Supervised statistical learnning

5 aprile 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. *supervised statistical learning* (focus on the course) involves building a model for predicting an output based on one or more inputs. Here we have a *response variable*;
2. *unsupervised statistical learning* is used to describe the associations and patterns among a set of input measures and there is *no outcome measure*.

Remark 1 (Application of statistical learning). Some examples:

- identify the risk factors for prostate cancer (based on clinical and demographic variables).
- predict whether a patient hospitalized due to a heart attack will have a second heart attack (based on demographic, diet and clinical measurements).
- identify the numbers in a handwritten ZIP code (from a digitized image).

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

1. *an outcome measurement*, usually quantitative or categorical that we wish to predict; if the outcome measurement is *quantitative* this is a *regression problem*, otherwise if qualitative it's a *classification problem*

2. a set of *features*/covariates that will be used for the prediction;
3. a *training dataset*, in which we observe both the outcome and feature measurements for a set of units.

We build a **prediction model**, or *learner*, which will enable us to predict the outcome for new unseen objects. A **good learner** is one that accurately predicts the outcome.

Important remark 3 (Truth bombs). We have that:

- there is no single best method for all the problems:
  -
  - different problems have different features and what is best for one problem is not best for another
  - furthermore different tools are best suited for different tasks: eg certain tools better for prediction other for inference
- 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 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
- There is a **tradeoff between bias and variance** of the estimator (in the reducible error part): the more flexible a model is the less is the bias but more is the variance due to the sample we used in the training/estimate.
- 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
- Tipically one compare different parametrization and different method developed in the training set on a new validation set to check what is the error of different model/parametrization (considered that unfortunately it's only a single test set).

1.2 Estimating f and the bias variance tradeoff

Important remark 4 (General framework). Generally speaking we:

- observe a response Y and p different predictors, $X_1, \ldots, X_p$
- assume that there is some relationship between Y and $\mathbf{X} = (X_1, \ldots, X_p)$, 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 $X_1, \ldots, X_p$ in the population

  - 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)

In this formulation, f represents the systematic information that $\mathbf{X}$ provides about Y.

1.2.1 Reason for estimating f

Important remark 5 (Reasons to estimate f). Two main reasons:

1. **prediction**: find a rule to guess a costing or difficult variable on new observation.
   In many situations, data on a set of inputs $\mathbf{X}$ are available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can 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*, in the sense that one is not typically concerned with the exact form of $\hat{f}$, provided that it yields accurate predictions for Y.

2. **inference**: study the relation between X and Y.
   Other than make prediction we can estimate f to understand how Y changes as a function of $X_1, \ldots, X_p$. Contrary to prediction here $\hat{f}$ cannot be treated as a black box, because we need to know its exact form.
   Example questions could be: which predictors are associated with the response? Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

1.2.2 Methods for estimating f

Definition 1.2.1 (Training dataset)**.** It's the set of n different data points we observe on which we train, or teach, the chosen method how to estimate f. Here

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

2. y_i will be the response variable for the i-th observation.

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

Remark 2. The goal is to apply a statistical learning method to the training data to estimate the unknown function f therefore to find a function $\hat{f}$ such that $Y \approxeq \hat{f}(X)$ for any observation (X, Y).

Remark 3 (Objective and methods of estimate). Depending on whether our ultimate/main **goal** is prediction or inference different *methods* for estimating f may be appropriate:

- 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 6 (Simple vs complex: prediction vs interpretability). 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* than other (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 4. In the following we highlight the two main methods of estimation: parametric and non-parametric methods

1.2.2.1 Parametric methods

Definition 1.2.2 (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 7 (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 8 (Pros/cons). Parametric estimation is much simpler. The potential disadvantage is that the model we choose will usually not match the true unknown f. However

- if the chosen model is far from the true f (eg too simple), our estimate will be poor (simpler model tends to have more bias, but will be less sensitive to a changed training sample)

- fitting a more flexible/complex to better approximate the functional form of f requires estimating a greater number of parameters, which exposes to risk of *overfitting* the data, which essentially means they follow the errors, or noise, too closely (more exposed to variability of the training sample and then sucks on prediction of new data)

So in parametric methods we need to be careful on model/parameters tuning.

1.2.2.2 Nonparametric methods

Definition 1.2.3. Nonparametric methods do not make explicit assumptions about the functional form of f.

Important remark 9 (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 10 (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 5*.* There is no single method that dominates all others over all possible data sets: one specific method may work best on a particular dataset, other methods may work better on a similar but different data set.
Hence it is important to decide for any given dataset which method produces the best results.
We need some way to measure how well its predictions actually match the observed data: there are two type error which can be computed

Definition 1.2.4 (MSE (training))**.** Defined as

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

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. 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.
It is:

- computed on the training data and therefore is also referred as *training MSE*;

- it is actually used (is minimized) during model building.

Important remark 11. However, 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 1.2.5 (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 12 (Choosing the method with best test MSE)*.* We want to choose the method/model that gives the **lowest test MSE**, as opposed to the lowest training MSE.
We avoid choosing on training MSE provides because we would have no guarantee that such method will also have the lowest test MSE. Furthermore, 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 13. Regardless of whether or not overfitting has occurred, we almost always expect the training MSE to be smaller than the test MSE because most statistical learning methods either directly or indirectly seek to minimize the training MSE (eg OLS).

1.2.4 The bias-variance tradeoff

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

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

where y_0 is the random variable representing one point we want to predict in the test set, $x_0 = (x_{01}, \dots, x_{0p})$ 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.

Important remark 14. The components of the test error are:

1. the **reducible error**. In general, $\hat{f}$ 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 (how close we get to the functional form in the population): this is part of the reducible error and decreases with model complexity

[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)

(b) the **variance** of the estimator (variability of fitting the model in different training set): this is part of the *reducible error* and increases with model complexity

2. the **irreducible error**: error due to ε (it is its variance). $y_0 = f(x_0) + \varepsilon$ is also a function of ε, which by definition cannot be predicted using x_0. Therefore variability associated with ε 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 (that may contain unmeasured variables that are useful in predicting Y or unmeasurable variation).
It's error that i may have if I change repeatedly my training set. The expected test MSE can never lie below $\text{Var}[\varepsilon]$ the irreducible error.

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 $\mathbb{E}[f(x_0)] = f(x_0)$; 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:

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

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

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

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

$$= \underbrace{\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)^2\right]}_{\text{Bias}\left(\hat{f}\right)^2} + \underbrace{\mathbb{E}\left[\varepsilon^2\right]}_{\text{Var}[\varepsilon]} + \underbrace{\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)^2\right]}_{\text{Var}[\hat{f}]} + \dots$$

$$\dots + 2\,\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)\right]\underbrace{\mathbb{E}[\varepsilon]}_{=0} + 2\,\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right]\right)\right]\underbrace{\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)\right]}_{=0} + 2\underbrace{\mathbb{E}[\varepsilon]}_{=0}\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right] - \hat{f}\right)\right]$$

$$= \underbrace{\text{Bias}\left(\hat{f}\right)^2 + \text{Var}\left[\hat{f}\right]}_{\text{reducible}} + \underbrace{\text{Var}[\varepsilon]}_{\text{irreducible}}$$

In the last step:

- all the double products disappears for different reasons: one is that $\mathbb{E}[\varepsilon] = 0$, 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

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

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

  and constitutes the second part of the reducible error;

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

□

Important remark 15 (The bias-variance tradeoff). In the reducible error part we have a bias-variance tradeoff: the more flexible a model is the less is the bias (we're approximating) but chances are our estimator is very variable (approximating to much at data).
The relative rate of change of variance and bias determines whether the test MSE increases or decreases.
Thus in order to minimize the expected test error, we need to select a statistical learning method that reaches a good compromise and for which both variance and squared bias are low:

- 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.

This is referred to as a trade-off because it is easy to obtain a method with extremely low bias but high variance (for instance, by drawing a curve that passes through every single training observation) or a method with very low variance but high bias (by fitting a horizontal line to the data).

- *Variance* 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 statistical methods have higher variance

- *Bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model (for example, imposing a linear relationship in a real-life problem). Generally, more flexible methods result in less bias

1.3 Evaluating MSE

Important remark 16. How can we select the method that minimizes the test MSE?

- if *we have a separate test dataset* available we can select the learning method for which the test MSE is smallest.

- 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.

1.3.1 Validation set approach

The *validation set approach* is a very simple strategy to estimate the test error associated with fitting a particular statistical learning method on a set of observations.

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* or hold-out set;

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

Important remark 17 (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 for the model fit on the entire data set;
2. the estimate of the test error rate can be highly variable depending on which splitting (which observations are in the training set and which in the validation set).

Important remark 18 (Alternative three way splitting)*.* If we are in data-rich situation, and there is a model selection problem (which is usually the case) the best approach is to randomly divide the dataset into *three* parts:

- a *training* set (eg 50%): used to fit the models;
- a *validation* set (eg 25%): used for model selection by estimating/comparing prediction errors (eg we train different models and check here which method to use);
- a *test* set (eg 25%): used to check the error for the final chosen model in a fresh dataset.

Otherwise splitting in two developing on the training and using the test set repeatedly (both for choosing the model, with smallest error, and having the same error as final error estimate), this would make the test error of the final chosen model 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 and refitting a model of interest on each sample.

Remark 6. Such an approach may allow us to obtain information that would not be available from fitting the model only once using the original training sample. They can be computationally expensive, but generally not prohibitive. The main method we use is crossvalidation.

Important remark 19 (Crossvalidation (CV)). Cross-validation is a resampling method that can be used:

- **model assessment**/estimate MSE: to estimate the test error associated with a given statistical learning method in order to evaluate its performance, eg to determine how well a given statistical learning procedure can be expected to perform on independent data;
- **model selection/fine tuning**: to identify the procedure (among several statistical learning methods) that results in the lowest test error; or even different parametrization of the same model (eg select the appropriate level of flexibility)

1.3.2.1 Leave-one-out cross validation (LOOCV)

Remark 7. A first cv method, somewhat similar to the validation set approach.

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

- the single observation (x_1, y_1) composes the validation set;
- the remaining observations $(x_2, y_2), \dots, (x_n, y_n)$ make up the training set on which the model is trained
- the model trained is used to predict $\hat{y}_1$
- 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 a poor estimate because it is highly variable, since it is based upon a single observation, (x_1, y_1).

So we have n errors that have to be sumup: the LOOCV estimate for the test MSE is the average of these n test error estimates

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

Important remark 20 (Pros/cons vs validation set approach). Compared to validation set approach, LOOCV:

- has less bias: the method is fitted using training sets that contain $n - 1$ observations, almost as many as are in the entire data set;

- tends not to overestimate the test error rate;
- it has no randomness, so performed multiple times will always yield the same results;
- has prediction for different units which are somehow correlated being developed on similar training set: thus the mean error have more variability (variance of the sum/mean has the covariance part as well);
- can be be expensive to implement (time consuming if n is large).
  In case of linear regression, a shortcut makes the cost of LOOCV the same as that of a single model fit, by calculating the error this way

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

  this is the ordinary MSE (with $\hat{y}_i$ is the i-th fitted value from the original least squares fit) except the i-th residual is divided by $1 - h_i$ where and h_i is the leverage statistic which can be found in the i-th place of the diagonal of the hat matrix $\mathbf{H}$

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

  which is the one that premultiplied to the outcome returns the prediction ($\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$)

1.3.2.2 k-fold cross-validation

Remark 8. An alternative to LOOCV is k-fold CV.

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

- the fold in turn 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 procedure is repeated k times (one for each fold considered as validation set) ending in k MSE_i, and the k-fold CV estimate is computed averaging them:

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

Remark 9. LOOCV is a special case of k-fold CV in which $k = n$.

Important remark 21 (k-fold vs LOOCV). With respect to LOOCV, k-fold cv ($k < n$);

- has computational advantages;
- 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;

- often gives more accurate estimates of the test error
- estimates have more bias, therefore from the perspective of bias reduction, it is clear that LOOCV is to be preferred (training set is smaller in k-fold):
  - 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).
- estimates are more stable since have less correlated prediction (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);
  - since the mean of many highly correlated quantities has higher variance than mean of uncorrelated/less correlated quantities, the test error estimate from LOOCV tends to have higher variance than that from k-fold CV.

1.3.2.3 CV error for classification

Important remark 22 (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 10. Consider a classification problem with a large number of predictors, (eg genomic or proteomic applications) and we need to choose among them.

Remark 11 (The wrong way). A wrong typical approach is:

1. find a subset of good predictors with fairly strong (univariate) correlation with the class labels;
2. use just this subset of predictors to build a multivariate classifier;
3. use cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model.

This is not a correct application of cross-validation because these units are already used/seen and estimating the error is biased: selecting the best variable at first will falsely low the error estimate in CV because we already optimized. 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.

Important remark 23 (The right way)*.* To do model selection using CV:

1. divide the samples into K cross-validation folds (groups).
2. for each fold $k = 1, 2, \dots, K$:
   (a) find a subset of good predictors with fairly strong (univariate) correlation with the class labels, using all of the samples except those in fold k;
   (b) using just this subset of predictors, build a multivariate classifier, using all of the samples except those in fold k;
   (c) use the classifier to predict the class labels for the samples in fold k.

The error estimates from step 2(c) are then accumulated over all K folds, to produce the cross-validation estimate of prediction error.

Capitolo 2

Classification

2.1 Classifier evaluation evaluation

Remark 12. 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 $\{(x_1, y_1), \dots, (x_n, y_n)\}$, where the outcome $y_1, \dots, y_n$.

2.1.1 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 $y_i = \hat{y}_i$.

The equation 2.1 computes the training error rate. A good classifier is one for which the test error is smallest.

2.1.2 Other measures

Important remark 24. Within a binary classification context, and especially in case of unbalanced groups it's important to consider other measures for performance other than simple accuracy/test error.

Important remark 25 (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.
- TN (true negatives): negative units correctly labeled

	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

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

Remark 13. 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:

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

and coincides with what presented in the previous section.

Important remark 26 (Class imbalance problem). When one class (tipically the main class of interest, eg positive class), is rare looking only at overall classification performace can be misleading.
By looking at the overall error rate we could have good performance, but we can only note this if we look at the confusion matrix or at other non overall measures. Therefore It is useful to look at confusion matrix in everycase, don't rely on general accuracy measures only.
Furthermore mMany classifier minimize the overall error rate, so tend to privilege the classification on the most represented group). If the minority class is very tiny the classifier could put all the units erroneously in the most represented class.

Remark 14. In general: problem is that looking all together may not optimize the performance in the class we're more insterested in.

Example 2.1.1. For example, there may be a rare class, such as "cancer". An accuracy rate of, say, 97% may make the classifier seem quite accurate, but what if only, say, 3% of the training units are actually cancer? Clearly, an accuracy rate of 97% may not be acceptable - the classifier could be correctly labeling only the noncancer samples and misclassifying all the cancer ones.

Important remark 27. Alternatively one can:

- present different statistics (which focus on component separately) such as these duos, alternatively:
  1. sensitivity + specificity;

2. 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

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

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

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

(can also be named recall or true positive rate)

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:

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

2.2 Bayes classifier

Important remark 28. The Bayes classifier:

- is the **gold standard classifier**: it's possible to show that the test error rate is minimized, the Bayes classifier produces the lowest possible test error rate (called *the Bayes error rate*, basically the unreducible error);
- unfortunately is a **theoretical 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).
- many approaches are inspired by this classifier and attempt to estimate the conditional distribution $\mathbb{P}(Y|X)$ to classify a given observation to the class with highest estimated probability;

- being gold standard is useful for comparing with new methods in research
-
- the name come from the fact that it simply applies the bayes theorem to choose the population;
- is not the naive Bayes which we see in the following.

Definition 2.2.1. The Bayes classifier 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

$$\mathbb{P}\left(Y = h|X = x_0\right) \quad \text{is largest.}$$

The probabilities $\mathbb{P}\left(Y = h|X = x_0\right)$ are calculated according to Bayes theorem

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

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);
- $\mathbb{P}\left(Y = h|X = x_0\right)$ is the posterior probability of belonging to population h given we have observed x_0

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

Remark 15. 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 hit the prevision/classification is $\max_h \mathbb{P}\left(Y = h|X = x_0\right)$ so the probability of a miss (the error rate at $X = x_0$) will be

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

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)

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

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.

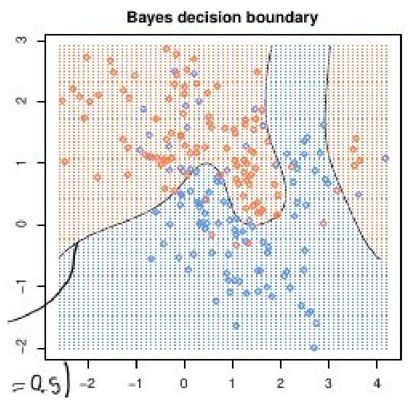


Figura 2.1: Bayes decision boundary.

Example 2.2.3 (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:

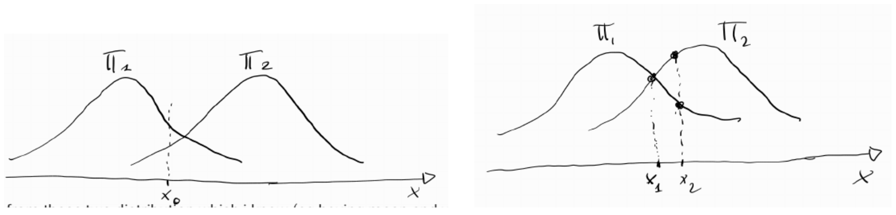
```
library(mvtnorm)
red <- rmvnorm(100, mu1, sigma1)
blue <- rmvnorm(100, mu2, sigma2)
# densities can be then obtained with dmvnorm function
```

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.

