Machine Learning: Tasks

Roberto Esposito
roberto.esposito@unito.it

These slides are based on Peter Flach's “Machine Learning” text book and on the slides made available with it

Predictive machine learning scenarios

- A predictive machine learning is one where given a dataset of *labelled* data, a learning algorithm is asked to build a *model* capable of making predictions about "some property" of new (previously unseen) examples.

- Depending on the kind of labels attached to the data and of which "*property*" the algorithm is asked to predict, we can distinguish between the following tasks: *classification, scoring and ranking, probability estimation, and regression.*

Classification

A classifier is a mapping:

$$\hat{c} : \mathscr{X} \rightarrow \mathscr{C}$$

where:

$$\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$$

the "hat" over the name of the classifier denotes that the classifier is an approximation of the true but unknown function c.

An example is a pair:

$$(x, c(x)) \in \mathscr{X} \times \mathscr{C}$$

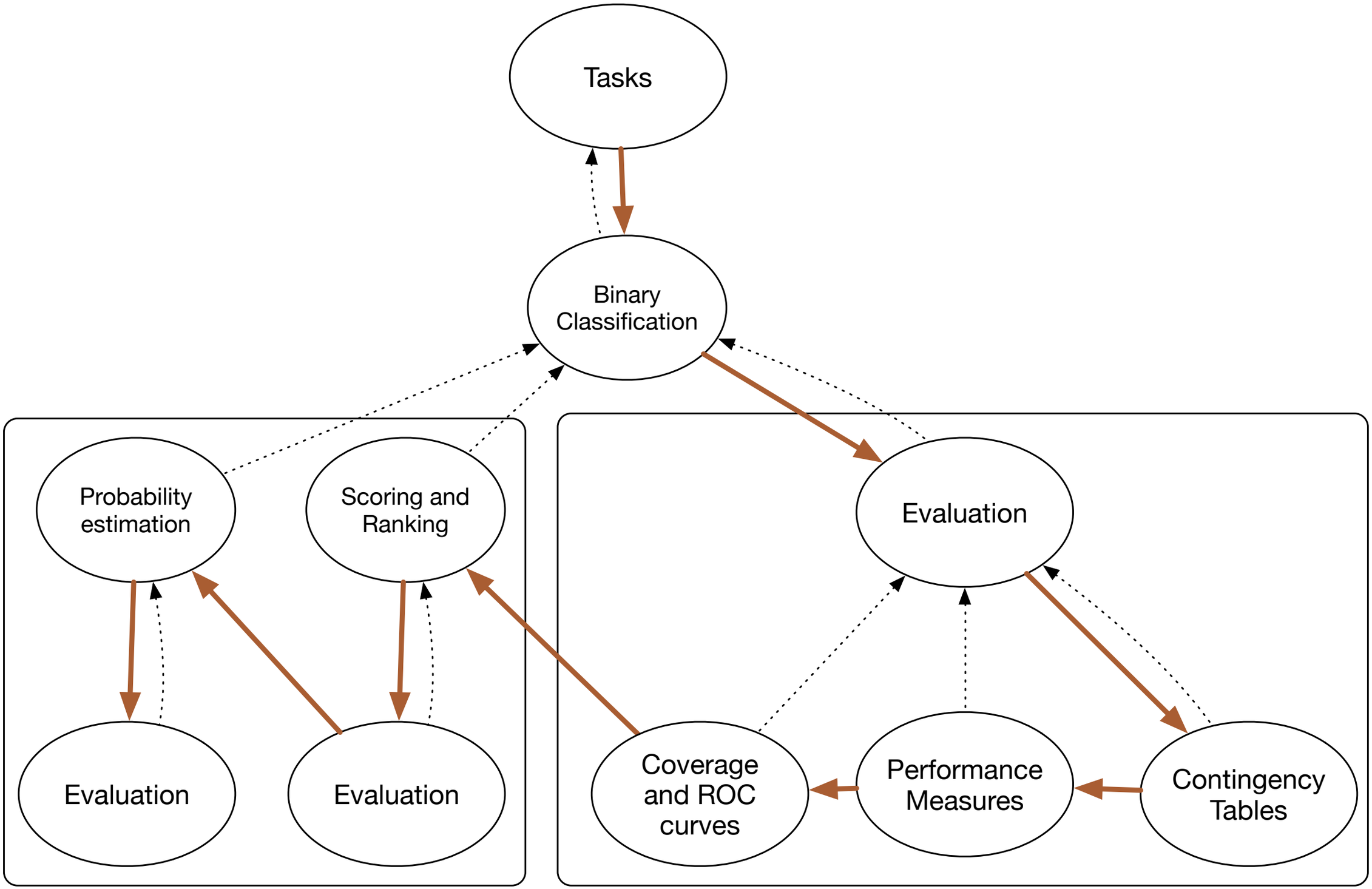
where x is an "instance" and $c(x)$ is the *true* class of the instance (possibly contaminated by noise).

Classification

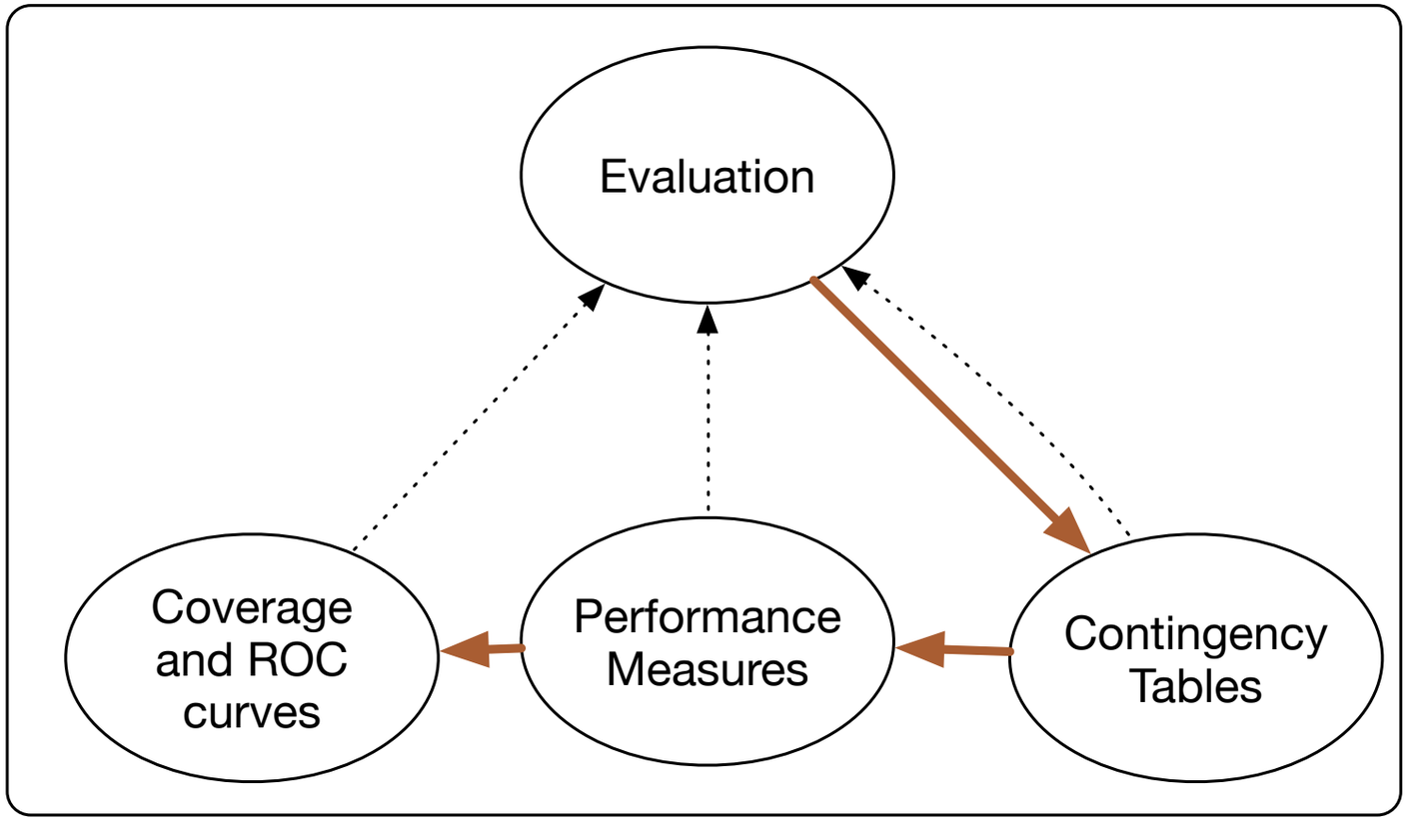
Learning a classifier involves constructing the function $\hat{c}$ such that it matches c as closely as possible (and not just on the training set, but ideally on the entire instance space $\mathscr{X}$).

