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

ECS607/766 Data Mining
Week 4: Classification II

Dr Jesús Requena Carrión

19 Oct 2018

Agenda

Recap (with some extras)

Performance metrics

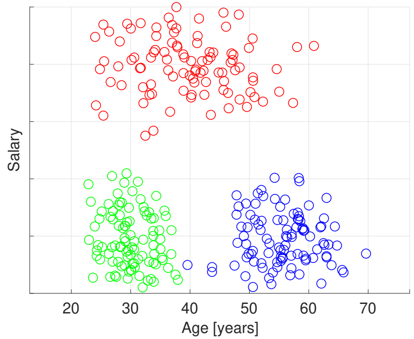
Bayesian methods

Tree methods

Validation approaches

Classifiers in the predictors space

Classification 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 a collection of **decision regions** (or **boundaries between regions**).

We have discussed two types of classifiers: **linear classifiers** (parametric) and **kNN classifiers** (non-parametric).

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

Furthermore, given a point with an extended predictor 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**.

Given an extended predictor vector $\boldsymbol{x}_i$

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

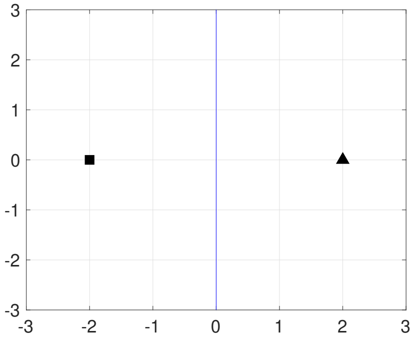
where $p(\boldsymbol{x}_i)$ is a **logistic function**.

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

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

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

Example I



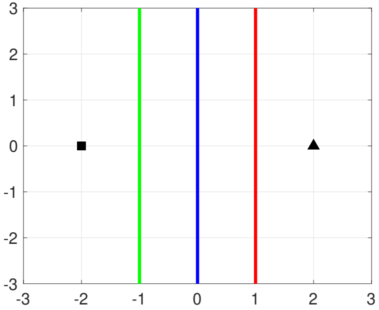
- Let's define $d_i = \boldsymbol{w}^T \boldsymbol{x}_i$
- We can rewrite the logistic function as

$$p(d_i) = \frac{e^{d_i}}{1 + e^{d_i}}$$

- For instance $p(0) = 0.5$, $p(1) \approx 0.73$, $p(2) \approx 0.88$, $p(-1) \approx 0.27$ and $p(-2) \approx 0.12$

Then $p(\triangle) \approx 0.88$, $1 - p(\square) \approx 0.88$ and $L = p(\triangle)\,(1 - p(\square)) \approx 0.77$.

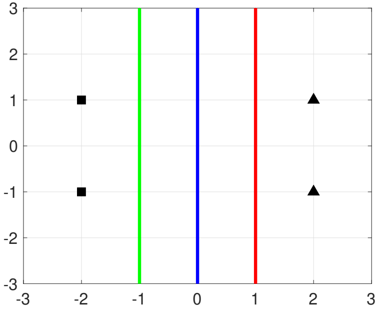
Example II



The global certainty of each classifier (i.e. boundary) is:

- $L = p(\triangle)\,(1 - p(\square)) \approx 0.70$
- $L = p(\triangle)\,(1 - p(\square)) \approx 0.77$
- $L = p(\triangle)\,(1 - p(\square)) \approx 0.70$

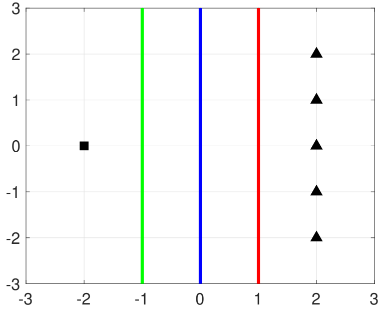
Example III



The global certainty of each classifier (i.e. boundary) is:

- $L \approx 0.49$
- $L \approx 0.60$
- $L \approx 0.49$

Example IV



The global certainty of each classifier (i.e. boundary) is:

- $L \approx 0.20$
- $L \approx 0.47$
- $L \approx 0.57$

Align your metrics with your objectives

Agenda

Recap (with some extras)

Performance metrics

Bayesian methods

Tree methods

Validation approaches

Accuracy and error rate

Our notion of **goodness** is formulated in a **performance metric**, which in turn allows us to define the **best solution** to a given problem. So far, we have considered two equivalent metrics:

- **Accuracy**: Proportion of correctly classified samples
- **Error rate**: Proportion of mistakes

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

- All mistakes are considered, **irrespective of the class**
- **Imbalanced datasets** are not accounted for
- The notion of classifier **calibration is absent**

Confusion matrix

The goal of a **confusion matrix** is to show the classifier's performance in each class (usually known as **positive** and **negative**) separately.

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

The cells in the confusion matrix can represent **number of samples** or **rates** (ratios), which are useful when dealing with imbalanced datasets:

- 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

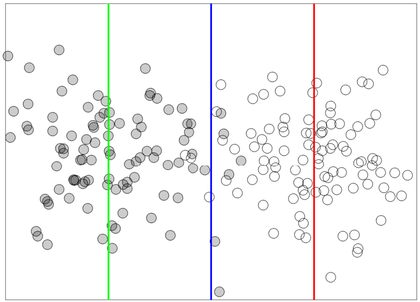
- $TP = 4 \rightarrow TPR = 4/6 \approx 0.66$
- $FP = 1 \rightarrow FPR = 1/4 = 0.25$
- $FN = 2 \rightarrow FNR = 2/6 \approx 0.33$
- $TN = 3 \rightarrow TNR = 3/4 = 0.75$
- $A = (4+3)/10 = 7/10 = 0.7$
- $E = (1+2)/10 = 3/10 = 0.3$

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

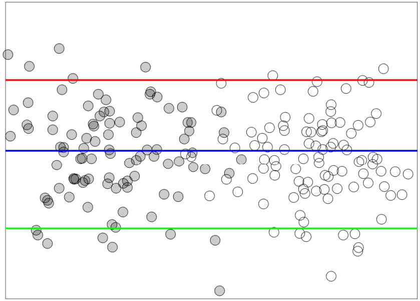


	Actual	
Predicted	1.00	0.50
	0	0.50

	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 curve

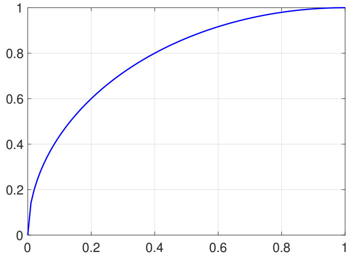
Ideally, we would like our **TPR (sensitivity) to be as close to 1 as possible** and our **FPR (fall-out) to be as close to 0 as possible**.

The ROC (*receiver operating characteristic*) curve shows how the **TPR and the FPR change as we move the boundary of our classifier**, so that we can **calibrate our classifier** and achieve the target performance.

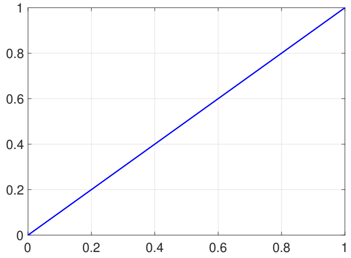
The so-called **area under the curve (AUC) is a measure of goodness for a calibratable classifier** and it gives the average value of sensitivity for all possible values of specificity. Good classifiers will have AUC close to 1, bad classifiers close to 0.5.

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)

Performance metrics

Bayesian methods

Tree methods

Validation approaches

Quick estimation

A certain disease has a prevalence of 1 %, i.e. we expect one individual out of 100 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

Mathematically, 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 for which $P(y_i = C|\boldsymbol{x}_i)$ is largest. This is called the **Bayes classifier**.

Up untill now, we have studied two classifiers, namely **logistic regression** and **kNN**. They both use an estimation of $P(y_i = C|\boldsymbol{x}_i)$:

- Logistic regression, uses the logistic curve.
- kNN, uses the proportions 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 labelled sample i with predictors $\boldsymbol{x}_i$ and label y_i, and a collection of classes C_j, Bayes Theorem can be written (ignoring a few nuisances) 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.