Example 2.2.4 (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



(a) (b)

Figura 2.2: Bayes univariate

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.

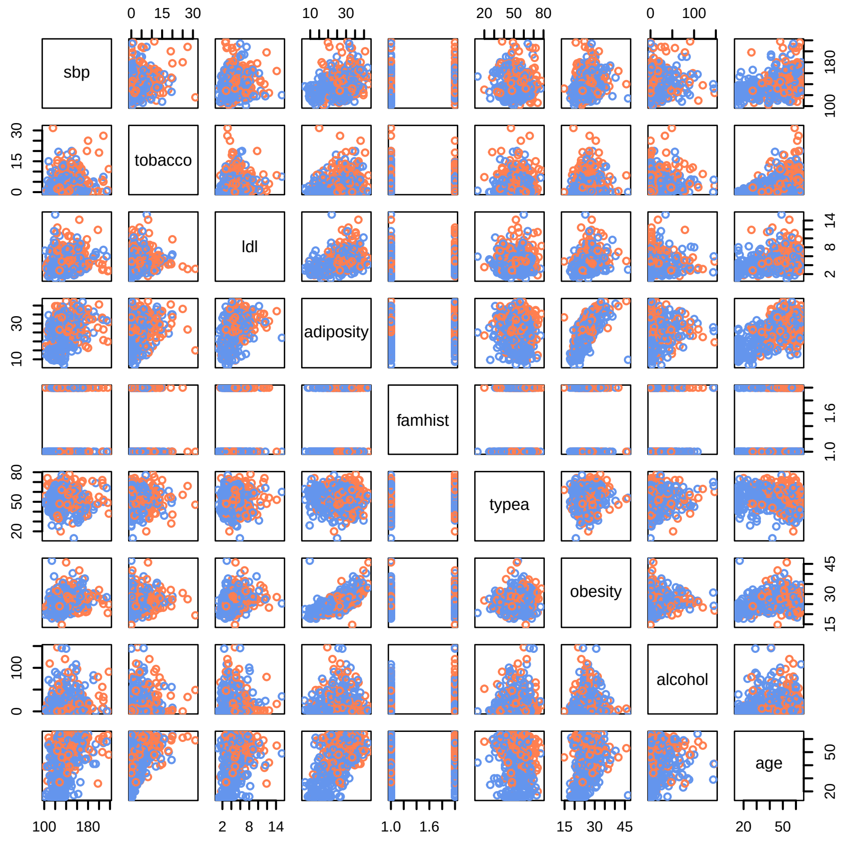
2.3 Classification dataset: SAheart

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 29 (Why not linear regression?). Considering a dichotomic outcome Y, how can we model the relationship between X and Y, that is $\mathbb{P}(Y = 1|X) = p(X)$?

- if we use a linear regression model we could use the outcome variable directly to represent these probabilities

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

  and predict chd if $\hat{Y} > 0.5$ or no chd otherwise. However 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

- to avoid the problems with linear regression rather than modeling the dichotomic response Y directly with linear regression, logistic regression *models the probability* that Y *belongs to a particular category* given the covariates. We must model $p(X)$ using a function that gives outputs between 0 and 1 for all values of X. In logistic regression, we use the *logistic function*, so probability is modeled like

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

  Such model can be estimated via maximum likelihood.
  After estimation for any covariate pattern we can have a predicted probability that will range between 0 and 1. For example, one might predict chd = 1 for any individual for whom the probability is larger than 0.5. Alternatively, lower or higher thresholds can be chosen.
  Now, for high level of X we predict a probability close to, but never above, one. The logistic function will always produce an S-shaped curve, and so regardless of the value of X, we will obtain a sensible prediction.

Important remark 30 (Log odd and Linear predictor). After a bit of manipulation, we find that

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

The quantity $p(X)/(1-p(X))$ is called the *odds*, and can take on any value between 0 and ∞: values close to 0 indicate very low probability of event, ∞ very high.
By taking the logarithm of both sides, we arrive at

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

The left-hand side is called the *log-odds* or *logit.*
We see that 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 31 (Relationship between X and $\mathbb{P}(Y=1)$). 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

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

$\beta_0, \ldots, \beta_p$ are estimated 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 = 'response')

##        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
## ---------
folds_err <- rep(NA, 5) # contains cross validation error
yhat <- rep(NA, nrow(SAheart)) # predicted value for each observation
set.seed(1234)
folds <- sample(1:5, nrow(SAheart), replace = TRUE)
## select only
sel_df <- 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 1:5){ # go through the different folds
    x_train <- sel_df[folds != i, ]  # training set X, all but the considered folds
    x_test <- sel_df[folds == i, ]   # test set X
    y_test <- sel_df$chd[folds == i] # test set Y to compute error
    fold_mod <- glm(chd ~ ., data = x_train, family = binomial) # fold model using th
    phat <- predict(fold_mod, newdata = x_test, type = "response") # probs of success
    y_hat <- ifelse(phat > 0.5, 1, 0) # dichotomized test predictions
    folds_err[i] <- mean(y_test != y_hat) # estimate/store this fold's error
    yhat[folds == i] <- y_hat # storing the predictions
}

folds_err

## [1] 0.3068182 0.2941176 0.3333333 0.2111111 0.2340426

mean(folds_err) # overall 5-fold cv error

## [1] 0.2758846

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

##      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 32 (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)
- we 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 considere 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 $x^t\mathbf{w}$ (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 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:

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

  $D(w)$ 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 33. Some remarks:

- 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);o

- the distributions were summarised in terms of their first and second-order moments: if the underlying distributions could be completely described by those moments (gaussian we're looking at you) we have optimality in the sense that rule that we find as result is actually approximating the bayes error done by the bayes classifier (the gold standard).

- in general such distributions are termed elliptical distributions and by far the most important special case is the multivariate normal distribution.

- 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):

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

So for these distributions an optimum classification can be obtained by comparing 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 the threshold change to 0) we have:

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

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

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

    obtained before (before introducing normality hypothesis). Here S^{-1} is the sample estimate of $\mathbf{\Sigma}$.

By following two different approaches, one based on Fisher that yields $D(\mathbf{w})$, and the other based on Bayes rules, in case the population are infact gaussian, we find something similar.

The classification rule will consist of comparing $x'\Sigma^{-1}(\mu_1-\mu_0)$ with a threshold (which depends on the μ_k , Σ, the priors and the costs)

Important remark 34. 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.
- since Fisher's method is based on second-degree terms in the $\mathbf{x}$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: using OLS, it is possible to formulate a class membership pre- diction 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 $\textcircled{2}$, $\overline{x}_{(1)2}$ 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 $\textcircled{p}$.
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:

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

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

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

To recap $\boldsymbol{\beta}$ is proportional to $\mathbf{S}^{-1}(\overline{\mathbf{x}}_{(1)} - \overline{\mathbf{x}}_{(0)})$ (which we found in the classifier before assuming gaussianity). So there are connection between all the three; for this reason the classification they produce is somewhat related (not the same but correlated).

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

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

- without making the multivariate normal assumption (without using bayes) 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:

- B is the between variance/covariance matrix,

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

  with two classes the first ratio was simplified

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

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

That is we are trying to find that direction 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

$$\mathbf{Bw} - \lambda\mathbf{Sw} = 0.$$

This two-sided eigenvalue equation has solutions for values of λ satisfying $|B - \lambda S| = 0$. That is has solutions for eigenvalues of $S^{-1}B$.
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 $\mathbf{w}_k^T\mathbf{Swk} = 1$ then $\mathbf{WSW}^T = \mathbf{I}$ where w_k^T is the kth 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.

Remark 16. In general

- Overfitting in LDA: In the practical case, the higher the ratio of parameters p to number of samples n, the more we expect this overfitting to play a role.

- if there are too few positive predicted the researcher could diminish the treshold to mark positive: however this make worse prediction on the negative (more false positive) and overall error rate should be checked

2.5.4 Example

Example 2.5.1. 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 #### - 2024-03-06
library(lbdatasets)
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

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

## Doing LDA
library(MASS)

##
## Attaching package:  'MASS'
## The following object is masked from 'package:lbdatasets':
##
##     anorexia

out.lda <-lda(chd ~ ., data=SAheart[train,])
out.lda
```

```
## Call:
## lda(chd ~ ., data = SAheart[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

## Error
pred.lda <- predict(out.lda, newdata=SAheart[-train,])
## pred.lda
table(Actual=SAheart$chd[-train], LDA=pred.lda$class)

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

mean(SAheart$chd[-train]!=pred.lda$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?

```
## Comparison with Logistic Regression ####
out.lr <- glm(chd ~ ., data=SAheart[train,], family=binomial)
summary(out.lr)

##
## Call:
## glm(formula = chd ~ ., family = binomial, data = SAheart[train,
##     ])
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -5.754107   1.930020  -2.981  0.00287 **
## sbp              0.013621   0.009185   1.483  0.13808
## tobacco          0.073181   0.038374   1.907  0.05652 .
## ldl              0.271471   0.088687   3.061  0.00221 **
## adiposity        0.056835   0.046894   1.212  0.22551
## famhistPresent   0.676832   0.340035   1.990  0.04654 *
## typea            0.048751   0.019762   2.467  0.01363 *
## obesity         -0.148012   0.076710  -1.930  0.05367 .
## alcohol         -0.001007   0.006402  -0.157  0.87508
## age              0.020755   0.017144   1.211  0.22604
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 291.22  on 230  degrees of freedom
## Residual deviance: 230.23  on 221  degrees of freedom
## AIC: 250.23
##
## Number of Fisher Scoring iterations: 4

phat <- predict(out.lr, newdata = SAheart[-train,], type="response") #response t
yhat.lr <- ifelse(phat > 0.5, 1, 0)

table(Actual=SAheart$chd[-train],LR=yhat.lr)

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

mean(SAheart$chd[-train]!=yhat.lr)

## [1] 0.2683983
```

The LDA and logistic regression predictions are almost identical.

2.5.5 Statquest explanation of LDA

Usa le n covariate per creare un unico nuovo asse (e proiettare i dati su questo in maniera tale che siano i più separati possibili): il nuovo asse è creato seguendo due criteri contemporaneamente. Vogliamo che i dati proiettati sul nuovo asse siano

- massimizzare la differenza tra le medie dei due gruppi
- minimizzare la variabilità entro ciascuna categoria

Per considerare i due criteri contemporaneamente nel ratio

$$\frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}$$

che se si massimizza assolve la funzione di massimizzare e minimizzare le robe di sopra let's call the distance between the two means $d = \mu_1 - \mu_2$ and then the maximization criteria becomes $\frac{d^2}{}$. Nothing changes if we have more variable in the two class case.
In case we have three categories in the outcome (2/n covariates) the two things change, but barely:

- how one measures the distance between means: instead of measuring the distance between the two means, we find a central point of the means, then we measure the distances d_1^2 , d_2^2, d_3^3 of each groups mean with respect to the central mean. The equation we want to maximize becomes

$$\frac{d_1^2 + d_2^2 + d_3^2}{s_1^2 + s_2^2 + s_3^2}$$

- for three groups outcome the object is to create two axes to create a plane to separate the data (i tre punti medi formano un piano) not a line/single dimension. Proiettiamo i dati su questo nuovo piano. Il primo asse è quello più importante per la separazione, il secondo è piu secondario

2.6 Naive Bayes Classifier

2.6.1 The classifier

ß Naive Bayes and k-nearest neighbor are two simple but powerful classifier.

Remark 17. Starting from Naive Bayes, in a practical situation if we would to apply the Bayes classifier (gold standard classifier) we would need to have the multivariate density for each group we want to classify.
Estimate them in a multivariate setting is not that easy, especially if we don't assume them gaussian (eg we use nonparametric methods)
Naive Bayes classifier is a technique that has remained popular over the years: it is especially **appropriate when** the dimension p of the feature space is high, making density estimation unattractive.

Definition 2.6.1 (Naive Bayes)**.** The naive Bayes model assumes that given 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);
- each univariate density f_{hj} (the variable j in group $Y = h$) can be supposed gaussian (so we estimate its parameter only) otherwise we estimate 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 35. While the assumption of independence of feature within class is generally not true it does simplify the estimation dramatically and at the same time the classifiers often outperform far more sophisticated alternatives.

Remark 18. 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 19. This is the most widely used nonparametric approach

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

Important remark 36 (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 37 (Cons). We have:

- 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.
- there are discontinuities in its probability estimates at each 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 38 (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 20. Aside from naive bayes which handle different variables separatedly, if this is not the case the curse of dimensionality in extension to multivariate histograms
If each variable is separately divided into 10 cells, and if there are 10 variables, then there are 10^{10} cells in the multivariate space. Most of the multivariate cells will be empty, so 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 39. Kernel density estimation is a generalisation of the histogram ap- proach which overcomes:

- the discontinuity problem
- the problem of where to locate the cells
- the multivariate nature of most classification problems.

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, $f(x)$, in the point x from a sample of n points randomly drawn from f: that is to obtain $\hat{f}(x)$;
- the greater the proportion of the sample points lying within the vicinity of x, the higher should be $\hat{f}(x)$.
- if we take an interval/bandwidth h centered on x we could estimate the probability $f(x)$ 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.
  In this case bandwidth is the width of a moving window on the real line centered on point of interest which determines which points are considered

- problem with the previous approach is that whenever a point enter/leaves the interval $[x-h, x+h]$ there will be a discontinuous jump of $\pm\frac{1}{n}$, so the shape would not be smooth (similarly to histograms)
- the 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*: with size that decay as the distance between the center/point of interest x and point considered increases. The more the point in the bandwidth is near to the center, the more will it contribute to density of the center
  Let $X_1, X_2 \dots, X_n$ denote a sample of size n from a random variable with density f; density estimate in the center x, using the sample X_i, is
  $$\hat{f}x = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)$$
  where
  - K (the *kernel function*) is usually a unimodal probability density function, with unitary integral, symmetric around 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.
- a first decision is *which kernel to adopt*: there are several, but the impact is not so much (a much more important choice is the bandwidth h:
  - *gaussian*: in this case the parameter h is the standard deviation and the function is the standard normal density function
  - *Epanechnikov*:
    $$K(x) = \frac{3(1-x^2)}{4} \qquad -1 \le x < 1$$
  - *triangle* kernel
    $$K(x) = 1 - |x| \qquad -1 \le x < 1$$
- the second decision is the *width/spread of the kernel*: h is sim,ilar 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).
- of we adopt a proper density kernel (such as the gaussian) the estimation resultimng will be a proper density since the kernel estimator inherits the analytic properties of the function K.
  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)

## remove the response and the categorical variable
x <- SAheart[,-c(5,10)]
y <- SAheart[,10]
n <- nrow(SAheart)

k <- 5
set.seed(1234)
folds <- sample(1:k, n, replace = TRUE)
err.cv.g <- err.cv.k <- rep(NA, k) # CV error (one for each fold) for the two methods
y.hat.g  <- y.hat.k  <- rep(NA, n) # predicted labels (one for unit) for the two methods

## prima di fare andare il ciclo per intero assegnare i = 1 e
## testare il codice
for (i in 1:k){ # Crossvalidation
    x.train <- x[folds!=i,]
    y.train <- y[folds!=i]
    y.test <- y[folds==i]
    x.test <- x[folds==i,]

    ## Naive Bayes with kernel density (usekernel = TRUE)
    out.nb.k <- NaiveBayes(x = x.train,
                           grouping = as.factor(y.train),
                           usekernel = TRUE)
    temp <- predict(out.nb.k, newdata = x.test)$class # class to extract the predicted label
    err.cv.k[i] <- mean(y.test != temp)
    y.hat.k[folds==i] <- temp

    ## Naive Bayes with Gaussian density (usekernel = FALSE)
    out.nb.g <- NaiveBayes(x = x.train,
                           grouping = as.factor(y.train),
                           usekernel = FALSE)
    temp <- predict(out.nb.g,newdata = x.test)$class
    err.cv.g[i]<-mean(y.test!=temp)
    y.hat.g[folds==i] <- temp
}

addmargins(table(Actual=y,NB.k=y.hat.k))

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

mean(err.cv.k)

