MAT 3373
Methods of Machine Learning

Chapter 3
Classification and Supervised Learning

P. Boily (uOttawa)

Fall – 2023

Outline

Outline

Outline

Outline

Outline

Main References:

- *Data Understanding, Data Analysis, and Data Science*, chapter 21.
- *Data Understanding, Data Analysis, and Data Science*, chapter 19.

3 – Classification and Supervised Learning

In Chapter 1 (*Machine Learning 101*), we provided a (basically) math-free general overview of machine learning.

In this chapter, we continue our mathematical treatment of **supervised learning**, with a focus on **classification**, **ensemble learning**, and **non-parametric supervised methods**.

This is a continuation of the treatment provided in Chapter 2 (*Regression and Value Estimation*) and a prelude to Chapter 4 (*Clustering*).

3.1 – Classification Overview

We discuss **classification** in the same context as we discussed **regression** and **value estimation** in Chapter 2:

- **training sets** Tr and **testing sets** Te for a dataset with n observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$,
- p **predictors** $X_1, \ldots, X_p$, and
- a **response variable** Y.

Note: in the **design matrix** $\mathbf{X}$, a row corresponds to the **signature vector** of an observation (the **values of the predictors**); when we write $\mathbf{x}$ or $\boldsymbol{\beta}$, we typically understand those to be **column vectors**.

We sometimes abuse the notation: when in doubt, all matrices/vectors involved must have **compatible dimensions** when multiplied or compared.

3.1.1 – Formalism

In a **classification setting**, the response Y is **categorical**: $Y \in \mathcal{C}$, where $\mathcal{C} = \{C_1, \ldots, C_K\}$, but the SL objectives remain the same:

- build a **classifier** $C(\mathbf{x}^*)$ **assigning a label** $C_k \in \mathcal{C}$ to Te observations $\mathbf{x}^*$;
- understand the **role of the predictors in the assignment**, and
- assess the **uncertainty** and the **accuracy** of the **classifier**.

Main difference with regression/value estimation: no $\mathrm{MSE}-$**type** metric to evaluate the **classifier performance**.

What is the counterpart of the regression function

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]?$$

For $1 \leq k \leq K$, let $p_k(\mathbf{x}) = P(Y = C_k \mid \vec{X} = \mathbf{x})$, the **most numerous categorical label** of observations for which the **signature vector** is $\mathbf{x}$.

The **Bayes optimal classifier** at $\mathbf{x}$ is the function

$$C(\mathbf{x}) = C_j, \quad \text{where } p_j(\mathbf{x}) = \max\{p_1(\mathbf{x}), \ldots, p_K(\mathbf{x})\}.$$

With too few observations at $\vec{X} = \mathbf{x}$, we use **nearest neighbour averaging**:

$$\hat{C}(\mathbf{x}) = C_j, \quad \text{where } \tilde{p}_j(\mathbf{x}) = \max\{\tilde{p}_1(\mathbf{x}), \ldots, \tilde{p}_K(\mathbf{x})\},$$

and $\tilde{p}_k(\mathbf{x}) = P(Y = C_k \mid \vec{X} \in N(\mathbf{x}))$ and $N(\mathbf{x})$ is a **neighbourhood** of $\mathbf{x}$.

The **misclassification error rate** is

$$\mathcal{E}_{\text{Te}} = \frac{1}{M} \sum_{j=N+1}^{M} \mathcal{I}[y_j \neq \tilde{C}(\mathbf{x}_j)],$$

where $\mathcal{I}$ is the **indicator function**

$$\mathcal{I}[\text{condition}] = \begin{cases} 0 & \textbf{if the condition is false} \\ 1 & \textbf{otherwise} \end{cases}$$

The **Bayes optimal classifier** $C(\mathbf{x})$ is the **optimal classifier** w.r.t. to $\mathcal{E}_{\text{Te}}$.

In most cases, we cannot find this classifier exactly.

The **Bayes error rate**

$$\eta_{\mathbf{x}} = 1 - \mathrm{E}\left[\max_k P(Y = C_k \mid \vec{X} = \mathbf{x})\right]$$

corresponds to the **irreducible error** ($\mathrm{Var}(\varepsilon)$ from the regression context) and provides a **lower limit** on any classifier's **expected error**.

- some classifiers build models $\hat{C}(\mathbf{x})$ which **directly** approximate $C(\mathbf{x})$ (**support vector machines**, **naïve Bayes classifiers**, etc.)
- some build models $\hat{p}_k(\mathbf{x})$ for the **conditional probabilities** $p_k(\mathbf{x})$, $1 \leq j \leq K$, which are then used to build $\hat{C}(\mathbf{x})$ (**logistic regression**, **GAM**, k**NN**, etc.).

The latter models are **calibrated** (the values of $\hat{p}_k(\mathbf{x})$ represent **relative probabilities**); the former are **non-calibrated** (only the **most likely outcome** is provided).

3.1.2 – Model Evaluation

The **confusion matrix** of a classifier on Te evaluates its performance:

		prediction	
		0	1
actual	0	TP	FN
	1	FP	TN

- TP: **true positive**
- FN: **true negative**
- FP: **false positive**
- TN: **true negative**.

There are various **classifier evaluation metrics**.

If the testing set Te has M observations, then:

- **accuracy**: measures the correct classification rate $\dfrac{\mathrm{TP}+\mathrm{TN}}{M}$
- **misclassifcation**: is $\dfrac{\mathrm{FP}+\mathrm{FN}}{M} = 1 - \textbf{accuracy}$
- **false positive rate** (FPR): $\dfrac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$
- **false negative rate** (FNR): $\dfrac{\mathrm{FN}}{\mathrm{TP}+\mathrm{FN}}$
- **true positive rate** (TPR): $\dfrac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
- **true negative rate** (TNR): $\dfrac{\mathrm{TN}}{\mathrm{FP}+\mathrm{TN}}$

Other measures: F_1-score, Matthews' correlation coefficient, multi-label metrics, etc. (do not get beholden to **a single** metric).

3.1.3 – Bias-Variance Trade-Off

The **bias-variance trade-off** also applies to classifiers, although the **decomposition** is $\mathcal{E}_{\mathrm{Te}}$ is necessarily different.

In a $k-$**nearest neighbours (**k**NN)** classifier, say, the prediction for a **new** observation with predictors $\mathbf{x}^*$ is the **most frequent class label** of the k nearest neighbours to $\mathbf{x}^*$ in Tr on which $\hat{C}_{k\mathrm{NN}}(\mathbf{x})$ is **trained**.

As $k \nearrow$, the **complexity** of $\hat{C}_{k\mathrm{NN}}(\mathbf{x}) \searrow$ decreases, and *vice-versa* (why?). We would thus expect:

- a model with a large k to **underfit** the data;
- a model with a small k to **overfit** the data, and
- models in the "Goldilock zone" to strike a balance between **prediction accuracy** and **interpretability of the decision boundary**.

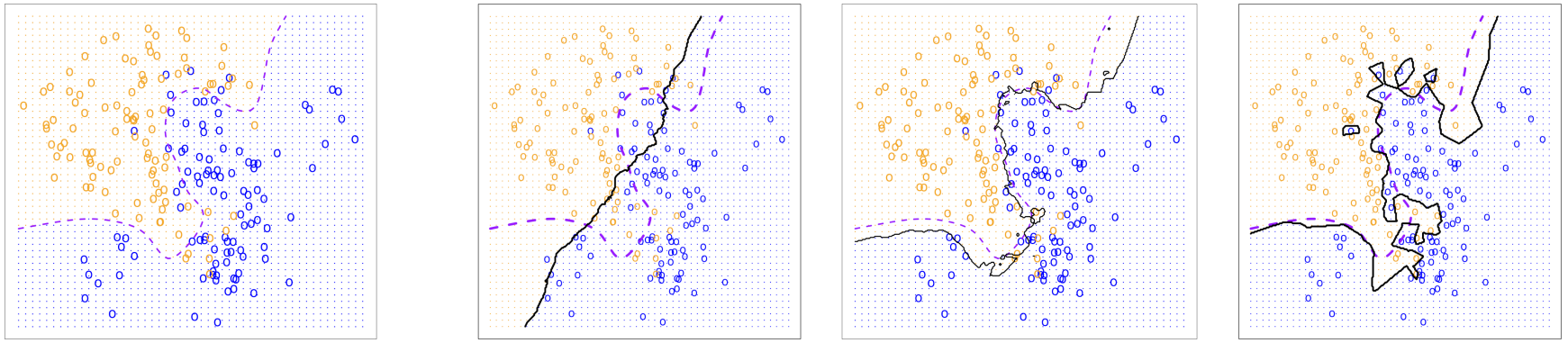


Illustration of the accuracy-boundary interpretability trade-off for classifiers on an artificial dataset (from left to right):

- Bayes optimal classifier $C(\mathbf{x})$
- **underfit** $\hat{C}_{100\text{NN}}(\mathbf{x})$ model
- **Goldilock** $\hat{C}_{10\text{NN}}(\mathbf{x})$ model
- **overfit** $\hat{C}_{1\text{NN}}(\mathbf{x})$ model.

Interplay: **prediction accuracy** vs. **decision boundary complexity**.

Comparison Between kNN and OLS

We illustrate the bias-variance trade-off by comparing OLS (**rigid**, **simple** model) and kNN (**flexible**, **complex** model).

Given an input vector $\mathbf{z} \in \mathbb{R}^p$, ($k$NN) predicts the response Y as:

$$\hat{Y} = \text{Avg}\{Y(\mathbf{x}) \mid \mathbf{x} \in N_k(\mathbf{z})\} = \frac{1}{k} \sum_{\mathbf{x} \in N(\mathbf{z})} Y(\mathbf{x}),$$

where $Y(\mathbf{x})$ is the known response at predictor $\mathbf{x} \in \text{Tr}$ and $N_k(\mathbf{z})$ is the set of the k **training** observations **nearest to** $\mathbf{z}$.

In classification, kNN models use the **mode** instead of the **average**.

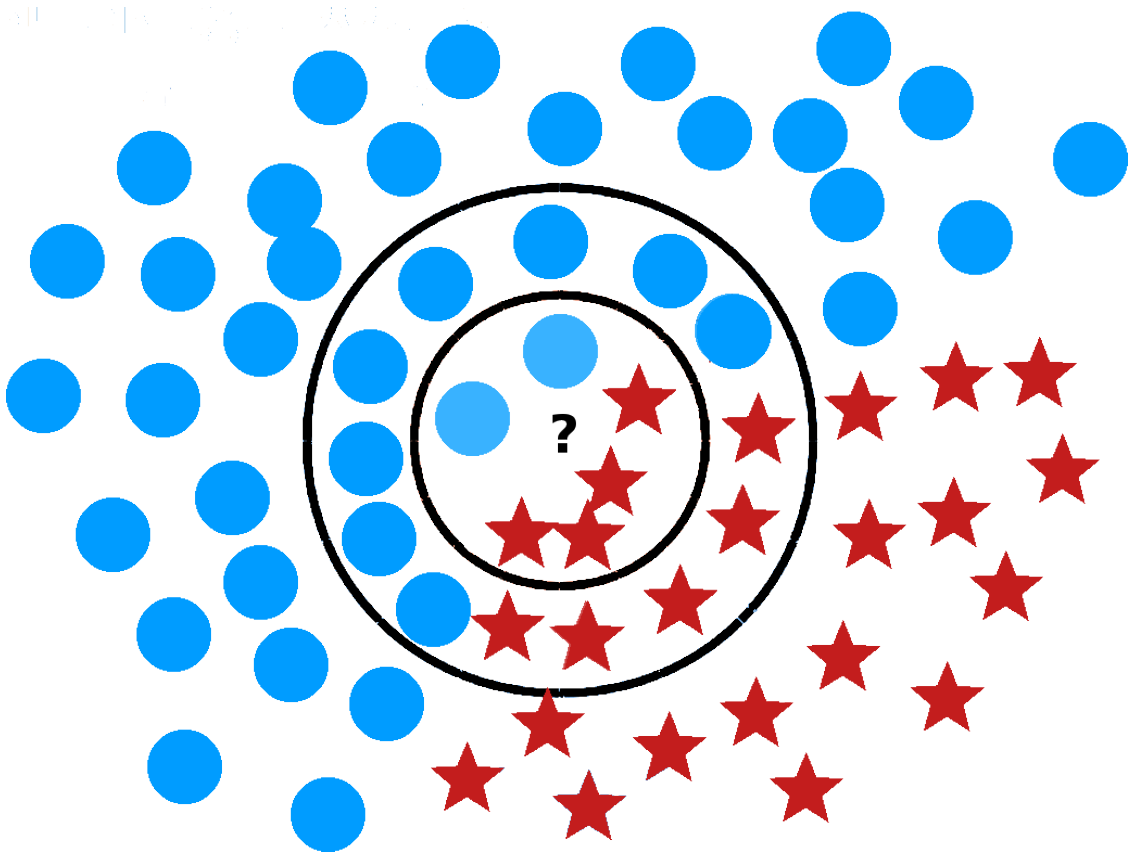


Illustration of kNN

The prediction may depend on the **value of k**: the 6NN prediction would be a **red star**; the 19NN prediction would be a **blue disk**.

Example: the following classification example illustrates some of the bias-variance trade-off **consequences**.

Consider a training dataset Tr consisting of 200 observations with features $(x_1, x_2) \in \mathbb{R}^2$ and responses $y \in \{\textbf{BLUE}_{(=0)}, \textbf{ORANGE}_{(=1)}\}$.

Let $[\cdot] : \mathbb{R} \to \{\textbf{BLUE}, \textbf{ORANGE}\}$ denote the function

$$[w] = \begin{cases} \textbf{BLUE} & w \leq 0.5 \\ \textbf{ORANGE} & w > 0.5 \end{cases}$$

Linear Fit

We fit an OLS model

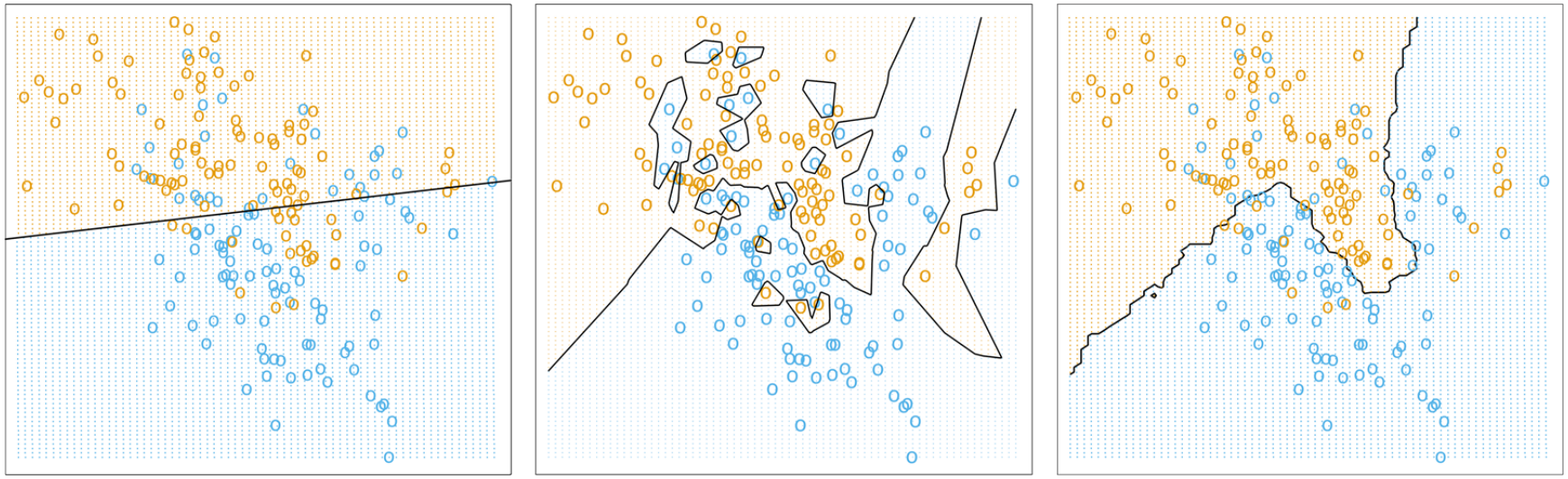
$$\hat{y}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 \quad \text{on } \mathrm{Tr};$$

the **class prediction** is $\hat{g}(\mathbf{x}) = [\hat{y}(\mathbf{x})]$. The **decision boundary**

$$\partial_{\mathrm{OLS}} = \{(x_1, x_2) \mid \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5\}$$

is shown on the next slide.

The boundary is a straight line which can be described using only2 **effective parameters** – their number is a measure of a model's **complexity**.



Classification based on OLS (left), 1NN (middle), and 15NN (right).

There are several **misclassifications** on both sides of ∂_{OLS}; even though errors seem to be **unavoidable**, the OLS model is **too rigid**.

How does that compare to the kNN models?

kNN Fit

If $\hat{y}(\mathbf{x})$ is the proportion of **ORANGE** points in $N_k(\mathbf{x})$, then

$$\hat{g}(\mathbf{x}) = [\hat{y}(\mathbf{x})].$$

The decision boundaries $\partial_{1\text{NN}}$ and $\partial_{15\text{NN}}$ are also displayed.

They are both **irregular**: $\partial_{1\text{NN}}$ is **overfit**; $\partial_{15\text{NN}}$ seems **less likely** to be – although neither is great w.r.t. to **interpretability** (what does that mean?).

The **effective parameters** are not as obviously defined for kNN models.

One approach: kNN is a model that fits 1 parameter (a **mean**) to each **ideal** (non-overlapping) **neighbourhood** in the data.

The number of **effective parameters** is $\approx$ equal to the number of these **neighbourhoods**:

$$\frac{N}{k} \approx \begin{cases} 13 & \text{when } k = 15 \\ 200 & \text{when } k = 1 \end{cases}$$

The kNN models are **complex**, in comparison with the OLS model.

There are **no** misclassification for $k = 1$, and **some** for $k = 15$, but **not as many** as for the OLS model.

The 15NN model seems to **strike a balance** between various competing properties; it is likely nearer the "**sweet spot**" of the test error curve.

Remember: we have not evaluated these models on Te – we have only described some of their behaviours on Tr.

Conclusions

The **OLS model** is:

- **stable** – adding a few training observations is **not likely** to alter the fit substantially;
- **biased** – the assumptions of a valid linear fit is **questionable**.

The k**NN models** are:

- **unstable** – adding a few training observations is **likely** to alter the fit substantially (especially for **small** k);
- **unbiased** – **no apparent** assumptions are made about the data.

The best approach depends on the ultimate task: description, prediction, etc.

In predictive DS, ML, and AI, the **validity of modeling assumptions** takes a backseat to a model's ability to **make good predictions on new (and unseen) observations**.

Expectation: models whose assumptions are met are **more likely** to make good predictions than models whose assumptions are **not** met**, but this does not need to be the case**.

OLS theory is mature and extensive, and we could have discussed a number of their other features and properties.

Important notes: ML methods are not meant to **replace** classical statistical analysis methods but to **complement** them.

They simply provide different approaches to **gain insights from data**.

Summary

Qualitative variables take values in an **unordered** set $\mathcal{C} = \{C_1, \ldots, C_K\}$:

- hair colour $\in \{\text{black}, \text{red}, \text{blond}, \text{grey}, \text{other}\}$
- email message $\in \{\text{ham}, \text{spam}\}$
- life expectancy $\in \{\text{high}, \text{low}\}$

For a training set Tr with observations $(\vec{X}, Y) \in \mathbb{R}^p \times \mathcal{C}$, the **classification problem** is to build a function $\hat{C} : \mathbb{R}^p \to \mathcal{C}$ to approximate the **optimal Bayes classifier** $C : \mathbb{R}^p \to \mathcal{C}$ which minimizes $\mathcal{E}_{\mathrm{Te}}$.

Typically, the classifier $\hat{C}$ is **built** on a training set

$$\text{Tr} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{N}$$

and **evaluated** on a testing set

$$\text{Te} = \{(\mathbf{x}_i, y_i)\}_{i=N+1}^{M}.$$

Often, we are more interested in the probabilities

$$\pi_k(\mathbf{x}) = P\{\hat{C}(\mathbf{x}) = C_k \mid \vec{X} = \mathbf{x}\}, \quad k = 1, \ldots, K$$

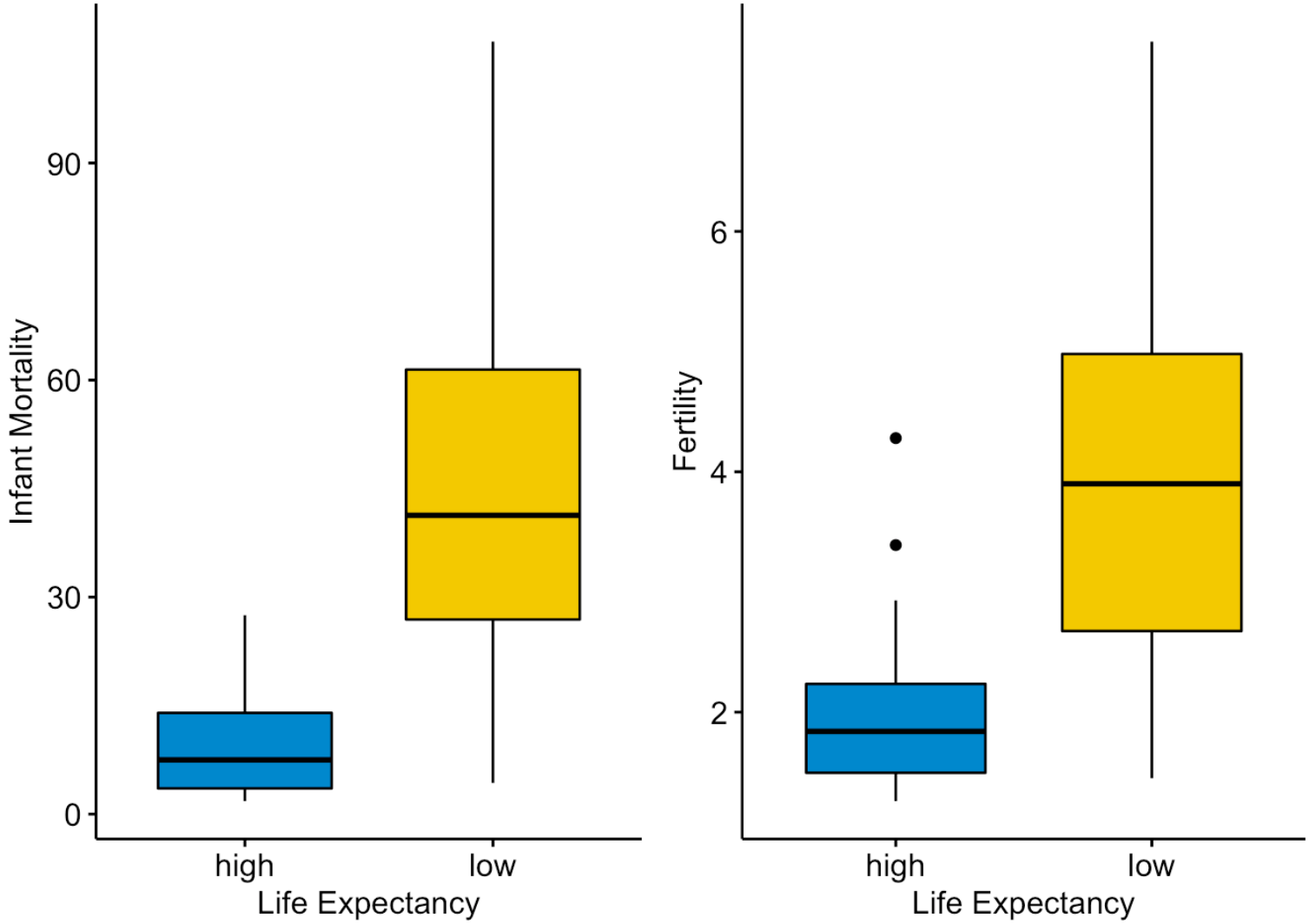
than in the classification predictions $\hat{C}(\mathbf{x})$ themselves.

3.1.4 – Example: Gapminder Dataset

We revisit the 2011 `Gapminder` dataset, with life expectancy recorded as "**high**" ($\texttt{LE} = 1$) if it falls above 72.45 (the **median** in 2011), and as "**low**" ($\texttt{LE} = 0$) otherwise (DUDADS, 21.2, *Simple Classifcation Methods*).

```
infant_mortality    fertility        LE
Min.   :  1.800   Min.   :1.260   high:83
1st Qu.:  7.275   1st Qu.:1.792   low :83
Median : 16.900   Median :2.420
Mean   : 27.333   Mean   :2.931
3rd Qu.: 41.125   3rd Qu.:3.908
Max.   :106.800   Max.   :7.580
```

We model the response **LE** (Y) as a linear response of the predictors X_1 (**infant mortality**) and X_2 (**fertility**), using OLS.


Infant Mortality
90
60
30
0
high
low
Life Expectancy
Fertility
6
4
2
high
low
Life Expectancy

We build an OLS regression of Y on $\vec{X}$ over Tr to obtain the model

$$\hat{Y} = 1.001 - 0.012 \cdot \textbf{infant mortality} - 0.060 \cdot \textbf{fertility},$$

from which we would classify an observation's LE as

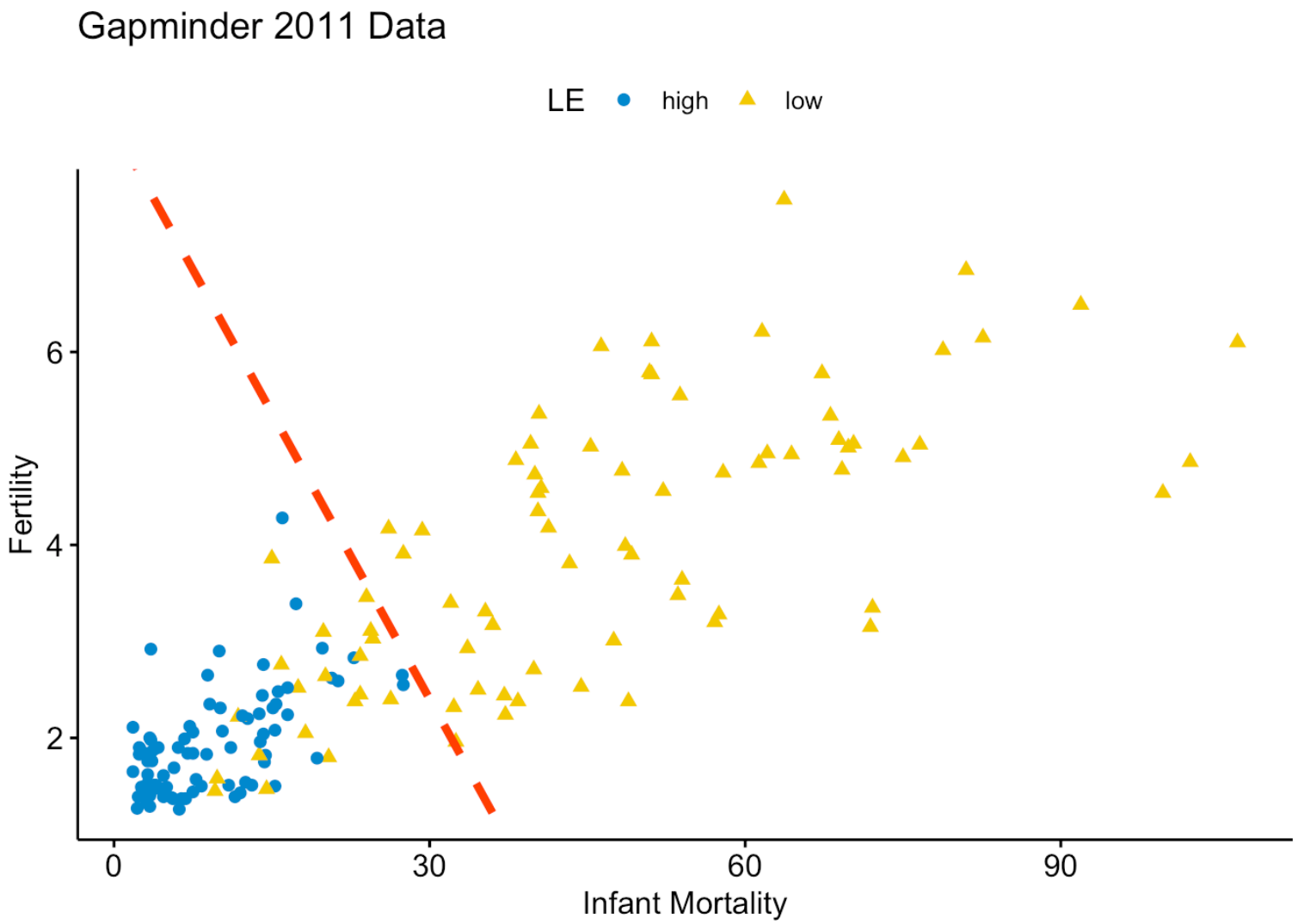
$$\hat{C}(\vec{X}) = \begin{cases} \texttt{high} & \text{if } \hat{Y} > 0.5 \\ \texttt{low} & \text{else} \end{cases}$$

If the **decision boundary** splits the data at $\hat{Y} = 0.5$ ($\gamma \in [0, 1]$), then it solves

$$0.5 = \gamma = \beta_0 + \beta_1 X_1 + \beta_2 X_2, \quad \beta_2 \neq 0,$$

so that

$$X_2 = \left(\frac{\gamma - \beta_0}{\beta_2}\right) - \frac{\beta_1}{\beta_2} X_1 = \left(\frac{0.5 - 1.001}{-0.060}\right) - \frac{-0.012}{-0.060} X_1.$$


Gapminder 2011 Data
LE
high
low
Fertility
6
4
2
0
30
60
90
Infant Mortality

OLS does a **decent job** since Tr is roughly **linearly separable** over the predictors. **This will not usually be the case, however**.

In this example, the optimal regression function is

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}] = P(Y = 1 \mid \vec{X} = \mathbf{x}) = p_1(\mathbf{x})$$

because Y is a **binary variable**; perhaps $f(\mathbf{x})$ could then be used to **directly classify** and determine the **class probabilities** for the data, in which case there would be no need for a **separate classification apparatus**.

Major drawback: with OLS, we assume that $f(\mathbf{x}) \approx \mathbf{x}^\top \boldsymbol{\beta}$, so we need to insure that $\hat{f}_{\mathsf{OLS}}(\mathbf{x}) \in [0, 1]$ for all $\mathbf{x} \in \mathrm{Te}$ (unlikely).

Another problem: if the response $Y = \{0, 1\}$ arises from OLS, we have

$$Y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i, \quad \mathbf{x}_i \in \mathrm{Te}\,.$$

Then for all $\mathbf{x}_i \in \mathrm{Te}$,

$$\varepsilon_i = Y_i - \mathbf{x}_i^\top \boldsymbol{\beta} = \begin{cases} 1 - \mathbf{x}_i^\top \boldsymbol{\beta} & \text{if } Y_i = 1 \\ -\mathbf{x}_i^\top \boldsymbol{\beta} & \text{if } Y_i = 0 \end{cases}$$

When $\vec{X} = \mathbf{x}_i$, OLS assumes that $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, which is clearly not the case here.

Furthermore, if $Y_i \sim \mathcal{B}(n, p_1(\mathbf{x}_i))$ are independent, then

$$\begin{aligned} \mathrm{Var}(Y_i) &= p_1(\mathbf{x}_i)(1 - p_1(\mathbf{x}_i)), \qquad \text{and} \\ \mathrm{Var}(\varepsilon_i) &= \mathrm{Var}(Y_i - p_1(\mathbf{x}_i)) = \mathrm{Var}(Y_i) = p_1(\mathbf{x}_i)(1 - p_1(\mathbf{x}_i)), \end{aligned}$$

which is not constant as it depends on $\mathbf{x}_i \implies$ OLS is **not appropriate**.

There is another way in which OLS could fail. When $\mathcal{C}$ contains more than 2 level ($\mathcal{C} = \{\texttt{low}, \texttt{medium}, \texttt{high}\}$), Y is encoded using numerals to facilitate OLS implementation:

$$Y = \begin{cases} 0 & \text{if } \texttt{low} \\ 1 & \text{if } \texttt{medium} \\ 2 & \text{if } \texttt{high} \end{cases}$$

This encoding suggests an **ordering**/**scale** between the levels (the difference between "high" / "medium" is **equal** to that between "medium" / "low", and **half again as large** as that between "high" / "low").

All-in-all, OLS is **not a good fit/modeling approach** to estimate

$$p_k(\mathbf{x}) = P(Y = C_k \mid \vec{X} = \mathbf{x}).$$

3.2 – Logistic Regression

The problems discussed in 3.1.4 point to OLS **not being a great classification model** ... but it still provided a **decent decision boundary** in the Gapminder example.

This suggests that we should not automatically reject the possibility of:

1. **transforming the data**, and

2. determining if the OLS framework (and machinery) provides an appropriate modeling strategy on the **transformed data**.

In **logistic regression**, we seek an **invertible** transformation $g : \mathbb{R} \to [0, 1]$, with $g(y^*) = y$, $g^{-1}(y) = y^*$ ($Y-$quantities must behave as **probabilities**).

3.2.1 – Formulation

In the $2-$class setting, we search for $g(y^*)$ to approximate

$$p_1(\mathbf{x}) = P(Y = 1 \mid \vec{X} = \mathbf{x}).$$

The idea is to run OLS on a transformed training set

$$\mathrm{Tr}^* = \{(\mathbf{x}_i, y_i^*)\}_{i=1}^N,$$

and to **transform the results back** using $y_i = g(y_i^*)$.

Commonly-used functions: the **probit** model ($g_P(y^*) = \Phi(y^*)$, where Φ is the c.d.f. of $\mathcal{N}(0,1)$, out-of-scope) and the **logit** model.

3.2.2 – Logit Model

The logit model uses the transformation

$$y = g_L(y^*) = \frac{e^{y^*}}{1 + e^{y^*}}.$$

It is such that

$$g_L^{-1}(0) = -\infty, \quad g_L^{-1}(1) = \infty, \quad g_L^{-1}(0.5) = 0, \quad \text{etc.}$$

We solve for y^* in order to get a **transformed** response $y^* \in \mathbb{R}$, as opposed to the **original restricted** response $y \in [0, 1]$):

$$p_1(\mathbf{x}) = \frac{e^{y^*}}{1 + e^{y^*}} \iff y^* = g_L^{-1}(y) = \ln\left(\frac{p_1(\mathbf{x})}{1 - p_1(\mathbf{x})}\right).$$

It is the **log-odds** transformed observations that we fit with an OLS model:

$$\hat{Y}^* = \ln\left(\frac{p_1(\mathbf{x})}{1-p_1(\mathbf{x})}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = \mathbf{x}^\top \boldsymbol{\beta}.$$

We approximate $p_1(\mathbf{x})$ by estimating y^* and using the **logit transformation** to recover y.

If $\mathbf{x}^\top \hat{\boldsymbol{\beta}} = 0.68$, say, then

$$\hat{y}^* = \ln\left(\frac{\hat{p}_1(\mathbf{x})}{1-\hat{p}_1(\mathbf{x})}\right) = 0.68 \quad \text{and} \quad \hat{p}_1(\mathbf{x}) = \frac{e^{y^*}}{1+e^{y^*}} = \frac{e^{0.68}}{1+e^{0.68}} = 0.663.$$

Depending on the **decision rule threshold** γ, we predict that $\hat{C}(\mathbf{x}) = C_1$ if $p_1(\mathbf{x}) > \gamma$ or $\hat{C}(\mathbf{x}) = C_2$, otherwise.

The coefficients $\hat{\boldsymbol{\beta}}$ are found by **maximizing the likelihood**:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} f(Y_i) = \prod_{y_i=1} p_1(\mathbf{x}_i) \prod_{y_i=0} (1 - p_1(\mathbf{x}_i))$$

$$= \prod_{y_i=1} \frac{\exp(\mathbf{x_i}^\top \boldsymbol{\beta})}{1 + \exp(\mathbf{x_i}^\top \boldsymbol{\beta})} \prod_{y_i=0} \frac{1}{1 + \exp(\mathbf{x_i}^\top \boldsymbol{\beta})} \quad \text{or, more simply:}$$

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \{L(\boldsymbol{\beta})\} = \arg\max_{\boldsymbol{\beta}} \{\ln L(\boldsymbol{\beta})\}$$

$$= \arg\max_{\boldsymbol{\beta}} \left\{ \sum_{y_i=1} \ln p_1(\mathbf{x}_i) + \sum_{y_i=0} \ln(1 - p_1(\mathbf{x}_i)) \right\}$$

$$= \ldots \textbf{(terms in } \boldsymbol{\beta} \textbf{ and the observations } \mathbf{x}_i\textbf{)}.$$

The optimizer $\hat{\boldsymbol{\beta}}$ is then found using numerical methods; in `R`, the function `glm()` computes the **MLE** directly.

3.2.3 – Example: Gapminder Dataset

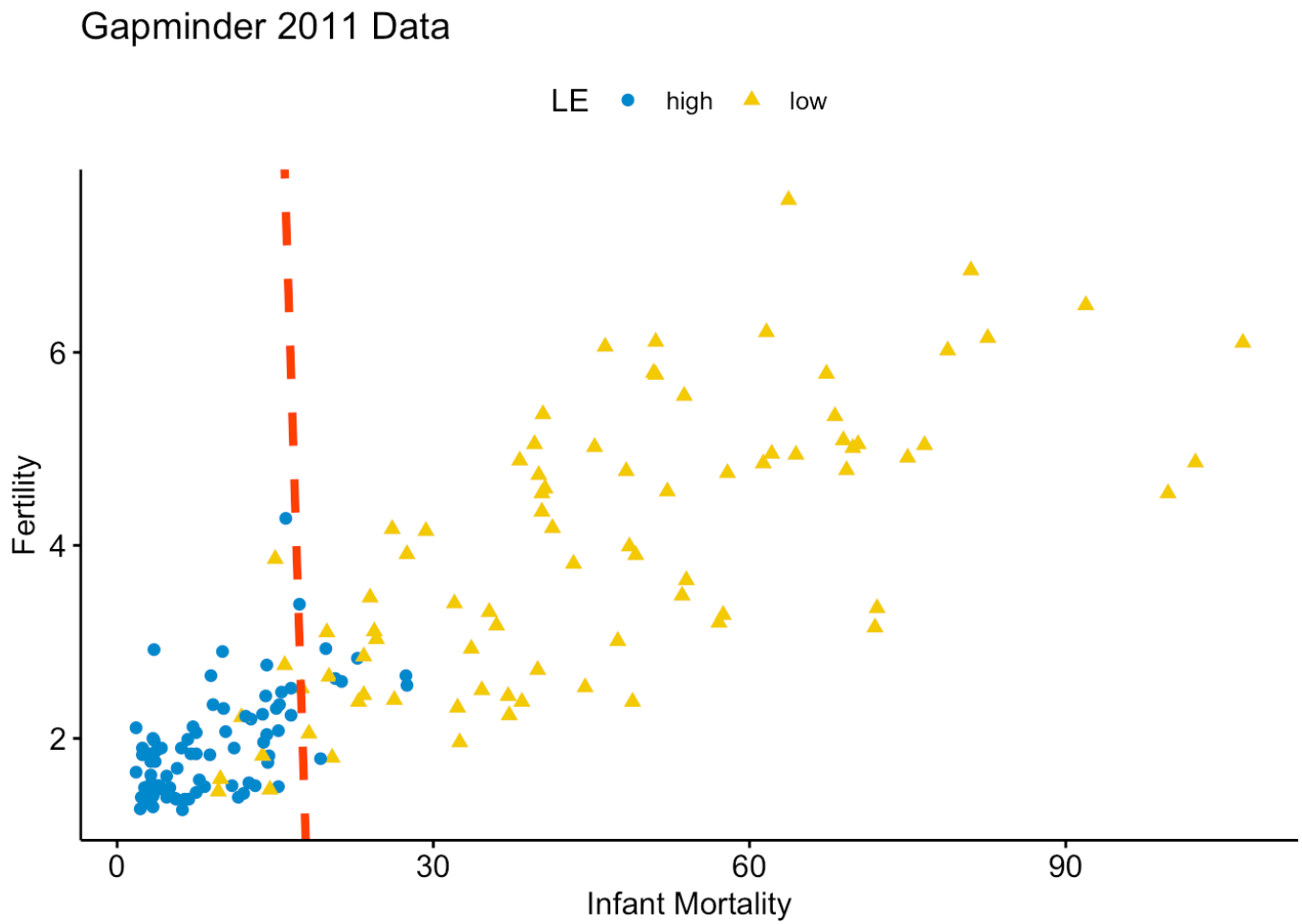
Using the `Gapminder` data from Section 3.1.4, we obtain the following logistic regression (LR) model (DUDADS, 21.2.1, *Logistic Regression*):

```
Coefficients:
     (Intercept)  infant_mortality          fertility
         4.58733           -0.22499           -0.06495

Degrees of Freedom: 165 Total (i.e. Null);  163 Residual
Null Deviance:      230.1
Residual Deviance: 78.17    AIC: 84.17
```

Thus

$$\hat{y}^* = \ln\left(\frac{P(Y = \texttt{high} \mid \vec{X})}{1 - P(Y = \texttt{high} \mid \vec{X})}\right) = 4.59 - 0.22X_1 - 0.06X_2.$$



Compare the LR decision boundary for threshold $\gamma = 0.5$ with the OLS decision boundary

What is the estimated probability that the life expectancy is `high` in a country with infant mortality 15 and fertility 4?

By construction,

$$\begin{aligned} p_1(Y = \texttt{high} \mid X_1 = 15, X_2 = 4) &\approx g_L([1, 15, 4]^\top \hat{\boldsymbol{\beta}}) \\ &= \frac{\exp(4.59 - 0.22(15) - 0.06(4))}{1 + \exp(4.59 - 0.22(15) - 0.06(4))} \\ &= \frac{\exp(0.9526322)}{1 + \exp(0.9526322)} = 0.72. \end{aligned}$$

How does this square up with the statistical learning framework of Chapters 1 and 2?

No Te has made an appearance, no **misclassification** or MSE rates have been calculated.

We remedy the situation by **randomly** selecting $N = 116$ observations, say, into a **training set** Tr, on which we build a logistic regression model:

```
Coefficients:
     (Intercept)  infant_mortality          fertility
          6.1194           -0.2050            -0.6653

Degrees of Freedom: 115 Total (i.e. Null);  113 Residual
Null Deviance:      159.1
Residual Deviance: 50.83    AIC: 56.83
```

Thus, $\hat{y}^* = 6.12 - 0.21x_1 - 0.67x_2$ (note that it is a different model than the one built on the **full dataset**).

Let Te be the $M = 50$ observations **not** in Tr; on it, we compute

$$\hat{p}_i = P(Y_i = \texttt{high} \mid X_1 = x_{1,i}, X_2 = x_{2,i}) = \frac{\exp(\hat{y}_i^*)}{1 + \exp(\hat{y}_i^*)}.$$

