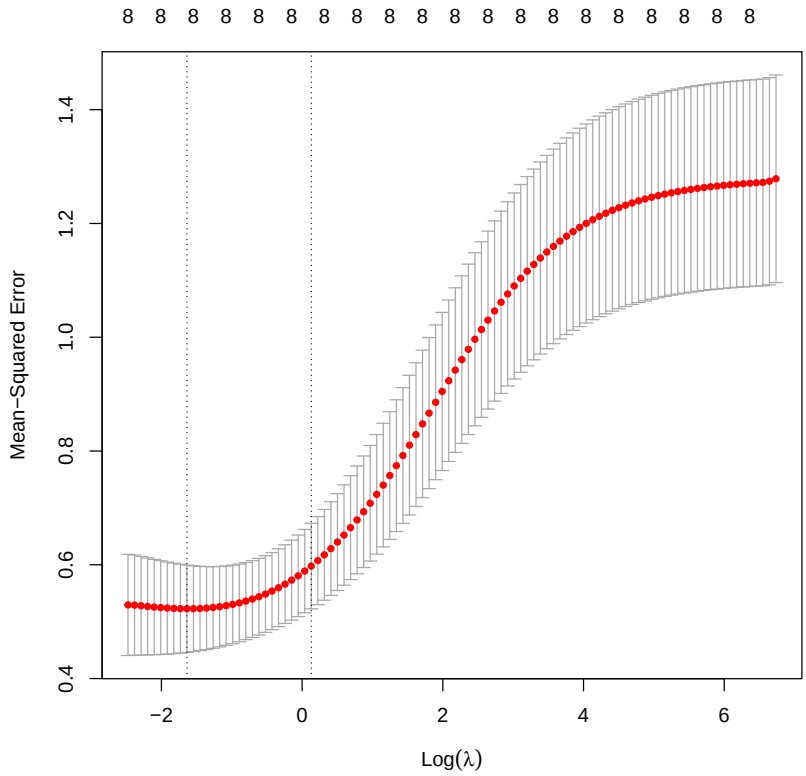
1. Perform regularization via ridge regression model; use cross-validation to choose the tuning parameter λ. Estimate the test error.

```
library(glmnet)

## Caricamento del pacchetto richiesto:  Matrix
## Loaded glmnet 4.1-8

## create the model matrix
x <- model.matrix(lpsa ~ ., data = prostate)[, -1]
y <- prostate$lpsa

## sequence of lambda we try. (unless the procedure tries 100 values
## of its choice)
grid <- 10^seq(10, -2, length = 100) # it's decreasing

## Estimate of ridge model (alpha = 0); the glmnet standardizes by
## default so that they are on the same scale (however the set of
## coefficients its returns is on the original scale).  To turn off
## this default setting, use the argument standardize=FALSE.
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)
names(ridge.mod)

##  [1] "a0"        "beta"      "df"        "dim"       "lambda"    "dev.ratio" "nulldev"
##  [9] "jerr"      "offset"    "call"      "nobs"

## once estimated (for each lambda), coef returns a vector of
## coefficient for each values of lambda
dim(coef(ridge.mod))

## [1]   9 100

## coef with the first lambda
ridge.mod$lambda[1] # the first lambda

## [1] 1e+10
```

```r
coef(ridge.mod)[,1] # tiny but not 0 (the first has a great lambda, lot penaliza

##  (Intercept)       lcavol      lweight          age         lbph          svi
## 2.478387e+00 8.260413e-11 1.340776e-10 3.019569e-12 1.642903e-11 1.813000e-10
##      gleason        pgg45
## 6.773326e-11 1.984884e-12

## coef with the first lambda
ridge.mod$lambda[50] # the first lambda

## [1] 11497.57

coef(ridge.mod)[,50] # tiny but not 0 (the first has a great lambda)

##  (Intercept)       lcavol      lweight          age         lbph          svi
## 2.477232e+00 7.182894e-05 1.165889e-04 2.625030e-06 1.428550e-05 1.576425e-04
##      gleason        pgg45
## 5.888743e-05 1.725644e-06

## coefs with the last lambda
ridge.mod$lambda[100]

## [1] 0.01

coef(ridge.mod)[,100] # low lambda: results similar to ls

##  (Intercept)       lcavol      lweight          age         lbph          svi
##  0.151011516  0.553882957  0.620449634 -0.020607196  0.095120620  0.750074993
##      gleason        pgg45
##  0.051957450  0.004273852

## if we do ols with lambda = 0 we get not the same results because
## of rounding

## Once estimated the model we can use the predict() to obtain
## coefficients for a new value of lambda (say 50); it's done by
## interpolation of closest coefficients lambda-near
predict(ridge.mod, s = 50, type = "coefficients")[1:9,]

##  (Intercept)       lcavol      lweight          age         lbph          svi
## 2.2285644901 0.0158698147 0.0259008542 0.0005438826 0.0031592531 0.0344937600
##      gleason        pgg45
## 0.0125369896 0.0003683045

## plotting della norma euclidea dei vettori
res <- apply(coef(ridge.mod)[-1, ],
             2,
             function(x) sum(x^2)) # euclidean distance
plot(res)
```




```
## as lambda decrease (from left to right) the norm increasese
## due to reduced penalization


## 10-fold CV for ridge regression
## -------------------------------
## cv.glmnet does cv and returns optimal values for lambda
n <- nrow(x)
set.seed(1234)
## id di training (test ottenuti sottraendoli)
train <- sample(1:n, ceiling(n/2))
test <- -train
cv.out <- cv.glmnet(x[train,], y[train], alpha = 0)
## behavior of the error for different value of lambda (log-zed)
plot(cv.out)
```




```
## at the top we have only 8, meaning that we're not dropping
## covariates, all retained in the model (being ridge)
## when small we are ols like, for increasing level we introduce
## bias in order to reduce the variance. this to get the smallest
## error. However best results here are in the first part

round(cv.out$lambda, 2) # the value tried
```

```
##   [1] 841.95 767.16 699.00 636.91 580.33 528.77 481.80 438.99 400.00 364.46 33
##  [14] 251.21 228.89 208.56 190.03 173.15 157.77 143.75 130.98 119.34 108.74  9
##  [27]  74.95  68.29  62.23  56.70  51.66  47.07  42.89  39.08  35.61  32.44  2
##  [40]  22.36  20.38  18.57  16.92  15.41  14.04  12.80  11.66  10.62   9.68
##  [53]   6.67   6.08   5.54   5.05   4.60   4.19   3.82   3.48   3.17   2.89
##  [66]   1.99   1.81   1.65   1.51   1.37   1.25   1.14   1.04   0.95   0.86
##  [79]   0.59   0.54   0.49   0.45   0.41   0.37   0.34   0.31   0.28   0.26
##  [92]   0.18   0.16   0.15   0.13   0.12   0.11   0.10   0.09   0.08
```

```
names(cv.out)
```

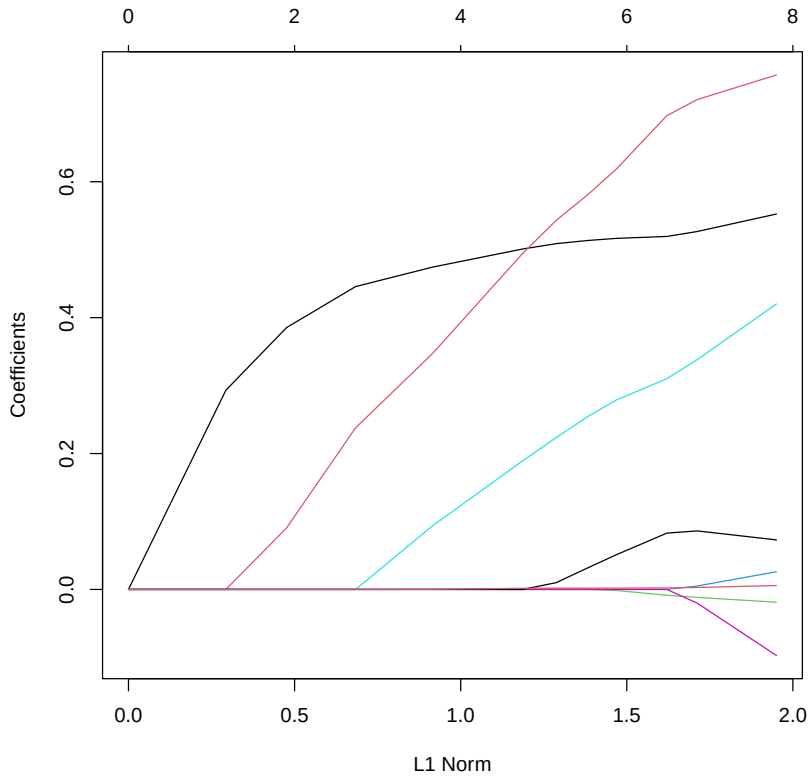
```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"       "nzero"
##  [8] "name"       "glmnet.fit" "lambda.min" "lambda.1se" "index"
```

```
## - lambda.min is the lambda for which we obtain a minimum error
## - lambda.1se return a lambda which error is largest but not
##   significantly different from the overall minimum (at most 1
##   standard error above)

## prof uses best lambda: let's save so we can use prediction using
## the optimal lambda
(best.lambda <- cv.out$lambda.min)

## [1] 0.194502

log(best.lambda) # where is on the plot

## [1] -1.637313

## predict can return a lot of stuff (regression coefficients,
## response etc). we need to choose what to output of regression
## model

## for y predictions none is needed other than things below
## otherwise look at type options (eg "coefficients")
pred.ridge <- predict(cv.out, # the estimated models
                      s = best.lambda, # optimal lambda to consider
                      newx = x[test, ], # not newdata, validation set
                      x = x[train, ], # training set sometimes needed
                      y = y[train]) # for returning to estimation to have precise
# test MSE associated with the best lambda
mean((y[test]-pred.ridge)^2)

## [1] 0.5881267

## finally, we obtain our ridge regression model on the full data
## set, using the value of lambda chosen by cross-validation, and examine
## the coefficient estimates
predict(cv.out,
        s = best.lambda,
        type = "coefficient")

## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) -0.701679435
## lcavol       0.445444435
## lweight      0.661092462
## age         -0.012850019
## lbph         0.033915259
## svi          0.406831058
## lcp         -0.023539376
## gleason      0.107551195
## pgg45        0.003797866
```

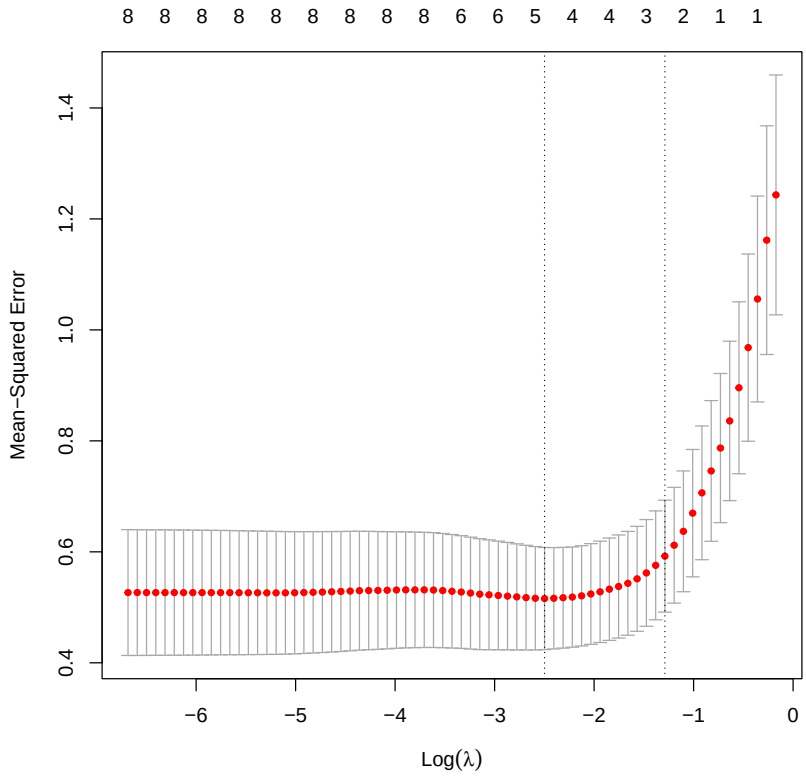
```
# none of the coefficients are zero: ridge regression does not
# perform variable selection

## Compare accuracy with OLS (s = 0 and exact = TRUE)
pred.ols <- predict(cv.out,
                    s = 0, # this is lambda = 0
                    newx = x[test,], x = x[train,], y = y[train],
                    exact = TRUE)
mean((y[test]-pred.ols)^2)

## [1] 0.5912257

## there is a difference, not great, but shrinking a little bit
## (lambda was not huge at all) return a smaller error in the
## validation set. So ridge regression with a wise choice of lambda
## can outperform least squares as well as the null model on the
## prostate data set.
```

2. Perform regularization via lasso regression model; use cross-validation to choose the tuning parameter λ. Estimate the test error. How many predictors are retained?

```
## Let's check lasso performance: everything equal except alpha = 1
out.lasso <- glmnet(x[train,], y[train], alpha=1, lambda=grid)
## With plot we can see how the coefficients change the value
## according to the L1 norm of the overall vector of betas
plot(out.lasso)

## Warning in regularize.values(x, y, ties, missing(ties), na.rm
= na.rm):  si riduce a valori unici di 'x'
```




```
## Choice of optimal lambda in 10-fold CV
## ----------------------------------------
set.seed(1234)
cv.out <- cv.glmnet(x[train,], y[train], alpha=1)
(best.lambda <- cv.out$lambda.min)

## [1] 0.0822596

plot(cv.out) # here the number of coefficients decreases
```




```
## check MSE
lasso.predict <- predict(cv.out,
                         s = best.lambda,
                         newx = x[test,],
                         x = x[train,],
                         y = y[train])
mean((y[test]-lasso.predict)^2)

## [1] 0.6162576

## MSE in the test is very similar the one of ridge regression

# The optimal lambda (chosen in CV) returns a model with
# 5 predictors out of 8 (intercetta esclusa)
(beta.hat.lasso <- predict(cv.out,
                           s = best.lambda,
                           type="coefficient"))

## 9 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -0.551095256
## lcavol       0.511154037
```

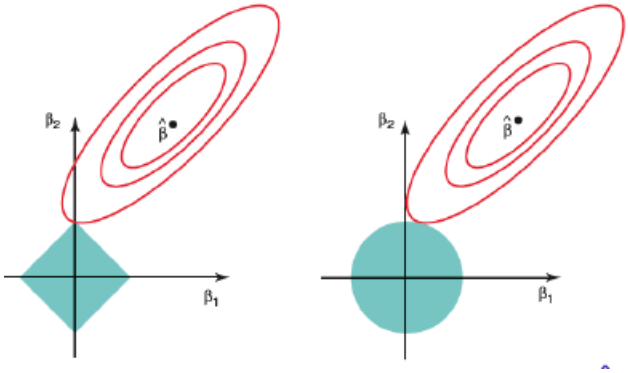


Figura 3.3: Ridge vs lasso1.

```
## lweight      0.561684111
## age          .
## lbph         .
## svi          0.238609720
## lcp          .
## gleason      0.020700893
## pgg45        0.001915265
```

3.3.5 Ridge and lasso as constrainted minimization

Important remark 56. We can think ridge and lasso as a minimization subject to constraint s. By doing this:

NB: bah secondo me non le chiede

- the selection feature of lasso
- the connection of ridge/lasso with best subset selection

will be clearer. In both ridge and lasso we want to find the vector β which minimizes the residual sum of square:

$$\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\}$$

subject to, respectively, to the following constraints:

$$\sum_{i=1}^{n} \beta_j^2 \leq s, \text{for ridge}$$

$$\sum_{i=1}^{n} |\beta_j| \leq s, \text{for lasso}$$

s can be seen as a kind of budget we can pay.

Important remark 57 (Ridge vs lasso in variable selection). In case of 2 variable, so we can visualize in the plane (figure 3.3):

- the least square solution is the center who minimize the paraboloid of RSS depending on β_1, β_2. can be seen above a concentrical ellipses of increasing RSS going in the outward direction;

- the ridge regression estimates (intersection between red and green circle, right figure) have the smallest RSS out of all points that lie within the circle defined by the circle $\beta_1^2 + \beta_2^2 = s$ (circle with radius s being) being $\beta_1^2 + \beta_2^2 \leq s$;

- lasso coefficient estimates (intersection between red circle and green diamond, left figure) have the smallest RSS out of all points that lie within the diamond defined by $|\beta_1| + |\beta_2| \leq s$ (it has vertexes in $(0, s), (0, -s), (s, 0), (-s, 0)$)

In this setup it's easier for lasso returning a solution where one of the coefficient is 0 (angles in the diamond) due to the very form of the diamond and the circle, while it's a bit more difficult when the intersection is between two circles/ellypses to go on.
Regarding the constraint/budget s, if

- is large (setting $\lambda = 0$ is like having an high budget) then the least squares solution falls within the budget;

- is small, the concentric least square/RSS will be tangent to the budget in one point that will be the solution; being the lasso solution with spigoli respect to the ridge (arrotondata) it's easier to match a corner of the budget where one of the two coefficient is 0

Important remark 58 (Connection between lasso/ridge and best subset). Best subset can be seen as

$$\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\}$$

subject to

$$\sum_{j=1}^{p} I(\beta_j \neq 0) \leq s$$

This amounts to finding a set of coefficient estimates such that RSS is as small as possible, subject to the constraint that no more than s coefficients can be nonzero (eg i can afford at most 6 predictors to be different from 0)

Remark 21. Finally straight lasso was the first version: generalization are available. *Group lasso* is the generalization which handles the categorical covariates.