## [1] 0.3422304

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

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

mean(err.cv.g) # the gaussian returns the smallest error

## [1] 0.3054338
```

2.7 k-NN Classifier

Remark 21. It's a simple method which does not require to train a proper model. It can be used for regression as well.

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 the neighbors k points in the training data that are closest to x_0 , represented by $\mathcal{N}_0$;
2. estimates the conditional probability for class h as the fraction of points in $\mathcal{N}_0$ whose response values equal h:

$$\mathbb{P}\left(Y = h | X = x_0\right) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = h)$$

3. Finally, k-NN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.

Remark 22. 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 40. Some remarks

- it's a memory based classifier and require no model to be fit.
- we need to choose k, the number of neighbor: let it be *odd*

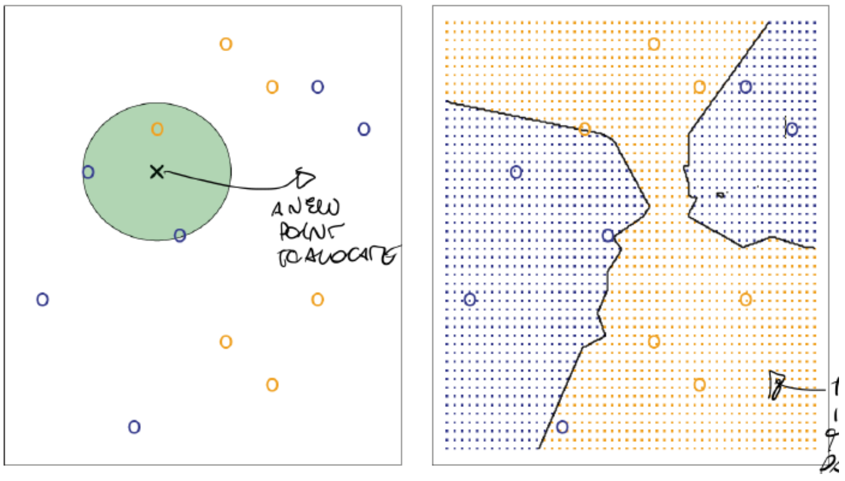


Figura 2.3: knn1

- given a query point x_0 , we find the k training points $x_{(r)}$, $r = 1, \dots, k$ closest in **distance** to x_0 , and then classify using majority vote among the k neighbors (in case of *ties*, broken at random), so
  - we need to choose a *distance*: assuming the features are real-valued, we use Euclidean distance $d(i) = \left\|x_{(i)} - x_0\right\|$
  - 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
- in general if number of variables is greater than number of observation, distances we find are not reliable
- despite being simple k-NN can often produce classifiers surprisingly close to the optimal Bayes classifier

Important remark 41 (choice of k). It has a drastic effect on the k-NN classifier obtained. In figure 2.4 various choices for k with the bayes optimal classifier in red (which is the same in the three images) and the area represent the classification done with knn

- 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. (In general, as we use more flexible classification methods, the training error rate will decline but the test error rate may not.)
- 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:

- if a test set is available we calculate there the classification error
- otherwise (as always) we choose k by crossvalidation

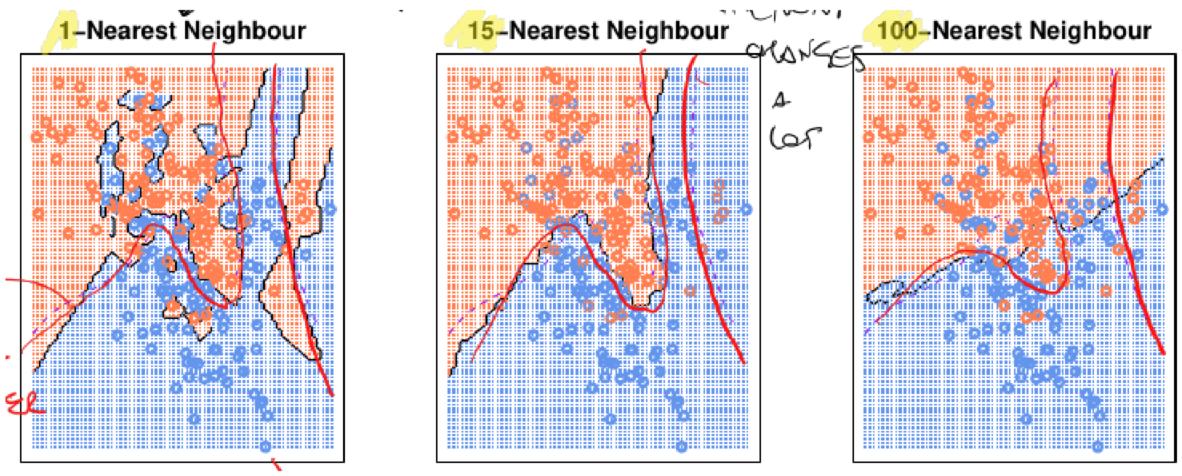


Figura 2.4: knn2

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)
colnames(SAheart)

##  [1] "sbp"       "tobacco"   "ldl"       "adiposity" "famhist"   "typea"
##  [7] "obesity"   "alcohol"   "age"       "chd"

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

## 1) splitting in training set and validation set
## -----------------------------------------------
set.seed(1234)
index <- sample(1:n, size = ceiling(n/2)) ## extract the index of approximately half
## otherwise we could have done: sample(c(TRUE, FALSE), size = ceiling(n/2), replace
train <- x[ index,]
valid <- x[-index,]
train.std <- scale(train, center = TRUE, scale = TRUE) # standardize X in train set
## (btw both TRUE could be omitted being default)
ytrain <- SAheart$chd[index]
ntr <- nrow(train.std)
```

```
## 2) choosing optimal k
## ---------------------
## here we use only the train dataset which is splitted
## in train and test. We don't touch the validation dataset which will
## be used at the end
set.seed(1234)
folds <- sample(1:5, ntr, replace = TRUE) # labels for crossvalidation (note done only on the t
k <- c(1, 3, 5, 11, 15, 25, 45, 105) # n of neighbors we want to consider
err.cv <- 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
    x.test <- train.std[folds == i,]
    x.train <- train.std[folds != i,]
    y.train <- ytrain[folds != i]
    y.test <- ytrain[folds == i]
    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])
        err.cv[i,j] <- mean(yhat != y.test)
    }
}
err.cv # 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(err.cv) # 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(err.cv))) # best k using 5-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 validation set and obtaining the error estimate.

## in this phase, if we standardized the x in the training phase
## we should standardize the y based on the values of x train

## in general if we'standardize the train set before applying the
## estimation procedure, then when we go to the 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)

## compute mean and stddev of X in the training
mean.tr <- colMeans(train)
sd.tr <- apply(train, 2, sd)
## standardization of x in the validation set
valid.std <- valid
for (j in 1:ncol(valid))
    valid.std[,j] <- (valid[,j]-mean.tr[j])/sd.tr[j]
# prediction on the validation set
yhat.valid <- knn(train = train.std,  # train: we use all the train data
                  cl = ytrain,        # class: all the y from the train data
                  test = valid.std, # test: standardized x on the validation set
                  k = 11) # optimal k found before

## error on the validation set (-index)
addmargins(table('pred' = yhat.valid, 'actual' = SAheart$chd[-index])) # confusion ma

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

mean(yhat.valid!=SAheart$chd[-index]) # error estimate

## [1] 0.3506494

(24+57)/231

## [1] 0.3506494
```

Capitolo 3

Model selection and regularization

Important remark 42. In the regression setting on p covariate with n observation:

- if $n >> p$ the estimates tend to have low variance;
- if n is not much larger than p, then there can be a lot of variability in the fit, resulting in overfitting and consequently poor predictions on test units;
- if $n < p$, then there is no longer a unique least squares coefficient estimate: the variance is infinite so the method cannot be used at all.

Three solution are:

- identifying a subset of relevant covariates (*model/subset selection*) and use only them;
- fit the model on all the p predictor but then *penalize/shrink estimates* toward 0 by using ridge (only shrink) or lasso (can set to 0) and this reduce variability of estimates (the model could be a slightly more biased but the estimates are less variable).
- create new variable in lower dimension starting from the first initial p (*dimension reduction*): projecting the p predictors into a M-dimensional subspace, where $M < p$, obtained by computing M different linear combinations, or projections, of the variables, which will then be used as predictors in the regression model

which are the focus of this chapter

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.

```r
options(width = 100)
library(lbdatasets)
head(prostate)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa train
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829  TRUE
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189  TRUE
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189  TRUE
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189  TRUE
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636  TRUE
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678  TRUE
```

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 Subset selection

3.2.1 Best subset selection

Remark 23. Having p predictor we fit a separate least squares regression for each possible combination of the p predictors identifying the best.

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 $k = 1, 2, \dots, p$:
    - 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 $\mathcal{M}_0, \dots, \mathcal{M}_p$ using metrics such as crossvalidated prediction error, C_p (AIC), BIC or adjusted R^2

Important remark 43 (Pros/cons)*.* As pros:

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

As cons:

- *computational limitations*: there are 2^p models that involve subset of p predictors so the number grows rapidly as p increases.
- 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, we're a bit overfitting.

3.2.2 Stepwise selection

Remark 24. Stepwise selection explores a far more restricted set of models compared to best subset selection. The main approaches are: forward, backward, hybrid.

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 $k = 0, \dots, p-1$:
   - 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 $\mathcal{M}_0, \dots, \mathcal{M}_p$ using metrics such as crossvalidated prediction error, C_p (AIC), BIC or adjusted R^2

Important remark 44 (Pros/cons)*.* Pros:

- it requires estimating significantly less models: involves fitting one null model, along with $p-k$ models in the k-th iteration, for $k = 0, \dots, p-1$, 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: therefore despite tends to do well in practice it is not guaranteed 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 45 (Pros/cons)*.* we have:

- pros: investigates the same number of models as the forward, that is $1 + p(p+1)/2$
- cons: similarly to forward does not 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 46 (Pros/cons)*.* This approach mimics the best subset selection while retaining the computational advantages of forward and backward stepwise selection

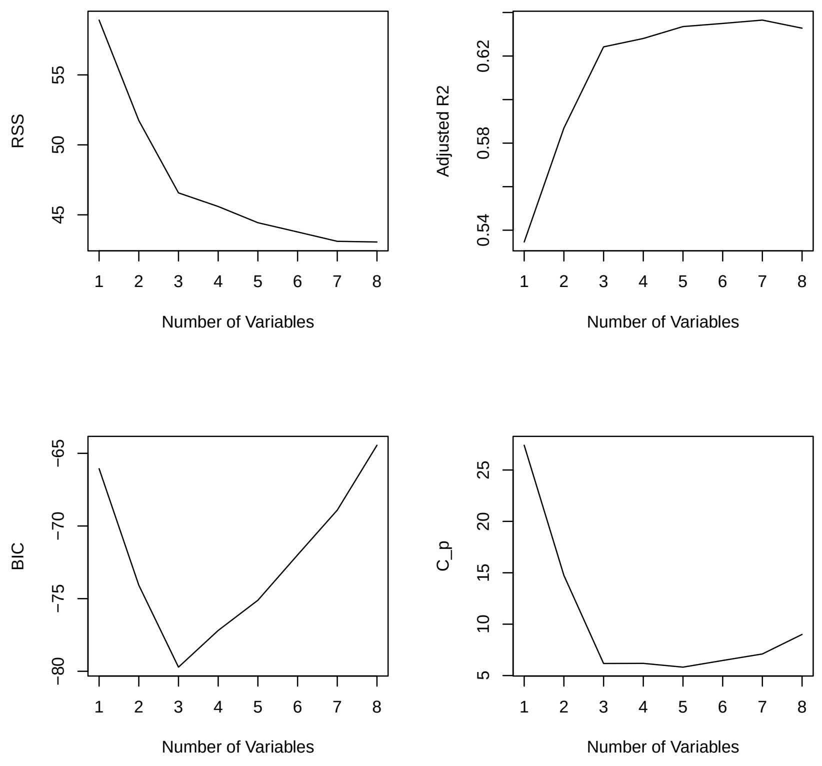
3.2.3 Exercise

Example 3.2.1. Using the prostate dataset:

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

```
## Model Selection #### 2024-03-06
library(lbdatasets)
library(leaps)
summary(prostate)

##      lcavol           lweight           age             lbph              svi
##  Min.   :-1.3471   Min.   :2.375   Min.   :41.00   Min.   :-1.3863   Min.   :
##  1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00   1st Qu.:-1.3863   1st Qu.:
##  Median : 1.4469   Median :3.623   Median :65.00   Median : 0.3001   Median :
##  Mean   : 1.3500   Mean   :3.629   Mean   :63.87   Mean   : 0.1004   Mean   :
##  3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00   3rd Qu.: 1.5581   3rd Qu.:
##  Max.   : 3.8210   Max.   :4.780   Max.   :79.00   Max.   : 2.3263   Max.   :
```

```
##       lcp              gleason          pgg45              lpsa           train
##  Min.   :-1.3863   Min.   :6.000   Min.   :  0.00   Min.   :-0.4308   Mode :logical
##  1st Qu.:-1.3863   1st Qu.:6.000   1st Qu.:  0.00   1st Qu.: 1.7317   FALSE:30
##  Median :-0.7985   Median :7.000   Median : 15.00   Median : 2.5915   TRUE :67
##  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

## rm the last column which is not used
x <- prostate[,-ncol(prostate)]
## number of predictors
p <- ncol(x) - 1

### 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=x) # method="exhaustive" is by default best subset
(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="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

## if we look at r^2 we choose the model with 7 variables, at bic
## with 3

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

```
##  (Intercept)       lcavol      lweight          age         lbph          svi
##  0.494154754  0.569546032  0.614419817 -0.020913467  0.097352535  0.752397342
##        pgg45
##  0.005324465

## Stepwise forward
## ----------------
out.fs <- regsubsets(lpsa ~ ., data = x, 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 = x, 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 = x, 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.

```r
## ----------------------------
## 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 coeffcents of the model with id parameters(id is t
    ## 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
set.seed(1234)
folds <- sample(1:K, nrow(x), replace = T)
## 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){
    ## divide in train and validation set
    ## apply regsubset to data matrix X excludign
    best.fit <- regsubsets(lpsa ~ ., data = x[folds!=i,])
    ## now calculate the error for each number of predictrs
    for (j in 1:p){
        pred <- predict(best.fit, newdata=x[folds==i,], id=j)
        cv.errors[i,j] <- mean((x$lpsa[folds==i]-pred)^2)
    }
}

## Error in parse_only(code):  could not find function "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 predictors

## 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):  no non-missing arguments to min; returning
Inf

## Warning in max(x):  no non-missing arguments to max; returning
-Inf

## Error in plot.window(...):  need finite 'ylim' values
## 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=x)
coef(reg.best,3)

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

3.3 Shrinkage/penalization

Remark 25. In this approach we fit a model containing all p predictors but shrinking the estimates toward 0.
It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.
The methods are ridge and lasso

3.3.1 Ridge

Definition 3.3.1 (OLS fitting)**.** In least square fitting the estimates for $\beta_0, \dots, \beta_p$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 second term $\lambda \sum_{j=1}^{p} \beta_j^2$ is the *shrinkage penalty*: it is small when $\beta_1, \dots, \beta_p$ are close to zero, and so it has the effect of shrinking the estimates of β_j towards zero.

Important remark 47 (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 **??** the coefficients of the prostate dataset

Remark 26 (Choosing λ)*.* Ridge regression will produce a different set of coefficient estimates, $\hat{\beta}_\lambda^R$ for each value of λ: selecting a good value for λ is critical. We tipically do the estimates for several λ and then choose the best λ by crossvalidation

Important remark 48 (Scaling)*.* 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.
So in order to put all the different variable/scales on the same level, it is best to apply ridge regression after standardizing the predictors
All of the standardized predictors will have a standard deviation of one. As a result the final fit will not depend on the scale on which the predictors are measured.

Important remark 49 (Performance improvement vs OLS)*.* Ridge regression's advantage over least squares is due to bias-variance trade-off:

- in general when the relationship between the response and the predictors is close to linear, the least squares estimates will have low bias but may have high variance (a small change in the training data can cause a large change in the least squares coefficient estimates).
- when p is almost as large as n, the least squares estimates will be extremely variable: here is best to shrink!
- If $p > n$, then the least squares estimates do not even have a unique solution, whereas ridge regression can still perform well by trading off a small increase in bias for a large decrease in variance. Here there's no alternatives to shrinking

When applied, as λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias

Important remark 50 (Pros/cons). Regarding the pros, ridge regression:

- works best when the least squares estimates have high variance;
- has substantial computational advantages over best subset selection (compute one model for each λ instead of 2^p models)

For the cons:

- Ridge will include all p predictors in the final model (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 $\lambda = \infty$) and this could be a problem in *interpretation* in settings in which the number of variables p is quite large.

For this last reason we have lasso

3.3.2 LASSO

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

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

Important remark 51. Has a similar formulation to ridge (again $\lambda \geq 0$ is a tuning parameter to be choosen in CV); the only difference is in the penalization term where the β_j^2 term (ℓ_2 penalty) is substituted by $|\beta_j|$ (ℓ_1 penalty).
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 $\sum_{i=1}^{n} |x_i|$.
This little change make force some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large so lasso performs variable selection and is preferred.
We say that the lasso yields/returns *sparse models* (models that involve only a subset of the variables, some other are 0).

3.3.3 Another formulation for ridge and lasso: constraint minimization

We can think ridge and lasso as a minimization subject to constraint s. That is

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

subject to, respectively

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

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

s can be seen as a kind of budget we can pay.
In case of 2 variable we can visualize in the plane

- the ridge regression estimates 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 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 the graph 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. Regarding our constraint s, if

- is large (non restrictive 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
- setting $\lambda = 0$ is like having an high budget

Important remark 52 (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)

3.3.4 Comparing ridge and lasso

lasso has a major advantage over ridge regression, in that it produces simpler and more interpretable models that involve only a subset of the predictors. However, which method leads to better prediction accuracy?

- Neither ridge regression nor the lasso will universally dominate the other: cross-validation can be used to determine which approach is better on a particular dataset.
- The lasso generally performs better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero.
- Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.

3.3.5 Selecting the tuning parameter

Implementing ridge regression and the lasso requires a method for selecting a value for the tuning parameter λ, or equivalently, the value of the constraint s. We do it by cross-validation:

1. we choose a grid of λ values, 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.

3.3.6 Exercise

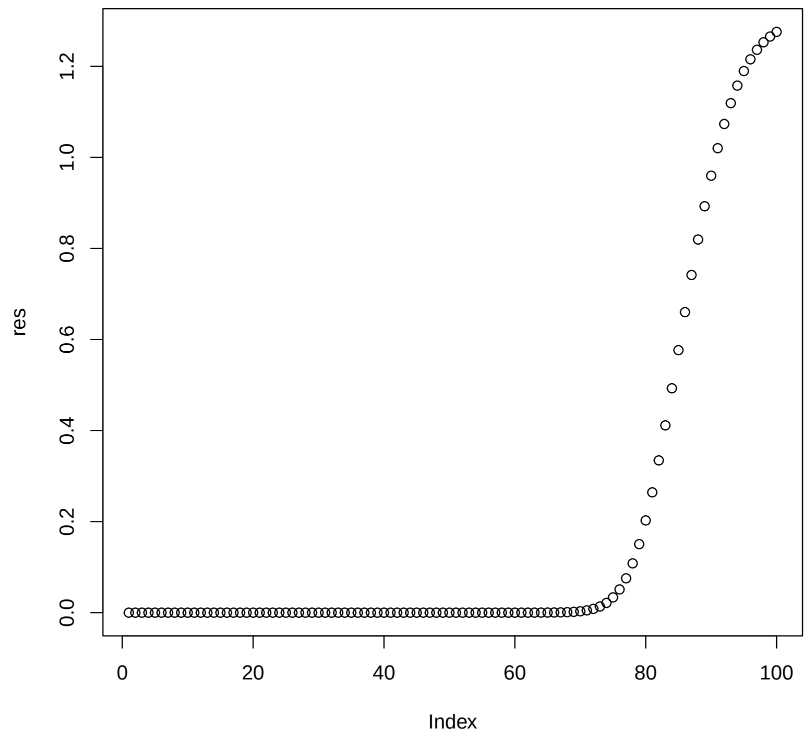
We will use the `glmnet` package in order to perform ridge regression and the lasso. The main function in this package is `glmnet`, which can be used to fit ridge regression models, lasso models, and more. We must pass in an `x` matrix as well as a `y` vector (we do not use the common formula syntax).

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

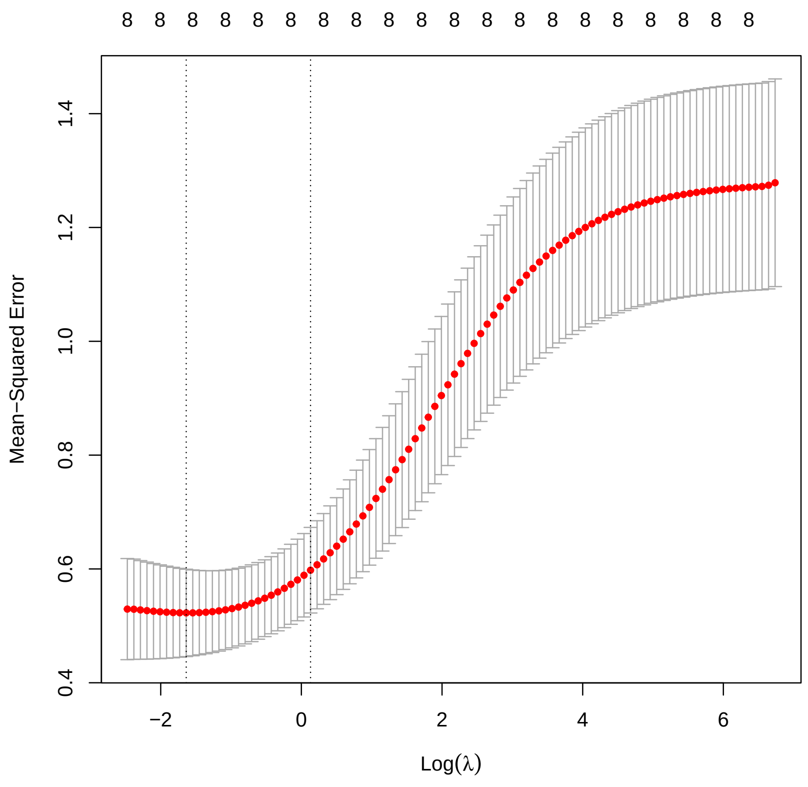
```
library(glmnet)