We obtain

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    0.00    0.36    1.00    1.00
```

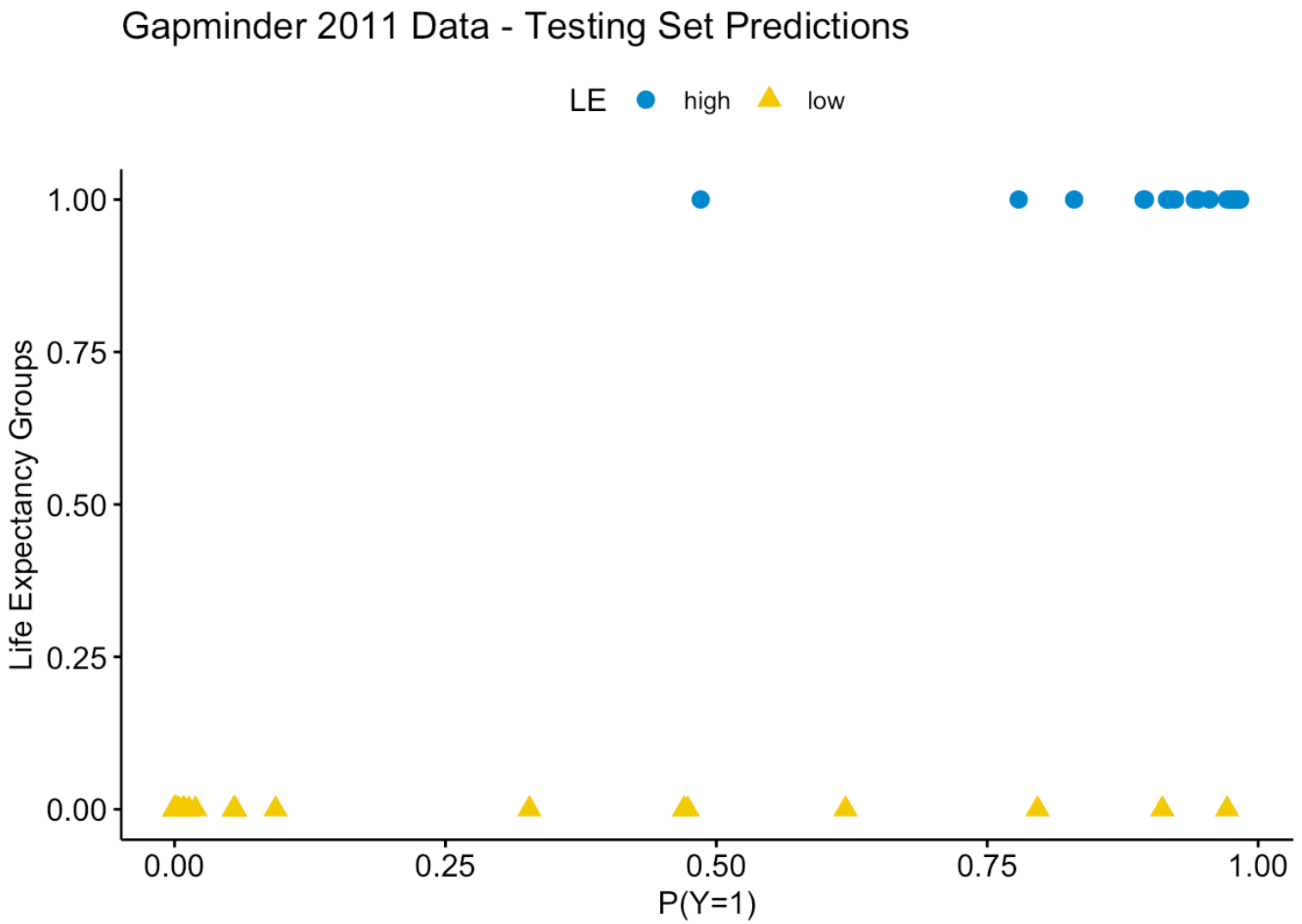
and

$$\mathrm{MSE}_{\mathrm{Te}} = \frac{1}{50}\sum_{i=1}^{50}(\hat{p}_i - \mathcal{I}[Y_i = \texttt{high}])^2 = 0.075.$$

Is this a **good** test error?

It is difficult to answer without more context.

Perhaps a more intuitive way to view the situation is to make **actual predictions** and to explore their **quality**.


Gapminder 2011 Data - Testing Set Predictions
LE
high
low
Life Expectancy Groups
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
P(Y=1)

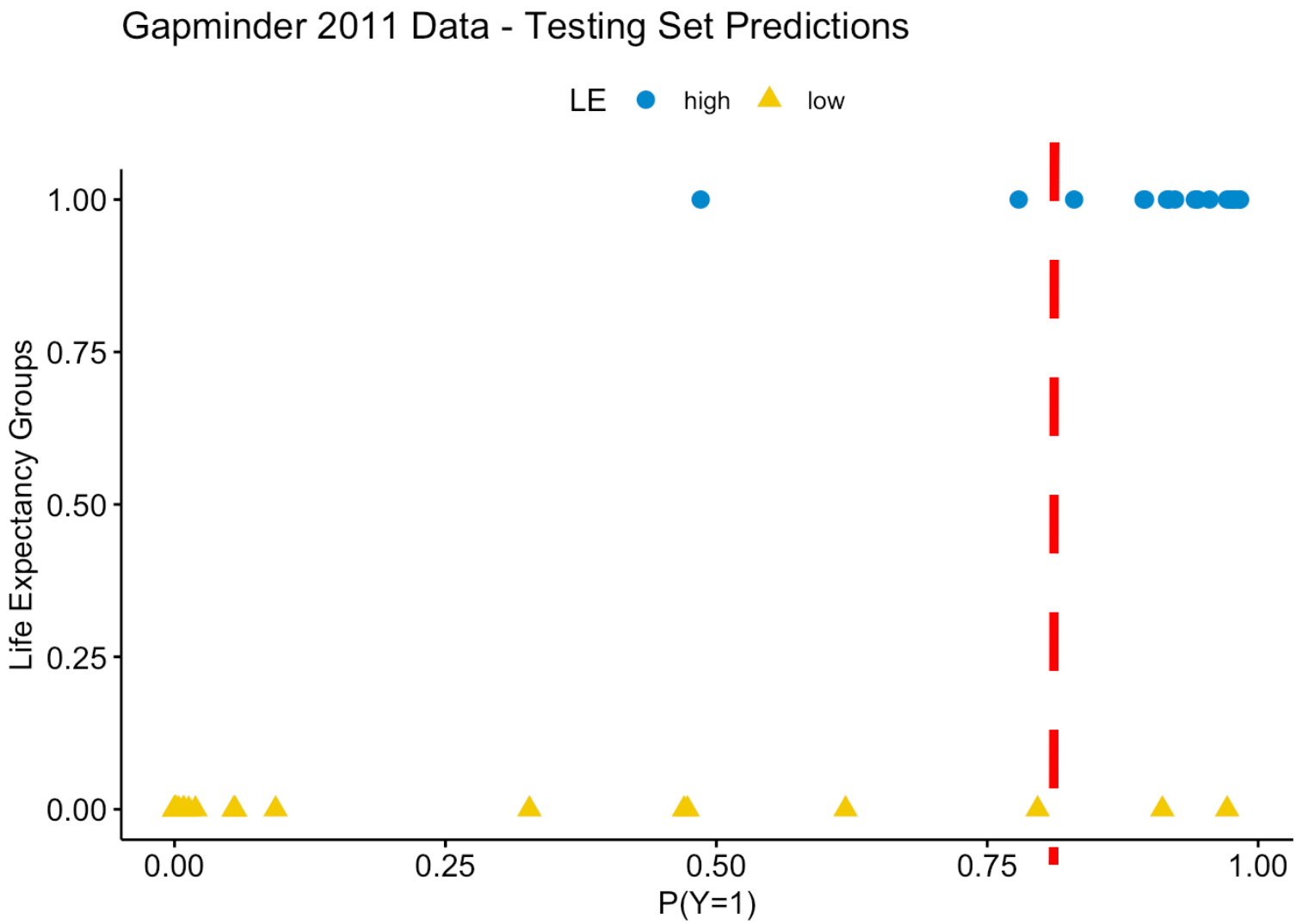
For $\alpha \in [0, 1]$, set

$$\text{pred}_i(\alpha) = \begin{cases} \texttt{high} & \text{if } \hat{p}_i > \alpha \\ \texttt{low} & \text{else} \end{cases}$$

In the specific version of Te used in this example, 36% of the nations had a `high` life expectancy.

If we set $\alpha = 0.81$, then the model predicts that 36% of the Te nations will have a `high` life expectancy; the **confusion matrix** on Te is shown below:

$\alpha = 0.81$		**prediction**	
		0	1
actual	0	30	2
	1	2	16


Gapminder 2011 Data - Testing Set Predictions
LE
high
low
Life Expectancy Groups
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
P(Y=1)

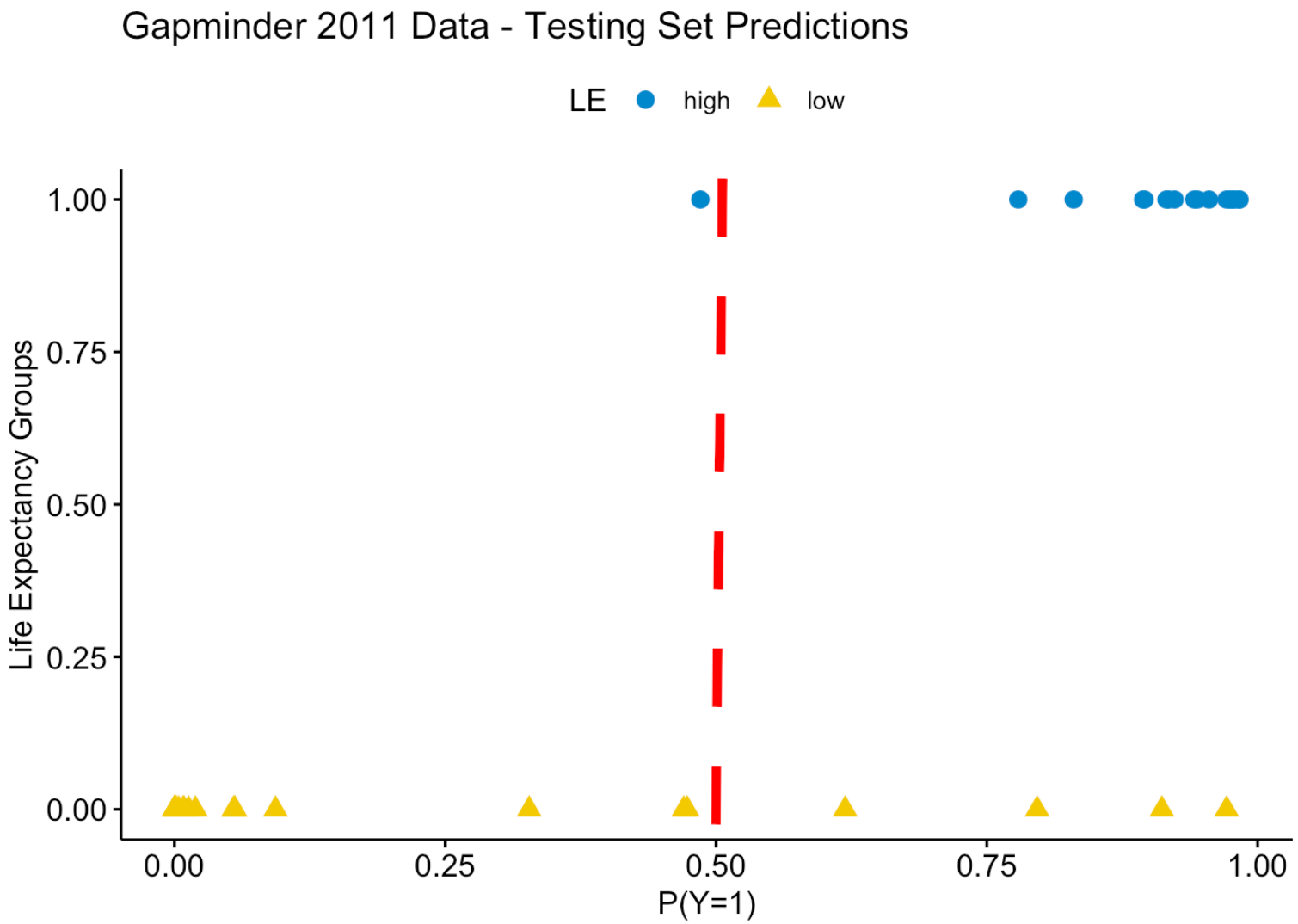
But why pick $\alpha = 0.81$ instead of $\alpha = 0.5$, say?

In a sense, the latter choice could prove the only rational choice in the absence of information.

In that case, 42% of nations are predicted to have `high` life expectancy, and the **confusion matrix** on Te is as below.

$\alpha = 0.50$		**prediction**	
		0	1
actual	0	28	4
	1	1	17

We will revisit this question shortly.


Gapminder 2011 Data - Testing Set Predictions
LE
high
low
Life Expectancy Groups
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
P(Y=1)

3.3 – Discriminant Analysis

In **LR**, we model $P(Y = C_k \mid \mathbf{x})$ directly *via* the **logistic function**

$$p_1(\mathbf{x}) = \frac{\exp(\mathbf{x}^\top \hat{\boldsymbol{\beta}})}{1 + \exp(\mathbf{x}^\top \hat{\boldsymbol{\beta}})}.$$

LR may be contra-indicated when there are $3+$ response levels, or when:

- the classes are **well-separated**, as the coefficient estimates may be **unstable** (adding a single point to Tr could change them substantially);
- Tr is **small** and the predictors are roughly **Gaussian** in each of the classes $Y = C_k$, as the coefficient estimates may also be **unstable**;

In **discriminant analysis (DA)**, we instead model

$$P(\mathbf{x} \mid Y = C_k),$$

the distribution of the **predictors** $\vec{X}$, conditional on the **response level** $Y = C_k$, and use Bayes' Theorem to obtain

$$P(Y = C_k \mid \mathbf{x}),$$

the probability of observing the **response** $Y = C_k$, conditional on $\vec{X} = \mathbf{x}$.

Let $\mathcal{C} = \{C_1, \ldots, C_K\}$ be the K response levels, $K \geq 2$; let π_k be the probability that a response lies in C_k, for $k \in \{1, \ldots, K\}$; π_k is the **prior**

$$\pi_k = P(Y = C_k) = \frac{|C_k|}{N}.$$

Let $f_k(\mathbf{x}) = P(\mathbf{x} \mid Y = C_k)$ be the **conditional density** of the distribution of $\vec{X}$ in C_k; we expect $f_k(\mathbf{x})$ to be **large** if there is a **high** probability that an observation in C_k has a predictor $\vec{X} \approx \mathbf{x}$, and **small** otherwise.

According to Bayes' Theorem, then

$$\begin{aligned} p_k(\mathbf{x}) = P(Y = C_k \mid \mathbf{x}) &= \frac{P(\mathbf{x} \mid Y = C_k) \cdot P(Y = C_k)}{P(\mathbf{x})} \\ &= \frac{P(\mathbf{x} \mid Y = C_k) \cdot P(Y = C_k)}{P(\mathbf{x} \mid Y = C_1) \cdot P(Y = C_1) + \cdots + P(\mathbf{x} \mid Y = C_K) \cdot P(Y = C_K)} \\ &= \frac{\pi_k f_k(\mathbf{x})}{\pi_1 f_1(\mathbf{x}) + \cdots + \pi_K f_K(\mathbf{x})}. \end{aligned}$$

For an observation $\mathbf{x} \in \mathrm{Te}$, the DA classifier is $\hat{C}_{\mathsf{DA}}(\mathbf{x}) = C_{\arg\max_j \{p_{j(\mathbf{x})}\}}$.

3.3.1 – Linear Discriminant Analysis

To say more, we need to make **additional assumptions** on the nature of the **underlying distributions**.

With only one predictor ($p = 1$), we make the **Gaussian assumption**,

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2\right],$$

where μ_k and σ_k are the mean and the standard deviation, resp., of the predictor for all observations in class C_k.

Any other predictor distribution could be used if **appropriate** for Tr; we could also assume that $\sigma_k \equiv \sigma$ or $\mu_k \equiv \mu$ across classes.

If we do assume that $\sigma_k \equiv \sigma$ for all k, then

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2\right]}{\pi_1 \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right] + \cdots + \pi_K \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_K}{\sigma}\right)^2\right]}$$

$$= \frac{\pi_k \exp\left[-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2\right]}{\pi_1 \exp\left[-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right] + \cdots + \pi_k \exp\left[-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2\right]}$$

$$p_k(x) = \frac{\pi_k \exp\left[\frac{\mu_k}{\sigma^2}\left(x - \frac{\mu_k}{2}\right)\right] \exp\left(-\frac{x^2}{2\sigma^2}\right)}{\left\{\pi_1 \exp\left[\frac{\mu_1}{\sigma^2}\left(x - \frac{\mu_1}{2}\right)\right] + \cdots + \pi_K \exp\left[\frac{\mu_K}{\sigma^2}\left(x - \frac{\mu_K}{2}\right)\right]\right\} \exp\left(-\frac{x^2}{2\sigma^2}\right)}$$

$$= \pi_k \exp\left[\frac{\mu_k}{\sigma^2}\left(x - \frac{\mu_k}{2}\right)\right] \cdot A(x).$$

If we are only interested in **classification**, we do not need to compute the actual probabilities $p_k(x)$ **directly**; the **discriminant score** for each class could prove more useful:

$$\delta_k(x) = \ln p_k(x) = \ln \pi_k + x\frac{\mu_k}{\sigma^2} - \frac{\mu_k}{2\sigma^2} + \ln A(x).$$

Since $\ln A(x)$ is the same for all k, we can **drop it** as it does not contribute to **relative differences** in class scores.

Given an observation $x \in \mathrm{Te}$, the **linear discriminant analysis** (LDA) classifier with $p = 1$ is $\hat{C}_{\mathsf{LDA}}(\mathbf{x}) = C_{\arg\max_j \{\hat{\delta}_{j(\mathbf{x})}\}}$.

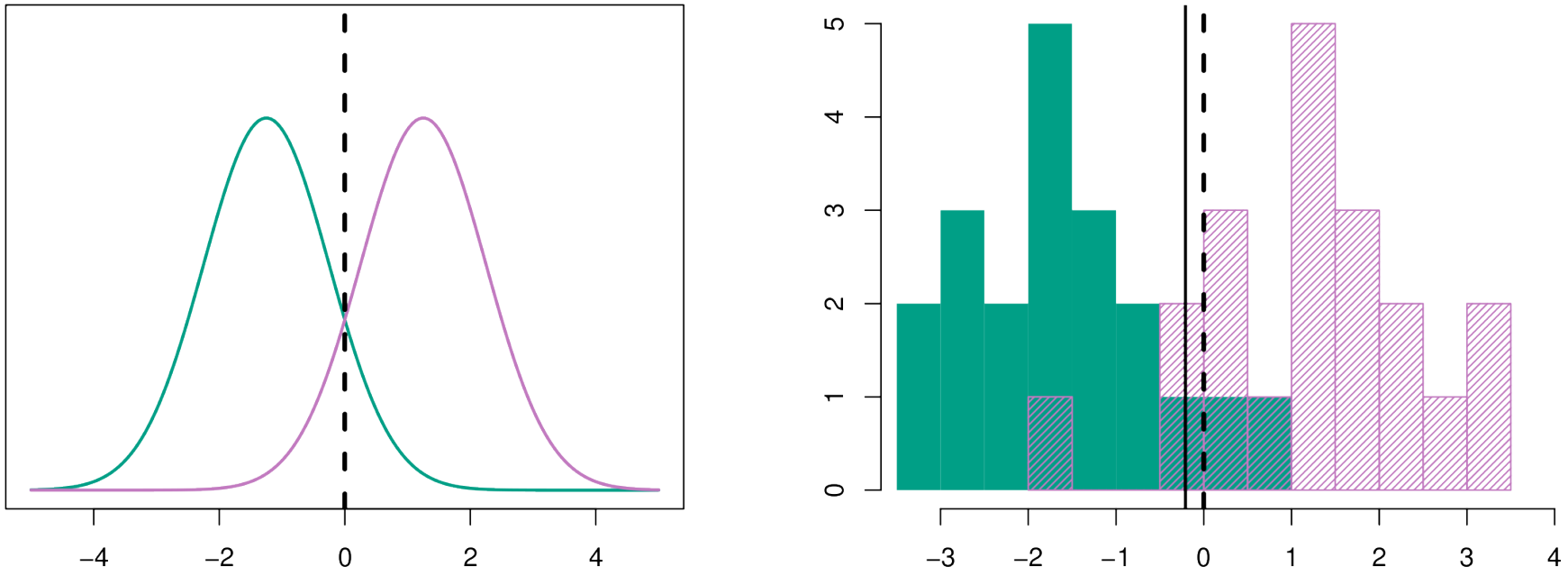
The "**linear**" in LDA comes from the linearity of the discriminant scores δ_k, after the $\ln A(x)$ term has been dropped.

If $K = 2$ and $\pi_1 = \pi_2 = 0.5$, the **midpoint** $x^* = \frac{1}{2}(\mu_1 + \mu_2)$ plays a crucial role. The discriminant scores $\delta_1(x)$ and $\delta_2(x)$ meet when

$$x^* \frac{\mu_1}{\sigma^2} - \frac{\mu_1^2}{2\sigma^2} = x^* \frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2} \implies x^* = \frac{\mu_1 + \mu_2}{2},$$

as long as $\mu_1 \neq \mu_2$. If $\mu_1 < \mu_2$, say, then the **decision rule** simplifies to

$$\hat{C}(x) = \begin{cases} C_1 & \text{if } x \leq x^* \\ C_2 & \text{if } x > x^* \end{cases}$$



Midpoint of two theoretical normal distributions (dashed line); midpoint of two empirical normal distributions (solid line). Observations to the left of the decision boundary are classified as green, those to the right as purple.

In practice, we estimate π_k, μ_k and σ from Tr:

$$\hat{\pi}_k = \frac{N_k}{N}, \quad \hat{\mu}_k = \frac{1}{N_k} \sum_{y_i \in C_k} x_i$$

$$\hat{\sigma}^2 = \sum_{k=1}^{K} \frac{N_k - 1}{N - K} \left(\frac{1}{N_k - 1} \sum_{y_i \in C_k} (x_i - \hat{\mu}_k)^2 \right).$$

If there are $p > 1$ predictors, we adapt the Gaussian assumption to $\mathbb{R}^p$:

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} \left|\boldsymbol{\Sigma}_k\right|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k), \right],$$

where $\boldsymbol{\mu}_k = (\overline{X_1}, \ldots, \overline{X_p})$, $\boldsymbol{\Sigma}_k(j,i) = \mathrm{Cov}(X_i, X_j)$ for all $\vec{X}$ with $Y = C_k$.

If we assume that $\sigma_k \equiv \boldsymbol{\Sigma}$ for all k, then the discriminant score is linear in $\mathbf{x}$:

$$\delta_{k;\text{LDA}}(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln \pi_k = c_{k,0} + \mathbf{c}_k^\top \mathbf{x}.$$

We can estimate $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}$ from the data, from which we recover

$$P(Y = C_k \mid \mathbf{x}) \approx \hat{p}_k(\mathbf{x}) = \frac{\exp(\hat{\delta}_{k;\text{LDA}}(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\hat{\delta}_{j;\text{LDA}}(\mathbf{x}))}.$$

The decision rule is as before: given an observation $\mathbf{x} \in \text{Te}$, the LDA classifier with $p > 1$ is $\hat{C}_{\text{LDA}} = C_{\arg\max_j \{\hat{\delta}_{j;\text{LDA}}(\mathbf{x})\}}$.

3.3.2 – Quadratic Discriminant Analysis

The assumption that the conditional probability functions be Gaussians with the **same covariance** in each training class may be a stretch.

If $\boldsymbol{\Sigma}_i \neq \boldsymbol{\Sigma}_j$ for at least one pair of classes $(i \neq j)$, then a similar process gives rise to **quadratic discriminant analysis** (QDA), which reduces to discriminant scores

$$\begin{aligned}\delta_{k;\text{QDA}}(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}) + \ln \pi_k \\ &= -\frac{1}{2}\mathbf{x}^\top \boldsymbol{\Sigma}_k^{-1}\mathbf{x} + \mathbf{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k + \ln \pi_k.\end{aligned}$$

To learn **LDA**, we must estimate $Kp + \frac{p(p+1)}{2}$ parameters from Tr: p parameters for each $\hat{\boldsymbol{\mu}}_k$ and $1 + \cdots + p$ parameters for $\hat{\boldsymbol{\Sigma}}$.

To learn **QDA**, we must estimate $K\left(p + \frac{p(p+1)}{2}\right)$ from Tr: p parameters for each $\hat{\boldsymbol{\mu}}_k$ and $1 + \cdots + p$ parameters for each $\hat{\boldsymbol{\Sigma}}_k$.

QDA is thus **more** complex (and so **more** flexible) than LDA.

The latter is recommended if Tr is **small**; the former if Tr is **large**. Either way, LDA will yield **high** bias if the $\boldsymbol{\Sigma}_k \equiv \boldsymbol{\Sigma}$ assumption is invalid.

Note that LDA gives rise to **linear separating hypersurfaces** and QDA to **quadratic** ones.

What about other types of assumptions?

Gaussian Naïve Bayes Classification

If we assume that each $\mathbf{\Sigma}_k$ is **diagonal** (i.e., that the features are **independent** in each class), we get the **Gaussian naïve Bayes classifier** (GNBC), with discriminant scores given by

$$\delta_{k;\text{GNBC}}(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{p}\frac{(x_j - \mu_{k,j})^2}{\sigma_{k,j}^2} + \ln \pi_k.$$

This approach can also be used for **mixed feature vectors**, by using **combinations of p.m.f. and p.d.f.** in $f_{k,j}(x_j)$, as required.

The assumption of independence is usually "**naïve**"; but GNBC can still prove useful when p is **too large**, and both LDA and QDA break down.

We will re-visit NBC shortly.

3.3.3 – Logistic Regression (Reprise)

We can also recast the $2-$class LDA model as

$$\ln\left(\frac{p_0(\mathbf{x})}{1-p_0(\mathbf{x})}\right) = \ln(p_0(\mathbf{x})) - \ln(p_1(\mathbf{x})) = \delta_0(\mathbf{x}) - \delta_1(\mathbf{x}) = a_0 + \mathbf{a}^\top \mathbf{x},$$

which has the same form as LR, **but it is not LR**:

- in **LR**, the parameters are estimated using the **maximum likelihood** $P(Y \mid \mathbf{x})$;
- in **LDA**, the parameters are estimated using the **full likelihood**

$$P(\mathbf{x} \mid Y)P(\mathbf{x}) = P(\mathbf{x}, Y).$$

3.3.4 – Example: Gapminder Dataset

Using the `Gapminder` data with the same Tr ($N = 116$) and Te ($M = 50$) from Section 3.2.3, we obtain the following LDA and QDA models (DUDADS, 21.2.2, *Discriminant Analysis*).

On the training set Tr, we find:

$$N_0 = 51,\ N_1 = 65,\ \hat{\pi}_0 = 51/116,\ \hat{\pi}_1 = 65/116,$$

$$\hat{\boldsymbol{\mu}}_0 = (45.40, 4.08)^\top,\ \hat{\boldsymbol{\mu}}_1 = (9.57, 1.92)^\top$$

$$\boldsymbol{\Sigma}_0 = \begin{pmatrix} 496.51 & 23.38 \\ 23.38 & 2.17 \end{pmatrix},\ \boldsymbol{\Sigma}_1 = \begin{pmatrix} 42.79 & 2.14 \\ 2.14 & 0.31 \end{pmatrix}$$

$$\hat{\boldsymbol{\mu}} = (25.30, 2.87)^\top,\ \boldsymbol{\Sigma} = \begin{pmatrix} 557.89 & 30.51 \\ 30.51 & 2.27 \end{pmatrix}$$

After **inverting** the covariance matrices, we get

$$\hat{\delta}_{0;\text{LDA}} = -4.78 - 0.06x_1 + 2.65x_2$$
$$\hat{\delta}_{1;\text{LDA}} = -2.28 - 0.11x_1 + 2.31x_2$$