Binary classification

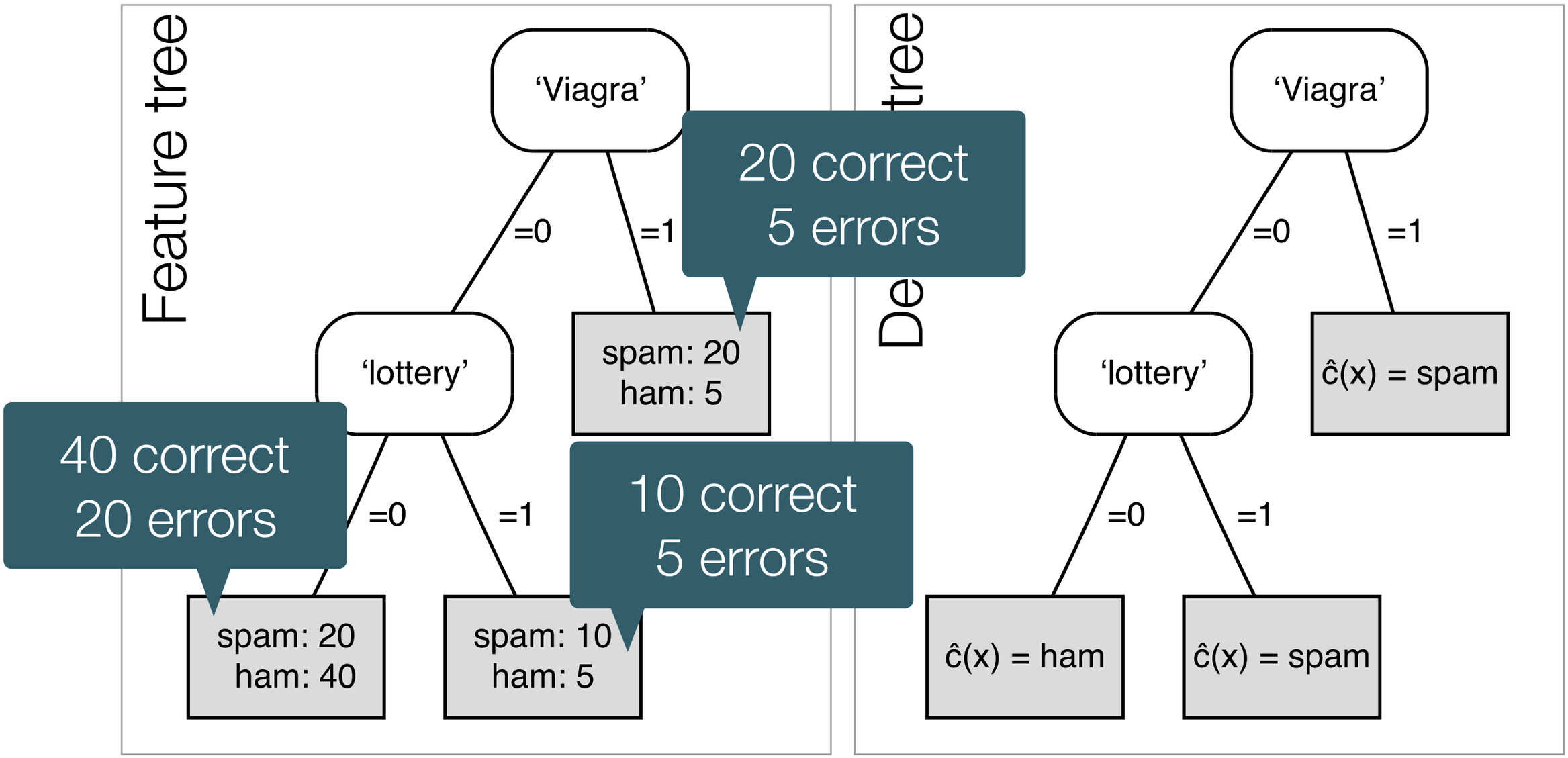
Binary classification topics



Evaluation

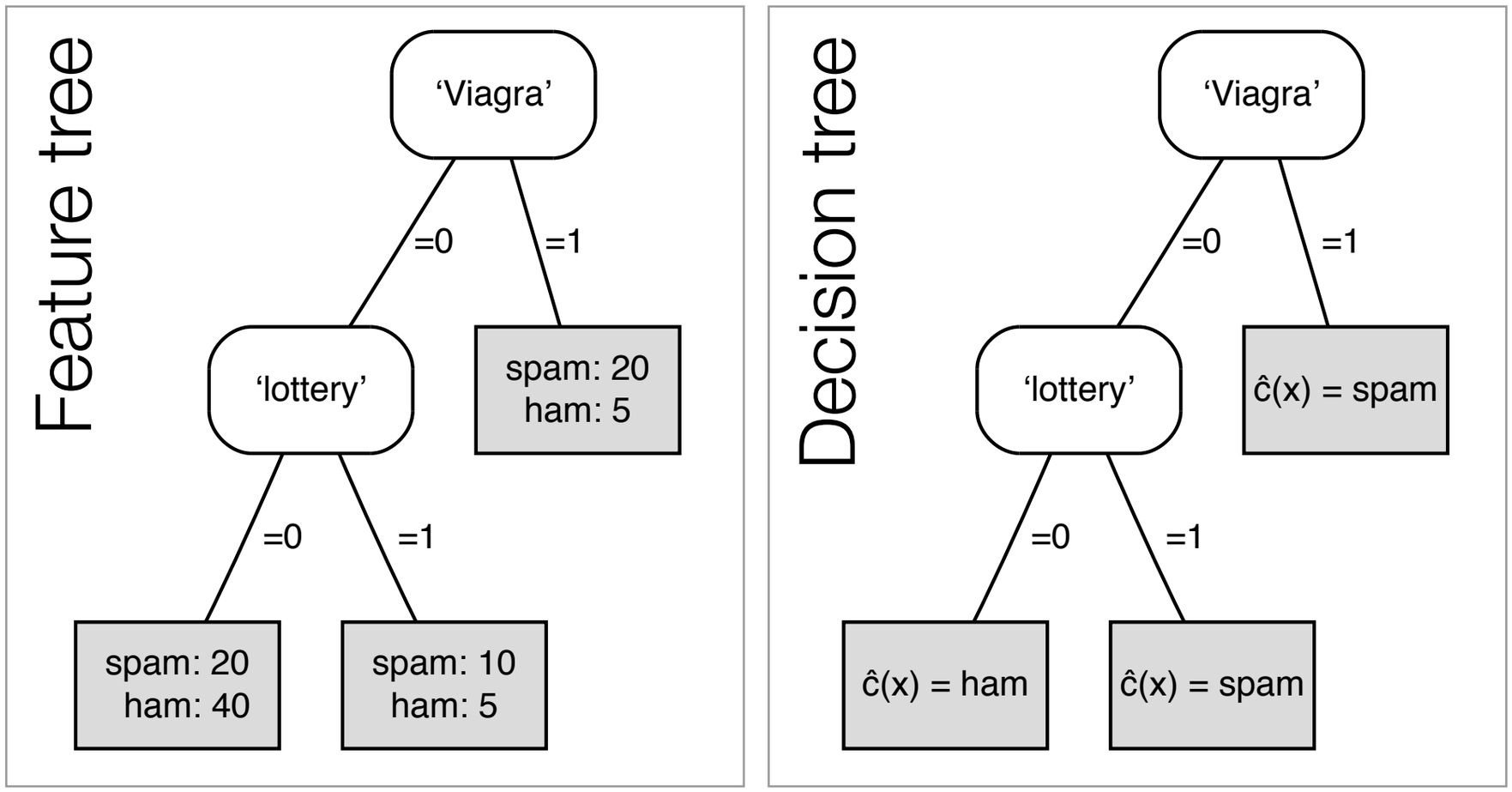

Evaluation
Coverage and ROC curves
Performance Measures
Contingency Tables

A decision tree



accuracy = (40 + 10 + 20)/100 = 70%

Contingency table



	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

Contingency table

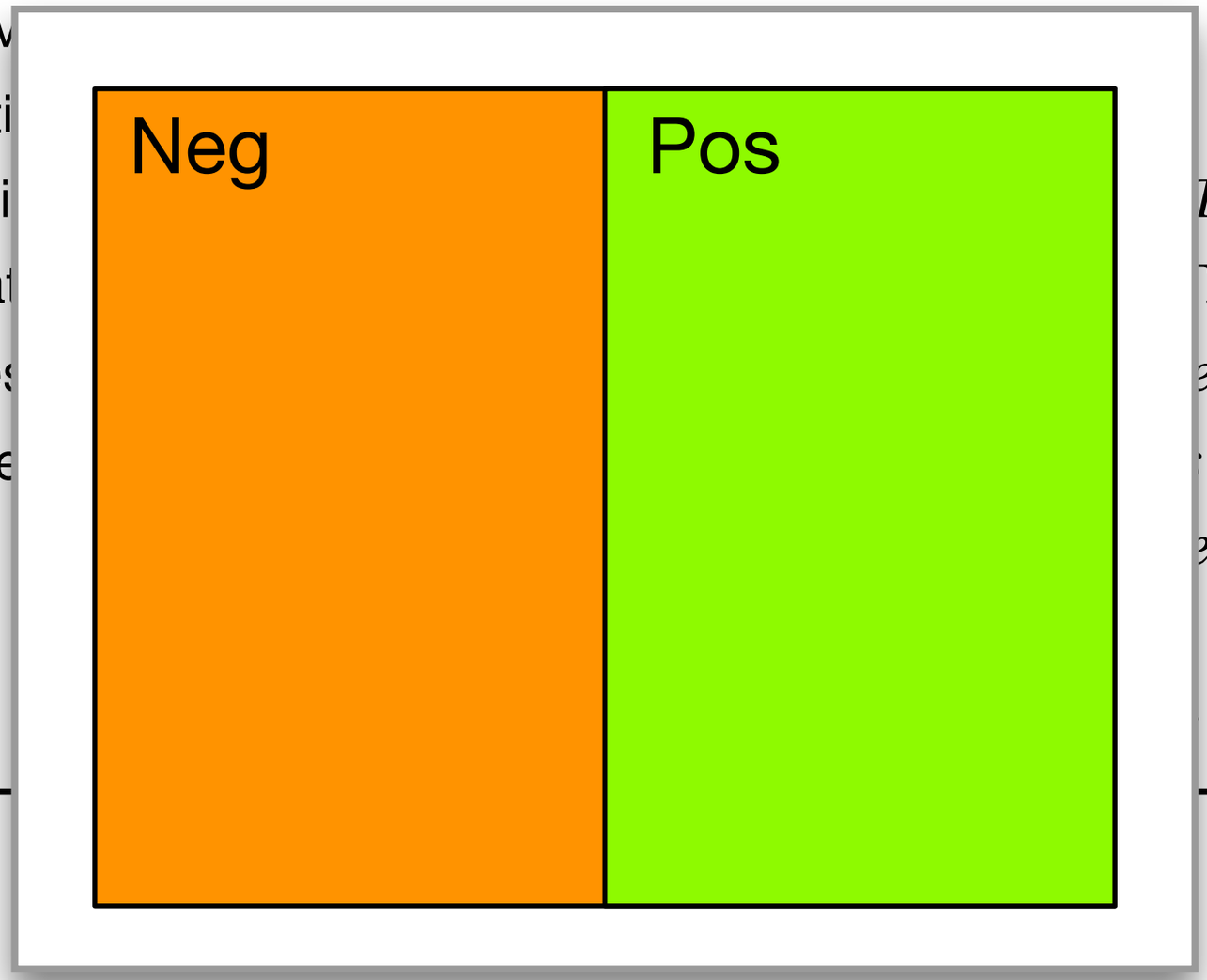
	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	TP	FN	Pos
Actual ⊖	FP	TN	Neg
	40	60	100

Performance measures

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos / \lvert Te \rvert$	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos / neg$	Pos / Neg	
(*) accuracy	$acc = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

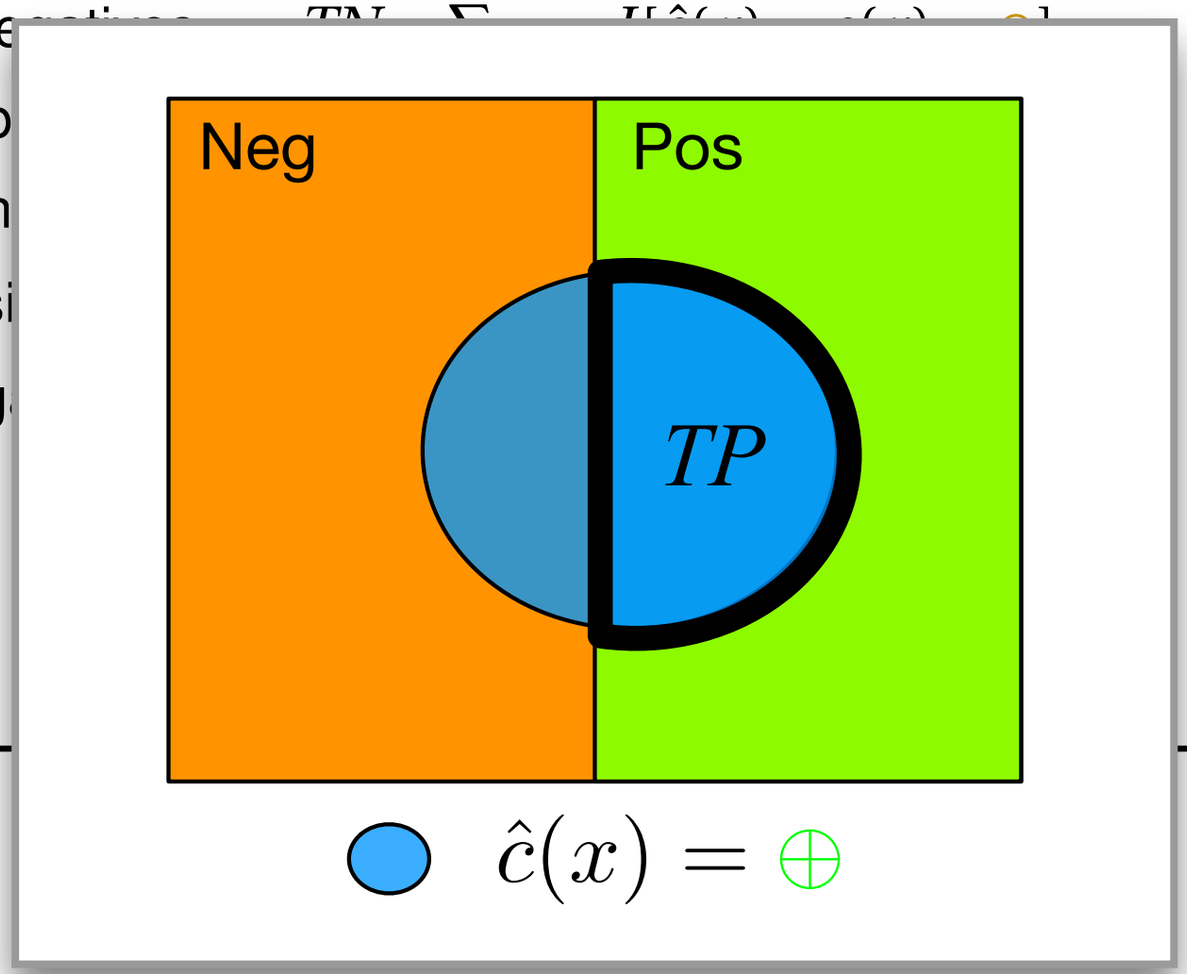
Performance measures

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positiv			
number of true negati			
number of false positi		TN	
number of false negat		TP	
proportion of positives		$e\rvert$	$P(c(x) = \oplus)$
proportion of negative			$P(c(x) = \ominus)$
class ratio		eg	
(*) accuracy			$P(\hat{c}(x) = c(x))$
(*) error rate			$P(\hat{c}(x) \neq c(x))$



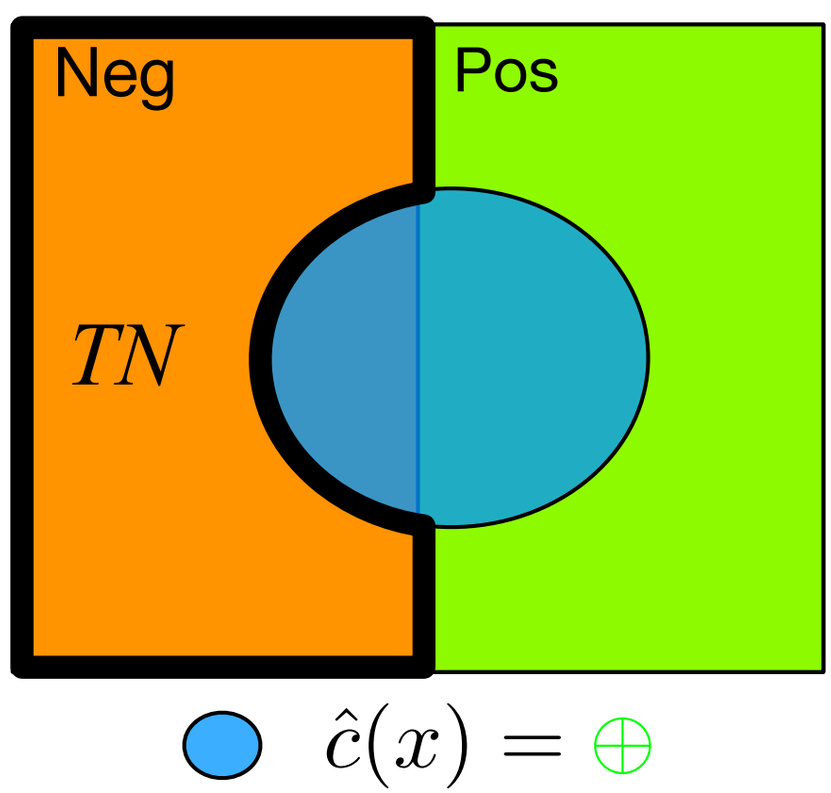
Performance measures

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true ne	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false p		$Neg - TN$	
number of false n		$Pos - TP$	
proportion of posi		$Pos / \lvert Te \rvert$	$P(c(x) = \oplus)$
proportion of neg		$1 - pos$	$P(c(x) = \ominus)$
class ratio		Pos / Neg	
(*) accuracy			$P(\hat{c}(x) = c(x))$
(*) error rate		$1 - acc$	$P(\hat{c}(x) \neq c(x))$

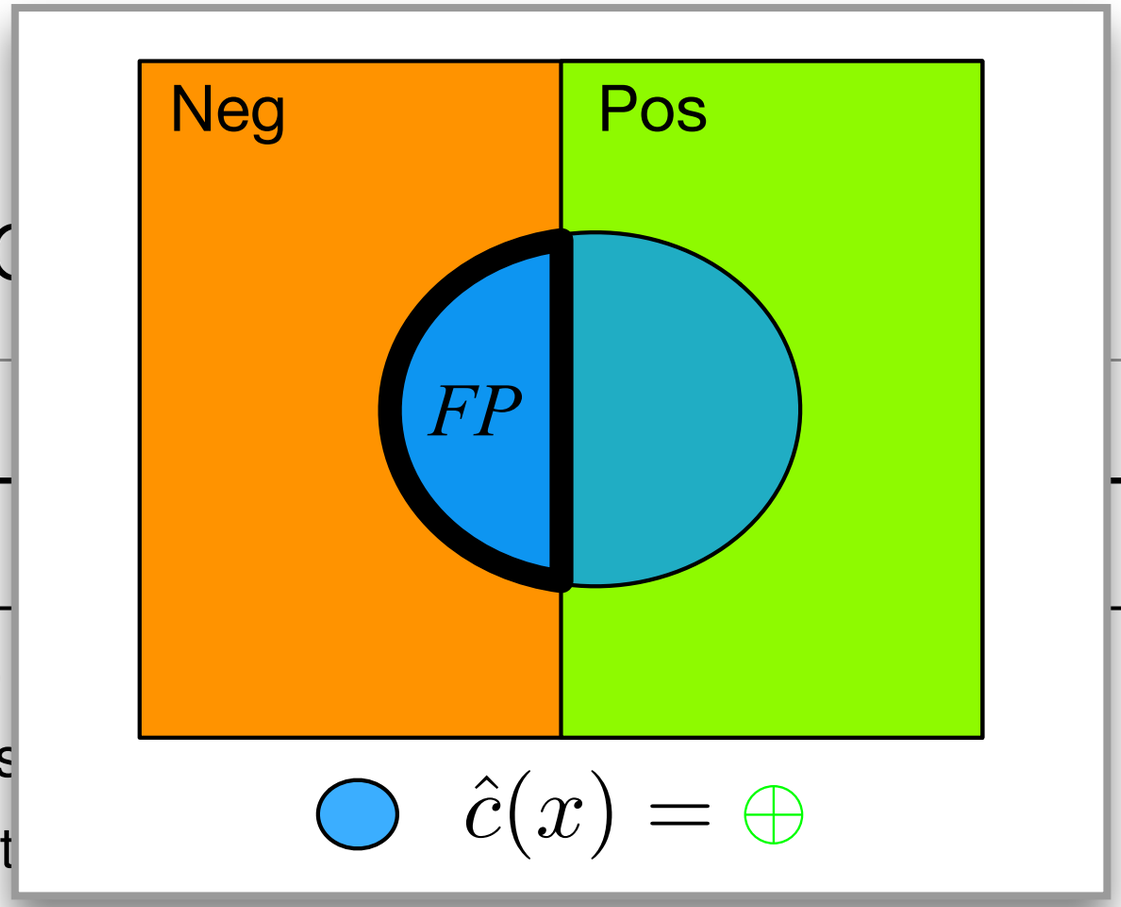


Performance measures

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives		$- TN$	
number of false negative		$- TP$	
proportion of positives		$/ \lvert Te \rvert$	$P(c(x) = \oplus)$
proportion of negatives		pos	$P(c(x) = \ominus)$
class ratio		$/ Neg$	
(*) accuracy			$P(\hat{c}(x) = c(x))$
(*) error rate		acc	$P(\hat{c}(x) \neq c(x))$