3.3.6 Additiona notes on ridge and lasso*

bah secondo me non iede

3.3.6.1 Ridge regression estimation

The original expression can be rewritten in matrix form as follows:

$$\begin{aligned} RSS + \lambda \sum_{j=1}^{p} \beta_j^2 &= \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \sum_{j=1}^{p} \beta_j^2 \\ &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta} \\ &= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{y} + \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{X}\boldsymbol{\beta} + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta} \\ &= \mathbf{y}^T \mathbf{y} - 2\boldsymbol{\beta}^T \mathbf{X}^T \mathbf{y} + \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{X}\boldsymbol{\beta} + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta} \end{aligned}$$

In order to find $\hat{\beta}^T$ that minimizes such equation

$$\begin{aligned} \frac{\partial RSS + \lambda \sum_{j=1}^{p} \beta_j^2}{\partial \beta^T} &= -2\mathbf{X}^T \mathbf{y} + 2(\mathbf{X}^T \mathbf{X})\boldsymbol{\beta} + 2\lambda \boldsymbol{\beta} \\ &= 2(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})\boldsymbol{\beta} - 2\mathbf{X}^T \mathbf{y} \end{aligned}$$

putting this last $= 0$ we end with

$$\hat{\beta}^R = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

In the standard OLS regression the estimator are found like following

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

So in ridge regression the penalization makes $\lambda \mathbf{I}$ to appear: adding this simple λ on the diagonal of $\mathbf{X}^T \mathbf{X}$.
For the record $\mathbf{X}^T \mathbf{X}$ is the Gram matrix: if we're working with centered data this is the total sum of squares, the variability of X, the numerator of covariance matrix (or n times the covariance matrix).
If $\lambda = 0$ we have the OLS solution while if λ increases penalization plays a role. Just by adding a λ on the diagonal of $X^T X$ makes possible to reduce the variability ($X^T X$ increased diagonal is at the denominator, reducing the variance) of the estimates and allowing to have estimates even if $p \geq n$.
If λ is high the weight of the norm of beta vector is huge in the minimization and in the minimization we want to reduce the norm of the vector, that is, sends some coefficients toward zero (coefficients are no longer unbiased btw).

3.3.6.2 Beta interpretation in a simple fictious case

Remark 22. In this simple case more insight on how ridge and lasso plays on coefficient beta vector.

Important remark 59 (Setup and OLS solution). Let's consider a simple special case with $n = p$, and X a diagonal matrix with 1 on-diagonal and 0 off-diagonal elements. To simplify the problem further, assume also that we are performing regression without an intercept. With these assumptions the RSS to be minimized finding is $\beta_1, \ldots, \beta_p$:

$$RSS = \sum_{j=1}^{p} (y_i - \beta_j)^2$$

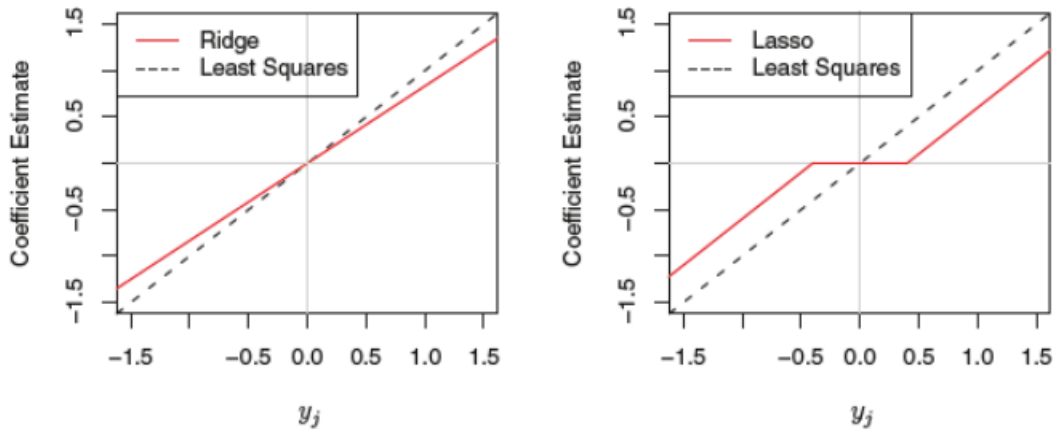


Figura 3.4: Ridge vs lasso2.

and in this case the least squares solution is given by

$$\hat{\beta}_j = y_j$$

Important remark 60 (Ridge solution). In ridge regression we have to minimize

$$\begin{aligned}\Phi &= RSS + \lambda \sum_{j=1}^{p} \beta_j^2 = \sum_{j=1}^{p} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \\ &= \sum_{j=1}^{p} y_j^2 - 2 \sum_{j=1}^{p} y_j \beta_j + \sum_{j=1}^{p} \beta_j^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \\ &= \sum_{j=1}^{p} y_j^2 - 2 \sum_{j=1}^{p} y_j \beta_j + (1+\lambda) \sum_{j=1}^{p} \beta_j^2\end{aligned}$$

For the optimization

$$\frac{\partial \Phi}{\partial \beta_j} = -2y_j + 2(1+\lambda)\beta_j = 0$$

We end in $\hat{\beta}_j^R = \frac{y_j}{1+\lambda}$. Considering that $\hat{\beta}_j^{OLS} = y_j$ then

$$\hat{\beta}_j^R = \frac{\hat{\beta}_j^{OLS}}{1+\lambda} \tag{3.1}$$

So the ols beta is moved toward zero by the $(1+\lambda)$ parameter

Important remark 61. For the lasso regression we have to minimize:

$$\begin{aligned}\Phi &= RSS + \lambda \sum_{j=1}^{p} |\beta_j| = \sum_{j=1}^{p} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \\ &= \sum_{j=1}^{p} y_j^2 - 2 \sum_{j=1}^{p} y_j \beta_j + \sum_{j=1}^{p} \beta_j^2 + \lambda \sum_{j=1}^{p} |\beta_j|\end{aligned}$$

In order to optimize

$$\frac{\partial \Phi}{\partial \beta_j} = -2y_j + 2\beta_j + \lambda \frac{\beta_j}{|\beta_j|} = 0$$

Now explore the different options:

- if $\beta_j > 0$ then $\frac{\beta_j}{|\beta_j|} = 1$ and

  $$\frac{\partial \Phi}{\partial \beta_j} = -2y_j + 2\beta_j + \lambda = 0$$

  thus

  $$\hat{\beta}_j^L = \frac{2y_j - \lambda}{2} = y_j - \frac{\lambda}{2}$$

  Given that $\beta_j > 0$ then $\hat{\beta}_j^L > 0$ and so $y_j - \frac{\lambda}{2} > 0 \implies y_j > \frac{\lambda}{2}$. Thus if $y_j > \frac{\lambda}{2}$ then $\hat{\beta}^L = \hat{\beta}^{OLS} - \frac{\lambda}{2}$

  TODO: non chiarissimo qua il perché ma a lezione non ha spiegato $\beta_j > 0 \implies \hat{\beta}_j^L > 0$

- if $\beta_j < 0 \implies \frac{\beta_j}{|\beta_j|} = -1$ thus

  $$\frac{\partial \Phi}{\partial \beta_j} = -2y_j + 2\beta_j - \lambda = 0$$

  and

  $$\hat{\beta}_j^L = \frac{2y_j + \lambda}{2} = y_j + \frac{\lambda}{2}$$

  For the same reason of above $\hat{\beta}_j^L < 0$ and thus $y_j < -\frac{\lambda}{2}$; in this case $\hat{\beta}_j^L = \hat{\beta}_j^{OLS} + \frac{\lambda}{2}$

- in the remaining case ($|y_j| \leq \lambda$) $\beta_j = 0$ and so $\hat{\beta}_j^L$

TODO: non chiarissimo manco qui ma lei non è stata cristallina

Thus in this setting the lasso estimates take the form

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & \text{if } y_j > \lambda/2 \\ y_j + \lambda/2 & \text{if } y_j < -\lambda/2 \\ 0 & \text{if } |y_h| \leq \lambda/2 \end{cases} = \begin{cases} \hat{\beta}^{OLS} - \lambda/2 & \text{if } y_j > \lambda/2 \\ \hat{\beta}^{OLS} + \lambda/2 & \text{if } y_j < -\lambda/2 \\ 0 & \text{if } |y_h| \leq \lambda/2 \end{cases} \tag{3.2}$$

Remark 23. All in all equations 3.1 and 3.2, that is the comparation of lasso/ridge with OLS, are represented in figure (fig 3.4):

- in ridge regression, each least squares coefficient estimate is shrunken *by the same proportion* (divided by $1 + \lambda$);
- the lasso shrinks each least squares coefficient towards zero by a *constant amount*, $\lambda/2$; the least squares coefficients that are less than $\lambda/2$ in absolute value are shrunken entirely to 0.

Lasso sets to 0 the already smallest coefficients while reducing the others; ridge diminish them the same way.

3.4 Dimension reduction methods

Remark 24. These approaches:

- transforms (by linear combination) the p predictors into a set of $M < p$ predictors/components, and then fit a least squares model using the transformed variables.
  Key idea of these approaches is that often a small number of components/linear combination of the starting variables suffices to explain most of the variability in the data as well as the relationship with the response;
- are not a variables selection method: we create *new* variables;
- there are many methods: we focus on principal component regression

3.4.1 Principal component regression

Definition 3.4.1 (Principal component regression)**.** The procedure:

- we find $M < p$ linear combinations of our original p predictors

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j$$

  for some constant $\phi_{1m}, \dots, \phi_{pm}$, with $m = 1, \dots, M$.
  The ϕ_{jm} are choosed and Z_m calculated using principal component analysis (by constructing the first M principal components, $Z_1, \dots, Z_M$);
- we fit the linear regression model using the new principal components and standard regression, eg ols

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \varepsilon_i, \quad i = 1, \dots, n$$

Important remark 62 (Variable standardization). When performing PCR, (unless all variable are already on the same scale, eg kg) it is generally recommended to standardize, in order to have all variables on the same scale. Otherwise high-variance variables will tend to play a larger role in the principal components obtained (and measurement scale would have an effect on the final PCR model)

Important remark 63 (Performance vs standard regression). If:

- the constant of the transformation $\phi_{1m}, \dots, \phi_{pm}$ are chosen wisely;
- the directions in which $X_1, \dots, X_p$ show most variation and are syntesized by the method are associated with Y (eg PCA only looks/uses X, does not look at Y);

dimension reduction approaches *can outperform least squares* regression on the original $X_1, \dots, X_p$, if the increased bias is mitigated by reduced variance/overfitting. For the second reason (there's no direct link with X and Y in PCA) it's useful to look at PCA in the context of CV/test error on prediction.

Important remark 64 (Nuber of components). Regarding M:

- as more principal components are used in the regression model, the bias decreases, but the variance increases (see below bias/variance tradeoff);
- when $M = p$ then PCR amounts simply to a least squares fit (con fattori comunque trasformati linearmente)
- the number of principal components M is typically chosen by cross-validation.

Important remark 65 (Number of components and bias variance tradeoff). The bias variance tradeoff is given by the constraint that arise since

$$\sum_{m=1}^{M} \theta_m z_{im} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{jm} x_{ij} = \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{jm} x_{ij} = \sum_{j=1}^{p} \beta_j x_{ij}$$

where

$$\beta_j = \sum_{m=1}^{M} \theta_m$$

This constraint on β_j has the potential to bias the coefficient estimates. However, in situations where p is large relative to n, selecting a value of $M \ll p$ can significantly reduce the variance of the fitted coefficients.

3.4.2 Exercise

Reduce the dimensionality of the data via Principal Components Regression (PCR) to the prostate data, in order to predict lpsa. Choose the optimal number M of PCs to be retained via 10-fold CV. How many components should be retained? Evaluate its test set performance.

There is a nice package `pls` to do all this, but we will be doing the same thing even by standard methods, because the package works only with continue response so in case we have a classification problem we need to extract principal components and use them in any model/learner we want by hand (and we need to know how to do that normally).

3.4.2.1 Principal components regression

Principal components regression (PCR) can be performed using the pcr() function, which is part of the pls library. syntax similar to lm, with a few additional options.

- scale=TRUE has the effect of standardizing each predictor, prior to generating the principal components, so that the scale on which each variable is measured will not have an effect.
- setting validation="CV" causes pcr to compute the ten-fold cv error for each possible value of M , the number of principal components used.

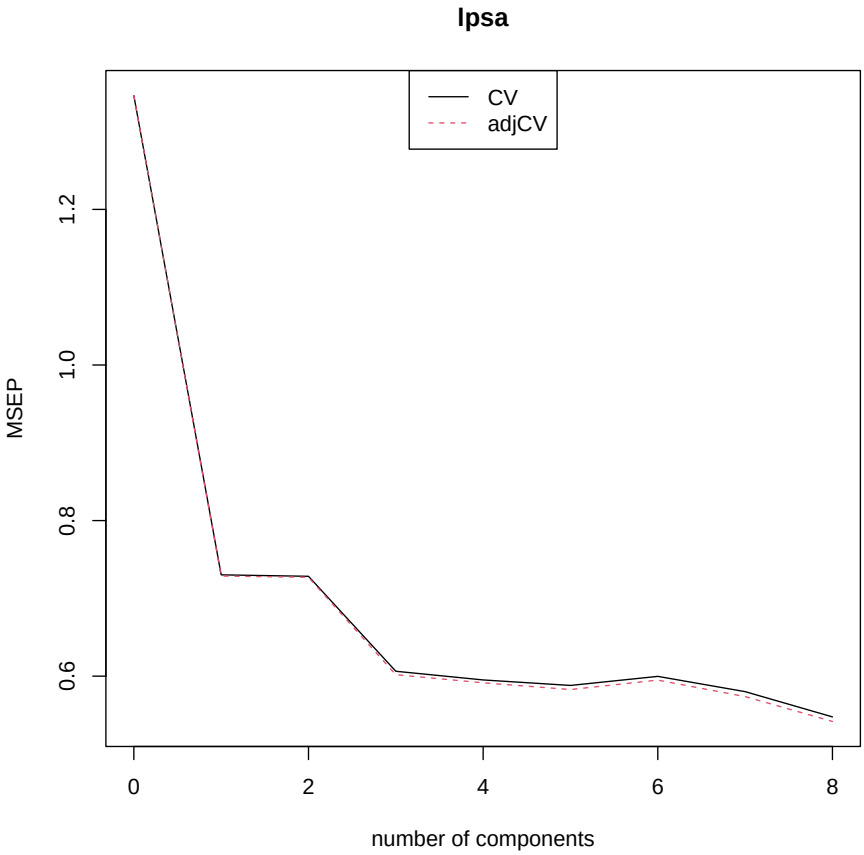
```
library(pls)
```

```
##
## Caricamento pacchetto:  'pls'
## Il seguente oggetto è mascherato da 'package:stats':
##
##     loadings
```

```
set.seed(1234)
pcr.fit <- pcr(lpsa ~ .,
               data = prostate,
               scale = TRUE, # specify since it's not default
               validation ="CV")

summary(pcr.fit)
```

```
## Data:  X dimension: 97 8
##  Y dimension: 97 1
## Fit method: svdpc
## Number of components considered: 8
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## CV            1.16   0.8545   0.8534   0.7786   0.7714   0.7669   0.7744   0.7616
## adjCV         1.16   0.8536   0.8527   0.7758   0.7690   0.7634   0.7713   0.7574
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       42.01    62.61    74.81    82.71    88.75    94.28    97.56   100.00
## lpsa    47.04    47.67    58.61    59.45    60.73    61.66    64.21    66.34
```

```
## one can also plot the cross-validation scores using the
## validationplot() function. Using val.type="MSEP" will cause the
## cross-validation MSE to be plotted.
validationplot(pcr.fit, val.type="MSEP", legendpos = "top")
```



In the ouput of summary:

- Fit method: dice quale metodo (spectral decomposition o SVD decomposition, qui svd). Spectral decomposition can only be applied to square matrix while svn can be carried out for any matrix
- Validation: tells the metrics used for error, here root mean squared error (RMSE); in order to obtain the usual MSE, we must square this quantity.
- the CV score is provided for each possible number of components, ranging from M = 0 onwards (with 8 components it's basically linear regression with new variables since there is no dimension reduction)
  We see that the smallest cross-validation error occurs when $M = 8$ components are used. This amounts to simply performing least squares, because when all of the components are used in PCR no dimension reduction occurs.
  However, from the plot we also see that the cross-validation error is roughly the same when only three component are included in the model. This suggests that a model that uses just a small number of components might suffice.
- `summary` function also provides the percentage of variance explained in the predictors and in the response using different numbers of components.

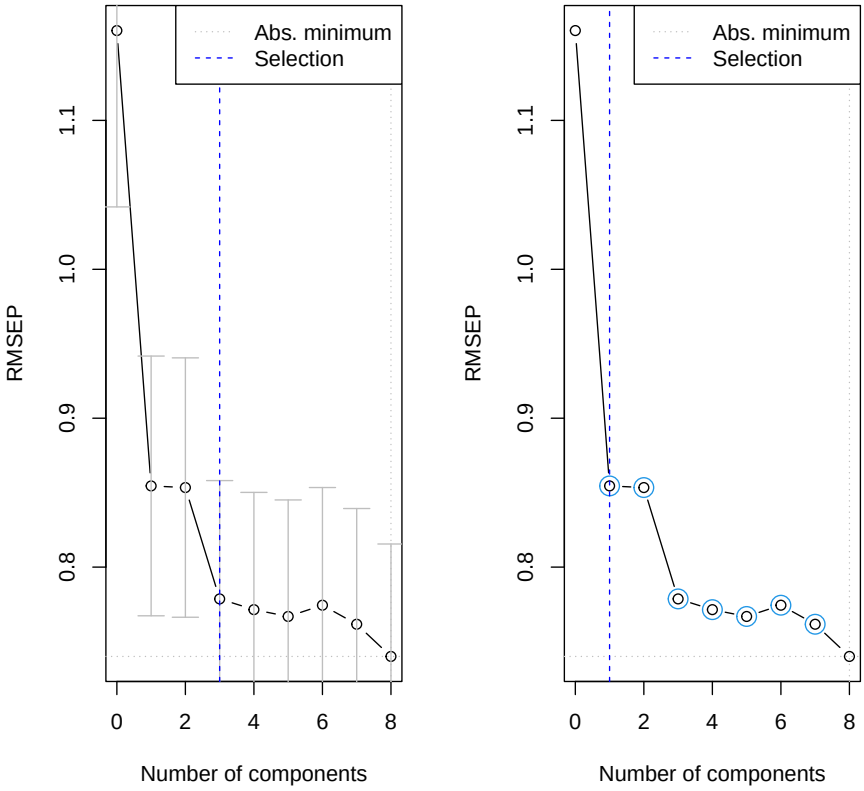
Briefly, we can think of this as the amount of information about the predictors or the response that is captured using M principal components. For example, setting M = 1 only captures 42.01% of all the variance, or information, in the predictors. In contrast, using $M = 6$ increases the value to 94.28%. If we were to use all $M = p = 8$ components, this would increase to 100%.