$$\hat{\delta}_{0;\text{QDA}} = -4.66 - 0.002x_1^2 + 0.01x_1 + 0.04x_1x_2 + 1.81x_2 - 0.47x_2^2$$
$$\hat{\delta}_{1;\text{QDA}} = -6.70 - 0.02x_1^2 - 0.13x_1 + 0.24x_1x_2 + 7.01x_2 - 2.43x_2^2$$

The **class probability estimates** are thus

$$\hat{p}_{1;\text{LDA}} = \frac{\exp(\hat{\delta}_{1;\text{LDA}})}{\exp(\hat{\delta}_{0;\text{LDA}}) + \exp(\hat{\delta}_{1;\text{LDA}})},$$
$$\hat{p}_{1,\text{QDA}} = \frac{\exp(\hat{\delta}_{1;\text{QDA}})}{\exp(\hat{\delta}_{0;\text{QDA}}) + \exp(\hat{\delta}_{1;\text{QDA}})}$$

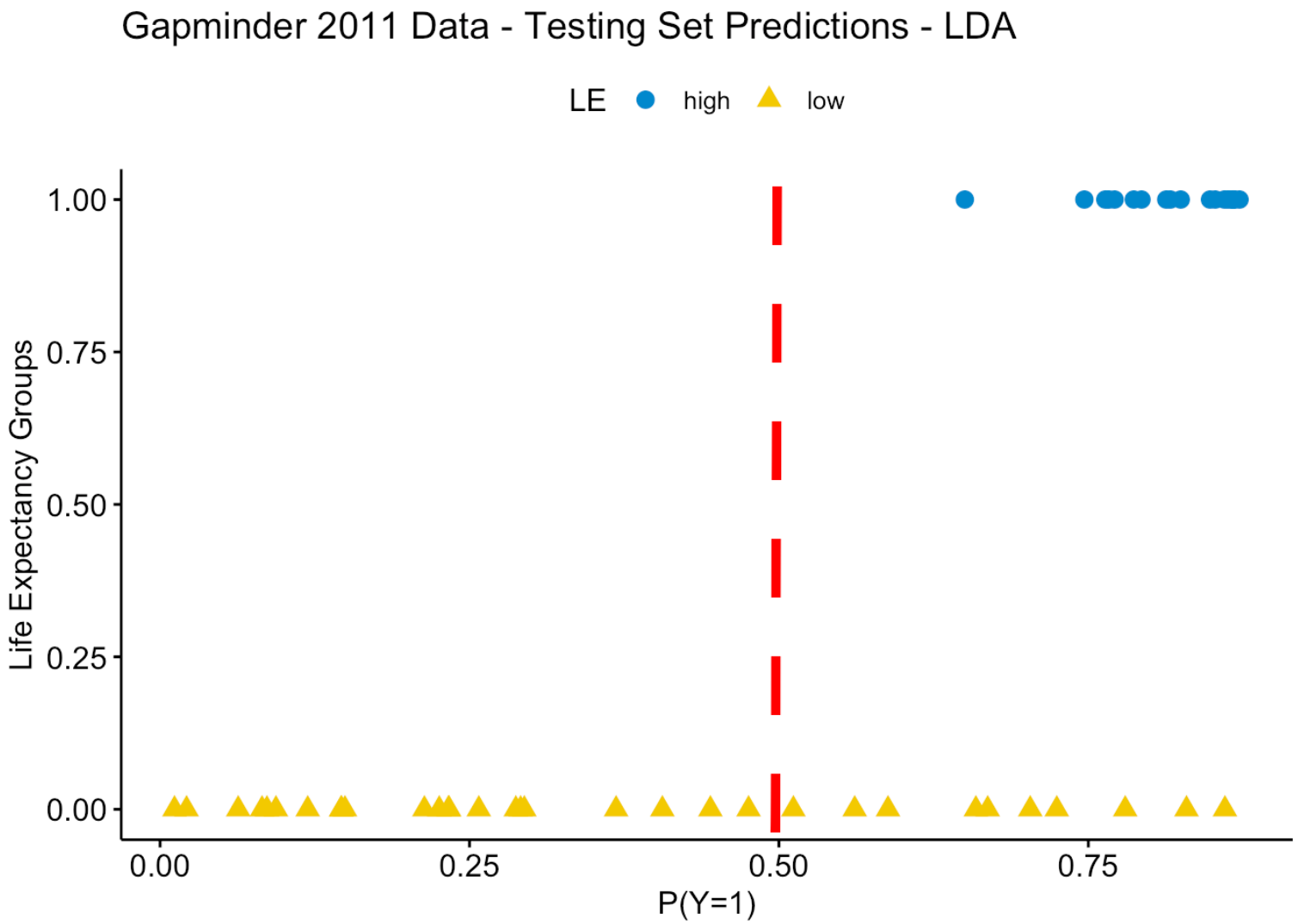
Given an observation $\mathbf{x} \in \mathrm{Te}$, we use a **decision rule** based on the probabilities $\hat{p}_0(\mathbf{x}), \hat{p}_1(\mathbf{x})$ and a **decision threshold** $\alpha \in (0,1)$.

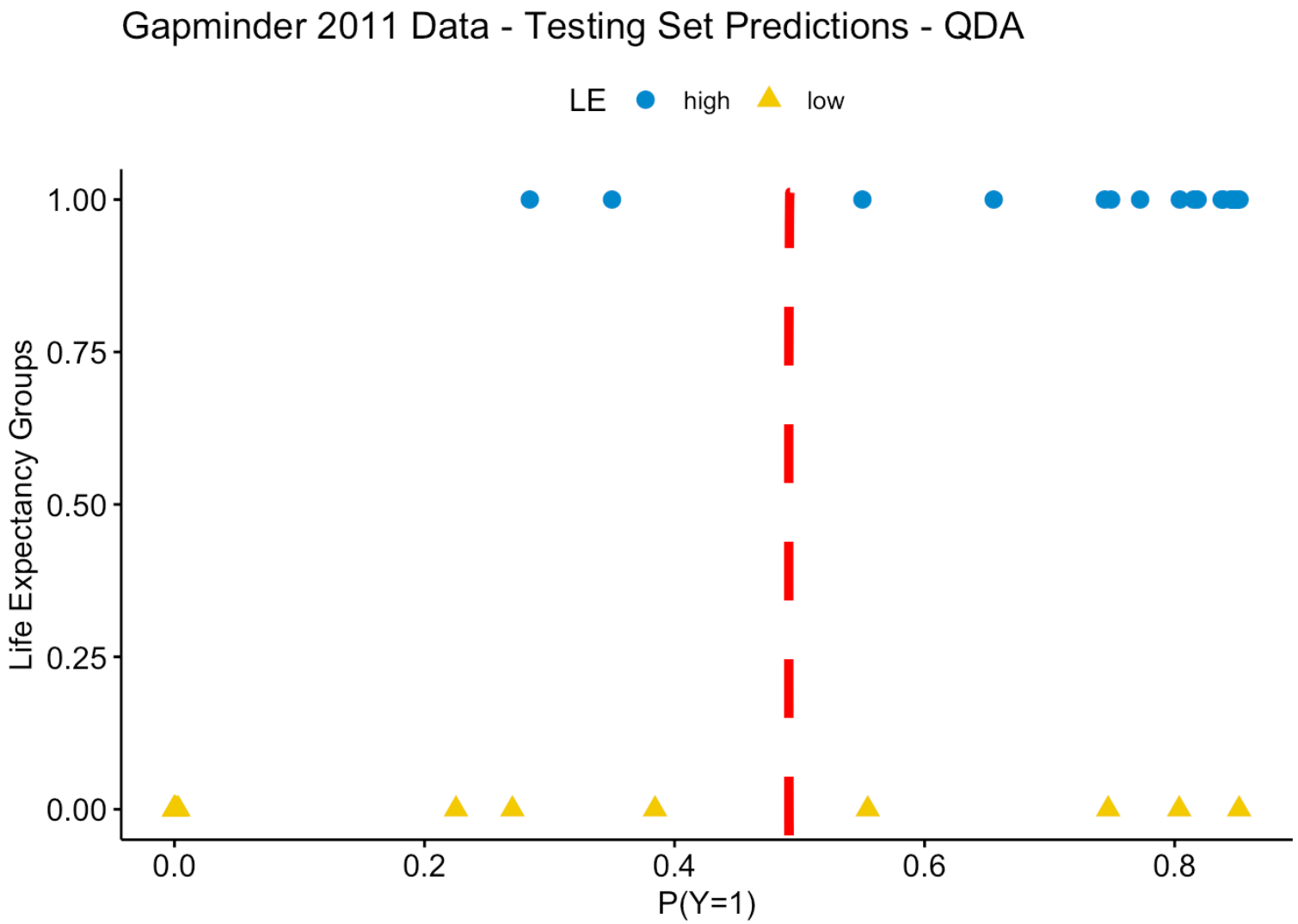
If the decision threshold set at $\alpha = 0.5$, the LDA and QDA `LE` classifiers are defined on Te by

$$\hat{C}_{\alpha;\mathsf{LDA}}(\mathbf{x}) = \begin{cases} 1 \ (\texttt{high}) & \text{if } p_{1;\mathsf{LDA}}(\mathbf{x}) \geq 0.5 \\ 0 \ (\texttt{low}) & \text{else} \end{cases}$$

$$\hat{C}_{\alpha;\mathsf{QDA}}(\mathbf{x}) = \begin{cases} 1 \ (\texttt{high}) & \text{if } p_{1;\mathsf{QDA}}(\mathbf{x}) \geq 0.5 \\ 0 \ (\texttt{low}) & \text{else} \end{cases}$$

The predictions are displayed on the next pages.


Gapminder 2011 Data - Testing Set Predictions - LDA
LE
high
low
Life Expectancy Groups
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
P(Y=1)


Gapminder 2011 Data - Testing Set Predictions - QDA
LE
high
low
1.00
0.75
0.50
0.25
0.00
Life Expectancy Groups
0.0
0.2
0.4
0.6
0.8
P(Y=1)

The $\alpha = 0.5$ confusion matrices for the LDA and QDA classifiers on Te are:

LDA $\alpha = 0.5$		**prediction** 0	1	**QDA** $\alpha = 0.5$		**prediction** 0	1
actual	0	22	10	**actual**	0	28	4
	1	0	18		1	2	16

In the LDA case, we see that:

- **accuracy** $= \frac{22+18}{22+10+0+18} = 80\%$
- **misclassification rate** $= \frac{10+0}{22+10+0+18} = 20\%$
- $\mathrm{FPR} = \frac{0}{0+18} = 0\%$
- $\mathrm{FNR} = \frac{10}{22+10} = 31.25\%$
- $\mathrm{TPR} = \frac{22}{22+10} = 68.75\%$
- $\mathrm{TNR} = \frac{18}{0+18} = 100\%$

In the QDA case, we see that:

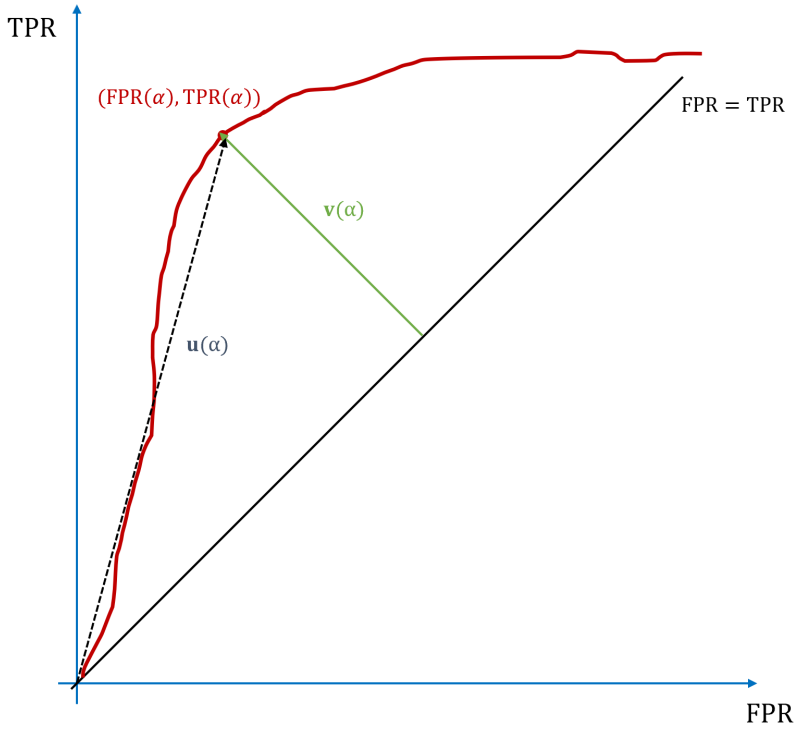
- **accuracy** $= \frac{28+16}{28+4+2+16} = 88\%$
- **misclassification rate** $= \frac{4+2}{28+4+2+16} = 12\%$
- $\mathrm{FPR} = \frac{2}{2+16} = 11.1\%$
- $\mathrm{FNR} = \frac{4}{28+4} = 12.5\%$
- $\mathrm{TPR} = \frac{28}{28+4} = 87.5\%$
- $\mathrm{TNR} = \frac{16}{2+16} = 88.9\%$

At first glance, it would certainly seem that the QDA model performs **better** (at a decision threshold of $\alpha = 0.5$), but its FPR is **not ideal**.

What would be the ideal value of α?

How would we find it?

3.4 – ROC Curve



The **receiver operating characteristic** (ROC) curve plots the TPR against the FPR for classifiers by varying the **decision threshold** α in $[0, 1]$.

The important realization is that a classifier that is **completely random** would lie on the line $\mathrm{TPR} = \mathrm{FPR}$. Thus, the **ideal threshold** α is the one associated with the model which is **farthest** from that line.

Let $\mathbf{u}(\alpha)$ join $\mathbf{0}$ to the coordinates $(\mathrm{FPR}(\alpha), \mathrm{TPR}(\alpha))$ of the classifier with threshold α, and let $\mathbf{v}(\alpha)$ be the vector through $(\mathrm{FPR}(\alpha), \mathrm{TPR}(\alpha))$ and **perpendicular** to the line $\mathrm{TPR} = \mathrm{FPR}$. The **ideal** α^* satisfies:

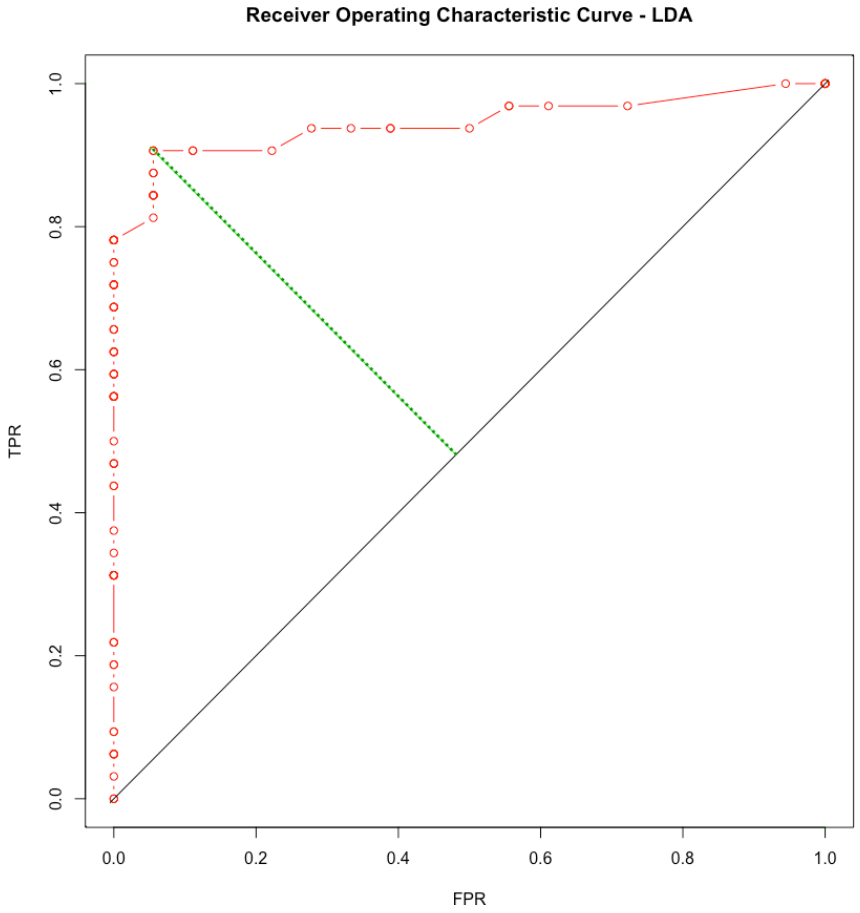
$$\begin{aligned}
\alpha^* &= \arg\max_{\alpha}\{\|\mathbf{v}(\alpha)\|\} = \arg\max_{\alpha}\{\|\mathbf{v}(\alpha)\|^2\} \\
&= \arg\max_{\alpha}\left\{\left\|\mathbf{u}(\alpha) - \mathrm{proj}_{(1,1)}\mathbf{u}(\alpha)\right\|^2\right\} \\
&= \arg\max_{\alpha}\left\{\left\|(\mathrm{FPR}(\alpha), \mathrm{TPR}(\alpha)) - \mathrm{proj}_{(1,1)}(\mathrm{FPR}(\alpha), \mathrm{TPR}(\alpha))\right\|^2\right\} \\
&= \arg\max_{\alpha}\left\{\|(\mathrm{FPR}(\alpha) - \mathrm{TPR}(\alpha), \mathrm{TPR}(\alpha) - \mathrm{FPR}(\alpha))\|^2\right\} \\
&= \arg\max_{\alpha}\{(\mathrm{FPR}(\alpha) - \mathrm{TPR}(\alpha))^2\}.
\end{aligned}$$

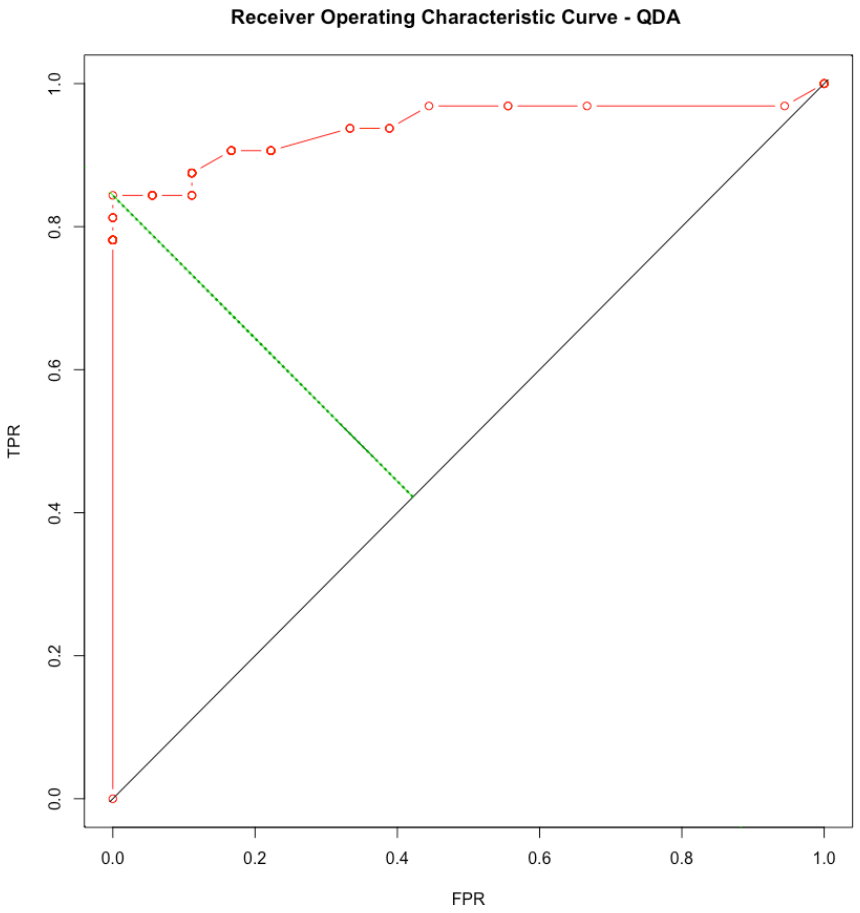
Example

For the LDA and QDA classifiers built in Section 3.3.4, the false positive rates (FPR), false negative rates (FNR), true positive rates (TPR), true negative rates (TNR), and **misclassification rates** for decision threshold α varying from 0.01 to 0.99 by steps of size 0.01 are computed in DUDADS (21.2.3, $ROC\ Curve$).

In both frameworks, a number of models have identical $(\mathrm{FPR}, \mathrm{TPR})$ coordinates.

With the LDA model, the ideal threshold is $\alpha^*_{\mathrm{LDA}} = 0.73$ (coordinates $(0.056, 0.906)$); with the QDA model, the ideal threshold is $\alpha^*_{\mathrm{QDA}} = 0.28$ (coordinates $(0, 0.844)$).


Receiver Operating Characteristic Curve - LDA
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic Curve - QDA
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

The corresponding confusion matrices are shown below.

LDA $\alpha^*_{\text{LDA}} = 0.73$		**prediction** 0	1	**QDA** $\alpha^*_{\text{QDA}} = 0.28$		**prediction** 0	1
actual	0	29	3	**actual**	0	27	5
	1	1	17		1	0	18

- **accuracy** $= 92\%$
- **misclassification rate** $= 8\%$
- FPR $= 5.6\%$
- FNR $= 9.4\%$
- TPR $= 90.6\%$
- TNR $= 94.4\%$

- **accuracy** $= 90\%$
- **misclassification rate** $= 10\%$
- FPR $= 0\%$
- FNR $= 15.6\%$
- TPR $= 84.4\%$
- TNR $= 100\%$

Which model is **best**? It depends on the context of the task, and on the consequences of the choice. What makes the most sense here?

- Is there a danger of **overfitting**?
- Is **parameter tuning** acceptable, from a data massaging perspective?
- What effect does the **choice of priors** have?

This procedure helps us find an **optimal** threshold α, but there is another aspect of the ROC curve that may be of interest: in general, the larger the **area under the ROC curve** is, the more likely the model is to "behave" for **non-optimal** decision thresholds. The metric is known as **ROC AUC**.

Technically, it takes on values between 0 and 1, but we since a classifier that is wrong **more often than expected** indirectly provides a classifier that is right **more often than expected**, we can focus instead on the area between the curve and the line $\mathrm{TPR} = \mathrm{FPR}$.

3.5 – Rare Occurrences

We briefly touch on the problem of **rare ocurrences** (**unbalanced** dataset).

Scenario: we are trying to detect fraudulent transactions. We can build **classifiers** to approach this task using any number of methods ... but there is a potential problem.

If $(100-\varepsilon)\%$ of observations belong to the **normal** category, and $\varepsilon\%$ to the **special** category, the model that predicts that **EVERY observation** is normal has $(100-\varepsilon)\%$ accuracy.

In practice, the **vast majority** of transactions are legitimate, so that $0 < \varepsilon \ll 1$; the above model has **almost-perfect** accuracy, but **it misses the point of the exercise entirely**.

Solution: either we **modify the algorithms** to take into account the **asymmetric cost** of making a classification error (through so-called **cost-sensitive classifiers** or **one-class models** – we will not discuss those further), or we **modify** Tr to take into account the **imbalance in the data**.

In the latter approach, we could try to obtain **more training data**: the simplest solution is not always feasible ($\$\$$, **no data left to collect**).

Even when feasible, there is no guarantee that the new training data **follows the same pattern** as the original Tr $\Longrightarrow$ back to square one.

An alternative is to create a **new** Tr by **undersampling** the majority classes or **oversampling** the under-represented classes.

We assume for now that there are only **two** classes in the data (the strategies can easily be adapted to **multi-class problems**).

Undersampling

Write $\mathrm{Tr} = L \sqcup M_c$, where:

- L consists of all observations in the **majority** case, and
- M_c of all observations in the **minority** case.

By assumption, $|L| \gg |M_c|$ and $L \cap M_c \cap \varnothing$.

Split L into K subsets $L_1, \ldots, L_K$, each roughly of the **same size**.

K should be selected so that $|L_i| \not\gg |M_c|$, for all i – i.e., $|L_i|$ could be larger or smaller than $|M_c|$, but not substantially so $\implies$ **balanced datasets**.

We then construct K training sets

$$\mathrm{Tr}_1 = L_1 \sqcup M_c, \ \ldots, \ \mathrm{Tr}_K = L_K \sqcup M_c.$$

For each $1 \leq i \leq K$, we train a **classifier** C_i on Tr_i (ideally using the same algorithm for all Tr_i, although that is not strictly necessary).

Finally, we combine the predictions using **bagging** or other **ensemble learning** methods (see Section 3.7).

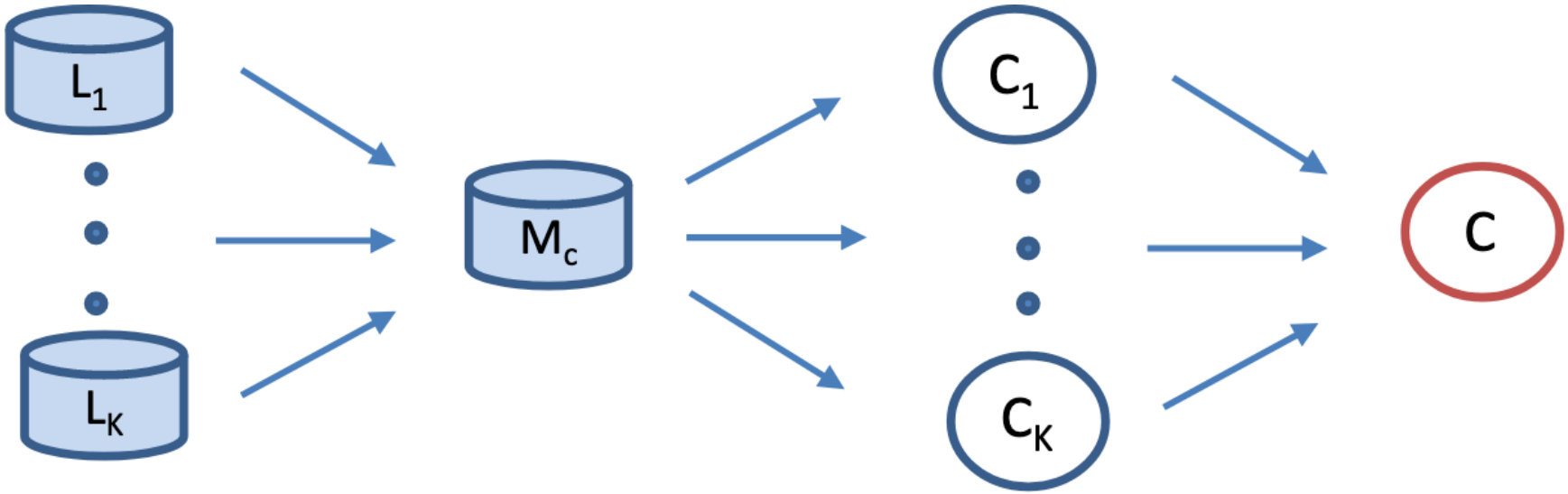


Illustration of undersampling

Oversampling

Alternatively, we can **oversample** the minority cases M_c to create **balanced datasets**, but that introduces dependency in the data, which can impact the method's **bias** and **variability**.

Synthetic Minority Oversampling Technique (SMOTE) is a common approach which creates "**synthetic**" examples rather than oversampling **with replacement** – the same idea is used to create samples for handwriting recognition by perturbing training data (e.g., rotating, skewing, etc.):

1. select **random** integers $k \ll \ell$;

2. draw a **random sample** $\mathcal{V}_\ell$ of size ℓ from the **minority class** M_c;

3. for each $\mathbf{x} \in \mathcal{V}_\ell$, find the k **NN** of $\mathbf{x}$ in M_c, say $\mathbf{z}_{\mathbf{x},1}, \ldots, \mathbf{z}_{\mathbf{x},k}$;

4. compute the vectors $\mathbf{v}_{\mathbf{x},1}, \ldots, \mathbf{v}_{\mathbf{x},k}$, **originating from** $\mathbf{x}$ and **ending at** each of the $\mathbf{z}_{\mathbf{x},1}, \ldots, \mathbf{z}_{\mathbf{x},k}$;

5. draw **random** values $\gamma_1, \ldots, \gamma_k \sim \mathcal{U}(0,1)$, and multiply $\mathbf{v}_{\mathbf{x},i}$ by γ_i, for each $1 \leq i \leq k$;

6. the points found at $\mathbf{x} + \gamma_i \mathbf{v}_{\mathbf{x},i}$, $1 \leq i \leq k$, are added to the set M_c.

This procedure is repeated until $|L| \not\gg |M_c|$.

There are variants, where we always use the same $k, \ell, \mathcal{V}_\ell$, or where we only pick **one** of the k nearest neighbours, etc. In general, SMOTE increases **recall**, but it comes at the cost of lower **precision** (look those up!).

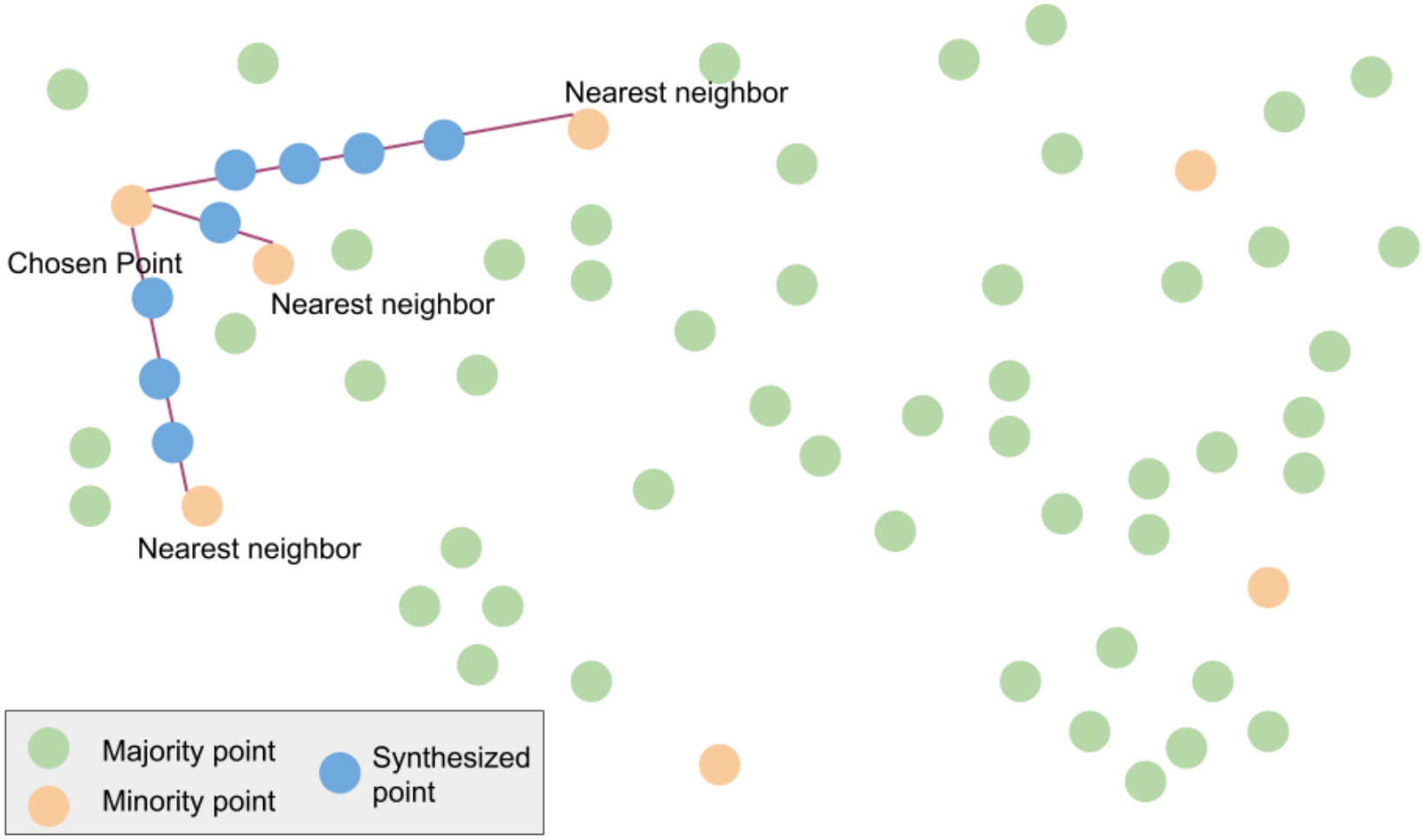
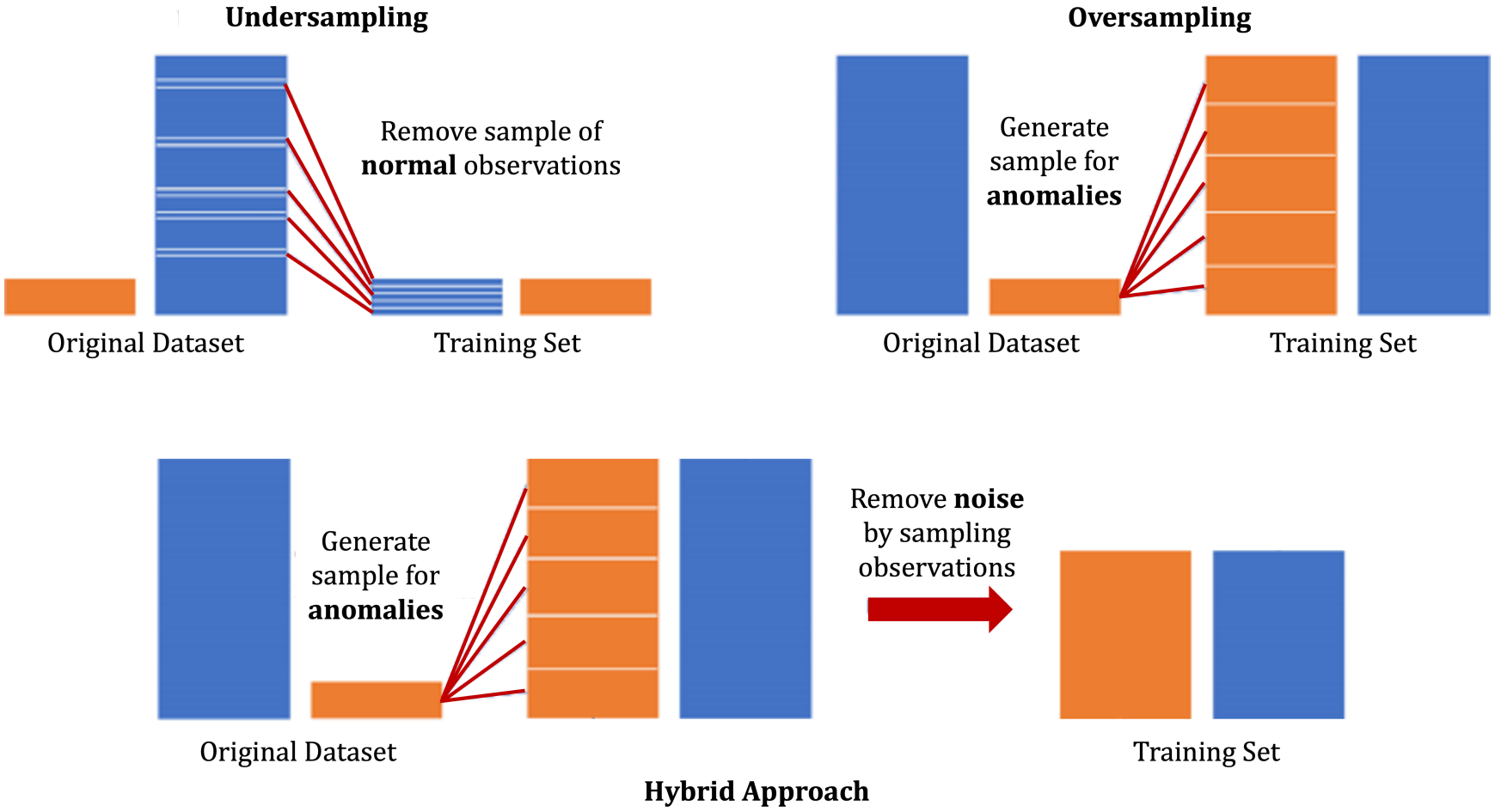


Illustration of oversampling (SMOTE)


Undersampling
Remove sample of
normal observations
Original Dataset
Training Set
Oversampling
Generate
sample for
anomalies
Original Dataset
Training Set
Generate
sample for
anomalies
Original Dataset
Remove noise
by sampling
observations
Training Set
Hybrid Approach

3.6 – Tree-Based Methods

Tree-based methods (TBM) **stratify** (or **segment**) the predictor space into a small number of **"simple" regions**.

The **splitting rules** can be summarized using a **tree**.

TBM are **simple** and **easy to interpret**, but they lag behind the best SL methods in terms of **predictive accuracy** (although, see **boosting** and **random forests**).

At times, however, the **ease of interpretability** is more crucial than the **lessened accuracy**.

TBM are applicable to both **regression** and **classification** problems.

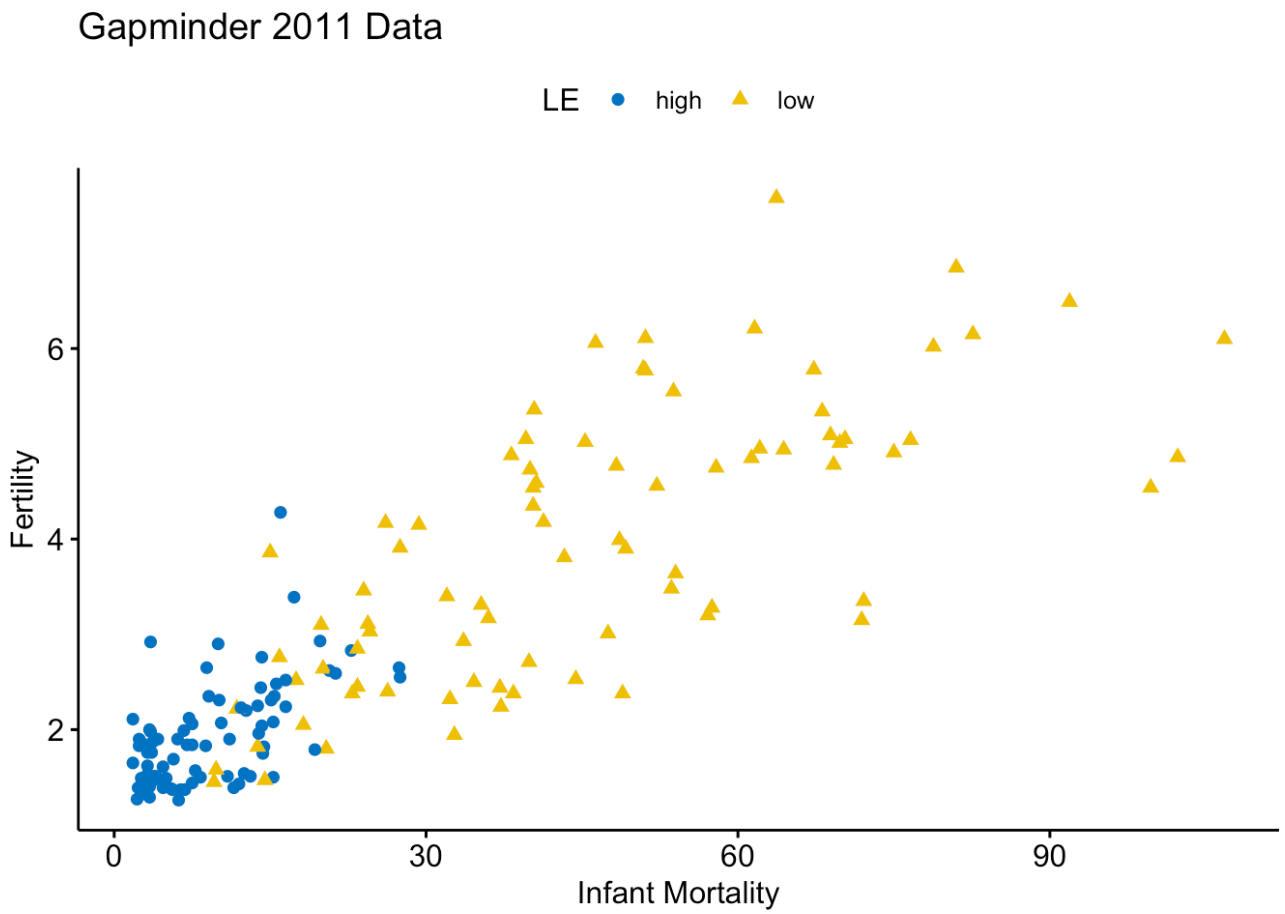
3.6.1 – Regression Trees

Consider once again the 2011 Gapminder dataset:

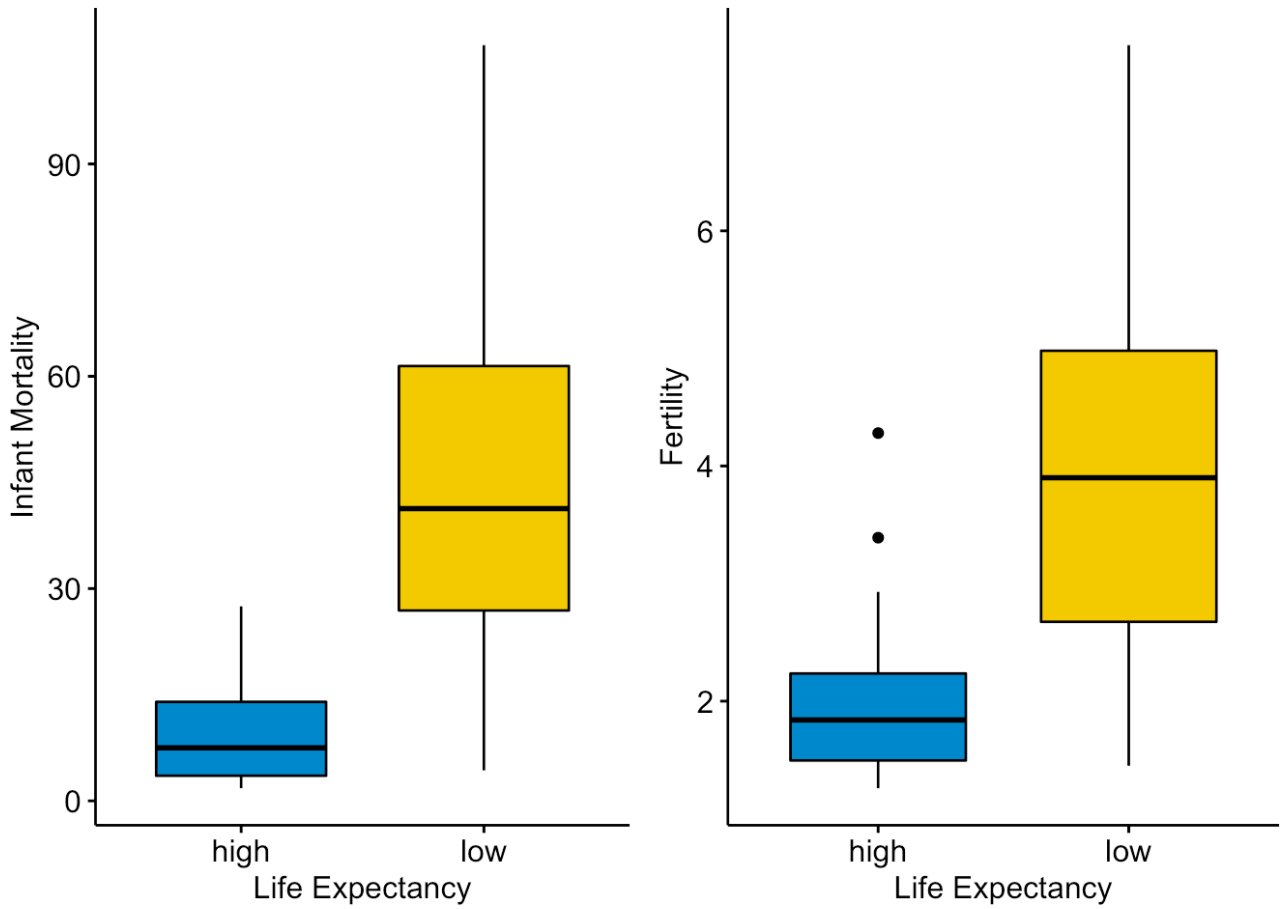
- the **response** Y is the life expectancy of nations in 2011;
- the **predictors** X_1 and X_2 are the 2011 fertility rates and infant mortality rates per nation.

When X_1, X_2 are **both high**, Y is **low** (**below** 72.45); when X_1, X_2 are **both low**, Y is **high** (**above** 72.45). What is the pattern "**in the middle**"?

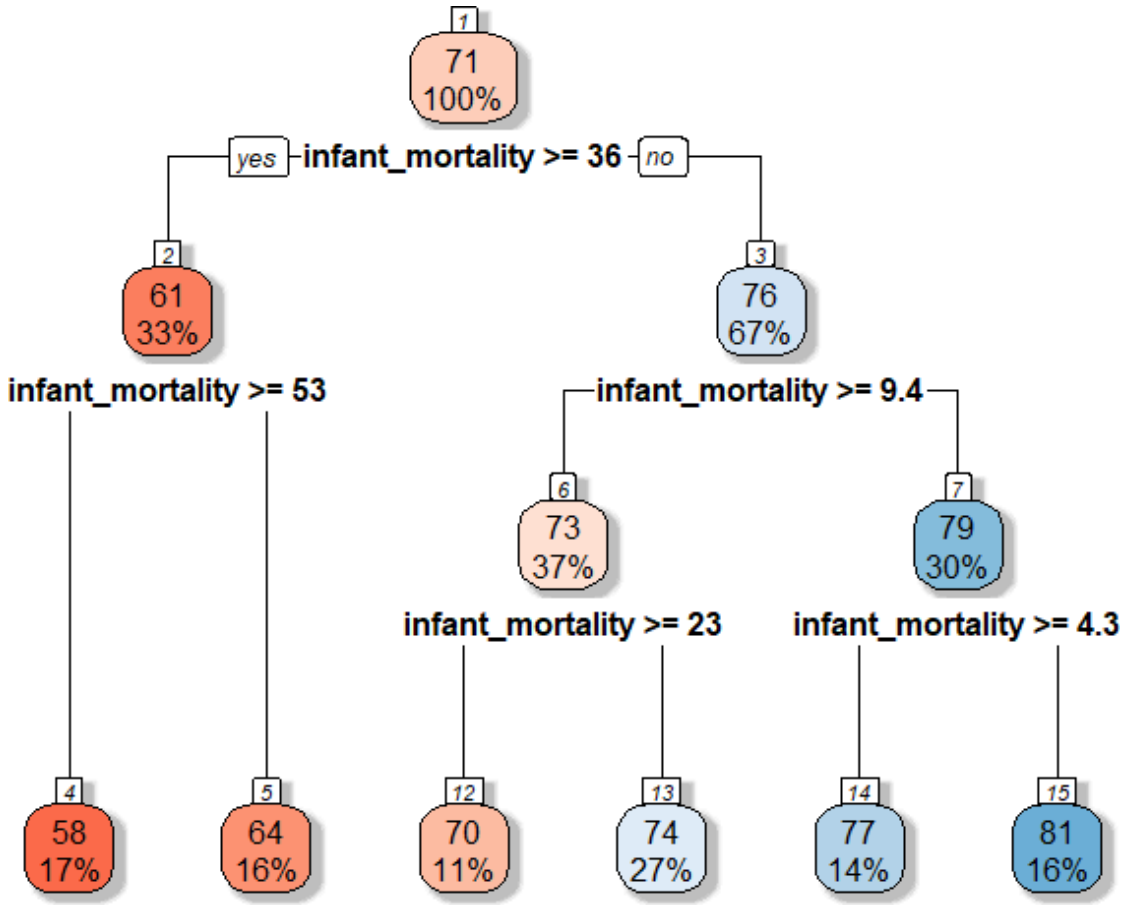
We build a **regression tree** to find out.



2011 Gapminder dataset



Predictors vs. response in the 2011 Gapminder dataset



Possible regression tree for the (full) dataset ($N = 166$ observations)

The **tree** can also be displayed as:

```
1) root (166) 70.82349
   2) infant_mortality>=35.65 (54) 60.85370
     4) infant_mortality>=52.9 (28) 58.30714 *
     5) infant_mortality< 52.9 (26) 63.59615 *
   3) infant_mortality< 35.65 (112) 75.63036
     6) infant_mortality>=9.35 (62) 72.89516
      12) infant_mortality>=22.85 (18) 69.50000 *
      13) infant_mortality< 22.85 (44) 74.28409 *
     7) infant_mortality< 9.35 (50) 79.02200
      14) infant_mortality>=4.25 (23) 76.86087 *
      15) infant_mortality< 4.25 (27) 80.86296 *
```

How do we read these R outputs?

Node 1 is the tree's **root** with 166 observations (100% of the dataset); the average life expectancy for these observations is $70.82 \approx 71$.

The **root** (**initial node**) is also the tree's first **branching point**, separating the observations into **two groups** (**sub-trees**):

- node 2 with 54 observations (33%), with "**infant mortality** ≥ 35.65", for which the average life expectancy is 60.85, and
- node 3 with 112 observations (67%), with "**infant mortality** < 35.65", for which the average life expectancy is 75.63.

Note that $54 + 112 = 166$ and that

$$\frac{54(60.81) + 112(75.63)}{54 + 112} = 70.82.$$

Node 2 is a **branching (internal) node**; it is further split into **two groups** (**sub-sub-trees?**):

- node 4 with 28 observations (17%), with "**infant mortality** ≥ 52.9", for which the average life expectancy is 58.31, and
- node 5 with 26 observations (16%), with "**infant mortality** < 52.9", for which the average life expectancy is 63.30.

Note that $28 + 26 = 54$ and that

$$\frac{28(58.31) + 26(63.60)}{28 + 26} = 60.85.$$

Both nodes 4 and 5 are **leaves** (**final nodes**, **terminal nodes**); the tree does not grow any further on that branch.

The tree **also** continues to grow from **branching** node 3, leading to 4 **leaves** on that branch.

In total:

- 6 **leaves**;
- 5 **branching nodes** (including the **root**);
- tree **depth** $= 3$ (excluding the **root**).

In this example, only 1 predictor is needed: infant mortality.

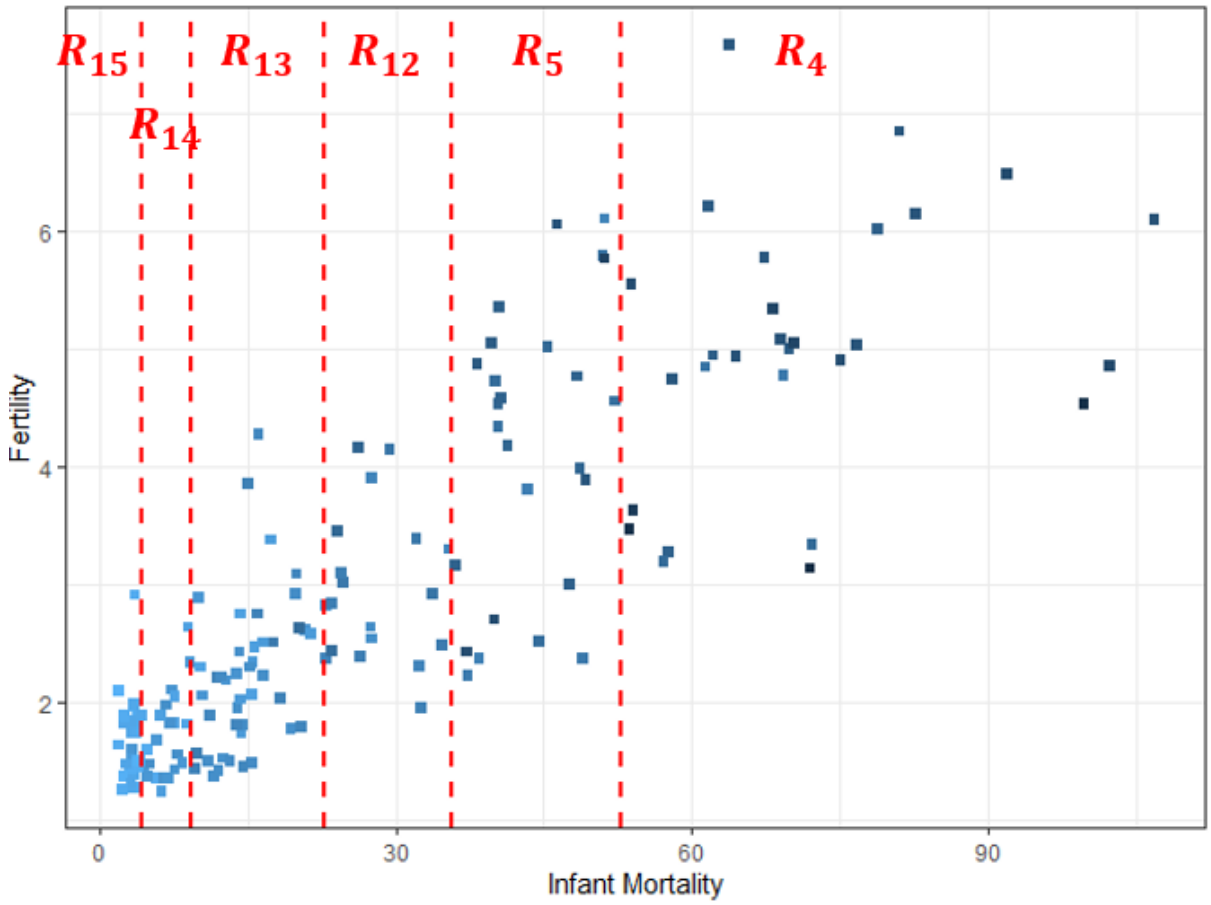
new obs.: **infant mortality** $= 21 \implies$ new obs. lands in node 13
predicted life expectancy: 74.28.

Tree diagrams are useful heuristics (no need for multi-dimensional charts), but they obscure **predictor space stratification**.

Write:

$$R_4 = \{(\text{infant mortality}, \text{fertility}) \mid \text{infant mortality} \geq 52.9\}$$
$$R_5 = \{(\text{infant mortality}, \text{fertility}) \mid 36.65 \leq \text{infant mortality} < 52.9\}$$
$$R_{12} = \{(\text{infant mortality}, \text{fertility}) \mid 22.85 \leq \text{infant mortality} < 35.65\}$$
$$R_{13} = \{(\text{infant mortality}, \text{fertility}) \mid 9.35 \leq \text{infant mortality} < 22.85\}$$
$$R_{14} = \{(\text{infant mortality}, \text{fertility}) \mid 4.25 \leq \text{infant mortality} < 9.35\}$$
$$R_{15} = \{(\text{infant mortality}, \text{fertility}) \mid \text{infant mortality} < 4.25\}$$

Since only infant mortality is involved in the definition of the tree's **terminal nodes**, the regions are **vertical strips**.



Stratification of the predictor space for the 2011 Gapminder dataset

The regression tree **model** for life expectancy in the dataset is thus:

$$\hat{y}_i = f(\mathbf{x}_i) = \text{Avg}\{y \mid \mathbf{x} \in R_{j(i)}\} = \begin{cases} 58.3, & j(i) = 4 \\ 63.6, & j(i) = 5 \\ 69.5, & j(i) = 12 \\ 74.3, & j(i) = 13 \\ 76.9, & j(i) = 14 \\ 80.9, & j(i) = 15 \end{cases}$$

where $R_{j(i)}$ is the region in which $\mathbf{x}_i \in \text{Tr}$ falls.

This is not the only way to stratify the data: how is the tree **optimal?** Why is it not using the second predictor (fertility)?

Building A Regression Tree

The process is quite simple:

1. divide the predictor space $\mathcal{X} \subseteq \mathbb{R}^p$ into a **disjoint union** of J **regions**: $\mathcal{X} = R_1 \sqcup \cdots \sqcup R_J$;

2. for any $\mathbf{x} \in R_j$, $\hat{y}(\mathbf{x}) = \mathrm{Avg}\{y(\mathbf{z}) \mid \mathbf{z} \in R_j \cap \mathrm{Tr}\}$.

In Step 1, we use **hyperboxes** with affine boundaries perpendicular/parallel to the p predictor hyperplanes (ditto).

Step 2 says that regression trees are **locally constant** (easier to manage).

Objective: how do we find the **optimal partition** $\mathcal{R}$ of $\mathcal{X}$?

Solution: the optimal partition $\mathcal{R} = (R_1, \ldots, R_J)$ **minimizes**

$$\mathrm{SSE} = \sum_{j=1}^{J} \left(\sum_{\mathbf{x}_i \in R_j \cap \mathrm{Tr}} (y_i - \hat{y}_{R_j})^2 \right), \quad \hat{y}_{R_j} : \textbf{mean response of } y \textbf{ in } R_j \cap \mathrm{Tr}.$$

Ideally: compute SSE for all $\mathcal{R}$, and pick $\mathcal{R}^*$ that **minimizes** SSE.

Problem: not **computationally feasible**, in general.

Instead, we grow trees using **recursive binary splitting**, which is both:

- **top-down** – starts at the **root** and splits $\mathcal{X}$ recursively with 2 branches;
- **greedy** – at each step of the splitting process, the best choice is made **there and now**, rather than by looking at **long-term consequences**.

Recursive Binary Splitting Regression Tree Algorithm

1. Let $\hat{y}_0 = \text{Avg}\{y(\mathbf{x}_i) \mid i = 1, \ldots, N, \ \mathbf{x}_i \in \text{Tr}\}$.

2. Set the baseline $\text{SSE}_0 = \sum_{i=1}^{N} (y_i - \hat{y}_0)^2$.

3. For each $k = 1, \ldots, p$, order the predictor values x_k of X_k in Tr:

$$\min\{x_{i,k}\}_{i=1}^{N} = v_{k,1} \leq v_{k,2} \leq \cdots \leq v_{k,N} = \max\{x_{i,k}\}_{i=1}^{N}.$$

4. For each $k = 1, \ldots, p$, set the **potential branching points**

$$s_{k,\ell} = \tfrac{1}{2}(v_{k,\ell} + v_{k,\ell+1}), \quad \ell = 1, \ldots, N-1.$$

5. For each $k = 1, \ldots, p$, $\ell = 1, \ldots, N-1$, define

$$R_1(k,\ell) = \{\vec{X} \in \mathbb{R}^p \mid X_k < s_{k,\ell}\}, \;\; R_2(k,\ell) = \{\vec{X} \in \mathbb{R}^p \mid X_k \geq s_{k,\ell}\}.$$

Note that $\mathcal{X} = R_1(k,\ell) \sqcup R_2(k,\ell)$, for all k, ℓ.

6. For each $k = 1, \ldots, p$, $\ell = 1, \ldots, N-1$, set

$$\mathrm{SSE}_1^{k,\ell} = \sum_{m=1}^{2} \left(\sum_{\vec{X}_i \in R_m(k,\ell)} (y_i - \hat{y}_{R_m(k,\ell)})^2 \right),$$

where $\hat{y}_{R_m(k,\ell)} = \mathrm{Avg}\{y(\mathbf{x_i}) \mid i = 1, \ldots, N, \;\; \mathbf{x}_i \in \mathrm{Tr} \cap R_m(k,\ell)\}$.

7. Find k^*, ℓ^* for which $\mathrm{SSE}_1^{k,\ell}$ is **minimized**.

8. Define the **children sets** $R_1^L = R_1(k^*, \ell^*)$ and $R_1^R = R_2(k^*, \ell^*)$.

9. While at least one children set R_μ^ν does not meet a **stopping criterion**, repeat steps 3 to 8, searching and minimizing SSE over $\mathcal{X} \cap R_\mu^\nu$, and producing a **binary split** $R_{\mu+1}^L$, $R_{\mu+1}^R$.

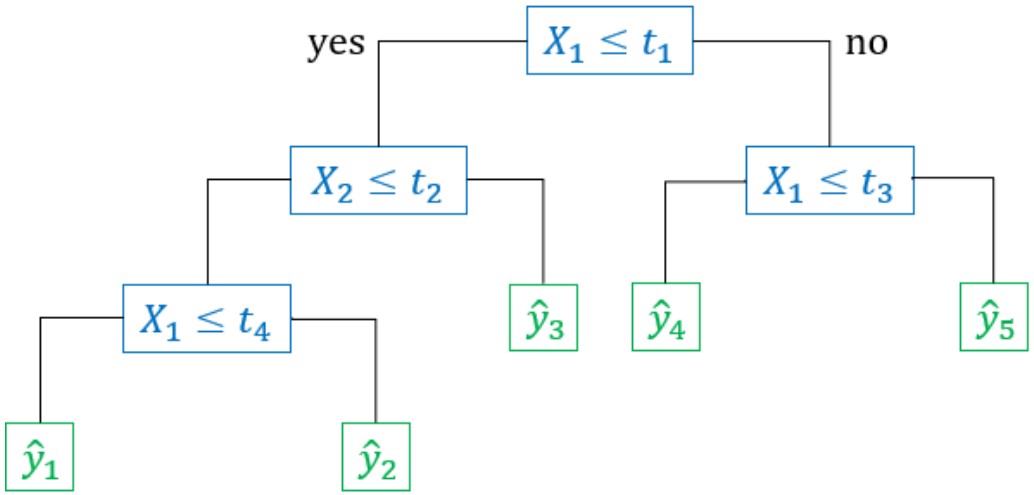
10. Once the stopping criterion is met for all children sets, the tree's growth ceases; $\mathcal{X}$ has been partitioned into J regions (renaming as necessary)

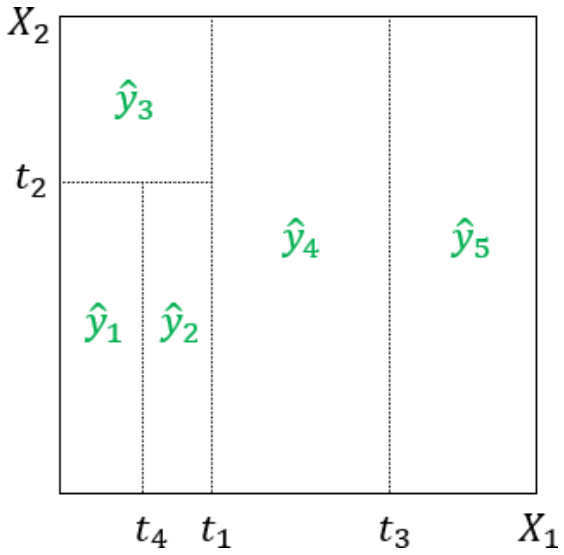
$$\mathcal{X} = R_1 \sqcup \cdots \sqcup R_J,$$

on which the regression tree **predicts** the J **responses** $\{\hat{y}_1, \ldots, \hat{y}_J\}$, according to $\hat{y}_j = \mathrm{Avg}\{y(\mathbf{x}_i) \mid i = 1, \ldots, N,\ \mathbf{x} \in \mathrm{Tr} \cap R_j\}$.

Stopping criteria: pure leaves, min. # of obs., Gini impurity, etc.

For $\mathrm{Tr} = \{(x_{1,i}, x_{2,i}, y_i)\}_{i=1}^{N}$, the algorithm might produce the tree below.





Generic recursive binary partition regression tree over $\mathcal{X} \subseteq \mathbb{R}^2$, with 5 leaves

In `R`, the algorithm is implemented in the `rpart` function `rpart()`.

3.6.2 – Tree Pruning

A regression trees with **unchecked growth** is prone to **overfitting**; it makes **good** predictions on Tr, but **poor** ones on Te, because the tree might be **too complex** – **data noise** is misinterpreted as **data signal**.

A **simpler** tree (**lower** complexity) might lead to **better** interpretability (and **lower** variance), at the cost of **reduced** accuracy (**higher** bias).

Intuition: let $\hat{f}_{\mathrm{Tr}}$ be the tree grown from the algorithm on Tr. With a different Tr^*, we may obtain a different tree $\hat{f}_{\mathrm{Tr}^*}$.

If we are building **small** trees, there are **fewer** ways for the predictions $\hat{f}_{\mathrm{Tr}}(\mathbf{x})$ to **vary** with Tr at $\mathbf{x} \in \mathcal{X}$. But **small** trees are also **less likely** to be **accurate** at $\mathbf{x} \in \mathcal{X}$ ($\hat{f}_{\mathrm{Tr}}(\mathbf{x}) \not\approx f(\mathbf{x})$) as they are **simpler**.

Once we have **grown** T_0 until an appropriate **stopping criterion** is met, we attempt to **prune** it to obtain an **optimal subtree**, as follows.

We use **cost complexity pruning (CCP)** to build a sequence of **subtrees** (starting at the **root**) indexed by the **complexity parameter** $\alpha \geq 0$. For each such α, find a **subtree** $T_\alpha \subseteq T_0$ which minimizes

$$\text{SSE} + \text{complexity penalty} = \left(\sum_{m=1}^{|T|} \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \right),$$

where $|T|$ is the number of leaves in T; when α is large, it is **costly** to produce a **complex** tree.

Tree pruning is similar to the **bias-variance trade-off** or **regularization**: a good tree balances considerations of **fit** and **complexity**.

Pruning Algorithm

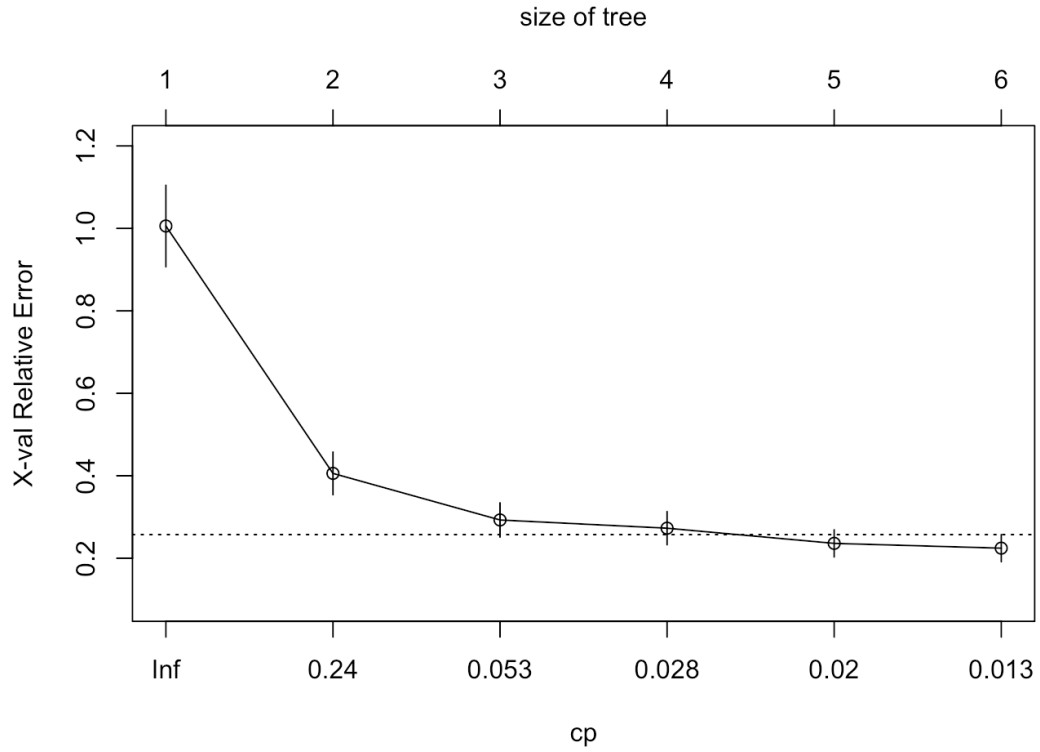
If T_0 has been grown on Tr, using a given stopping criterion:

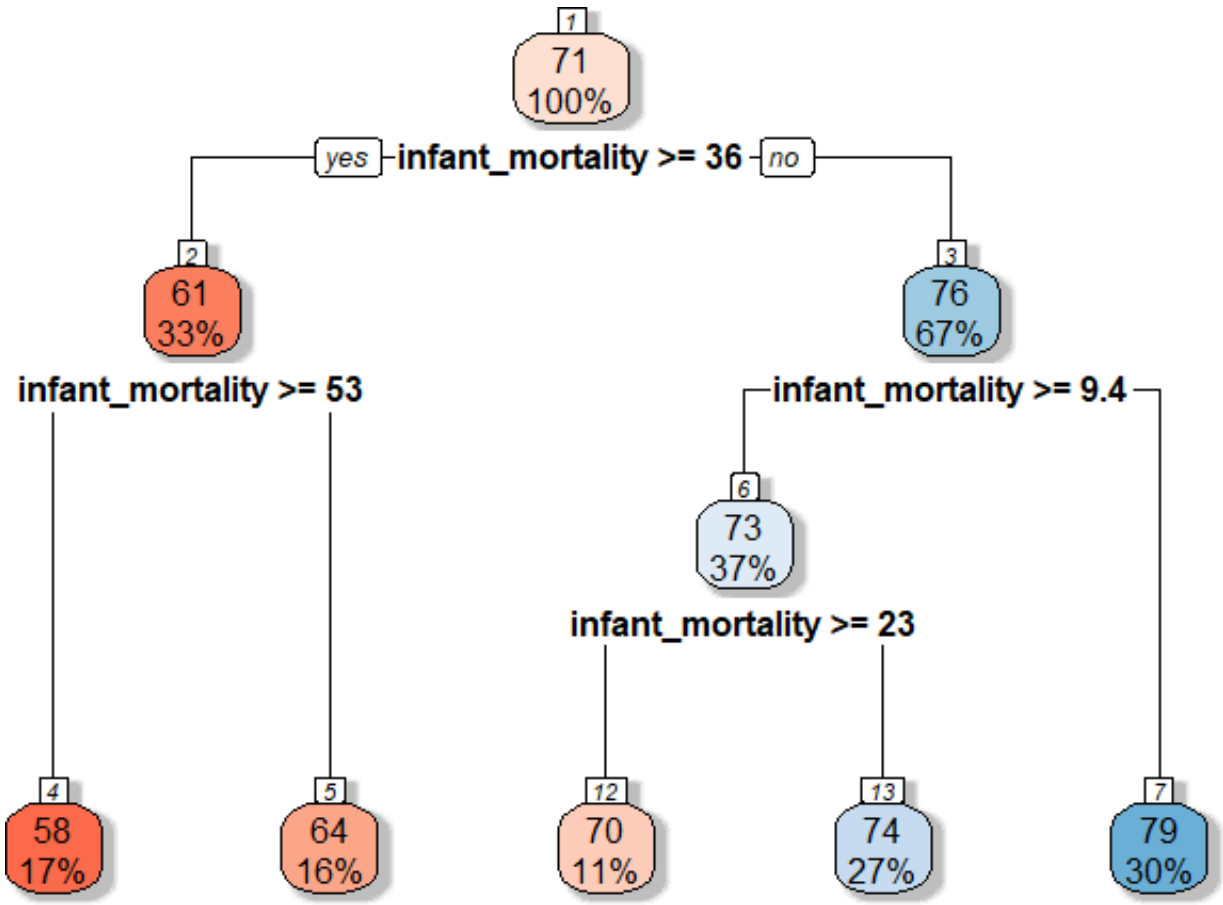
1. apply CCP to T_0 to obtain a "**sequence**" T_α of subtrees of T_0;
2. divide Tr into K **folds** ;
3. for $k = 1, \ldots, K$, build a regression tree $T_{\alpha;k}$ on $\mathrm{Tr} \setminus \mathrm{Fold}_k$ and evaluate

$$\widehat{\mathrm{MSE}}(\alpha) = \mathrm{Avg}\{\mathrm{MSE}_k(\alpha) \text{ of } T_{\alpha;k} \text{ on } \mathrm{Fold}_k \mid k = 1, \ldots, K\};$$

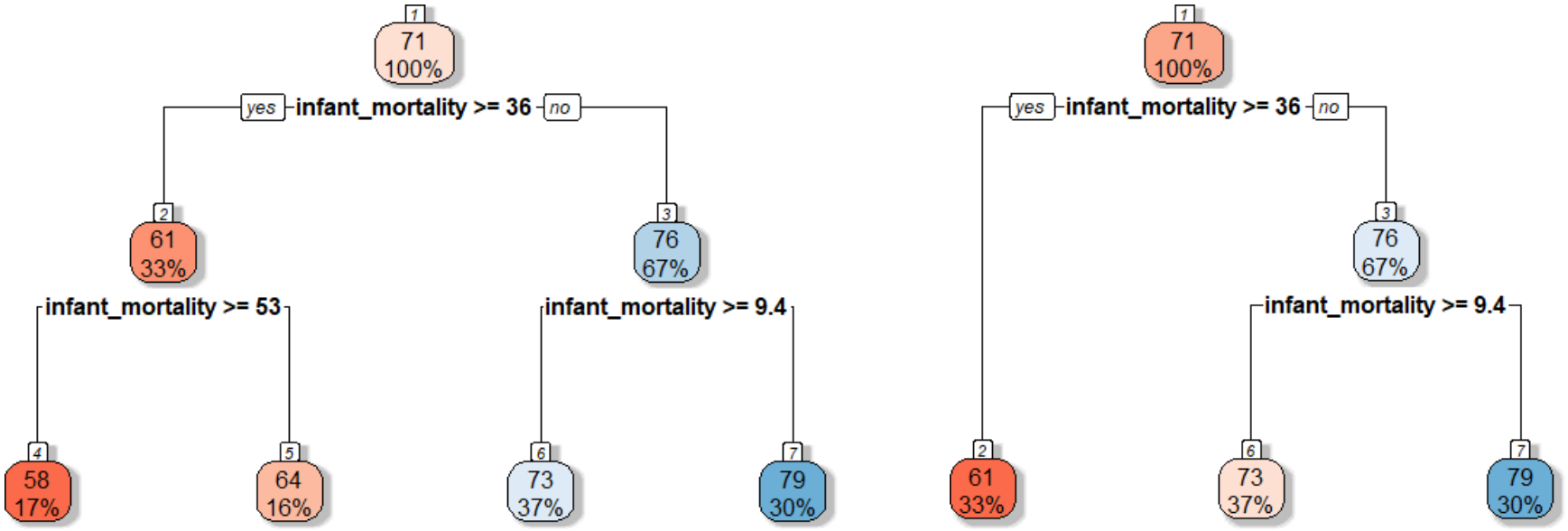
4. return T_{α^*} from step 1, where $\alpha^* = \arg\min_\alpha\{\widehat{\mathrm{MSE}}(\alpha)\}$.

We prune the Gapminder 2011 tree in `R` using the `rpart` functions `plotcp()` (α is denoted by `cp`) and `rpart()` (see DUDADS, 20.3, *Tree-Based Methods*, "Pruning Algorithms" for code).





Pruned tree, with $\alpha = 0.02$



Pruned tree, with $\alpha = 0.028$ (left) and $\alpha = 0.06$ (right)

The trees' complexity **increases** when α **decreases** (T_0: original tree).

What is T_{α^*} in this case? The rightmost α lying above the **horizontal line** in the CP plot is a good choice for α^*, in practice ($\alpha^* = 0.028$).

3.6.3 – Classification Trees

The **classification** approach is much the same. Throughout, let $\hat{p}_{j,k}$ be the **proportion of Tr of class k in leaf R_j**. Then:

1. prediction in a leaf is either the **class label mode** or the **relative frequency** of the class labels;

2. SSE must be replaced by some other **measure of fit**, such as

   - the **classification error rate**

$$E = \sum_{j=1}^{J} (1 - \max_k \{\hat{p}_{j,k}\});$$

- the **Gini index**, which measures the total variance across classes

$$G = \sum_{j=1}^{J} \sum_{k} \hat{p}_{j,k}(1 - \hat{p}_{j,k}),$$

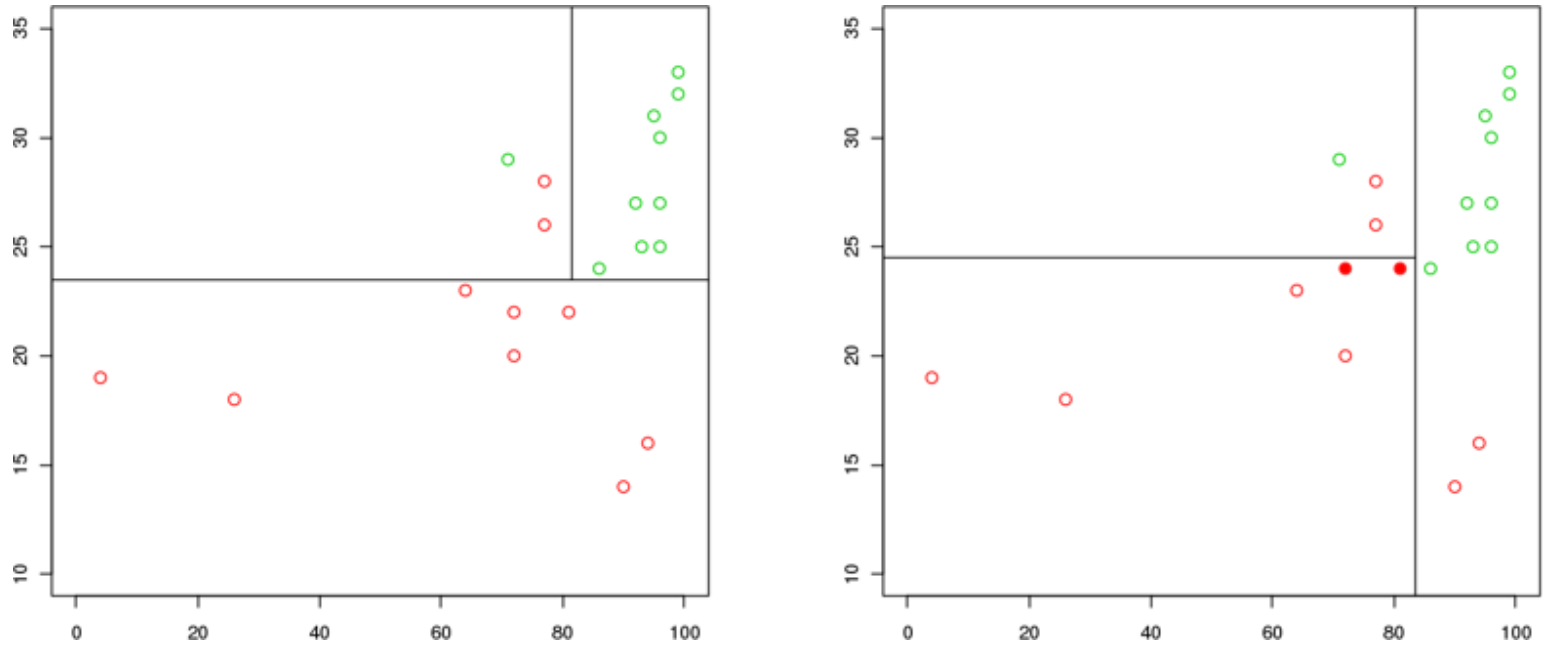
  which is small when the nodes are **almost pure** ($\hat{p}_{j,k} \approx 0$ or 1), and
- the **cross-entropy deviance**

$$D = -\sum_{j=1}^{J} \sum_{k} \hat{p}_{j,k} \ln \hat{p}_{j,k},$$

  which behaves like the Gini index, numerically.

Classification and regression trees (CART) have **high variance** and their structure is **unstable** – different Tr can give rise to **wildly varying trees**.

Extreme example: modifying the level of only one of the predictors in only two observations can yield a tree with a completely different **topology**:



This **lack of robustness** is a strike against CART **use**; but their relative ease of implementation explains their popularity (also: **stumps**, **boosting**).

3.6.4 – Examples: Iowa Housing Data

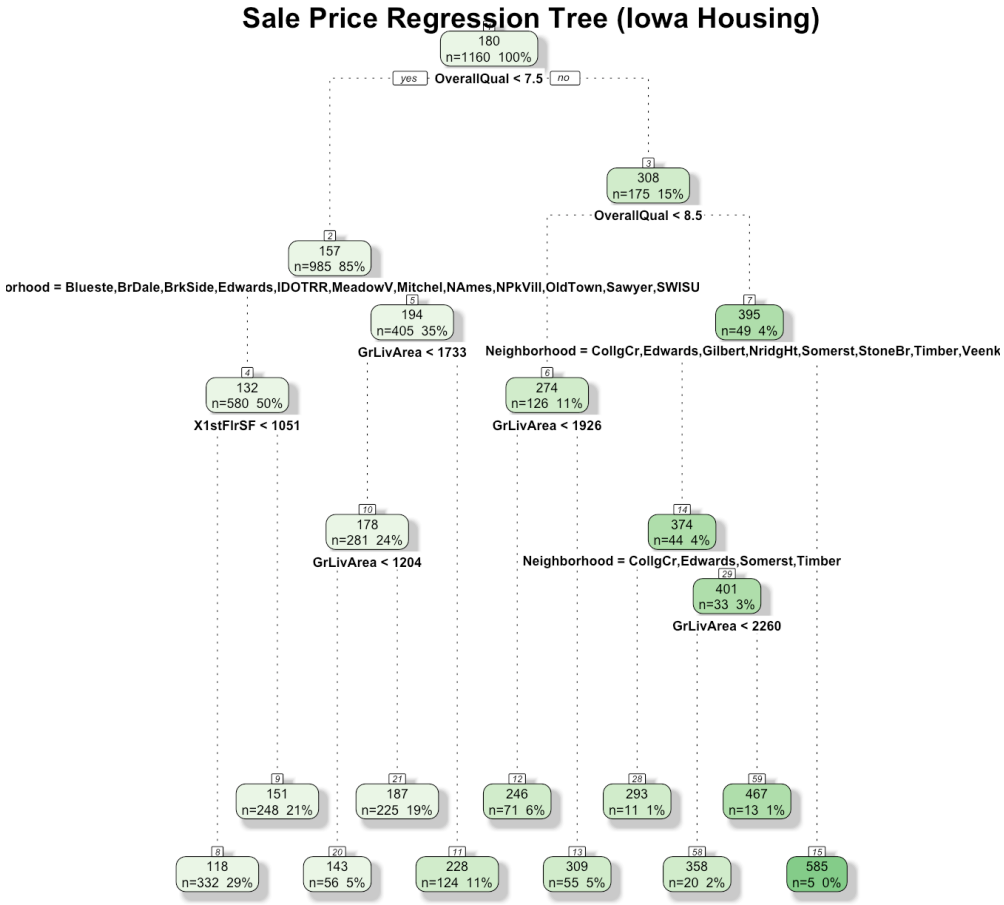
We now revisit the **Iowa Housing Price** data (`VE_Housing.csv`), containing 1460 observations of 61 variables relating to the **selling price** `SalePrice` of houses in Ames, Iowa (the data has been processed).

In order for that model to be useful for predictions, we must have a sense of its MSE on new observations.

The training set Tr consists of $N = 1160$ randomly selected observations; the remaining $M = 300$ observations are set aside for the testing set Te.

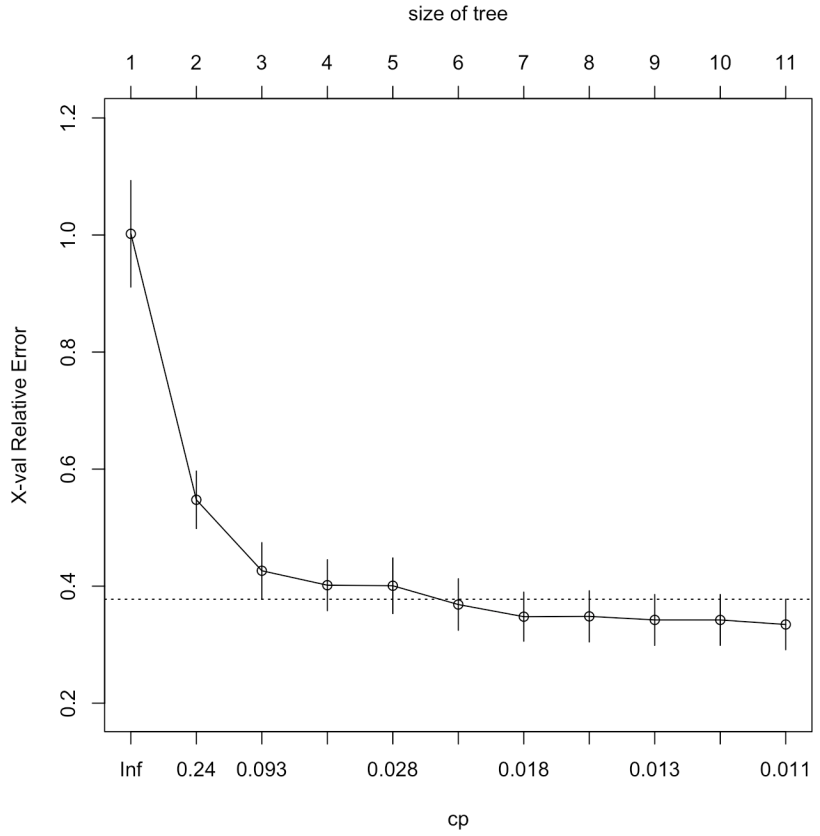
We build a **CART model** for the response variable, requiring a minimum of 5 observations per leaf.

(See DUDADS, 20.3, *Tree-Based Methods*, "Examples" for code).

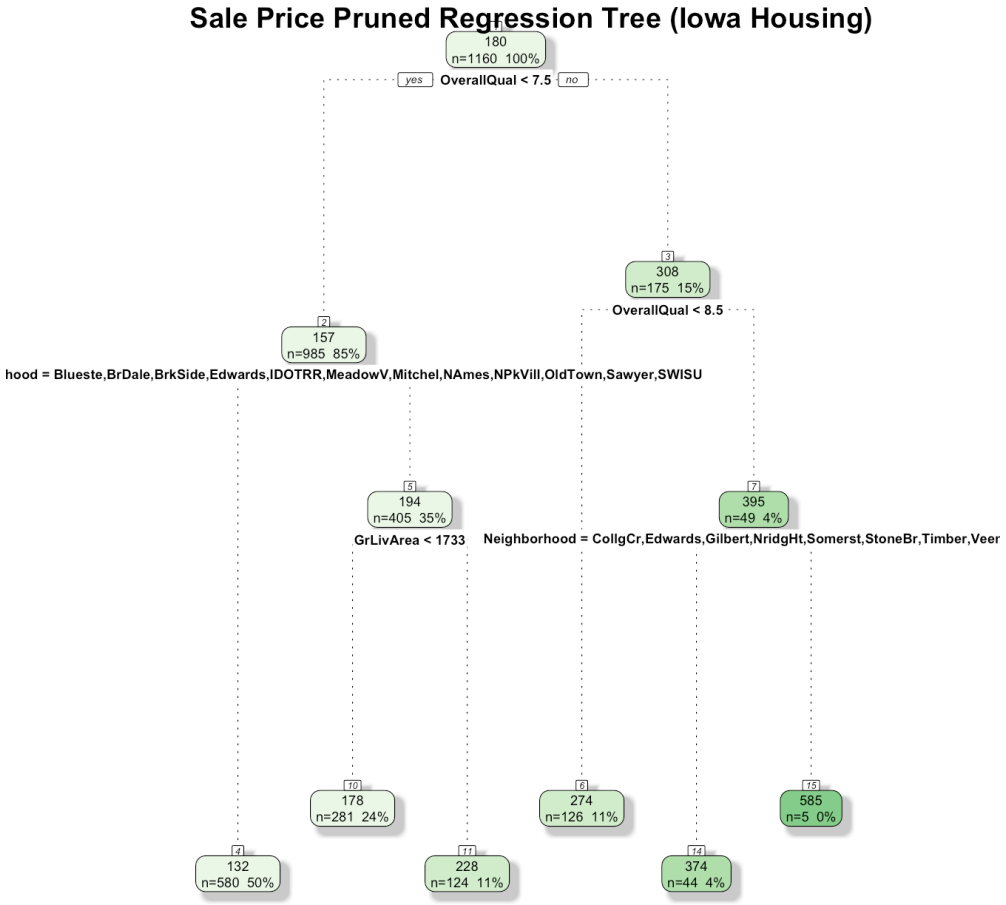


There are 11 leaves in total

Next we use `rpart`'s `plotcp()` to determine how to **prune the tree**.



We see that the complexity parameter should be $\alpha \approx 0.024$



The pruned tree has 6 leaves

Next, we determine how well **FT** and **PT** perform as predictive models on Te.

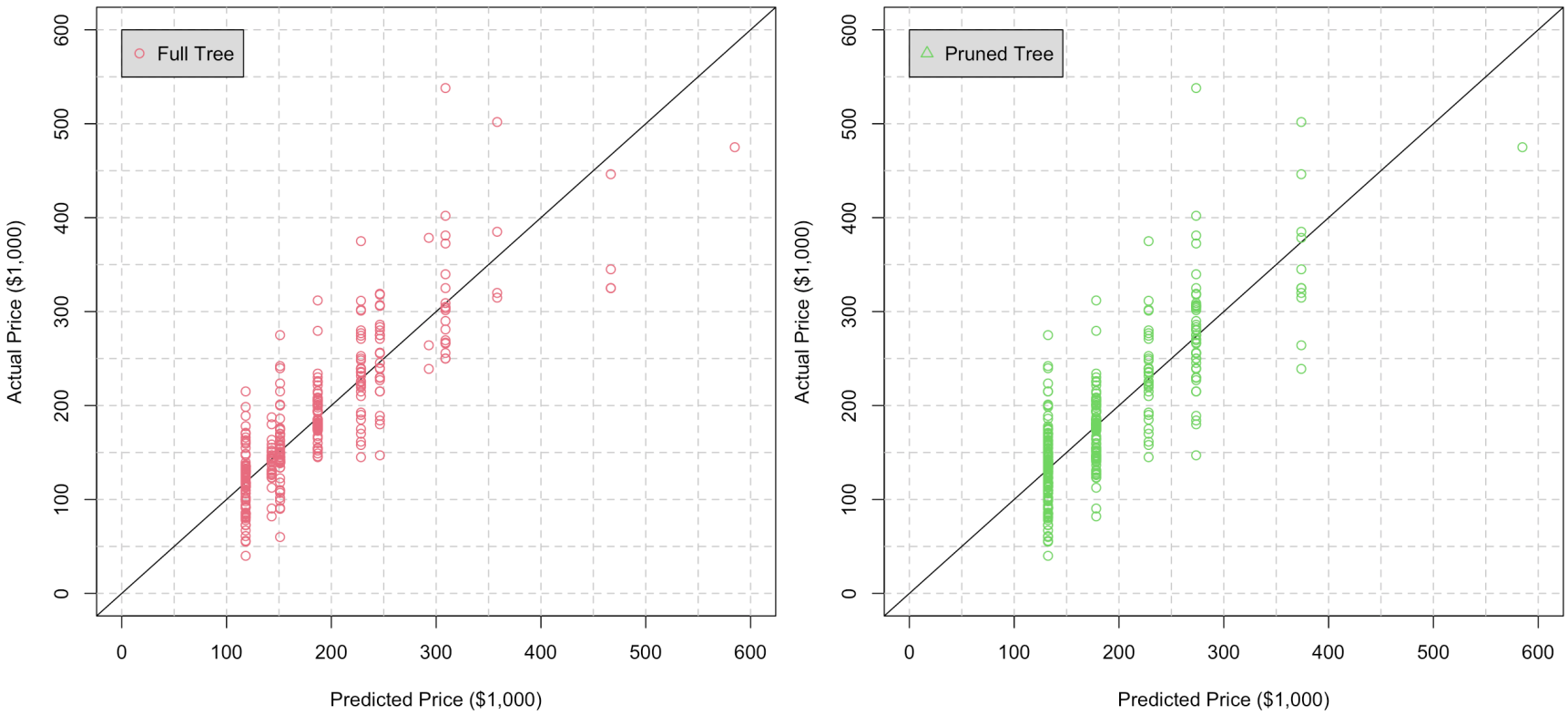
For the full tree, we compute the reduction in SSE as:

$$1 - \mathrm{NMSE}_{\mathsf{FT}} = 1 - \frac{\sum_{i=1}^{M} \left(\hat{y}_{i,\mathsf{FT}} - y_i\right)^2}{\sum_{i=1}^{M} \left(\overline{y} - y_i\right)^2} = 0.703$$

For the pruned tree, the corresponding reduction is:

$$1 - \mathrm{NMSE}_{\mathsf{PT}} = 0.646$$

This suggests that pruning is a reasonable approach. The predictions of both trees are plotted against the actual `SalePrice` in Te.



Given that CART can only predict a small number of selling prices (the **leaves**), the $\hat{y}_i$ are reasonably accurate ($\rho_{\text{FT}} = 0.84$, $\rho_{\text{PT}} = 0.80$).

3.7 – Ensemble Learning

In practice, individual learners are often **weak** – they perform better than **random guessing**, but not *that much* better.

In the late 80's, Kearns and Valiant asked: "can a set of weak learners be used to create a strong learner?" Yes, *via* **ensemble learning** methods.

Absolutely insane example: scientists trained 16 **pigeons** (**weak** learners, one would assume) to identify pictures of magnified biopsies of possible breast cancers.

On average, each **pigeon** had an accuracy of about 85%; when the **most popular answer** in the group was selected, the accuracy jumped to 99%.

3.7.1 – Bagging

Bagging is an extension of **bootstrapping**, which is used in situations where it is nearly impossible to compute the variance of a quantity of interest by exact means (see Section 2.4.2).

But it is also used to **improve the performance** of statistical learners, especially those that exhibit **high variance** (such as CART).

Low variance methods (those for which the results, structure, predictions, etc. **remain roughly the same** with different Tr, such as OLS when $N/p \gg 1$) do not benefit as much from the use of **ensemble learning**.

Averaging can be used to **reduce the variance** of a ML method.

If $Z_1, \ldots, Z_B$ are **independent** predictions at $\mathbf{x} \in \mathrm{Te}$, say, with

$$\mathrm{Cov}(Z_i, Z_j) = \begin{cases} \sigma^2 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

then $\overline{Z}$ is also a prediction at $\mathbf{x} \in \mathrm{Te}$ and CLT $\Longrightarrow$

$$\mathrm{Var}(\overline{Z}) = \mathrm{Var}\left(\frac{Z_1 + \cdots + Z_B}{B}\right) = \frac{1}{B^2}\mathrm{Var}(Z_1 + \cdots + Z_B)$$

$$= \frac{1}{B^2}\sum_{i,j=1}^{B}\mathrm{Cov}(Z_i, Z_j) = \frac{1}{B^2}\sum_{k=1}^{B}\mathrm{Var}(Z_i) = \frac{\sigma^2}{B} \leq \sigma^2, \quad \forall B \in \mathbb{N}.$$

But we do not usually have access to **multiple** $(B > 1)$ Tr sets, so ... ?

Resampling methods can be used to generate multiple **bagging training sets** Tr_i from the original training set Tr.

Let $B > 1$ be an integer. We generate B **bootstrapped** Tr_i by sampling $N = |\mathrm{Tr}|$ observations from Tr **with replacement**, yielding

$$\mathrm{Tr}_1, \ldots, \mathrm{Tr}_B.$$

Next, we train a model $\hat{f}_i$ (for **regression**) or $\hat{C}_i$ (for **classification**) on each Tr_i, $i = 1, \ldots, B$; for each $\mathbf{x}^* \in \mathrm{Te}$, we then have B **predictions**

$$\hat{f}_1(\mathbf{x}^*), \ldots, \hat{f}_B(\mathbf{x}^*) \quad \text{(for \textbf{regression})}$$
$$\hat{C}_1(\mathbf{x}^*), \ldots, \hat{C}_B(\mathbf{x}^*) \quad \text{(for \textbf{classification}).}$$

The **bagging prediction** at $\mathbf{x}^* \in \mathrm{Te}$ is the **average** or the **mode**:

$$\hat{f}_{\mathrm{Bag}}(\mathbf{x}^*) = \frac{1}{B}\sum_{i=1}^{B} \hat{f}_i(\mathbf{x}^*) \quad \text{(for regression)},$$

$$\hat{C}_{\mathrm{Bag}}(\mathbf{x}^*) = \mathrm{Mode}\{\hat{C}_1(\mathbf{x}^*), \ldots, \hat{C}_B(\mathbf{x}^*)\} \quad \text{(for classification)}.$$

Bagging is particularly helpful for **CARTs**; to take full advantage of bagging, use **deep trees** (i.e., **no pruning**) – their complexity leads to **high variance** but **low bias** at the individual model level (BVTO).

In practice, the bagged **tree** predictions will also have **low bias**, but the bagging process **reduces the variance**; using 100s/1000s **trees** typically produces **greatly improved predictions** (at the cost of **interpretability**).

Out-of-Bag Error Estimation

Model j is fit to the **bootstrapped** Tr_j, $j = 1, \ldots, B$. On average, Tr_j contains $\approx 2/3$ distinct observations of Tr (ex.) $\implies$ $\approx 1/3$ of Tr is not used to build model j (**out-of-bag (OOB) observations**).

We can predict the response y_i for observation i in Tr using **only those models** for which $\mathbf{x}_i$ is OOB; there should be $\approx B/3$ OOB predictions, and

$$\hat{y}_{\mathrm{OOB},i} = \mathrm{Avg}\{\hat{f}_j(\mathbf{x}_i) \mid \mathbf{x}_i \in \mathrm{OOB}(\mathrm{Tr}_j) = \mathrm{Tr} \setminus \mathrm{Tr}_j\} \quad \text{(for \textbf{regression})}$$

$$\hat{y}_{\mathrm{OOB},i} = \mathrm{Mode}\{\hat{C}_j(\mathbf{x}_i) \mid \mathbf{x}_i \in \mathrm{OOB}(\mathrm{Tr}_j)\} \quad \text{(for \textbf{classification}).}$$

OOB $\mathrm{MSE}_{\mathrm{Tr}}$ and OOB $\mathcal{E}_{\mathrm{Tr}}$ are thus good estimates of $\mathrm{MSE}_{\mathrm{Te}}$ and $\mathcal{E}_{\mathrm{Te}}$ since **none of the predictions** $\hat{y}_{\mathrm{OOB},i}$ are made by models that used Te observations in their training $\implies$ no need for costly **CV** Te error estimate.

Variable Importance Measure

Bagging improves the accuracy of **stand-alone** models, but at the cost of **reduced interpretability**, especially for **CART**: the bagged **tree** predictions cannot usually be expressed with a **single tree**.

In a **CART**, the relative importance of each **feature/variable** is linked to the **hierarchy of splits** $\Longrightarrow$ the most "**important**" variables appear in splits that are **close** to the root.

For bagged **regression trees**, we summarize the importance of each variable by measuring the **total decrease** in SSE due to splits over a given predictor $\Longrightarrow$ we compare SSE in trees **with** these splits against SSE in trees **without** these splits, averaged over the B **bagged** trees.

For bagged **classification trees**, use the **Gini index** instead of SSE.

Another approach: weigh the importance of a factor **inversely proportionally** to the level in which it appears (if at all) in **each tree** and to average over **all bagging trees**.

Example: if predictor X_1 appears in the 1st split level of bagged tree 1, the 4th split level of bagged tree 2, and the 3rd split level of bagged tree 5, whereas predictor X_2 appears in the 2nd, 2nd, 3rd, and 5th split levels of bagged trees 2, 3, 4, 5 (resp.), then the relative importance of each predictor over the 5 bagged trees is

$$X_1 : (1 + 1/4 + 0 + 0 + 1/3) \cdot 1/5 = 19/60 = 0.32$$
$$X_2 : (0 + 1/2 + 1/3 + 1/3 + 1/5) \cdot 1/5 = 23/75 = 0.31;$$

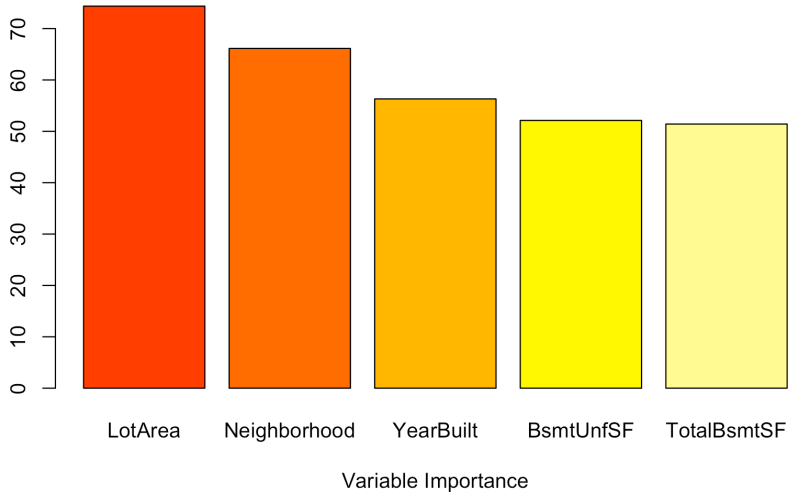
the first variable is **nominally more** important than the second.

Example: Iowa Housing Data

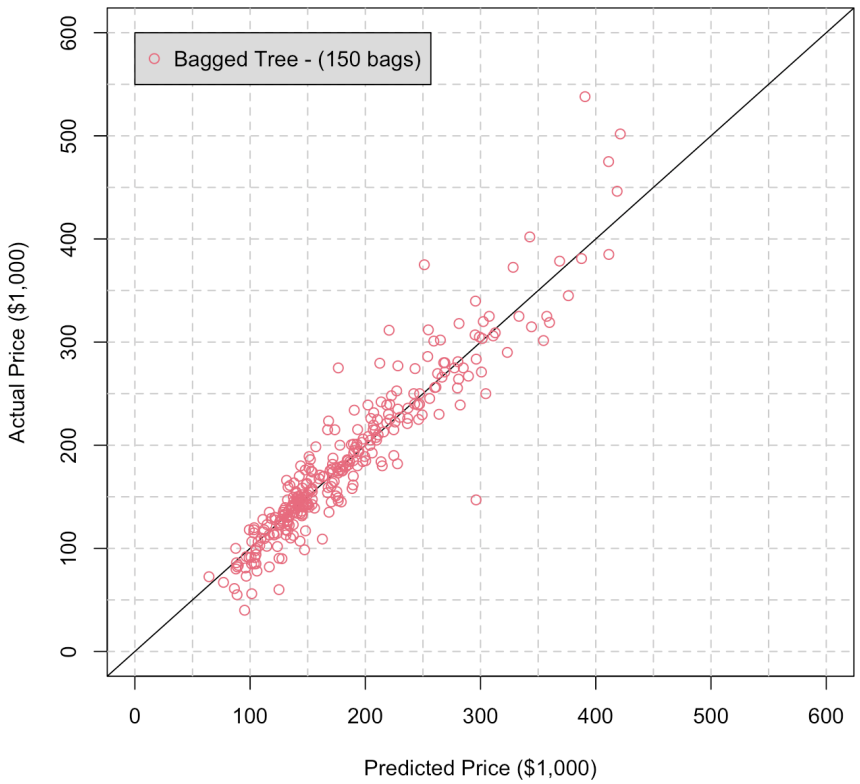
We revisit the **Iowa Housing** data example of the previous section, in which we used a Tr with $N = 1160$, relating to the selling price `SalePrice` of houses in Ames, Iowa.

We build a regression tree bagging model using $B = 150$ bags, and with an OOB error estimate.

The 5 most important variables are shown in the graph opposite.



(See DUDADS, 21.5.1, *Bagging*, for code).



The correlation between predicted bagged and actual sale prices on Te is $\rho = 0.94$; how does that compare to the previous MARS and CART models?

3.7.2 – Random Forests

The main idea behind **bagging** is to fit a model on **various** Tr_i and to use **CLT** + **independence of bagged predictions** to reduce the variance.

In practice, the latter assumption is **rarely met**: if there are a small number of **strong** predictors wrt to the response in Tr, each **bagged model** $\hat{f}_i/\hat{C}_i$ built on the bootstrapped training set Tr_i is likely to be similar to the others, and the various predictions are **unlikely to be un-**correlated, so that

$$\mathrm{Var}(\hat{y}^*_{\mathrm{Bag}}) \neq \frac{\sigma^2}{B}, \quad \mathbf{x}^* \in \mathrm{Te};$$

averaging **highly correlated** quantities **does not reduce** the variance significantly: the CLT assumption of independence is **necessary**.

We can **decorrelate** the bagging models with a small tweak, leading to variance reduction for the bagged predictions.

Random forests (RF) also build models on B **bootstrapped** Tr_i, but each model uses a **random subset** of predictors.

In a CART, every time a split is considered, the allowable predictors are selected from a **random subset** of m predictors out of the **full** p predictors.

We lose out on **performance** by selecting predictors **randomly**, but we also reduce the chance of the models being **correlated**. For $\mathbf{x}^* \in \mathrm{Te}$, the B predictions are combined as in bagging to yield the **RF prediction** $\hat{y}^*_{\mathrm{RF}}$.

If $m = p$, RF reduces to a **bagged model**; in practice we use $m \approx \sqrt{p}$ for classification and $m \approx p/3$ for regression. When the predictors are **highly correlated**, however, **even smaller** values of m are recommended.

Example: Wines Dataset

The `wine.csv` dataset consists of $n = 178$ wine samples (observations) and $p = 13$ predictors. The response variable is the wine class: 1, 2, or 3.

We implement a $70\%/30\%$ separation for $\mathrm{Tr}\,/\,\mathrm{Te}$.

As there are $p = 13$ predictors in the dataset, we could use $m \approx \sqrt{13} \approx 4$ predictors at each split. But the variables may be correlated, so we should also try a RF model with a small m ($= 2?$).

We use $B = 500$ trees. In this example, the choice of m only introduces slight differences, but this will not always be the case.

(See DUDADS, 20.5,2 *Random Forests*, for code).