## Loading required package:  Matrix
## Loaded glmnet 4.1-8

library(lbdatasets)

# create the model matrix
x <- prostate[,-ncol(prostate)] # rm the last var of the data
x <- model.matrix(lpsa ~ ., data = x)[,-1]
```

```
y <- prostate$lpsa
p <- ncol(x)-1

## 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

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

##  (Intercept)       lcavol      lweight          age         lbph          svi
## 2.478387e+00 8.260413e-11 1.340776e-10 3.019569e-12 1.642903e-11 1.813000e-10 5.202940e
##      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 4.523718e
##      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)
```

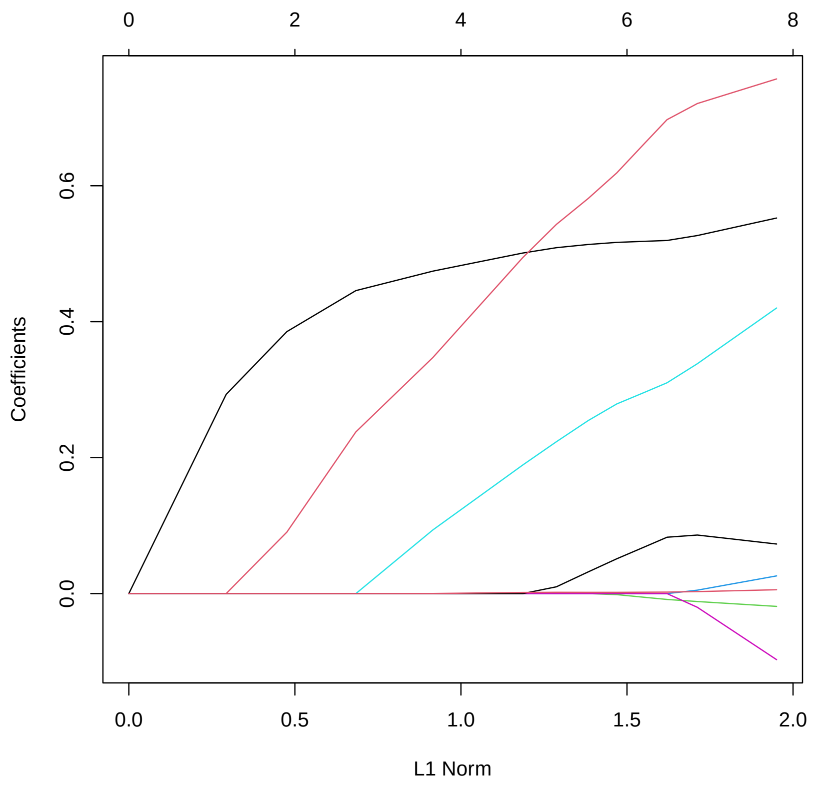


```
## at the top we have only 8, meaning that we're not dropping
## covariates, all retained in the model (being ridge)
## when small we are ols like, for increasing level we introduce
## bias in order to reduce the variance. this to get the smallest
## error. However best results here are in the first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

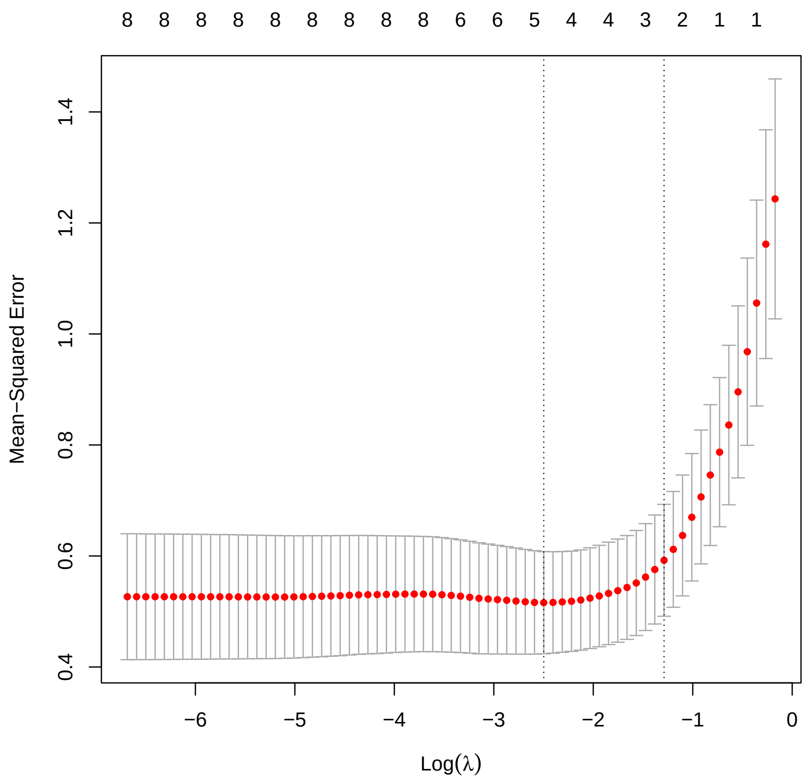
```
# none of the coefficients are zero: ridge regression does not
# perform variable selection

## 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):  collapsing to unique 'x' values
```



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

3.4 Dimension reduction methods

We now explore a class of approaches that transform (linear combination) the predictors and then fit a least squares model using the transformed variables. These provide a simple way to perform regression using $M < p$ predictors, but they are not a feature selection method.
We will refer to these techniques as dimension reduction methods: in particular we see the Principal Components Regression approach
The procedure is as follow:

- we find $M < p$ linear combinations of our original p predictors, as linear combination of the original variables

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

  for some constant $\phi_{1m}, \ldots, \phi_{pm}$, with $m = 1, \ldots, M$.
  The ϕ_{jm} are choosed and Z_m calculated using principal component analysis constructing the first M principal components, $Z_1, \ldots, Z_M$

- we then fit the linear regression model using the new variables/principal component and using standard least squares regression

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

  Key idea is that often a small number of principal components suffice to explain most of the variability in the data, as well as the relationship with the response. In other words, we assume that the directions in which $X_1, \ldots, X_p$ show the most variation are also associated with Y. While this assumption is not guaranteed to be true, it often turns out to be a reasonable enough approximation to give good results.
  If assumption underlying PCR holds/the constant of the transformation Z_m, $\phi_{1m}, \ldots, \phi_{pm}$ are chosen wisely, then such dimension reduction approaches can often outperform least squares regression on the original $X_1, \ldots, X_p$, if the increased bias is mitigating by reduced overfitting

- regarding the number of components M, as more principal components are used in the regression model, the bias decreases, but the variance increases.
  When $M = p$ then PCR amounts simply to a least squares fit using all of

the original predictors.
The number of principal components M is typically chosen by cross-validation.

Important remark 53 (Constraint 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 $M << p$ can significantly reduce the variance of the fitted coefficients.

Important remark 54 (Variable standardization). Finally:

- When performing PCR, it is generally recommended to standardize, to have all variables are 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.
- if the variables are all measured in the same units (say, kilograms, or inches), then one might choose not to standardize them.

3.4.1 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.

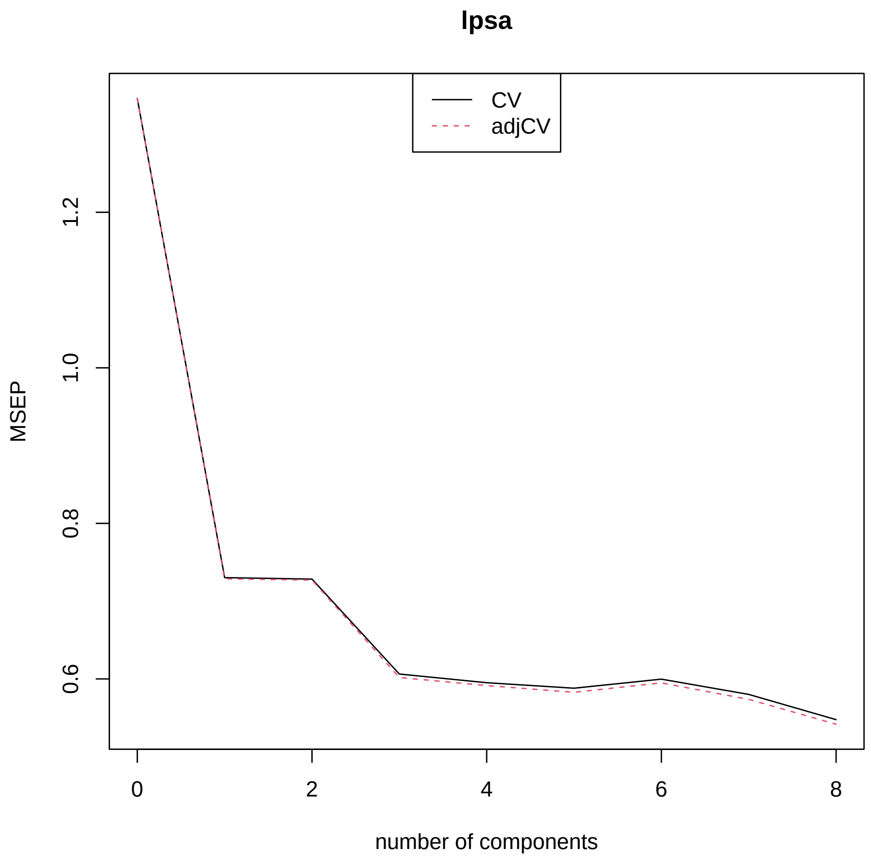
There is a nice package `pls` to do all this, but we will be doing the same thing even by standard methods, because the package works only with continue response so in case we have a classification problem we need to extract principal components and use them in any model/learner we want by hand (and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

3.4.1.1 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)

##
## Attaching package:  'pls'
## The following object is masked from 'package:stats':
##
##     loadings

library(lbdatasets)
set.seed(1234)

pcr.fit <- pcr(lpsa ~ .,
               data = prostate[,-ncol(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  8 comps
## CV            1.16   0.8545   0.8534   0.7786   0.7714   0.7669   0.7744   0.7616   0.7400
## adjCV         1.16   0.8536   0.8527   0.7758   0.7690   0.7634   0.7713   0.7574   0.7361
##
## 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:

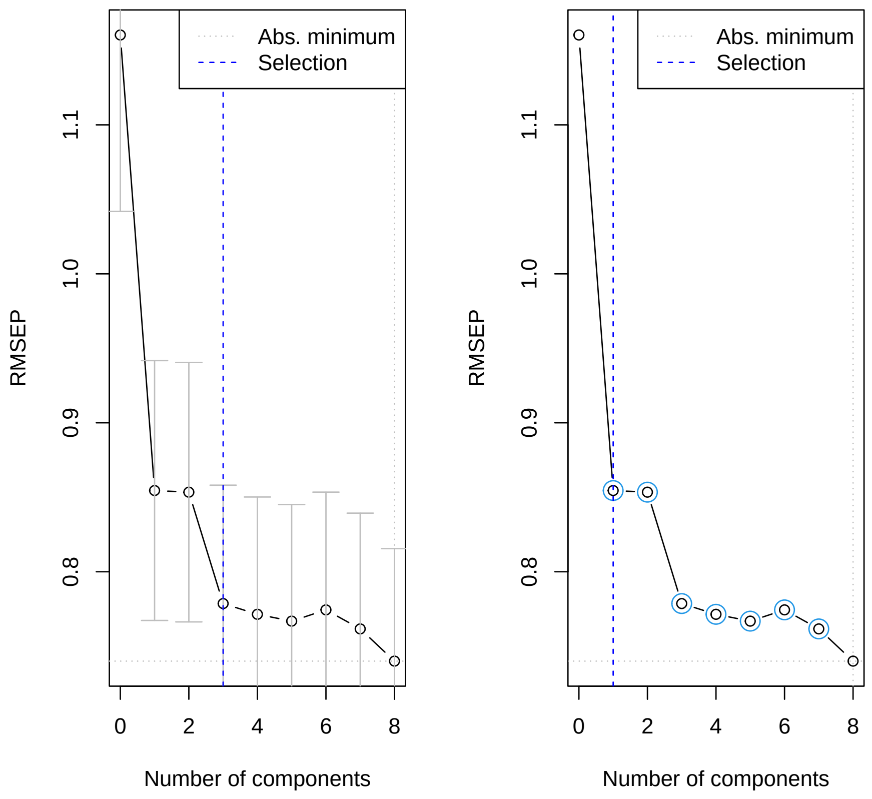
- Fit method: dice quale metodo (spectral decomposition o SVD decomposition, qui svd). Spectral decomposition can only be applied to square matrix while svn can be carried out for any matrix
- Validation: tells the metrics used for error, here root mean squared error (RMSE); in order to obtain the usual 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.

```r
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

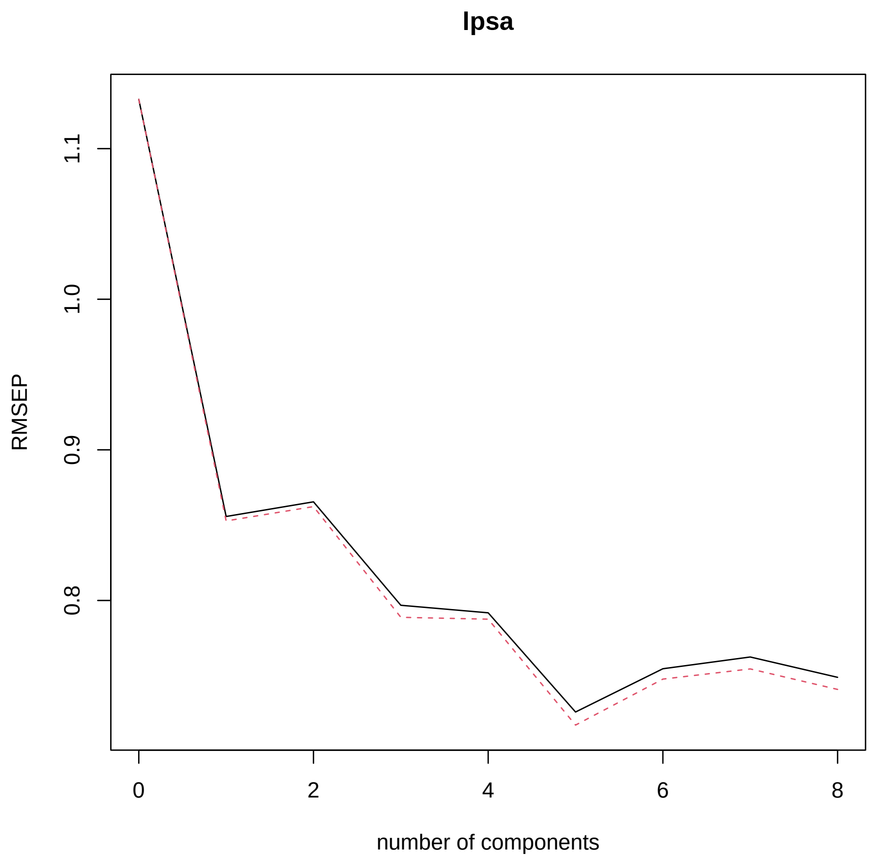
3.4.1.2 Test error estimate of the PCR: Training + Validation set

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

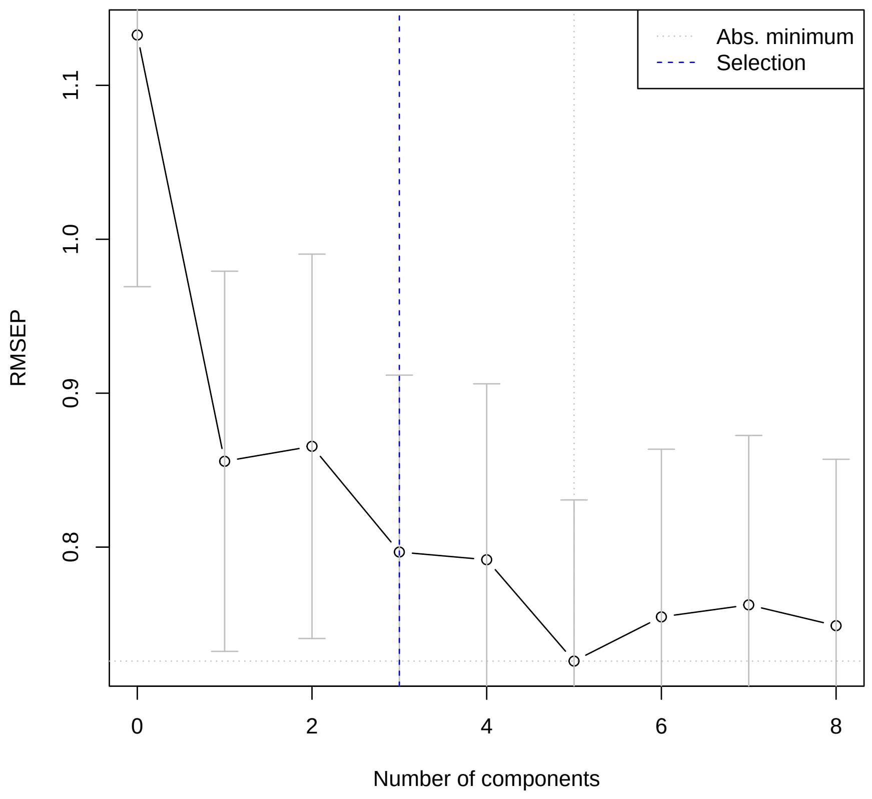
As seen for opther cases if we work with standardized data in the training set we must standardize in the test set using the parameter of the training set (here the function handle it I guess)

```
## 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[,-ncol(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.

3.4.1.3 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
```

3.4.1.4 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
```

Capitolo 4

Tree based methods

4.1 Basic trees

Trees

- stratifies the predictor space into a number of consequent/simple regions: then 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
- are simple and useful for interpretation
- can be applied to both regression (regression tree) and classification (classification tree) problems

4.1.1 Regression tree

It consists of a series of splitting rules, starting at the top of the tree. 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
- the predicted lpsa for these latter units is given by the mean response value for the units in the dataset with a gleason ≥ 6.5. Therefore the average log PSA is 2.896 .
- Patients with gleason$<$ 6.5 are further subdivided by age: if they are younger than 67 they have an average lpsa equal to 1.530, otherwise equal to 2.441, on average.
- the tree stratifies or segments the patients into *three regions* of predictor space (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 , and R_3 are known as *terminal nodes* or leaves of the tree.

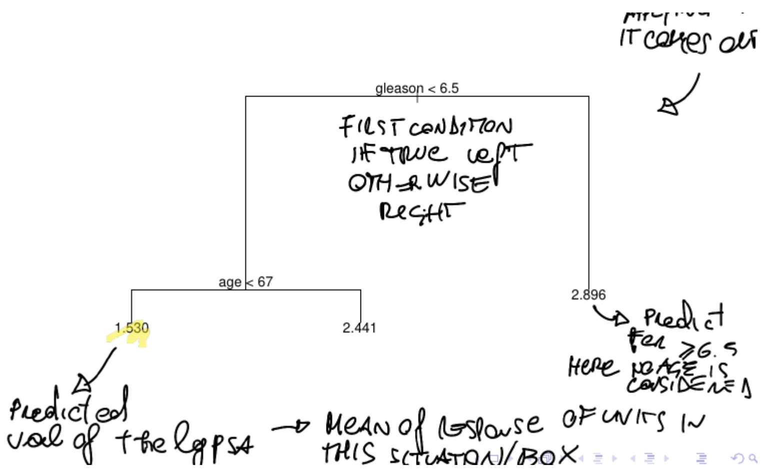

IT COMES OUT
gleason < 6.5
FIRST CONDITION IF TRUE LEFT OTHERWISE RIGHT
age < 67
1.530
2.441
2.896
Predict for ≥6.5 Here NO AGE IS CONSIDERED
Predicted val of the lpsa → MEAN of response OF UNITS IN THIS SITUATION/BOX

Figura 4.1: Tree.

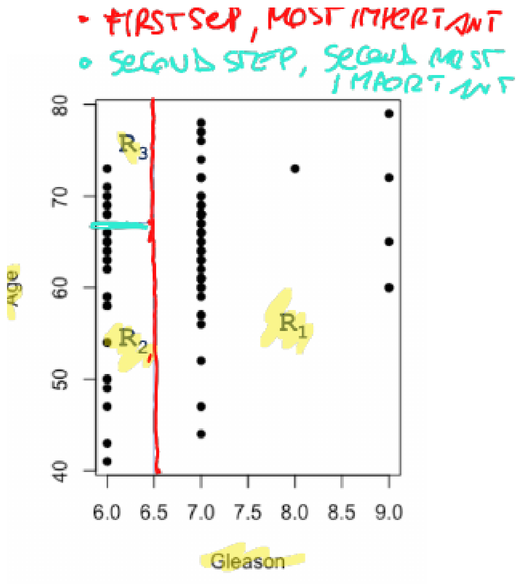


Figura 4.2: Tree.

- 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
- Pros of this analysis: it's an *oversimplification* of the true relationship between gleason, age, and lpsa. But it *easy to interpret/explain*

How do we build a regression tree?

1. Divide the predictor space - that is, the set of possible values for $X_1, X_2, ..., X_p$ - into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$
2. For every observation that falls into the region R_j , make the same prediction, which is simply the mean of the response values for the training observations in R_j .

We divide the predictor space into high-dimensional rectangles for simplicity: the goal is to find the boxes $R_1, R_2, ..., R_J$ that minimize

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

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.

Since is computationally infeasible to consider every possible partition of the feature space into J boxes we takea *recursive binary splitting* approach that 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 at that particular step (rather than looking ahead and picking a split that will lead to a better tree in some future step).

In order to perform the recursive splitting:

1. Select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|Xj < s\}$ and $\{X|Xj \geq s\}$ leads to the *greatest possible reduction in RSS*. In greater deta, 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 equation

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

   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 $(R_2(j, s)$
2. Repeat the process starting from the two previously identified region, looking for the best predictor and best cutpoint to minimize RSS within each region considered

3. The process continues until a stopping criterion is reached (eg we may continue until no region contains more than five observations)

Important remark 55 (Tree problems). One of the problems of trees (pros: no assumptions at all) is that they tend to overfit by looking at optimality everywhere if not properly handled: the resulting tree might be too complex.
A smaller tree with fewer splits (that is, fewer regions $R_1, ..., R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

Remark 27. To have simpler tree we can finetune the stopping criteria or grow a very large/free tree T_0 and then prune it back in order to obtain a subtree.

Our goal is to select a subtree that leads to the lowest test error rate

- : given a subtree, we can estimate its test error using cross-validation or the validation set approach. However, estimating the cross-validation error for every possible subtree would be too cumbersome
- rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each α there correspond a subtree $T \subset T_0$ such that

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

  where:

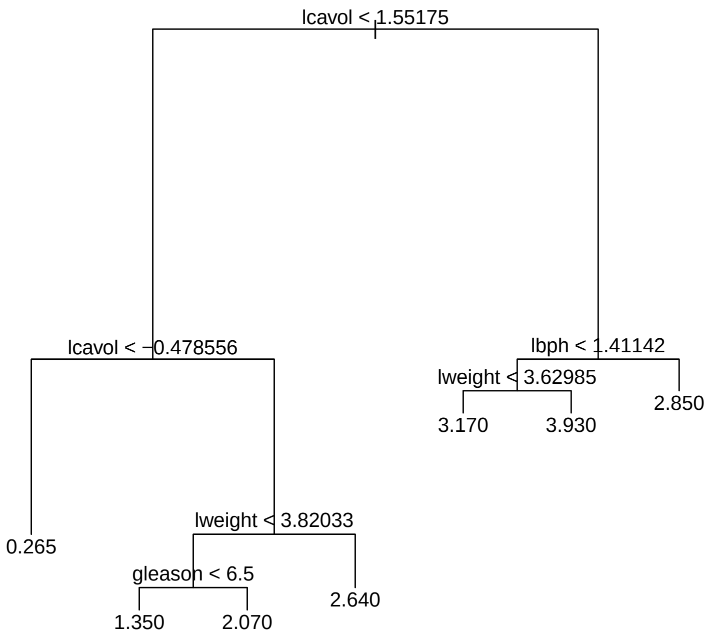
  - $|T|$ indicates the number of terminal nodes of the tree T ;
  - R_m is the rectangle corresponding to the m-th terminal node
  - $\hat{y}_{R_m}$ is the predicted response associated with R_m , i.e. the mean of the training observations in R_m .
  - the tuning parameter α (similar to ridge and lasso λ) handle penalization for complexity:
    * if $\alpha = 0$ we don't penalize and choose the more complex tree (the subtree T will simply equal T_0 , and the equation just measures the training error);
    * if α increases there is a price to pay for having a tree with many terminal nodes, and so the quantity will tend to be minimized for a smaller subtree. Therefore we will choose simpler trees(so the bias increase and the variance decrease)
    * again α is choosen in cross validation

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[, -ncol(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[, -ncol(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)
```

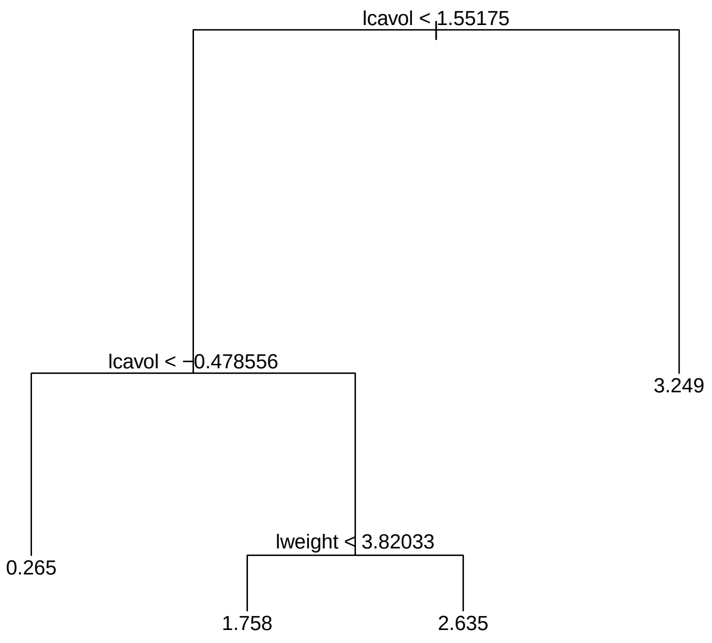


```
## prediction and error of the regression tree
yhat <- predict(tree_pros,
                newdata = prostate[-train,-ncol(prostate)])
mean((prostate$lpsa[-train]-yhat)^2)
```

```
## [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,-ncol(prostate)])
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; In interpreting we are also interested in the class proportions among the training observations
- construction is similar to regression tree: use recursive binary splitting but instead of RSS we try to minimize:

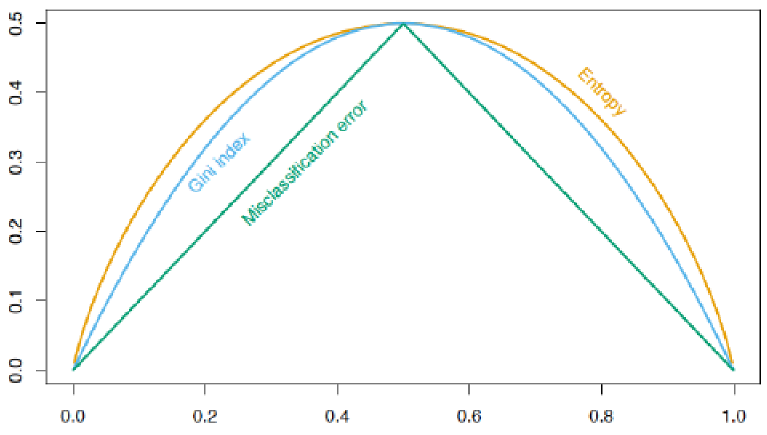


Figura 4.3: Tree.

- *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 mth region that are from the kth class.
This index however is not optimal/sufficiently sensitive for tree growing, 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 of total variance across the K classes if all the $\hat{p}_{mk}$ are close to zero or one the index take a small value (which is want we search for).
For this reason the Gini index is referred to as a measure of node *purity*: a small value indicates that a node contains predominantly observations from a single class.

- *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 $\hat{p}_{mk}$'s are all near zero or near one

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

Important remark 56. Entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate.

Example 4.1.1. Eg in a two-class problem with 400 observations in each class (denote this by (400, 400)), suppose:

- one split created nodes (300, 100) and (100, 300)
- the other created nodes (200, 400) and (200, 0).

Both splits produce a misclassification rate of 0.25, but the second split produces a pure node and is probably preferable.

Important remark 57. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

NB: quindi se ho capito bene si cresce con gini/entropia e si taglia con errore

4.1.2.1 Exercise

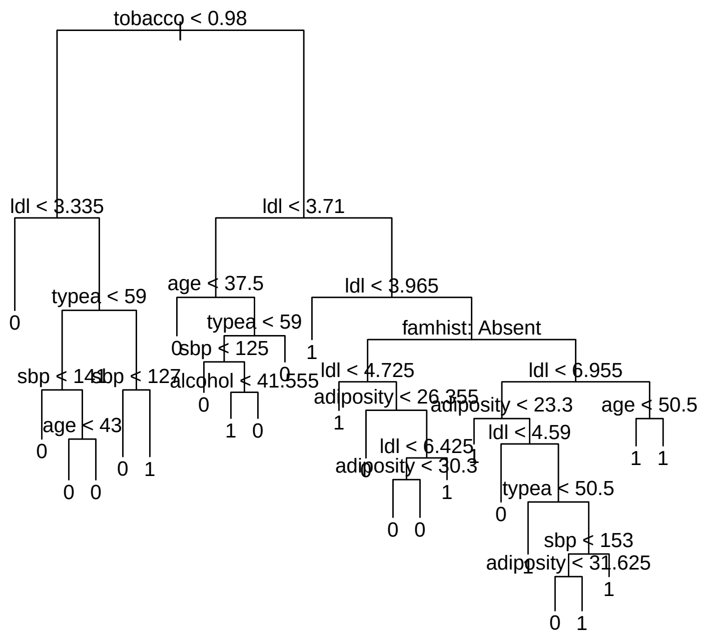
- Fit a classification tree in order to predict chd using all variables and estimate the test error via validation set approach.

```
## 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"   "famhist"
## 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):  object 'heart' not found

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

## Error in eval(expr, envir, enclos):  object 'cv_heart' not
found

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

## Error in eval(expr, p):  object 'best_size' not found

plot(pruned_heart)

## Error in eval(expr, envir, enclos):  object 'pruned_heart'
not found

text(pruned_heart, pretty = 0)

## Error in eval(expr, envir, enclos):  object 'pruned_heart'
not found

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

```
## Error in eval(expr, envir, enclos):  object 'pruned_heart'
not found

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

## Error in eval(expr, envir, enclos):  object 'yhat_pruned' not
found

mean(yhat_pruned != heart_test$chd)

## Error in h(simpleError(msg, call)):  error in evaluating the
argument 'x' in selecting a method for function 'mean':  object
'yhat_pruned' not found

## 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

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

whereas 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? It 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

Important remark 58 (pros/cons of trees). Pros:

- Trees are very easy to explain to people;

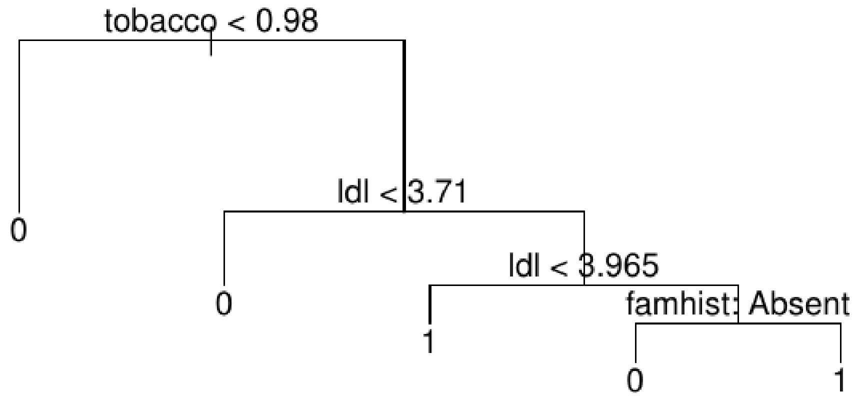


Figura 4.4: Tree.