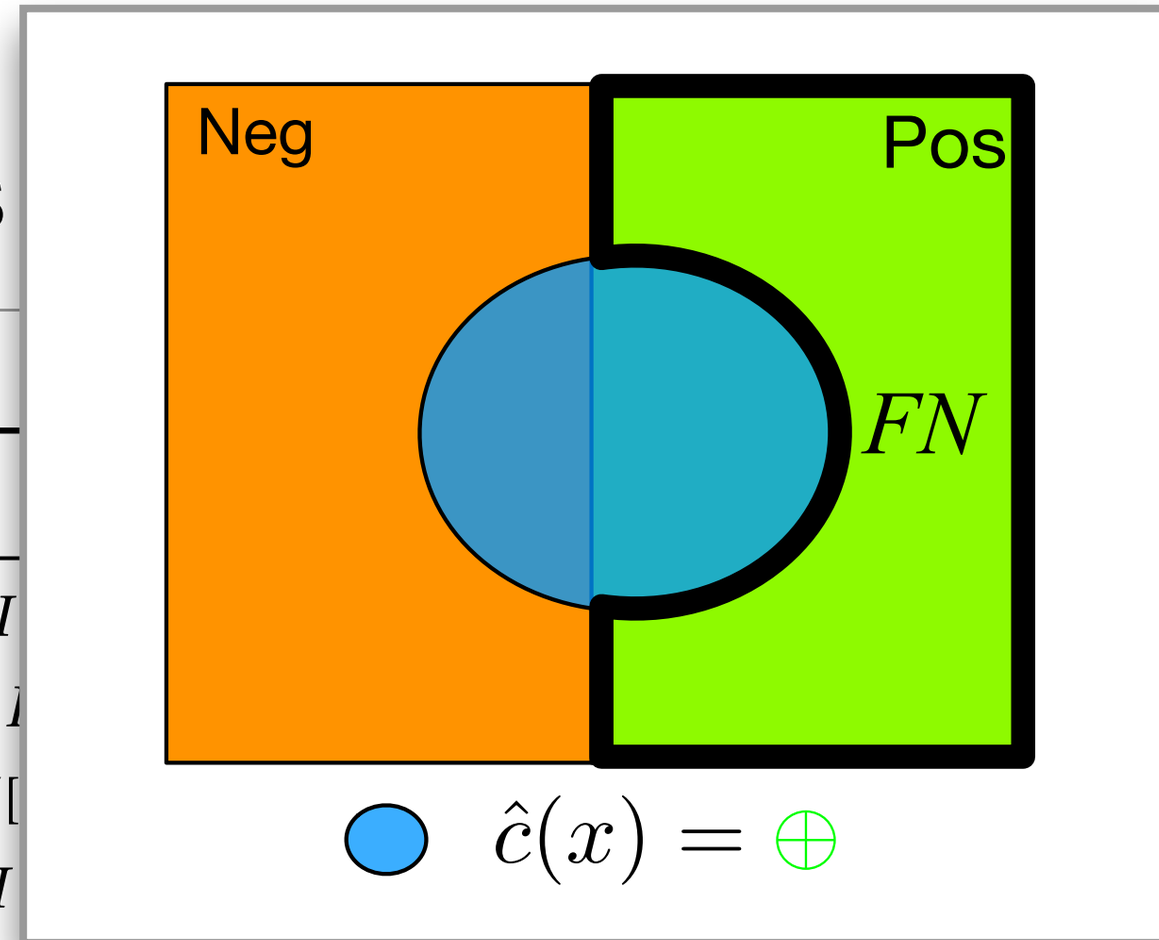


Performanc



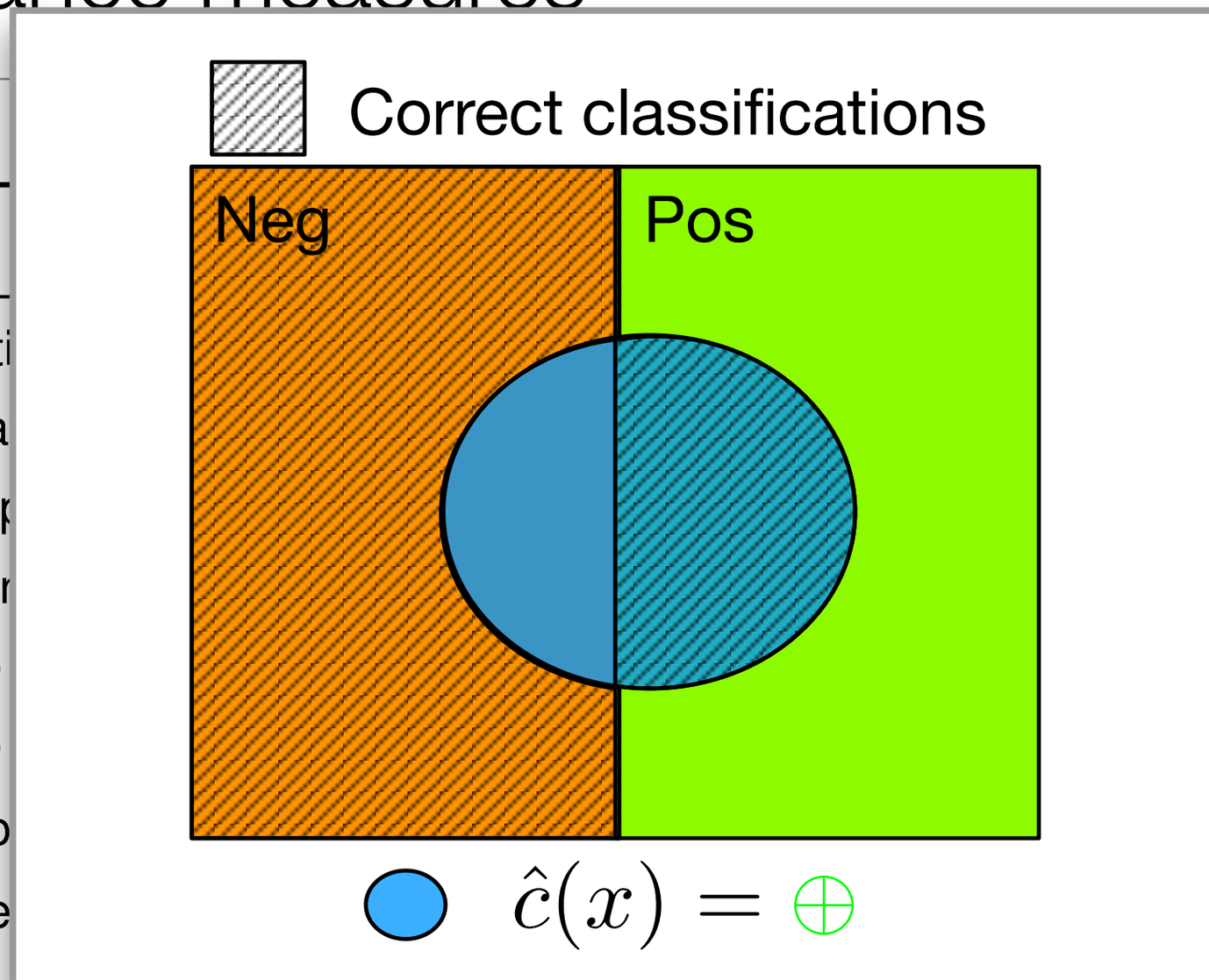
Measure		*Equal to*	*Estimates*
number of positives			
number of negatives		$\|Te\| - Pos$	
number of true posit			
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{\|Te\|} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos/\|Te\|$	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{\|Te\|} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos/neg$	Pos/Neg	
(*) accuracy	$acc = \frac{1}{\|Te\|} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\|Te\|} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

Performance measures



Measure	*Definition*			*s*
number of positives	$Pos = \sum_{x \in Te} I$			
number of negatives	$Neg = \sum_{x \in Te} I$			
number of true positives	$TP = \sum_{x \in Te} I[$			
number of true negatives	$TN = \sum_{x \in Te} I$			
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$		
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$		
proportion of positives	$pos = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos / \lvert Te \rvert$	$P(c(x) = \oplus)$	
proportion of negatives	$neg = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$	
class ratio	$clr = pos / neg$	Pos / Neg		
(*) accuracy	$acc = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$	
(*) error rate	$err = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$	

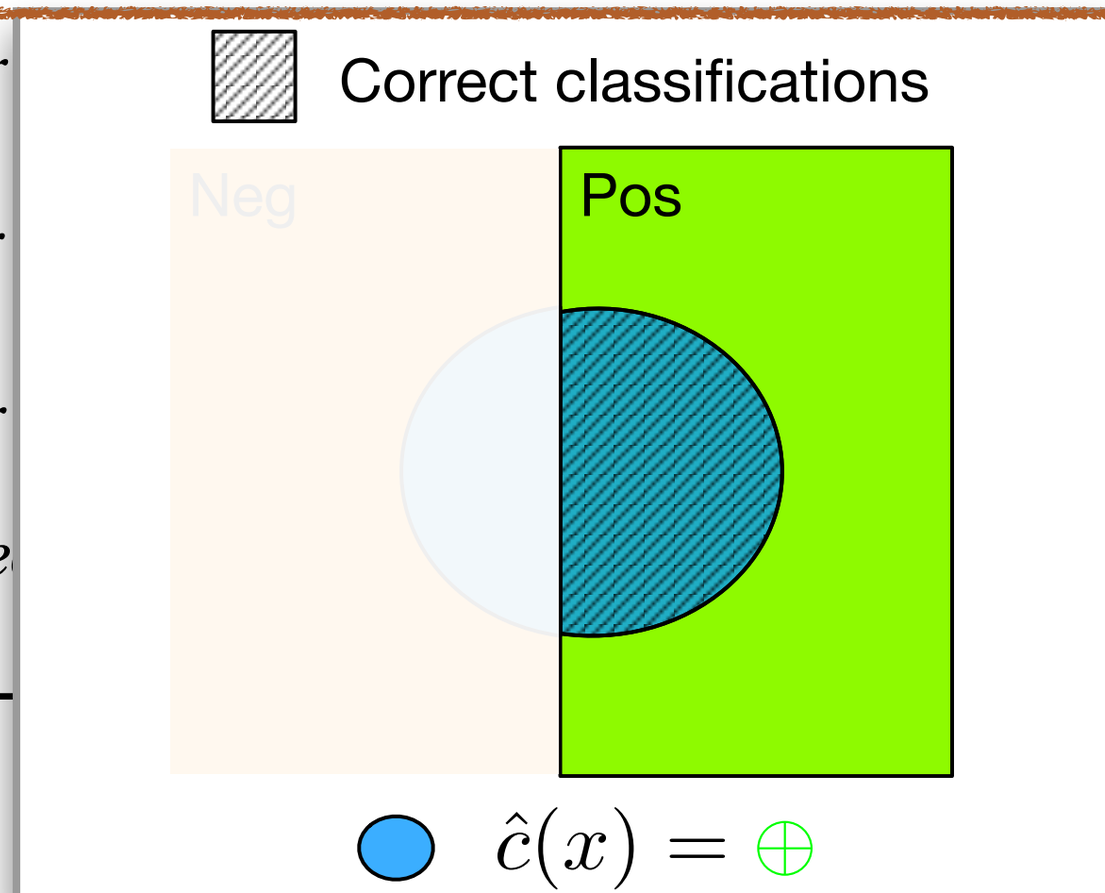
Performance measures



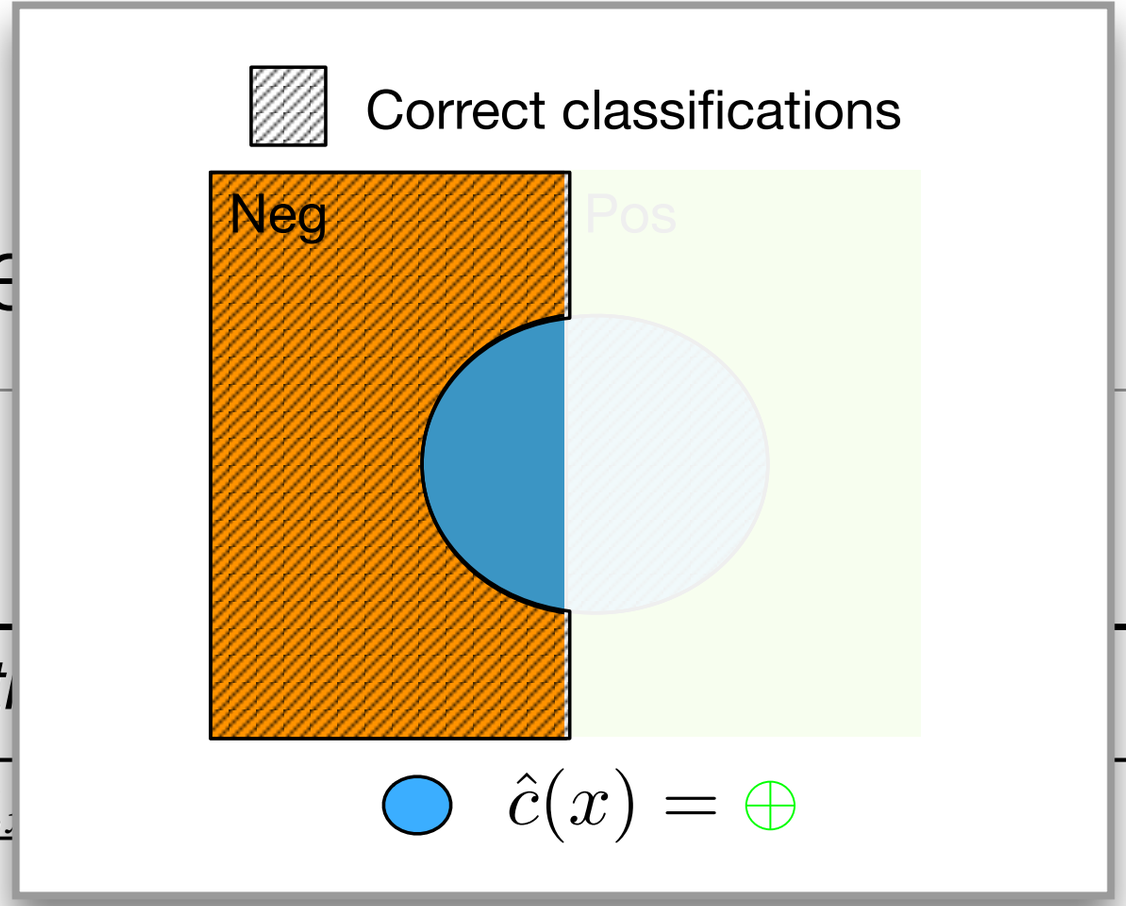
Measure		*l to*	*Estimates*
number of positi			
number of nega		*Pos*	
number of true p			
number of true r			
number of false		*TN*	
number of false		*TP*	
proportion of po		*Te*|	$P(c(x) = \oplus)$
proportion of ne		*s*	$P(c(x) = \ominus)$
class ratio	$cr = pos/neg$	*Pos/Neg*	
(*) accuracy	$acc = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