```
No. of variables tried at each split: 2
        OOB estimate of  error rate: 0.7%
Confusion matrix:
   1  2  3 class.error
1 47  0  0  0.00000000
2  0 54  1  0.01818182
3  0  0 40  0.00000000

No. of variables tried at each split: 4
        OOB estimate of  error rate: 1.41%
Confusion matrix:
   1  2  3 class.error
1 46  1  0  0.02127660
2  0 54  1  0.01818182
3  0  0 40  0.00000000
```

3.7.3 – Boosting

Boosting does not involve bootstrap sampling, but fits models on a **hierarchical sequence of residuals**, in a **slow manner**. For **regression problems**, we proceed as follows:

1. set $\hat{f}(\mathbf{x}) = 0$ and $r_i = y_i$ for all $\mathbf{x}_i \in \mathrm{Tr}$;

2. for $b = 1, 2, \ldots, B$:
   i. fit a model $\hat{f}^b$ to the training set $\mathrm{Tr}_b = (\mathbf{X}, \mathbf{r})$;
   ii. update the regression function $\hat{f} := \hat{f} + \lambda \hat{f}^b$;
   iii. update the residuals $r_i := r_i - \lambda \hat{f}^b(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \mathrm{Tr}$;

3. output the boosted model $\hat{f}_{\mathrm{Boost}}(\mathbf{x}) = \lambda(\hat{f}^1(\mathbf{x}) + \cdots + \hat{f}^B(\mathbf{x}))$.

Let's do a run through:

1. we start with $\hat{f} \equiv 0$ and $\mathbf{r} = \mathbf{y}$;

2. for $b = 1$:

   i. $\hat{f}^1$ is the approximation of the true regression function built from Tr using whatever algorithm we are considering;
   ii. we update $\hat{f} := 0 + \lambda \hat{f}^1 = \lambda \hat{f}^1$;
   iii. we update $r_i := r_i - \lambda \hat{f}^1(\mathbf{x}_i) = y_i - \lambda \hat{f}^1(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \mathrm{Tr}$;

   note: whenever $\lambda \hat{f}^1$ is "good", $r_i \approx 0$; otherwise, $|r_i|$ is "**large**";

2. for $b = 2$:

   i. $\hat{f}^2$ is the approximation of the true regression function on $\mathrm{Tr}_2 = (\mathbf{X}, \mathbf{r})$;
   ii. we update $\hat{f} := \hat{f} + \lambda \hat{f}^2 = \lambda(\hat{f}^1 + \hat{f}^2)$;
   iii. we update $r_i := r_i - \lambda \hat{f}^2(\mathbf{x}_i) = y_i - \lambda(\hat{f}^1(\mathbf{x}_i) + \hat{f}^2(\mathbf{x}_i))$ for all $\mathbf{x}_i \in \mathrm{Tr}$;

note: whenever $\lambda(\hat{f}^1 + \hat{f}^2)$ is "good", $r_i \approx 0$; otherwise $|r_i|$ is "**large**";

2. ...

2. for $b = B$:

   i. $\hat{f}^b$ is the approximation of the true regression function on $\mathrm{Tr}_B = (\mathbf{X}, \mathbf{r})$;
   ii. we update $\hat{f} := \hat{f} + \lambda \hat{f}^B = \lambda(\hat{f}^1 + \cdots + \hat{f}^B)$;
   iii. we update $r_i := r_i - \lambda \hat{f}^B(\mathbf{x}_i) = y_i - \lambda(\hat{f}^1(\mathbf{x}_i) + \cdots + \hat{f}^B(\mathbf{x}_i))$ for all $\mathbf{x}_i \in \mathrm{Tr}$;

   note: whenever $\lambda(\hat{f}^1 + \cdots + \hat{f}^B)$ is "good", $r_i \approx 0$; else $|r_i|$ is "**large**";

3. set $\hat{f}_{\mathrm{Boost}} = \lambda(\hat{f}^1 + \cdots + \hat{f}^B)$.

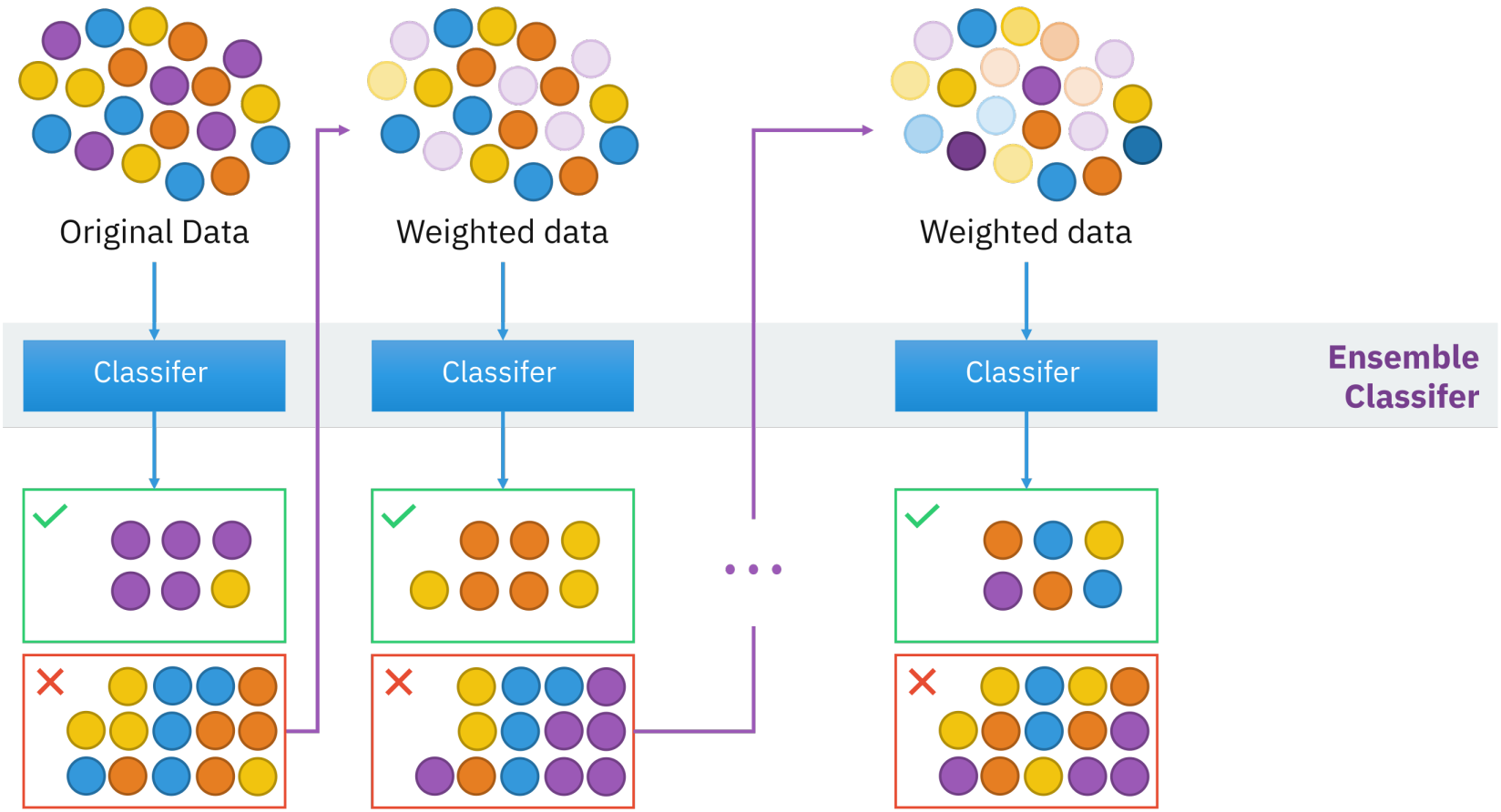
Essentially, at each step b, we find the best fit for the residuals and add a smaller version of that model to the current model.

This version of the **boosting** algorithm requires three **tuning parameters**:

- the **number of models** B, which can be selected through cross-validation (boosting can overfit if B is too large);
- the **shrinkage parameter** λ (typically, $0 < \lambda \ll 1$), which controls the **boosting learning rate** (a small λ needs a large B, in general); the optimal λ and B can be found *via* cross-validation, and
- not explicitly stated – we also need the learning models to reach some **complexity threshold**.

Variants: allowing for **classification**; allowing for **varying weights** depending on performance regions in predictor space .

NFL $\Longrightarrow$ no SL algorithm is always best regardless of context/data, the combination of **AdaBoost** (next) with **weak CART** learners is seen by many as the best "**out-of-the-box**" classifier (for **noisy** data, use **BrownBoost**).



by Sirakorn [https://commons.wikimedia.org/w/index.php?curid=85888769] – own work, CC BY-SA 4.0

Example: Credit Dataset

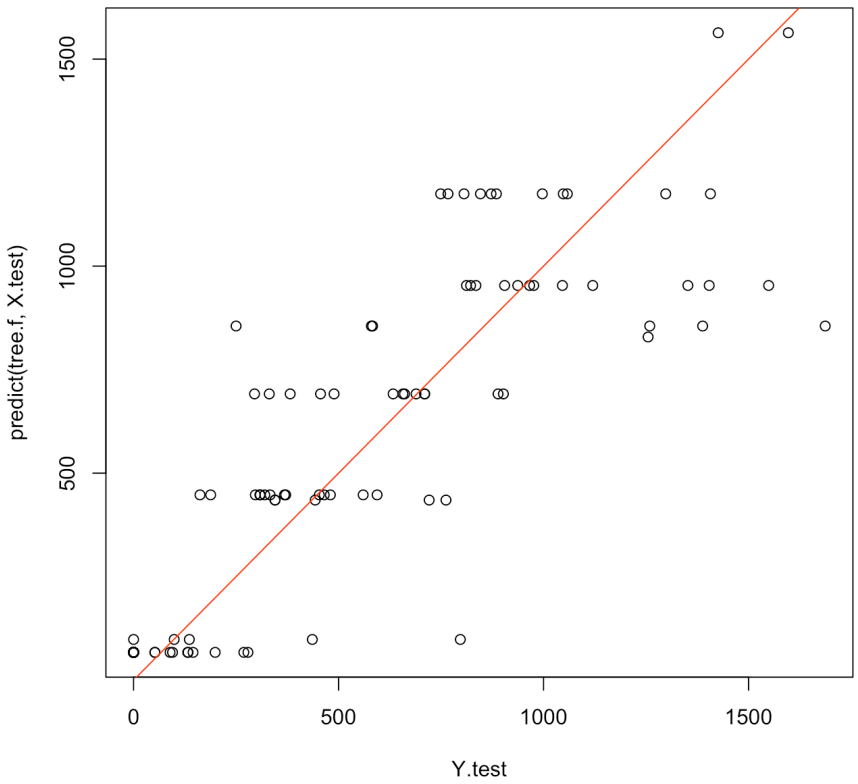
Consider the `Credit.csv` dataset; the task is to determine the credit card `Balance` based on a number of other factors. The dataset has 400 obs. with 10 predictors and 1 response:

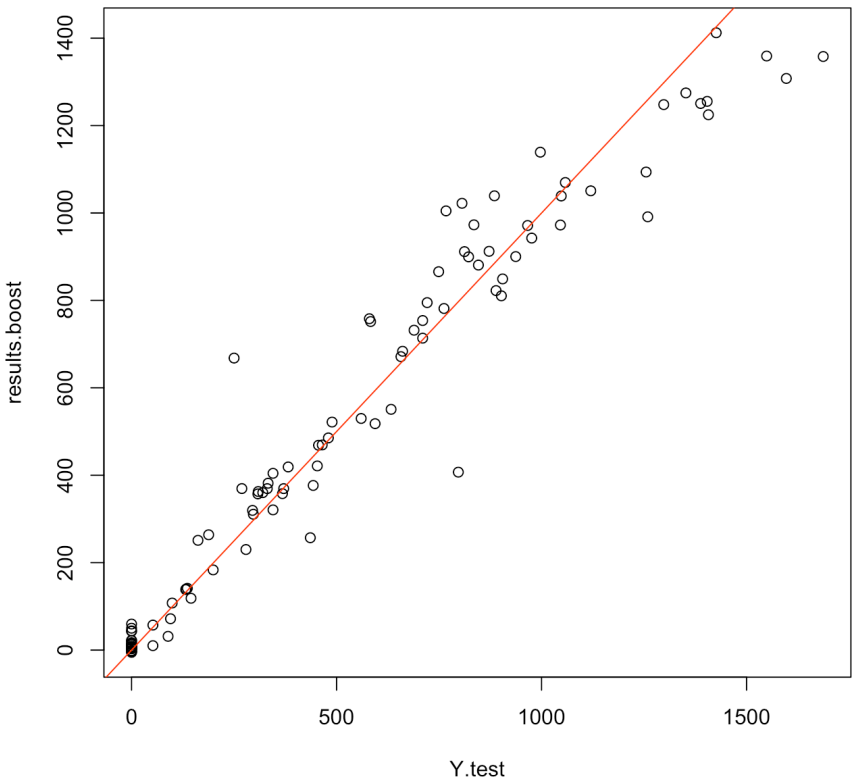
`Income, Limit, Rating, Cards, Age, Education, Gender, Student, Married, Ethnicity.`

We create binary variables for all categorical levels and pick $N = 300$ obs. for Tr; we use $\lambda = 0.005$ as a shrinkage parameter and $B = 2000$ models.

We start by building $\hat{f}^1$ with a **weak CART**; its predictions on Te are shown on the next slide, side-by-side with those of the complete model $\hat{f}_{\mathrm{Boost}}$.

(See DUDADS, 20.5,3 *Boosting*, for code).

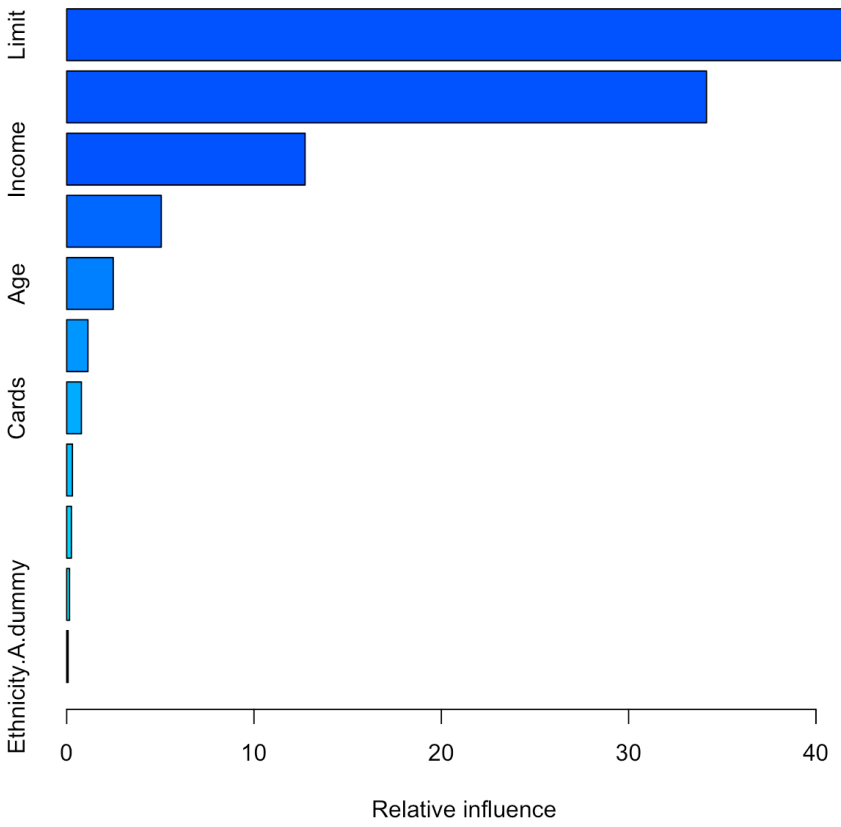




$B = 1$: middling; $\rho = 0.86$

$B = 2000$: much improved; $\rho = 0.97$

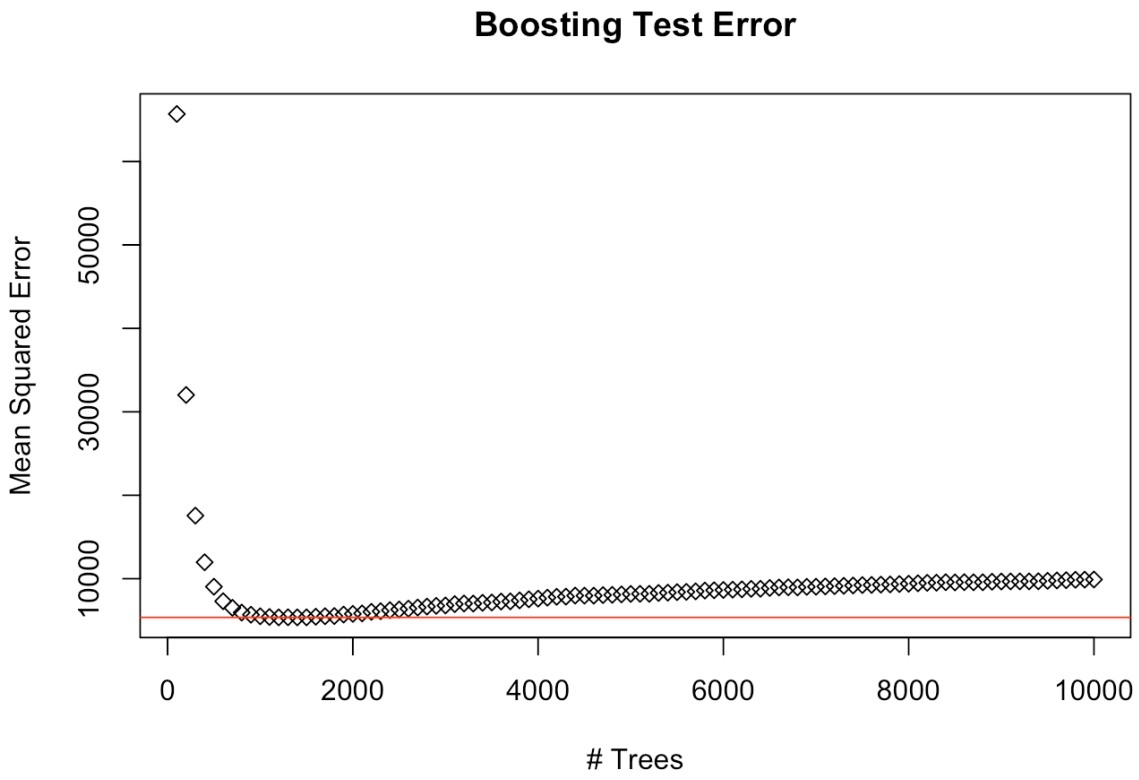
The influential predictors are on the next slide.



var	rel.inf
Limit	42.87367610
Rating	34.15766697
Income	12.72692829
Student.dummy	5.04913001
Age	2.48572012
Education	1.13432102
Cards	0.78545388
Ethnicity.AA.dummy	0.30692238
Married.dummy	0.25299822
Gender.dummy	0.15135520
Ethnicity.A.dummy	0.07582781

Not surprisingly, ethnicity and gender have **little** influence on the model.

How do we determine the optimal number of models B to use with λ?



For $\lambda = 0.005$, $B = 1200$ yields sensibly the same ρ as $B = 2000$.

AdaBoost

Adaptive Boosting (AdaBoost) adapts boosting to a **set of models** in order to minimize the error made by the **individual weak models** (e.g., "**stubby**" CART).

"**Adaptive**" $\Longrightarrow$ any new **(weak)** learner added to the boosted model is **modified to improve the predictive power** on instances which were "mis-predicted" by the (previous) boosted model.

Main idea: build a **weighted sum** of **weak** learners whose weights are adjusted so that the prediction error is **minimized**.

Consider a **binary classification context**, where

$$\mathrm{Tr} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\} \quad \text{and} \quad y_i \in \{-1, +1\} \quad \forall i = 1, \ldots, N.$$

For $f_b(\mathbf{x}) \in \{-1, +1\}$ and $c_b \in \mathbb{R}^+$, $b = 1, \ldots, B$, the **boosted classifier** is a function

$$F(\mathbf{x}) = \sum_{b=1}^{B} c_b f_b(\mathbf{x}).$$

The **class prediction** at $\mathbf{x}$ is $\operatorname{sgn}(F(\mathbf{x}))$ (more accurate $f_b \implies c_b$ larger).

AdaBoost contribution: for each $\mu \in \{1, \ldots, B\}$, the **weak learner** f_μ is trained on a **weighted version** of Tr, with observations that are misclassified by the **partial** boosted model

$$F_\mu(\mathbf{x}) = \sum_{b=1}^{\mu-1} c_b f_b(\mathbf{x})$$

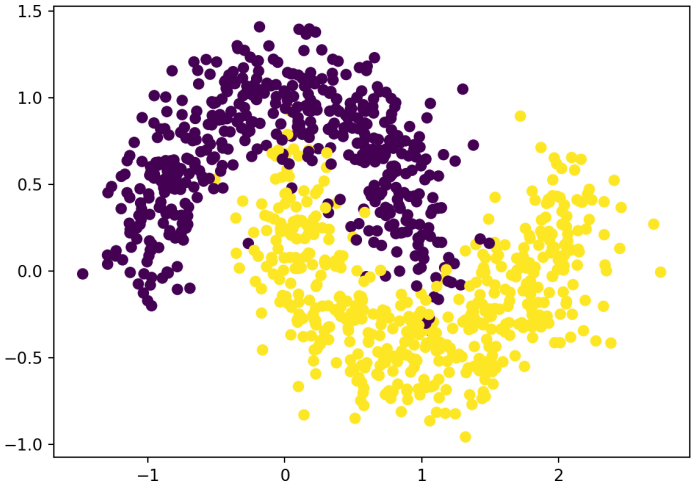
given **larger** weights; **AdaBoost** estimates the weights w_i, $i = 1, \ldots, N$ at each of the boosting steps $b = 1, \ldots, B$.

Given a weak learner, **Real AdaBoost** does away with the constants c_b:

1. **initialize** the weights $\mathbf{w}$, with $w_i = 1/N$, for $1 \leq i \leq N$;
2. for $b = 1, \ldots, B$:
   i. fit the class probability estimate $p_m(\mathbf{x}) = P(y = 1 \mid \mathbf{x}, \mathbf{w})$;
   ii. define $f_b(\mathbf{x}) = \frac{1}{2} \log \frac{p_b(\mathbf{x})}{1 - p_b(\mathbf{x})}$;
   iii. set $w_i \leftarrow w_i \exp\{-y_i f(\mathbf{x}_i)\}$, for $1 \leq i \leq N$;
   iv. re-normalize so that $\|\mathbf{w}\|_1 = 1$;
3. output the classifier $F_{\text{Ada}}(\mathbf{x}) = \text{sgn}\left\{\sum_{b=1}^{B} f_b(\mathbf{x})\right\}$.

Boosting is susceptible to **overfitting** $\implies$ use CV to find optimal B.

Example: Two-Moons Dataset

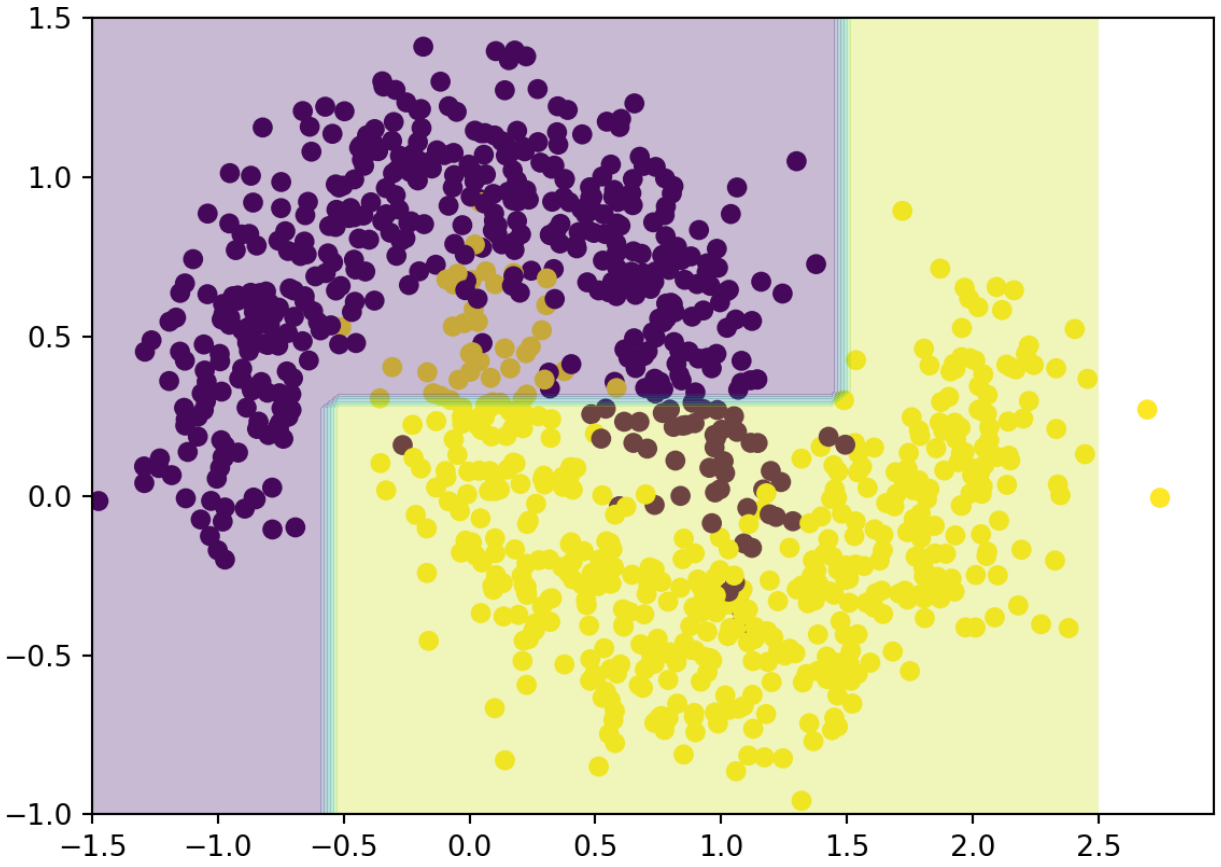


We use a decision tree classifier (weak learner) to learn an AdaBoost classifier on the classic $Two\text{-}Moons$ dataset, consisting of two interleaving half circles with added noise.

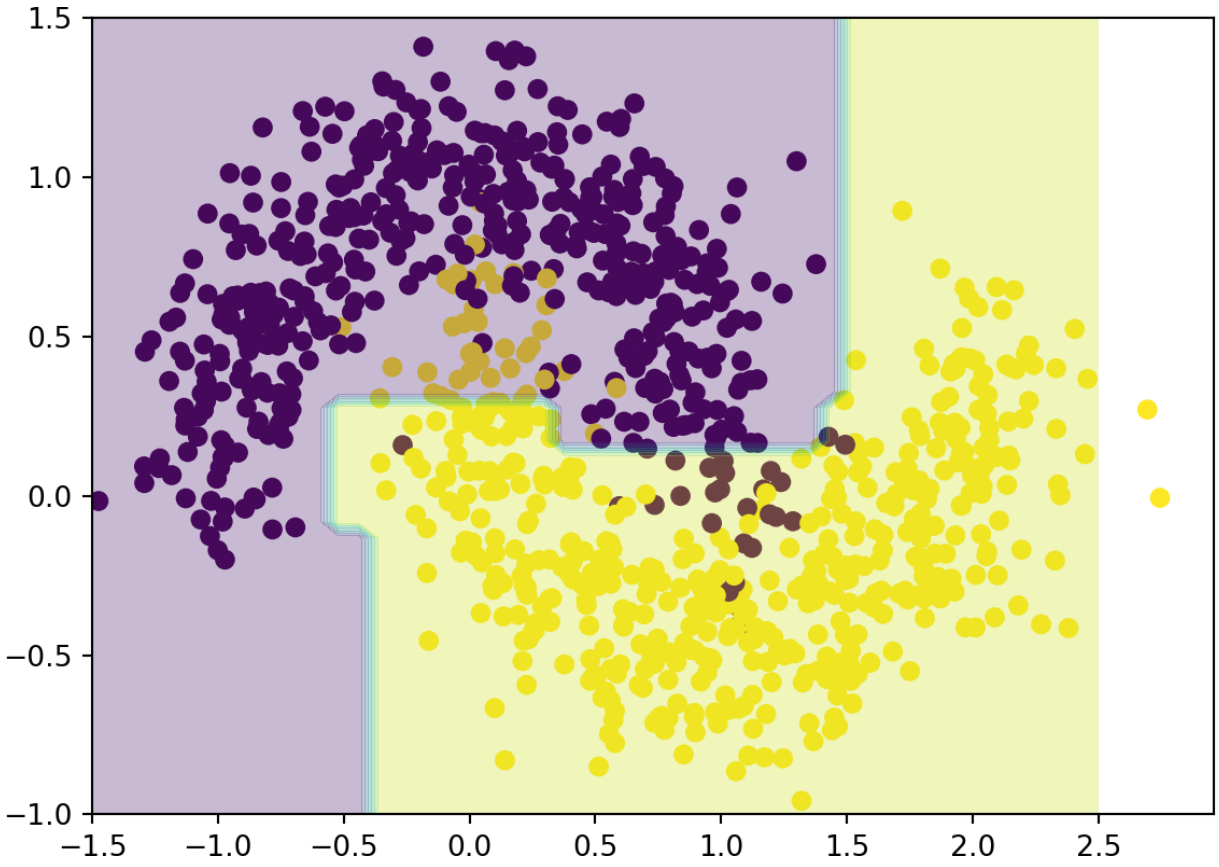
We treat it as a Tr and use **cross-validation** to estimate the test error.

We classify the data using a boosted **CART** (with max. depth $= 3$) and learning rate $\ell = 1/10$ (see the `AdaBoostClassifier()` documentation).

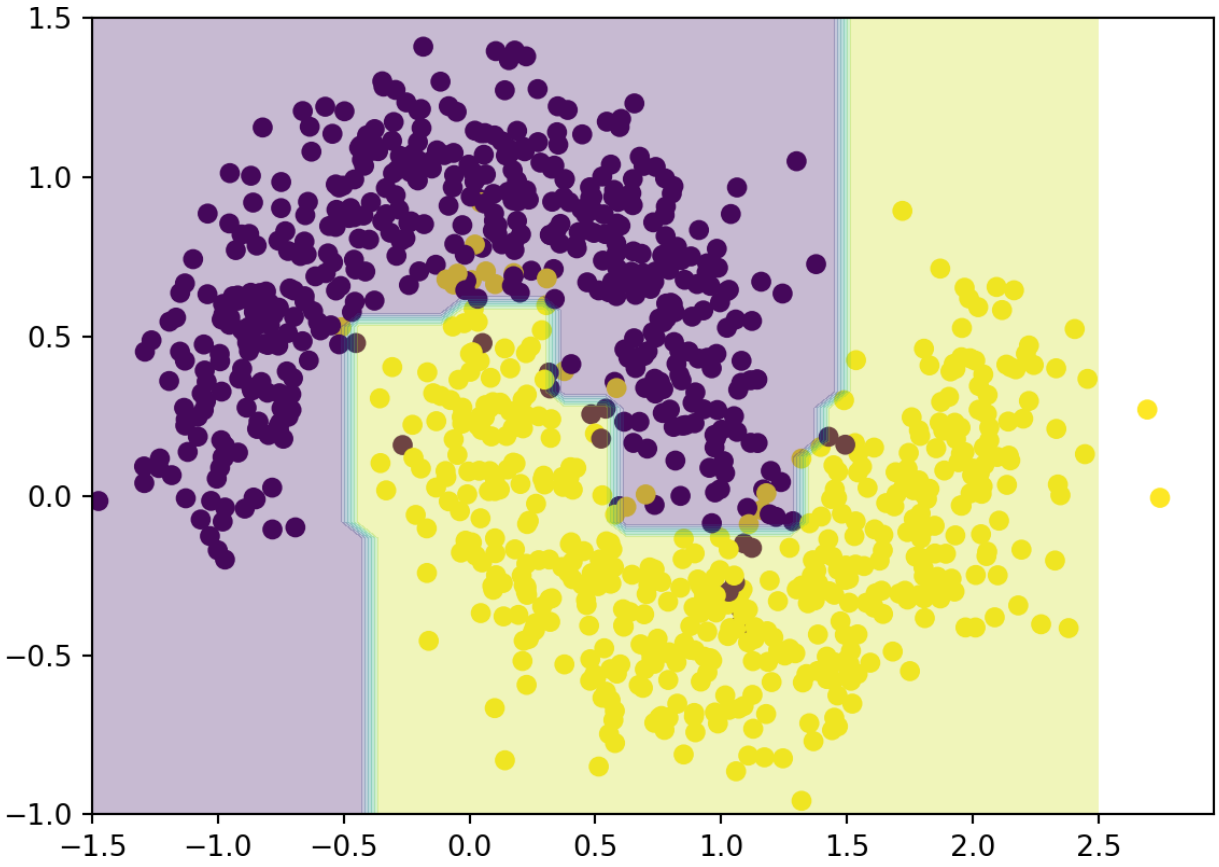
(See DUDADS, 20.5,3 $Boosting$, for code).



Unboosted CART provides a **poor** fit.



AdaBoost CART with $B = 5$ provides a **slightly better** fit.



AdaBoost CART with $B = 10$ provides a **decent** fit.

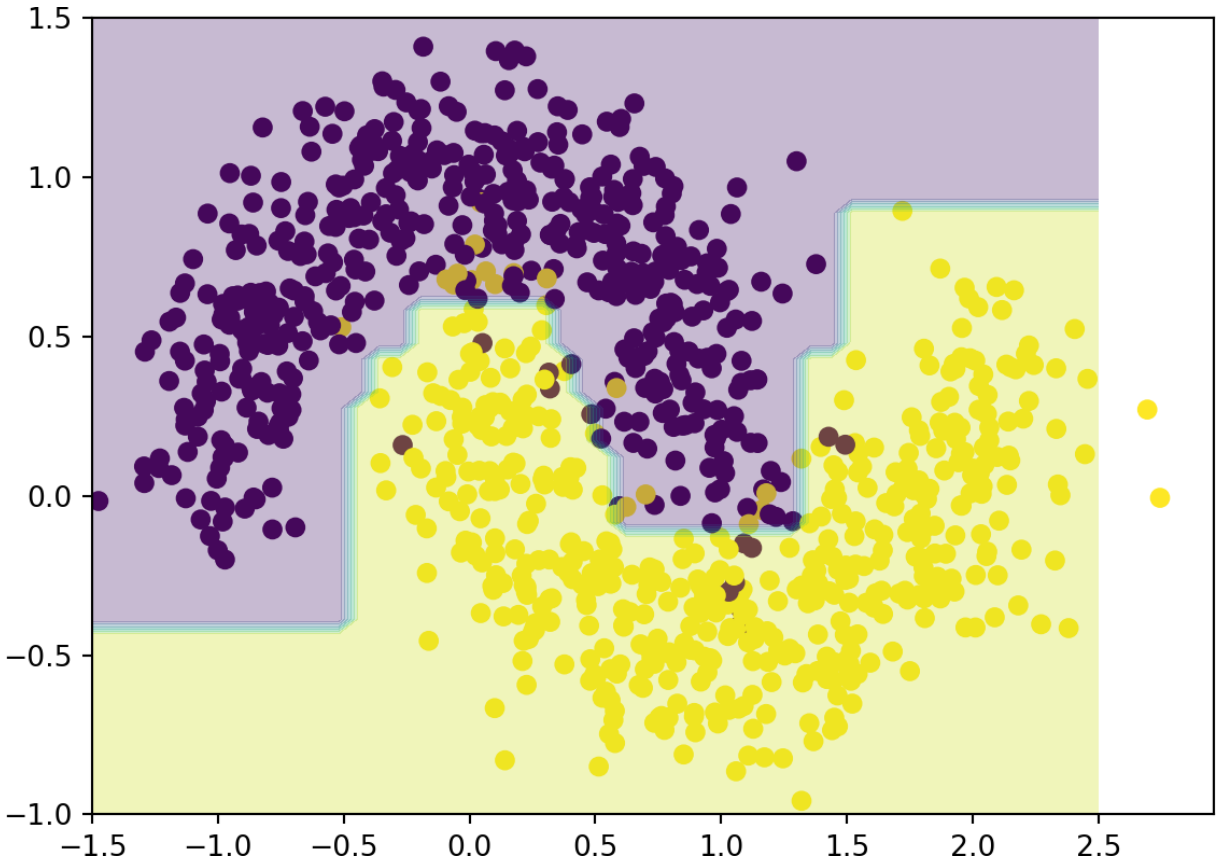
The AdaBoosted tree is better at capturing the dataset's **apparent structure**, but we must evaluate on an independent Te to reduce the risk of **overfitting** (AdaBoost is sensitive to outliers and noise).

One solution: adjust the **learning rate** (AdaBoost step size).

We can find optimal values of ℓ and B by conducting a **grid search** (cross-validation on a grid of parameters).

For the given seed, the parameters that provide the best cross-validation fit for the data are $\ell = 1/20$ and $B = 200$ (these could change with a different seed, or with a different grid search parameters, or a different implementation, etc.).

The plot of the model with these parameters indeed shows that the fit is quite acceptable.



AdaBoost CART with $B = 200$ and $\ell = 1/20$ minimizing the CV error.

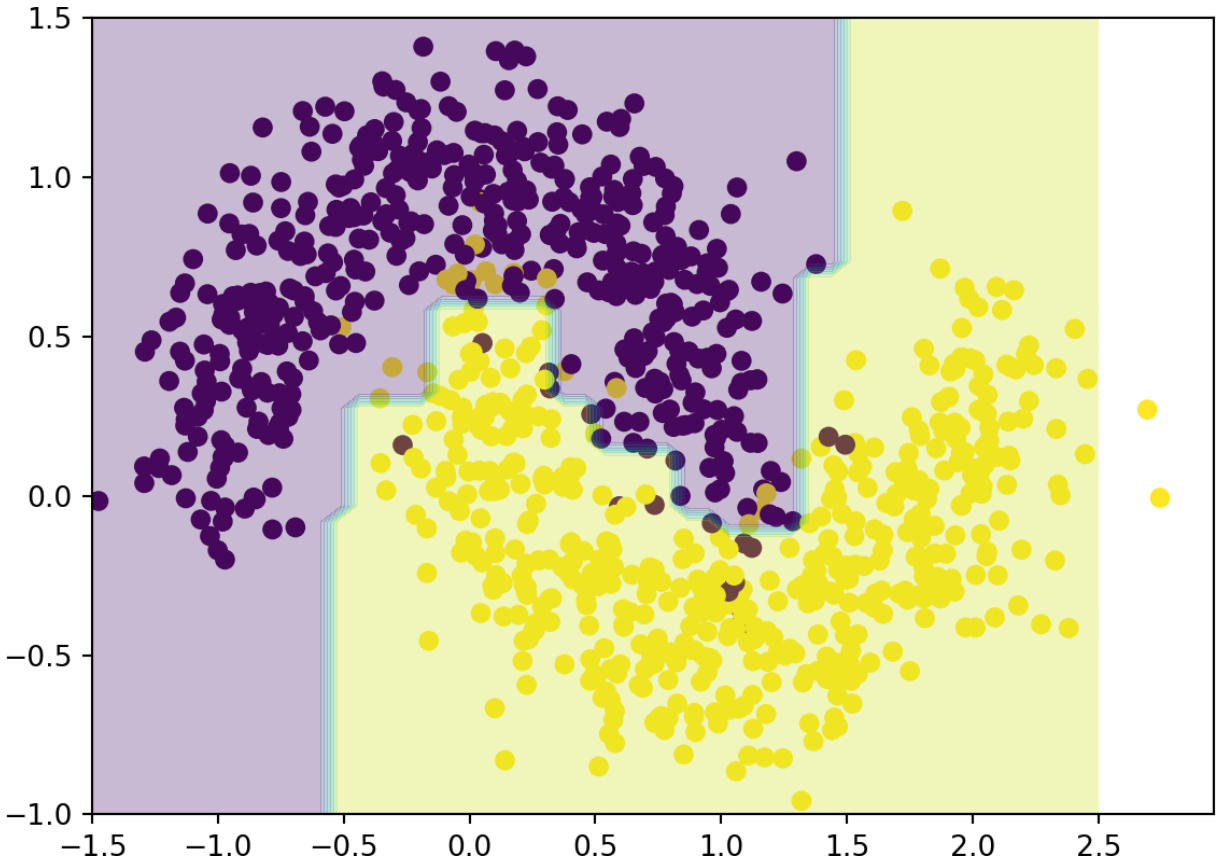
Gradient Boosting

Gradient boosting (GB) "**learns from its mistakes**" (as does AdaBoost):

1. fit a **weak learner** to the data and compute the **residuals** generated by this model;
2. train the **weak learner** on the **residuals**;
3. add the resulting model to the first one (with a **learning rate**)
4. train the **weak learner** on the **residuals** of the **combined** model
5. repeat steps 3-4 a "sufficient" number of times

The final GB model is a (weighted) sum of the initial learner and all the models trained on the **chain of residuals**.

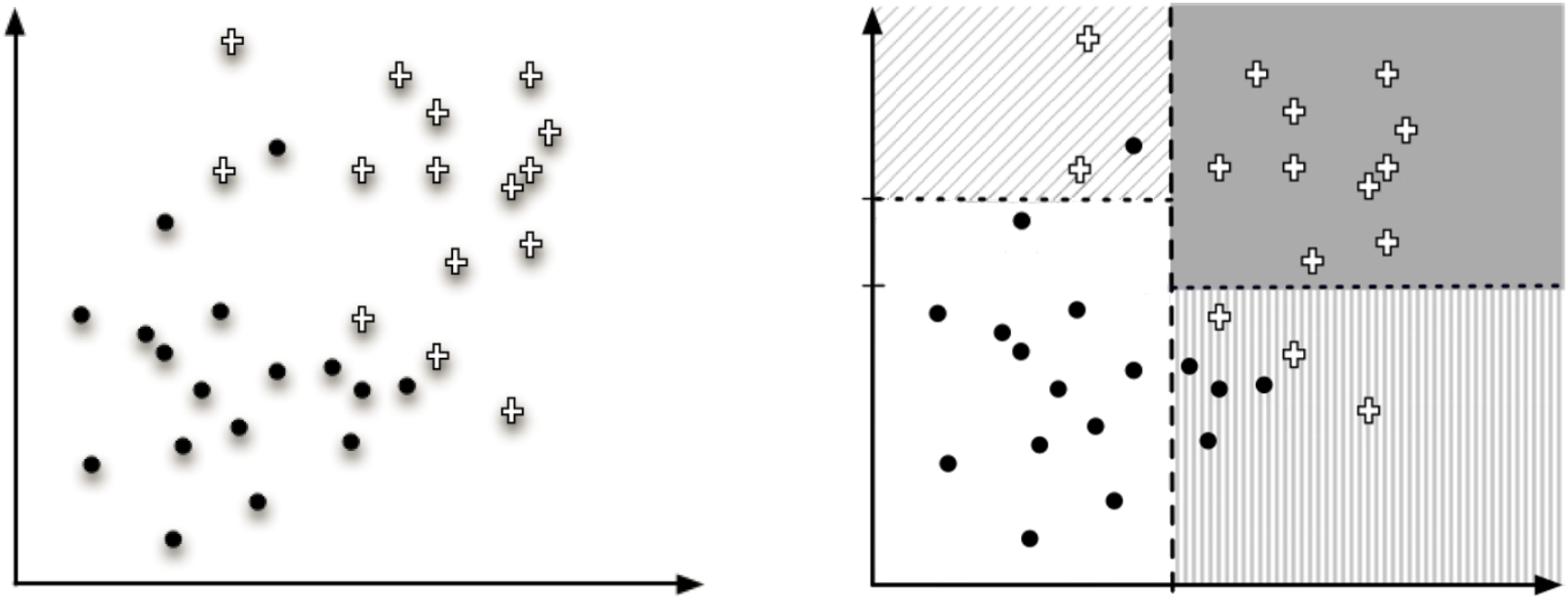
(See https://www.data-action-lab.com/2019/07/31/boosting-with-adaboost-and-gradient-boosting/ for details).



Gradient boosted CART classifier on the Two-Moons dataset.

3.8 – Support Vector Machines

This next classifier is more **sophisticated**, from a mathematical perspective.



Two-class artificial dataset (left) and classification tree (right).

The artificial data has 3 features: X_1 and X_2 (**numerical**), Y (**categorical**, represented by different symbols).

In the **classification tree** on the right, two of the leaves are **pure**, but the risk of **misclassification** is fairly large in the other 2.

The tree is not **unique**, but any other tree with separators parallel to the axes will only be **marginally better**, at best.

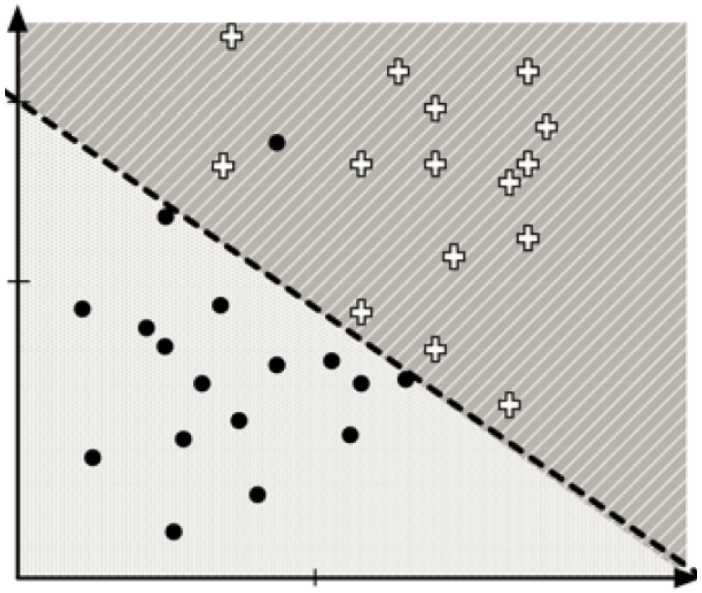
Without access to more features, that tree is as good as it gets ...

... although we could create an **intricate** decision tree with more than $2^2 = 4$ separating lines, but that is undesirable for a well-fitted tree.

Support vector machines (SVM) attempt to find **hyperplanes** that separate the **classes** in the feature space.

3.8.1 – Separating Hyperplanes

But it is easy to draw a **decision curve** which improves on the effectiveness of the decision tree, with a single **misclassified observation** (overfitting?).



Separator on a two-class artificial dataset.

Separating hyperplanes do not always exist; we may need to:

- extend our notion of **separability**, and/or
- extend the **feature space** so classification becomes possible.

A **hyperplane** $H_{\boldsymbol{\beta},\beta_0} \subseteq \mathbb{R}^p$ is a "**flat**" subset of $\mathbb{R}^p$, with

$$\dim\left(H_{\boldsymbol{\beta},\beta_0}\right) = p - 1;$$

in other words, $H_{\boldsymbol{\beta},\beta_0}$ can be described by

$$H_{\boldsymbol{\beta},\beta_0} : \beta_0 + \boldsymbol{\beta}^\top \mathbf{x} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0.$$

The vector $\boldsymbol{\beta}$ is **normal** to $H_{\boldsymbol{\beta},\beta_0}$.

If $\beta_0 = 0$, $H_{\boldsymbol{\beta},\beta_0}$ goes **through the origin in** $\mathbb{R}^p$.

Set $F(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}$; $F(\mathbf{x}) > 0$ for points on one "side" of $H_{\boldsymbol{\beta},\beta_0}$ and $F(\mathbf{x}) < 0$ for points on the other. Note that $F(\mathbf{x}) = 0$ for points on $H_{\boldsymbol{\beta},\beta_0}$.

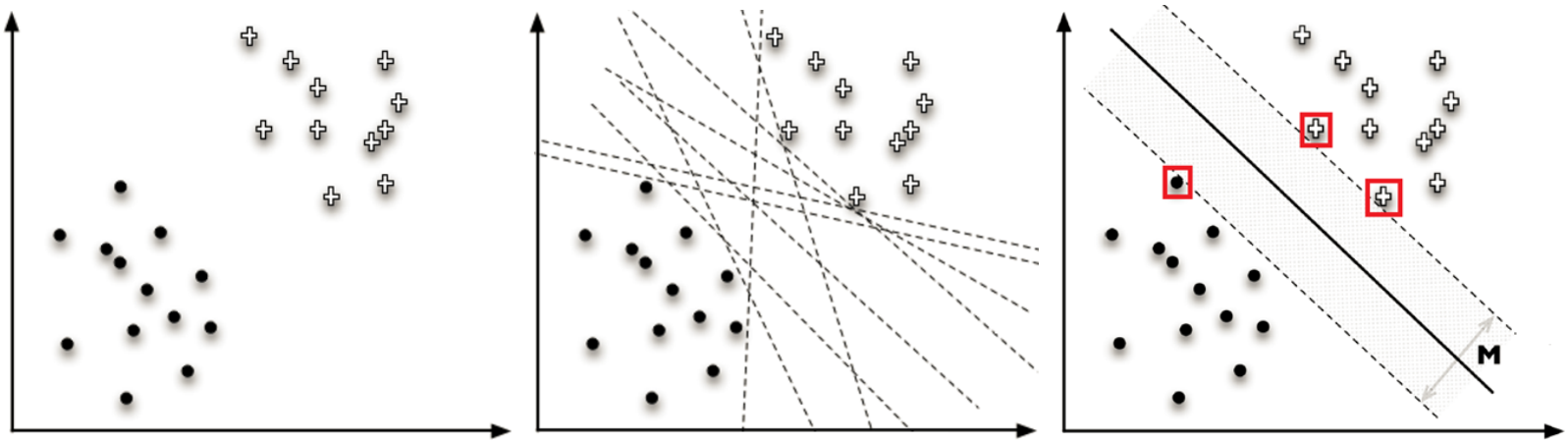
In a **binary classification** problem with $\mathcal{C} = \{C_1, C_2\} = \{\pm 1\}$, if

$$y_i F(\mathbf{x}_i) > 0, \qquad \text{for all } (\mathbf{x}_i, y_i) \in \mathrm{Tr}$$

(or, $y_i F(\mathbf{x}_i) < 0$ for all $(\mathbf{x}_i, y_i) \in \mathrm{Tr}$), then $F(\mathbf{x}_i) = 0$ determines a **separating hyperplane** for Tr (which does not need to be unique), and we say that Tr is **linearly separable**.

Among all separating hyperplanes, the one which provides the **widest separation** between the two classes is the **maximal margin hyperplane**.

Training observations on the boundary of the **separating strip** (or those in the **strip** itself) are the data's **support vectors**.



Artificial linearly separable subset of a two-class dataset (left), with separating hyperplanes (centre), maximal margin hyperplane with support vectors (right)

3.8.2 – Formulation

The classification problem simplifies to **constrained optimization**:

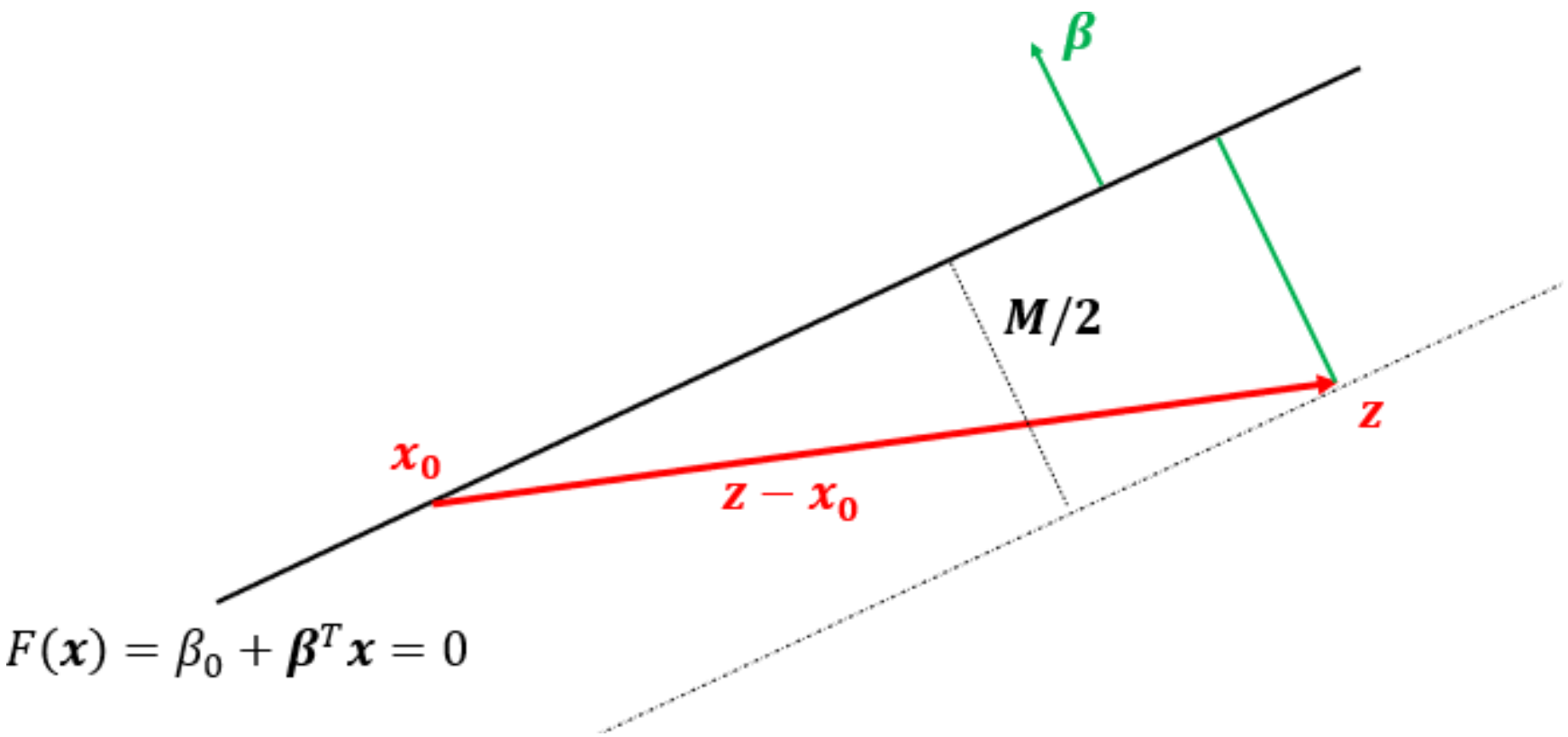
$$(\boldsymbol{\beta}^*, \beta_0^*) = \arg\max_{(\boldsymbol{\beta},\beta_0)} \{M_{(\boldsymbol{\beta},\beta_0)}\} \quad \text{s.t.} \quad y_i(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i) \geq M_{(\boldsymbol{\beta},\beta_0)}$$

for all $(\mathbf{x}_i, y_i) \in \mathrm{Tr}$, with MMH given by $F(\mathbf{x}) = \beta_0^* + \boldsymbol{\beta}^*\mathbf{x} = 0$.

The **canonical representation** of the MMH is the one for which $|F(\mathbf{x}^*)| = 1$ for all support vectors $\mathbf{x}$.

The distance from the canonical MMH $H_{\boldsymbol{\beta},\beta_0}$ to any point $\mathbf{z}$ can be computed using **vector projections**.

Let $\mathbf{x}_0$ be a point on MMH: $F(\mathbf{x}_0) = \beta_0 + \boldsymbol{\beta}^\top\mathbf{x}_0 = 0$.


β
$M/2$
x_0
$z - x_0$
z
$F(x) = \beta_0 + \beta^T x = 0$

In particular, note that $\beta_0 = -\boldsymbol{\beta}^\top \mathbf{x}_0$. Then,

$$\frac{M}{2} = \operatorname{dist}\left(\mathbf{z}, H_{\boldsymbol{\beta},\beta_0}\right) = \left\|\operatorname{proj}_{\boldsymbol{\beta}}(\mathbf{z} - \mathbf{x}_0)\right\| = \left\|\frac{\boldsymbol{\beta}^\top(\mathbf{z} - \mathbf{x}_0)}{\|\boldsymbol{\beta}\|^2}\boldsymbol{\beta}\right\|$$

$$= \frac{|\boldsymbol{\beta}^\top(\mathbf{z} - \mathbf{x}_0)|}{\|\boldsymbol{\beta}\|^2}\|\boldsymbol{\beta}\| = \frac{|\boldsymbol{\beta}^\top \mathbf{z} - \boldsymbol{\beta}^\top \mathbf{x}_0|}{\|\boldsymbol{\beta}\|} = \frac{|F(\mathbf{z})|}{\|\boldsymbol{\beta}\|}.$$

If $\mathbf{z}$ is a **support vector**, then $F(\mathbf{z}) = 1$, and

$$\frac{M}{2} = \operatorname{dist}\left(\mathbf{z}, H_{\boldsymbol{\beta},\beta_0}\right) = \frac{1}{\|\boldsymbol{\beta}\|}.$$