In order to choose the optimal number of components we use `selectNcomp` which has two strategies implemented:

- the `onesigma` heuristic: chooses the model with fewest components that is still less than one standard error away from the overall best model;

- the second strategy (`randomization`) employs a permutation approach, and basically tests whether adding a new component is beneficial at all. It is implemented backwards, again taking the global minimum in the crossvalidation curve as a starting point, and assessing models with fewer and fewer components: as long as no significant deterioration in performance is found (by default on the $\alpha = 0.01$ level), the algorithm continues to remove components.

```
par(mfrow = c(1,2))
ncomp.onesigma <- selectNcomp(pcr.fit, method = "onesigma", plot = TRUE)
ncomp.permut <- selectNcomp(pcr.fit, method = "randomization", plot = TRUE)
```



In the plot above we see that the minimum error is reached for 8 components, but

- if we look at the standard error (left graph) for three components the error is not different from the solution with 8 (and it is the solution with least components having this feature so this is choosen)
- in the permutation approach one component is choosen

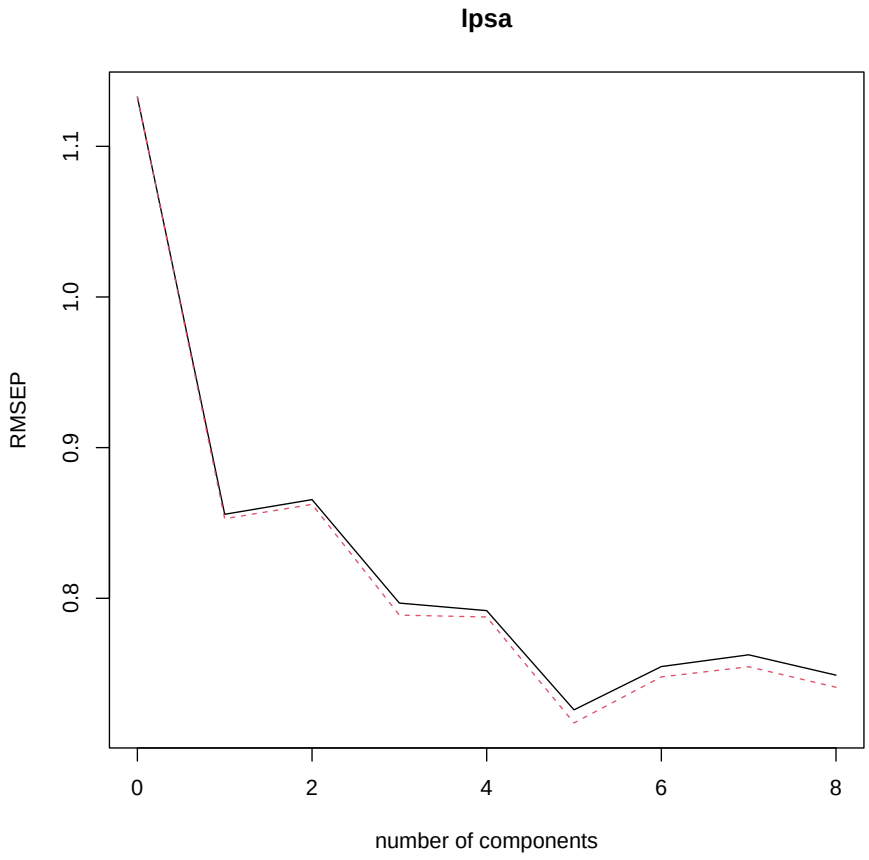
3.4.2.2 Test error estimate of the PCR: Training + Validation set

We now perform PCR on the training data and evaluate its test set performance.

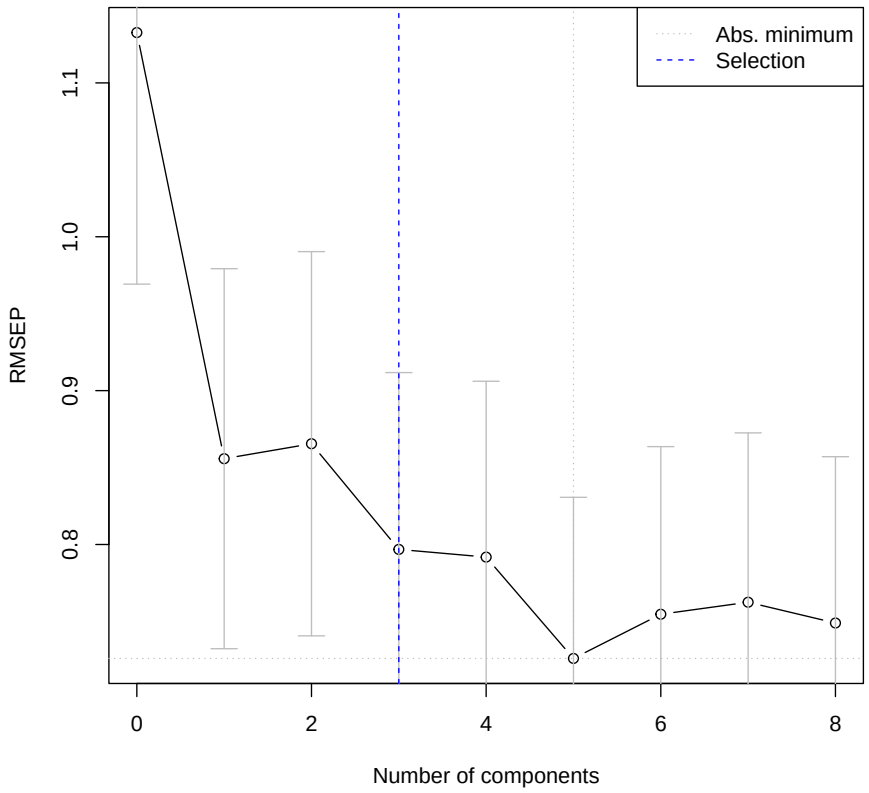
As seen for opther cases if we work with standardized data in the training set we must standardize in the test set using the parameter of the training set (here the function handle it I guess)

```
## Validation set approach ####
set.seed(1234)
n <- nrow(prostate)
train <- sample(1:n, ceiling(n/2))
test <- -train
```

```
set.seed(1234)
pcr.out <- pcr(lpsa ~ .,
               data = prostate,
               subset = train,
               scale = TRUE,
               validation = "CV")
validationplot(pcr.out)
```




```
## here the minimum is observed for 5 components but we apply
## selectNcomp with onesigma to see if we can reduce them
selectNcomp(pcr.out, method = "onesigma", plot = TRUE)
```




```
## [1] 3

## quindi scegliamo tre components con ncomp = 3
pred.pcr <- predict(pcr.out, prostate[test, 1:8], ncomp=3)
mean((prostate$lpsa[test]-pred.pcr)^2)

## [1] 0.5802516
```

This test set MSE is competitive with the results obtained using ridge regression and the lasso. However, as a result of the way PCR is implemented, the final model is more difficult to interpret because it does not perform any kind of variable selection or even directly produce coefficient estimates.

3.4.2.3 PCR via eigen

Now we see the same shit with standard principal component; at first by using the eigen() function to find the eigen vectors of the correlation matrix of X. Here as in what follows there is nothing random.

```
## PCR via eigen
## -------------
## extract eigenvectors from correlation matrix of training set
x.pcs <- eigen(cor(prostate[train, 1:8]))$vectors
## train and test sets
train.x <- prostate[train, 1:8]
test.x <- prostate[-train, 1:8]
# standardize the test set according to the param in the training set
test.std <- test.x
for (j in 1:ncol(test.x)){
    test.std[,j] <- (test.x[,j] - mean(train.x[,j]))/sd(train.x[,j])
}

## we use the first three eigenvectors (which are the loading of our
## principal components) to project training and test dataset into the
## new components space and to fit a model in the new space
x.train <-scale(train.x,T,T) %*% x.pcs[,1:3]
x.test <-as.matrix(test.std) %*% x.pcs[,1:3]

## create the final db for estimation
df.pcs <- data.frame(y= c(prostate$lpsa[train], prostate$lpsa[-train]),
                     rbind(x.train,x.test))

## now we estimate the model in the train set
out.lm <- lm(y ~ ., data = df.pcs[1:length(train), ])
## eval prediction in the test set
y.hat <- predict(out.lm, newdata = df.pcs[-c(1:length(train)),])
## compute error
mean((df.pcs$y[-c(1:length(train))]-y.hat)^2)

## [1] 0.5802516
```

3.4.2.4 PCR via svd

Alternatively, by using the svd() function to find the right singular vectors of matrix X:

```
## Alternatively, with SVD
## scale the train dataset
x.std <- scale(prostate[train,1:8],TRUE,TRUE)
## calculate the svd of the standardized train data
svd.x <- svd(x.std)
## extract the first three columns (eigenvectors)
x.pcs <- svd.x$v[,1:3]
## project the train and test
xx.train <- x.std %*% x.pcs
xx.test <- as.matrix(test.std) %*% x.pcs

## below the same as previously viewed
```

```
df.svd <- data.frame(y=c(prostate$lpsa[train], prostate$lpsa[-train]),
                     rbind(xx.train, xx.test))
out.svd <- lm(y ~ ., data=df.svd[1:length(train),])
yy.hat <- predict(out.svd, newdata=df.svd[-c(1:length(train)),])
mean((df.svd$y[-c(1:length(train))]-yy.hat)^2)

## [1] 0.5802516
```

3.4.3 Appendix on principal components analysis (PCA)*

NB: Again, secondo me non chiede

Important remark 66 (PCA). It:

- is an unsupervised approach, since it involves only a set of features $X_1, \ldots, X_p$, and no associated response Y.
- allow us to summarize a large set of correlated variables with a smaller number of representative variables that collectively explain most of the variability in the original set.
- is based on the idea is that each of the n observations lives in p-dimensional space (but not all of these dimensions are equally interesting): each dimension found by PCA is a linear combination of the p features;
- it's non random procedure, it's deterministic

Remark 25. The first principal component of a set of features $X_1, X_2, \ldots, X_p$:

- is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \ldots \phi_{p1}X_p$$

  which has the largest variance. By normalized linear combination, we mean that

$$\sum_{j=1}^{p} \phi_{j1}^2 = 1$$

- the elements $\phi_{11}, \ldots, \phi_{p1}$ are called *loadings* of the first principal component;
- the loadings make up the *principal component loading vector*, $\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$.

Given a $n \times p$ data set X, how do we compute the first principal component?

1. since we are only interested in variance, we assume that each of the variables in X has been centered to have mean zero
2. then we then look for the linear combination of the sample feature values of the form

$$z_1 = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \ldots \phi_{p1}x_{ip}$$

that has largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. In other words, the first principal component loading vector solves the optimization problem

$$\max_{\phi_{11},\ldots,\phi_{p1}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} x_{ij} \right)^2 \right\}, \quad \text{subject to } \sum_{j=1}^{p} \phi_{j1}^2 = 1$$

Such expression can be written as $\frac{1}{n} \sum_{i=1}^{n} z_{i1}^2$. Since $\frac{1}{n} \sum_{i=1}^{n} x_{ij} = 0$ the average of the $z_{11}, \ldots, z_{n1}$ will be zero as well

3. we refer to $z_{11}, \ldots, z_{n1}$ as the *scores of the first principal component*: we are maximizing the sample variance of the n values of z_{i1}

4. In matrix form:

$$\text{Var}\left[Z_1\right] = \text{Var}\left[\phi_{11} X_1 + \ldots + \phi_{p1} X_p\right] = \text{Var}\left[X\phi_1\right] = \phi_1^T \text{Var}\left[X\right] \phi_1$$

Denoting $\text{Var}\left[X\right] = \Sigma$, we want to find vector ϕ_1 such that

$$\max_{\phi_1} \left\{ \phi_1^T \Sigma \phi_1 \right\}, \quad \text{subject } \phi_1^T \phi_1 = 1$$

We can restate the problem in the constrained optimization framework based on Lagrange multipliers by defining the function:

$$\Psi = \phi_1^T \Sigma \phi_1 - \lambda_1 (\phi_1^T \phi_1 - 1)$$

and by looking for the vector ϕ_1 that maximizes it. This optimization problem can be solved by differentiating Ψ with respect to ϕ_1 and equating to 0 all the partial derivatives:

$$\begin{aligned} \frac{\partial \Psi}{\partial \phi_1^T} &= 2\Sigma\phi_1 - 2\lambda_1\phi_1 = 0 \\ &= \Sigma\phi_1 - \lambda_1\phi_1 = 0 \end{aligned}$$

Therefore,

$$\Sigma\phi_1 = \lambda_1\phi_1$$

Last identity show the relationship between the *eigenvalues* and the *eigenvectors* of the *covariance matrix* Σ: λ_1 is an eigenvalue of Σ and ϕ_1 the corresponding eigenvector.

If we multiply both sides by ϕ_1^T, because of the unit norm constraint we obtain:

$$\phi_1^T \Sigma \phi_1 = \lambda_1 \phi_1^T \phi_1 = \lambda_1$$

λ_1 exactly coincides with the variance of Z_1 , i.e. with the quantity we want to maximize. Therefore, in order to derive the linear combination having the largest variance, we simply need to consider the *largest eigenvalue* of Σ and ϕ_1 will be the *corresponding eigenvector*.

The loading vector ϕ_1 with elements $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$ defines a direction in feature space along which the data vary the most. If we project the n data points $x_1, \ldots, x_n$ onto this direction, the projected values are the principal component scores $z_{11}, \ldots, z_{n1}$ themselves. After the first principal component Z_1 of the features has been determined, we can find the second principal component Z_2 . The second principal component is the linear combination of $X_1, \ldots, X_p$ that has maximal variance out of all linear combinations that are orthogonal to Z_1 , therefore $\phi_2^T \phi_1 = \phi_1^T \phi_2 = 0$.

In other words, the constrained maximization problem now consists in maximizing, with respect to ϕ_2 , the following function:

$$\Psi = \phi_2^T \Sigma \phi_2 - \lambda_2(\phi_2^T \phi_2 - 1) - \lambda 3 \phi_2^T \phi_1$$

We take the first derivative w.r.t. ϕ_2 and set it equal to zero:

$$\frac{\partial \Psi}{\partial \phi_2^T} = 2\Sigma\phi_2 - 2\lambda_2\phi_2 - \lambda_3\phi_1 = 0$$

By multiplying both sides by ϕ_1^T we obtain

$$2\phi_1^T \Sigma \phi_2 - 2\lambda_2 \phi_1^T \phi_2 - \lambda_3 \phi_1^T \phi_1 = 0$$

As ϕ_1^T is an eigenvector of Σ, $\phi_1^T \Sigma = \lambda_1 \phi_1^T$. Therefore $\lambda_3 = 0$.

The problem we need to solve is then $\Sigma\phi_2 - \lambda_2\phi_2 = 0$ or, equivalently, $\Sigma\phi_2 = \lambda_2\phi_2$.

λ_2 is an eigenvalue of Σ and ϕ_2 is the corresponding eigenvector. As we are looking for the linear combination having the largest variance not accounted for by Z_1 we will choose the second largest eigenvalue of Σ and the corresponding eigenvalue. $Z_2 = X\phi_2$ is the second principal component.

The above process can be continued for all principal components. We will derive as many principal components as the observed variables. In general, the k-th PC of X is $Z_k = X\phi_k$, and $V(Zk) = \lambda_k$ where λ_k is the k-th largest eigenvalue of Σ, and ϕ_k is the corresponding eigenvector

It is possible to show that the first and the second principal components are *uncorrelated*. The same holds for any pair of principal components.

3.4.3.1 Principal components from SVD

Alternatively, principal components can be found by using the singular value decomposition of matrix X:

$$X = UDV^T$$

where

- U is an $n \times p$ orthogonal matrix ($U^T U = I_p$) whose columns u_j are called the *left singular vectors*;
- V is a $p \times p$ orthogonal matrix ($V^T V = I_p$) with columns v_j called the *right singular vectors*;
- D is a $p \times p$ diagonal matrix, with diagonal elements $d_1 \geq d_2 \geq \ldots \geq d_p \geq 0$ known as the *singular values*

The columns of $Z = UD$ are called the *principal components* of X.

Capitolo 4

Tree based methods

4.1 Basic trees

Trees stratifies the predictor space into a number of consequent/simple regions. Can be applied to both regression (regression tree) and classification (classification tree) problems: to make prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs

4.1.1 Regression tree

Definition 4.1.1 (Tree)**.** It consists of a series of splitting rules to classify units in groups and provide a group prediction for a quantitative outcome.

Example 4.1.1. An example in fig 4.1 for prostate lpsa using gleason and age:

- the top split (most important) assigns observations having gleason < 6.5 to the left branch and observations with a gleason ≥ 6.5 to the right
- patients with gleason < 6.5 are further subdivided by age: if they are younger than 67 the predicted lpsa is 1.530, otherwise equal to 2.441. the predicted lpsa for these latter units is given by mean lpsa among the subject with gleason < 6.5, respectively with age < 67 and ≥ 67
- patients with gleason ≥ 6.5 go to the right, where the predicted log PSA is 2.896 .