Performance measures

Measure	*Definition*	*Equal to*	*Estimates*
true positive rate, sensitivity, recall	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\oplus]}{\sum_{x \in Te} I[c(x)=\oplus]}$	TP/Pos	$P(\hat{c}(x) = \oplus \mid c(x) = \oplus)$
true negative rate, specificity	tnr		$P(\hat{c}(x) = \ominus \mid c(x) = \ominus)$
false positive rate, false alarm rate	fpr	nr	$P(\hat{c}(x) = \oplus \mid c(x) = \ominus)$
false negative rate	fnr	pr	$P(\hat{c}(x) = \ominus \mid c(x) = \oplus)$
precision, confidence	pre		$P(c(x) = \oplus \mid \hat{c}(x) = \oplus)$

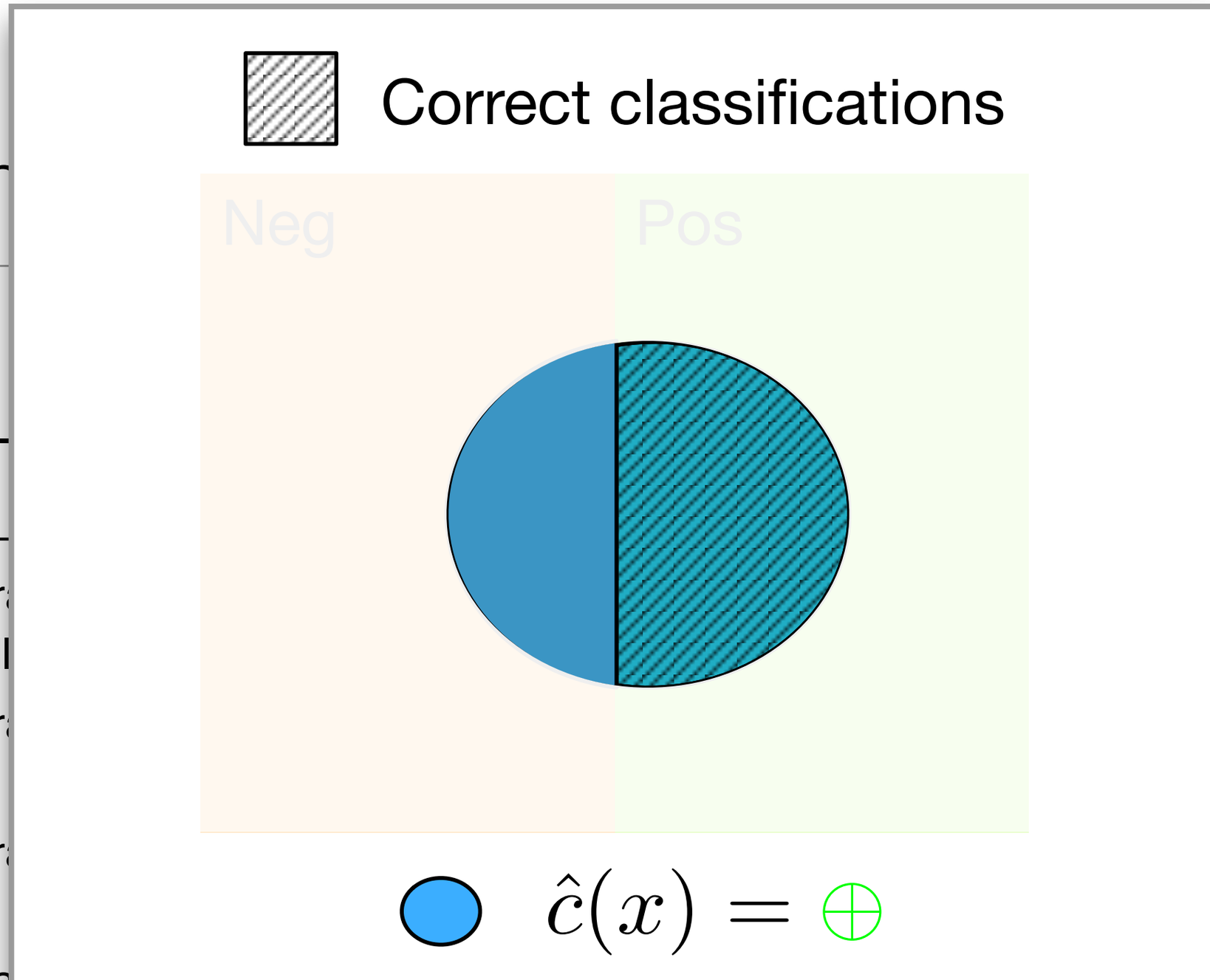


Performance me



Measure	*Definiti*		*Estimates*
true positive rate, sensitivity, recall	$tpr = \sum$		$P(\hat{c}(x) = \oplus \mid c(x) = \oplus)$
true negative rate, specificity	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	TN/Neg	$P(\hat{c}(x) = \ominus \mid c(x) = \ominus)$
false positive rate, false alarm rate	$fpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	$FP/Neg = 1 - tnr$	$P(\hat{c}(x) = \oplus \mid c(x) = \ominus)$
false negative rate	$fnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	$FN/Pos = 1 - tpr$	$P(\hat{c}(x) = \ominus \mid c(x) = \oplus)$
precision, confidence	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[\hat{c}(x) = \oplus]}$	$TP/(TP + FP)$	$P(c(x) = \oplus \mid \hat{c}(x) = \oplus)$

Perform



Measure				*mates*
true positive r sensitivity, recall				$x) = \oplus \mid c(x) = \oplus)$
true negative r specificity				$x) = \ominus \mid c(x) = \ominus)$
false positive r false alarm rate				$x) = \oplus \mid c(x) = \ominus)$
false negative ra		$\sum_{x \in Te} I[c(x)=\oplus]$		$x) = \ominus \mid c(x) = \oplus)$
precision, confi-dence	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\oplus]}{\sum_{x \in Te} I[\hat{c}(x)=\oplus]}$	$TP/(TP+FP)$		$P(c(x) = \oplus \mid \hat{c}(x) = \oplus)$

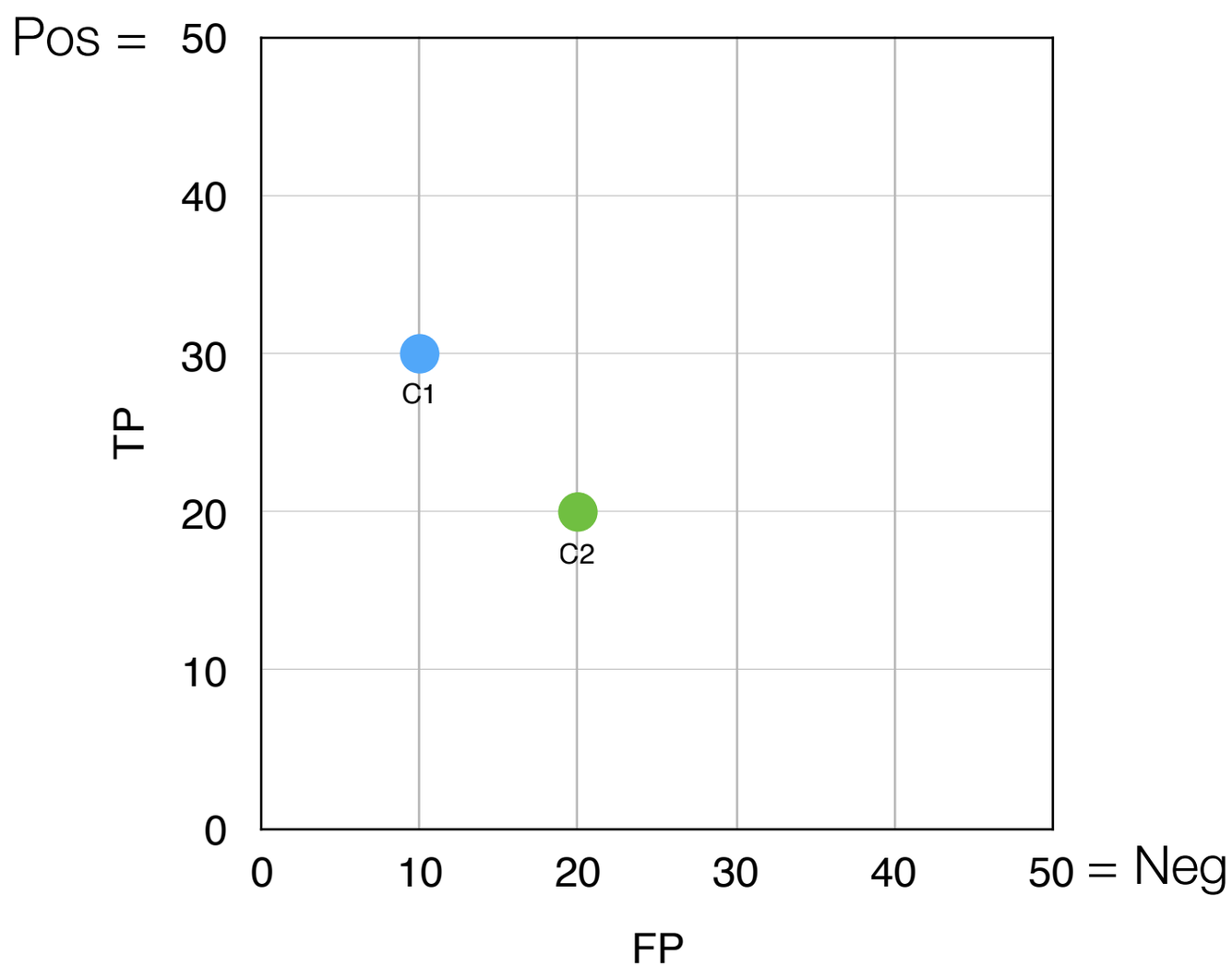
Contingency table

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

Coverage Plot

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

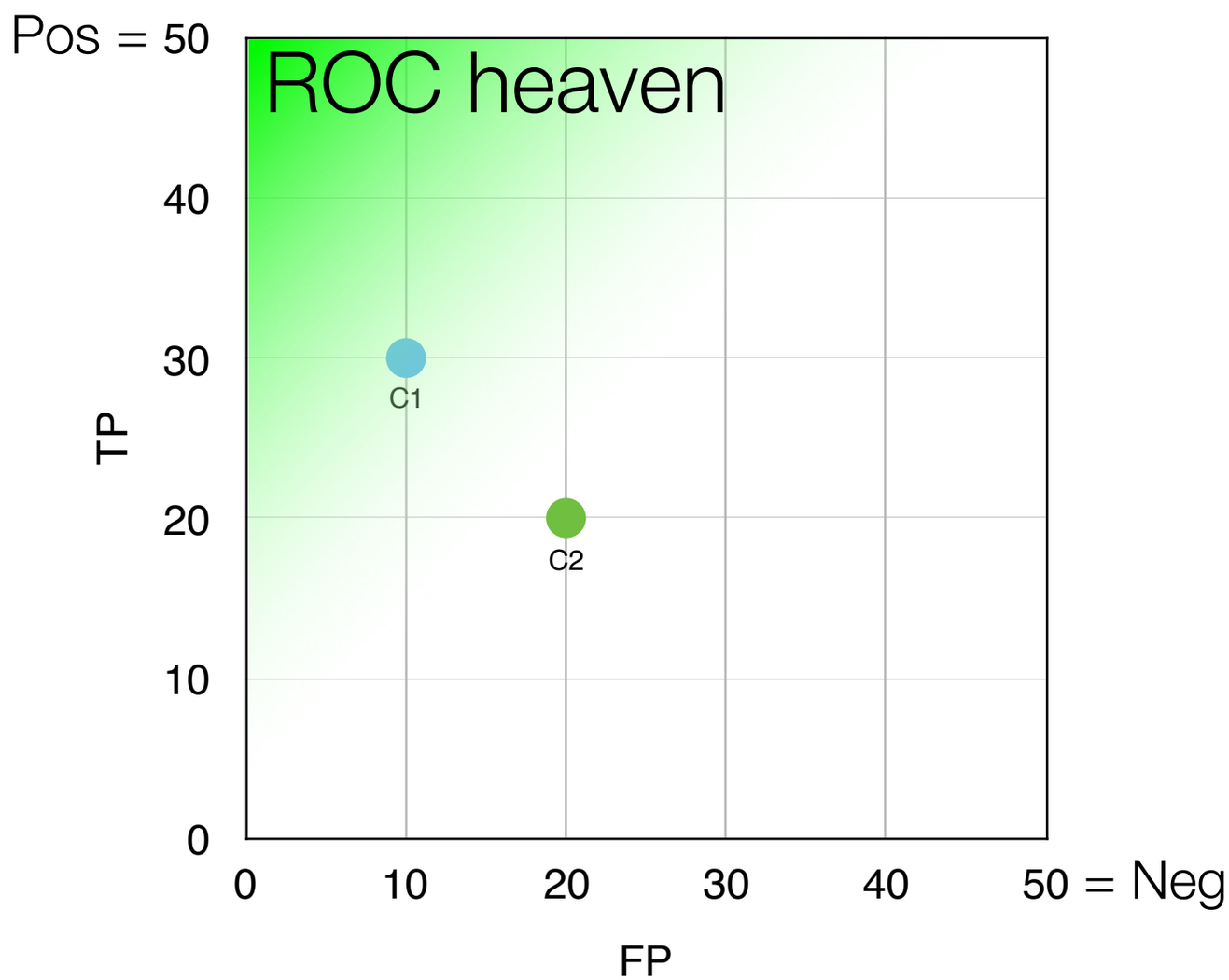
	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**20**	**30**	50
Actual ⊖	**20**	**30**	50
	40	60	100