- can be displayed graphically and are easily interpreted even by a non expert (esppecially if they are small)
- can easily handle qualitative predictors without the need to create dummy variables

Cons:

- trees do not have the same level of predictive accuracy as some of the other regression and classification approaches
- can be very non-robust: a small change in the data can cause a large change in the final estimated tree

However by aggregating many decision trees the predictive performance of trees can be substantially improved. Bagging and random forest are method to fix tree problems

4.2 Bagging

It's the usage of bootstrap to pimp decision trees prediction (its main focus rather than interpretation where a single tree is simpler) but can be applied to other predictive methods as well.
Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning methods.
Recall that given a set of n independent observations $Z_1, ..., Z_n$ each with variance σ^2 , the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n. Averaging a set of observations reduces variance
The idea of bagging is, instead of having a single tree, i build several trees. We don't have many training set so i use bootstraps,

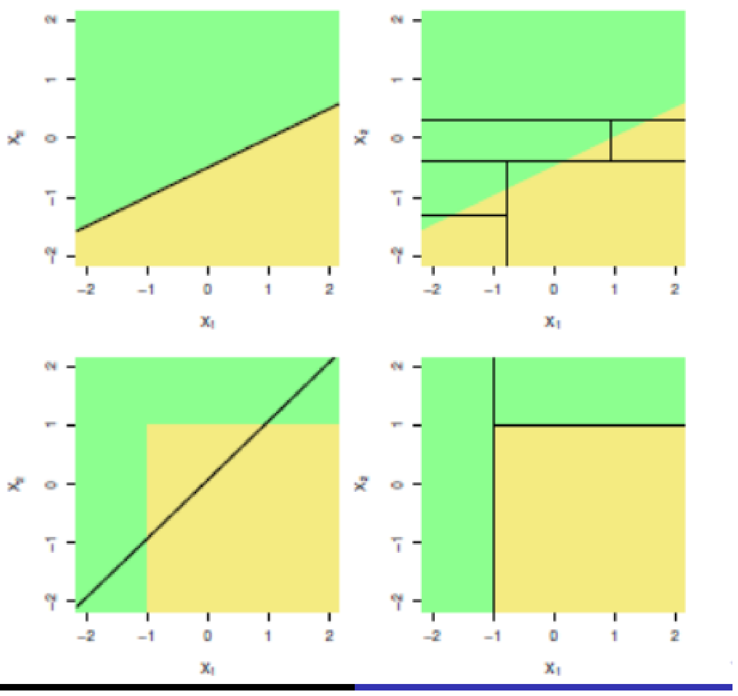


Figura 4.5: Tree.

- 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).

- we then train our method on the b-th bootstrapped training set in order to get the estimate the tree $\hat{f}^{*b}(x)$ and finally average all the predictions, to obtain

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

  In case Y is qualitative we can record the class predicted by each of the B trees and for prediction take a majority vote

- the rees are grown deep, and are not pruned: hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance

- Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

In figure 4.6 the number of trees tend to stabilize the test error (this to choose the number of trees and bootstrap sample)

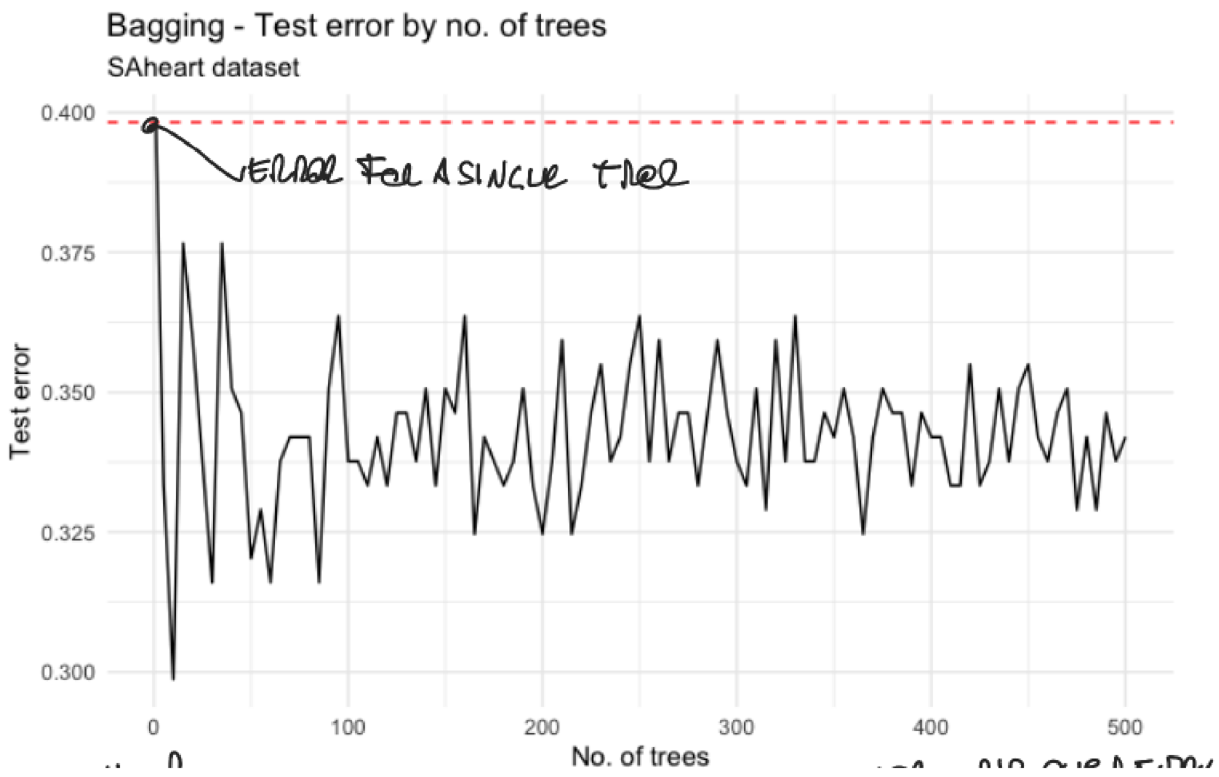


Figura 4.6: Tree.

4.2.1 OOB error estimation

Out-of-Bag (OOB) Error Estimation:

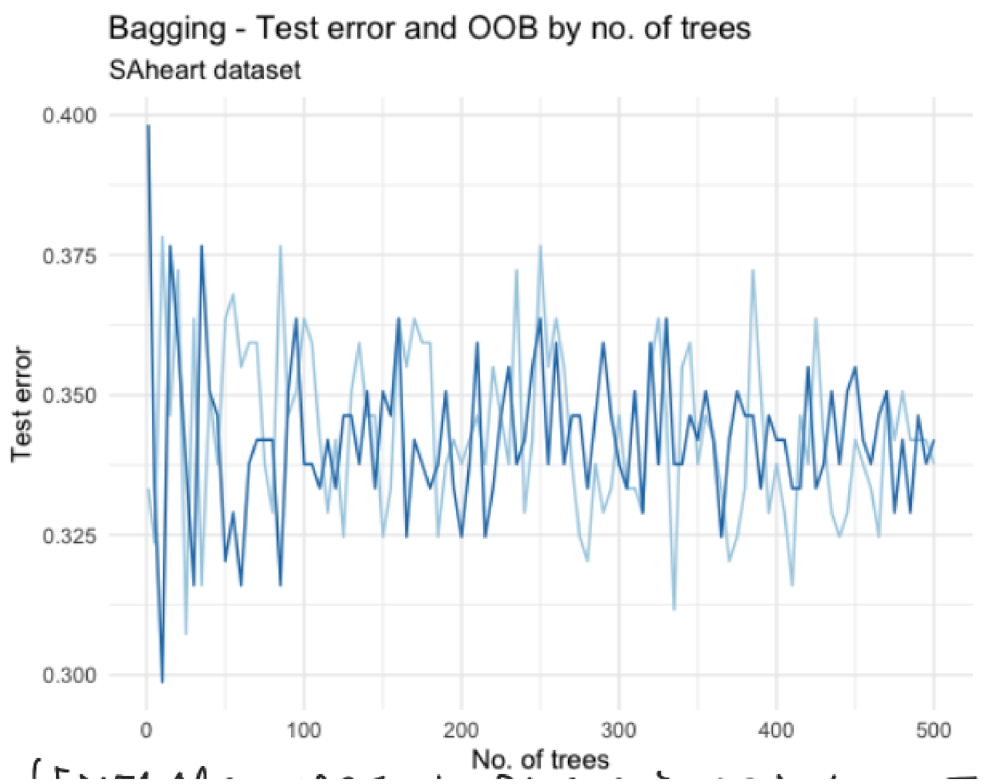
- with bagging there's no need of cv to estimate to estimate the test error of a bagged model.
- In the bootstrap, on average, each created dataset/tree makes use of around 2/3 of the observations.
- The remaining 1/3 of the observations not used to fit a given bagged tree are called out-of-bag (OOB) observations (for a single bootrap sample). These units can be used to estimate error
- the response for the i-th observation can be predicted using the trees in which that observation was OOB (not included in the bootstrap sample).
- this will yield around $B/3$ predictions for the i-th observation: in order to obtain a single prediction for the unit we can 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.
- It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.
- OOB error is a valid estimate of the test error for the bagged model (fig 4.7 the two curves overlap a lot)
- if data are scarse do use bagging to avoid a test sample, but if a comparison has to be made with other methods a separate set is needed

4.2.2 Variable importance measures

Other than OOB error this is another free gift: we have a ranking of variable importance.
With bagging focus is clearly on prediction; although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using the RSS (regression) or the Gini index (classification):

- regression: in the case of bagging, we can record the total amount that the *RSS is decreased* due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- classification: we can add up the total amount that the *Gini index is decreased* by splits over a given predictor, averaged over all B trees

In figure 4.8 and example using SAheart data



MEAN of THE BOOT error ON NOT INCLUDED

THE TWO CURVES OVERLAP A LOT

is needed

[illegible]

Figura 4.7: Tree.

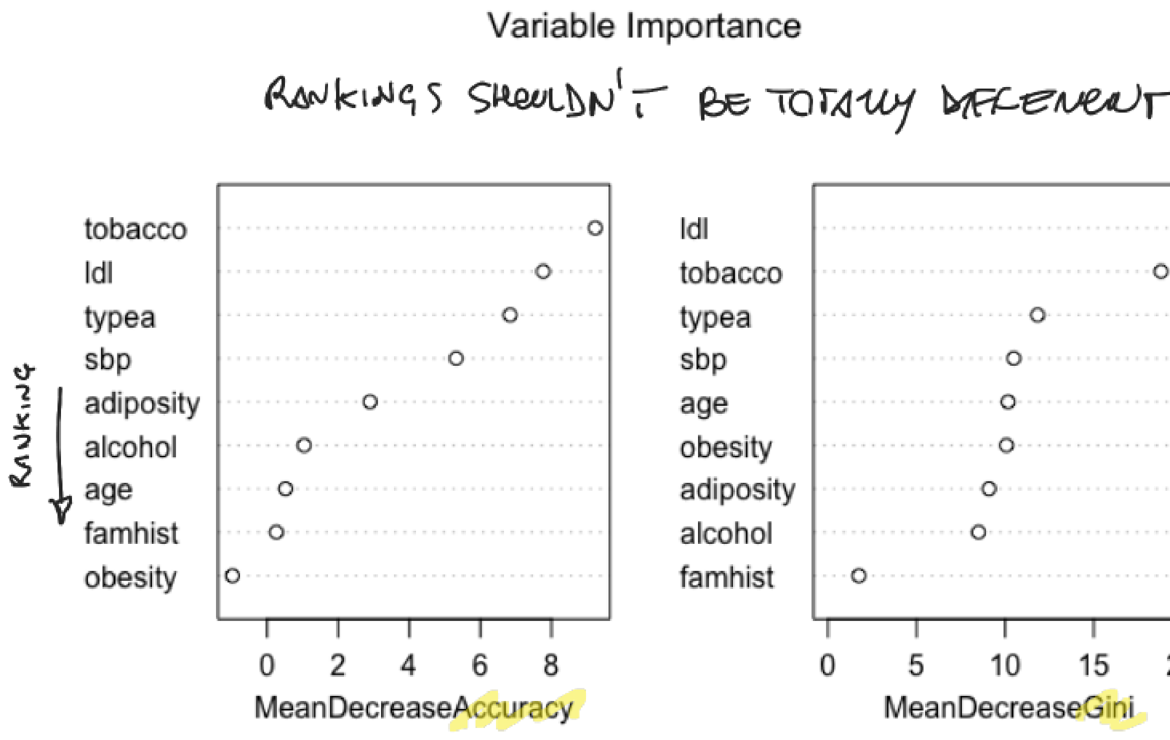


Figura 4.8: Tree.

4.3 Random forest

This method improve over bagging trees by dcorrelating the trees a bit:

- suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated.
- Random forests overcome this problem by forcing each split to consider only a subset of the predictor
- Compared to bagging less correlated and variance is lower/less dependence from the original sample

The procedure of random forest decorrelates the tree as follow:

- we make the bootstrap sample
- in building the tree, each time a split is considered a random sample of m predictors is chosen as split candidates from the full set of p predictors: tipically $m \approxeq \sqrt{p}$ for classification and $m \approxeq p/3$ for regression (so not even allowed to consider a majority of aailable predictors).
  The split is allowed to use only one of those m predictors and so there are no variables dominating no more 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.

