School of Electronic Engineering and Computer Science
Queen Mary University of London

ECS766 Data Mining
Week 4: Classification II

Dr Jesús Requena Carrión

16 Oct 2019

Agenda

Recap (with some extras)

Understanding classification performance

Bayesian methods

Tree methods

More on errors, optimisation and validation

The best diagnostic machine

As the hospital lead data scientist, you are responsible for selecting the best diagnostic machine for a certain disease. You are presented with three machines A, B and C, which you test in a group of patients whose diagnose (presence or absence of the disease) you already know.

The resulting accuracy of each machine after testing is shown below. Which one would you choose?

(a) Machine A: 6 % of correct diagnoses

(b) Machine B: 89 % of correct diagnoses

(c) Machine C: 56 % of correct diagnoses

What is the most appropriate quality metric?

You are still the hospital's lead data scientist and ask yourself what the most useful metric for evaluating a diagnostic machine could be. Which option would you choose among the following?

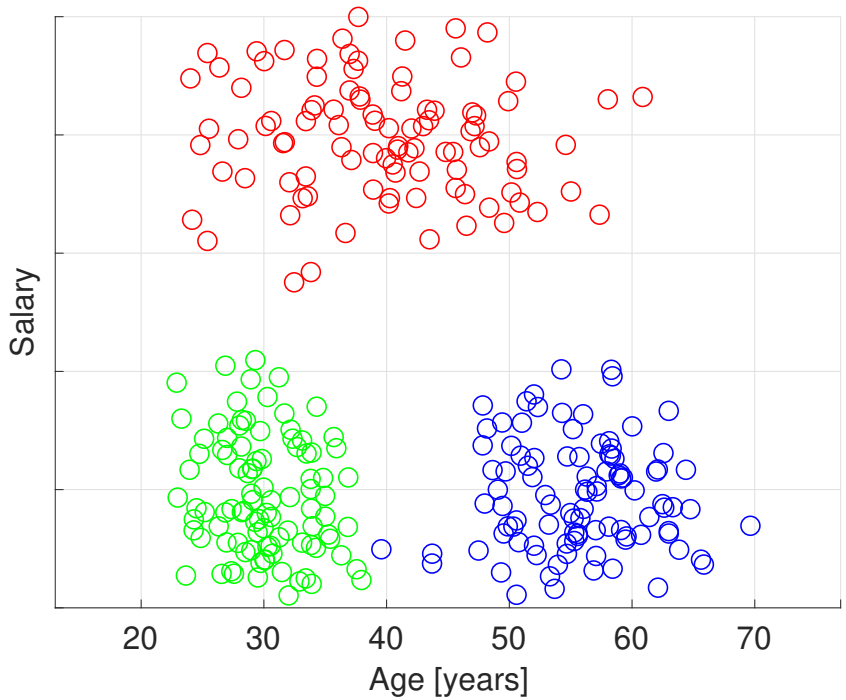
(a) Accuracy, i.e. proportion of correct diagnoses

(b) Either proportion of healthy patients correctly identified or proportion of ill patients correctly identified

(c) Something else

Classifiers in the predictor space

Datasets can be visualised as **collections of points in the predictor space, represented by different symbols**.



Opinion: o → Good, o → Neutral, o → Bad

- The **coordinates** of each point correspond to the **values of its predictors**
- The **symbol** represents the **value of its response**

Classifiers as decision regions

By representing datasets in the predictor space, we can see a classifier as **decision regions** (or **boundaries between regions**). So far, we have discussed **linear** (parametric) and **kNN** (non-parametric) classifiers.

Linear classifiers are defined by a coefficient vector $\boldsymbol{w}$. Specifically, the **equation $\boldsymbol{w}^T\boldsymbol{x} = 0$ defines a linear boundary** in the predictor space (point in 1D, straight line in 2D, plane in 3D, etc).

Furthermore, given a point with an associated vector $\boldsymbol{x}_i$, the quantity $\boldsymbol{w}^T\boldsymbol{x}_i$ **can be seen as the distance to the boundary**. The sign of this distance depends on the side of the boundary where $\boldsymbol{x}_i$ resides.

Certainty in linear classifiers and the logistic function

We used the notion of distance $\boldsymbol{w}^T\boldsymbol{x}_i$ to define the **classifier's certainty that a sample belongs to one class or another**. Using the language of probability, this certainty is a **conditional probability**.

If $\boldsymbol{w}^T\boldsymbol{x} = 0$ separates classes A and B, then:

- $p(\boldsymbol{x}_i) = P(y_i = B|\boldsymbol{x}_i)$ is the probability that $\boldsymbol{x}_i$ belongs to class B
- $1 - p(\boldsymbol{x}_i) = P(y_i = A|\boldsymbol{x}_i)$ is the probability that $\boldsymbol{x}_i$ belongs to A

where $p(\boldsymbol{x}_i)$ is a **logistic function**. We can denote its dependence on $\boldsymbol{w}$ by writing $p(\boldsymbol{x}_i; \boldsymbol{w})$ instead.