Important remark 67 (Tree working). We have that:

- the tree stratifies the patients into into J distinct and non-overlapping regions $R_1, \ldots, R_J$ of predictor space (possible values of $X_1, \ldots, X_p$). In the examples are (fig 4.2):

$$R_1 = \{X | gleason \geq 6.5\}$$
$$R_2 = \{X | gleason < 6.5, age < 67\}$$
$$R_2 = \{X | gleason < 6.5, age \geq 67\}$$

  the regions R_1, R_2, R_3 are known as *terminal nodes* or *leaves* of the tree;

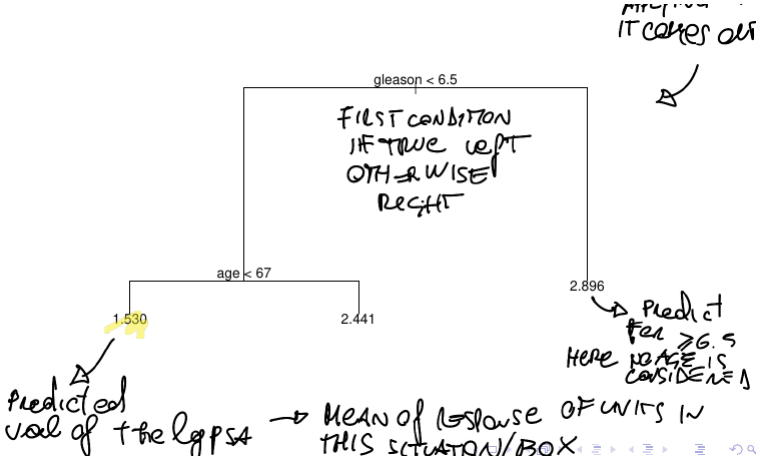


Figura 4.1: Tree.

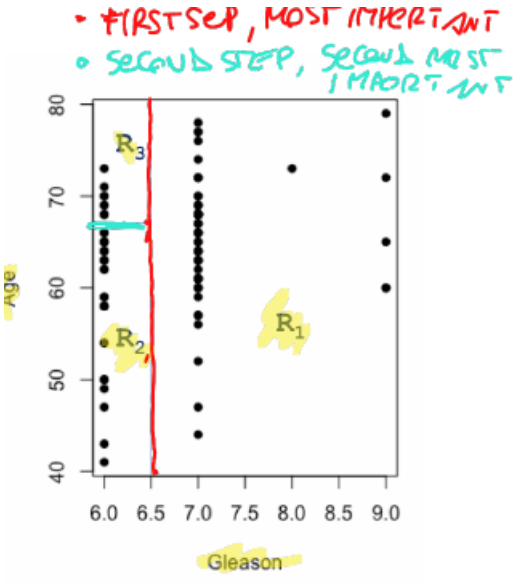


Figura 4.2: Tree.

- points where the predictor space is split are called *internal nodes* (gleason < 6.5 and age < 67 are); upper internal nodes (here gleason) are the most important;

- for every observation that falls into the region R_j, the tree makes the same prediction, which is the mean of the response values for the training observations in R_j.

Important remark 68 (Tree construction). To construct the tree (in the framework of regression):

- the goal is to find the boxes $R_1, \dots, R_J$ that minimize the

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  where y_i is the observed value and $\hat{y}_{R_j}$ is is the mean response for the training observations within the j-th box.

- to do that, since considering every possible partition of the feature space into J boxes is computationally infeasible, a *recursive binary splitting* approach is taken

- in particular, we select to split using the predictor X_j and the cutpoint s (splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$) that leads to the *greatest possible reduction in RSS*.
  So for any j and s, we define the pair of half-planes

$$R_1 = \{X|X_j < s\}, \quad R_2 = \{X|X_j \geq s\}$$

  and we seek the values of j and s that minimize the sum of the RSS in the two subgroups

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

  where $\hat{y}_{R_1}$ is the mean response for the training observations in $(R_1(j,s)$, $\hat{y}_{R_2}$ is the mean response for the training observations in $(R_2(j,s)$

- we then repeat the process starting from the two previously identified region, looking for the best predictor and best cutpoint to minimize RSS within each sub region considered;

- the process continues until all the terminal nodes are composed of 1 observation or, better, a stopping criterion is met (eg we may continue until no region contains more than 5 observations).

Remark 26. The algorithm is

- *top-down*: it begins at the top of the tree (where all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

- *greedy*: at each step of the tree-building, best split is made considering training data and possible condition available at that particular step (rather than looking ahead and picking a split that will lead to a better tree in some future step).

Important remark 69 (Pros/cons). Pros:

- trees are *easy to interpret* (can be displayed graphically) and *explain* to people even to non expert (especially if they are small)
- no assumptions
- somewhat perform variable selection (displaying the most important variables for splitting)
- "model"/display interaction between variable in a simple way

Cons

- if not properly handled the resulting tree might be too complex and could *overfit* data.
- can be very *sensible to sample*/variable estimator: a small change in the data can cause a large change in the final tree.
  Tree semplification helps here: often a smaller tree with fewer splits (that is, fewer regions $R_1, \dots, R_J$) might lead to lower variance and better interpretation (at the cost of a little bias). To do that we can use both:
  1. a stricter *stopping criteria* (eg set 10 units instead of 5 in the ending leaf)
  2. tree *pruning*, we grow a free tree T_0 and then cut/remove the leves that are not particularly useful.
- generally predictive accuracy isn't as good as some other regression and classification approaches.
  However by *aggregating* many decision trees ((bagging and random forest)) the predictive performance of trees can be substantially improved

Important remark 70 (Pruning). We consider a sequence of subtrees of a given tree and a *cost-complexity function* to be minimized, depending on nonnegative tuning parameter α: the cost complexity to be minimized is

$$\text{cost complexity} = \underbrace{\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2}_{RSS} + \alpha\,|T| \quad \text{is as small as possible}$$

where:

- $|T|$ indicates the number of terminal nodes of the tree T (so its it's complexity);
- R_m is the rectangle corresponding to the m-th terminal node
- $\hat{y}_{R_m}$ is the predicted response associated with R_m , i.e. the mean of the training observations in R_m .

- the tuning parameter α (similar to ridge and lasso λ) handle penalization for complexity:
  - if $\alpha = 0$ we don't penalize and choose the more complex tree
  - if α increases there is a price to pay for having a tree with many terminal nodes, $\alpha |T|$ will tend to be minimized for a smaller subtree and thus we will tend to choose simpler trees (so the bias increase and the variance decrease)

  Again, since the final aim is prediction, α is choosen in cross validation

Important remark 71 (Building a regression tree algorithm). We:

1. use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observation;
2. apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees as a function of α;
3. use K-fold cross-validation to choose α. That is divide the training observation into k-folds: foreach $k = 1, \ldots, K$
   (a) repeat steps 1 and 2 on all but the k-th fold of the training data;
   (b) evaluate the mean squared prediction error on the data in the left-out k-th fold, as a function of α

   Then average the results for each value of α and pick α to minimize the average error
4. Return the subtree from step 2 that corresponds to the chosen value of α

4.1.1.1 Exercise

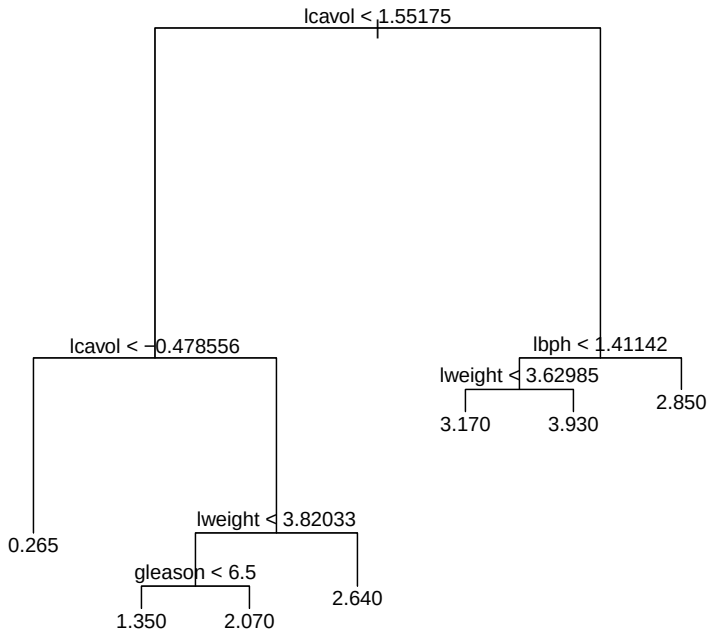
In R there are two libraries for trees: `tree` and `rpart`, here we use `tree`. Now

1. Fit a classification tree in order to predict chd using all variables and estimate the test error via validation set approach.

```
library(lbdatasets)
library(tree)

## Regression trees
n <-nrow(prostate)
set.seed(1234)
train <- sample(1:n, ceiling(n/2))

## grow the tree on train observation
tree_pros <- tree(lpsa ~ .,
                  data = prostate,
                  subset = train)
## A summary will tell the formula, the variables used in tree
```

```
## construction, the number of terminal nodes, residual mean deviance
## (overeall residual sum of square / (number of units - terminal
## nodes))
summary(tree_pros)

##
## Regression tree:
## tree(formula = lpsa ~ ., data = prostate, subset = train)
## Variables actually used in tree construction:
## [1] "lcavol"  "lweight" "gleason" "lbph"
## Number of terminal nodes:  7
## Residual mean deviance:  0.2413 = 10.13 / 42
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.19200 -0.35850  0.07017  0.00000  0.29270  0.82980

## Let's see the tree ascii form
print(tree_pros, digits = 3)

## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 49 60.300 2.410
##    2) lcavol < 1.55175 28 25.800 1.770
##      4) lcavol < -0.478556 5  1.410 0.265 *
##      5) lcavol > -0.478556 23 10.600 2.100
##       10) lweight < 3.82033 14  5.280 1.760
##         20) gleason < 6.5 6  1.900 1.350 *
##         21) gleason > 6.5 8  1.610 2.070 *
##       11) lweight > 3.82033 9  1.090 2.640 *
##    3) lcavol > 1.55175 21  8.370 3.250
##      6) lbph < 1.41142 12  3.290 3.550
##       12) lweight < 3.62985 6  0.986 3.170 *
##       13) lweight > 3.62985 6  0.557 3.930 *
##      7) lbph > 1.41142 9  2.580 2.850 *

## the tree starts with the root having 49 units, we have the deviance
## and the predicted outcome is 2.4 which is the same as the overall
## mean
mean(prostate$lpsa[train])

## [1] 2.405669

## the first split is done for lcavol < 1.55175 (or > 1.55); under
## 1.55175 we have 28 units, with a deviance of 26 and if we stop here
## we have 1.8 as prediction.
## a further split is applying lcavol < -0.478556, and since we have a
## * at the end, this indicates this is a terminal node
```

```
## to plot the tree
plot(tree_pros)
text(tree_pros, digits = 3)
```




```
## prediction and error of the regression tree
yhat <- predict(tree_pros,
                newdata = prostate[-train,])
mean((prostate$lpsa[-train]-yhat)^2)

## [1] 1.087141
```

2. prune the tree and estimate again the test error.

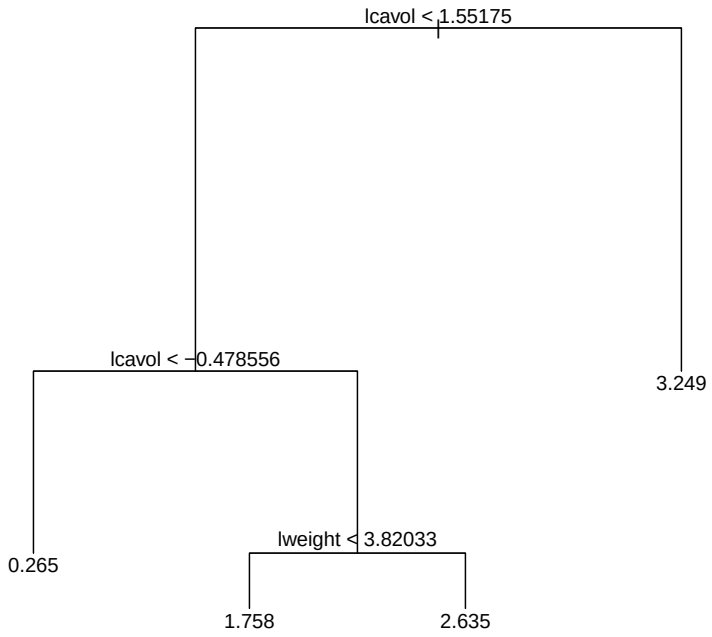
```
## Pruning of the regression tree
## cv.tree performs cross validation pruning
set.seed(1234)
cv_prostate <- cv.tree(tree_pros,    # our tree
                       K = 5,        # n of folds
                       FUN = prune.tree) # pruning function for
                                         # regression (minimizes mse)
cv_prostate
```

```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1] 28.77525 30.25754 30.25754 28.76382 28.85240 42.90463 62.92109
##
## $k
## [1]      -Inf  1.752423  1.769232  2.498378  4.213196 13.850177 26.110924
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
## size is the number of terminal nodes (eg fully grown has 7
## terminal ndes)
## for different size of the tree the function returns the value of
## the (cost complexity) function we're trying to optimize
## deviance is the thing we want to optimize
## k is value of cost complexity (the more, the more we penalize for
## complexity)

## how to decide best no. of terminal nodes: looking at the deviance
## the minimum is the tree with 4 final nodes
(best_tn <- cv_prostate$size[which.min(cv_prostate$dev)])
```

```
## [1] 4
```

```
## prune accordingly to the best number of terminal nodes using
## prune.tree
pruned_prostate <- prune.tree(tree_pros, best = best_tn)
## the tree is much smaller/shorter
plot(pruned_prostate)
text(pruned_prostate)
```




```
## accuracy of the pruned tree
pruned_yhat <- predict(pruned_prostate,
                       newdata=prostate[-train,])
mean((prostate$lpsa[-train]-pruned_yhat)^2)

## [1] 1.017454
```

So here the pruned tree has a better performance with a test MSE of 1.017 (compared to the full regression tree one of 1.087)

4.1.2 Classification trees

When the outcome is categorical we have a classification tree and things are little different

- to make prediction we use majority vote: the predicted class in one leaf is the modal one among the training set
- considering a classification problem with $k = 1, \ldots, K$ groups, construction is similar to regression tree (use recursive binary splitting) but instead of RSS we try to minimize:

– *classification error rate*: the fraction of the training observations in a region that do not belong to the most common class

$$E = 1 - \max_k(\hat{p}_k)$$

where $\hat{p}_k$ represents the proportion of training observations in the m-th region/terminal node that are from the k-th class.
This index however is not optimal/sufficiently *sensible* for tree growing (overall measures that does not look at *leaves purity*), so we use the following which are better suited

– *gini index*: defined as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

is a measure (in)*purity* (a small value indicates that a node contains predominantly observations from a single class) of the m-th node. if all the $\hat{p}_{mk}$ are close to zero (and one is almoest 1) the index take a small value, which is want we search for (we want to minimize the Gini).

– *entropy*: similar to gini is defined as

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

here again, it is possible to show the entropy will take on a value near zero if the $\hat{p}_{mk}$ are all near zero or near one

Example 4.1.2 (Two classes example)**.** In case of two-classes-outcome, if p is the proportion of the second class, the three measures defined as:

$$\begin{aligned} E &= 1 - \max(p, 1-p) \\ G &= 2p(1-p) \\ D &= -p \log p - (1-p)\log(1-p) \end{aligned}$$

and depicted in figure 4.3

Example 4.1.3 (Example on why classification error rate is not sensitive)**.** Eg in a two-class problem with 400 observations in each class (denote this by (400, 400)). Suppose we have to choose between two splits:

- one possible split created one nodes (300, 100) and the other node (100, 300); the error made here from the classification tree is overall $200/800 = 0.25$ (100 hundred of missclassification from the first region, 100 from the second);
- a second possible split created nodes (200, 400) and (200, 0); again here the error made is 0.25.

Both splits produce a misclassification rate of 0.25, but the second split produces a pure node and is probably preferable (using the missclassification error this is lost however).

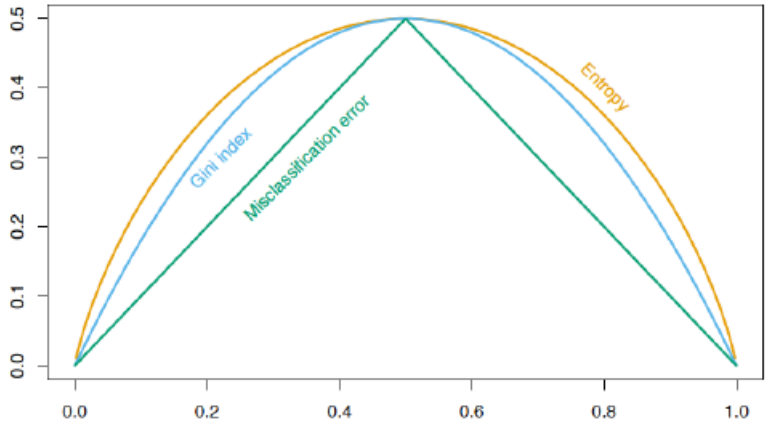


Figura 4.3: Tree.