Maximizing the margin M is thus equivalent to **minimizing** $\frac{\|\boldsymbol{\beta}\|}{2}$.

Since the "square function" is **monotonic**,

$$\arg\max_{(\boldsymbol{\beta},\beta_0)} \{M \mid y_i(\beta_0 + \beta^\top \mathbf{x}_i),\ \forall \mathbf{x}_i \in \mathrm{Tr}\}$$

is equivalent to

$$\arg\min_{(\boldsymbol{\beta},\beta_0)} \left\{ \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 \;\middle|\; y_i(\beta_0 + \beta^\top \mathbf{x}_i),\ \forall \mathbf{x}_i \in \mathrm{Tr} \right\}.$$

This QP problem can be solved by **Lagrange multipliers** (or numerically).
Key observation: the **representer theorem** allows us to re-write

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \quad \text{with} \quad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

The original QP becomes

$$\arg\min_{(\boldsymbol{\beta},\beta_0)} \left\{ \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j - \sum_{i=1}^{N} \alpha_i \;\middle|\; \sum_{i=1}^{N} \alpha_i y_i = 0,\ \forall \mathbf{x}_i, \mathbf{x}_j \in \mathrm{Tr} \right\}.$$

It can be shown that **all but** $L \ll N$ of the coefficients α_i are 0. The **support vectors** are those $\mathbf{x}_{i_k} \in \mathrm{Tr}$, $k = 1, \ldots, L$, for which $\alpha_{i_k} \neq 0$.

The **decision function** is defined by

$$T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{k=1}^{L} \alpha_{i_k} y_{i_k} \mathbf{x}_{i_k}^\top \mathbf{x} + \beta_0,$$

scaled so that $T(\mathbf{x}_{i_k}; \alpha) = y_{i_k} = \pm 1$ for each support vector $\mathbf{x}_{i_k}$.

The **class assignment** for any $\mathbf{x} \in \mathrm{Te}$ is thus

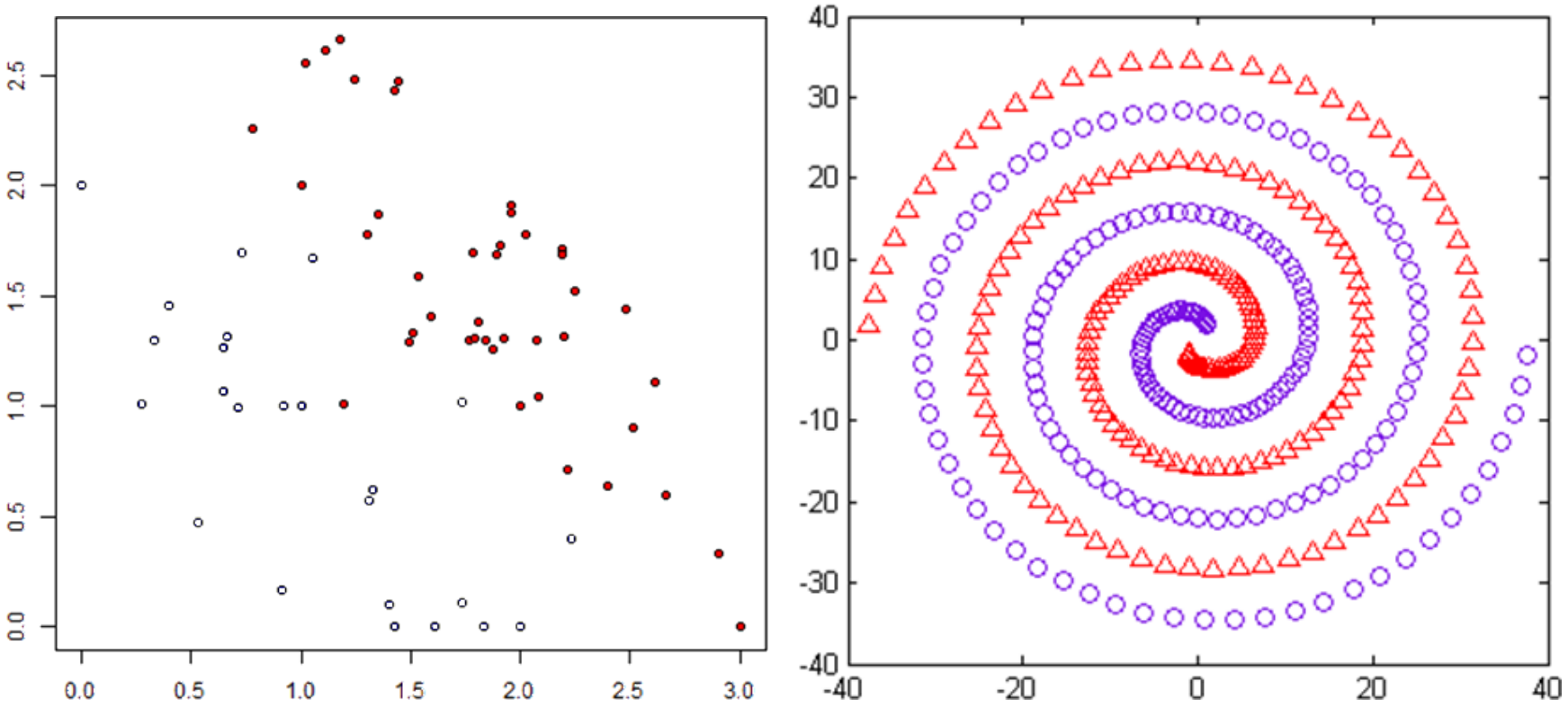
$$\mathrm{class}(\mathbf{x}) = \begin{cases} +1 & \text{if } T(\mathbf{x}; \alpha) \geq 0 \\ -1 & \text{if } T(\mathbf{x}; \alpha) < 0 \end{cases}$$

In practice (especially when $N < p$), the data is rarely **linearly separable** into distinct classes.

But even when it is, the data may be **noisy**, which could lead to **overfitting** $\Longrightarrow$ technically **optimal** but practically **sub-optimal** MM solutions.

How can we overcome these problems?

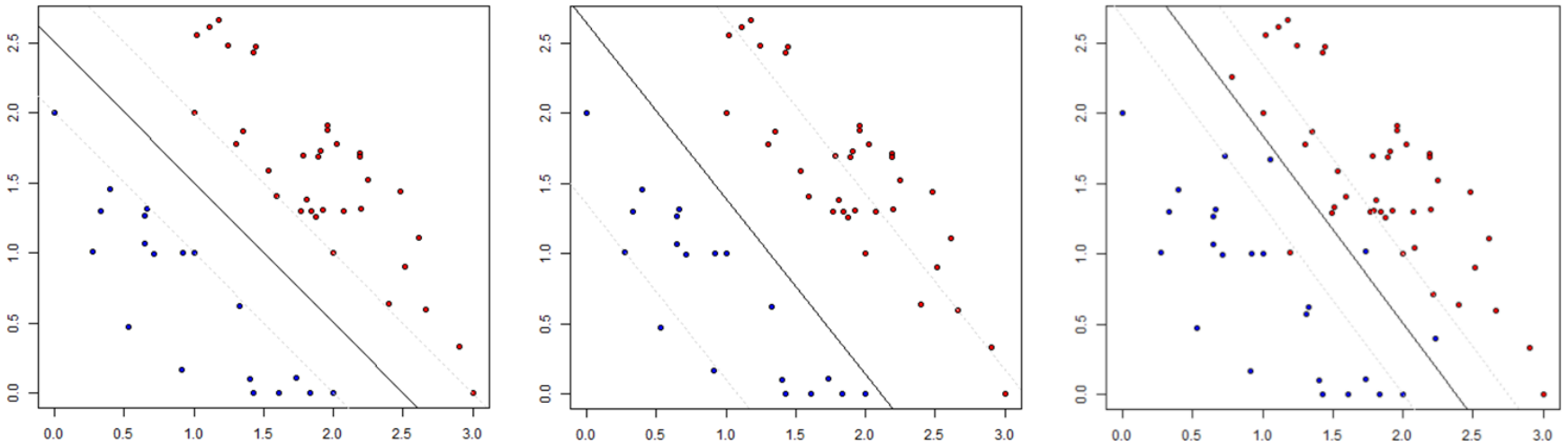
With **soft margins** and the **kernel trick**.



nonlinearly separable two-class datasets.

3.8.3 – Soft Margins

In applications, support vector classifiers optimize instead a **soft margin**, for which some **misclassifications** are permitted.



Hard margin for a linearly separable classifier (left); soft margin for a linearly separable classifier (middle); soft margin for a non linearly separable classifier (right).

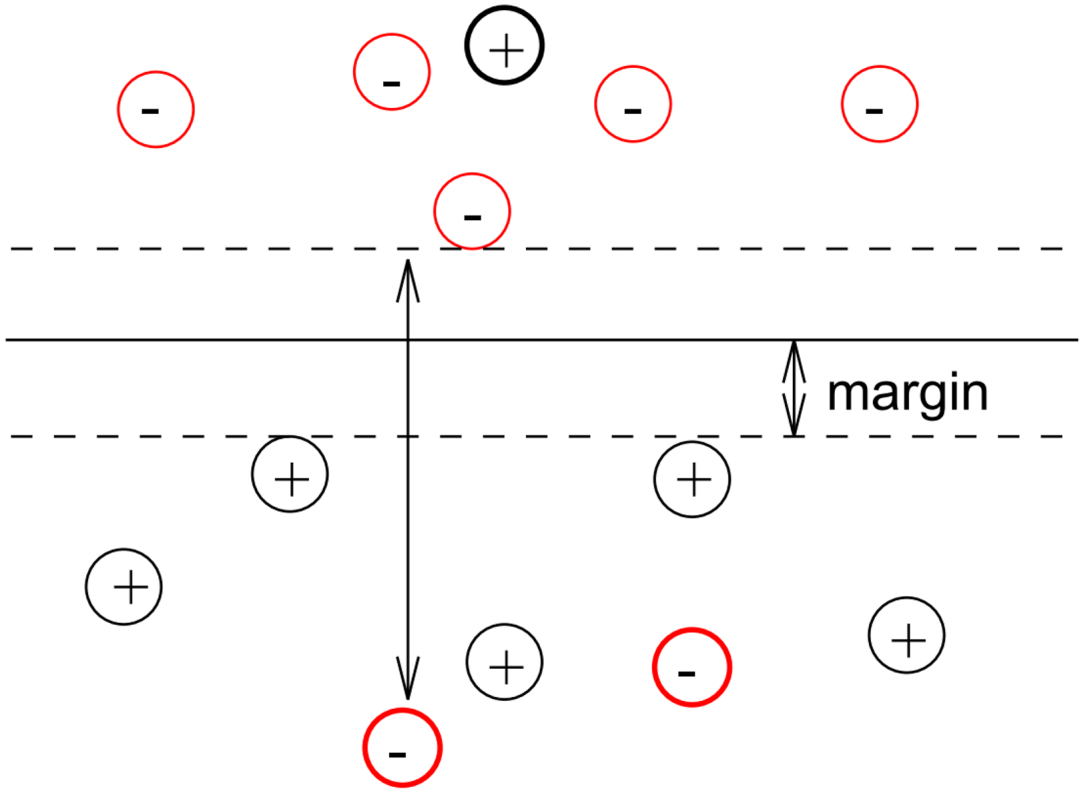
The **soft margin** problem can be written as

$$\arg\min_{(\boldsymbol{\beta},\beta_0)} \left\{ \frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} \;\middle|\; y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) \geq 1 - \varepsilon_i,\ \varepsilon_i \geq 0, \forall \mathbf{x}_i \in \mathrm{Tr},\ \|\boldsymbol{\varepsilon}\|_1 < C \right\},$$

where C is a **(budget) tuning parameter**, $\boldsymbol{\varepsilon}$ is a vector of **slack variables**, scaled so that $|F(\mathbf{x}^*)| = |\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}^*| = 1$ for any support vector $\mathbf{x}^*$.

Soft margin models are **robust** against anomalies (and reasonably accurate):

- if $\varepsilon_i = 0$, then $\mathbf{x}_i \in \mathrm{Tr}$ is **correctly classified**; it falls on the correct side of the hyperplane, and outside the maximum margin.
- if $0 < \varepsilon_i < 1$, then $\mathbf{x}_i \in \mathrm{Tr}$ is **acceptably classified**; it falls on the correct side of the hyperplane, but within the margin.
- if $\varepsilon_i \geq 1$, it is **incorrectly classified**; it falls on the wrong side of the hyperplane.


margin

$C = 0 \implies$ **no violations are allowed** ($\|\varepsilon\| = 0$) $\implies$ problem reduces to **hard margin SVM** (no solution if data is **not linearly separable**).

$N \geq C > 0 \in \mathbb{N}^{\times} \implies$ **at most** C obs. $\mathbf{x}_i \in \mathrm{Tr}$ are **misclassified**: if $i_1, \ldots, i_C$ are the **misclassified** indices, then $\varepsilon_{i_1}, \ldots, \varepsilon_{i_C} \geq 1$ and

$$\underbrace{C \geq \sum_{i=1}^{N} \varepsilon_i}_{\text{soft margin}} \geq \underbrace{\sum_{k=1}^{C} \varepsilon_{i_k} \geq C}_{\text{misclassification}}.$$

As C **increases**, tolerance for violations also **increases**, as does the **width** of the soft margin.

C plays the role of a **regularization parameter**, and is usually selected *via* **cross-validation**.

Small C $\Longrightarrow$ **hard** margins $\Longrightarrow$ **low** bias + **high** variance $\Longrightarrow$ small changes in the data may create **qualitatively different** margins.

Large C $\Longrightarrow$ **wide** and **soft** margins $\Longrightarrow$ misclassifications and **high** bias + **low** variance $\Longrightarrow$ small changes in the data are **unlikely** to change the margin significantly.

We can build a classifier through the representer theorem formulation as before, the only difference being that the **decision function** $T(\mathbf{x}; \boldsymbol{\alpha})$ is scaled so that $|T(\mathbf{x}_{i_k}; \boldsymbol{\alpha})| \geq 1 - \varepsilon_{i_k}$ for **every support vector** $\mathbf{x}_{i_k}$.

Finding the value of the regularization parameter C is difficult; an optimal value is obtained *via* a **tuning process**, which tries out various values and identifies the one that produces an optimal model.

Example: Gapminder Dataset

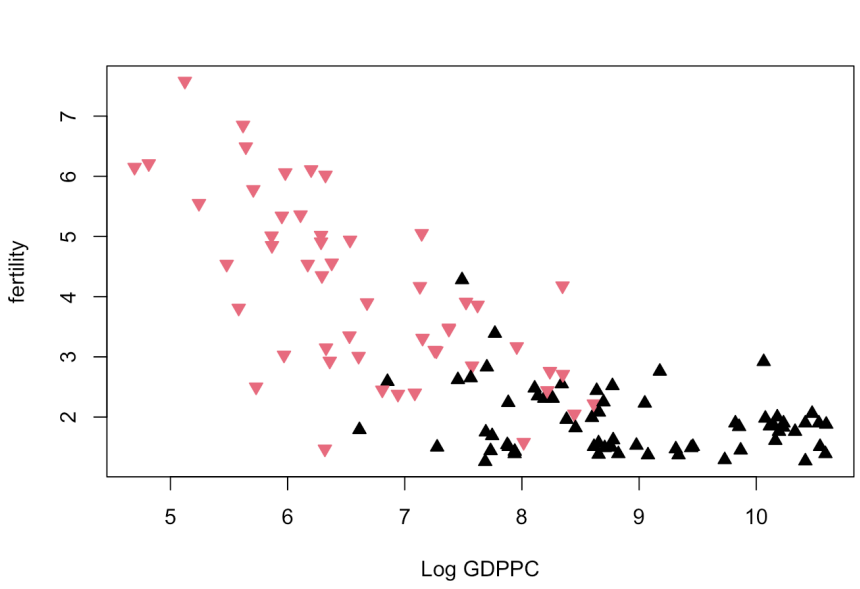
We illustrate these concepts using an example (DUDADS, 21.4.2, *Support Vector Machines*).

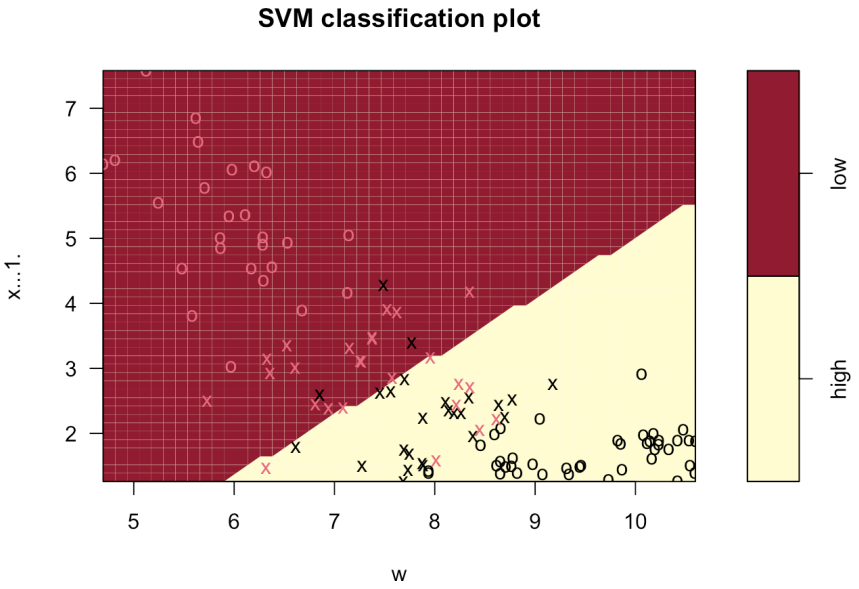
We train a SVM with $C = 0.1$ (obtained *via* a tuning procedure for C) for the 2011 `Gapminder` dataset to predict the **life expectancy** class `LE` Y in terms of the **fertility rate** X_1 and the **logarithm of GDP per capita** X_2; $n = 116$ observations are used for Tr, the rest are set aside in Te.

We run 7 SVM models: one each with cost parameters

$$C = 0.001, \quad 0.01, \quad 0.1, \quad 1, \quad 5, \quad 10, \quad 100.$$

A **tuning procedure** suggests using $C = 0.1$.





The confusion matrix of the model on Te is:

$C = 0.01$		**prediction** 0	1
actual	0	22	10
	1	1	17

3.8.4 – Nonlinear Boundaries and Kernels

If the **boundary** between two classes is **linear**, the SVM classifier described previously is a **natural way** to try to separate the classes.

In practice, however, the classes are **rarely** cleanly separated.

In both **hard** and **soft** margin SVM, the **objective** and **decision** functions take, respectively, the form

$$\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 = \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j - \sum_{i=1}^{N} \alpha_i; \quad T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{k=1}^{L} \alpha_{i_k} y_{i_k} \mathbf{x}_{i_k}^\top \mathbf{x} + \beta_0.$$

We do not **actually** need to know the **support vectors** $\mathbf{x}_{i_k}$ (or even the **observations** $\mathbf{x}_i$) to compute the decision function **values** – we only need access to the **inner products** $\mathbf{x}_i^\top \mathbf{x}_j$ or $\mathbf{x}_{i_k}^\top \mathbf{x}$, which are also denoted by $\langle \mathbf{x}_{i_k}, \mathbf{x} \rangle$ or $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$.

The **objective** and **decision** functions can thus be written as

$$\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 = \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^{N} \alpha_i; \quad T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{k=1}^{L} \alpha_{i_k} y_{i_k} \langle \mathbf{x}_{i_k}, \mathbf{x} \rangle + \beta_0.$$

The **kernel approach** replaces **inner products** by **kernels** (or **generalized inner products**):

$$\langle \mathbf{x}, \mathbf{w} \rangle \rightsquigarrow K(\mathbf{x}, \mathbf{w}).$$

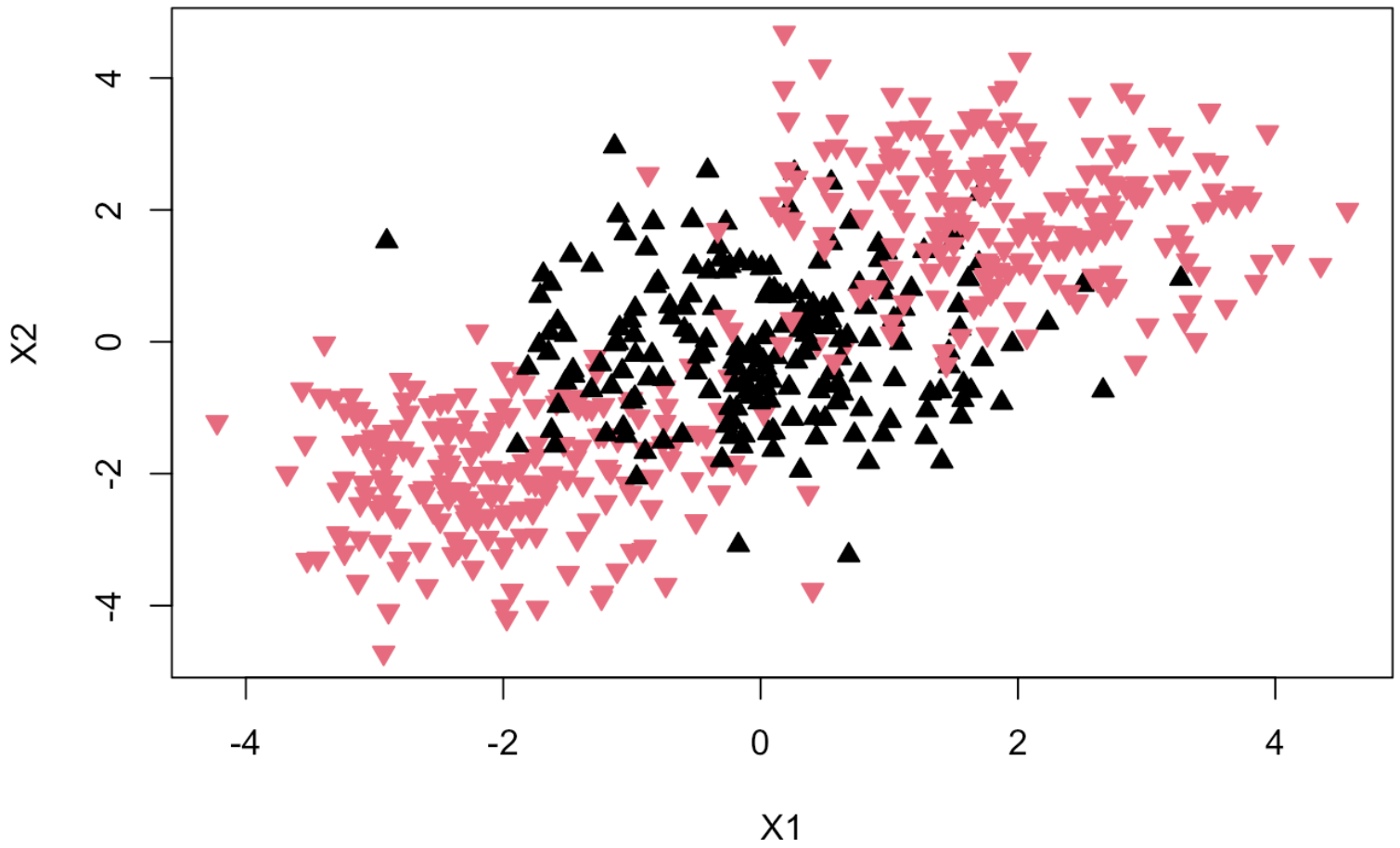
Kernels provide a measure of **similarity** between the observations $\mathbf{x}$ and $\mathbf{w}$ (as do **inner products**).

A **kernel** is a symmetric (semi-)positive definite operator $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_0^+$.
By analogy with **positive definite square matrices**, we have

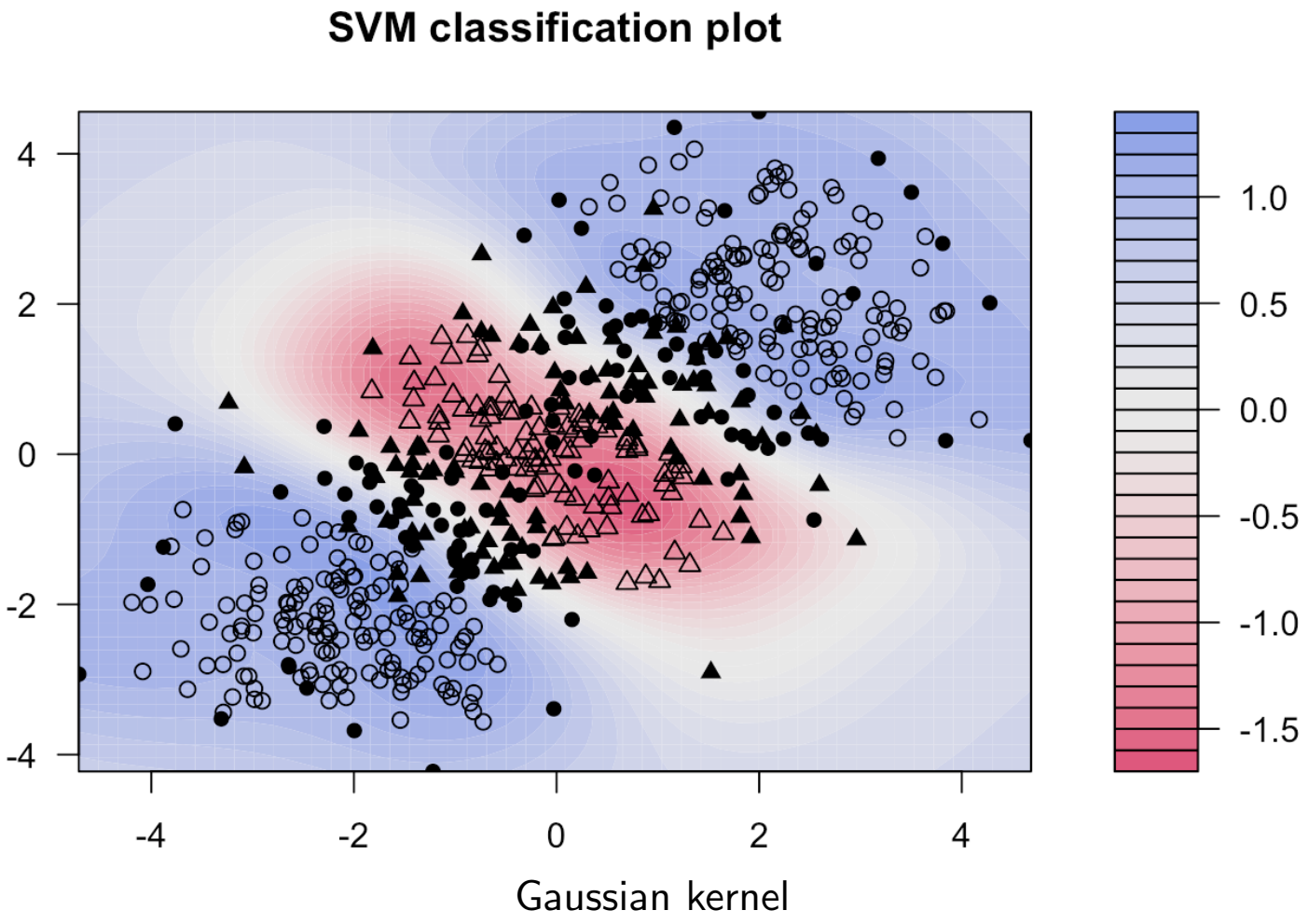
$$\sum_{i,j=1}^{N} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \quad \forall \mathbf{x}_i \in \mathbb{R}^p, c_j \geq 0.$$

Common ML kernels include:

- **linear** – $K(\mathbf{x}, \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$;
- **polynomial of degree** d – $K_d(\mathbf{x}, \mathbf{w}) = (1 + \mathbf{x}^\top \mathbf{w})^d$;
- **Gaussian** (or radial) – $K_\gamma(\mathbf{x}, \mathbf{w}) = \exp(-\gamma \|\mathbf{x} - \mathbf{w}\|_2^2)$, $\gamma > 0$;
- **sigmoid** – $K_{\kappa,\delta}(\mathbf{x}, \mathbf{w}) = \tanh(\kappa \mathbf{x}^\top \mathbf{w} - \delta)$, for allowable κ, δ.



Nonlinearly-separable artificial data


SVM classification plot
4
2
0
-2
-4
-4
-2
0
2
4
Gaussian kernel
1.0
0.5
0.0
-0.5
-1.0
-1.5

The **decision boundary** is computed in the same way as with **linear** SVM: the objective and decision functions are

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)-\sum_{i=1}^{N}\alpha_i; \quad T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}K(\mathbf{x}_{i_k},\mathbf{x})+\beta_0.$$

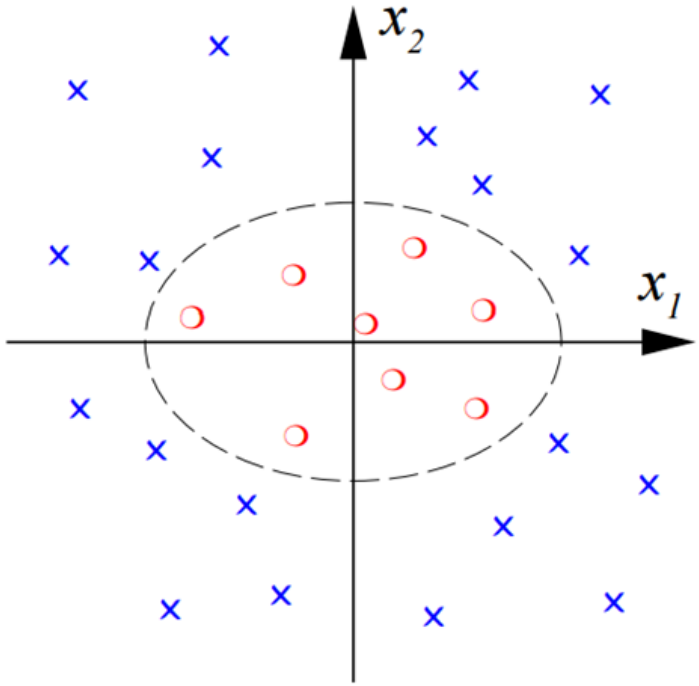
For the **radial kernel**:

- for instance, if a test observation $\mathbf{x}$ is **near** a training observation $\mathbf{x}_i$, then $\|\mathbf{x}-\mathbf{x}_i\|_2^2$ is **small** and $K_\gamma(\mathbf{x},\mathbf{x}_i)\approx 1$;
- otherwise, $\|\mathbf{x}-\mathbf{x}_i\|_2^2$ is **large** and $K_\gamma(\mathbf{x},\mathbf{x}_i)\approx 0$.

In other words, in the radial kernel framework, only those observations **close** to a test observation play a role in **class prediction**.

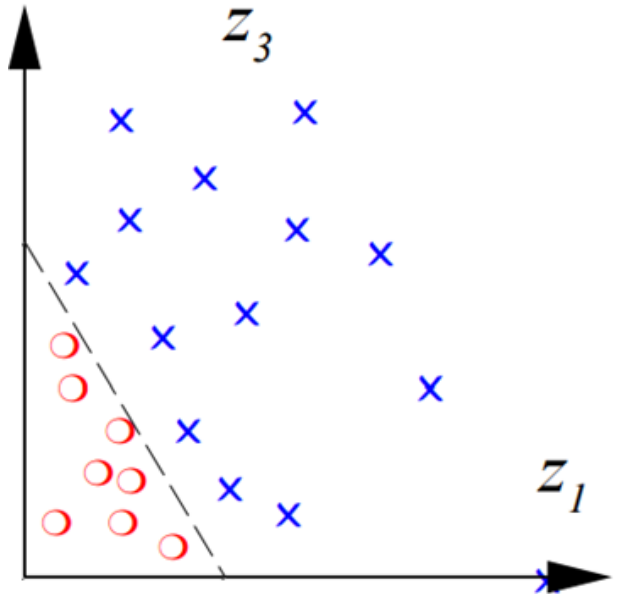
Kernel Trick

Why use kernels in the first place? Not all data sets are **linearly separable**.



The optimal margin separating "strip" is not linear.

One way out of this problem is to introduce a **transformation** Φ from the **original** $X-$feature space to a **higher-dimensional** (or at least, of the **same dimension**) $Z-$feature space in which the data is **linearly separable**, and to build a **linear SVM** on the transformed training observations $\mathbf{z}_i = \Phi(\mathbf{x})_i$.



Projection of the transformed linear problem in $Z-$space.

Does this go against **reduction strategies** to counter **the curse of dimensionality**? **No**; the added dimensions help "**unfurl**" the data.

The objective and decision functions take the form

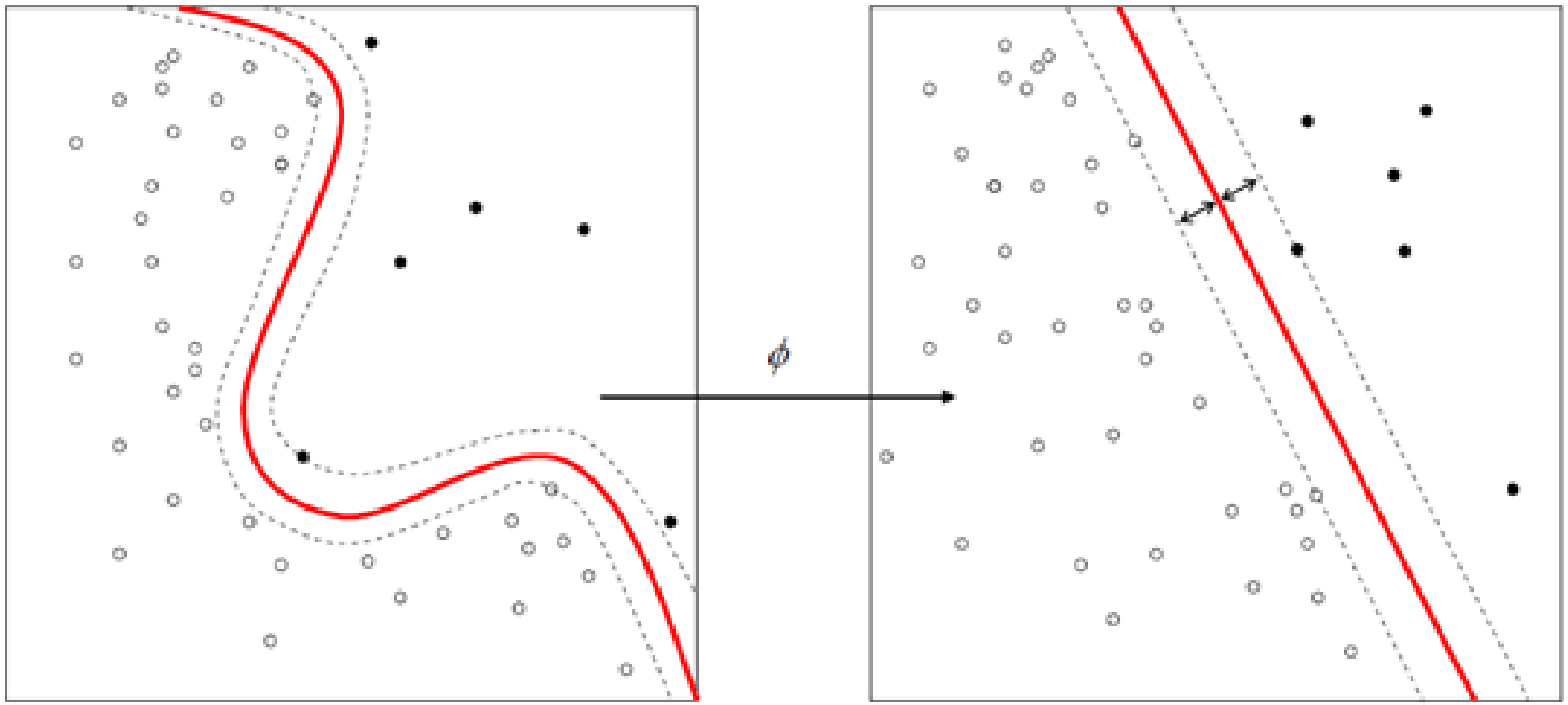
$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j)-\sum_{i=1}^{N}\alpha_i;\quad T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}\Phi(\mathbf{x}_{i_k})^\top\Phi(\mathbf{x})+\beta_0,$$

but the **linear SVM** is built in $Z-$space, not in $X-$space.

In the preceding example, we use some $\Phi:\mathbb{R}^2\to\mathbb{R}^3$; is it obvious that

$$\mathbf{z}=\Phi(\mathbf{x})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)$$

would work?



The **kernel trick** simply states that Φ can remain **unspecified** as long as we replace $\Phi(\mathbf{x})^\top \Phi(\mathbf{w})$ by a "**reasonable**" kernel $K(\mathbf{x}, \mathbf{w})$ (often radial).

3.8.5 – General Classification

What if the response variable has $K > 2$ classes? In the **one-versus-all** (OVA) approach, we fit K different $2-$class SVM decision functions $T_k(\mathbf{x}; \boldsymbol{\alpha})$, $k = 1, \ldots, K$; in each, class k versus the rest. The test observation $\mathbf{x}^*$ is assigned to the class for which $T_k(\mathbf{x}^*; \boldsymbol{\alpha})$ is **largest**.

In the **one-versus-one** (OVO) approach, we fit **all** $\binom{K}{2}$ pairwise $2-$class SVM classifiers $\text{class}_{k,\ell}(\mathbf{x})$, for training observations with class labels k, ℓ, where $k > \ell = 1, \ldots, K-1$. The test observation $\mathbf{x}^*$ is assigned to the class that **wins the most pairwise "contests"**.

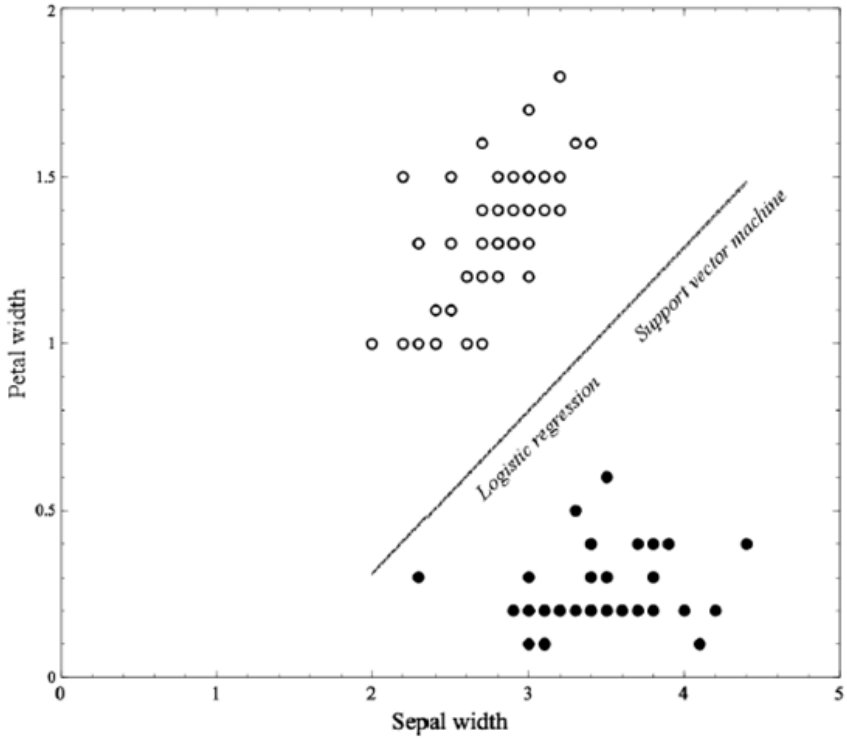
If K is large, $\binom{K}{2}$ might be **too large** to make **OVO** computationally efficient; when it is **small "enough"**, **OVO** is the recommended approach.

Final Comments

It is not always obvious whether one should use SVM or some other model:

- if classes are **(nearly) linearly separable**, SVM and LDA are usually **preferable to** logistic regression (LR);
- if they are not, LR together with a **ridge penalty** is $\approx$ **equivalent** to SVM;
- if the aim is to estimate **class membership probabilities**, it is preferable to use LR as SVM is not **calibrated** – the values of $T(\mathbf{x}; \boldsymbol{\alpha})$ have no intrinsic meaning, other than their **relative ordering**;
- **kernels** can be implemented with LR and LDA, with **increased** computational complexity.

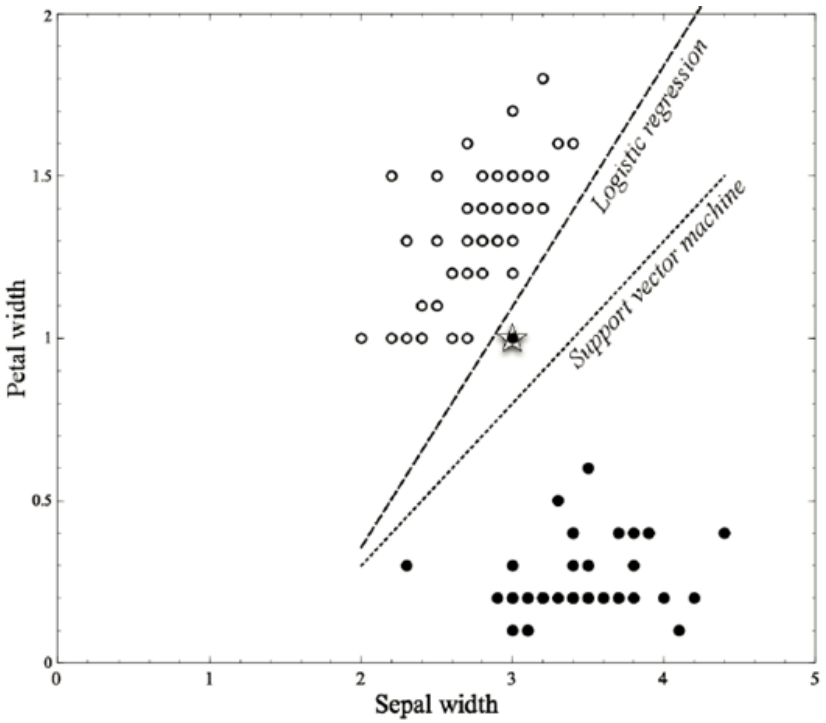
The **No Free Lunch Theorem** remains in effect: there is no **magical recipe**, although the next technique we discuss is often used as one...

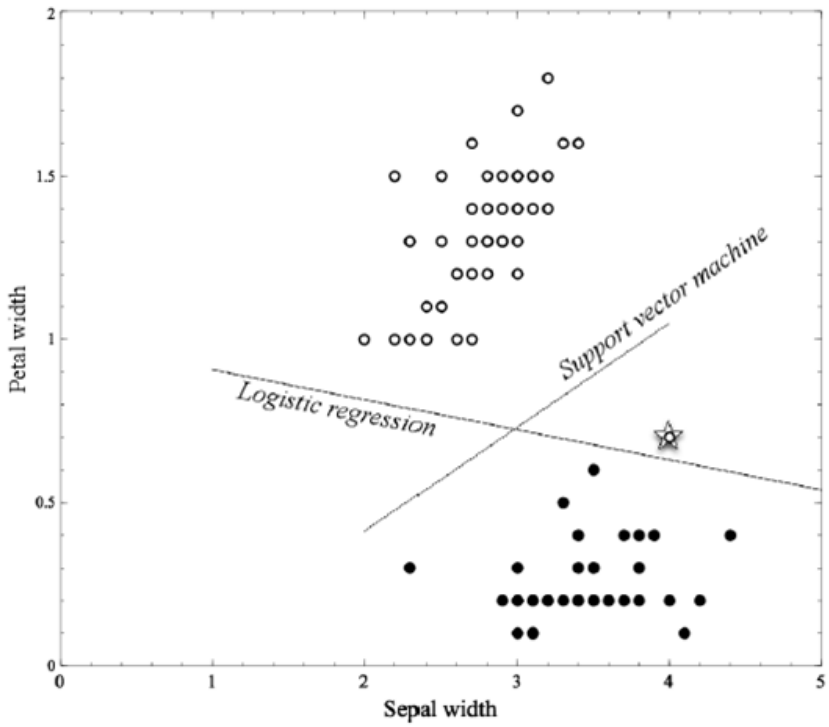


SVMs are well-protected against the risk of **overfitting** (due to the **small number** of support vectors).

In this dataset, the decision boundaries for LR and SVM **coincide**.

What happens when we add observations that leave the **clean separation** in question?





The SVM decision boundary is **smarter** about its reactions to the addition of "**outlying**" observations, thanks to its **soft margin**.

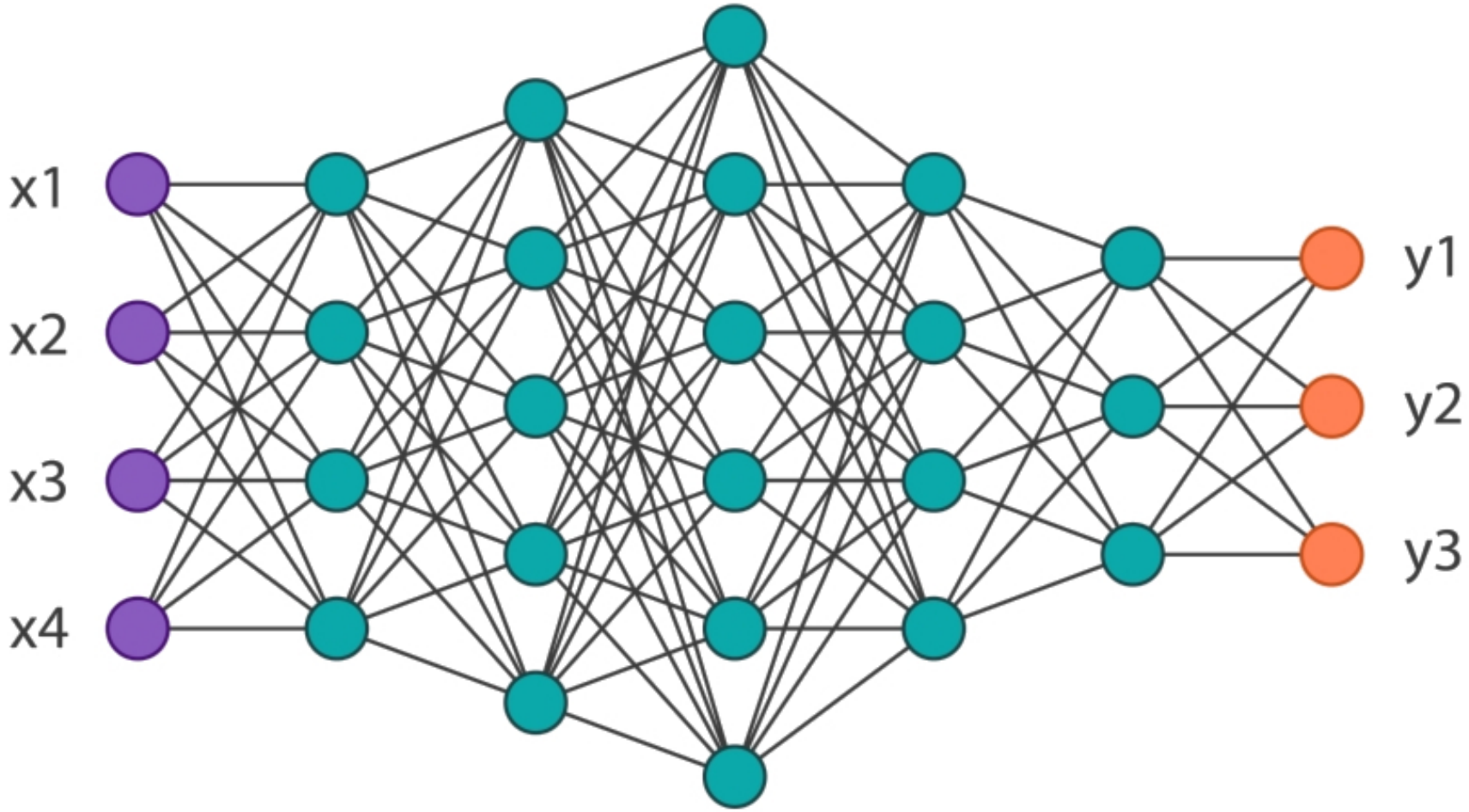
3.9 – Artificial Neural Networks

A trained **artificial neural network** (ANN) is a function that maps inputs to outputs in a useful way:

- it receive input(s);
- it compute values, and
- it provide output(s).

ANNs use a Swiss-army-knife approach to things (there are plenty of options, but **it's not always clear which one should be used**).

The user does not need to decide much about the function or know much about the problem space in advance (it's a **quiet model**).


x1
x2
x3
x4
y1
y2
y3

Algorithms allow ANN to **learn automatically** (i.e., we can generate the function and its internal values from Tr).

ANNs can be used for:

- supervised learning (**multi-layered feedforward neural networks**)
- unsupervised learning (**self-organizing maps**)
- reinforcement learning

Technically, the only requirement is the ability to **minimize** a cost function.

Popular accounts: *Neural Networks Demystified*, Welch Labs; *Neural Networks*, 3blue1brown [YouTube]; A Neural Network Playground, TensorFlow.org.

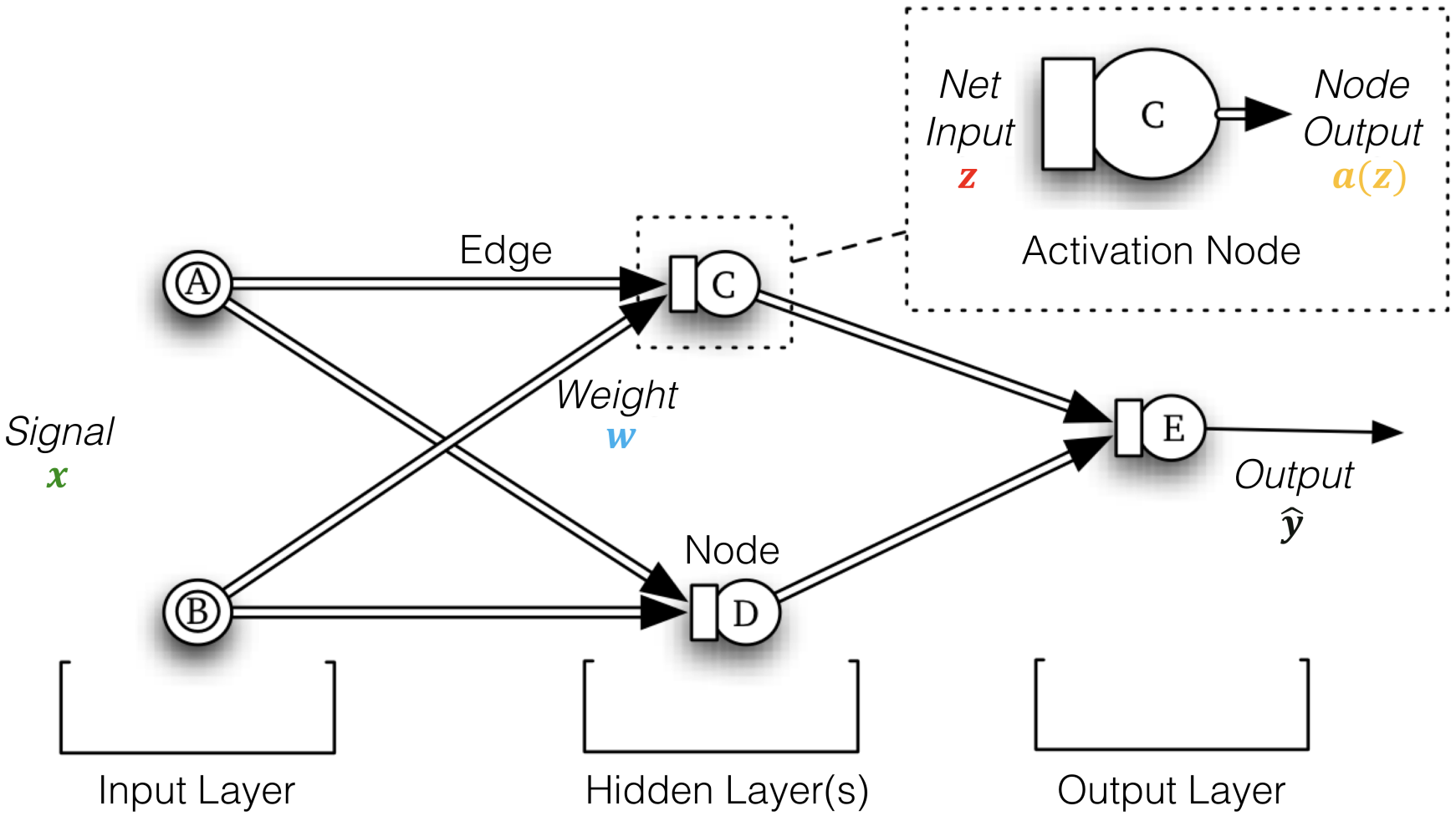
3.9.1 – Network Topology and Terminology

An **artificial neural network** is an interconnected group of **nodes**, inspired by a simplification of neurons in a brain but on much smaller scales.

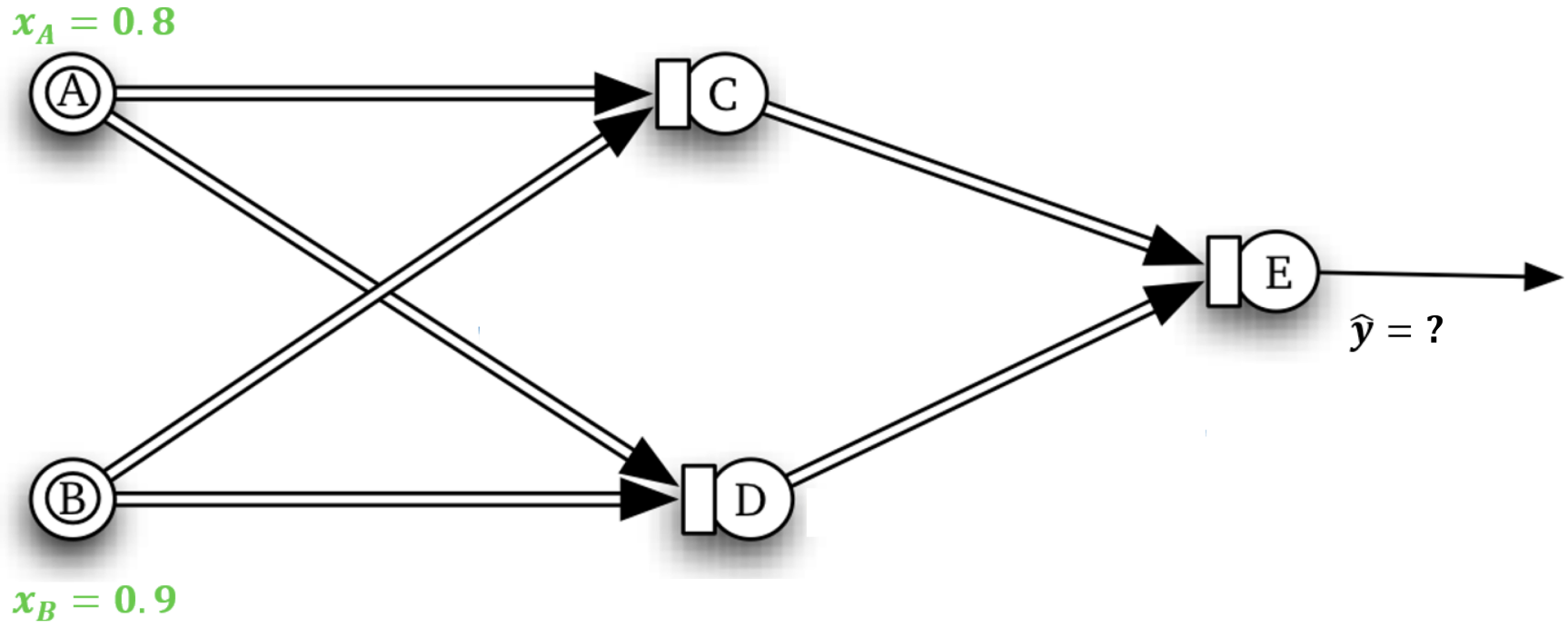