For a collection of labelled samples, the classifier's **global certainty** is:

$$L(\boldsymbol{w}) = \prod_{y_i=A} (1 - p(\boldsymbol{x}_i; \boldsymbol{w})) \prod_{y_i=B} p(\boldsymbol{x}_i; \boldsymbol{w})$$

The **best** classifier can be defined as the one with the **highest certainty**.

Positive and negative margins

Some textbooks use the notion of margin as a distance from a sample to the boundary such that:

- If the sample is correctly classified, the margin is positive
- If the sample is misclassified, the margin is negative

Accordingly, good classifiers will aim at **maximising the margin**.

There is a simple way to obtain the margin by expressing the label values as the values 1 and -1. The margin m_i of $\{(\boldsymbol{x}_i, y_i)\}$ can then obtained as

$$m_i = y_i \left[\boldsymbol{w}^T \boldsymbol{x}_i \right]$$

Irrespective of the class they belong to, **correctly classified samples have a positive margin** and **misclassified samples a negative one**.

Agenda

Recap (with some extras)

Understanding classification performance

Bayesian methods

Tree methods

More on errors, optimisation and validation

Accuracy and error rate

Our notion of **goodness** is encapsulated in a **metric**, which allows us to define the **best solution**. So far, we have used two equivalent metrics in classifiers:

- **Accuracy**: Proportion of correctly classified samples
- **Error rate**: Proportion of mistakes, a.k.a. misclassification rate

Accuracy and error rate are **easy to calculate and interpret**. However:

- All errors are considered, **irrespective of the class**
- **Imbalanced datasets** are not accounted for
- The notion of classifier **calibration** cannot be readily applied

Confusion matrix

The goal of a **confusion** or **contingency matrix** is to show the classifier's performance by looking at each class (**positive** and **negative**) separately.

		Actual class	
		Positive	Negative
Predicted class	Positive	True positive	False positive
	Negative	False negative	True negative

Note that the number of actual positive samples is TP+FN, the number of negative samples is FP+TN and finally the total number of samples is TP+FN+FP+TN.

Confusion matrix

The cells in the confusion matrix can also represent **rates** instead of **number of samples**, which is useful when dealing with imbalanced datasets:

		Actual class	
		Positive	Negative
Predicted class	Positive	TPR	FPR
	Negative	FNR	TNR

- TPR = TP/(TP+FN) → sensitivity
- TNR = TN/(TN+FP) → specificity
- FPR = FP/(FP+TN) → fall-out or 1-specificity
- FNR = FN/(FN+TP) → miss rate
- A = (TP+TN)/(TP+FP+FN+TN) → accuracy
- E = (FP+FN)/(TP+FP+FN+TN) → error rate

Confusion matrix: example

Number of samples

	Actual	
Predicted	4	1
	2	3

Rates

	Actual	
Predicted	4/6	1/4
	2/6	3/4

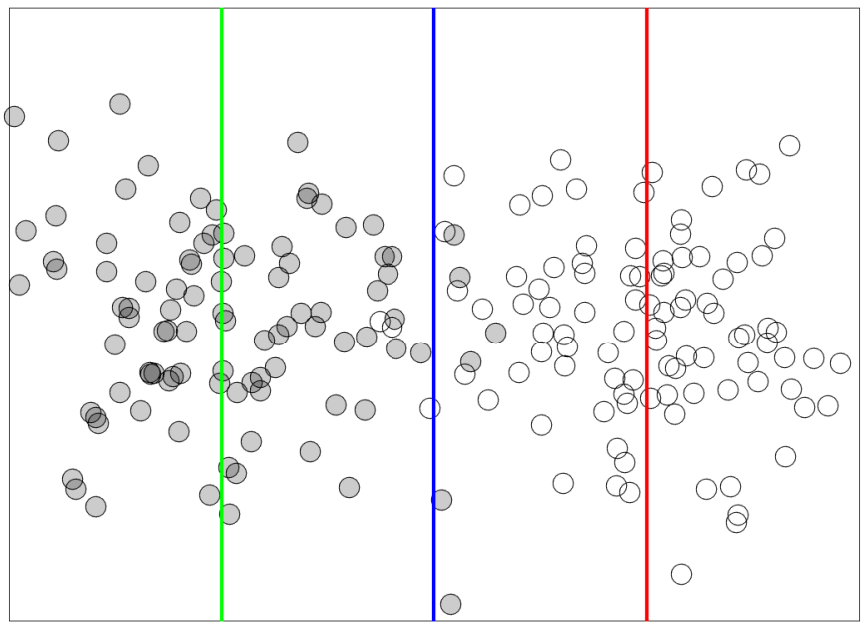
- TP $= 4 \rightarrow$ TPR $= 4/6 \approx 0.66$
- FP $= 1 \rightarrow$ FPR $= 1/4 = 0.25$
- FN $= 2 \rightarrow$ FNR $= 2/6 \approx 0.33$
- TN $= 3 \rightarrow$ TNR $= 3/4 = 0.75$
- A $= (4+3)/10 = 7/10 = 0.7$
- E $= (1+2)/10 = 3/10 = 0.3$

Confusion matrix: uses

The confusion matrix **defines different performance metrics** that can be considered for different applications:

- A bank offering loans might prefer to reduce the number of *bad* businesses that received a loan (FPR) rather than reduce the number of *good* businesses who are rejected (FNR)
- A high security system might prefer to reduce the number of undetected break-ins (FNR), rather than reduce the number of false alarms (FPR).
- A medical screening technique might aim at increasing the success in identifying an ill patient (TPR) rather than reducing the number of healthy patients who are wrongly diagnosed (FPR)

Moving the boundary: Calibration I

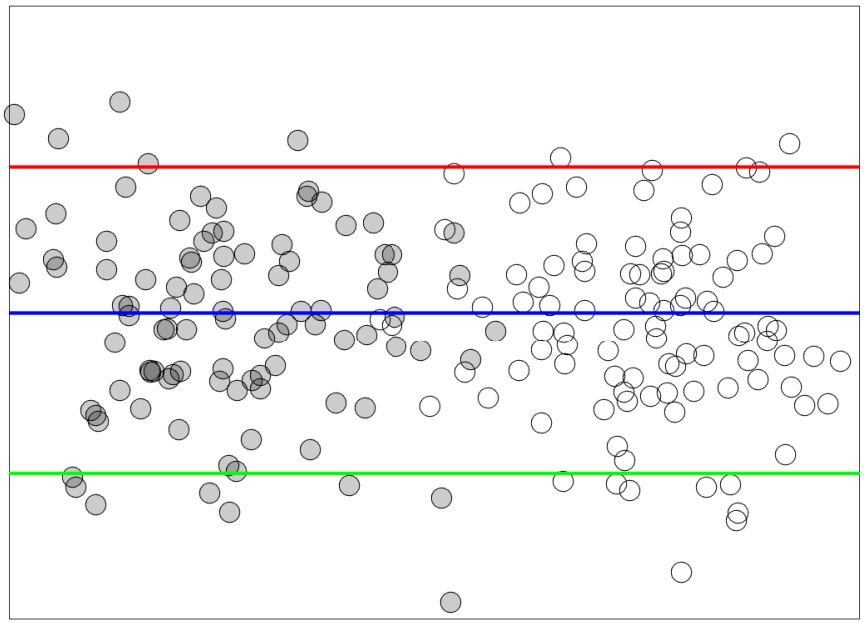


	Actual	
Predicted	1.00	0.50
	0	0.50

	Actual	
Predicted	0.95	0.05
	0.05	0.95

	Actual	
Predicted	0.50	0
	0.50	1.00

Moving the boundary: Calibration II



	Actual	
Predicted	0.05	0.05
	0.95	0.95

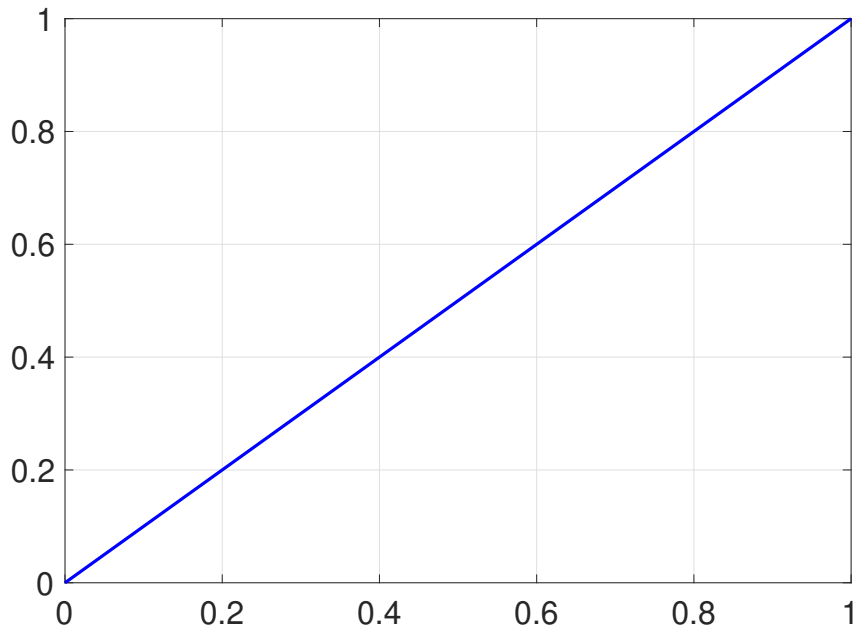
	Actual	
Predicted	0.50	0.50
	0.50	0.50

	Actual	
Predicted	0.92	0.92
	0.08	0.08

The ROC plane

Ideally, we would like our classifier to have a **TPR (sensitivity) as close to 1 as possible** and a **FPR (1-specificity) as close to 0 as possible**.

The ROC (*receiver operating characteristic*) plane represents the performance of a classifier in terms of its TPR and FPR. The closer to the top left corner, the better.