Coverage Plot

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

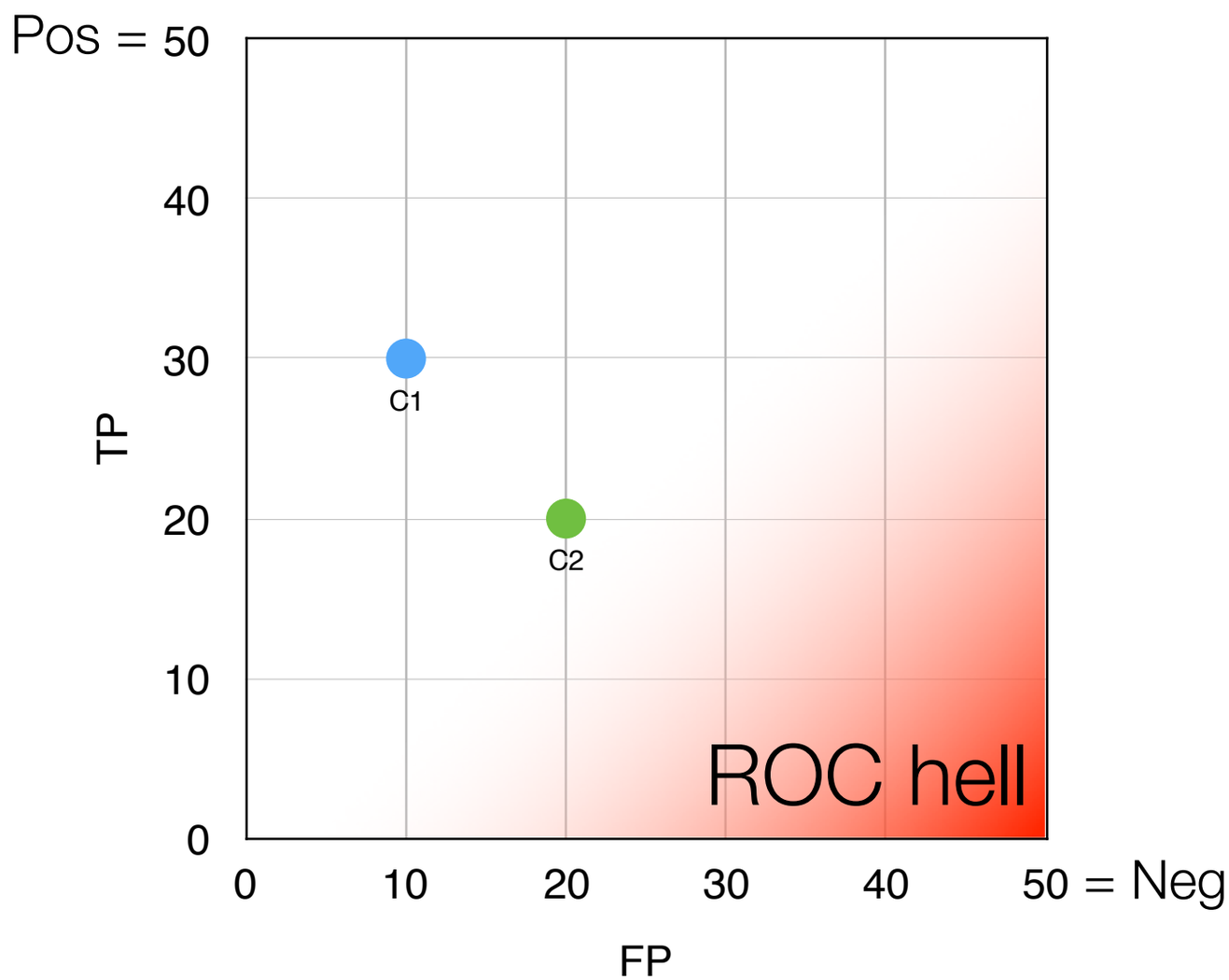
	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**20**	**30**	50
Actual ⊖	**20**	**30**	50
	40	60	100



Coverage Plot

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

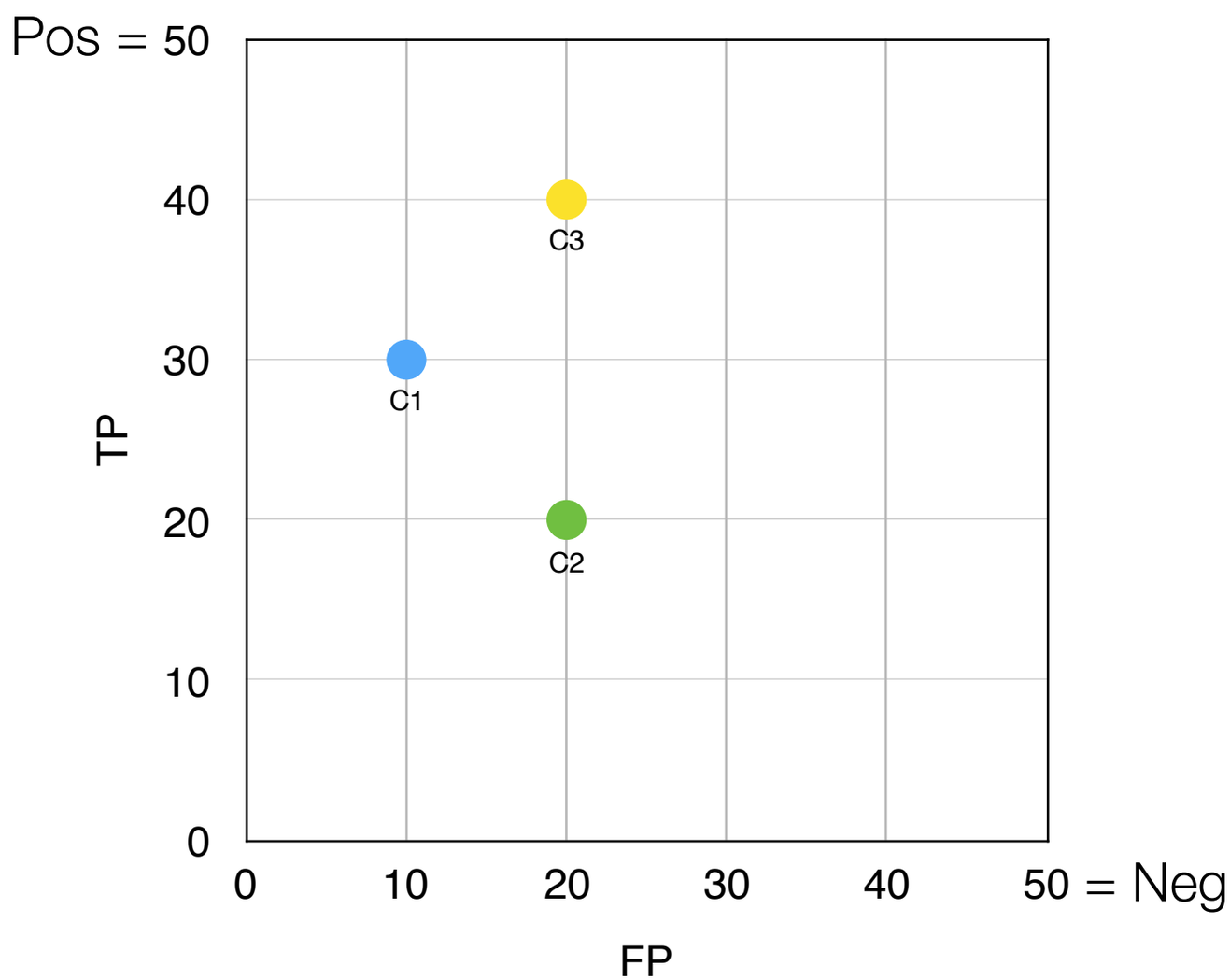
	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**20**	**30**	50
Actual ⊖	**20**	**30**	50
	40	60	100



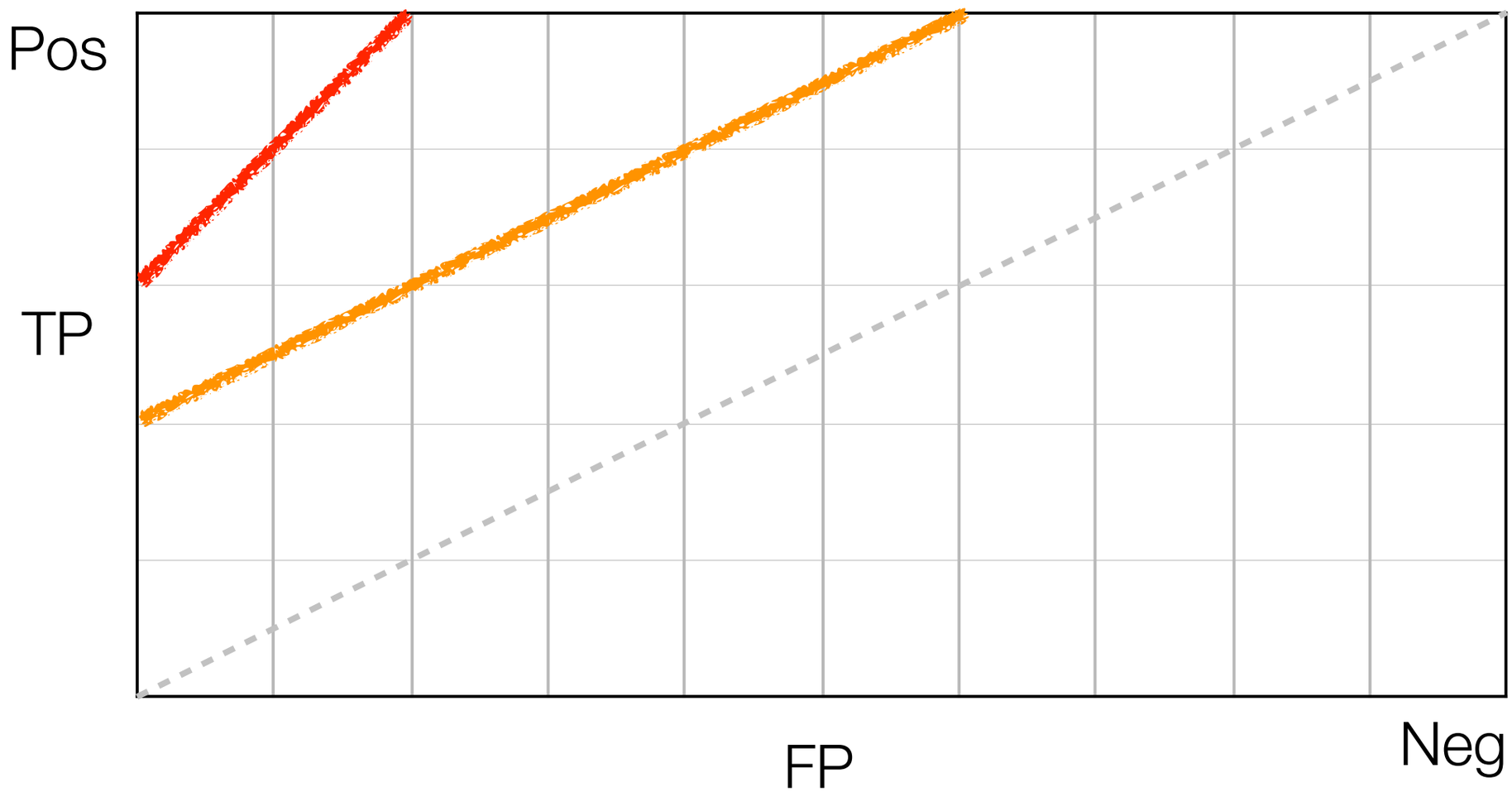
Coverage Plot

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**20**	**30**	50
Actual ⊖	**20**	**30**	50
	40	60	100



Coverage plots properties

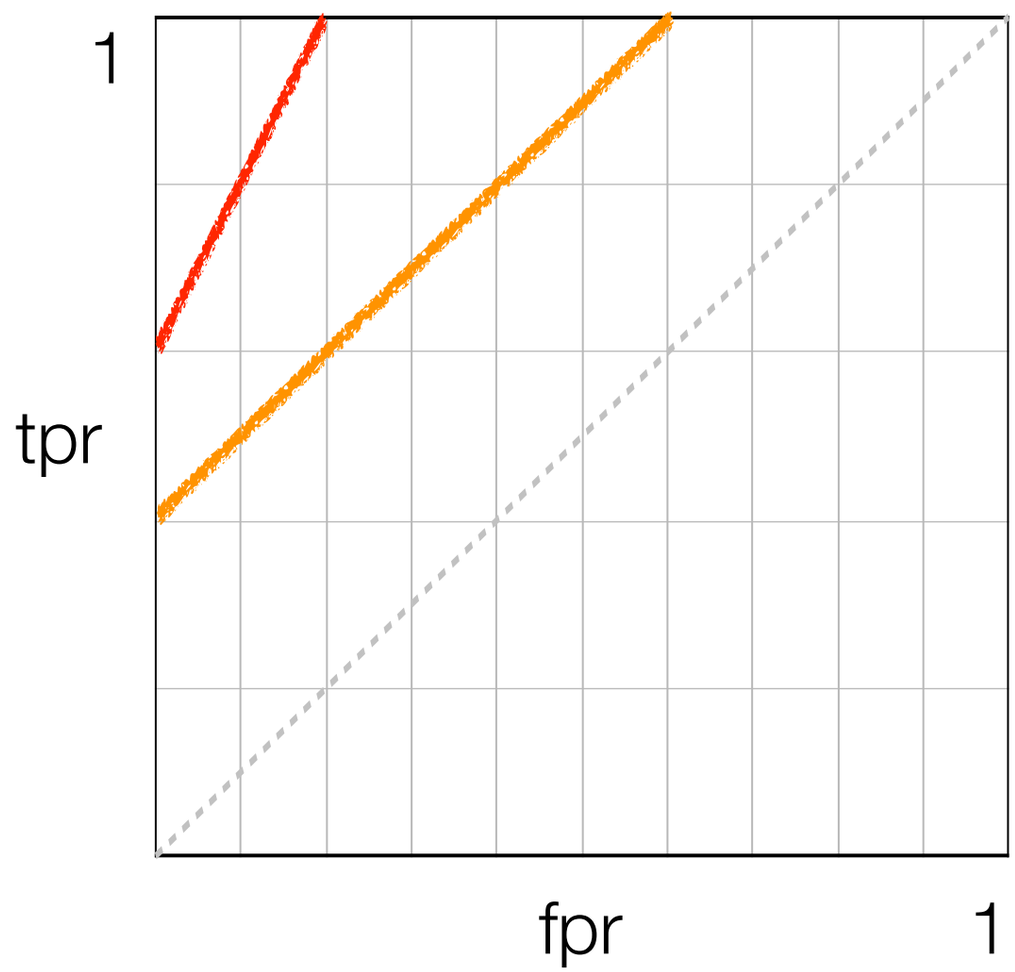


Classifiers with the same **accuracy** are on lines with slope 1

Classifiers with the same **avg recall** are on lines parallel to the main diagonal

avg recall = (recall + specificity)/2 = (pos-rec + neg-rec)/2 = (TP/POS + TN/NEG)/2