Important remark 72. In general/real life application:

- when **building** a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate (furthermore, they are both differentiable, and hence more amenable to numerical optimization)
- when **pruning** the tree the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

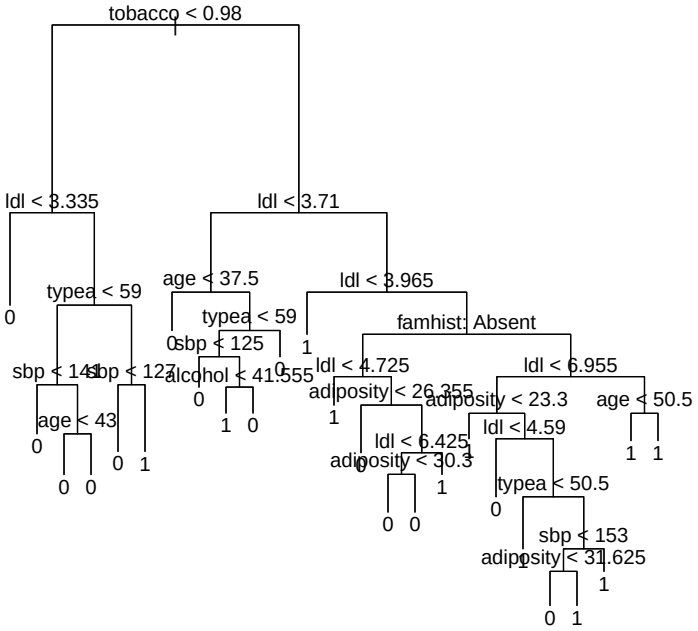
4.1.2.1 Exercise

- Fit a classification tree in order to predict chd using all variables and estimate the test error via validation set approach.

```
## Classification trees
n <- nrow(SAheart)

## For classification trees, the response MUST be a factor otherwise R
## will treat it as a regression tree.
## DON'T FIT THE TREE ON THE NUMERICAL RESPONSE, IT DOES NOT RETURN
## ANY WARNING: we convert as factor below

x <- SAheart[,-ncol(SAheart)]
y <- SAheart[,ncol(SAheart)]
heart <- data.frame(chd = as.factor(y), x)

### Accuracy of the fully grown tree (validation set approach)
set.seed(1234)
train <- sample(1:n, ceiling(n/2), replace = FALSE)
heart_test <- heart[-train,]
```

```
tree_heart <- tree(chd ~ ., heart, subset=train)
summary(tree_heart)

##
## Classification tree:
## tree(formula = chd ~ ., data = heart, subset = train)
## Variables actually used in tree construction:
## [1] "tobacco"   "ldl"       "typea"     "sbp"       "age"       "alcohol"   "
## Number of terminal nodes:  25
## Residual mean deviance:  0.5347 = 110.1 / 206
## Misclassification error rate: 0.1212 = 28 / 231

tree_heart

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##    1) root 231 291.200 0 ( 0.6753 0.3247 )
##      2) tobacco < 0.98 92  71.250 0 ( 0.8696 0.1304 )
##        4) ldl < 3.335 38   0.000 0 ( 1.0000 0.0000 ) *
##        5) ldl > 3.335 54  57.210 0 ( 0.7778 0.2222 )
##         10) typea < 59 40  26.010 0 ( 0.9000 0.1000 )
##           20) sbp < 141 23   0.000 0 ( 1.0000 0.0000 ) *
##           21) sbp > 141 17  18.550 0 ( 0.7647 0.2353 )
##             42) age < 43 9  12.370 0 ( 0.5556 0.4444 ) *
##             43) age > 43 8   0.000 0 ( 1.0000 0.0000 ) *
##         11) typea > 59 14  19.120 1 ( 0.4286 0.5714 )
##           22) sbp < 127 8   8.997 0 ( 0.7500 0.2500 ) *
##           23) sbp > 127 6   0.000 1 ( 0.0000 1.0000 ) *
##      3) tobacco > 0.98 139 191.500 0 ( 0.5468 0.4532 )
##        6) ldl < 3.71 37  38.630 0 ( 0.7838 0.2162 )
##         12) age < 37.5 10   0.000 0 ( 1.0000 0.0000 ) *
##         13) age > 37.5 27  32.820 0 ( 0.7037 0.2963 )
##           26) typea < 59 22  28.840 0 ( 0.6364 0.3636 )
##             52) sbp < 125 9   6.279 0 ( 0.8889 0.1111 ) *
##             53) sbp > 125 13  17.940 1 ( 0.4615 0.5385 )
##              106) alcohol < 41.555 8   8.997 1 ( 0.2500 0.7500 ) *
##              107) alcohol > 41.555 5   5.004 0 ( 0.8000 0.2000 ) *
##           27) typea > 59 5   0.000 0 ( 1.0000 0.0000 ) *
##        7) ldl > 3.71 102 140.800 1 ( 0.4608 0.5392 )
##         14) ldl < 3.965 5   0.000 1 ( 0.0000 1.0000 ) *
##         15) ldl > 3.965 97 134.400 1 ( 0.4845 0.5155 )
##           30) famhist: Absent 43  56.770 0 ( 0.6279 0.3721 )
##             60) ldl < 4.725 11  14.420 1 ( 0.3636 0.6364 ) *
##             61) ldl > 4.725 32  38.020 0 ( 0.7188 0.2812 )
##              122) adiposity < 26.355 9   0.000 0 ( 1.0000 0.0000 ) *
##              123) adiposity > 26.355 23  30.790 0 ( 0.6087 0.3913 )
##                246) ldl < 6.425 13  14.050 0 ( 0.7692 0.2308 )
##                  492) adiposity < 30.3 6   8.318 0 ( 0.5000 0.5000 ) *
```

```
##                  493) adiposity > 30.3 7   0.000 0 ( 1.0000 0.0000 ) *
##                247) ldl > 6.425 10  13.460 1 ( 0.4000 0.6000 ) *
##            31) famhist: Present 54  71.190 1 ( 0.3704 0.6296 )
##              62) ldl < 6.955 39  53.830 1 ( 0.4615 0.5385 )
##               124) adiposity < 23.3 7   5.742 1 ( 0.1429 0.8571 ) *
##               125) adiposity > 23.3 32  44.240 0 ( 0.5312 0.4688 )
##                 250) ldl < 4.59 6   0.000 0 ( 1.0000 0.0000 ) *
##                 251) ldl > 4.59 26  35.430 1 ( 0.4231 0.5769 )
##                   502) typea < 50.5 6   0.000 1 ( 0.0000 1.0000 ) *
##                   503) typea > 50.5 20  27.530 0 ( 0.5500 0.4500 )
##                    1006) sbp < 153 15  19.100 0 ( 0.6667 0.3333 )
##                      2012) adiposity < 31.625 9   6.279 0 ( 0.8889 0.1111 ) *
##                      2013) adiposity > 31.625 6   7.638 1 ( 0.3333 0.6667 ) *
##                    1007) sbp > 153 5   5.004 1 ( 0.2000 0.8000 ) *
##               63) ldl > 6.955 15  11.780 1 ( 0.1333 0.8667 )
##                126) age < 50.5 6   7.638 1 ( 0.3333 0.6667 ) *
##                127) age > 50.5 9   0.000 1 ( 0.0000 1.0000 ) *

## in these nodes deviance is a sort of entropy (n_i * log(proportions))
## if a node has 1 and 0 for probabilities, the node
## is pure in the sense that all the observation has the same group
plot(tree_heart)
text(tree_heart, pretty = 0) # pretty=0 allows to display the labels
```




```
                                        # of the categorical variables

## this tree is very rich, lot of terminal nodes. let's see MSE and
## confusion matrix ?predict.tree, by class we predict the response
## (highest posterior probability, ties splitted at random)
yhat_heart <- predict(tree_heart, newdata=heart_test, type="class")
table(yhat = yhat_heart, y = heart_test$chd)

##     y
## yhat  0  1
##    0 93 39
##    1 53 46

mean(yhat_heart != heart_test$chd)

## [1] 0.3982684
```

Performance is not that great. our tree could be to grown/overfitted

- Prune the tree and estimate again the test error.

```
## Procedure to prune is the same of regression tree (CV)
set.seed(1234)
## we use prune.misclass by optimizing for misclassification error
## rate
cv_heart <- cv.tree(tree_heart, FUN = prune.misclass)

## Error in eval(expr, p):  oggetto 'heart' non trovato

(best_size <- cv_heart$size[which.min(cv_heart$dev)])

## Error in eval(expr, envir, enclos):  oggetto 'cv_heart' non
trovato

pruned_heart <- prune.misclass(tree_heart, best = best_size)

## Error in eval(expr, p):  oggetto 'best_size' non trovato

plot(pruned_heart)

## Error in eval(expr, envir, enclos):  oggetto 'pruned_heart'
non trovato

text(pruned_heart, pretty = 0)

## Error in eval(expr, envir, enclos):  oggetto 'pruned_heart'
non trovato

## Confusion matrix: performance increase
yhat_pruned <- predict(pruned_heart,newdata=heart_test,type="class")

## Error in eval(expr, envir, enclos):  oggetto 'pruned_heart'
non trovato

table(yhat = yhat_pruned, y = heart_test$chd)

## Error in eval(expr, envir, enclos):  oggetto 'yhat_pruned'
non trovato

mean(yhat_pruned != heart_test$chd)

## Error in h(simpleError(msg, call)):  errore durante la valutazione
dell'argomento 'x' nella selezione di un metodo per la funzione
'mean':  oggetto 'yhat_pruned' non trovato

## with more leaves performance should decrease (being not best)
pruned_heart2<-prune.misclass(tree_heart, best=15)
yhat2<-predict(pruned_heart2,newdata = heart_test,type="class")
mean(yhat2!=heart_test$chd)

## [1] 0.3419913
```

4.1.3 Trees vs linear models

Regression and classification trees have a very different flavor from the more classical approaches for regression and classification. In particular,

- linear regression assumes a model of the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

- regression trees assume a model of the form

$$f(X) = \sum_{m=1}^{M} c_m \cdot I(X \in R_m)$$

  where $R_1, \ldots, R_m$ represent a partition of the feature space and c_m eg is mean in the regression case.

Which model is better depends on the problem at hand: if there is a highly non-linear and complex relationship between the features and the response, then decision trees may outperform classical approaches. Eg in figure 4.4

- in the top row: a two-dimensional classification example in which the true decision boundary is linear (left is analyzed with linear model, right with a tree)
- bottom row: here the true decision boundary is non-linear

4.2 Bagging

4.2.1 Introduction

Remark 27 (Idea/usage). Bagging (usage of *b*ootstrap for *agg*regation of results):

- takes origin from the fact that, given a set of n *independent* observations $Z_1, \ldots, Z_n$ each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n; that is averaging a set of observations reduces variance;
- is a general-purpose procedure *used to* reducing variability of a learning methods (tree or others) to enhance predictions;
- the *idea* of bagging is, instead of having a single learner, to build several independent learners by using bootstraps (samples will be independent since sampling is done with replacement);
- is mainly *focused* on prediction, rather than interpretation, where a single tree/method is simpler)

Definition 4.2.1 (Algorithm)**.** Steps are:

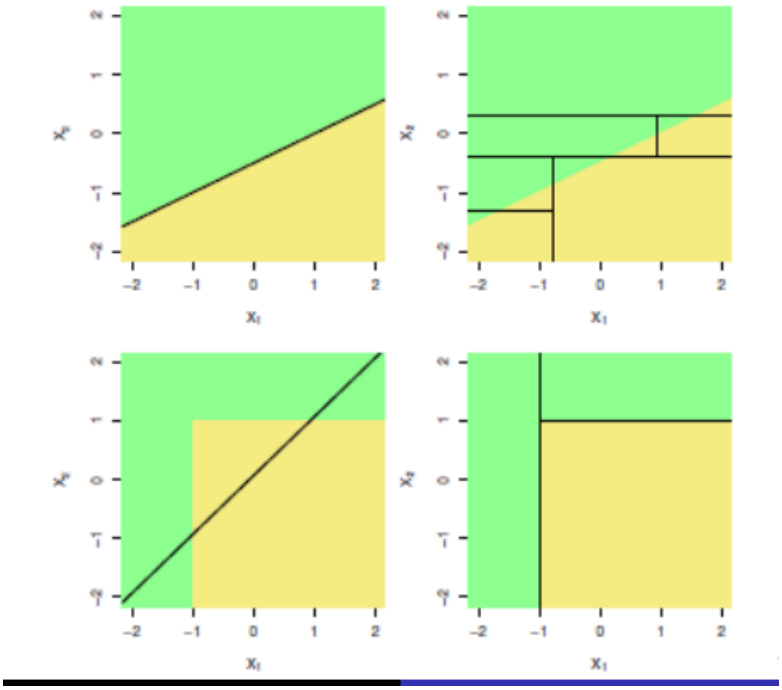


Figura 4.4: Tree.

- *generate* B different *bootstrapped training data sets*: the number of trees B is not a critical parameter with bagging (using a very large value of B will not lead to overfitting being the samples independent);
- *train the learner* on the b-th bootstrapped training set in order to get the estimate the tree $\hat{f}^{*b}(x)$.
  the trees are grown *till the end and not pruned*. Hence each individual tree will have high variance but low bias; averaging these B trees will reduces the variance
- we *average all the predictions from the bootstrap trees*, to obtain a single bagging prediction:

$$\hat{f}_{bag} = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

  In case Y is qualitative we can take a majority vote (predicted class is the modal one among the bagging trees)

Important remark 73 (Bagging performance and number of trees). Bagging has been demonstrated to give *impressive improvements in prediction accuracy.*
In figure 4.5 *the test error tends to stabilize with the number of trees* (estimated in a sample different from the testing from which the bootstrap procedure started). If we use few trees the error is very instable (with fully grown trees variability is very high); but as long as number of tree used increases, variability

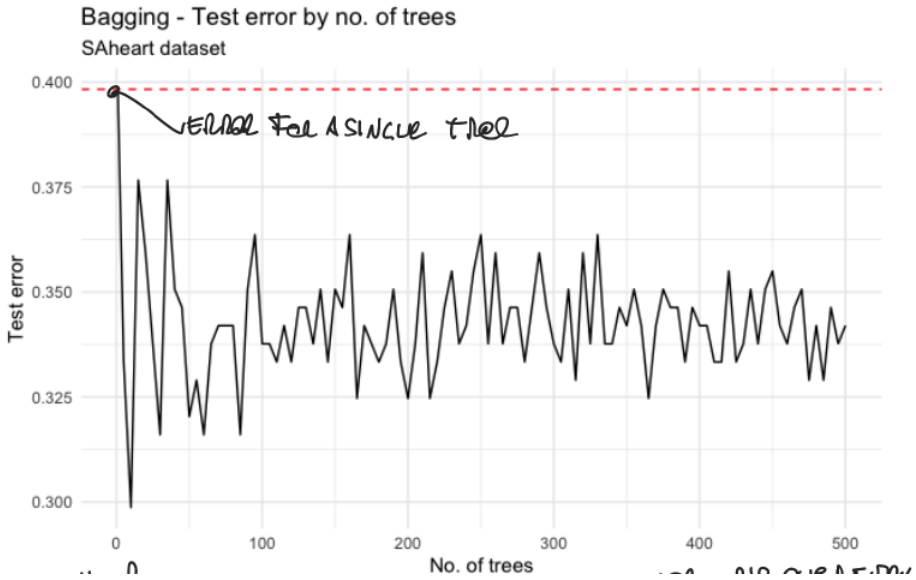


Figura 4.5: Tree.

in the prediction is squeezed and test error stabilized (at a level definitely below the single tree).

Remark 28. Furthermore bagging provides *two free gift*: out of bag error estimation and variable importance scoring.

4.2.2 OOB error estimation

Remark 29 (Idea). Actually there's no need of CV to estimate the test error of a bagged model since a single unit will be considered in some training sample but can be used as test for the trees that didn't use it

Important remark 74 (Out-of-Bag (OOB) Error estimation). The process is:

- the response for the i-th observation is predicted using the trees (will be approximately $B/3$) in which that observation was not included in the bootstrap sample;
- to obtain a single prediction we average these predicted responses (in regression) or can take a majority vote (in case of classification)
- after finding the predictions for all the units overall *OOB MSE* (for a regression problem) or *OOB classification error* (for a classification problem) can be computed.

Important remark 75 (OOB error performance). It can be shown that OOB error:

- is virtually equivalent to leave-one-out cross-validation error, if B sufficiently large;

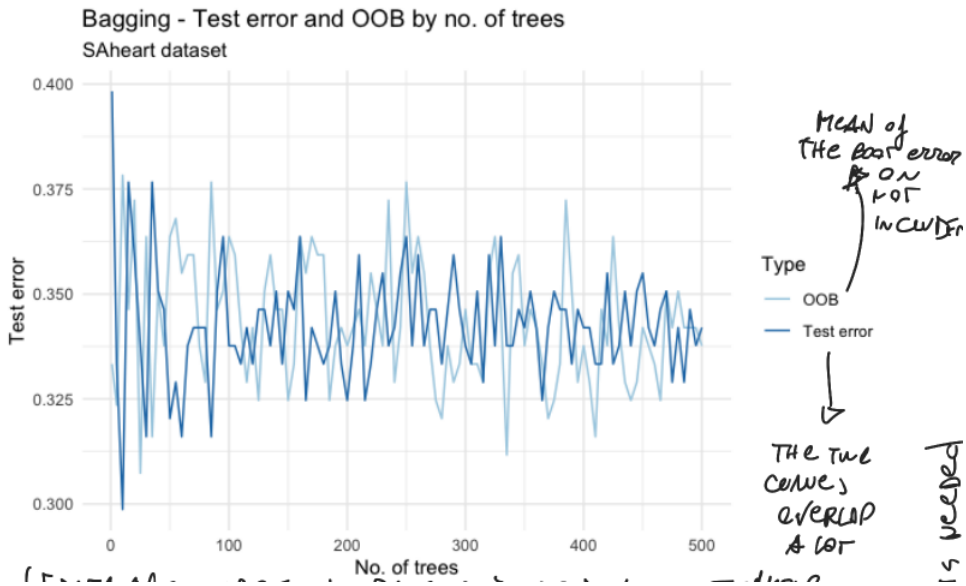


Figura 4.6: Tree.

- is an unbiased estimate of the test error for the bagged model: in fig 4.6 the two curves overlap a lot, that is the behaviour of the OOB error is not far (or significantly lower or upper) the one obtained from having a designed test sample.

Important remark 76 (Data scarcity and comparison with other). If data are scarse do use bagging to avoid a test sample and train the trees on much more data; however if a comparison has to be made with other methods a separate set is needed.

4.2.3 Variable importance measures

Remark 30. Despite being prediction the focus of bagging, another free gift (other than OOB) is having a ranking of variable importance.

Important remark 77 (Variable importance procedure). One can obtain an overall summary of the importance of each predictor (among the trees) in the following way

1. in *regression* we record the total amount that *RSS* is decreased on splits using a given predictor in all the B trees; in *classification* we do the same for *Gini index* (or for other indexes)

2. then we make an average over the B trees; a large value indicates an important predictor.