ANN are typically organized in **layers**. Layers are made up of a number of interconnected **nodes** which contain an **activation function**.

A **pattern** $\boldsymbol{x}$ (input, signal) is presented to the network *via* the **input layer**, which communicates with one or more **hidden layers**, where the actual processing is done *via* a system of weighted **connections** $\mathbf{W}$ (**edges**).

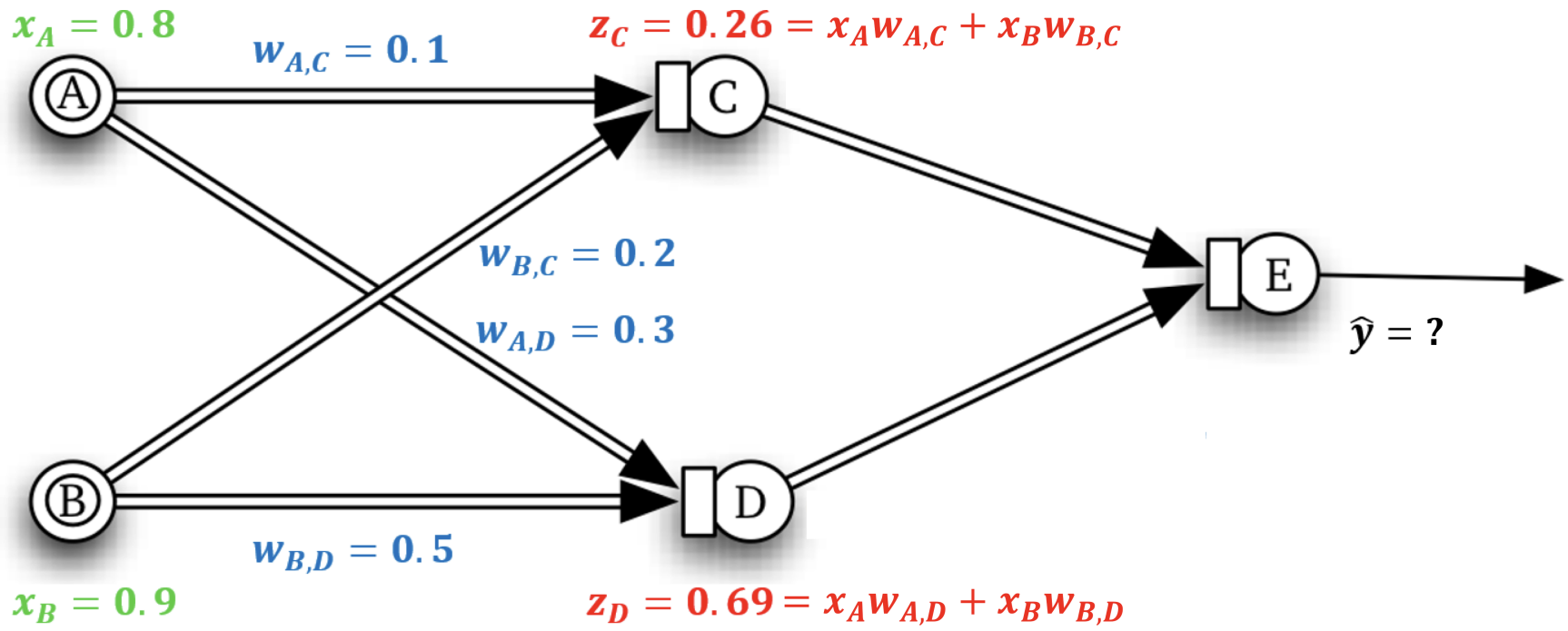
The hidden layers then link to an **output layer**, which outputs the **predicted response** $\hat{\boldsymbol{y}}$.


Net
Input
z
C
Node
Output
a(z)
Activation Node
Edge
A
C
Weight
w
Signal
x
E
Output
ŷ
Node
B
D
Input Layer
Hidden Layer(s)
Output Layer

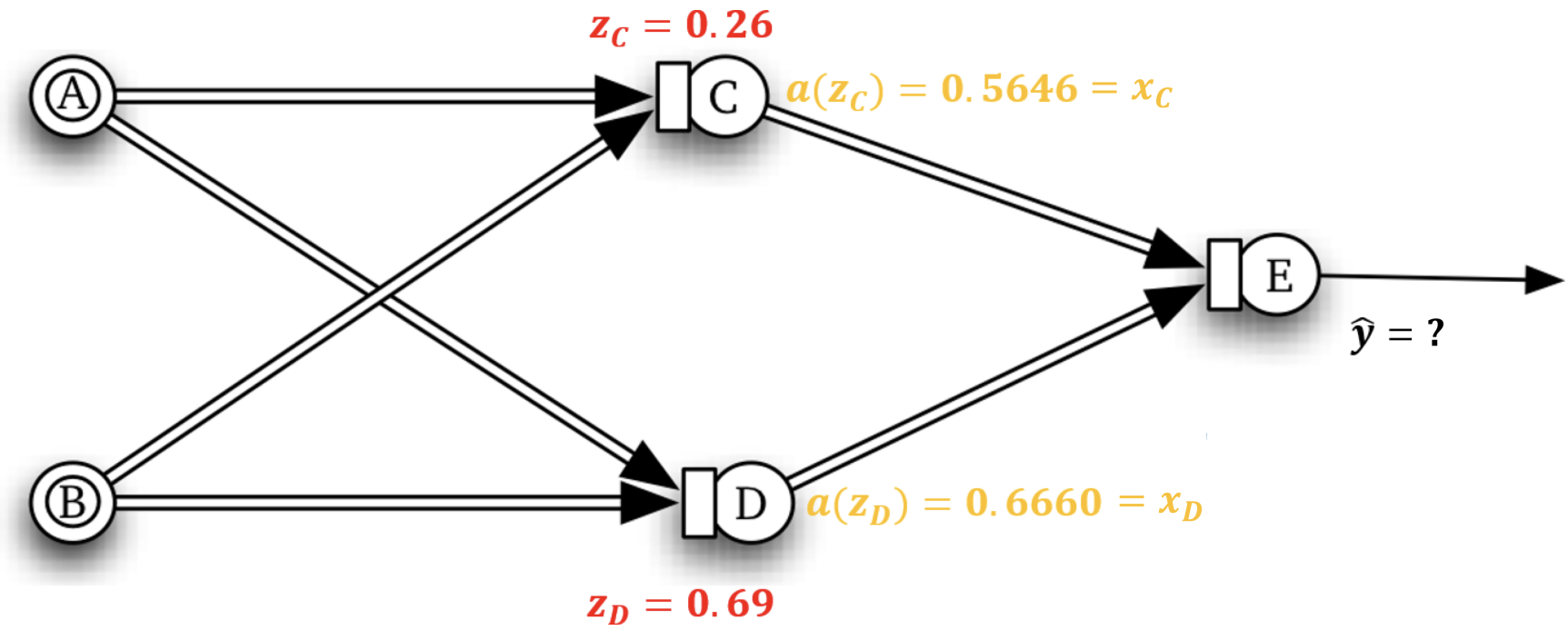
Feed-Forward Network: Function



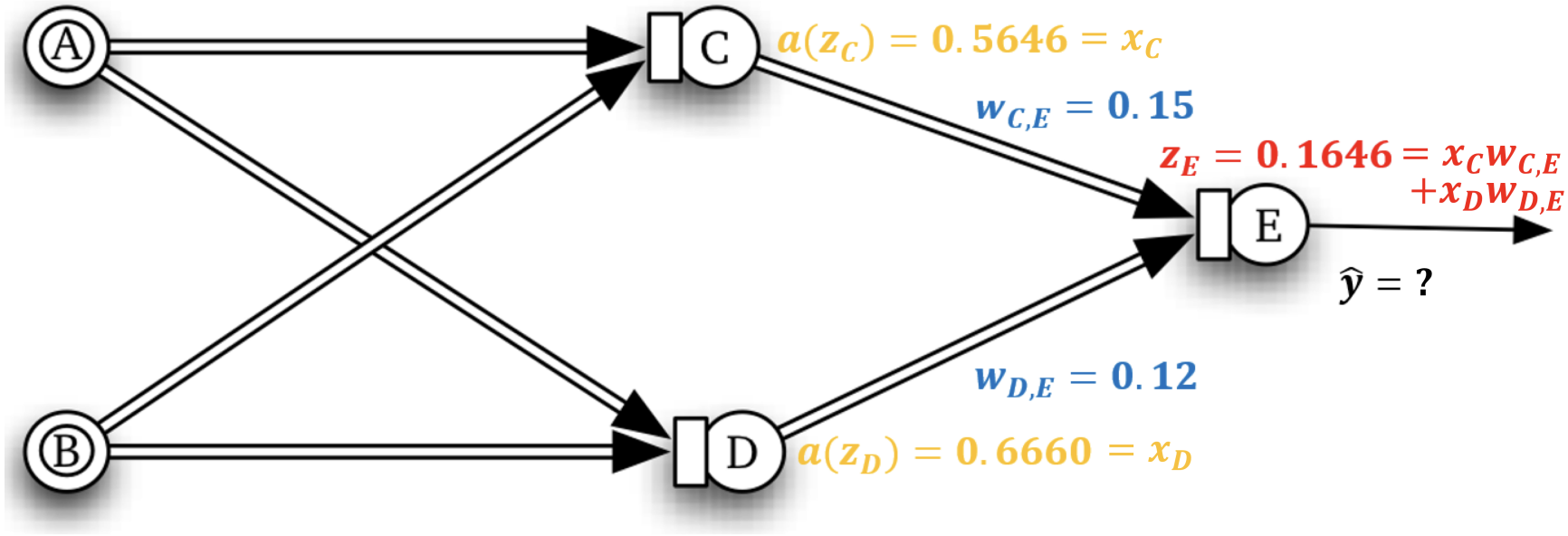
Feed-Forward Network: Signal Propagation



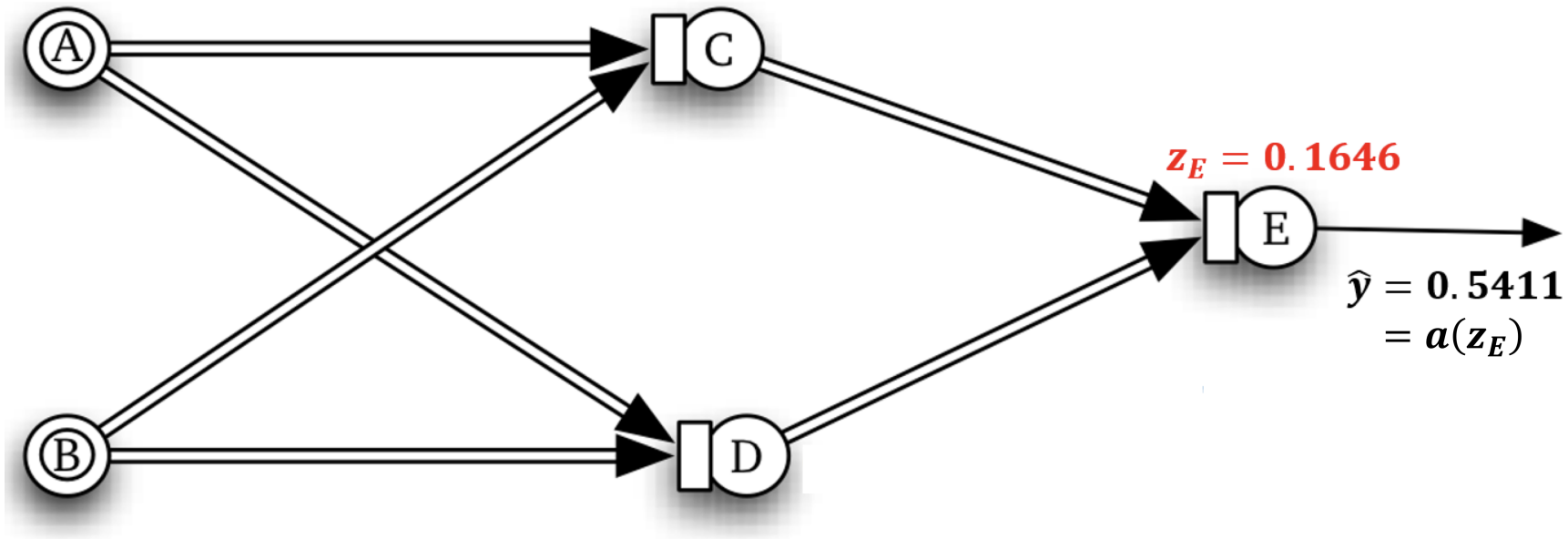
Feed-Forward Network: Signal Propagation



Feed-Forward Network: Signal Propagation



Feed-Forward Network: Signal Propagation



3.9.2 – Matrix Notation and Backpropagation

If the activation function is $a(z) = (1 + \exp(-z))^{-1}$, the network topology of the FF ANN can be re-written using **matrix notation**.

- **input layer** with p nodes:

$$\mathbf{X}_{N\times p} = \mathbf{X}_{n\times 2} = \begin{bmatrix} x_{A,1} & x_{B,1} \\ \vdots & \vdots \\ x_{A,N} & x_{B,N} \end{bmatrix};$$

- **weights** from **input layer** to **hidden layer** with M nodes:

$$\mathbf{W}^{(1)}_{p\times M} = \mathbf{W}^{(1)}_{2\times 2} = \begin{bmatrix} w_{AC} & w_{AD} \\ w_{BC} & w_{BD} \end{bmatrix};$$

- **hidden layer** with M nodes:

$$\mathbf{Z}^{(2)}_{N\times M} = \mathbf{Z}^{(2)}_{N\times 2} = \begin{bmatrix} z_{C,1} & z_{D,1} \\ \vdots & \vdots \\ z_{C,N} & z_{D,N} \end{bmatrix} = \mathbf{X}\mathbf{W}^{(1)};$$

- **activation function** on **hidden layer**:

$$\mathbf{a}^{(2)} = \begin{bmatrix} (1+\exp(-z_{C,1}))^{-1} & (1+\exp(-z_{D,1}))^{-1} \\ \vdots & \vdots \\ (1+\exp(-z_{C,N}))^{-1} & (1+\exp(-z_{D,N}))^{-1} \end{bmatrix} = g(\mathbf{Z}^{(2)});$$

- **weights** from **hidden layer** with M nodes to **output layer** with K nodes:

$$\mathbf{W}^{(2)}_{M\times K} = \mathbf{W}^{(2)}_{2\times 1} = \begin{bmatrix} w_{CE} \\ w_{DE} \end{bmatrix};$$

- **output layer** with K nodes:

$$\mathbf{Z}^{(3)}_{N\times K} = \mathbf{Z}^{(3)}_{N\times 1} = \begin{bmatrix} z_{E,1} \\ \vdots \\ z_{E,N} \end{bmatrix} = \mathbf{a}^{(2)}\mathbf{W}^{(2)};$$

- **activation function** on **output layer**:

$$\hat{\boldsymbol{y}} = \mathbf{a}^{(3)} = \begin{bmatrix} (1+\exp(-z_{E,1}))^{-1} \\ \vdots \\ (1+\exp(-z_{E,N}))^{-1} \end{bmatrix} = g(\mathbf{Z}^{(3)});$$

This FF ANN can then be expressed as:

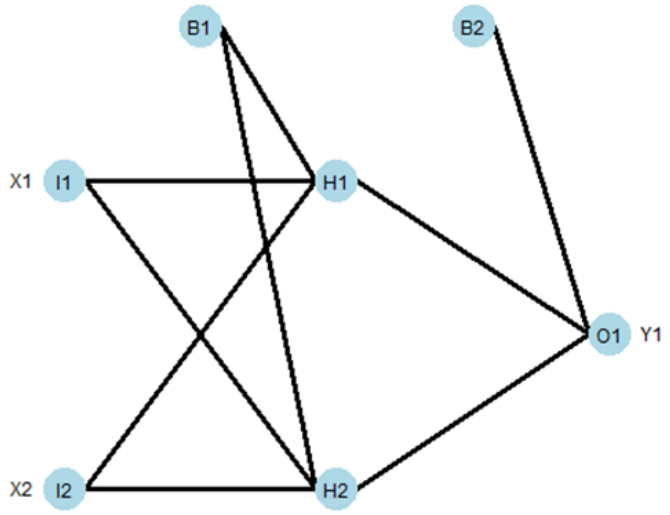
$$\hat{\boldsymbol{y}} = \mathbf{a}^{(3)} = g(\mathbf{Z}^{(3)}) = g\left[\mathbf{a}^{(2)}\mathbf{W}^{(2)}\right] = g\left[g\left(\mathbf{X}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}\right].$$

In a nutshell, at each node, the neural net computes a **weighted sum of inputs**, applies the **activation function**, and sends a **signal**, until it eventually reaches the **final output** node.

Bias Nodes

A **bias** term $\mathbf{b}$ can be introduced to all units in hidden and output layers, which act as intercepts for $\mathbf{Z}$:

$$\mathbf{Z}^{(2)} = \begin{bmatrix} b_C + z_{C,1} & b_D + z_{D,1} \\ \vdots & \vdots \\ b_C + z_{C,N} & b_D + z_{D,N} \end{bmatrix}.$$



Training ANN

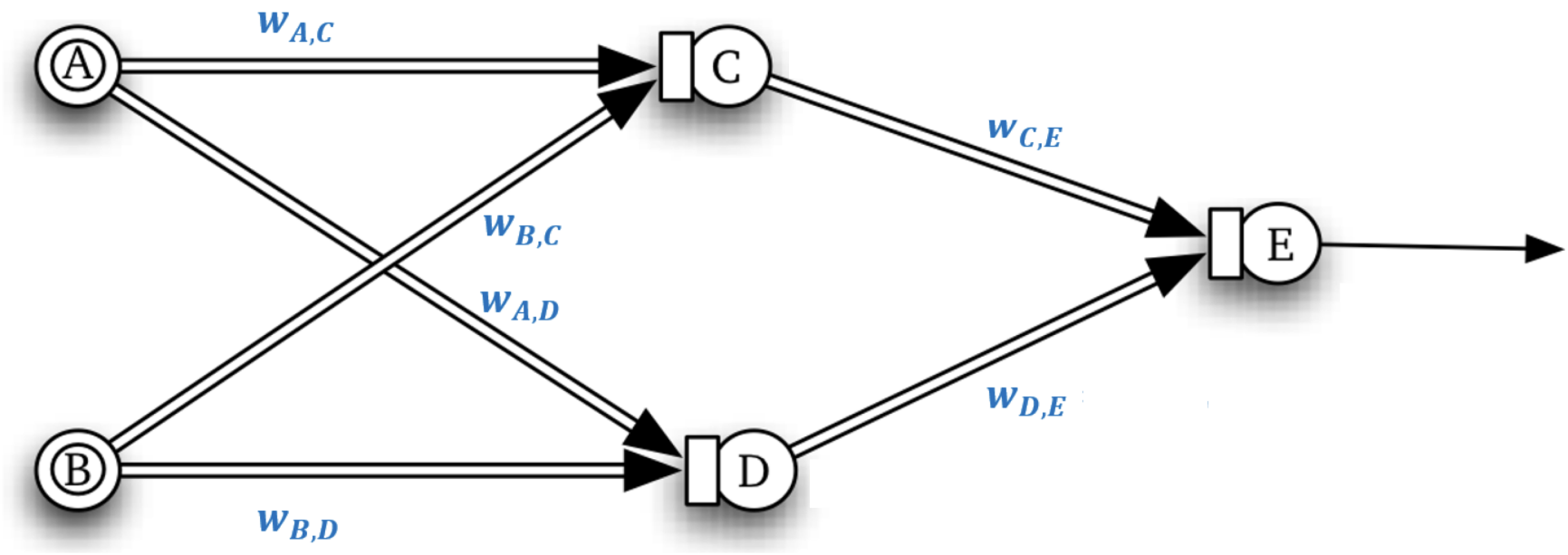
Given a signal, an ANN can produce an output, as long as the **weights** are specified.

For SL tasks, simply picking **weights at random** is a failing proposition.

Backpropagation is a method to optimize the choice of the weights against an **error function** $R(\mathbf{W})$.

It sounds mysterious, but it's just an application of the **chain rule** of calculus: we are looking for the weights that will **minimize** $R(\mathbf{W})$, i.e., find the **weights** $\mathbf{W}$ s.t. $\nabla_{\mathbf{W}} R(\mathbf{W}) = \mathbf{0}$.

This is usually done via **numerical methods**: **(stochastic) gradient descent**, (and variants), etc.



Let $\hat{y}_{i,\ell}(\mathbf{W})$ and $y_{i,\ell}$ be the **ANN** value/class label and the **observed** value/class label, respectively, for the ℓ^{th} output of the i^{th} case in Tr.

Let N be the number of observations in Tr and K be the number of output nodes in the ANN.

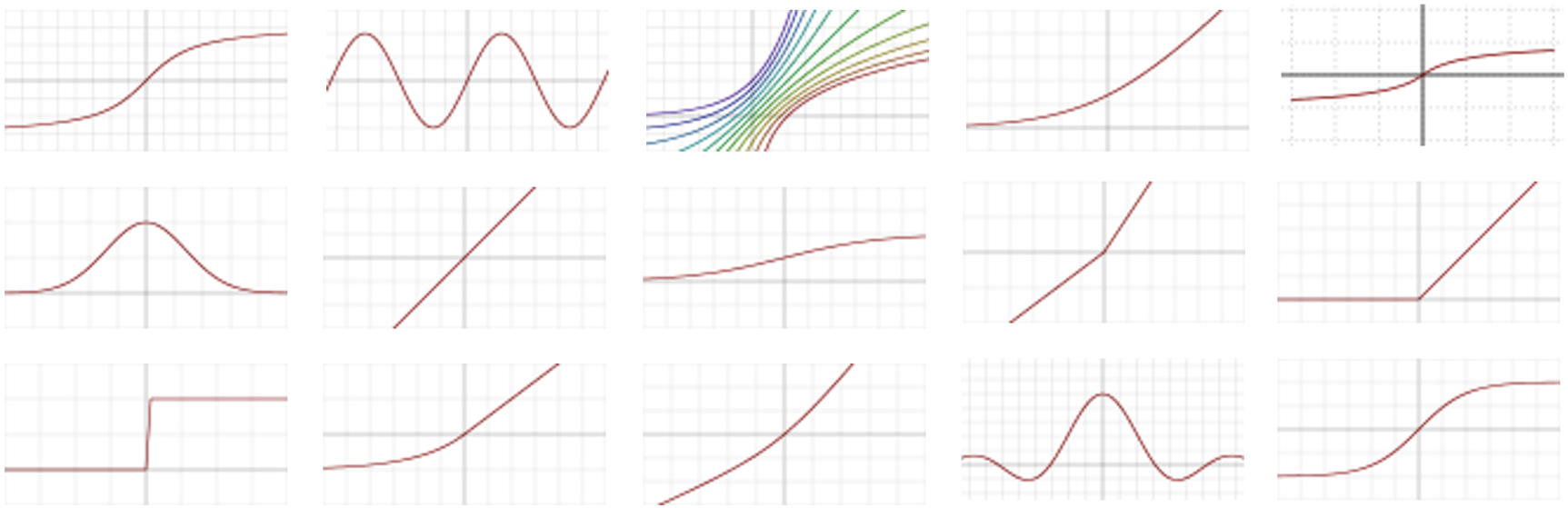
For regression problems, the **sum of squared errors (SSE)** is often used as the error function:

$$R(\mathbf{W}) = \sum_{i=1}^{N} \sum_{\ell=1}^{k} \left(\hat{y}_{i,\ell}(\mathbf{W}) - y_{i,\ell}\right)^2.$$

For classification problems, **cross-entropy** can be used:

$$R(\mathbf{W}) = -\sum_{i=1}^{N} \sum_{\ell=1}^{k} y_{i,\ell} \ln \hat{y}_{i,\ell}(\mathbf{W}).$$

Activation Functions



There are many **options**: what effect(s) does the choice have, if any? That question is difficult to answer with any degree of certainty.

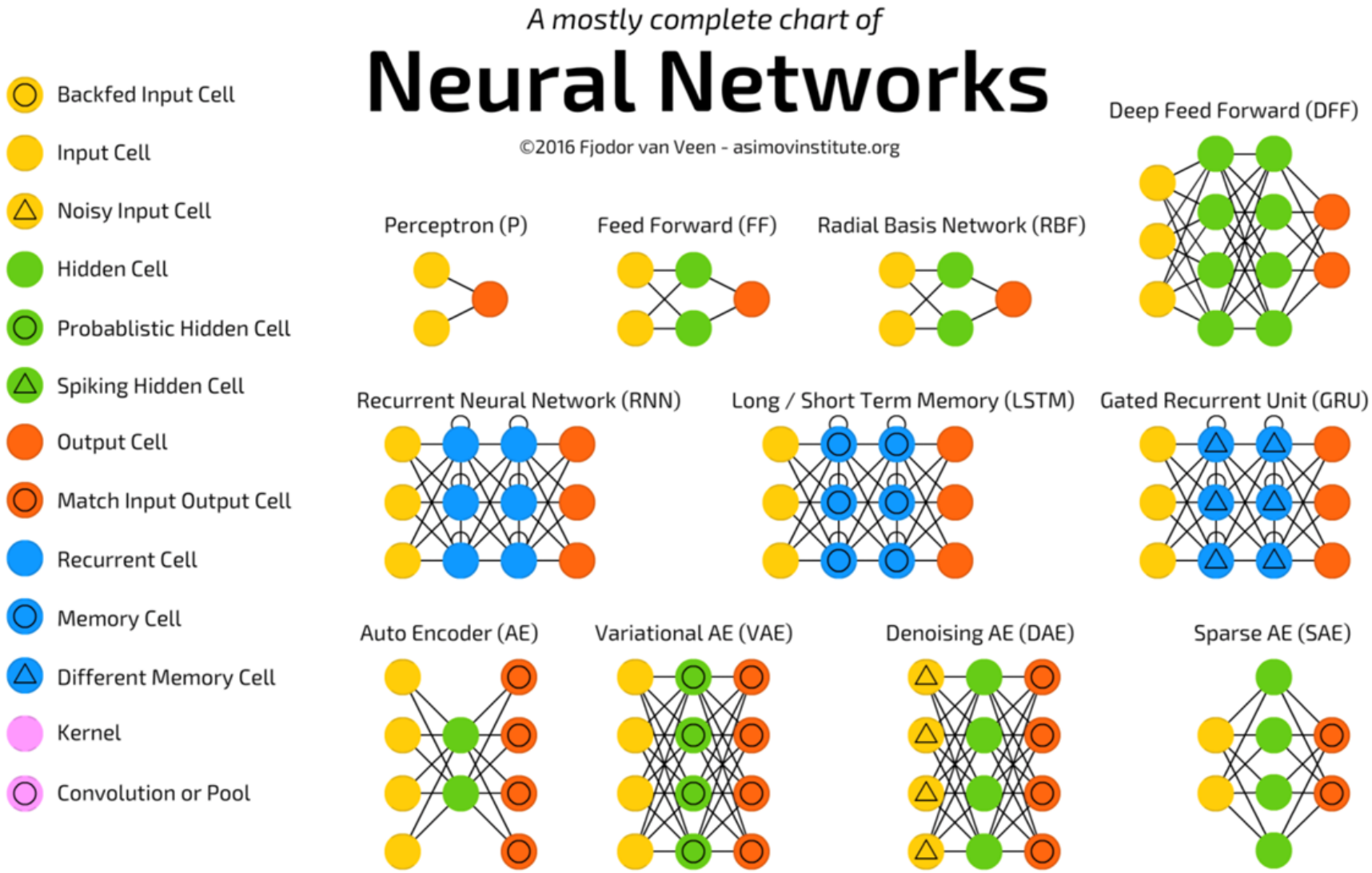
Deep Learning Networks

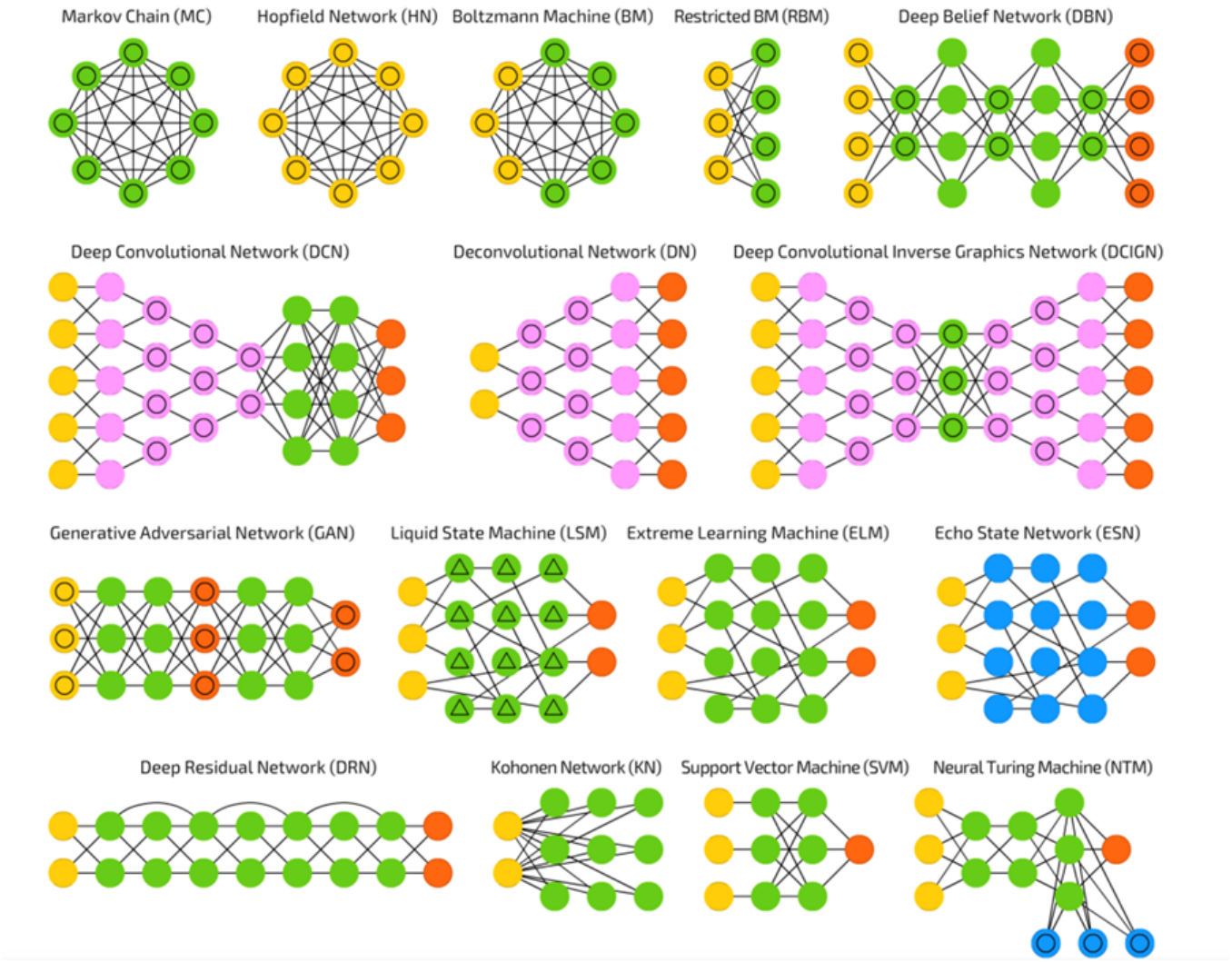
Deep Learning (DL) networks are simply ANN with a large number of **hidden layers** (and various types of **nodes**).

DL Types:

- **Convolution Neural Networks** (handwritten digit recognition, self-driving vehicles)
- **Recurrent Neural Networks** (NLP – speech recognition, machine translation, etc.)
- **Autoencoders**
- **Restricted Boltzmann Machines** (BellKor's Pragmatic Chaos, Netflix Prize, 2009)

A Mostly Complete Chart of Neural Networks, Fjodor van Veen, Asimov Institute, 2016.


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)


Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

3.9.3 – Notes and Comments

Pros: ANN can be quite **accurate** when making predictions ... with a **proper** set-up.

ANN often work when other things fail:

- when the relationship between attributes is **complex**
- when there are a lot of **dependencies**/**nonlinear relationships**
- when inputs are **messy** and/or **highly connected** (images, text and speech)
- **nonlinear** classification

ANN are relatively easy to **implement** (with available packages – e.g., keras, TensorFlow).

ANNs **degrade** gracefully (important in robotics).

Cons: ANN are relatively **slow** (to set-up and to run) and prone to **overfitting**.

Humans do not need to see more than a handful of cats and dogs early on to be able to correctly identify cats and dogs for the rest of their lives; ANN require **large/diverse** Tr to accomplish such tasks.

ANN are **black boxes** (unlike decision trees or logistic regression, say). Is **accuracy** more important than **interpretability**? That depends.

There is no process for selecting the **optimal network topology**.

Even when ANN perform better than other options, they may not perform **that much better** due to **No Free-Lunch Theorems**; and they're susceptible to various forms of **adversarial attacks**.

Accurate on average, but they can still be **spectacularly** wrong.


a man is riding a skateboard on a ramp

Weights and Activation Functions: weights that are near 0 make the activation function act like a **linear model**; it becomes **nonlinear** as the magnitude of weights **increase**.

Overfitting: with too many weights (**complex network topology**), ANN often **overfits** the data; **weight decay** (which adds penalty to error function, see **regularization**) may be used to mitigate this problem.

Scaling: signals propagate as **linear combinations** of inputs; scaling the data allows ANN to treat each variable **equally**.

Non-Convex Error Functions: ANN may have **multiple** minima, so ANN must be run **multiple times**.

Averaging: the **activation functions** are nonlinear, so multiple ANN runs $\Longrightarrow$ **average predictions** rather than **average weights**.

3.9.4 – Example: Wine Dataset

We consider the Wine.csv dataset, containing $n = 178$ sample of wines grown in the same region of Italy, which come from three different cultivars (**response**).

$p = 13$ chemical and non-chemical properties (**predictors**) were collected.

Variables:

- alcohol
- malic acid
- ash
- alkalinity of ash
- magnesium
- total phenols
- flavonoids
- non-flavonoid phenols
- proanthocyanins
- colour intensity
- hue
- OD280/OD315
- proline

Pre-Processing

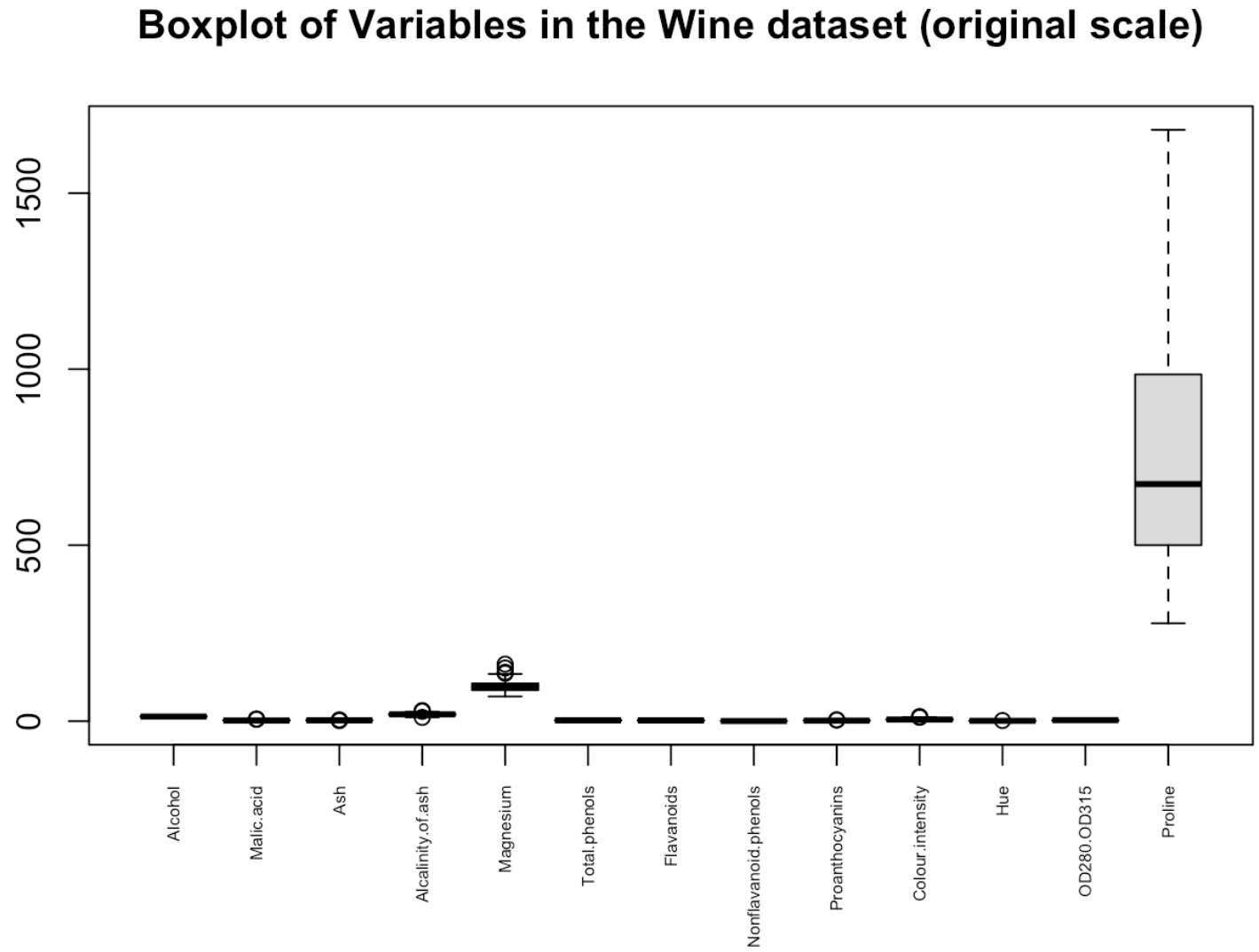
Flavonoid is removed due to **high correlation** with other variables.

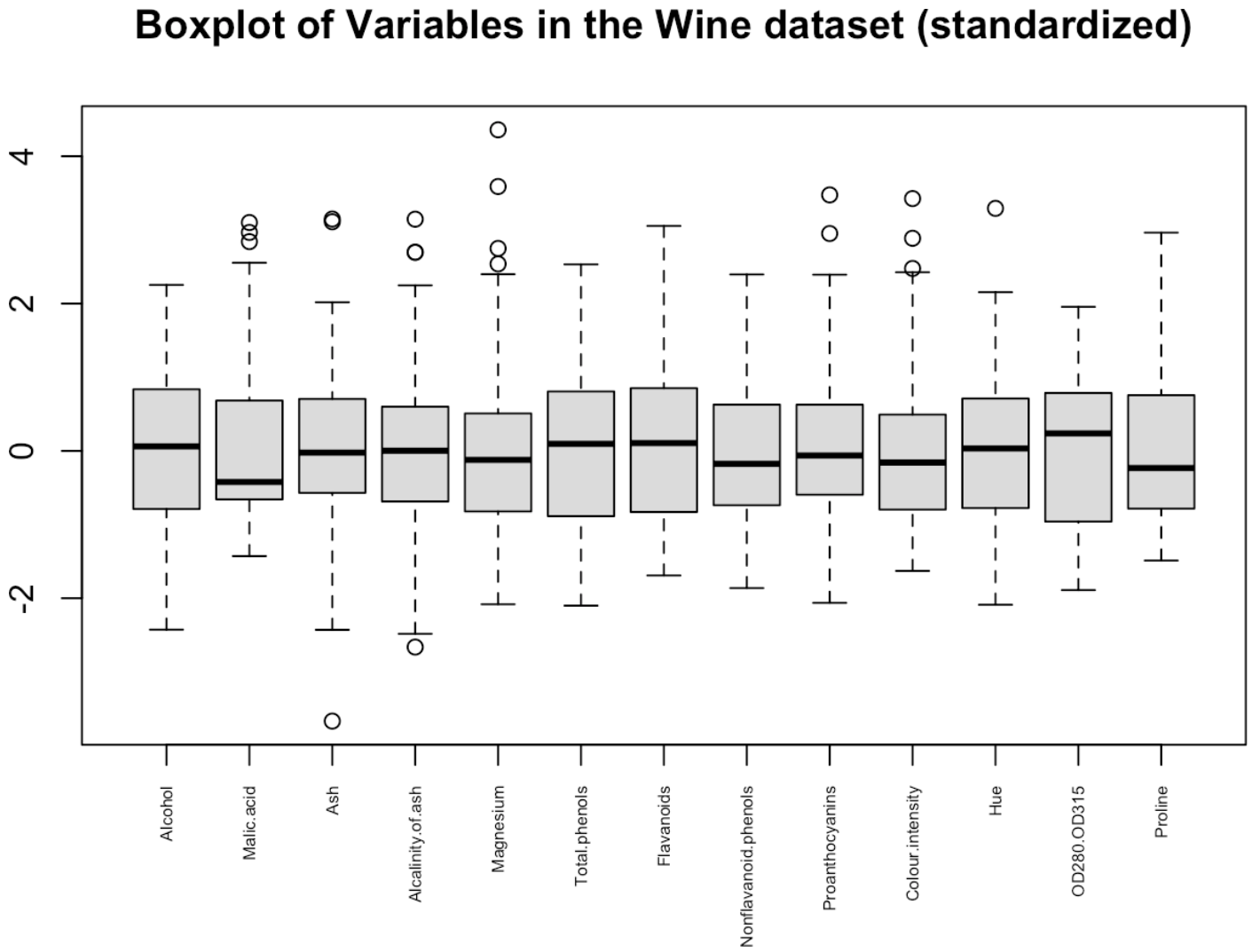
The data is **standardized** as the units of measurement are not provided in the data (and also because proline's **range**/**variability** masks everything else).

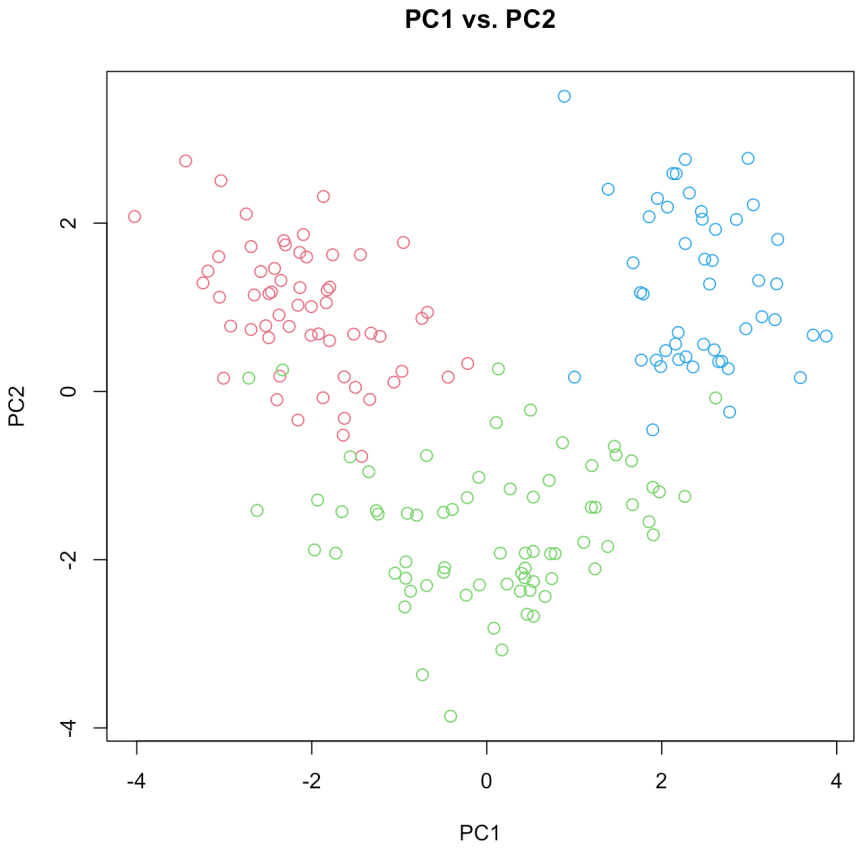
We perform PCA (see chapter 4) prior to running ANN on the dataset.

The data is separated into Tr (with $N = 140$ observations) and Te (with remaining $M = 38$ observations).

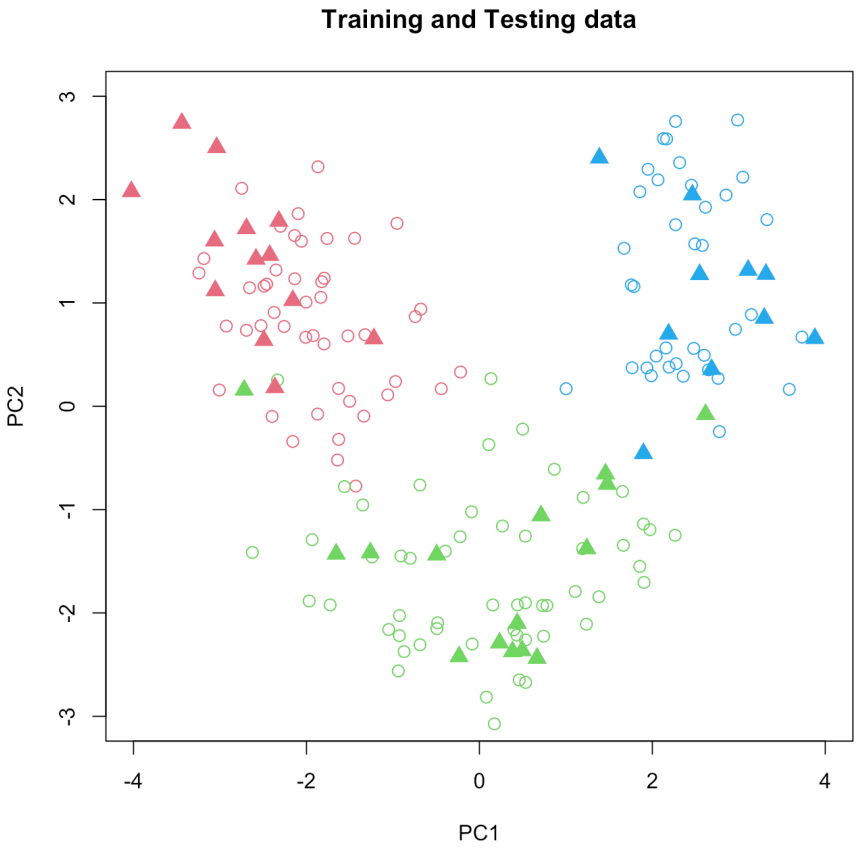
(DUDADS, 21.4.3, *Artificial Neural Networks*, for code).


Boxplot of Variables in the Wine dataset (original scale)
0
500
1000
1500
Alcohol
Malic.acid
Ash
Alcalinity.of.ash
Magnesium
Total.phenols
Flavanoids
Nonflavanoid.phenols
Proanthocyanins
Colour.intensity
Hue
OD280.OD315
Proline


Boxplot of Variables in the Wine dataset (standardized)
4
2
0
-2
Alcohol
Malic.acid
Ash
Alcalinity.of.ash
Magnesium
Total.phenols
Flavanoids
Nonflavanoid.phenols
Proanthocyanins
Colour.intensity
Hue
OD280.OD315
Proline


PC1 vs. PC2
PC2
PC1
-4
-2
0
2
4

Tr


Training and Testing data
PC2
PC1
-3
-2
-1
0
1
2
3
-4
-2
0
2
4

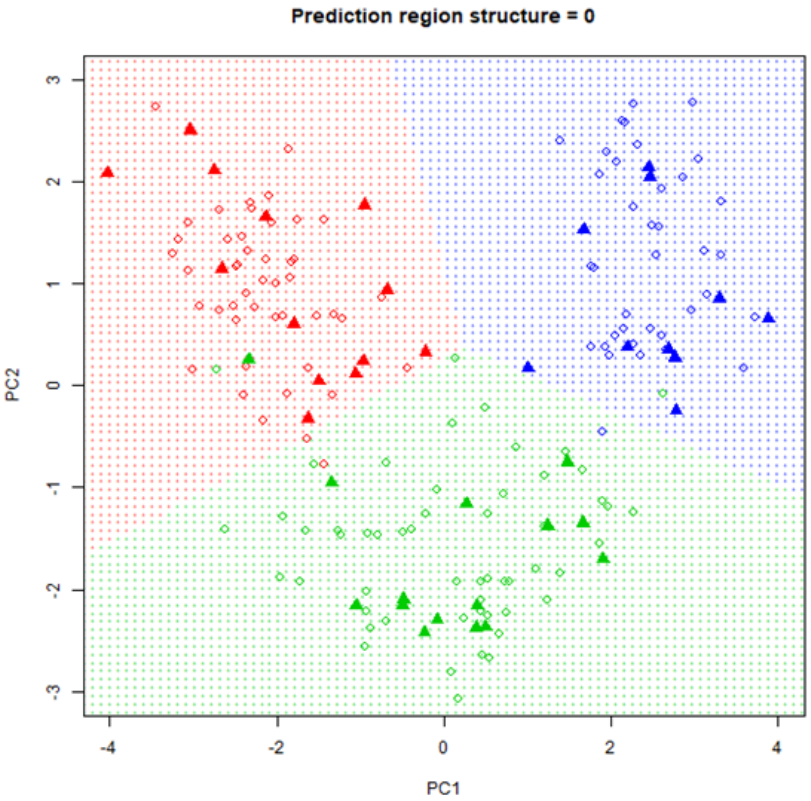
Tr + Te

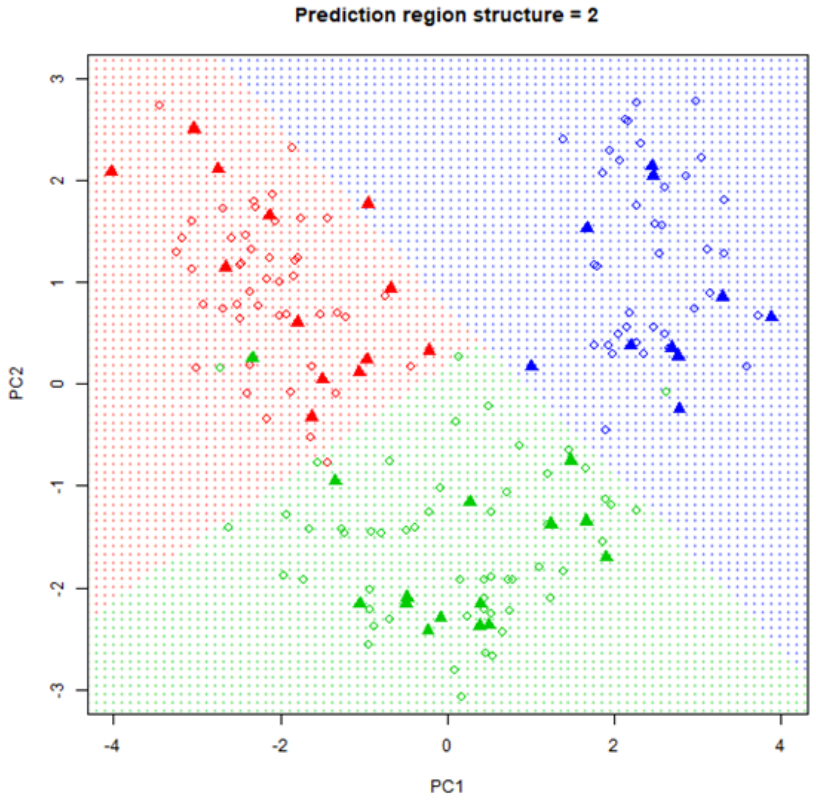
ANN Models

We fit 11 models in total, using only the first 2 **principal components**; 50 iterations each. We report on median (max) number of errors.

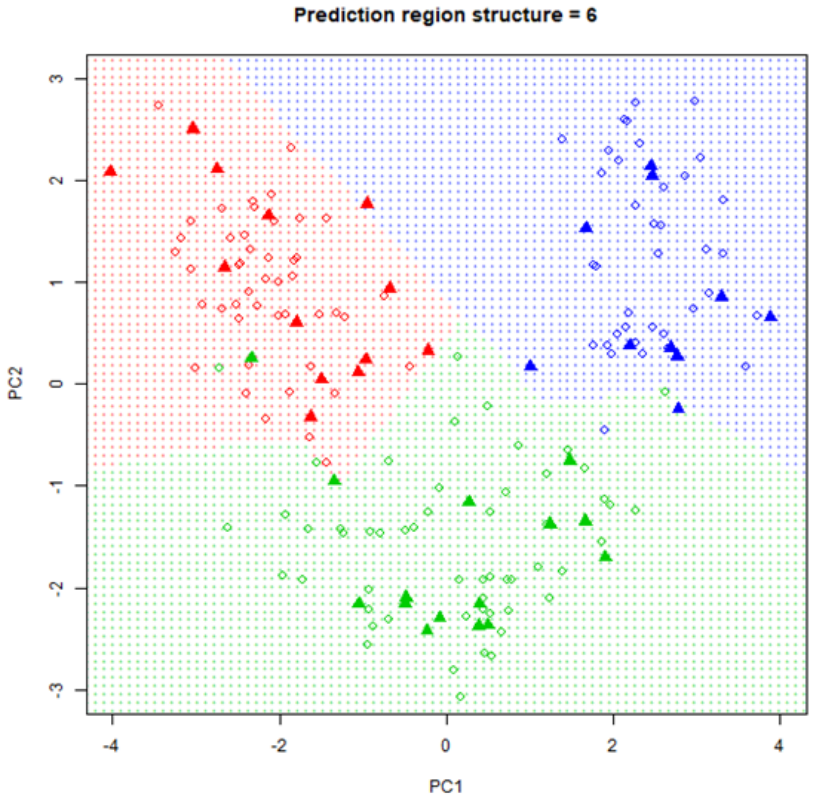
The simplest ANN has no hidden layers, while the most complicated network contains three hidden layers, each with 30 nodes.

0 hidden layers; 0 nodes $\Longrightarrow$ **linear**

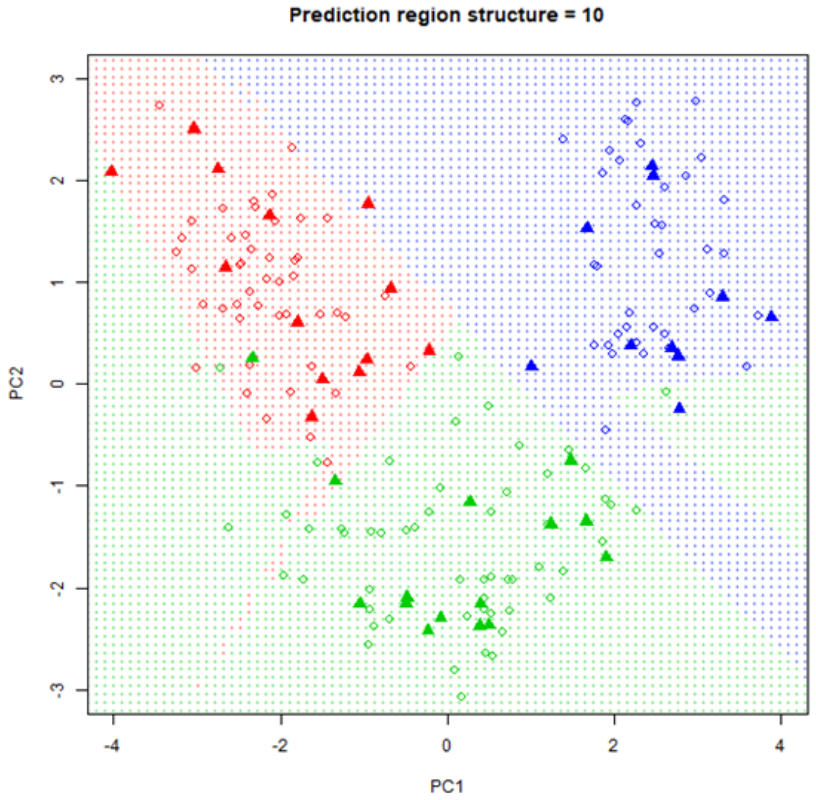


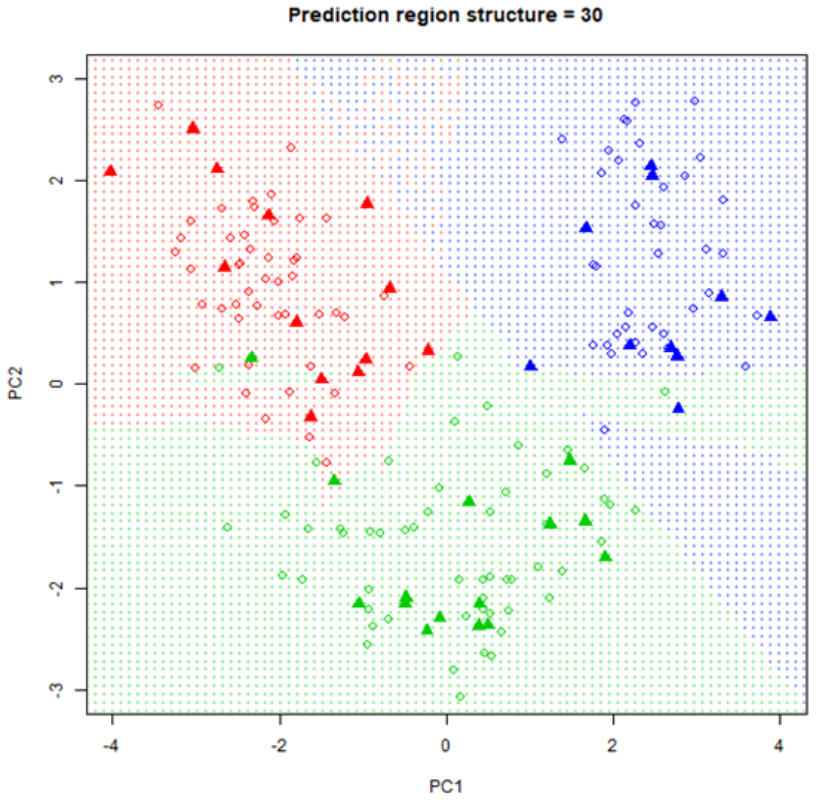


1 hidden layer; 2 nodes



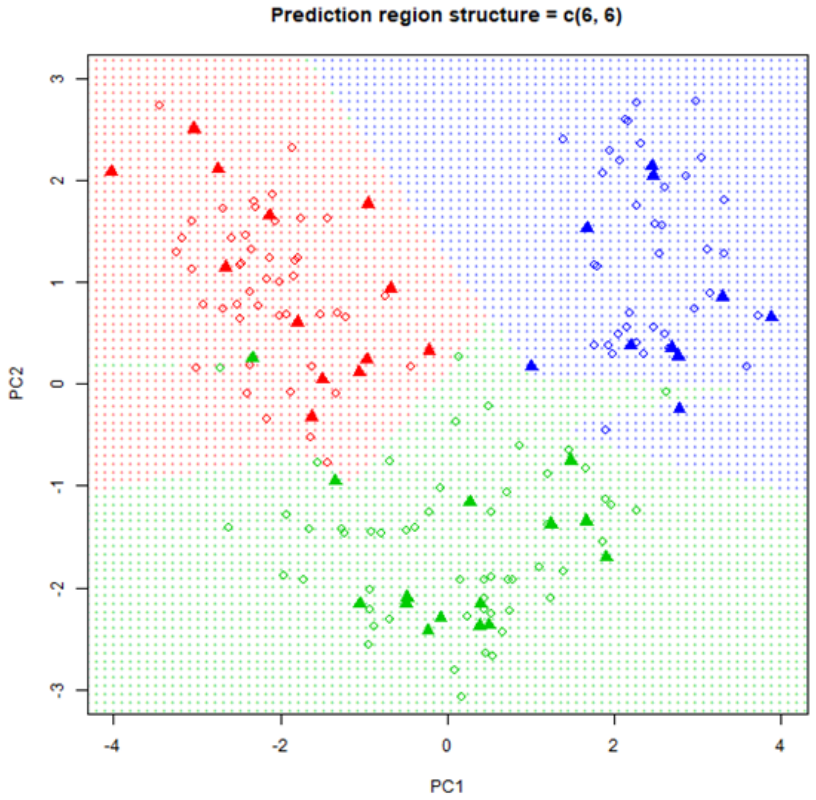
1 hidden layer; 6 nodes

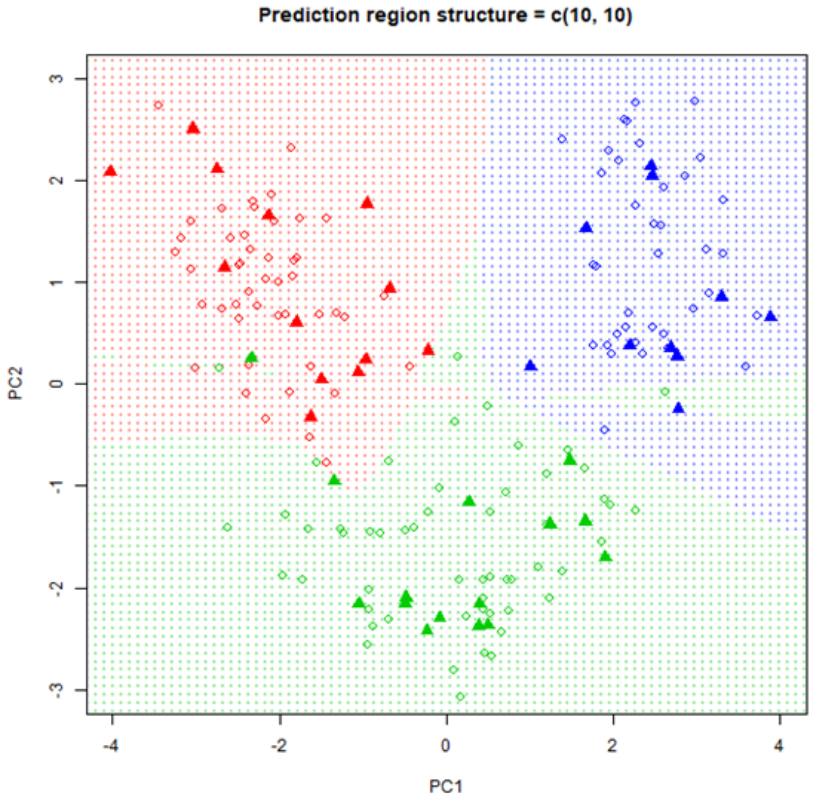




1 hidden layer; 10 nodes

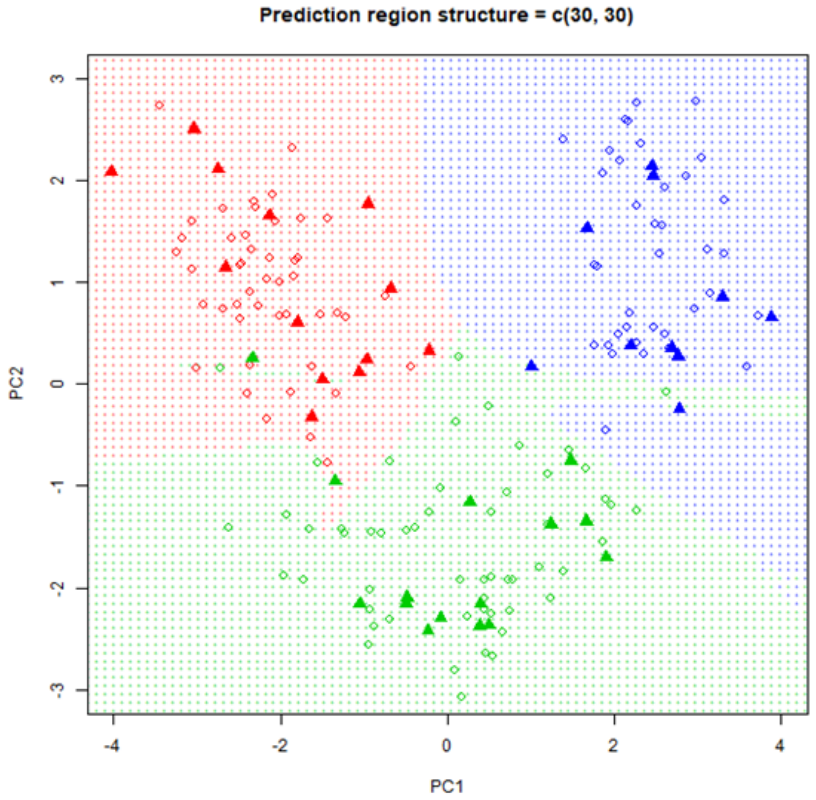
1 hidden layer; 30 nodes

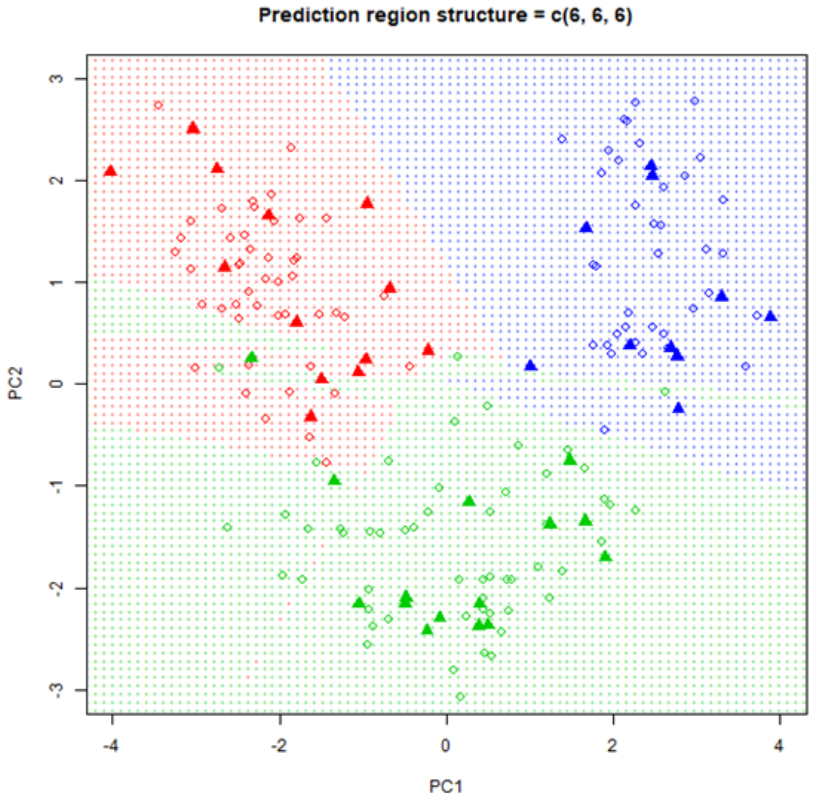




2 hidden layers; $6, 6$ nodes

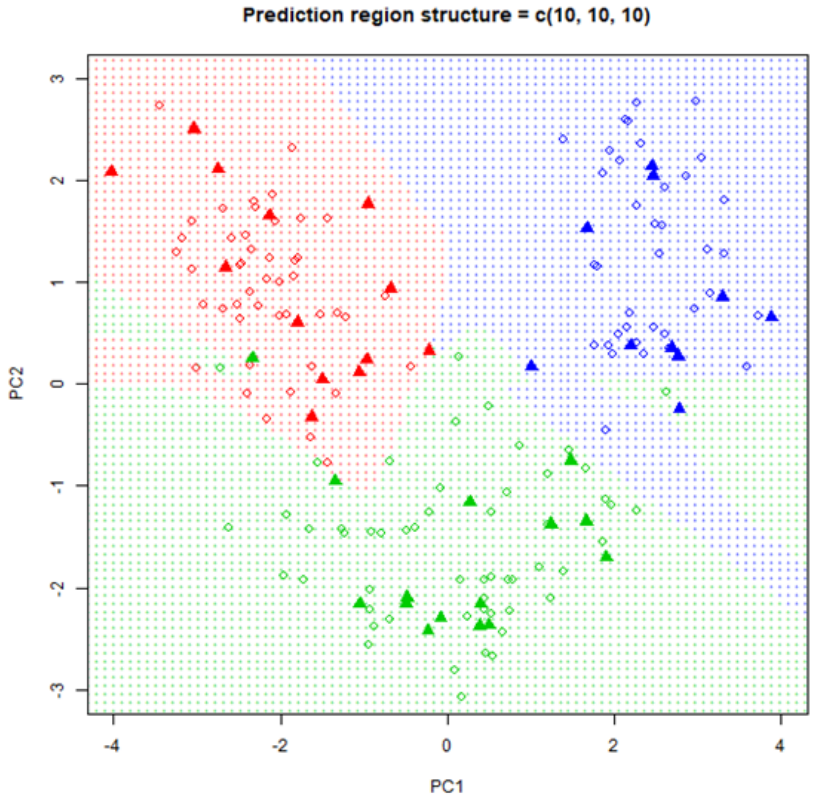
2 hidden layers; $10, 10$ nodes

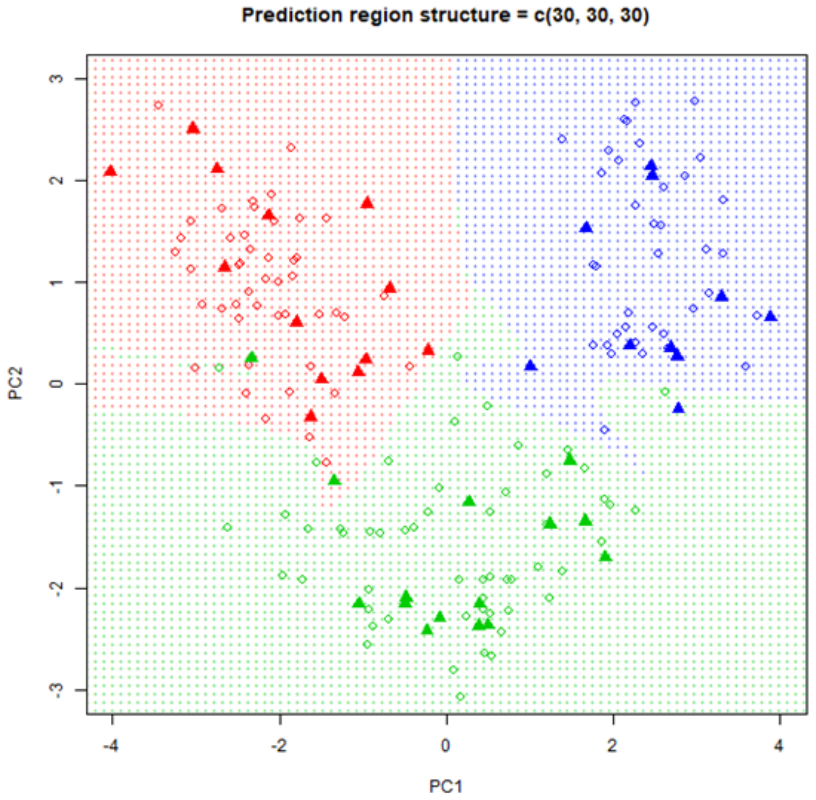




2 hidden layers; $30, 30$ nodes

3 hidden layers; $6, 6, 6$ nodes





3 hidden layers; $10, 10, 10$ nodes

3 hidden layers; $30, 30, 30$ nodes

Structure	Mis. (Tr)	Mis. (Te)
0	5 (5)	1 (1)
2	3 (5)	2 (3)
6	1 (3)	3 (6)
10	1 (3)	3 (6)
30	0 (0)	4 (8)
(6,6)	0 (2)	3.5 (7)
(10,10)	0 (2)	3 (6)
(30,30)	0 (0)	4 (6)
(6,6,6)	0 (2)	3 (6)
(10,10,10)	0 (1)	3 (5)
(30,30,30)	0 (0)	3.5 (5)

PC1 – PC2

Structure	Mis. (Tr)	Mis. (Te)
0	0 (0)	1 (1)
2	1 (2)	2 (3)
6	0 (0)	1 (2)
10	0 (0)	1 (2)
30	0 (0)	1 (2)
(6,6)	0 (0)	2 (3)
(10,10)	0 (0)	2 (3)
(30,30)	0 (0)	2 (3)
(6,6,6)	0 (0)	2 (4)
(10,10,10)	0 (0)	2 (4)
(30,30,30)	0 (0)	1 (2)

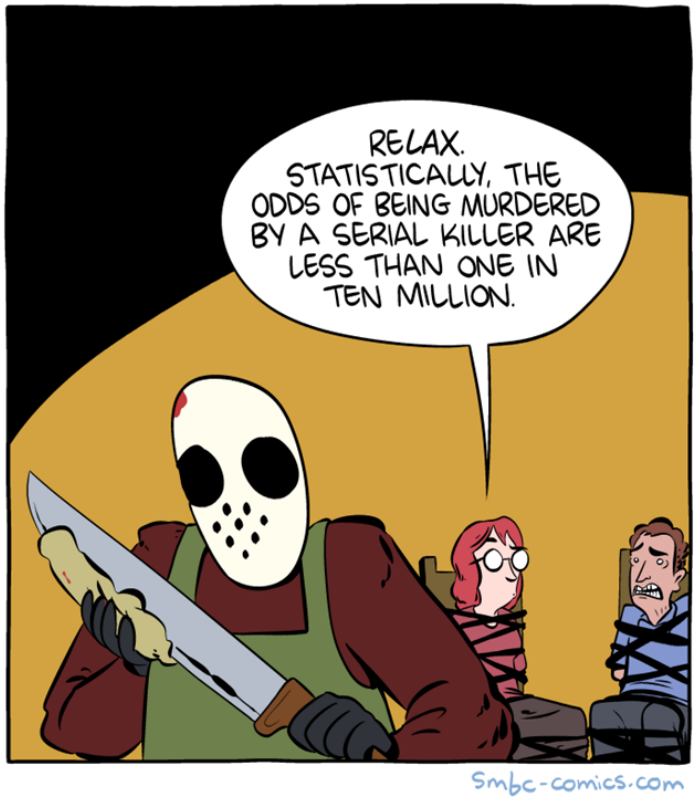
PC1 – PC6

Structure	Mis. (Tr)	Mis. (Te)
0	0 (0)	0 (0)
2	0 (0)	1 (3)
6	0 (0)	0 (1)
10	0 (0)	0 (1)
30	0 (0)	0 (1)
(6,6)	0 (0)	0 (1)
(10,10)	0 (0)	0 (1)
(30,30)	0 (0)	0 (2)
(6,6,6)	0 (0)	0 (1)
(10,10,10)	0 (0)	0 (2)
(30,30,30)	0 (0)	0 (1)

PC1 – PC 12

Note the bias-variance trade-off/overfitting as complexity **increases**.

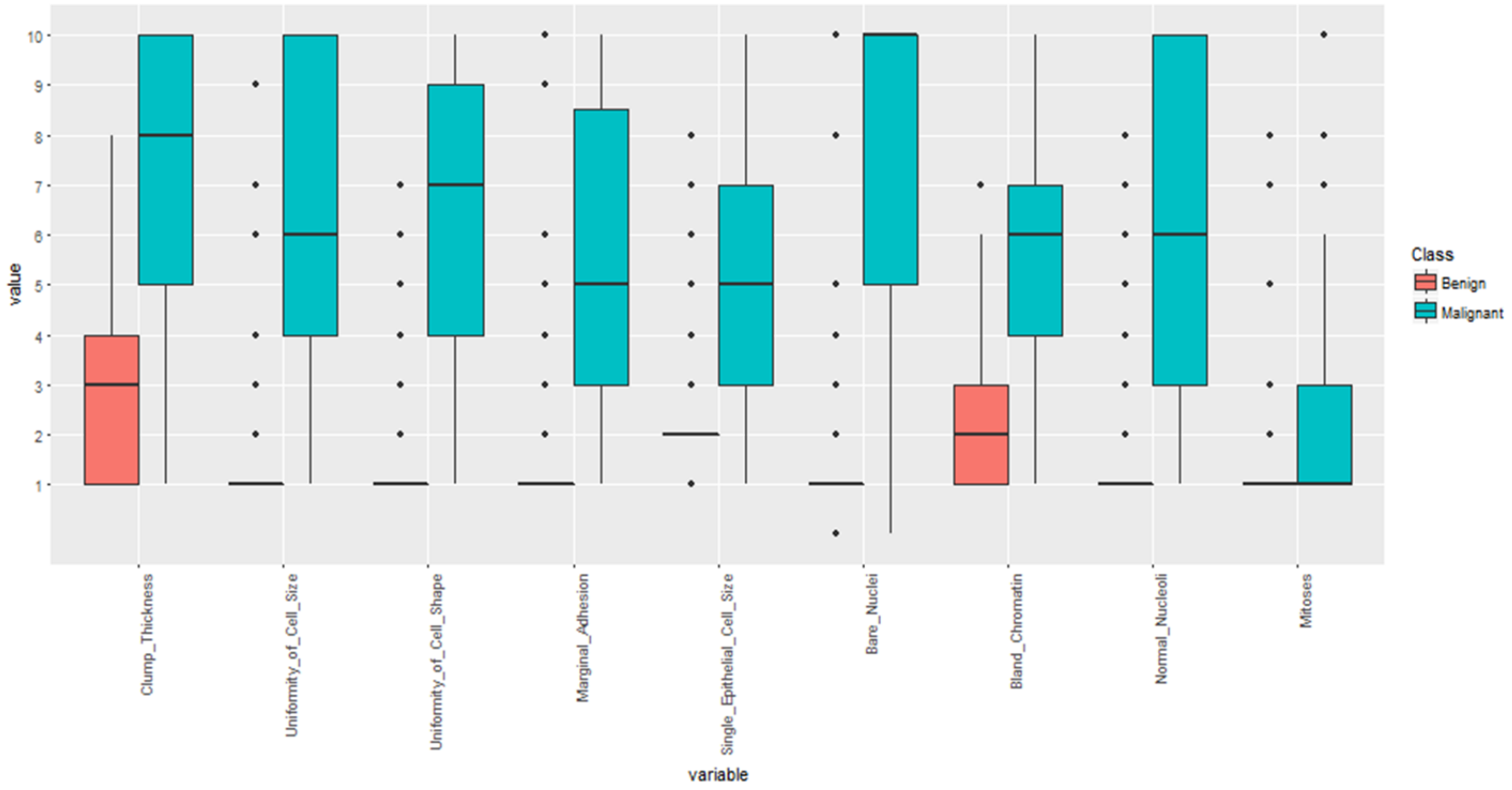
3.10 – Naïve Bayes Classification



Suppose we are interested in diagnosing whether a tumour is **benign** or **malignant**, based on $p = 9$ video imaging measurements.

All measurements are given on a scale of 1 to 10 (with missing values coded as 0); $n = 699$ observations have been collected, of which 458 are **benign**, and 241 are **malignant**.

$N = 559$ cases are used to **train** the classifier, $M = 140$ to **test** it.


value
10
9
8
7
6
5
4
3
2
1
Clump_Thickness
Uniformity_of_Cell_Size
Uniformity_of_Cell_Shape
Marginal_Adhesion
Single_Epithelial_Cell_Size
Bare_Nuclei
Bland_Chromatin
Normal_Nucleoli
Mitoses
variable
Class
Benign
Malignant

There are $M = 140$ undiagnosed subjects with measurements.

Can we determine whether their tumour is benign or malignant based on the score?

Naïve Bayes (NBC) is:

- easy to implement;
- naïve (simple assumptions);
- robust ,
- ... and in some sense, **optimal**

Observation	Clump Thickness	Uniformity of Cell Size	Bland Chromatin	Normal Nucleoli	Mitoses
1	3	1	3	1	1
2	6	8	3	7	1
3	4	1	3	1	1
4	2	1	3	1	1
5	1	1	3	1	1
86	1	2	1	1	1
87	3	1	1	1	1
88	4	2	2	1	1
89	1	1	2	1	1
90	4	3	3	3	1
136	1	1	2	1	1
137	1	1	1	1	1
138	3	1	2	1	2
139	2	1	1	1	1
140	4	8	10	6	1

In Tr, there are 376 benign and 183 malignant tumours.

(DUDADS, 21.4.4, *Naïve Bayes Classifiers*, for code).

3.10.1 – Theory

Let C_1 and C_2 be the two classification labels, and denote $\mathbf{x}^*, \mathbf{x}_i$ the predictor vectors of an observation in Te and of the ith observation in Tr.

Probability-based classifiers are built on Tr and compare

$$\pi_1^* = P(y^* \in C_1 \mid \vec{X} = \mathbf{x}^*) \quad \text{vs.} \quad \pi_2^* = P(y^* \in C_2 \mid \vec{X} = \mathbf{x}^*).$$

In the *Wisconsin Breast Cancer* dataset:

- $C_1 = C_B$: benign tumour; $C_2 = C_M$: malignant tumour
- $\mathbf{x} = (x_1, \ldots, x_9)$: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses
- $\pi_1^* \geq \pi_2^* \implies y^* \in C_B \implies$ tumour benign

The foundation of NBC rest on **Bayes' Theorem**, expressed as:

$$P(\text{hypothesis} \mid \text{data}) = \frac{P(\text{data} \mid \text{hypothesis}) \times P(\text{hypothesis})}{P(\text{data})}, \quad \text{or}$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} \propto \text{likelihood} \times \text{prior},$$

where

- **posterior:** based on the collected data, how likely is a given tumour to be benign (or malignant)?
- **prior:** in what proportion are tumours benign (or malignant), in general?
- **likelihood:** knowing a tumour is benign (or malignant), how likely is it that these particular measurements would have been observed?
- **evidence:** regardless of a tumour being benign or malignant, what is the chance that a tumour has the observed characteristics?

Using conditional probabilities, the classifier can be re-cast as comparing

$$\tilde{\pi}_1^* = P(\vec{X} = \mathbf{x}^* \mid y^* \in C_1) \times P(y^* \in C_1) \quad \text{vs.}$$
$$\tilde{\pi}_2^* = P(\vec{X} = \mathbf{x}^* \mid y^* \in C_2) \times P(y^* \in C_1);$$

the Breast Cancer classifier would then compare

$$\tilde{\pi}_1^* = P(x_1^*, \ldots, x_9^* \mid C_1) \times P(C_1) \quad \text{vs.}$$
$$\tilde{\pi}_2^* = P(x_1^*, \ldots, x_9^* \mid C_2) \times P(C_2).$$

NBC requires only one, **naïve** assumption: in a given class, the various measurements are independent.

In our case, this means that for benign (or for malignant) tumours, clump thickness is **independent** of uniformity of cell size, mitoses, and so on. **(?!)**