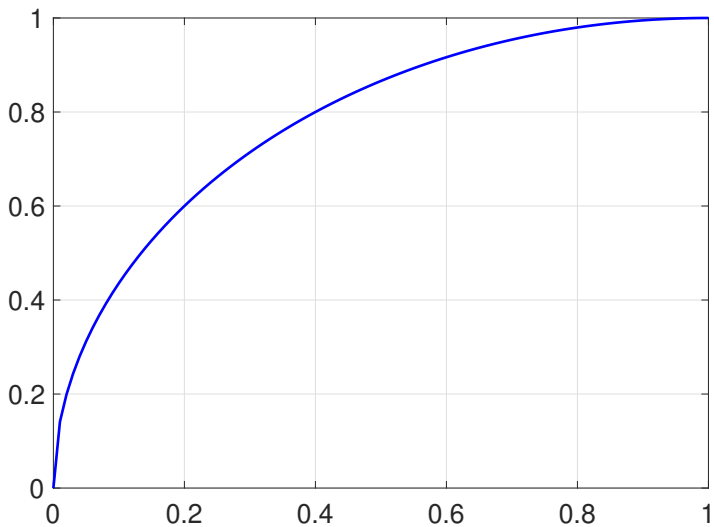
Calibration in the ROC plane: The ROC curve

Some classifiers can be calibrated by moving their boundary. This allows us to change its behavious until our target performance is achieved. The ROC curve shows how the **TPR and the FPR change as we calibrate the boundary of our classifier**.

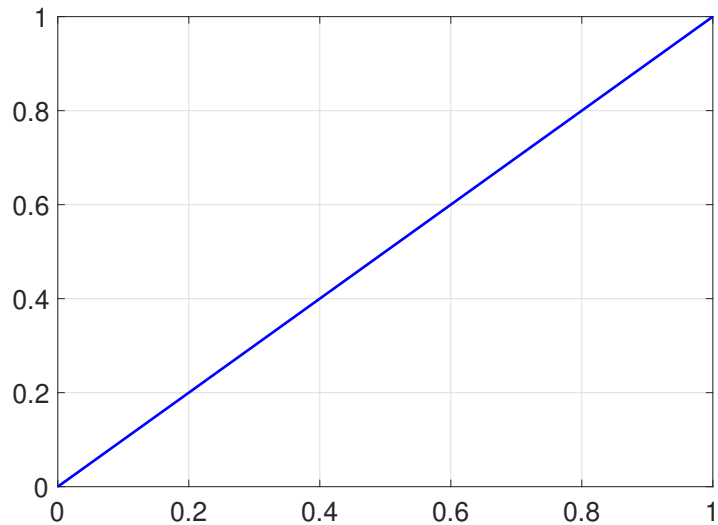
The so-called **area under the curve (AUC) is a measure of goodness** for a classifier that can be calibrated and it gives the average value of sensitivity for all possible values of specificity. Good classifiers will have AUC close to 1, bad classifiers close to 0.5.

Calibration in the ROC plane: The ROC curve

Good classifier (AUC $\approx$ 0.8)



Bad classifier (AUC = 0.5)



Can you think of a classifier whose AUC<0.5?

Agenda

Recap (with some extras)

Understanding classification performance

Bayesian methods

Tree methods

More on errors, optimisation and validation

Quick estimation

A certain disease has a prevalence of 0.1 %, i.e. we expect one individual out of 1000 to carry the disease. There is a medical test available such that:

- If you have the disease, the test is 100 % accurate
- If you don't have the disease, the test is 95 % accurate

If you take the medical test and the result is positive, the probability that you actually have the disease is:

(a) >97 %

(b) ≈ 50 %

(c) <3 %

Another view of classifiers

It can be shown that **the classifier with the lowest error rate is the one that assigns a sample to the most likely class**, i.e. the class C_j for which $P(y_i = C_j|\boldsymbol{x}_i)$ is largest. This is called the **Bayes classifier**.

Up untill now, we have studied two classifiers, namely **logistic regression** and **kNN**. They actually **imposed** a form for $P(y_i = C_j|\boldsymbol{x}_i)$:

- Logistic regression, uses the logistic function.
- kNN, uses the proportion of neighbours belonging to each class.

Can we do any better? How could we use any insight we might have about our data?

The Bayesian trick

Bayes Theorem is a fundamental result in statistics. If we have a sample $(\boldsymbol{x}_i, y_i)$ and a collection of classes C_j, Bayes Theorem can be written as:

$$P(y_i = C_j|\boldsymbol{x}_i) = \frac{P(\boldsymbol{x}_i|y_i = C_j)P(C_j)}{\sum_j P(\boldsymbol{x}_i|y_i = C_j)P(C_j)} = \frac{P(\boldsymbol{x}_i|y_i = C_j)P(C_j)}{P(\boldsymbol{x}_i)}$$

where $P(\cdot)$ denotes probability (we ignored a few nuisances!).

The idea now is

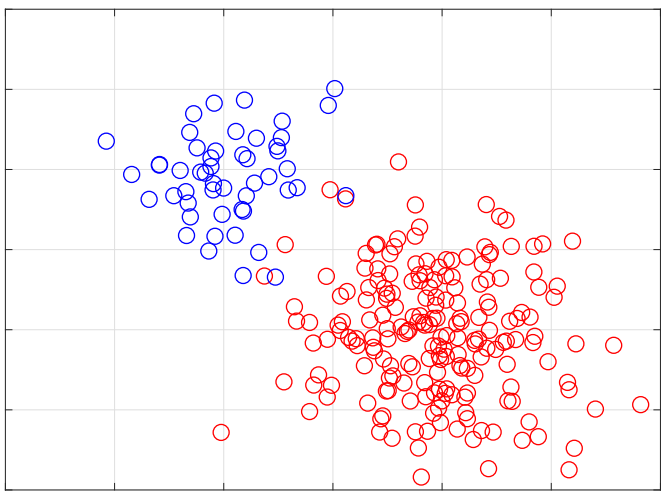
- We want to know $P(y_i = C_j|\boldsymbol{x}_i)$
- If we know the priors $P(\boldsymbol{x}_i|y_i = C_j)$ and $P(C_j)$ we could derive it!
- Do we know anything about those priors? Can we estimate them based on our data?