Example 4.2.1. In figure 4.7 and example using `SAheart` data.

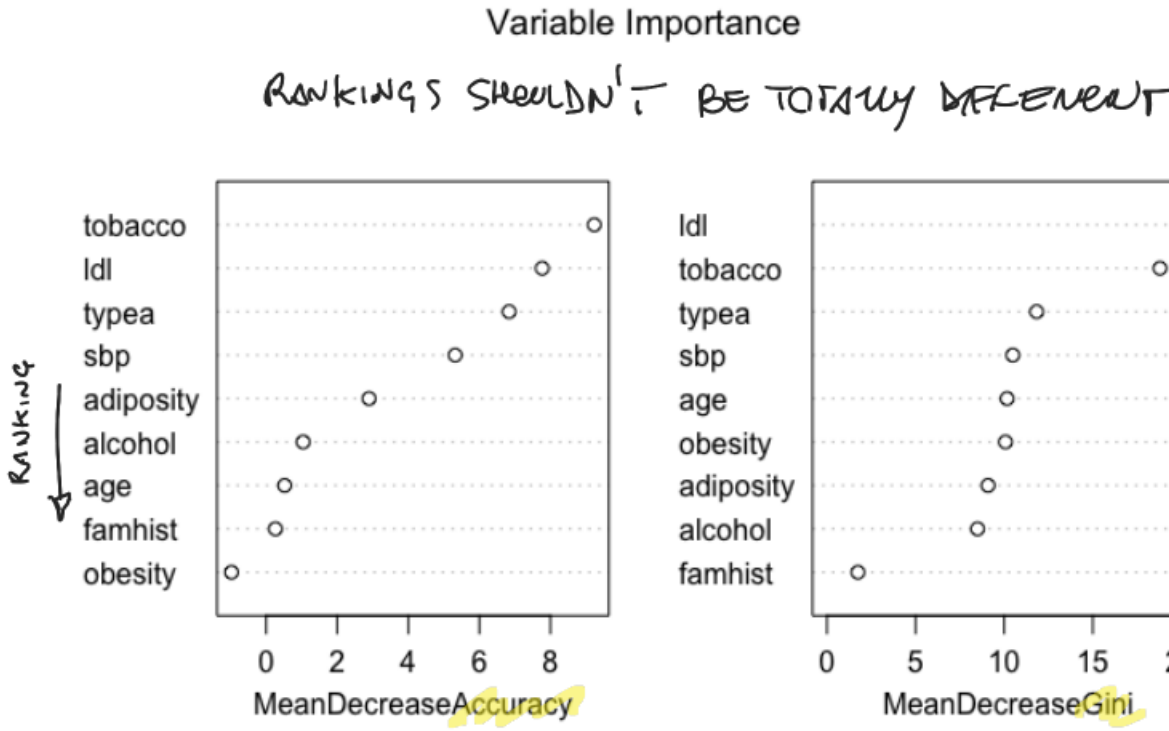


Figura 4.7: Tree.

4.3 Random forest

Important remark 78 (Problem with bagging). Suppose that there is *one very strong predictor* in the data set:

- thus in the collection of bagged trees, most/all will use this strong predictor in the top split;
- consequently, all of the bagged trees will look similar;
- hence the *predictions* from the bagged trees will be highly *correlated*.

Important remark 79 (Random forest). Random forest improve over bagging by decorrelating produced trees: at each split we force to consider only a subset of the predictors, having so less dependence from the original sample and less correlated trees

Definition 4.3.1 (Random forest procedure)**.** We:

- make the bootstrap sample
- in building the tree, each time a split is considered, a random number of $m < p$ predictors is chosen as split candidates (a random forest built using $m = p$ is just bagging) with lower numbers helpful when we have a large number of correlated predictors. Tipically we consider a minority number::
  - $m \approxeq \sqrt{p}$ for classification;
  - $m \approxeq p/3$ for regression.

  The split is allowed to use only one of those m predictors and so there are no variables dominating on all sample (on average $(p-m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance)
- after that the procedure is analogous at the bagging: so we make prediction, take the average/majority votes of trees prediction, and have OOB errors/variable importance as well

Important remark 80. Some final remarks:

- random forest compared to bagging will have a largest reduction in variance of predictions, since uses less correlated tree;
- a comparison of bagging and random forest in figure 4.8: not so clear but normally random forest outperform bagging in terms of predictions on a new test sample
- variable importance ranking can be done as well (figure 4.9) but bagging is better for the prof (while for prediction random forest is better)

4.4 Exercise bagging random forest

The R function is the same as bagging with random forest by default

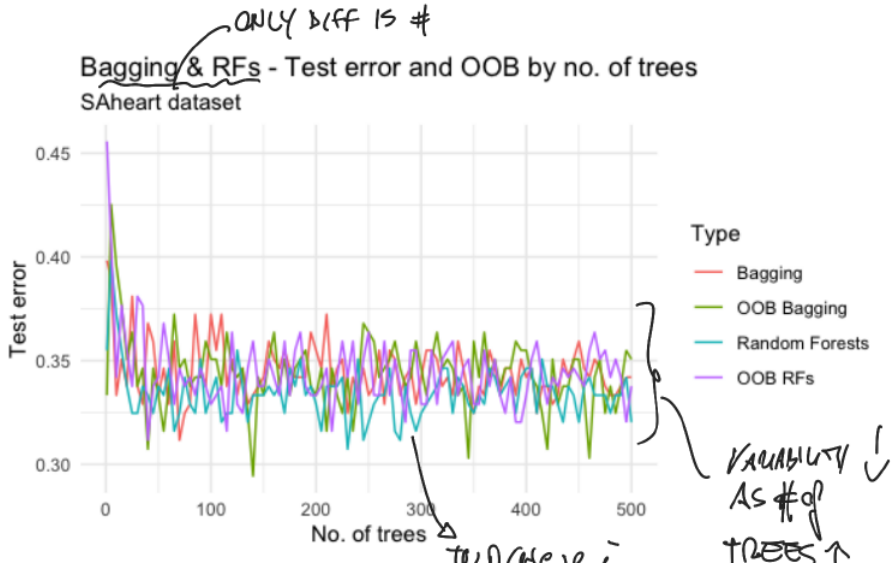


Figura 4.8: Tree.

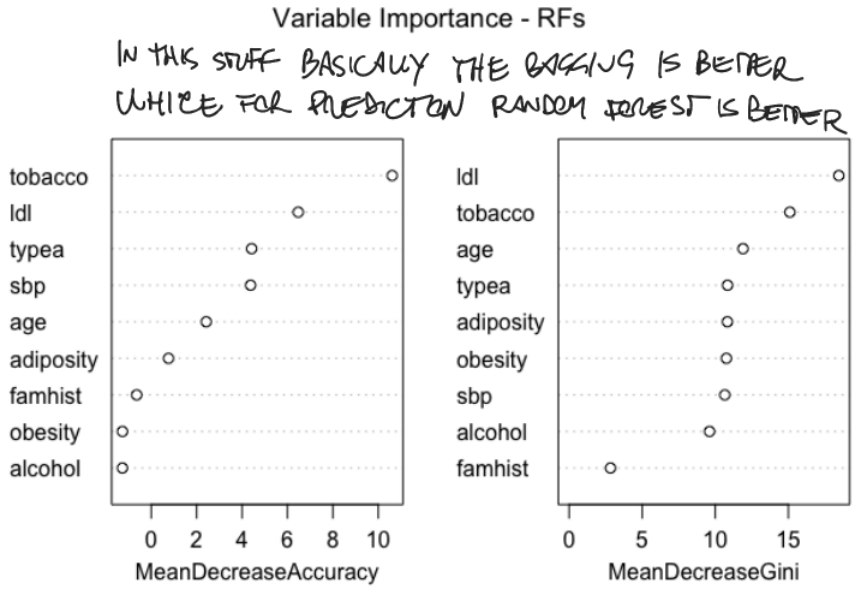


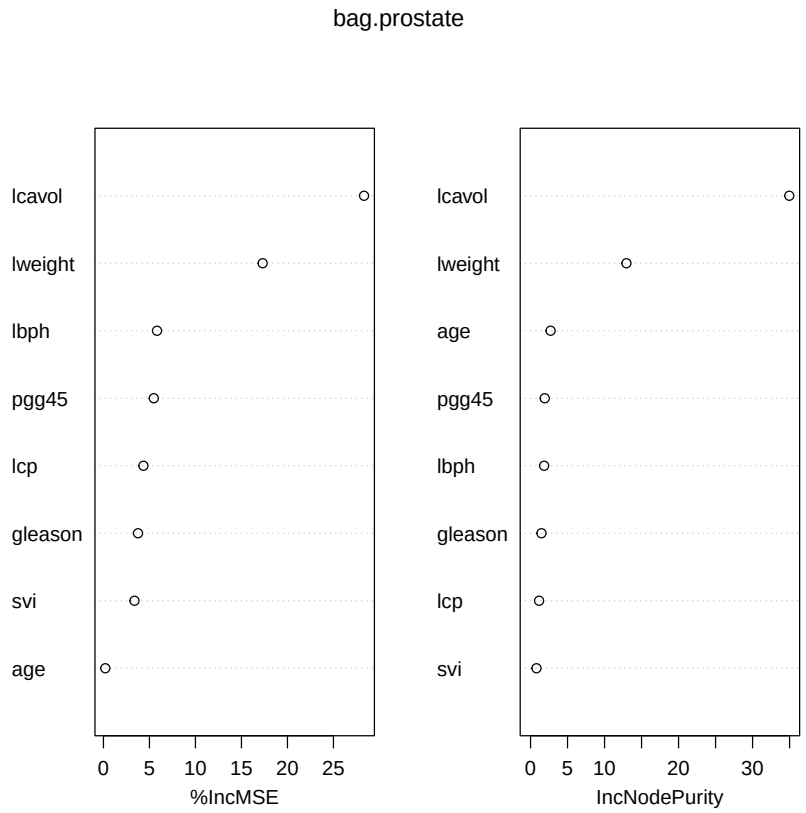
Figura 4.9: Tree.

4.4.1 Regression tree

1. Perform bagging on the dataset and estimate the test error. Which are the most important predictors?

```
## Regression - Bagging
## install.packages('randomForest')
library(lbdatasets)
library(randomForest)

## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.

## ?randomForest

p <- ncol(x) - 1 # no. of predictors
n <- nrow(prostate) # no. of units

## split in train and validation to estimate the error
set.seed(1234)
train <- sample(1:n, ceiling(n/2))
test <- -train
x_train <- prostate[train, ]
x_test <- prostate[test,]

## bagging
set.seed(1234)
bag.prostate <- randomForest(
    lpsa ~ .,
    data = x_train,
    mtry = p, # n of variable included as candidate at each split
    importance = TRUE # obtain vairable importance in output
)

bag.prostate

##
## Call:
##  randomForest(formula = lpsa ~ ., data = x_train, mtry = p, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 0.484815
##                     % Var explained: 60.62

names(bag.prostate) # other stuff look at manual in case

##  [1] "call"            "type"            "predicted"       "mse"             "rsq"
```

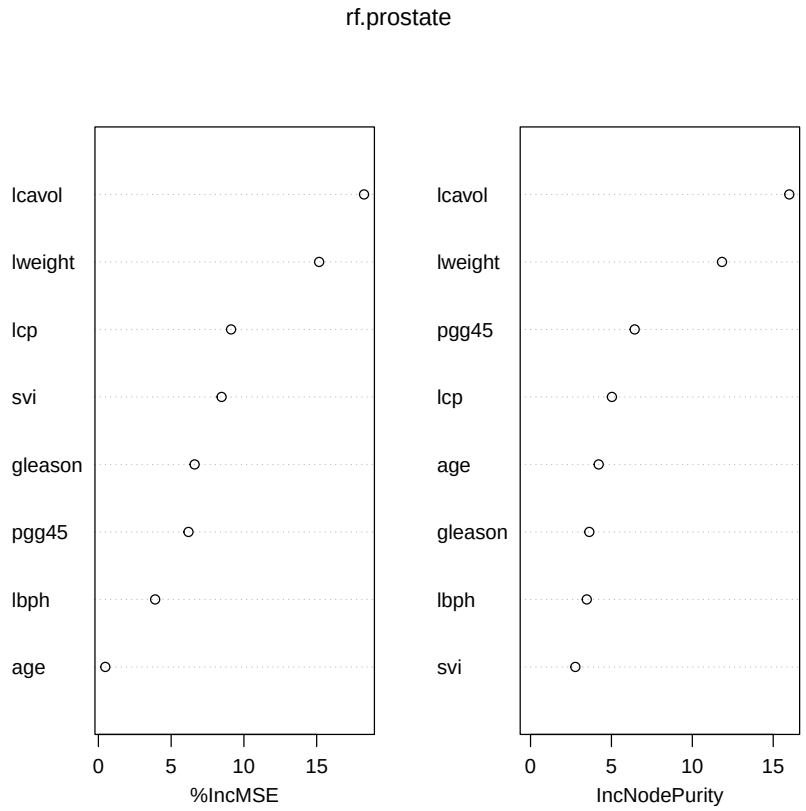
```
##  [6] "oob.times"       "importance"      "importanceSD"    "localImportance" "
## [11] "ntree"           "mtry"            "forest"          "coefs"           "
## [16] "test"            "inbag"           "terms"

## to see Variable importance
importance(bag.prostate) # not sorted

##            %IncMSE IncNodePurity
## lcavol  28.3260234    34.9610128
## lweight 17.3035106    12.9531038
## age      0.2041698     2.7083460
## lbph     5.8253118     1.8325671
## svi      3.3730284     0.8019928
## lcp      4.3471139     1.1530682
## gleason  3.7571030     1.4764065
## pgg45    5.4736326     1.9222442

varImpPlot(bag.prostate) # plot with ordered stuff
```



```
## Accuracy in the validation set
## we obtain the predicted values of yhat (which are calculated
## averaging the predicted values across the collection of trees)
yhat.bag <- predict(bag.prostate, newdata=x_test)
head(yhat.bag)

##         1         7        10        11        12        13
## 0.8368818 1.2985249 1.3713542 1.6794653 0.6157428 2.2060505

mean((yhat.bag - x_test$lpsa)^2) # MSE

## [1] 0.7560983
```

2. Run random forests on the dataset and estimate the test error. Which are the most important predictors?

```
### Regression - Random Forests ####
set.seed(1234)
## to have a random forest we remove the argument mtry = p
rf.prostate<-randomForest(lpsa ~ .,
                          data = x_train,
                          importance = TRUE)
rf.prostate

##
## Call:
##  randomForest(formula = lpsa ~ ., data = x_train, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 0.527365
##                     % Var explained: 57.17

varImpPlot(rf.prostate) # slightly changed (we trust the bagging for
```




```
                        # variable importance)

## Accuracy in the validation set
yhat.rf <- predict(rf.prostate, newdata = x_test)
head(yhat.rf)

##         1         7        10        11        12        13
## 0.8741139 1.5033930 1.2093863 1.6549818 1.1995982 2.1137464

mean((yhat.rf - x_test$lpsa)^2)

## [1] 0.7409796
```

4.4.2 Classification trees

1. Perform bagging on the dataset and estimate the test error. Which are the most important predictors?

```
## ### Classification - Bagging ####
## x <- SAheart[names(SAheart) %without% 'chd']
## y <- SAheart$chd
```

```
## heart <- data.frame(chd=as.factor(y), x)
## remember to set it as factor
heart <- SAheart
heart$chd <- as.factor(heart$chd)

n <- nrow(heart)
p <- ncol(heart) - 1

set.seed(1234)
train <-sample(1:n, ceiling(n/2), replace=FALSE)
test <- -train
heart_train <- heart[train, ]
heart_test  <- heart[test, ]

set.seed(1234)
## bagging with mtry = p
(bag.heart <- randomForest(chd ~ .,
                           data = heart_train,
                           mtry = p,
                           importance = TRUE))

##
## Call:
##  randomForest(formula = chd ~ ., data = heart_train, mtry = p,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 34.63%
## Confusion matrix:
##     0  1 class.error
## 0 125 31   0.1987179
## 1  49 26   0.6533333

## ## REMEMBER: ALWAYS CODE THE RESPONSE AS FACTOR FOR CLASSIFICATION
## ## FORESTS otherwise returns warning but does the predictions
## pippo2 <- randomForest(chd ~ .,
##                          data = SAheart,
##                          subset = train,
##                          mtry = p,
##                          importance = TRUE)

## prediction error
yhat.bag <- predict(bag.heart, newdata = heart_test)
table(yhat.bag, heart_test$chd)

##
## yhat.bag   0   1
##        0 121  55
##        1  25  30
```

```
mean(yhat.bag != heart_test$chd)

## [1] 0.3463203

## compared with OOB error we're more or less the same
## we dont see the varimportance
```

2. Run random forests on the dataset and estimate the test error. Which are the most important predictors?

```
### Classification - Random Forests ####
set.seed(1234)
## rm mtry
rf.heart <- randomForest(chd ~ .,
                         data = heart_train,
                         importance=TRUE)
yhat.rf <- predict(rf.heart, newdata=heart_test)
table(yhat.rf,heart_test$chd)

##
## yhat.rf   0   1
##       0 126  55
##       1  20  30

mean(yhat.rf!=heart_test$chd)

## [1] 0.3246753

## slightly better errors
```

4.5 Boosting

Remark 31. It's one of the most important recent developments in prediction methodology and another approach for improving performance over a decision tree. Can be used both for regression as well; here our focus is classification.

Important remark 81 (Boosting idea). Differently from bagging/RF we aggregate prediction of weak lerners (say tree) which are *not* grown independently (no bootstrap here):

- M tree are fit *sequentially* with M choosen via crossvalidation;
- each tree:
  - is typically very simple, boosting uses *stumps*; these are single split tree with only two terminal nodes;
  - is fit on a *weighted version of the training dataset that accounts for mistakes* that the learner made in previous steps;

- after fitting we *reweight the training data* to increase the weight of units wrongly classified;
- finally a *weighted majority vote* of the classifiers previously produced is obtained

Important remark 82 (Performance). For many learners/algorithms, this strategy dramatically improves performance (both bias/variance reduction).