Roc plots properties



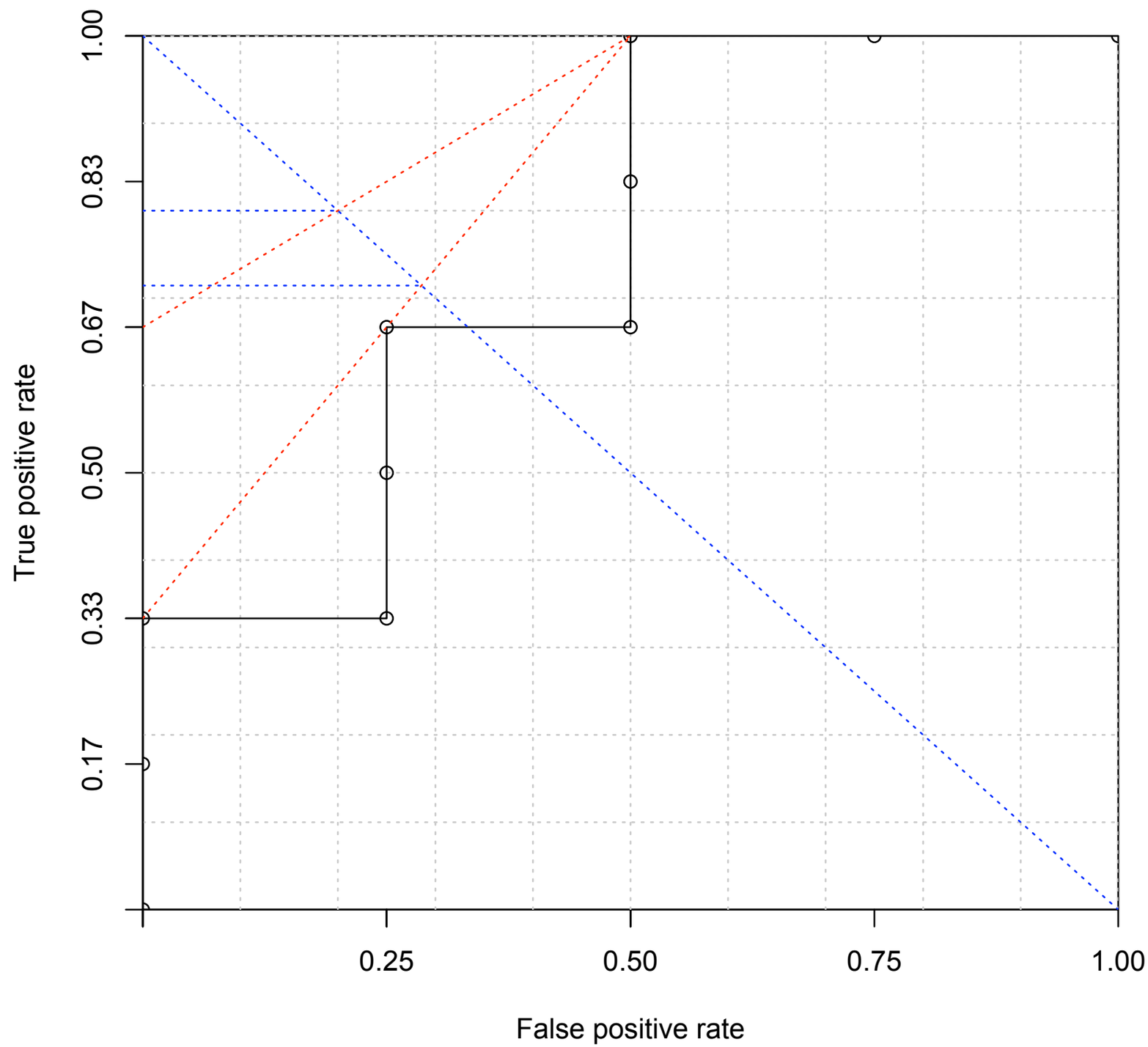
Classifiers with the same **accuracy** are on lines with slope Neg/Pos

Classifiers with the same **avg recall** are on lines parallel to the main diagonal (i.e., with slope 1)

Finding the optimal classifier

Each point in this ROC plot corresponds to a different threshold and, thus a different classifier.

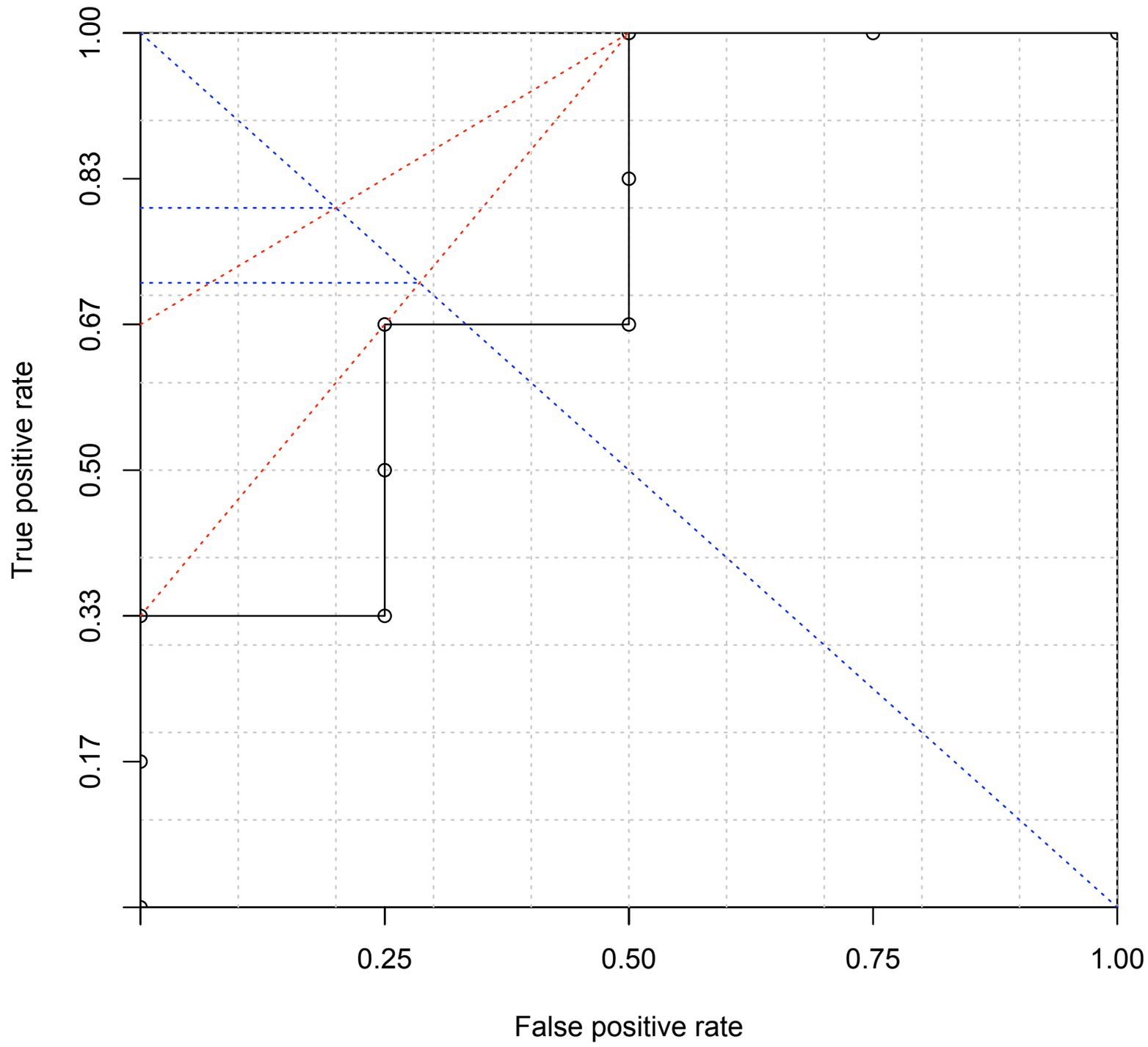
Which one is best?



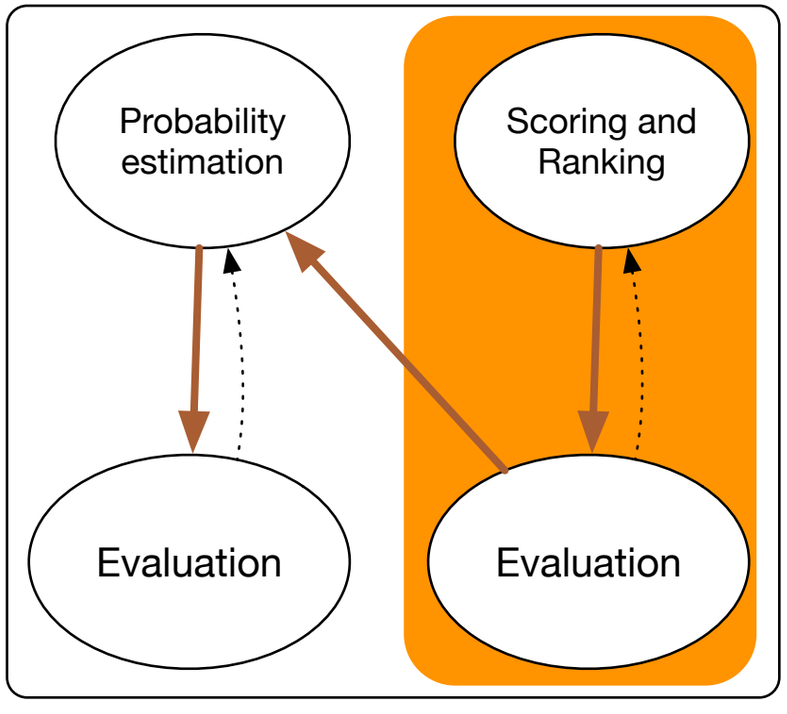
Finding the optimal point

Isometrics can be set to give more importance to one class even if the class ratio would suggest otherwise.

If I have different costs for FP and for FN, I can use isometrics of slope 1/c, where c = cost(FN)/cost(FP).



Scoring and ranking



Scoring classifier

A scoring classifier is a mapping:

$$\hat{\mathbf{s}} : \mathscr{X} \to \mathbb{R}^k$$

the boldface notation denotes that the output is a vector, i.e.:

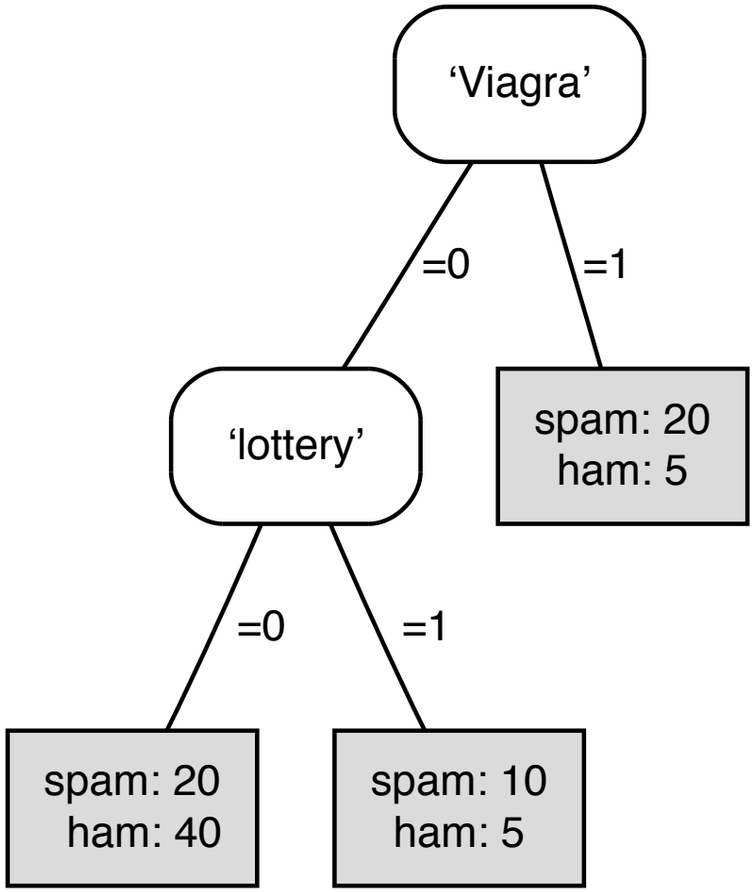
$$\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$$

where the *i*-th component is the score assigned to class C_i for instance x.

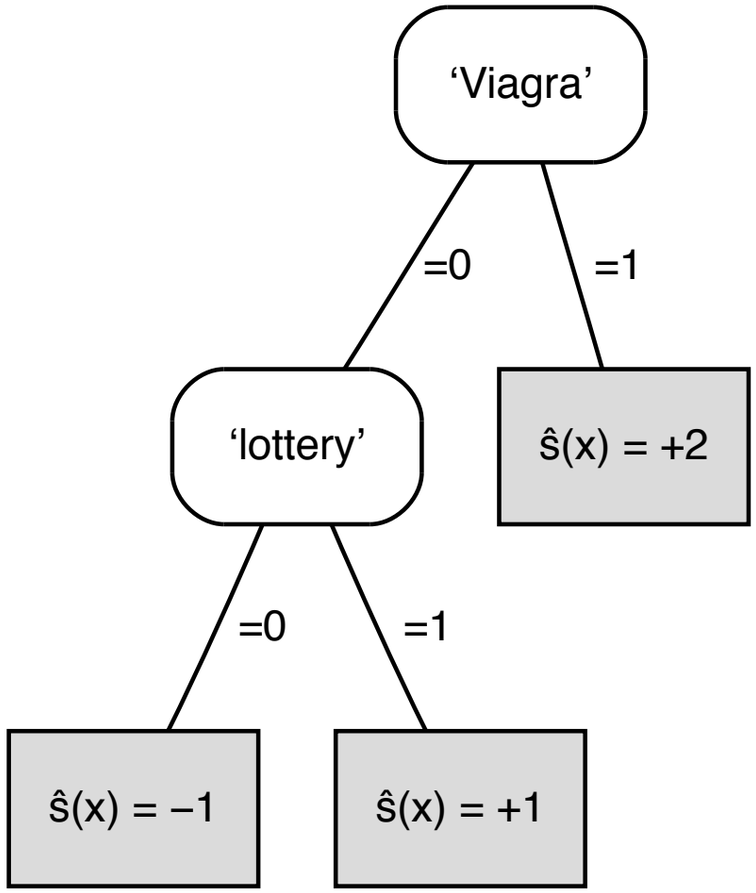
If we only have two classes, it usually suffices to consider the score for only one of the classes.

Important: here scores are always to be interpreted in the context of a classifier, they are not learned from “true” scores; rather, they are a measure of the confidence the classifier has in its prediction.