The Gaussian approximation

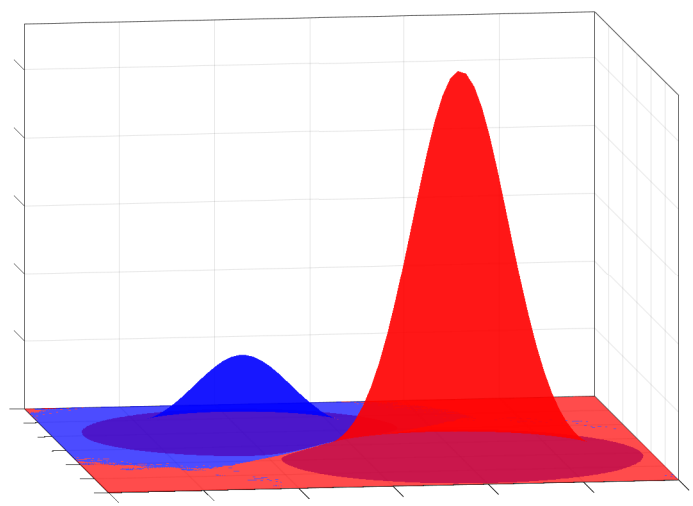
A common assumption is the **Gaussianity** of $P(\boldsymbol{x}_i|y_i = C_j)$:

- We estimate the mean $\boldsymbol{\mu}_j$ and standard deviation $\boldsymbol{\Sigma}_j$ of each class C_j from our data and produce an estimation for $P(\boldsymbol{x}_i|y_i = C_j)$
- Then we estimate $P(C_j)$ from our data
- We use Bayes Theorem to produce $P(y_i = C_j|\boldsymbol{x}_i)$ and create a Bayes classifier

Data samples



$P(\boldsymbol{x}_i|y_i = C_j)P(C_j)$



Bayes classifiers

Bayesian methods lead to decision regions that can be very similar to the ones built by using logistic regression when the distribution of predictors is Gaussian. However:

- Solutions based on **logistic regression can be very unstable**
- For a **small number of samples**, bayesian approaches can produce **more stable solutions**
- It is easier to work in a bayesian framework when we have **more than two classes**
- Using **prior knowledge** can be incorporated into a Bayesian model in a more straightforward way

Agenda

Recap (with some extras)

Understanding classification performance

Bayesian methods

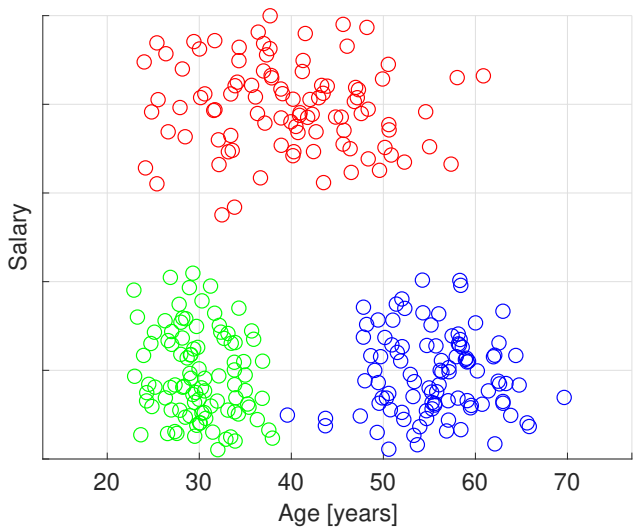
Tree methods

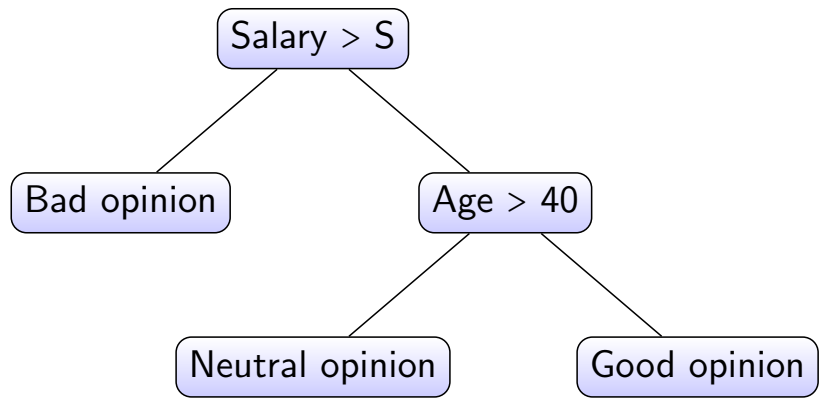
More on errors, optimisation and validation

What is tree?