Remark 32. Most commonly used version of the AdaBoost procedure (Freund and Schapire, 1996), also called Discrete AdaBoost.

Definition 4.5.1 (Discrete Adaboost algorithm)**.** In the two-class classification setting the steps are:

1. we have training data $(x_1, y_1), \ldots, (x_N, y_N)$ with x_i a vector valued feature and $y_i = -1$ or $= 1$ (artificially used as labels of the two groups) and start with equal weights for all the units $w_i = 1/N$, $i = 1, \ldots, N$;
2. we start the weighting adjustment process M times (giving higher weight to cases that are currently misclassified), repeating for $m = 1, \ldots, M$ (m is the number of iteration/subsequent trees):
   - fit the classifier (eg a tree) $f_m(x)$ producing values ± 1 using weights w_i on the training data
   - compute the average prediction error for the current iteration err_m and subsequently the log odd of correct classification c_m

$$err_m = \mathbb{E}_w \left[I(y \neq f_m(x)) \right] = \frac{\sum_{i=1}^{N} w_i I(y_i \neq f_m(x_i)}{\sum_{i=1}^{N} w_i}$$

$$c_m = \log \left(\frac{1 - err_m}{err_m} \right)$$

   where
   - $I(y_i \neq f_m(x_i))$ means unit i was wrongly classified
   - c_m, the log odd of correct classification, can be seen as *trustiness* of a single classifier: it's positive if the classifier predict correctly 50% of the weighted cases (or more) and is correlated with its performance
   - refresh the weights by setting

$$w_i \leftarrow w_i \exp \left[c_m I(y_i \neq f_m(x_i)) \right], \quad i = 1, \ldots, N$$

   the weight for the unit i increases if $c_m I(y_i \neq f_m(x_i)) > 0$, that is *if* current prediction for the unit was *wrong* and as much as the current *classifier is trustworthly.*
   Thus each successive classifier is forced to concentrate on those training observations that are missed by previous ones in the sequence.
   - renormalize weights so that $\sum_i w_i = 1$
3. the final classifier is a linear combination of the classifiers from each stage weighted by classifiers trustiness

$$F(x) = \sum_{m=1}^{M} c_m f_m(x)$$

4.5.1 Exercise

1. Fit boosted classification trees to the SAheart data set. Choose the best number of trees among 25, 50, 100, 150.
2. Estimate the test error and compare it with that of bagging and random forests.

```
## Boosting
library(gbm)

## Loaded gbm 2.1.8.1

n <- nrow(SAheart)
set.seed(1234)
train <- sample(1:n, ceiling(n/2), replace = FALSE)
test <- -train
## here we dont need to coerce to factor
heart_test <- SAheart[test,]
heart_train <- SAheart[train,]

## ?gbm::gbm
boost.out <- gbm(chd ~ .,
                 data = heart_train,
                 distribution = "bernoulli",
                 n.trees = 100,
                 # interaction.depth = 1, # complexity of model (1 is default)
                 bag.fraction = 1) # this is changed: we use all the training dataset
boost.out

## gbm(formula = chd ~ ., distribution = "bernoulli", data = heart_train,
##     n.trees = 100, bag.fraction = 1)
## A gradient boosted model with bernoulli loss function.
## 100 iterations were performed.
## There were 9 predictors of which 9 had non-zero influence.

## crucial parameter here is n.trees: we want to choose it using CV
n_train <- nrow(heart_train) # size of the training
set.seed(1234)
folds <- sample(1:5, n_train, replace=TRUE)
# table(folds)
B <- seq(from = 25, to = 200, by = 25) ## number of trees we consider
## matrix with cv error for each folds and number of trees
cv_err<-matrix(NA, 5, length(B),
               dimnames = list(NULL, paste0("B=",B[1:length(B)])))
for (i in 1:5){
    fold_test  <- heart_train[folds==i,]
    fold_train <- heart_train[folds!=i,]
    for (j in 1:length(B)){
        ## estimation
        boost.out <- gbm(chd ~ .,
```

```
                          fold_train,
                          distribution = "bernoulli",
                          bag.fraction = 1,
                          ## interaction.depth = 1,
                          n.trees = B[j])
        ## prediction
        yhat <- as.integer(predict(boost.out,
                        newdata = fold_test,
                        n.trees = B[j],
                        type = "response") > 0.5)
        ## error
        cv_err[i,j] <- mean(yhat != fold_test$chd)
    }
}
print(cv_err, digits = 3)

##       B=25  B=50  B=75 B=100 B=125 B=150 B=175 B=200
## [1,] 0.227 0.273 0.273 0.273 0.273 0.295 0.295 0.295
## [2,] 0.262 0.310 0.333 0.333 0.310 0.333 0.333 0.310
## [3,] 0.352 0.315 0.296 0.296 0.296 0.259 0.278 0.296
## [4,] 0.348 0.391 0.413 0.413 0.391 0.370 0.370 0.370
## [5,] 0.289 0.311 0.311 0.267 0.267 0.267 0.267 0.289

colMeans(cv_err)

##      B=25      B=50      B=75     B=100     B=125     B=150     B=175     B=200
## 0.2955489 0.3198963 0.3253023 0.3164134 0.3073037 0.3048558 0.3085595 0.3119458

(b_best <- B[which.min(colMeans(cv_err))])

## [1] 25

# Fit the best boosted trees on the whole training set
boost.train <- gbm(chd~.,
                   data = SAheart[train,],
                   distribution = "bernoulli",
                   n.trees = b_best,
                   ## interaction.depth = 1,
                   bag.fraction = 1)
yhat.te <- as.integer(predict(
    boost.train,
    newdata = SAheart[-train,],
    n.trees = b_best,
    type="response") > 0.5)
table(yhat=yhat.te,SAheart$chd[-train])

##
## yhat   0   1
##    0 133  71
##    1  13  14
```

```
mean(yhat.te!=SAheart$chd[-train])

## [1] 0.3636364

# 36% not very good compared to random forest and bagging seen
# yesterday
```

Capitolo 5

Support vector machine

SVM are:

- usable for regression and classification: we use them for the latter;
- a generalization of a simple/elegant classifier, the maximal margin classifier, which unfortunately cannot be applied to most data sets (requiring the classes to be separable by a linear boundary).

5.1 Maximal margin classifier

5.1.1 Hyperplanes

Definition 5.1.1 (Hyperplane)**.** In a p-dimensional space, a hyperplane is a flat affine subspace of dimension $p-1$.

Example 5.1.1. For a space where:

- $p = 2$, a hyperplane is a flat one-dimensional subspace, i.e. a line. In two dimensions ($p = 2$) a hyperplane is defined by the equation

  $$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

  for parameters β_0, β_1 and β_2.
  Any $X = (X_1, X_2)^T$ for which the equation holds is a point on the hyperplane.

- in the general p-dimensional case the hyperplane is defined by the equation:

  $$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0 \tag{5.1}$$

  defines a p-dimensional hyperplane, again in the sense that if a point $X = (X_1, X_2, \ldots, X_p)^T$ in p-dimensional space (i.e. a vector of length p) satisfies the equation, then X lies on the hyperplane.
  If $p = 3$, a hyperplane is a flat two-dimensional subspace, i.e. a plane, while if $p > 3$ it can be hard to visualize a hyperplane, but the notion of a $(p-1)$-dimensional flat subspace still applies

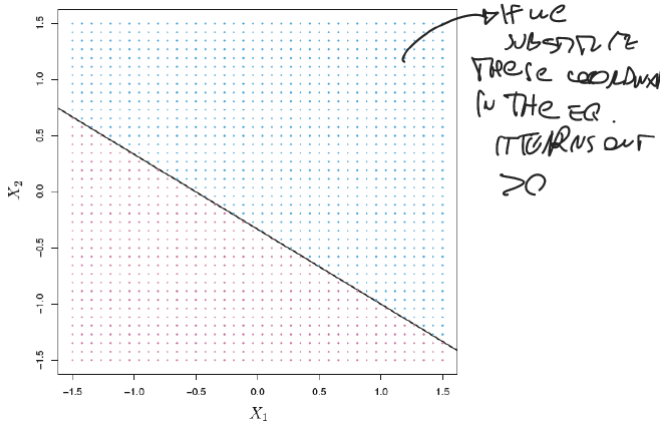


Figura 5.1: SVM.

5.1.2 Classification using a separating hyperplane

Important remark 83 (Classification using hyperplanes). In general, suppose that X does *not* satisfy 5.1; then it can be

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0$$
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0$$

In other words X lies on one or the other side of the hyperplane.
So we can think of the hyperplane as dividing p-dimensional space into two halves; one can easily determine on which side of the hyperplane a point lies by simply calculating the *sign* of the left hand side of 5.1.

Example 5.1.2. In figure 5.1 the hyperplane $1+2X_1+3X_2 = 0$ is shown. The blue region is the set of points for which $1 + 2X_1 + 3X_2 > 0$, and the purple region is the set of points for which $1 + 2X_1 + 3X_2 < 0$.

Important remark 84 (Setting). Now we suppose that:

- we have a $n \times p$ data matrix X that consists of n training observations in p-dimensional space

$$x_1 = \begin{bmatrix} x_{11} \\ \ldots \\ x_{1p} \end{bmatrix}, \ldots, x_n = \begin{bmatrix} x_{n1} \\ \ldots \\ x_{np} \end{bmatrix}$$

  and that these observations comes from two groups, that is $y_1, \ldots, y_n \in \{-1, 1\}$, where -1 represents one class and 1 the other class;

- we also have a test observation, a p-vector of observed features $x^* = (x_1^*, \ldots, x_p^*)^T$.

- the goal is to develop a classifier based on the training data that will correctly classify the test observation based upon the concept of a separating hyperplane.

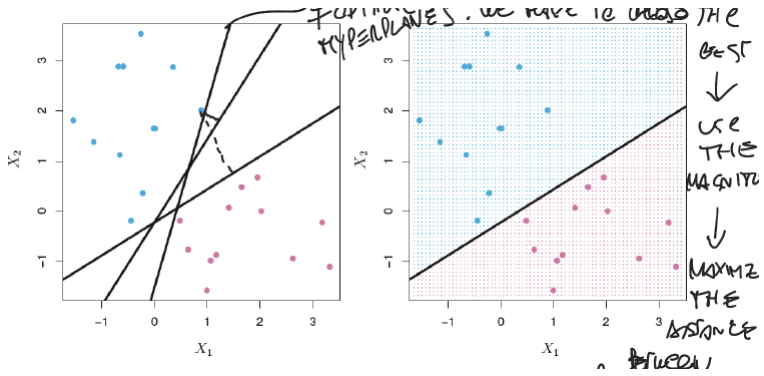


Figura 5.2: SVM.

Important remark 85 (Hyperplane $\exists$ assumption). Now suppose that an hyperplane that separates perfectly the training observations according to their class labels y_i exists:

- that optimal separating hyperplane has the property that

$$\begin{cases} \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0, & \text{if } y_i = 1 \\ \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0, & \text{if } y_i = -1 \end{cases}$$

  or succintly, given that $y_i \in \{-1, 1\}$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0, \quad \forall i = 1, \ldots n$$

- if such a separating hyperplane exists, we can use it to construct a very natural classifier: we classify the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \ldots + \beta_p x_p^*$:
  - if $f(x^*) > 0$ we assign x^* observation to class 1;
  - if $f(x^*) < 0$ we assign it to class -1;
- we could also make use of the magnitude of $f(x^*)$
  - if $f(x^*)$ is far from zero, then x^* lies far from the hyperplane so we can be confident about its class assignment
  - if $f(x^*)$ is close to zero, we are less certain about the assignment

In principle it remains to find these β; a first idea could be to find the betas for which the distance between points and line is maximum

5.1.3 Maximal margin classifier

Remark 33 (Idea). In general if our data can be perfectly separated using a hyperplane, then there will exists an *infinite* number of such hyperplanes; (figure 5.2 given different possible hyperplanes, to choose the best using the magnitude, one natural way is maximizing the distance between points and the candidate hyperplanes)

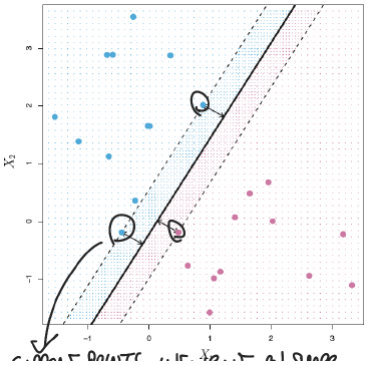


Figura 5.3: SVM.

Definition 5.1.2 (Maximal margin hyperplane)**.** It is the hyperplane that is farthest from the training observations.

Important remark 86 (Construction). To determine it we:

- compute the *distances* (perpendicular) from each training observation to a given separating hyperplane;
- the *smallest of such distance*, the minimal distance from the observations to the hyperplane, is called the **margin**;
- the *maximal margin hyperplane* is the separating hyperplane for which the margin is largest;
- in the maximal margin hyperplane (eg figure 5.3) there are several training observations that are equidistant from the margin (and lie along the dashed lines indicating the width of the margin).
  These observations are called **support vectors**: they are vectors in p-dimensional space which "support" the maximal margin hyperplane, in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well (while moving any other observations would not).

Definition 5.1.3 (Maximal margin classifier)**.** Classifier which uses the maximal margin hyperplane.

Important remark 87 (Maximal margin classifier construction). The steps:

- having n observation $x_1, \ldots, x_n \in \mathbb{R}^p$ and associated class label $y_1, \ldots, y_n \in \{-1, 1\}$ the maximal margin hyperplane is the solution to the maximization of the margin

$$\max_{\beta_0,\ldots,\beta_p} M, \text{subject to,} \tag{5.2}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M, \quad \forall i = 1, \ldots, n \tag{5.3}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{5.4}$$

  where :

- M is the distance of the closest point to the margin,
- we want $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$ to be maximum: this constraint guarantees that each observation will be on the correct side of the hyperplane, provided that $M > 0$
- the 5.4 is just a constraing/normalization/rescaling; is not really a constraint on the hyperplane, since if

$$\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} = 0$$

defines a hyperplane then so does

$$k(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) = 0$$

for any $k \neq 0$.
However 5.4 adds meaning to 5.3; one can show that with this constraint the perpendicular distance from the ith observation to the hyperplane is given by

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots \beta_p x_{ip})$$

Therefore the constraints 5.4 and 5.3 ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane. Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \ldots, \beta_p$ to maximize M.
And this motherfucka is exactly the definition of the maximal margin hyperplane.

Important remark 88 (Final remarks). We have that:

- **existence**: *if* a separating hyperplane exists, maximal margin classifier is a very natural way to perform classification; however, *in many cases* observations belonging to two classes are not always perfectly separable by a hyperplane, and so *there is no maximal margin classifier*. In this case, the optimization problem has no solution with $M > 0$;
- **overfitting**: maximal margin classifier can lead to *overfitting* givin too much importance to the nearest few observations, especially if p is large; a different sample could produce sensibly different classifier.
  In figure 5.4 we only added the circled image on the right: the addition of a single observation leads to a dramatic change in the maximal margin hyperplane.

5.2 Support vector classifier

Definition 5.2.1 (Soft margin hyperplane)**.** We can develop a **soft margin** hyperplane, that separates *almost perfectly* the classes and allows some level of error: we allow to misclassify a few observations (a parameter which can be finetuned) which will be *within the margin* or even in the *incorrect side of the hyperplane*

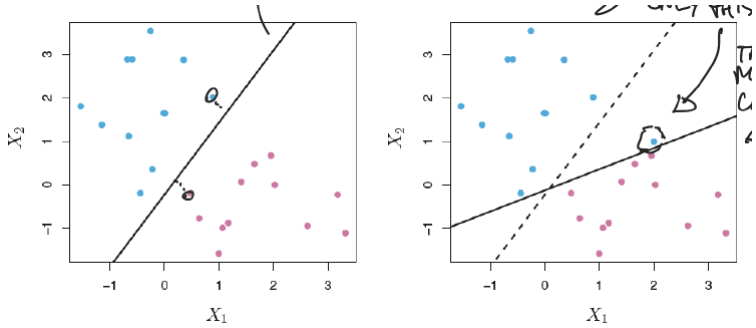


Figura 5.4: SVM.

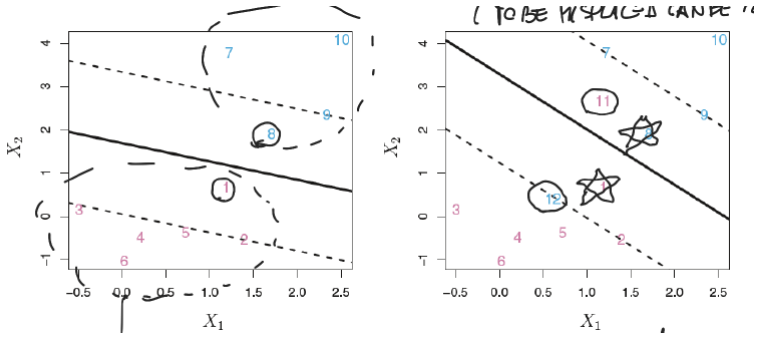


Figura 5.5: SVM.

Example 5.2.1. In figure 5.5:

- on the left two observation (1 and 8) are classified correctly (in the correct side of the plane) but within the margin;
- on the right we add two observation (11 and 12) that will be wrongly classified.