A scoring tree



A **feature tree** with training set class distribution on the leaves



A **scoring tree** using the logarithm of the class ratio as scores

Margin

If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity:

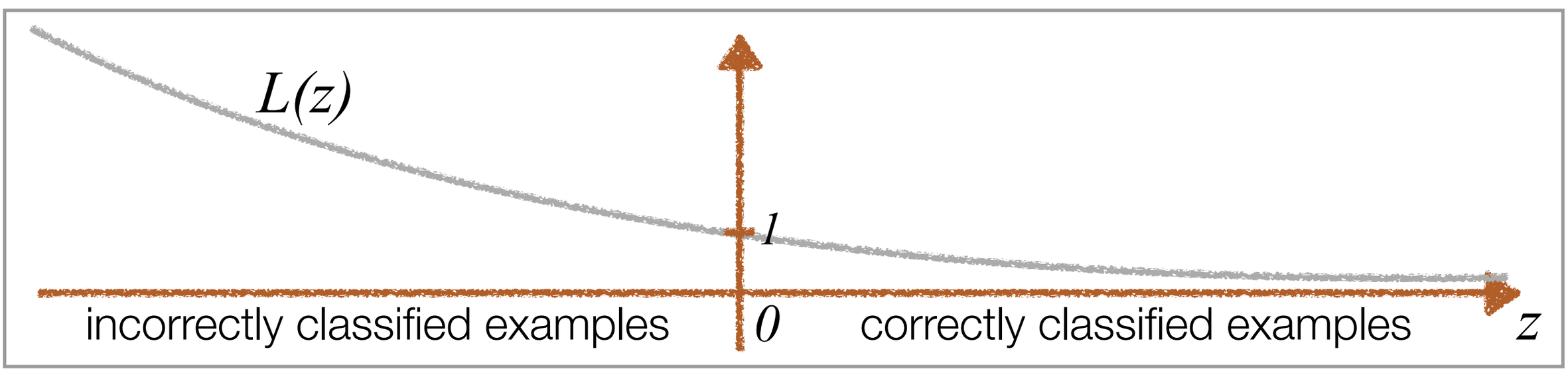
$$z(x) = c(x)\hat{s}(x) = \begin{cases} +|\hat{s}(x)| & \text{if } \hat{s} \text{ is correct on } x \\ -|\hat{s}(x)| & \text{otherwise} \end{cases}$$

is called the **margin** assigned by the scoring classifier to the example

Loss functions

We would like to reward large positive margins, and penalise large negative ones. This is achieved by means of a so-called **loss function** $L : \mathbb{R} \to [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

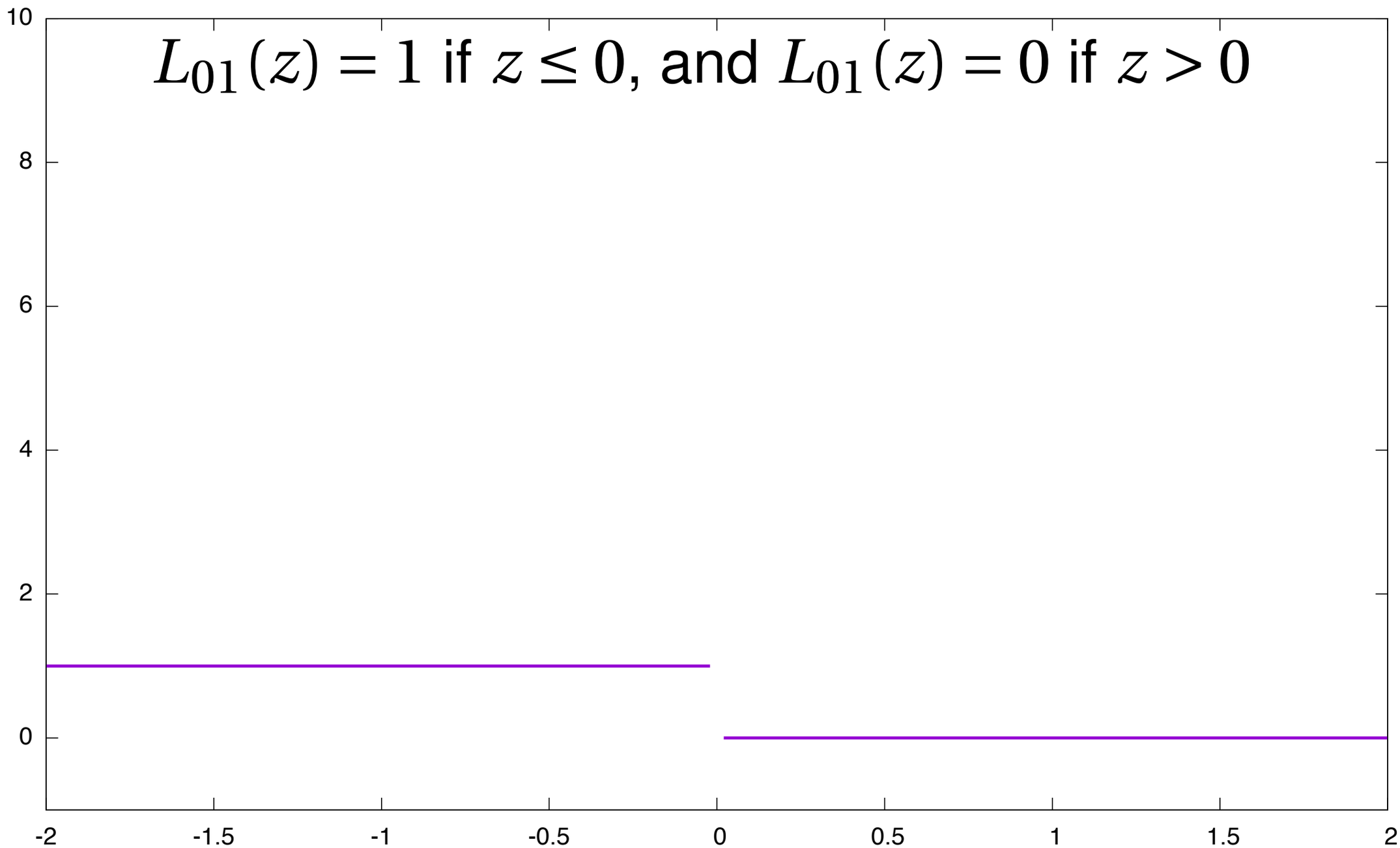
We will assume that $L(0) = 1$, which is the loss incurred by having an example on the decision boundary. We furthermore have $L(z) \geq 1$ for $z < 0$, and usually also $0 \leq L(z) < 1$ for $z > 0$.



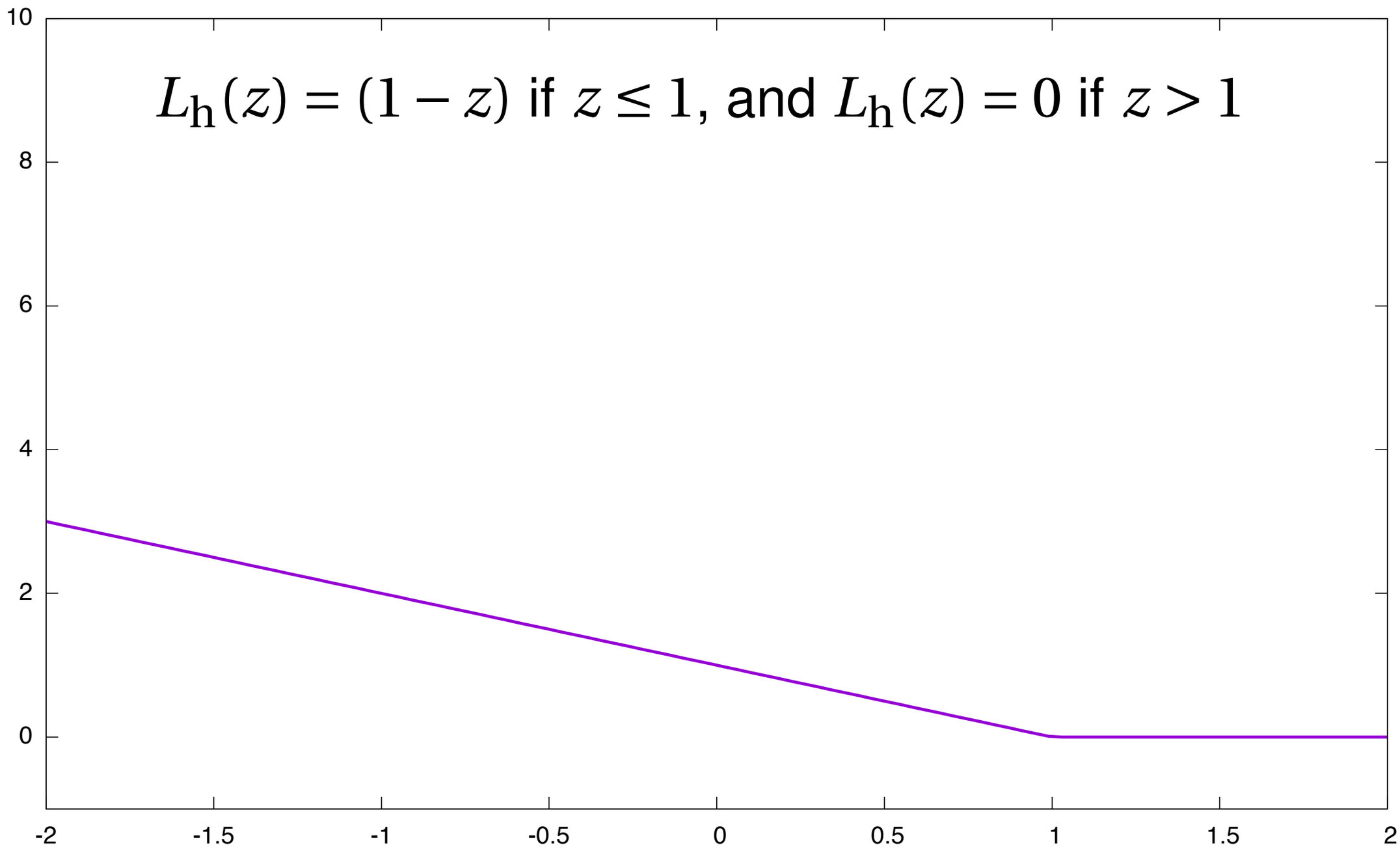
Loss functions

Loss functions are particularly important during learning since they are used to guide the search for the optimal solution. Changing the loss function may significantly change the quality of the output of the learning algorithm.