Tree-based classifiers partition the predictor space by implementing a sequence of splitting rules that can be represented as a tree consisting of **internal nodes**, **branches** and **leaf nodes**.

Consider the following dataset where o = good opinion, o = neutral opinion and o = bad opinion.





Visually, we would conclude: *if your salary is high, your opinion is bad; if it's low and you are young, your opinion is good; otherwise, it's neutral*.

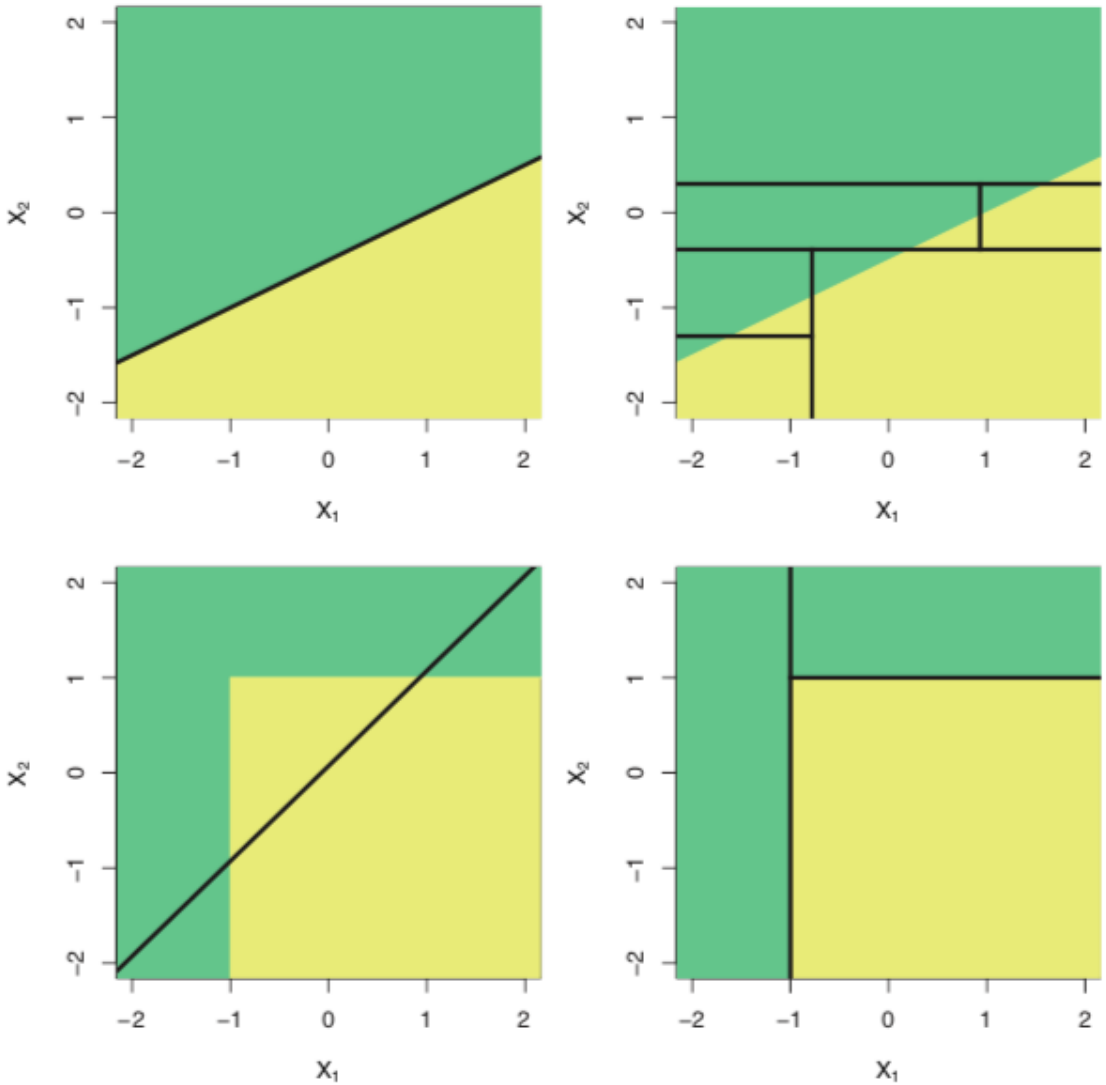
Binary splitting

Given a classification tree, its use and interpretation is very simple. However, how do we build a decision tree? A common approach is **recursive binary splitting**:

- We start with a **single region** at the top of the tree (all the observations)
- We recursively **split each existing region into two**, by choosing the predictor and corresponding threshold that maximises accuracy
- We **stop when a given criterion is met**, such as the number of samples in a region

Common metrics (such as the Gini index and the cross-entropy) measure the *purity* of nodes: a small value indicates a node contains observations from a single class.

Examples



Taken from *An Introduction to Statistical Learning* by G. James et al.

Final notes on trees

In addition to be conceptually simple, **trees work very well with categorical and numerical predictors**.