Important remark 89 (Derivation). The soft margin hyperplane is the solution to the following maximization problem:

$$\max_{\beta_0,\ldots,\beta_p,\varepsilon_1,\ldots,\varepsilon_n} M, \quad \text{subject to:} \tag{5.5}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i), \quad \forall i = 1, \ldots, n \tag{5.6}$$

$$\varepsilon_i \geq 0, \ \sum_{i=1}^{n} \varepsilon_i \leq C \tag{5.7}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{5.8}$$

where:

- M is still the width of the margin, the quantity to be maximized;
- the terms $\varepsilon_1, \ldots, \varepsilon_n$ in constraint 5.6 are *slack variables* that allow individual observations to be within the margin or even on the wrong side of the hyperplane. ε_i tells us where the ith observation is located:
  - $\varepsilon_i = 0$: i-th observation is on the correct side of the margin;
  - $\varepsilon_i > 0$: i-th observation is within the the margin (but right side of the hyperplane);
  - if $\varepsilon_i > 1$: then it is on the wrong side of the hyperplane.
- C is a nonnegative constant to be choosen that bounds the sum of the ε_i, so it determines the number/severity of the violations (to the margin and hyperplane) that we will tolerate:
  - if $C = 0$ then there is no tolerance for violations (even to the margin) and it must be that $\varepsilon_1 = \ldots = \varepsilon_n = 0$, in which case the problem amounts to the maximal margin hyperplane optimization problem
  - if $C > 0$ no more than C observations can be wrongly classified (be on the wrong side of the hyperplane) having each of them a cost of at least 1

Definition 5.2.2 (Support vectors here)**.** Here are *the observations within the margin* (in one or another side of the plane) the only influencing the hyperplane: observation correctly classified and that lies out of the margins does not play a role.

Important remark 90 (C and bias variance tradeoff). C, tipically chosen with CV, controls the bias-variance trade-off:

- when C is small (small budget, less tolerant to violations), we seek narrow margins that are rarely violated; this will produce a classifier that is highly fit to the data, have low bias but high variance;
- when C becomes larger, we become more tolerant of violations, the margin get wider; this will fit data less hard and result in a classifier potentially more biased but more stable/less variance.

In R C is cost of an error (inverse interpretation) so if the cost is small we can make more and viceversa if expensive we want to make less

Definition 5.2.3 (Support vector classifier)**.** It is the classifier that uses the developed soft margin; once we have solved the optimization problem we classify a test observation x^* by simply determining on which side of the hyperplane that is calculating the sign if

$$f(x^*) = \beta_0 + \beta_1 x_1^* + \ldots + \beta_p x_p^* = \begin{cases} > 0 & \text{then classified as } +1 \\ < 0 & \text{then classified as } -1 \end{cases}$$

Important remark 91 (Support vector classifier vs other). We have that:

- compared to maximal margin classifier it *increase robustness* to single/individual observations, leading to less overfitting and *better classification* in the testing sample;
- being influenced by support vectors, support vector classifier is *robust* to the behavior of *observations that are far away from the hyperplane* (since the classifier's decision rule is based only on a potentially small, depending on the magnitude of C, subset of the training observations) differently from other methodsq (eg LDA)
- only problem with support vector classifier is that they allows only for *linear boundaries*; for this we look at support vector machines (which are a generalization).

5.3 Support vector machines

Example 5.3.1. In figure 5.6 a support vector classifier (on the right) or any linear classifier will perform poorly here (should be below the line predict red, should be).

Remark 34 (Enlarge feature space?). We could address the problem of non-linear boundaries between classes by enlarging the feature space (eg using quadratic, cubic, and even higher-order polynomial functions of the predictors) but we could end up with a *huge number of features*:

- *computations* would become unmanageable.
- is number of parameter increase and we could increase *variance*

And this is where SVM kick in.

Important remark 92 (Support vector machine). This learner:

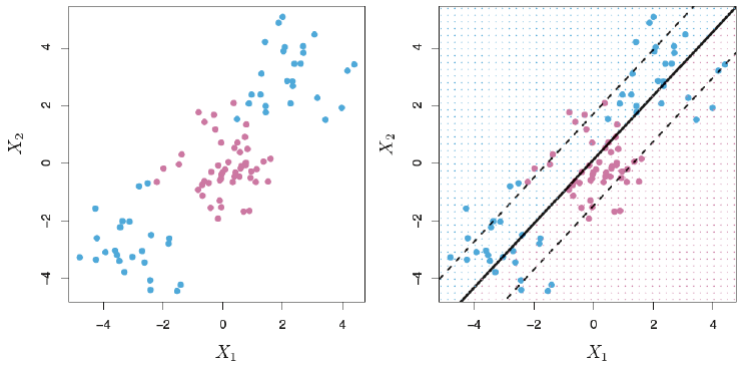


Figura 5.6: SVM.

- *optimize the search of the separating shit by looking at the dataset by rows* (instead of columns/features by a constant).
- use *kernels*, special function that allows to enlarge the feature space in order to accommodate a non-linear boundaries between classes.
  One example is the use *inner product* between observations

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

  In general, *kernel functions* can be seen *as a generalization* of the inner product of two vectors;

Example 5.3.2. For example the support vector classifier boundary we've seen before ($f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$) can be rewritten using the inner product

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

where there are n parameters α_i $i = 1, \ldots, n$, one per each training observation (which is somehow multiplied to the observation x we want to classify). So in order to evaluate the function $f(x)$, we need to compute the inner product between the new point x and each of the training points x_i .

Important remark 93 (Parameters α). We have that

- the number of parameter is typically increased going from p to n, thus allowing more flexibility;
- it can be shown that to estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0 all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_i' \rangle$ between all pairs of training observations. One could think $\binom{n}{2}$ can be cumbersome but if we want to enlarge our feature space this can be convenient;
- $\alpha_i \neq 0$ only for the support vectors: so if $\mathcal{S}$ is the collection of indices of these support points, we can rewrite any solution function as

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

which involves fewer terms and simplify procedure a lot

Definition 5.3.1 (Kernel)**.** It's a function that quantifies the similarity of two observations.

Important remark 94 (Which ones). The kernel choice is impactful and done via crossvalidation. We can choose among:

- the **linear kernel** defined as

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

(measure the distance between pairs if centered) results in the support vector classifier; basically, this kernel quantifies the similarity of a pair of observations using Pearson (standard) correlation.

- the **polynomial kernel** of degree $d > 0$ is the first *non-linear* one, defined as

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

Using such a kernel with $d > 1$ leads to a much more flexible decision boundary: it essentially amounts to fitting a support vector classifier in a higher-dimensional space involving polynomials of degree d, rather than in the original feature space.

- the **radial kernel** is another popular *non-linear* one:

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)^d$$

with $\gamma > 0$; this kernel

  - if an observation to be classified is distant from a training observation x_i the value of K above will be small
  - therefore remembering that the predicted class label for the test observation x^* is based on the sign of $f(x^*)$:

$$f(x^*) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x^*, x_i)$$

if a unit x_i is very different from the unit to be classified x^* then $K(x^*, x_i)$ will be very low and in general the training observations that are far from x^* will play essentially no role in the predicted class label for x.
Therefore, the radial kernel has *very local behavior*, in the sense that only nearby training observations have an effect on the class label of a test observation.

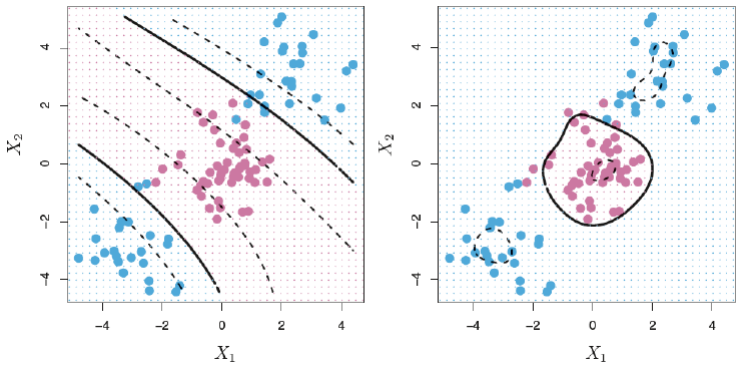


Figura 5.7: SVM.

Definition 5.3.2. When *the support vector classifier is combined with a non-linear kernel* (eg polynomial, radial), the resulting classifier is known as a *support vector machine.*

Example 5.3.3. In figure 5.7 both a polynomial kernel of degree 3 (left) and a radial kernel (right); going with a nonlinear kernel makes both capture the right decision boundary.

5.4 Exercise

1. Fit a Support Vector Classifier (i.e. SVM with linear kernel) to classify the observations on the training set. Choose in CV the optimal cost parameter between 0.001,0.01,0.1,1,5,10,100. Estimate the test error.

```
## Support Vector Machines ####
library(lbdatasets)
library(e1071)

## response variable here need to be recoded as factor otherwise it
## does regression
## x <- SAheart[,-ncol(SAheart)]
## y <- SAheart[,ncol(SAheart)]
heart <- SAheart
heart$chd <- as.factor(heart$chd)

## ?svm: linear kernel is support vector classifier. need to choose
## let's read an output for different costs before doin what requested
out.svm <- svm(chd ~ ., data = heart, kernel = "linear", cost = 10)
summary(out.svm)

##
## Call:
## svm(formula = chd ~ ., data = heart, kernel = "linear", cost = 10)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  278
##
##  ( 138 140 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
## it says number of vectors (points between the margins) is 278, 138
## comes from class 1 credo while 140 dall'altra.
##
## See what happens if we reduce the cost in R if the cost is small
## making a mistake is not a problem/cheaper so our margin will be
## large while for high cost the margin will be littler; a little
## margin is more risky for overfitting
out.svm <- svm(chd ~ ., data = heart, kernel="linear", cost=0.1)
summary(out.svm)
```

```
##
## Call:
## svm(formula = chd ~ ., data = heart, kernel = "linear", cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  284
##
##  ( 141 143 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
## being the margin higher, the number of support vector (points lying
## between the two margins) is increased
```

```r
## Support vector classifier choose cost parameter by cv, we don't do
## it by hand but we use tune which does wrap things and do it for us
n <- nrow(heart)
set.seed(1234)
train <- sample(1:n, ceiling(n/2), replace = FALSE)
test <- -train
heart_train <- heart[train, ]
heart_test <- heart[test, ]

set.seed(1234)
tune_out <- tune(svm, # function to tune
                 chd ~ ., # below the training function parameters
                 data = heart_train,
                 kernel = "linear",
                 ranges = list(cost=c(0.001,0.01,0.1,1,5,10,100)), # ranges of parameters
                 tunecontrol = tune.control(cross=10)) # to change the number of folds
# to change the no. of folds K -> cross=K
summary(tune_out)


##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.2777174
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.3251812 0.08884511
## 2 1e-02 0.3295290 0.09111136
## 3 1e-01 0.2777174 0.09309726
## 4 1e+00 0.2992754 0.08682559
## 5 5e+00 0.3036232 0.08577090
## 6 1e+01 0.3077899 0.07913370
## 7 1e+02 0.3036232 0.08577090


## the best parameter for cost is 0.1

## store the best model and use it for prediction
best_model <- tune_out$best.model
yhat.linear <- predict(best_model, newdata = heart_test)

## MSE
table(yhat.linear, heart_test$chd)
```

```
##
## yhat.linear   0   1
##           0 138  53
##           1   8  32

mean(yhat.linear != heart_test$chd)

## [1] 0.2640693

# 26% for support vector classifier
```

2. Fit an SVM using a (non-linear) radial kernel; tune the gamma (1,2,3,4,5) and the cost (0.1,1,10,100,1000) parameters in cross-validation. Estimate the test error.

```
### Support vector machine - Radial kernel
## we copypaste with minor changes
set.seed(1234)
tune_out<-tune(svm,
               chd ~ .,
               data=heart_train,
               kernel="radial", # this changed
               ranges = list(cost=c(0.001,0.01,0.1,1,5,10,100,1000), # this
                             gamma=c(0.1,0.2,0.3,0.4,0.5)), # changed
               tunecontrol = tune.control(cross=10))
## here the optimal parameter is with cost 1 and gamma 0.1
tune_out

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.1
##
## - best performance: 0.2905797

## summary(tune_out)

## store the best model for predictions and get the error
best_model_radial <- tune_out$best.model
yhat.radial<-predict(best_model_radial, newdata = heart_test)
table(yhat.radial, heart_test$chd)

##
## yhat.radial   0   1
```

```
##           0 134  62
##           1  12  23

mean(yhat.radial != heart_test$chd)

## [1] 0.3203463

## increased error: radial kernel works with points near, local
## behaviour

## Before going to the next point, see the effect of higher gamma (not
## optimal 0.1) which weights the distance between points, so taking a
## gamma higher make distances to weight a lot
out.radial <- svm(chd~.,
                  data = heart_train,
                  kernel = "radial",
                  gamma = 1,
                  cost = 0.1)
yyhat <- predict(out.radial, newdata = heart_test)
table(yyhat, heart_test$chd)

##
## yyhat   0   1
##     0 146  85
##     1   0   0

## and everything is put in the class 0 so for radial il crucial
## gamma parameter polinomial is less sensitive to gamma
```

3. Fit an SVM using a (non-linear) polynomial kernel; tune the degree (1,2,3,4,5) and the cost (0.1,1,10,100,1000) parameters in cross-validation. Estimate the test error.

```
### Support Vector Machines - Polynomial kernel
set.seed(1234)
tune_out<-tune(svm,
               chd ~ .,
               data = heart_train,
               kernel = "polynomial",
               ranges = list(cost=c(0.001,0.01,0.1,1,5,10,100,1000),
                             gamma=c(0.1,0.2), # free gift
                             degree=1:5),
               tunecontrol = tune.control(cross=10))
tune_out

##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma degree
##     1   0.1      1
##
## - best performance: 0.2777174

## summary(tune_out)

# best model and prediction/error
best_model_polynomial <- tune_out$best.model
yhat.polynomial <- predict(best_model_polynomial,
                           newdata = heart_test)

table(yhat.polynomial, heart_test$chd)

##
## yhat.polynomial   0   1
##               0 138  53
##               1   8  32

mean(yhat.polynomial != heart_test$chd)

## [1] 0.2640693

## 26% here again similarly to linear kernel
## both linear (1 degree solution is identical)
```

4. Finally one could set in `ranges` the kernel as well (but it takes time)

Capitolo 6

Further exercises

6.1 Mock exam

Load the workspace `MockExam.RData` into your R environment. Using command ls() you will see an object named `genes`. Such matrix includes 40 tissue samples with gene expression measurements on 1,000 genes. The first 20 samples are from healthy patients (class=1), while the second 20 are from a diseased group (class=2). Labels are included in object `genes_labs`.

```
load('data/MockExam.RData')
```

1. Apply the Lasso on the subset of units identified by `train` to perform variable selection of the model that has `genes_labs` as response. How many predictors are retained? Compute the test error estimate.

NB: she usually provides train and validation set

```
## Train
x <- as.matrix(genes[train,])
y <- as.factor(genes_labs[train])

## Test
x_test <- as.matrix(genes[-train,])
y_test <- as.factor(genes_labs[-train])

library(glmnet)
out.lasso <- glmnet(x, y, alpha=1, family="binomial")

## optimal lambda
set.seed(1234)
cv.lasso <- cv.glmnet(x, y, alpha=1, family="binomial")
best.lambda <- cv.lasso$lambda.min

## number of coefficients retained
lasso.coef <- predict(out.lasso, s=best.lambda, type="coefficients")
sum(lasso.coef[-1,1]!=0)

## [1] 23
```

```
## Test error estimate
lasso.pred<-predict(out.lasso,
                    s = best.lambda,
                    type = "class",
                    newx = x_test)
table(yhat=lasso.pred,y_test)

##     y_test
## yhat 1 2
##    1 5 0
##    2 0 5
```

The lasso with the tuned lambda retains 23 coefficients and returns a perfect accuracy, with 0 errors.

2. Run a random forest on the training set. Evaluate the importance of the genes in terms of average decrease of Gini index. Why is that measure connected with the variable importance?

```
## Random Forests
library(randomForest)
genes.df<-data.frame(y=as.factor(genes_labs), genes)
set.seed(1234)
out.rf <- randomForest(y ~ .,
                       data = genes.df,
                       subset = train,
                       importance = T)
imp.var <- importance(out.rf)
head(imp.var)

##            1         2 MeanDecreaseAccuracy MeanDecreaseGini
## X1  0.000000  0.000000             0.000000      0.000000000
## X2 -1.001002 -1.001002            -1.001002      0.003750000
## X3 -1.001002  1.001002            -1.001002      0.009244444
## X4  0.000000  0.000000             0.000000      0.000000000
## X5  0.000000  0.000000             0.000000      0.000000000
## X6  0.000000  0.000000             0.000000      0.000000000

## imp.var sorted according to decreasing values of Mean Decrease
## of Gini's index
sorted.imp.var <- imp.var[order(imp.var[,4], decreasing = T), ]
head(sorted.imp.var)

##             1        2 MeanDecreaseAccuracy MeanDecreaseGini
## X584 3.202185 3.380287             3.536849        0.4364903
## X600 3.675928 3.844871             3.931197        0.4130895
## X564 3.261145 3.496614             3.434435        0.3444252
## X539 3.257682 3.005987             3.254890        0.3207902
## X555 2.537508 2.929658             2.813713        0.3023572
## X540 2.992695 2.733159             3.029545        0.3015681
```

3. Retain a number of best predictors equal to that identified by the lasso. How many genes do the two methods have in common?

```
## top 23 variables names according to random forest
which.rf <- rownames(sorted.imp.var)[1:23]

## variable names selected by lasso
which.lasso  <- which(lasso.coef[-1,1]!=0)
which.lasso2 <-paste0("X",which.lasso) # add X to name

## checking intersection number of covariates from rf selected by lasso as well
## which.rf %in% which.lasso2
## sum(which.rf %in% which.lasso2)
length(intersect(which.rf, which.lasso2))

## [1] 12

## They have 12 variables in common
```

4. Compute the test error estimate of the random forest classifier.

```
## Compute the test error ####
yhat.rf <- predict(out.rf,newdata = genes.df[-train,])
table(yhat=yhat.rf,genes_labs[-train])

##
## yhat 1 2
##    1 5 0
##    2 0 5

## The test error estimate is zero
```

5. In this case, which classification method would you prefer and why? both method have zero error: there are reasons for both methods. logistic regression with lasso penalization allows us to have more interpretable results. But when the number of variable is very large random forest tends to perform better.
In this case we would go with lasso however.

NB: say something smart here, not only the most obvious least error