The idea now is

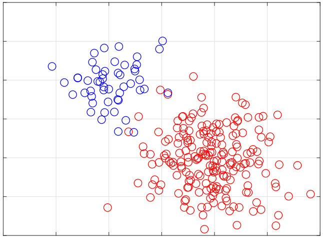
- We want to know $P(y_i = C_j|\boldsymbol{x}_i)$
- If we know the priors $P(\boldsymbol{x}_i|y_i = C_j)$ and $P(C_j)$ we could derive it!
- Do we know anything about those priors? Can we estimate them based on our data?

The Gaussian approximation

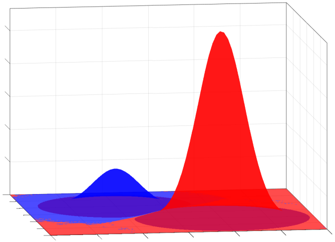
A common assumption is the **Gaussianity** of $P(\boldsymbol{x}_i|y_i = C_j)$:

- We estimate the mean μ_j and standard deviation σ_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** if the classes are well separated
- 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)

Performance metrics

Bayesian methods

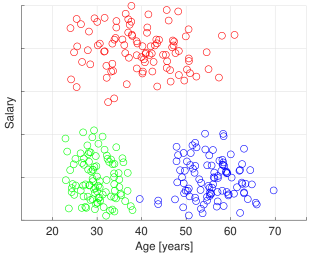
Tree methods

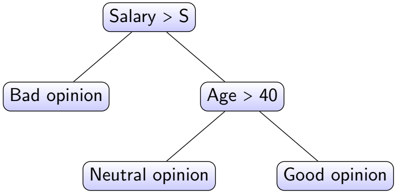
Validation approaches

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*.

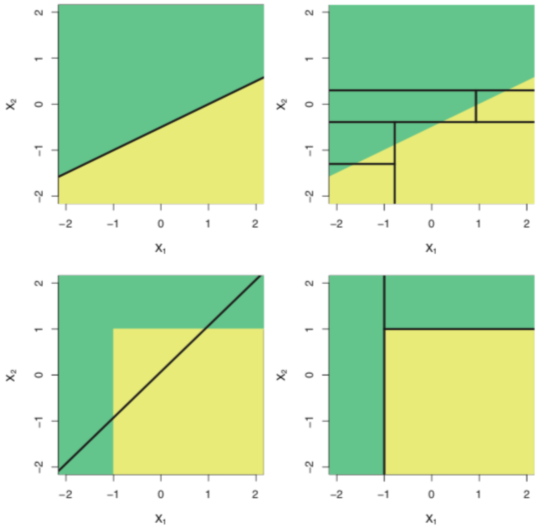
Binary splitting

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.

More 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)

Performance metrics

Bayesian methods

Tree methods

Validation approaches

Why do we need validation?

We already know that in general our **training error** will be different (and smaller!) than the error of our model when it is in production. By using a **test dataset** we can estimate a predictive performance of our model, but this dataset might not be always available.

Furthermore, we might want to **consider different types 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 allow us to **estimate of the test error by using our available dataset**.

Validation set approach

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

Our models are then fitted with the training part and the validation part is used to provide estimates of the test error rate (it is not the actual *test error rate*, as this is obtained with the test dataset, not available to us).

- The estimate of the test error **can be highly variable**, depending on the observations that are included in the validation dataset
- Since it generally uses fewer samples than the ones in the test dataset, it tends to **overestimate the test error**

Leave-one-out cross-validation (LOOCV)

This method also splits the available dataset into two datasets, namely a training and validation set. However, the **validation set contains only one samples** and **multiple splits are considered**.

If our dataset has N samples, we can produce N splits and estimate N different test errors. Based on them, the final error estimate is simply the average.

Compared to the validation set approach:

- It has less bias and doesn't tend to overestimate the error
- It always yields the same estimated error, as no random splitting is involved
- It can be expensive to implement, as it requires fitting a model N times

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 then carry k **rounds of training followed by validation**, each one of them 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.

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