However, **trees can be non-robust**, i.e. slightly different data can produce very different trees. In addition, the **risk of overfitting the data can be high**, as we might end up learning the individual samples in our training dataset. Typical approaches to overcome these include:

- Pruning a fully grown tree.
- Bagging, random forests and boosting can aggregate many decision trees from the same dataset to increase the predictive performance

Agenda

Recap (with some extras)

Understanding classification performance

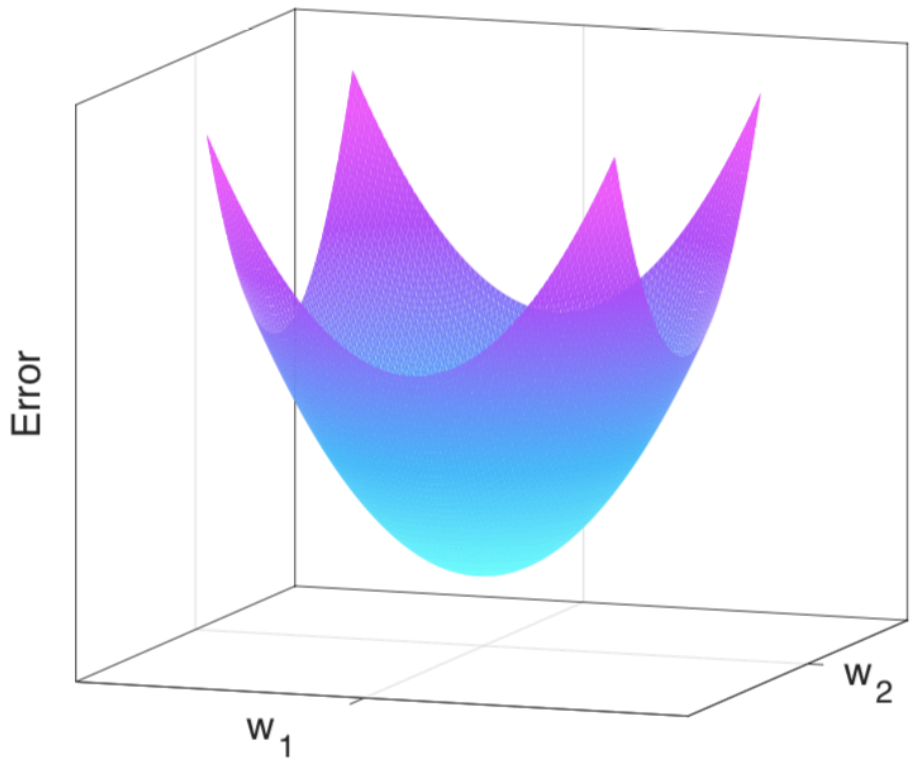
Bayesian methods

Tree methods

More on errors, optimisation and validation

The error surface

The error surface is the function mapping each model (defined by a parameter vector) to its error.



Looking at this surface, what is the best model? How do we find it?

Estimating error surfaces

More often than not, we will **not know the error surface**, we will only have data. Two questions arise:

- **Can we guess (estimate) the shape** of the error surface based on our data?
- **How good** is my guess? How can I improve it?

When dealing with errors, remember that:

- A given model will produce **different errors** when presented with **different data**
- In other words, the **error is a random quantity**
- When we talk about *the* error associated to a model, we are usually referring to an **overall error** in a statistical sense.

The error distribution

Multiple models and limited data: The Monkey Theorem

Random error surfaces

Common gradient descent approaches

In a **limited number** of cases we have **exact solutions** that identify the model with the minimum error. Even in these cases computing it might be **costly** (e.g. in large datasets calculating the inverse $(X^T X)^{-1}$ can be impractical) and therefore in general we will use **numerical approaches**.

Gradient descent updates iteratively the coefficients by estimating the gradient of the error surface. Common implementations include:

- **Batch**: The whole dataset is used for the estimation of the gradient
- **Online**: One sample is used for each gradient estimation
- **Stochastic**: A random subset (mini-batch) is used each time the gradient is estimated

Why do we need validation?

We already know that in general our **training error** will be different from the deployment error. The **test dataset** is used to estimate the predictive performance of our model, but this dataset might not be always available.

Furthermore, we might want to **consider different familes of models** (e.g. polynomials of different degrees) and we might ask which one is the most suitable or what the complexity of our model should be.

Cross-validation methods use **our available dataset** for two main purposes:

- Model assessment (estimate the test error)
- Model selection (identify the complexity of our model)

We reserve the term **test dataset** for the dataset that is used to assess the model during deployment, i.e. once the **final model** has been built.

Validation set approach

The validation set approach is the simplest method. It randomly splits the available dataset into a **training** and a **validation** (or hold-out) dataset.