This assumption simplifies the computation of the **likelihood** significantly:

$$P(\vec{X} = \mathbf{x}^* \mid y^* \in C_k) \approx \prod_{j=1}^{p} P(X_j = x_j^* \mid y^* \in C_k) = \tilde{\mathcal{L}}_k^*.$$

If p is small enough and Tr contains a sufficient number of observations, the **(joint)** probability $\tilde{\pi}_k^*$ can be computed directly, but the ability to do so decreases very quickly as p increases.

The individual **conditional probabilities** $P(X_j = x_j^* \mid y^* \in C_k)$ are nearly always calculable from the data.

Since the assumption of independence is not usually met in practice, however, $\tilde{\mathcal{L}}_k^*$ does not typically represent the **true likelihood**; in general,

$$\tilde{\tau}_k^* = \tilde{\mathcal{L}}_k^* \times P(C_k) \neq \tilde{\pi}_k^*.$$

3.10.2 – NBC Steps

Step 0: preliminary analyses (**visualization**, etc.)

Step 1: compute the **prior probabilities** $P(C_k)$

Step 2: choose conditional distributions for the **likelihoods**

Step 3: estimate **likelihood parameters**

Step 4: compute **likelihoods** $\tilde{L}_k^*$

Step 5: compute **posterior probabilities** $\tilde{\tau}_k^*$

Step 6: make **classification decisions**

Step 1 – Prior Probabilities

What is the proportion of benign/malignant tumours?

Option 1: we may not know much about these proportions, so we blindly assume that their proportions are equal:

$$P(C_B) = P(C_M) = 50\%;$$

***Option 2:** we can ask subject matter experts (SMEs), or base it on Tr:

$$P(C_B) = 376/559 = 67\%, \quad P(C_M) = 183/559 = 33\%;$$

Option 3: any other **reasonable** distribution.

Step 2 – Conditional Distributions

The choice of **conditional distributions** for the **likelihoods** depends on the nature of the variables and of the data.

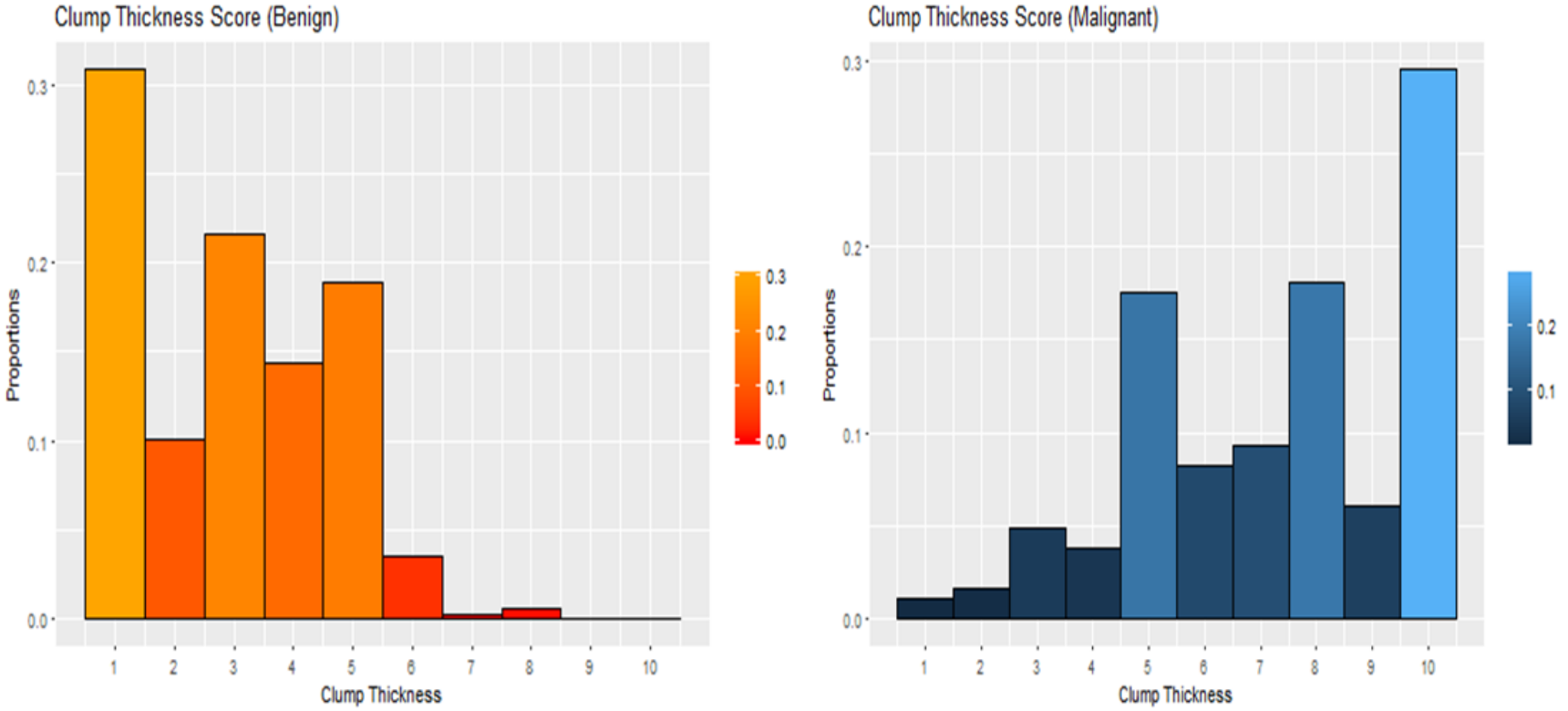
Option 1: perhaps the measurements follow normal distributions

$$X_{i,j} \mid C_k \sim \mathcal{N}(\mu_{j,k}, \sigma^2_{j,k});$$

***Option 2:** multinomial distributions

$$X_{i,j} \mid C_k \sim \text{Multinomial}(p_{1,j,k}, \ldots, p_{d,j,k});$$

Option 3: any other **reasonable** distribution.



$$p_{\ell,j,k} = (\#\text{ of observations for which } x_j = \ell \text{ in } C_k)/|C_k^{(j)}|$$

Step 3 – Parameter Estimation

In order to compute the **likelihood**, we need to estimate the parameters for each **conditional distribution**.

For each variable and tumour type, we need 9 parameters: $p_1, \ldots, p_9$ (and $p_{10} = 1 - p_1 - \cdots - p_9$).

We estimate them *via* **observed proportions** in Tr.

	Benign									Malignant								
Score	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
1	30.9%	83.2%	78.5%	81.9%	10.9%	85.4%	33.2%	88.0%	97.6%	1.1%	1.6%	1.1%	12.6%	0.5%	6.6%	1.1%	14.8%	50.8%
2	10.1%	7.7%	10.9%	8.5%	78.5%	4.3%	35.4%	6.9%	1.3%	1.6%	2.7%	3.3%	8.7%	9.8%	4.4%	3.3%	3.3%	13.1%
3	21.5%	6.6%	6.1%	6.4%	6.9%	2.9%	27.4%	2.4%	0.3%	4.9%	11.5%	8.7%	12.0%	16.4%	6.0%	16.4%	10.4%	14.8%
4	14.4%	1.6%	2.9%	1.1%	1.6%	1.3%	1.6%	0.3%	0.0%	3.8%	13.1%	13.7%	10.4%	14.2%	4.4%	12.6%	7.1%	6.0%
5	18.9%	0.0%	0.3%	0.8%	0.8%	2.4%	0.8%	0.5%	0.3%	17.5%	10.9%	12.0%	8.7%	16.9%	8.2%	14.2%	8.7%	2.2%
6	3.5%	0.3%	0.8%	0.8%	0.5%	0.0%	0.3%	0.8%	0.0%	8.2%	10.9%	9.8%	6.6%	15.3%	2.2%	2.7%	7.7%	1.1%
7	0.3%	0.3%	0.5%	0.0%	0.3%	0.0%	1.3%	0.3%	0.3%	9.3%	8.2%	12.0%	5.5%	4.9%	3.3%	25.7%	4.9%	3.3%
8	0.5%	0.0%	0.0%	0.0%	0.5%	0.5%	0.0%	0.8%	0.3%	18.0%	12.0%	12.6%	10.4%	8.7%	7.1%	12.6%	9.8%	3.3%
9	0.0%	0.3%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%	0.0%	6.0%	2.7%	2.2%	1.1%	0.5%	4.4%	4.9%	6.6%	0.0%
10	0.0%	0.0%	0.0%	0.3%	0.0%	0.8%	0.0%	0.0%	0.0%	29.5%	26.2%	24.6%	24.0%	12.6%	53.0%	6.6%	26.8%	5.5%

Step 4 – Likelihoods

Say the first test subject has the following **signature**:

Obs.	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
1	3	1	1	1	2	2	3	1	1

The probabilities for each test score are highlighted in **black** (benign = orange, malignant = blue). The **naïve** assumption says that the **likelihood** is obtained by multiplying the **individual probabilities**.

Score	Benign: Clump Thickness	Benign: Uniformity of Cell Size	Benign: Uniformity of Cell Shape	Benign: Marginal Adhesion	Benign: Single Epithelial Cell Size	Benign: Bare Nuclei	Benign: Bland Chromatin	Benign: Normal Nucleoli	Benign: Mitoses
1	30.9%	**83.2%**	**78.5%**	**81.9%**	10.9%	85.4%	33.2%	**88.0%**	**97.6%**
2	10.1%	7.7%	10.9%	8.5%	**78.5%**	**4.3%**	35.4%	6.9%	1.3%
3	**21.5%**	6.6%	6.1%	6.4%	6.9%	2.9%	**27.4%**	2.4%	0.3%
4	14.4%	1.6%	2.9%	1.1%	1.6%	1.3%	1.6%	0.3%	0.0%
5	18.9%	0.0%	0.3%	0.8%	0.8%	2.4%	0.8%	0.5%	0.3%
6	3.5%	0.3%	0.8%	0.8%	0.5%	0.0%	0.3%	0.8%	0.0%
7	0.3%	0.3%	0.5%	0.0%	0.3%	0.0%	1.3%	0.3%	0.3%
8	0.5%	0.0%	0.0%	0.0%	0.5%	0.5%	0.0%	0.8%	0.3%
9	0.0%	0.3%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.3%	0.0%	0.8%	0.0%	0.0%	0.0%
Likelihood									**9.06E-04**

Score	Malignant: Clump Thickness	Malignant: Uniformity of Cell Size	Malignant: Uniformity of Cell Shape	Malignant: Marginal Adhesion	Malignant: Single Epithelial Cell Size	Malignant: Bare Nuclei	Malignant: Bland Chromatin	Malignant: Normal Nucleoli	Malignant: Mitoses
1	1.1%	**1.6%**	**1.1%**	**12.6%**	0.5%	6.6%	1.1%	**14.8%**	**50.8%**
2	1.6%	2.7%	3.3%	8.7%	**9.8%**	**4.4%**	3.3%	3.3%	13.1%
3	**4.9%**	11.5%	8.7%	12.0%	16.4%	6.0%	**16.4%**	10.4%	14.8%
4	3.8%	13.1%	13.7%	10.4%	14.2%	4.4%	12.6%	7.1%	6.0%
5	17.5%	10.9%	12.0%	8.7%	16.9%	8.2%	14.2%	8.7%	2.2%
6	8.2%	10.9%	9.8%	6.6%	15.3%	2.2%	2.7%	7.7%	1.1%
7	9.3%	8.2%	12.0%	5.5%	4.9%	3.3%	25.7%	4.9%	3.3%
8	18.0%	12.0%	12.6%	10.4%	8.7%	7.1%	12.6%	9.8%	3.3%
9	6.0%	2.7%	2.2%	1.1%	0.5%	4.4%	4.9%	6.6%	0.0%
10	29.5%	26.2%	24.6%	24.0%	12.6%	53.0%	6.6%	26.8%	5.5%
Likelihood									**5.85E-11**

Step 5 – Posterior Probabilities

Recall that

$$\text{posterior} \propto \text{prior} \times \text{likelihood}.$$

Then,

Class	Prior	Likelihood	Posterior
Benign	6.73E-01	9.06E-04	6.09E-04
Malignant	3.27E-01	5.85E-11	1.92E-11

Based on NBC, we suspect that the first test subject's tumour is **benign** (and would be classified as such).

Step 6 – Classification Decisions

We can compute the **posterior probabilities** for each test case in a similar manner, which yields:

- **benign** – $78/140 = 55.7\%$
- **malignant** – $62/140 = 44.3\%$

So how well did NBC perform as a classifier?

Observation	Posterior (numerator)	
	Benign	Malignant
1	6.09E-04	1.92E-11
2	0.00E+00	1.50E-09
3	6.35E-04	2.14E-11
4	7.97E-04	2.87E-11
5	6.00E-04	5.85E-12
86	1.53E-04	5.68E-12
87	1.48E-02	1.92E-12
88	1.35E-04	2.24E-11
89	2.26E-02	1.28E-12
90	1.38E-06	8.81E-10
136	3.14E-03	3.83E-12
137	1.66E-03	4.07E-13
138	2.15E-04	1.48E-12
139	6.96E-03	6.39E-13
140	0.00E+00	2.81E-10

3.10.3 – Debriefing

		Prediction		
		Benign	Malignant	
Truth	Benign	77	5	82
	Malignant	1	57	58
		78	62	140

Overall **misclassification** rate:
$6/140 = 4.3\%$

Benign misclassification (FP) rate:
$5/82 = 6.1\%$

Malignant misclassification (FN) rate: $1/58 = 1.7\%$

Malignant detection is **more** accurate than benign detection.

Is the overall misclassification rate good? Well that depends on many things, such as:

- the **shape** of the data;
- the ultimate **application**;
- the **cost of misclassification**, etc.

But why did NBC fail in some cases?

Performance Evaluation

Note that some of the **posterior probabilities** are 0, because we did not jointly observe their scores in Tr.

Take a look at ID 23, say. Is anything **unusal** happening there?

ID	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	True class	Posterior	
											Benign	Malignant
2	6	8	8	1	3	4	3	7	1	Benign	0	1.50E-09
23	1	1	1	1	10	1	1	1	1	Benign	0	5.44E-13
33	8	4	4	5	4	7	7	8	2	Benign	0	1.43E-09
61	4	6	5	6	7	0	4	9	1	Benign	0	2.22E-09
74	3	4	5	3	7	3	4	6	1	Benign	3.80E-15	4.41E-10
102	6	3	2	1	3	4	4	1	1	Malignant	1.74E-09	8.57E-10

Generally, the characteristic of benign tumours is to have **low measurement scores** throughout, and $vice\text{-}versa$.

In case with ID 23, **only** the score from **single epithelial cell size** was abnormal (is it a 0? a 1? a 10**?**). All other scores point to ID 23 being **benign** (note that the posterior for malignant is **quite small as well**).

Also, if we did not observe a score in Tr, the posterior probability for any observation with that score has to be 0. How can we avoid this issue?

	Benign								
Score	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
1	30.9%	83.2%	78.5%	81.9%	10.9%	85.4%	33.2%	88.0%	97.6%
2	10.1%	7.7%	10.9%	8.3%	78.5%	4.3%	35.4%	6.9%	1.3%
3	21.5%	6.6%	6.1%	6.4%	6.9%	2.9%	27.4%	2.4%	0.3%
4	14.4%	1.6%	2.9%	1.1%	1.6%	1.3%	1.6%	0.3%	0.0%
5	18.9%	0.0%	0.3%	0.8%	0.8%	2.4%	0.8%	0.5%	0.3%
6	3.5%	0.3%	0.8%	0.8%	0.5%	0.0%	0.3%	0.8%	0.0%
7	0.3%	0.3%	0.5%	0.0%	0.3%	0.0%	1.3%	0.3%	0.3%
8	0.5%	0.0%	0.0%	0.0%	0.5%	0.5%	0.0%	0.8%	0.3%
9	0.0%	0.3%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.3%	0.0%	0.8%	0.0%	0.0%	0.0%

Alternative NBC

For each variable X_j, set a **base probability** ε_j, say $\varepsilon_j = 10^{-8}\%$.

Find the smallest $\neq 0$ probability $\min_j = \min\{x_j \mid \mathbf{x} \in \mathrm{Tr}\}$ (0.3%).

Set $\nu_j = \min_j \times \varepsilon_j$ $(3\times 10^{-11}\%)$. Sub the ρ_j **zero prob.** in column j by ν_j.

The probabilities now add to $1 + \rho_j \nu_j$ $(1+2(3\times 10^{-11}\%)$; **normalize** column j before proceeding to steps 4 – 6.

Original		Step 1		Step 2
Single Epithelial Cell Size		Single Epithelial Cell Size		Single Epithelial Cell Size
10.9%		10.9%		10.9%
78.5%		78.5%		78.5%
6.9%		6.9%		6.9%
1.6%	⇒	1.6%	⇒	1.6%
0.8%		0.8%		0.8%
0.5%		0.5%		0.5%
0.3%		0.3%		0.3%
0.5%		0.5%		0.5%
0.0%		3E-11%		3E-11%
0.0%		3E-11%		3E-11%

By assigning $\varepsilon_j \equiv 10^{-8}\%$ for all j, the total error rate decreased from $6/140$ to $5/140$.

The posterior probabilities on Te are now all > 0, but most of them are **quite small**.

ID 23 is now **correctly** classified as benign even with a **strong penalty** for the unusual measurement $\implies$ potential **recording/measurement error**.

The overall structure is **preserved** (**good**).

With enough evidence, we can correct the classification (also good).

The alternative probability can be **controlled**, as long as N is reasonably large (to cover most potential scenarios) $\implies$ approach is **sound**.

		Prediction (original)		
		Benign	Malignant	
Truth	Benign	77	5	82
	Malignant	1	57	58
		78	62	140

⇒

		Prediction (alternative)		
		Benign	Malignant	
Truth	Benign	78	4	82
	Malignant	1	57	58
		79	61	140

ID	True class	Posterior (original)			Posterior (alternative)	
		Benign	Malignant		Benign	Malignant
2	Benign	0	1.50E-09		9.03E-30	1.50E-09
23	Benign	0	5.44E-13	⇒	7.38E-13	5.47E-13
33	Benign	0	1.43E-09		1.64E-26	1.43E-09
61	Benign	0	2.22E-09		6.01E-24	2.22E-09
74	Benign	3.80E-15	4.41E-10		3.89E-15	4.43E-10
102	Malignant	1.74E-09	8.57E-10		1.78E-09	8.62E-10

3.10.4 – Notes and Comments

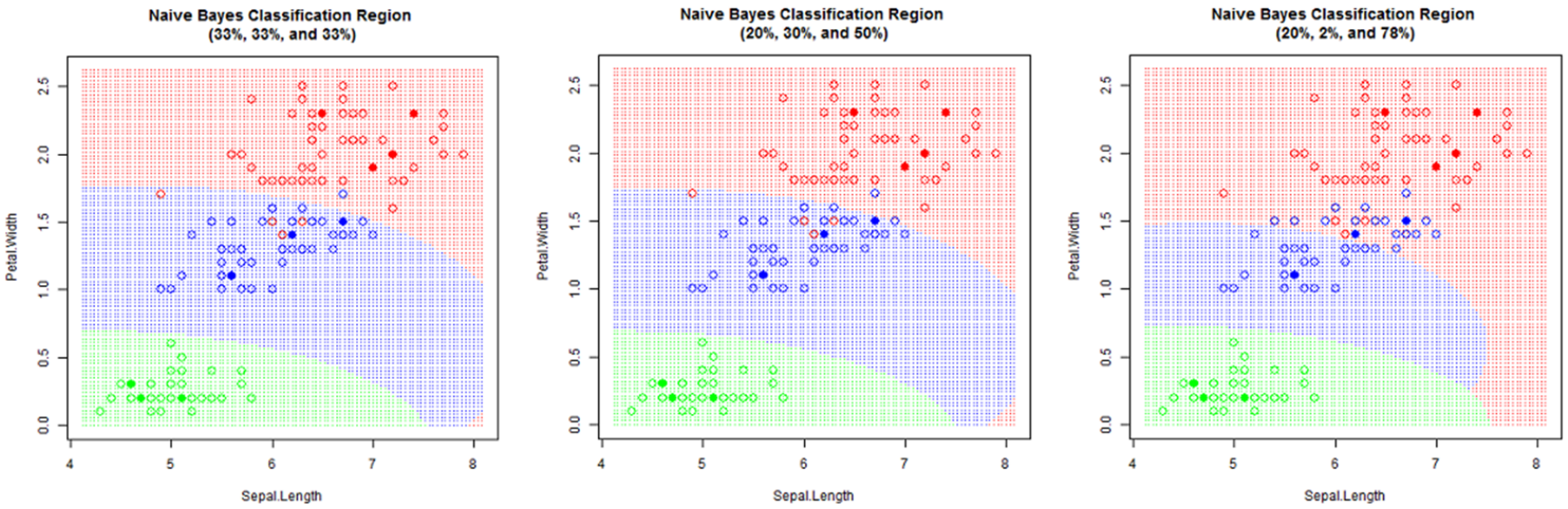
In general, the variables are **NOT independent**, yet NBC still seems to works well with test data (**ham** vs. **spam**).

NBC may be **robust** against departure from the **independence assumption**... the condition appears to be more **sufficient** than **necessary**.

Dependency among variable may change the **posterior probabilities**, but the class with maximum posterior probability is **often unchanged** (Domingos and Pazzani, 1997).

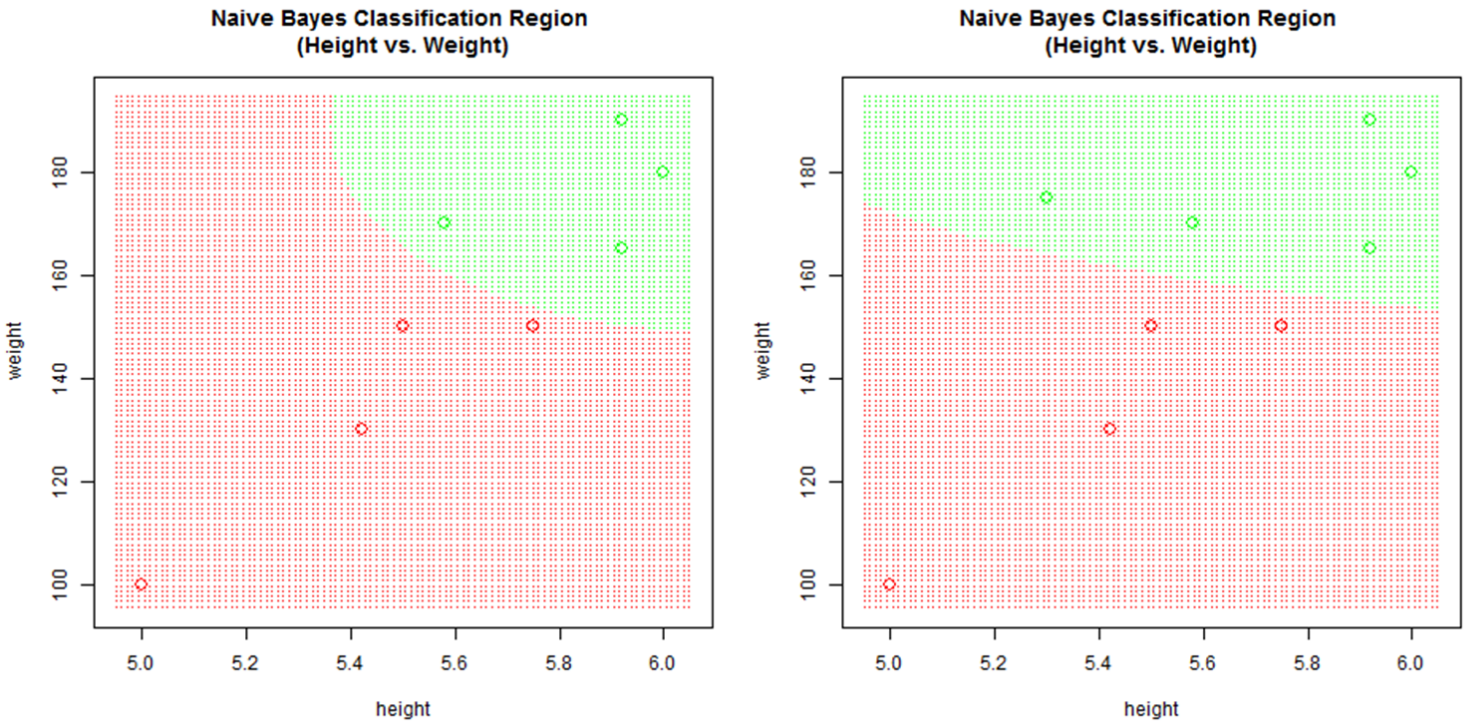
NBC does not produce good **probability estimates** in general because of the naïve assumption.

Effects of Prior Probabilities

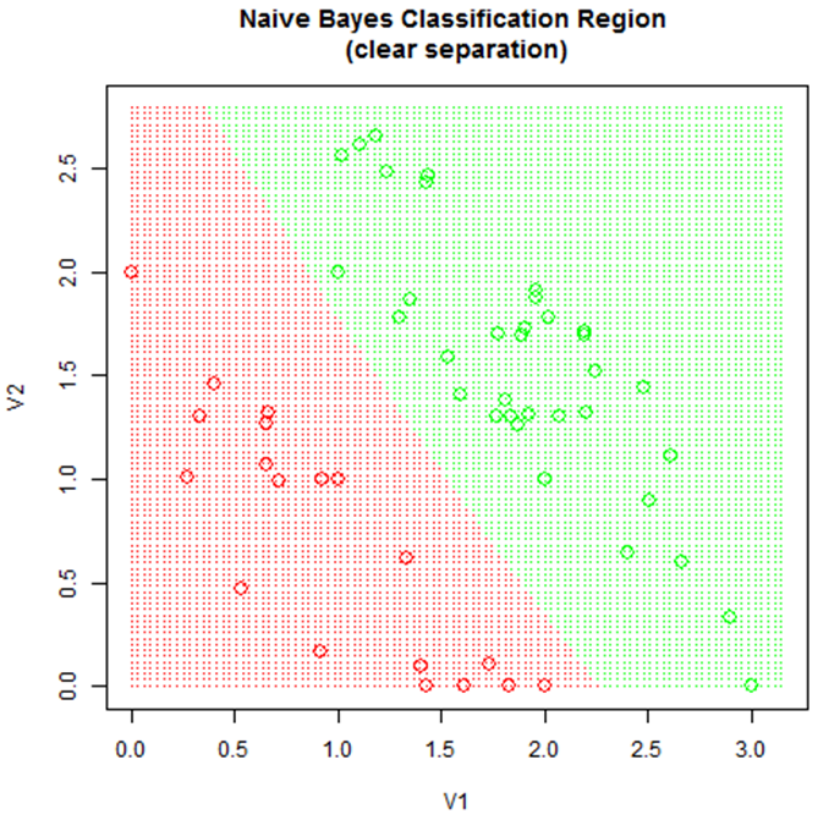


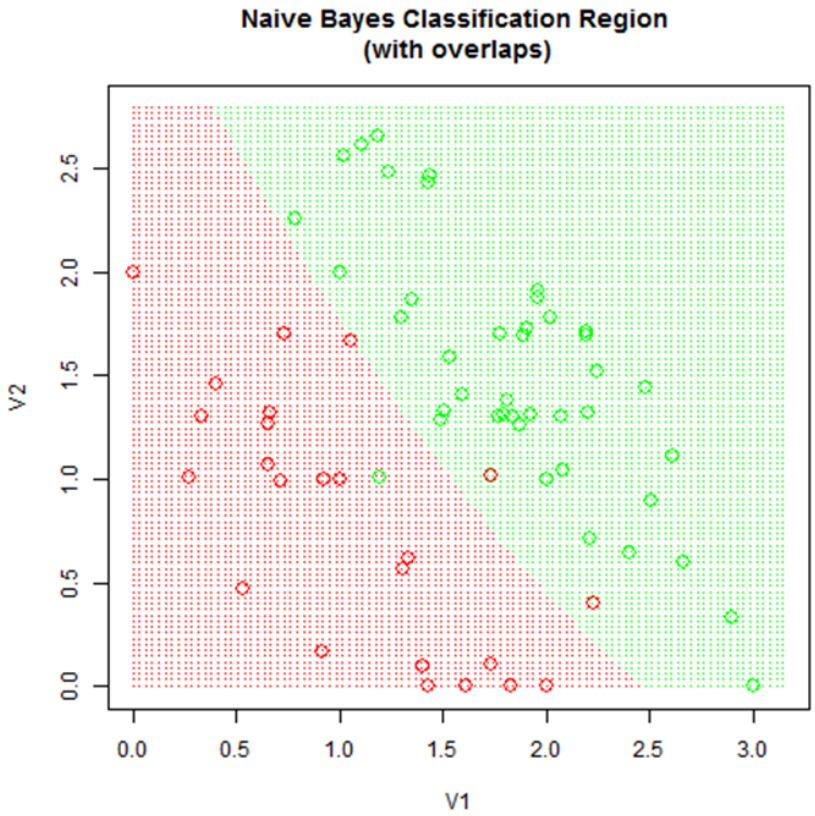
When there is some **overlap** in Tr, and the **prevalence rates** vary, then the classification result may be greatly affected.

Effects of Outliers



When $|\mathrm{Tr}|$ is **small**, outliers may modify the classifier to a **great** extent.





When $|\mathrm{Tr}|$ is **large**, outliers only modify the classifier to a **small** extent.

The Rest of the Classification and SL Picture

SVM Regression? Other types of boosting?

References

T.Hastie, R.Tibshirani, and J.Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd ed. Springer, 2008.

G.James, D.Witten, T.Hastie, and R.Tibshirani, An Introduction to Statistical Learning: With Applications in R. Springer, 2014.

C.C.Aggarwal, Ed., Data Classification: Algorithms and Applications. CRC Press, 2014.

C.C.Aggarwal, Data Mining: The Textbook. Springer, 2015.

B.Boehmke and B.Greenwell, Hands on Machine Learning with R. CRC Press.

F.Chollet, Deep Learning with Python, 1st ed. Manning Publications Co., 2017

Wikipedia, "Binary classification," 2021.

R.V.Hogg and E.A.Tanis, Probability and Statistical Inference, 7th ed. Pearson/Prentice Hall, 2006.

N.V.Chawla, K.W.Bowyer, L.O.Hall, and W.P.Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," Journal of A.I. Research, vol. 16, pp. 321–357, 2002.

M.H.Kutner, C.J.Nachtsheim, J.Neter, and W.Li, Applied Linear Statistical Models. McGraw Hill Irwin, 2004.

D.R.Hofstadter, Gödel, Escher, Bach: an Eternal Golden Braid. New York, NY: Basic Books, 1979.

F.Provost and T.Fawcett, Data Science for Business. O'Reilly, 2015.

I.Goodfellow, Y.Bengio, and A.Courville, Deep Learning. MIT Press, 2016.

D.Robinson, "What's the difference between data science, machine learning, and artificial intelligence?" Variance Explained, Jan. 2018.

C.Sheppard, Tree-Based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting. CreateSpace Independent Publishing Platform, 2017.

T.Hofmann, B.Schölkopf, and A.J.Smola, "Kernel Methods in Machine Learning," Annals of Statistics, vol. 36, no. 3, pp. 1171–1220, 2008.

N.Deng, Y.Tian, and C.Zhang, Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions. CRC Press/Chapman; Hall, 2013.

3Blue1Brown, "Deep Learning."

D.Gershgorn, "There's a glaring mistake in the way AI looks at the world," 2017.

A.Turing, "Computing machinery and intelligence," Mind, 1950.

Wikipedia, "Artificial intelligence," 2020.

M.Caudill, "Neural networks primer, part 1," AI Expert, vol. 2, no. 12, pp. 46–52, Dec. 1987.

Y.Dauphin, R.Pascanu, C.Gulcehre, K.Cho, S.Ganguli, and Y.Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." 2014.

R.S.Sutton, "Two problems with backpropagation and other steepest-descent learning procedures for networks," 1986.

S.Ruder, "An overview of gradient descent optimization algorithms." 2016.

I.J.Goodfellow, J.Shlens, and C.Szegedy, "Explaining and harnessing adversarial examples," ICLR, 2015.

G.James, D.Witten, T.Hastie, and R.Tibshirani, An Introduction to Statistical Learning: With Applications in R. Springer, 2014.

R.M.Levenson, E.A.Krupinski, V.M.Navarro, and E.A.Wasserman, "Pigeons (columba livia) as trainable observers of pathology and radiology breast cancer images," PLOS ONE, vol. 10, no. 11, pp. 1–21, Nov. 2015, doi: 10.1371/journal.pone.0141357.

J.Friedman, T.Hastie, and R.Tibshirani, "Additive logistic regression: A statistical view of boosting," Annals of Statistics, vol. 28, p. 2000, 1998.