Important remark 59. Some final remarks:

- A random forest built using $m = p$ amounts simply to bagging.
- A small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors.
- The R function is the same as bagging with random forest by default
- a comparison of bagging and random forest in figure 4.9: il turchese è un po' piu in basse mediamente del verde
- in variable importance ranking (figure 4.10) bagging is better while for prediction random forest is better

4.4 Exercise bagging random forest

4.4.1 Regression tree

1. Perfom bagging on the dataset and estimate the test error. Which are the most important predictors?

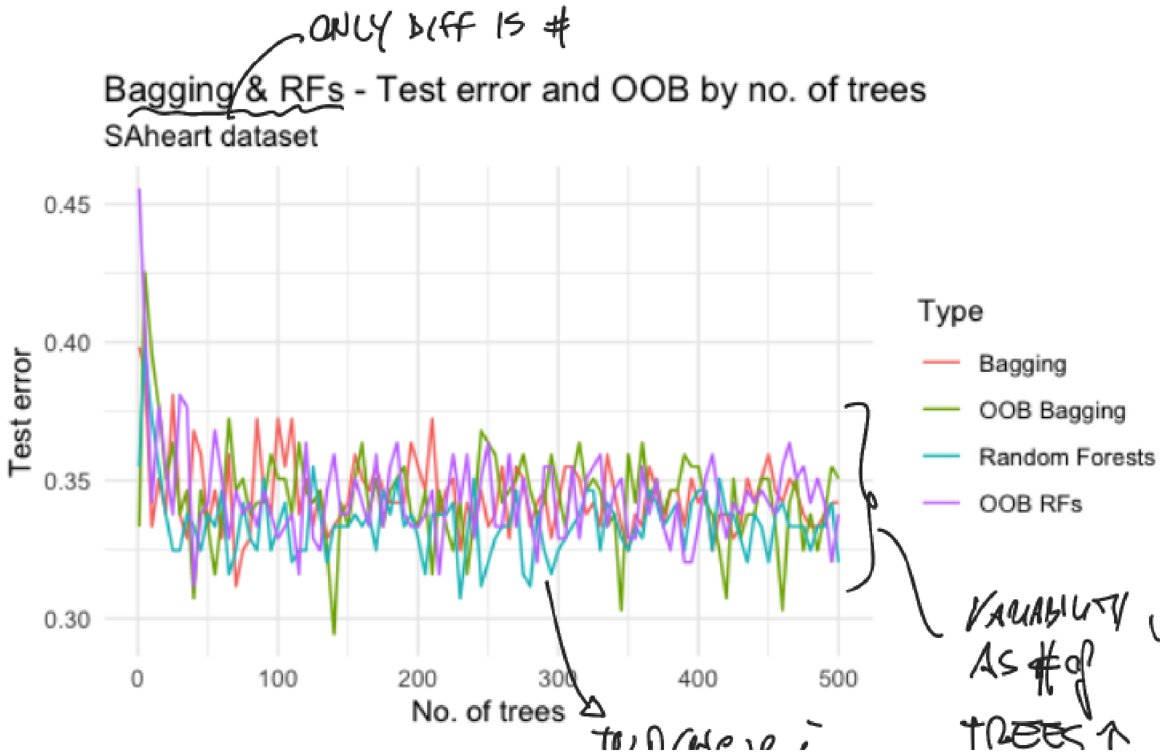
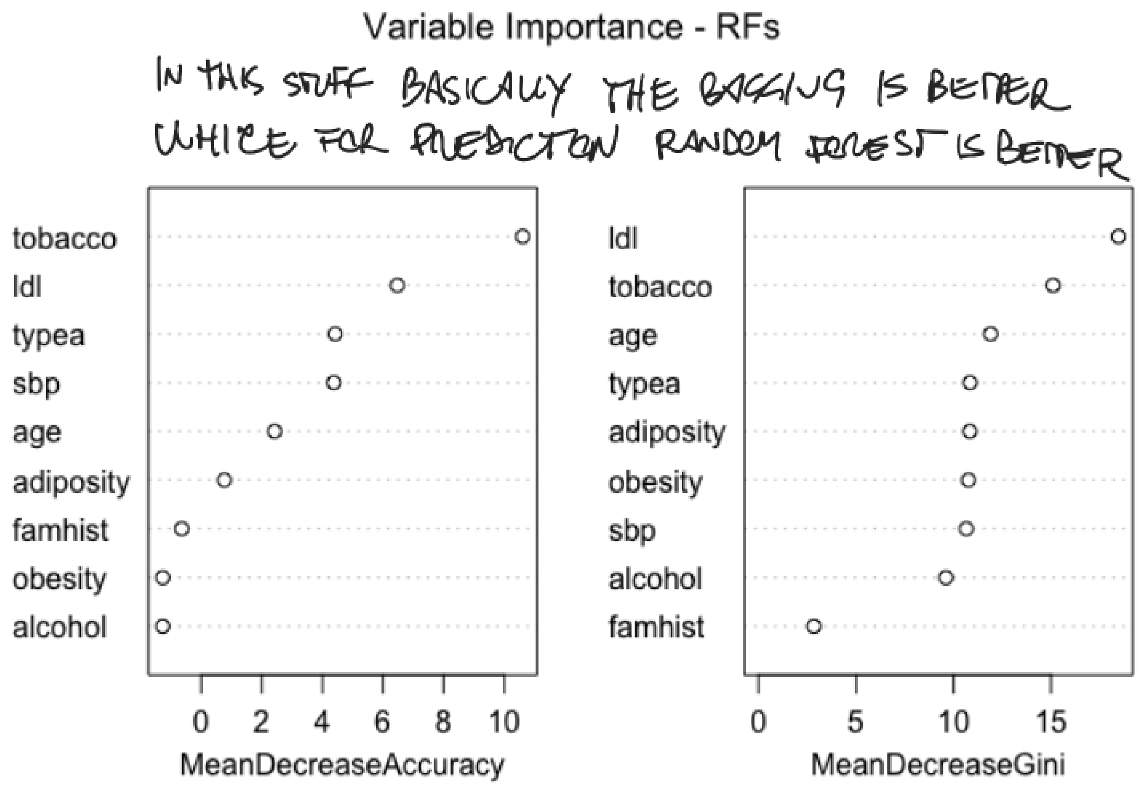


Figura 4.9: Tree.

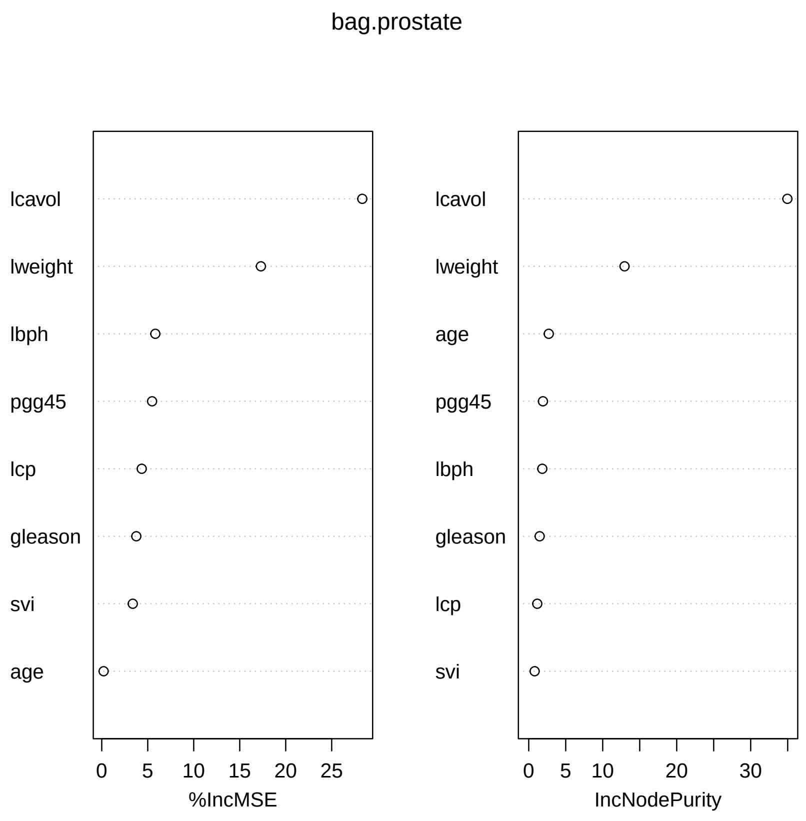


IN THIS STUFF BASICALLY THE BAGGING IS BETTER
WHILE FOR PREDICTION RANDOM FOREST IS BETTER

Figura 4.10: Tree.

```
## Regression - Bagging
# install.packages('randomForest')
library(lbdatasets)
library(randomForest)

## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.

# ?randomForest

x <- prostate[,-ncol(prostate)]
p <- ncol(x) - 1 # no. of predictors
n <- nrow(x) # no. of units

# split in train and validation to estimate the error
set.seed(1234)
train <- sample(1:n,ceiling(n/2))
x_test <- x[-train,]

## bagging
set.seed(1234)
bag.prostate <- randomForest(
  lpsa ~ .,
  data = x,
  subset = 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, 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"             "
##  [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
```

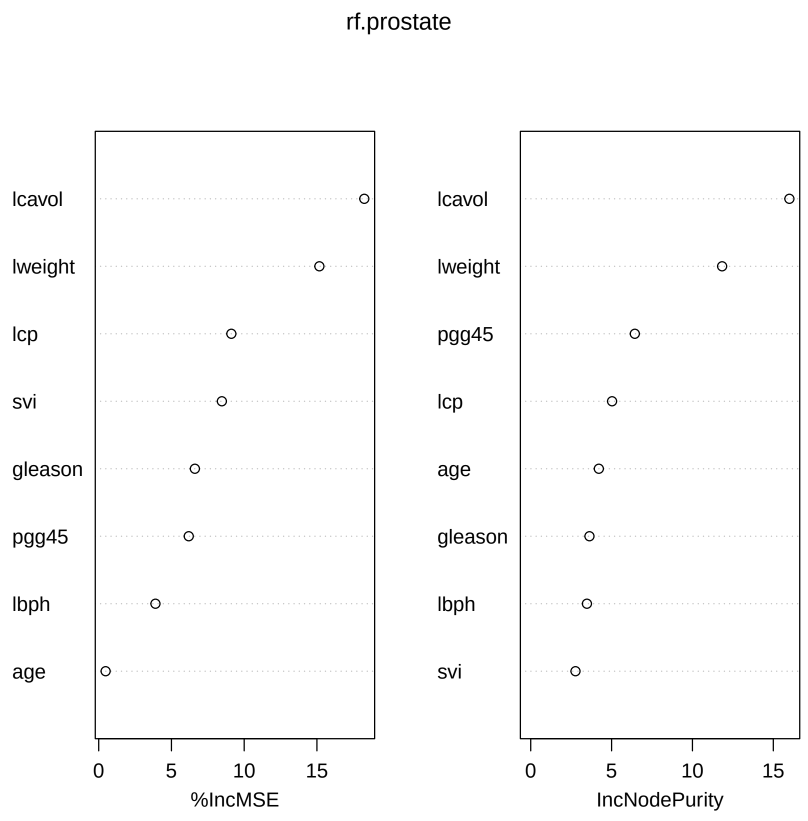


```
## Accuracy in the validation set
## we obtain the predicted values of yhat (which are calculated
## averaging the predicted values across the collection of trees)
yhat.bag <- predict(bag.prostate, newdata=x_test)
head(yhat.bag)

##         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,
                          subset = train,
                          importance = TRUE)
rf.prostate

##
## Call:
##  randomForest(formula = lpsa ~ ., data = x, importance = TRUE,      subset = t
##                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. Perfom bagging on the dataset and estimate the test error. Which are the most important predictors?

```
### Classification - Bagging ####
x <- SAheart[,-ncol(SAheart)]
y <- SAheart[,ncol(SAheart)]
```

```
## remember to set it as factor
heart <- data.frame(chd=as.factor(y),x)

n <- nrow(x)
p <- ncol(x)

set.seed(1234)
train <-sample(1:n, ceiling(n/2), replace=FALSE)
heart_test <-heart[-train,]

set.seed(1234)
## bagging with mtry = p
(bag.heart <- randomForest(chd ~ .,
                        data=heart,
                        subset = train,
                        mtry = p,
                        importance = TRUE))

##
## Call:
##  randomForest(formula = chd ~ ., data = heart, 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
pippo2 <- randomForest(chd ~ .,
                      data = SAheart,
                      subset = train,
                      mtry = p,
                      importance = TRUE)

## Warning in randomForest.default(m, y, ...):  The response has
five or fewer unique values.   Are you sure you want to do regression?

## 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,
                         subset = train,
                         importance=T)
rf.heart

##
## Call:
##  randomForest(formula = chd ~ ., data = heart, importance = T,      subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 33.33%
## Confusion matrix:
##     0  1 class.error
## 0 129 27   0.1730769
## 1  50 25   0.6666667

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 28. It's one of the most important recent developments in classification methodology and another approach for improving the predictions resulting from a decision tree (can be used both for regression as well but here our focus is classification).

Important remark 60 (Boosting idea). In bagging We have that each tree is built on a bootstrap data set, independent of the other trees. Boosting works differently:

- does not involve bootstrap sampling: each tree is fit on a modified version of the original data set;
- the trees are grown sequentially: each tree is grown using information from previously grown trees.
- it sequentially applyies a classification algorithm to reweighted versions of the training data and then take a weighted majority vote of the sequence of classifiers thus produced;
- starts with same weights and then increas the weight of units wrongly classified
- for many classification algorithms, this simple strategy results in dramatic improvements in performance.

Remark 29. Most commonly used version of the AdaBoost procedure (Freund and Schapire, 1996), also called Discrete AdaBoost.

Definition 4.5.1 (Discrete adaboost)**.** In the two-class classification setting:

- training data $(x_1, y_1), \dots, (x_N, y_N)$ with x_i a vector valued feature, $y_i = -1$ or 1
- classify units on the base of the sign of $F(x) = \sum_{m=1}^{M} c_m f_m(x)$ where each $f_m(x)$ is a classifier producing values plus or minus 1 and c_m are constants
- the procedure trains the classifiers $f_m(x)$ on weighted versions of the training sample, giving higher weight to cases that are currently misclassified; then the final classifier is a linear combination of the classifiers from each stage.

Important remark 61 (Discrete adaboost boosting). The algorithm:

1. start with common weights $w_i = 1/N$, $i = 1, \dots, N$ (the first step simply trains the classifier on the data in the usual mannerg)
2. repeat for $m = 1, \dots, M$:
    - fit the classifier (eg a tree) $f_m(x) \in \{-1, 1\}$ using weights w_i on the training data

- compute

$$err_m = \mathbb{E}_w\left[I(y \neq f_m(x))\right] = \frac{\sum_{i=1}^N w_i I(y_i \neq f_m(x_i))}{\sum_{i=1}^N w_i}$$
$$c_m = \log\left(\frac{1 - err_m}{err_m}\right)$$

  where

  - $I(y_i \neq f_m(x_i))$ means unit i was wrongly classified
  - c_m can be seen as trustiness of a single classifier (having within log the accuracy at numerator and misclassification rate at denominator so if ratio > 1 then $c_m > 0$)

- set $w_i \leftarrow w_i \exp\left[c_m I(y_i \neq f_m(x_i))\right]$, $i = 1, \ldots, N$: if $c_m I(y_i \neq f_m(x_i))$ is > 0 the weight increase and delta depends on the trustiness of the classifier.
  Then renormalize so that $\sum_i w_i = 1$

3. output the classifier sign $\left[\sum_{m=1}^M c_m f_m(x)\right]$

So after starting with common weigths (and having a first $m = 1$ iteration with those ones), for each successive iteration $m = 2, 3, \ldots, M$ the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. Specifically, at step m, those observations that were misclassified by the classifier $f_{m-1}(x)$ induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly. Thus as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence.

Thus each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence.

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)
heart_test <- SAheart[-train,]

## ?gbm::gbm
```

```
boost.out <- gbm(chd ~ .,   # we dont need to coerce to factor
                 data = SAheart[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 = SAheart[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
ntr <- length(train) # size of the training
heart_tr <- SAheart[train,]
set.seed(1234)
folds <- sample(1:5, ntr, 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
err.cv<-matrix(NA, 5, length(B),
               dimnames = list(NULL, paste0("B=",B[1:length(B)])))
for (i in 1:5){
    x.te <- heart_tr[folds==i,]
    x.tr <- heart_tr[folds!=i,]
    for (j in 1:length(B)){
        ## estimation
        boost.out <- gbm(chd ~ .,
                         x.tr,
                         distribution = "bernoulli",
                         bag.fraction = 1,
                         # interaction.depth = 1,
                         n.trees = B[j])
        ## prediction
        phat <- predict(boost.out,
                        newdata = x.te,
                        n.trees = B[j],
                        type = "response")
        yhat <- ifelse(phat > 0.5, 1, 0)
        ## error
        err.cv[i,j] <- mean(yhat != x.te$chd)
    }
}
err.cv
```

```
##           B=25      B=50      B=75     B=100     B=125     B=150     B=175      B=2
## [1,] 0.2272727 0.2727273 0.2727273 0.2727273 0.2727273 0.2954545 0.2954545 0.29545
## [2,] 0.2619048 0.3095238 0.3333333 0.3333333 0.3095238 0.3333333 0.3333333 0.30952
## [3,] 0.3518519 0.3148148 0.2962963 0.2962963 0.2962963 0.2592593 0.2777778 0.2962963
## [4,] 0.3478261 0.3913043 0.4130435 0.4130435 0.3913043 0.3695652 0.3695652 0.3695652
## [5,] 0.2888889 0.3111111 0.3111111 0.2666667 0.2666667 0.2666667 0.2666667 0.2888889

colMeans(err.cv)

##      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(err.cv))]

# 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)
phat.te <- predict(boost.train,
                   newdata = SAheart[-train,],
                   n.trees = b_best,
                   type="response")
yhat.te <- ifelse(phat.te > 0.5, 1, 0)
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

Again SVM are usable for regression and classification (we use them for classification)

SVM is a generalization of a simple/intuitive classifier called the maximal margin classifier (that unfortunately cannot be applied to most data sets, since it requires that the classes be separable by a linear boundary).

5.0.1 Maximal margin classifier

5.0.1.1 What is an hyperplane

Definition 5.0.1 (Hyperplane)**.** In a p-dimensional space, a hyperplane is a flat affine (= that need not pass through the origin) subspace of dimension $p-1$.

Example 5.0.1. Ie:

- for $p = 2$, a hyperplane is a flat one-dimensional subspace, i.e. a line
- for $p = 3$, a hyperplane is a flat two-dimensional subspace, i.e. a plane.
- if $p > 3$ dimensions, it can be hard to visualize a hyperplane, but the notion of a $(p-1)$-dimensional flat subspace still applies

In two dimensions a hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

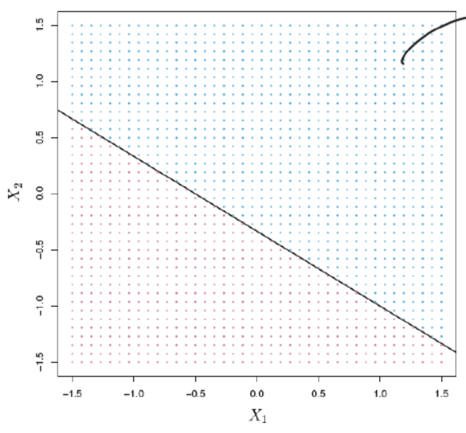
for parameters β_0, β_1 and β_2 .Any $X = (X_1, X_2)^T$ for which the equation holds is a point on the hyperplane The definition can be easily extended to the p-dimensional setting:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

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 1, then X lies on the hyperplane.

Now, suppose that X does not satisfy 1; if

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0$$
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0$$



If we substitute these coordinates in the eq. it turns out >0

Figura 5.1: SVM.

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 1.

Example 5.0.2. 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 $1 + 2X_1 + 3X_2 < 0$.