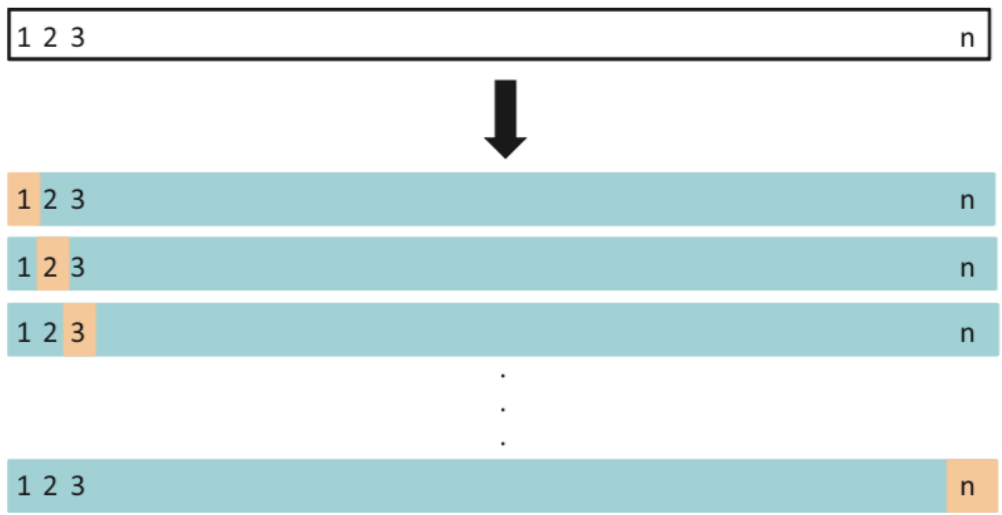
Taken from *An Introduction to Statistical Learning* by G. James et al.

Models are then fitted with the training part and the validation part is used to provide estimates of the test error rate.

- The estimate of the test error **can be highly variable**, depending on the samples that are included in the validation dataset
- Since it uses fewer samples for training, the resulting validation error tends to **overestimate the test error** (training the same model with more samples would result in a model with a lower test error)

Leave-one-out cross-validation (LOOCV)

This method also splits the available dataset into training and validation sets. However, the **validation set contains only one sample**.



Taken from *An Introduction to Statistical Learning* by G. James et al.

Multiple splits are considered and the final validation error is calculated as the average of individual validation errors (for N samples, we produce N splits and obtain N different validation errors).

Leave-one-out cross-validation (LOOCV)

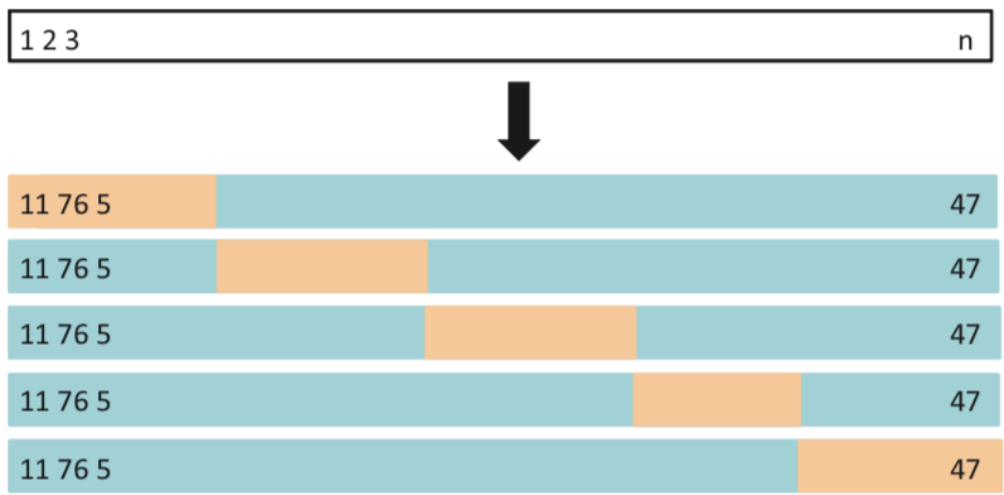
Compared to the validation set approach, the LOOCV is characterised by:

- Lower bias: it doesn't tend to provide overestimated errors, as almost all the samples are used for training
- It always yields the same estimated error, as no random splitting is involved
- It can be expensive to implement, as it requires fitting each model N times
- High error variance, as we work with N models which are almost identical (they are trained on practically the same samples)

k-fold cross-validation

In this approach **we divide randomly our available dataset into k groups** (also known as folds) of approximately equal size:

- We carry k **rounds of training followed by validation**, each one using a different fold for validation and the remaining for training
- The final estimation of the error is the average of the errors estimated in each round



Taken from *An Introduction to Statistical Learning* by G. James et al.

k-fold cross-validation

LOOCV is one a special case of k-fold cross-validation, where $k = N$.
Compared to LOOCV, k-fold cross-validation

- Is less expensive computationally
- Its bias is higher, as it uses fewer samples for training, although lower than the validation set approach
- Its variance is lower, as in LOOCV we are using training datasets that are almost identical, i.e. highly correlated

Validation: Final notes

As in many other areas of Data Science, statistical thinking is crucial to understand validation. Specifically, remember that:

- Validation provides with an **estimation for the performance** of a model during production or test
- The **quality of the estimation itself depends on the samples** being used, the fewer samples, the higher the variance of the estimation
- Validation and training are coupled in the sense that **the more samples we use for validation, the fewer we leave for training**
- Allocating many samples for validation leads to **good performance estimations but poorly fitted models**

Consider two following extreme cases: a validation set approach where 1 sample is used for training and the rest for validation, and the other way around. What would you think about the estimated performances?