5.0.2 Classification using a separating hyperplane

Now suppose that:

- we have a $n \times p$ data matrix X that consists of n training observations in p-dimensional space

$$x_1 = \begin{bmatrix} x_{11} \\ \dots \\ x_{1p} \end{bmatrix}, \dots, x_n = \begin{bmatrix} x_{n1} \\ \dots \\ x_{np} \end{bmatrix}$$

  and that these observations fall into two classes, that is $y_1, ..., y_n \in \{-1, 1\}$, where -1 represents one class and 1 the other class.

- we also have a test observation, a p-vector of observed features $x^* = (x_1^*, ..., x_p^*)^T$.

- The goal is to develop a classifier based on the training data that will correctly classify the test observation using its feature measurements specifically, based upon the concept of a separating hyperplane.

Now:

- suppose that a perfect separating hyperplane exists that separates the training observations perfectly according to their class labels, y_i

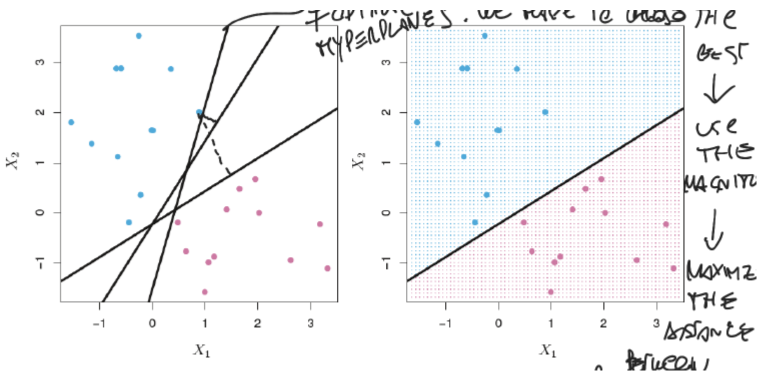


Figura 5.2: SVM.

- 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 $y_i \in \{-1, 1\}$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0, \quad \forall i = 1, \ldots n$$

- We want to find these β; if a separating hyperplane exists, we can use it to construct a very natural classifier: a test observation is assigned a class depending on which side of the hyperplane it is located.
  That is 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 $f(x^*)$ is is positive, then we assign the test observation to class 1;
  - otherwiser if $f(x^*)$ is is negative, then we assign it to class -1;

  We can also make use of the magnitude of $f(x^*)$
  - if $f(x^*)$ (a prediction that is positive or negative) is far from zero, then this means that x^* lies far from the hyperplane, and so we can be confident about our class assignment for x^*
  - if $f(x^*)$ is close to zero, then x^* is located near the hyperplane, and so we are less certain about the class assignment for x^*

5.0.3 Maximal margin classifier

In general:

- if our data can be perfectly separated using a hyperplane, then there will in fact exist an infinite number of such hyperplanes
- in figure 5.2 given different possible hyperplanes we have to choose the best using the magnitute; one natural choice is maximizing the distance between points and the candidate hyperplanes.

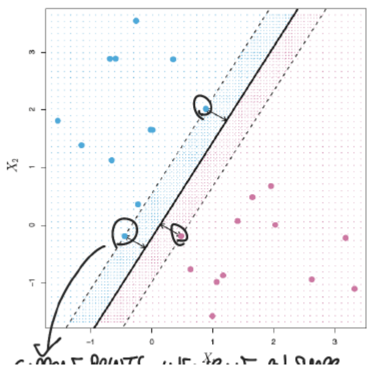


Figura 5.3: SVM.

- the maximal margin hyperplane (also known as the optimal separating hyperplane), is the one that is farthest from the training observations.
- to determine it we compute the distances (perpendicular) from each training observation to a given separating hyperplane;
- the smallest of such distance is the minimal distance from the observations to the hyperplane: it's called the *margin*.
- so the *maximal margin hyperplane* is the separating hyperplane for which the margin is largest: it is the hyperplane that has the greatest minimum distance to the training observations
- *maximal margin classifier* then classify a test observation based on which side of the maximal margin hyperplane it lies

Problems:

- although the maximal margin classifier is often successful, it can also lead to overfitting when p is large (it gives too importance to the nearest few observations)
- in figure 5.3 there are three training observations that are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin.
  These observations are known as *support vectors* since:
  - they are vectors in p-dimensional space
  - they "support" the maximal margin hyperplane: if these points were moved slightly then the maximal margin hyperplane would move as well (while moving any other observations would not).

For the construction of the maximal margin classifier:

- having n observation $x_1, \dots, x_n \in \mathbb{R}^p$ and associated class label $y_1, \dots, y_n \in \{-1, 1\}$ the maximal margin hyperplane is the solution to the optimization problem

$$\max_{\beta_0,\ldots,\beta_p} M, \text{subject to,} \tag{5.1}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M, \quad \forall i = 1, \ldots, n \tag{5.2}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{5.3}$$

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 $M > 0$ (actually this constraint requires that each observation be on the correct side of the hyperplane, with some cushion, provided that $M > 0$)
- the 5.3 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 kyperplane then so does

  $$k(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) = 0$$

  for any $k \neq 0$.
  However 5.3 adds meaning to 5.2; 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 5.3 and 5.2 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 $\beta_0, ..., \beta_p$ to maximize M.
  And this motherfucka is exactly the definition of the maximal margin hyperplane.

Final thoughts:

- maximal margin classifier is a very natural way to perform clas- sification, if a separating hyperplane exists.
- however, in many cases data are not perfectly separable, no separating hyperplane exists, and so there is no maximal margin classifier. In this case, the optimization problem has no solution with $M > 0$
- furthermore even the maximal margin classifier could be exposed to overfitting, looking especially at units near the margin; a different sample could produce sensibly different classifier;

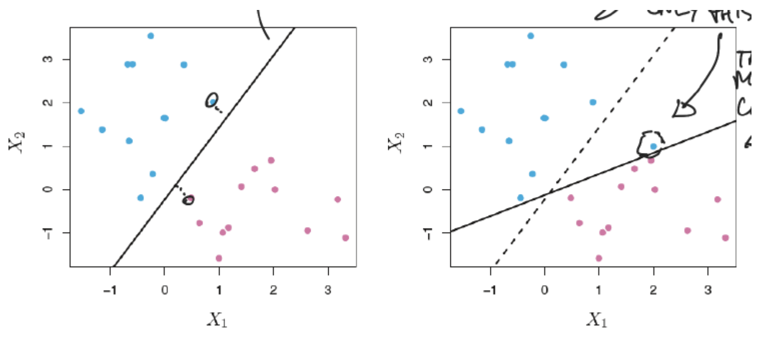


Figura 5.4: SVM.

- we can thus extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called *soft margin.*
- the generalization of the maximal margin classifier to the non- separable case is known as the support vector classifier.
- observations belonging to two classes are not always separable by a hyperplane.
- even if a separating hyperplane does exist, a classifier based on a separating hyperplane might not be desirable because can lead to sensitivity to individual observations.
- 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.
- however the resulting maximal margin hyperplane has a tiny margin: this is problematic because as discussed previously, the distance of an observation from the hyperplane can be seen as a measure of our confidence that the observation was correctly classified.

5.1 Support vector classifier

We might be willing to consider a classifier based on a hyperplane that does not perfectly separate the two classes. This would

- increase robustness to single/individual observations;
- could lead to a better classification of the training sample overall: it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations

The support vector classifier, sometimes called a *soft margin classifier*, allows a maximum number of observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane. In figure 5.5:

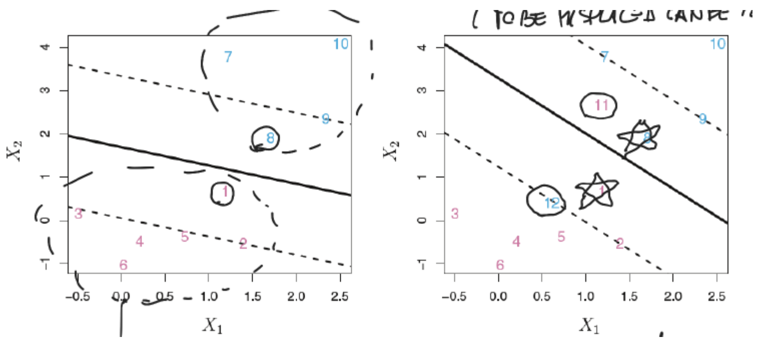


Figura 5.5: SVM.

- on left two observation (1 and 8) are in the correct side but in the margin;
- on right adding two observation (11 and 12) will be on the wrong side of the hyperplane (correspond to training observations that are misclassified by the classifier).

In support vector classifier the hyperplane is chosen to correctly separate most of the training observations into the two classes, but may misclassify a few observations (a parameter which can be finetuned). It is the solution to the following optimization problem

$$\max_{\beta_0,\ldots,\beta_p,\varepsilon_1,\ldots,\varepsilon_n} M, \text{subject to,} \tag{5.4}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i), \quad \forall i = 1, \ldots, n \tag{5.5}$$

$$\varepsilon_i \geq 0, \sum_{i=1}^{n} \varepsilon_i \leq C \tag{5.6}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{5.7}$$

where:

- M is still the width of the margin, the quantity to be maximizede
- the terms $\varepsilon_1, \ldots, \varepsilon_n$ in constraint 5.5 are *slack variables* that allow individual observations to be on the wrong side of the margin or 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 on the wrong side of the margin, and we say that the ith observation has violated the margin.
  - if $\varepsilon_i > 1$: then it is on the wrong side of the hyperplane.
- the tuning parameter C is a nonnegative constant to be choosen. It bounds the sum of the ε_i so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate.
  Therefore C is our budget; the number of points at most wrongly classified:

- if $C = 0$ then there is no budget for violations to the margin, and it must be the case that $\varepsilon_1 = ... = \varepsilon_n = 0$, in which case the problem amounts to the maximal margin hyperplane optimization problem (provided that the two classes are separable),
- if $C > 0$ no more than C observations can be on the wrong side of the hyperplane, because if an observation is on the wrong side of the hyperplane then $\varepsilon_i > 1$ and constraint 5.6 requires that $\sum_{i=1}^{n} \varepsilon_i \leq C$

Therefore

- as C increases we become more tolerant of violations to the margin, and so the margin will widen.
- as C decreases, we become less tolerant of violations to the margin and so the margin narrows

And thus C controls the bias-variance trade-off:

- when C is small, we seek narrow margins that are rarely violated; this amounts to a classifier that is highly fit to the data, which may have low bias but high variance.
- when C is larger, the margin is wider and we allow more violations to it; this amounts to fitting the data less hard and obtaining a classifier that is potentially more biased but may have lower variance.

C is typically chosen in crossvalidation. In R C is cost of an error (inverse interpretation)

Finally the classifier. 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

$$f(x^*) = \beta_0 + \beta_1 x_1^* + ... + \beta_p x_p^* = \begin{cases} > 0 & \text{then classified as } +1 \\ < 0 & \text{then classified as } -1 \end{cases}$$

Now only observations that either lie on the margin or that violate the margin will affect the hyperplane: an observation that lies strictly on the correct side of the margin does not affect the support vector classifier

Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as *support vectors.*

As the classifier's decision rule is based only on a potentially small (depending on the magnitude of C) subset of the training observations, the support vector classifier is quite robust to the behavior of observations that are far away from the hyperplane. Which is not the case for other classifiers:

- LDA otoh depeds on the mean of all of the observations within each class, as well as the within-class covariance matrix computed using all of the observations

- logistic regression has very low sensitivity to observations far from the decision boundary.

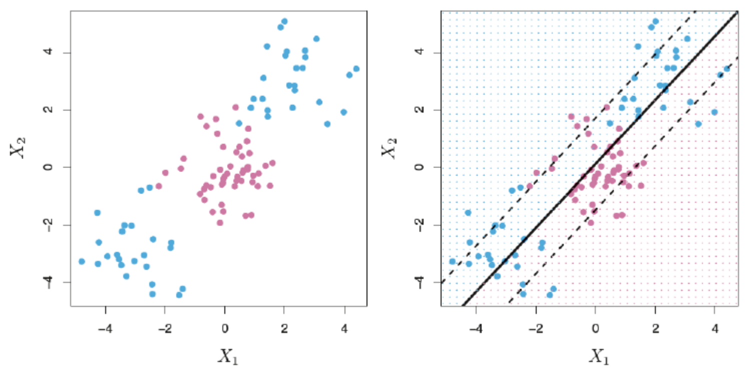


Figura 5.6: SVM.

5.2 Support vector machines

Only problem with support vector classifier is that they allows only for *linear boundaries.* Eg in figure 5.6 a support vector classifier (right) or any linear classifier will perform poorly here.
In the case of the support vector classifier, we could address the problem of possibly non-linear boundaries between classes by enlarg- ing the feature space using quadratic, cubic, and even higher-order polynomial functions of the predictors.
Eg rather than fitting a support vector classifier using p features $X_1, \ldots, X_p$ we could instead fit a support vector classifier using $2p$ features by adding the squares (or higher power or interaction).
There are many possible ways to enlarge the feature space, unless are careful, we could end up with a huge number of features:

- computations would become unmanageable.
- is number of parametere increase and we could increase variance

And this is where SVM kick in.
The support vector machine (SVM):

- is an extension of the support vector classifier that results from enlarging the feature space, in order to accommodate a non-linear boundary between the classes, using special functions called *kernels.*
- can be shown that the solutions of the optimization instead of looking at dataset by columns (multiplying features by a constant) look at them by rows, that is involve the *inner product* between observations. The inner product of two observation x_i and x_i' is given by

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

in general, kernels are generalization of this inner product.

- can be shown that the linear support vector classifier boundary ($f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$ can be rewritten as below represented as

  $$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

  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)

  - 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 .
  - regarding the αs, it can be shown furthermore 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
  - it turns out that α_i is nonzero only for the support vectors in the solution: 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 than

- Now suppose that every time the inner product appears in the representation of $f(x)$, or in a calculation of the solution for the support vector classifier, we replace it with *a generalization of the inner product* of the form $K(x_i, x_{i'})$ where K is some function that we will call a *kernel.* A kernel is a function that quantifies the similarity of two observations. For instance

  - we could simply take

    $$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

    (measure the distance between pairs if centered) which would just give us back the support vector classifier and it is known as a *linear kernel* because the support vector classifier is linear in the features; basically, this kernel quantifies the similarity of a pair of observations using Pearson (standard) correlation.
  - we could adopt

    $$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

    which is known as *polynomial kernel* of degree d ($d > 0$). Using such a kernel with $d > 1$ leads to a much more flexible decision boundary.

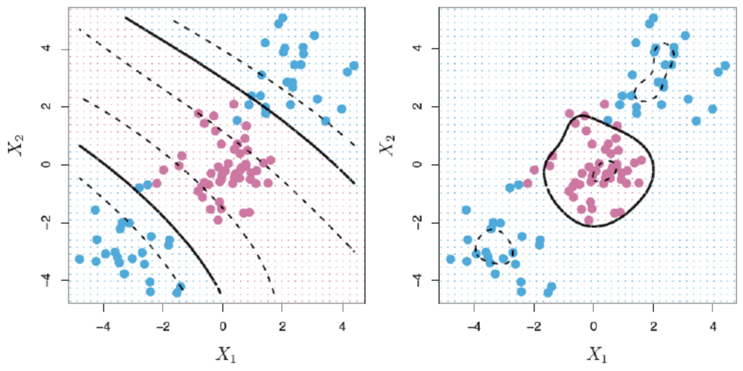


Figura 5.7: SVM.

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.

– Another popular non-linear kernel is the *radial kernel*

$$K(x_i, x_{i'}) = \exp\left(-\gamma\sum_{j=1}^{p}(x_{ij} - x_{i'j})^2\right)^d$$

with $\gamma > 0$.
Regarding 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.

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.* In figure 5.7 both a polynomial kernel of degree 3 (left) and a radial kernel (right) are capable of capturing the decision boundary.
Type of kernel chosen is important and we do it by crossvalidation.

5.3 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.df <- data.frame(chd=as.factor(y),x)

n <- nrow(SAheart)
set.seed(1234)
train <- sample(1:n,ceiling(n/2),replace = F)

## ?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.df, kernel="linear", cost=10)
summary(out.svm)

##
## Call:
## svm(formula = chd ~ ., data = heart.df, 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.df, kernel="linear", cost=0.1)
summary(out.svm)

##
## Call:
## svm(formula = chd ~ ., data = heart.df, 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

## 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
set.seed(1234)
tune_out <- tune(svm, # function to tune
                 chd ~ ., # below the training function parameters
                 data = heart.df[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.df[-train,])

## MSE
table(yhat.linear, SAheart$chd[-train])

##
## yhat.linear   0   1
##           0 138  53
##           1   8  32

mean(yhat.linear != SAheart$chd[-train])

## [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.df[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$best.model
yhat.radial<-predict(best_model_radial,newdata = heart.df[-train,])
table(yhat.radial,SAheart$chd[-train])

##
## yhat.radial   0   1
##           0 134  62
##           1  12  23

mean(yhat.radial!=SAheart$chd[-train])

## [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.df[train,],
                  kernel = "radial",
                  gamma = 1,
                  cost = 0.1)
yyhat <- predict(out.radial, newdata = heart.df[-train,])
table(yyhat,heart.df$chd[-train])

##
## 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.df[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.df[-train,])

table(yhat.polynomial,SAheart$chd[-train])

##
## yhat.polynomial   0   1
##               0 138  53
##               1   8  32

mean(yhat.polynomial!=SAheart$chd[-train])

## [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