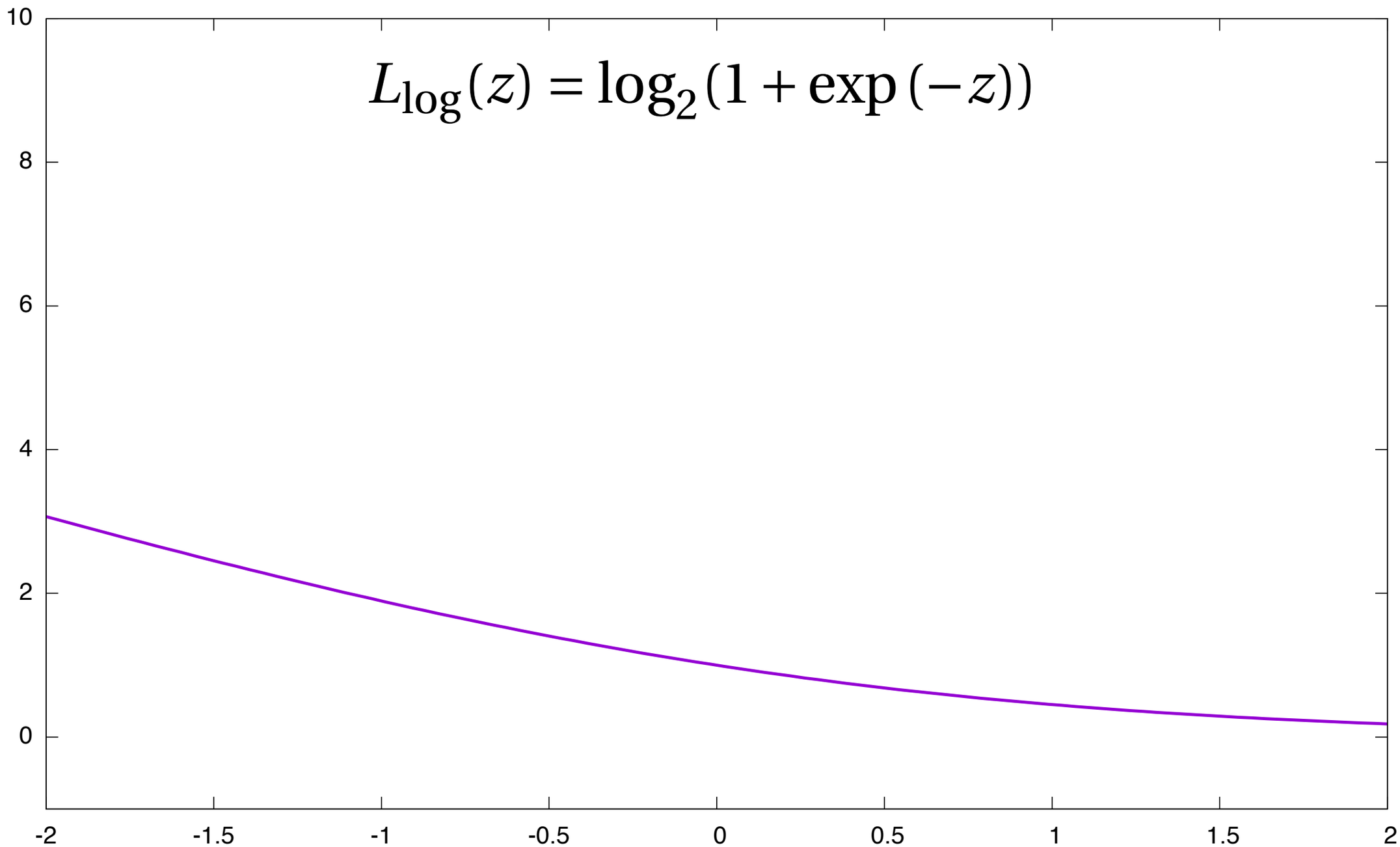
0-1 Loss



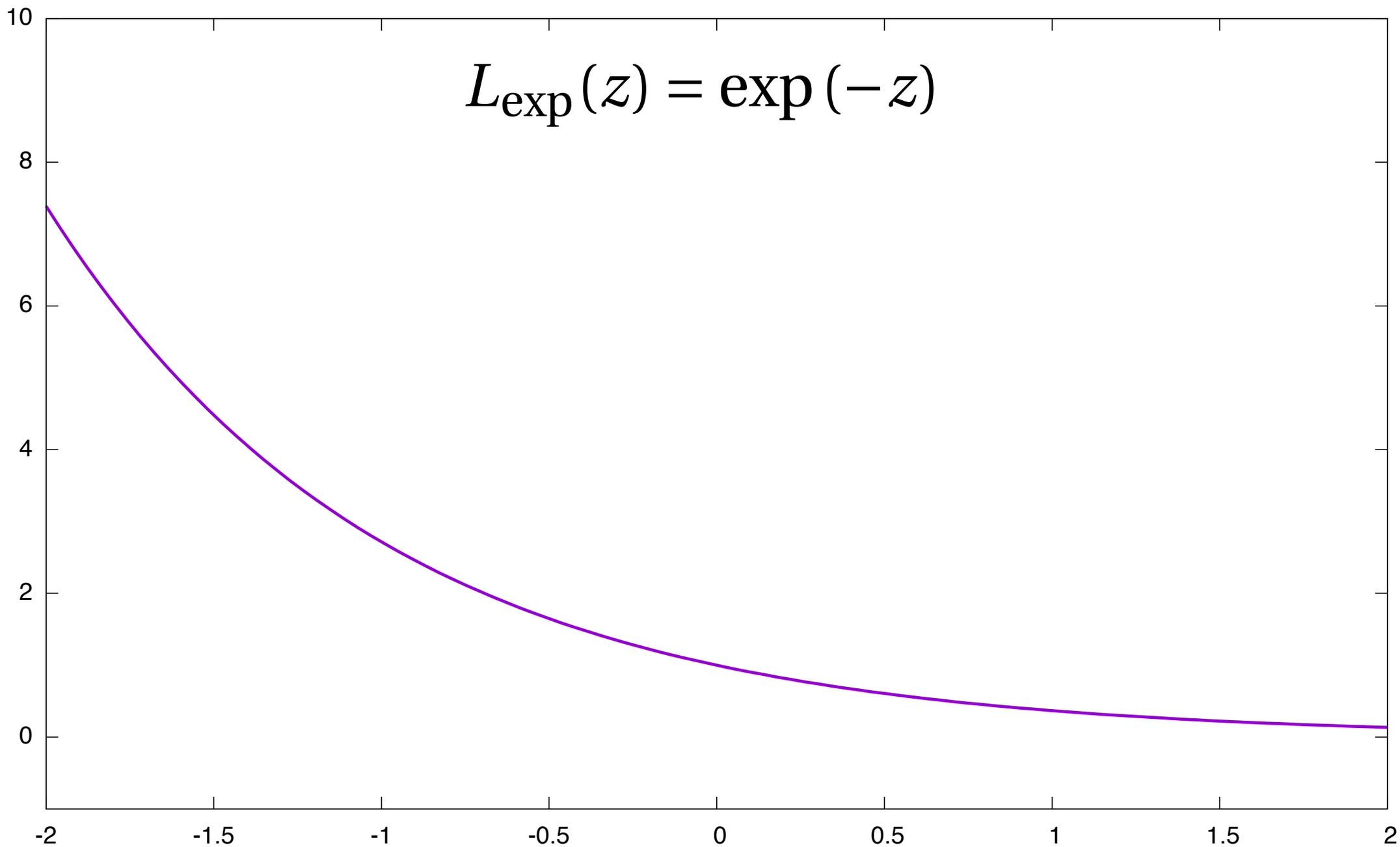
Hinge Loss


$L_{\mathrm{h}}(z) = (1 - z)$ if $z \leq 1$, and $L_{\mathrm{h}}(z) = 0$ if $z > 1$
10
8
6
4
2
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

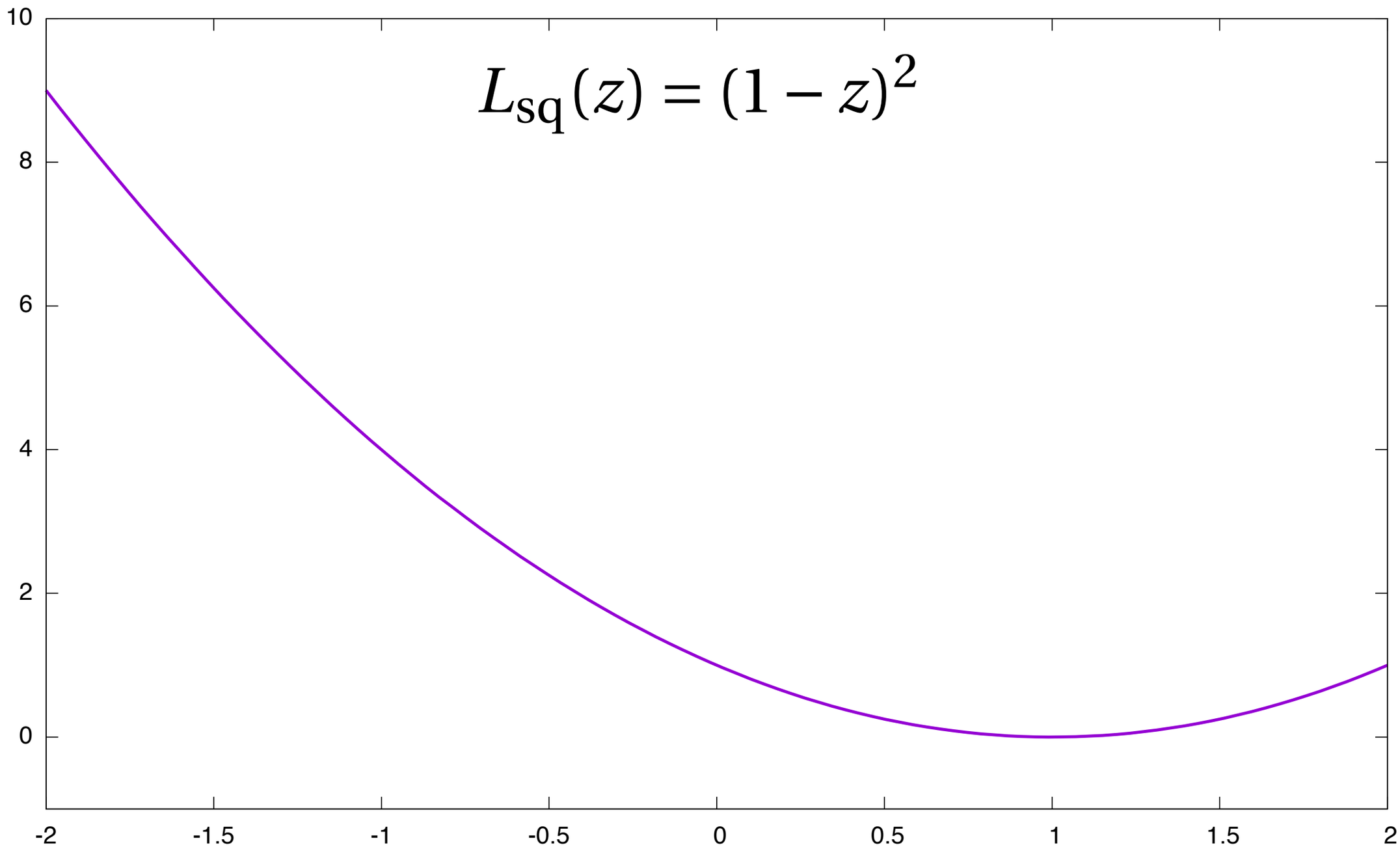
Logistic Loss



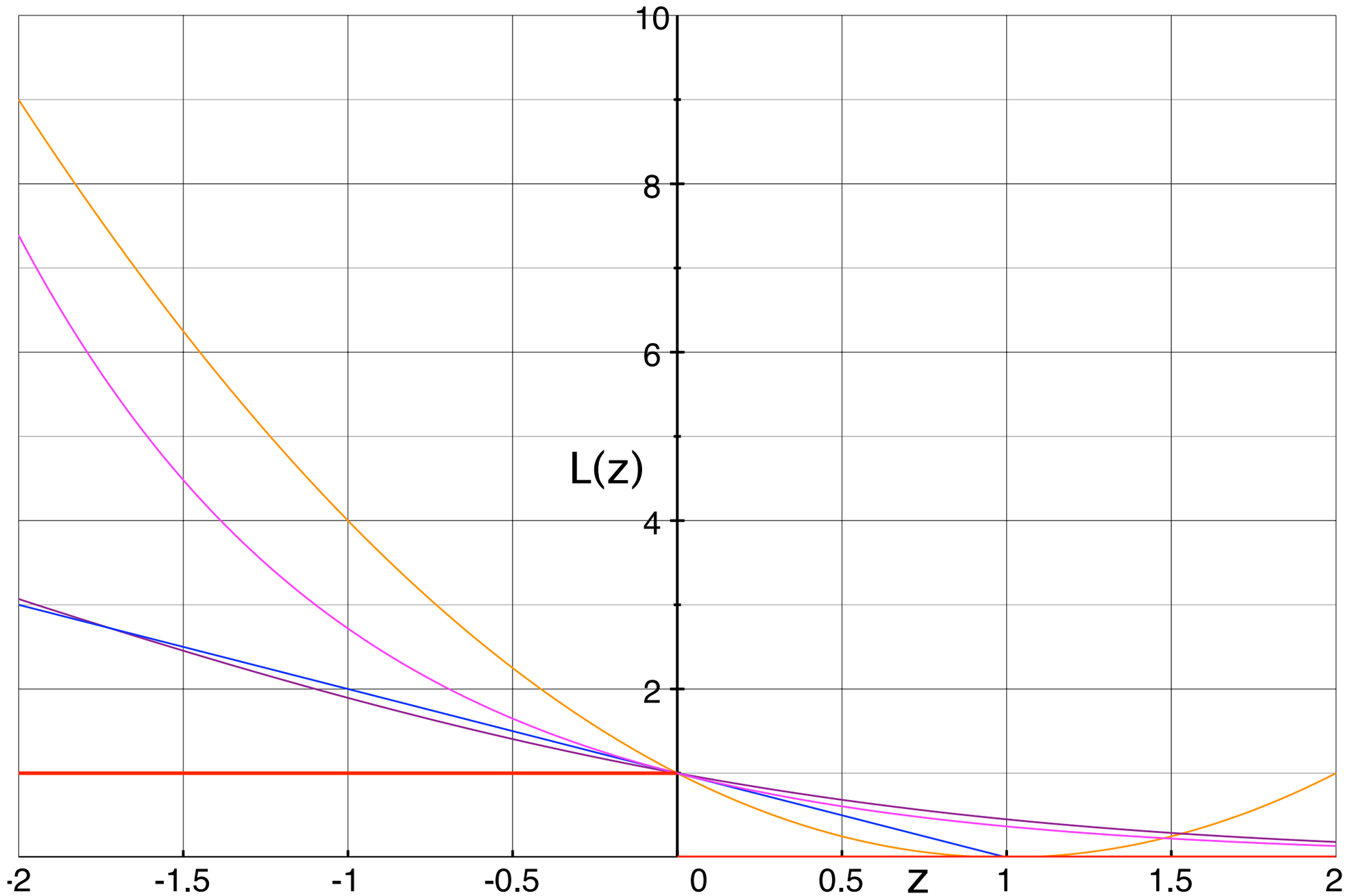
Exponential Loss



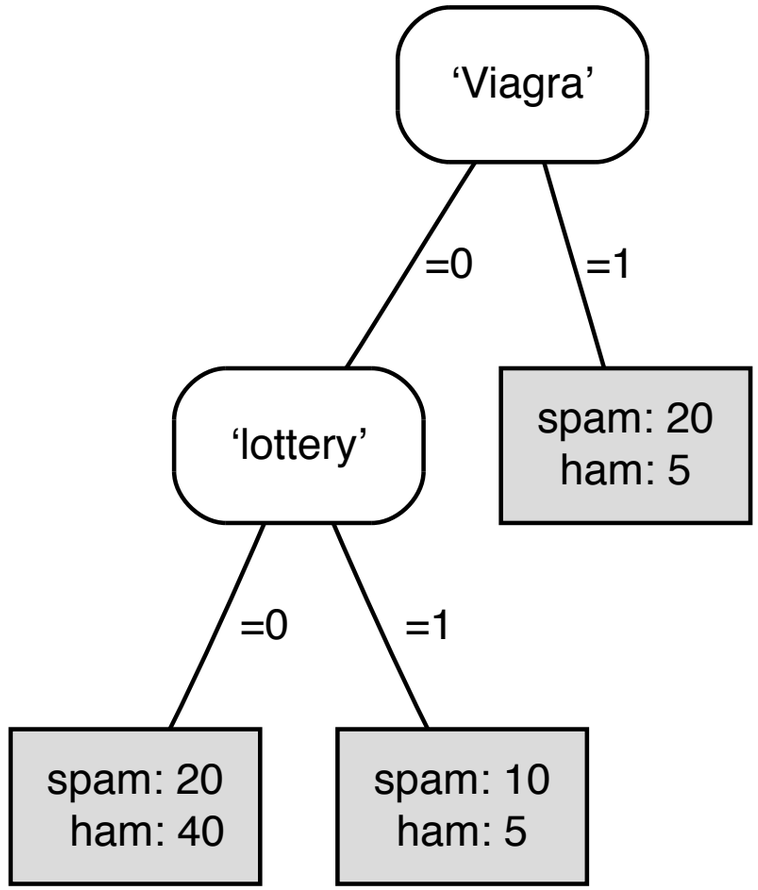
Squared Loss

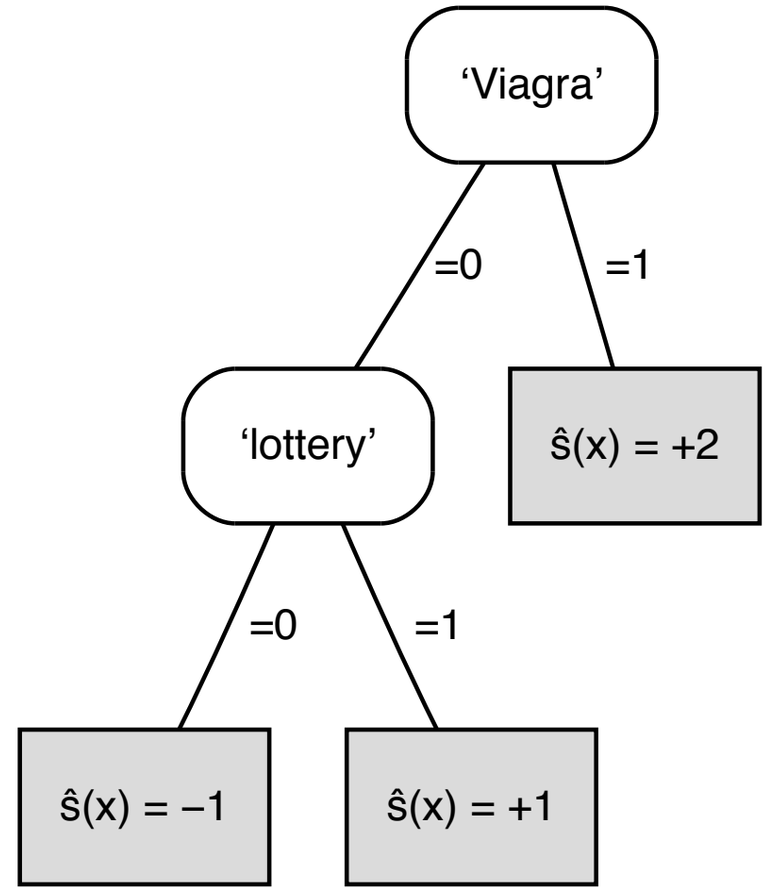


Loss functions



Ranking





A scoring function naturally induce a ranking function: it suffices to sort instances according to the received score.

The scoring on the right produce the following ranking of the tree leaves (and, thus, of the instances associated with them):

[20+,5-],[10+,5-],[20+,40-]

Ranking error rate

The **ranking error rate** is defined as:

$$rank\text{-}err = \frac{\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{Pos \cdot Neg}$$

1 point of penality due to a ranking error: a positive example is ranked below a negative example

1/2 point of penality for tying examples having different classes

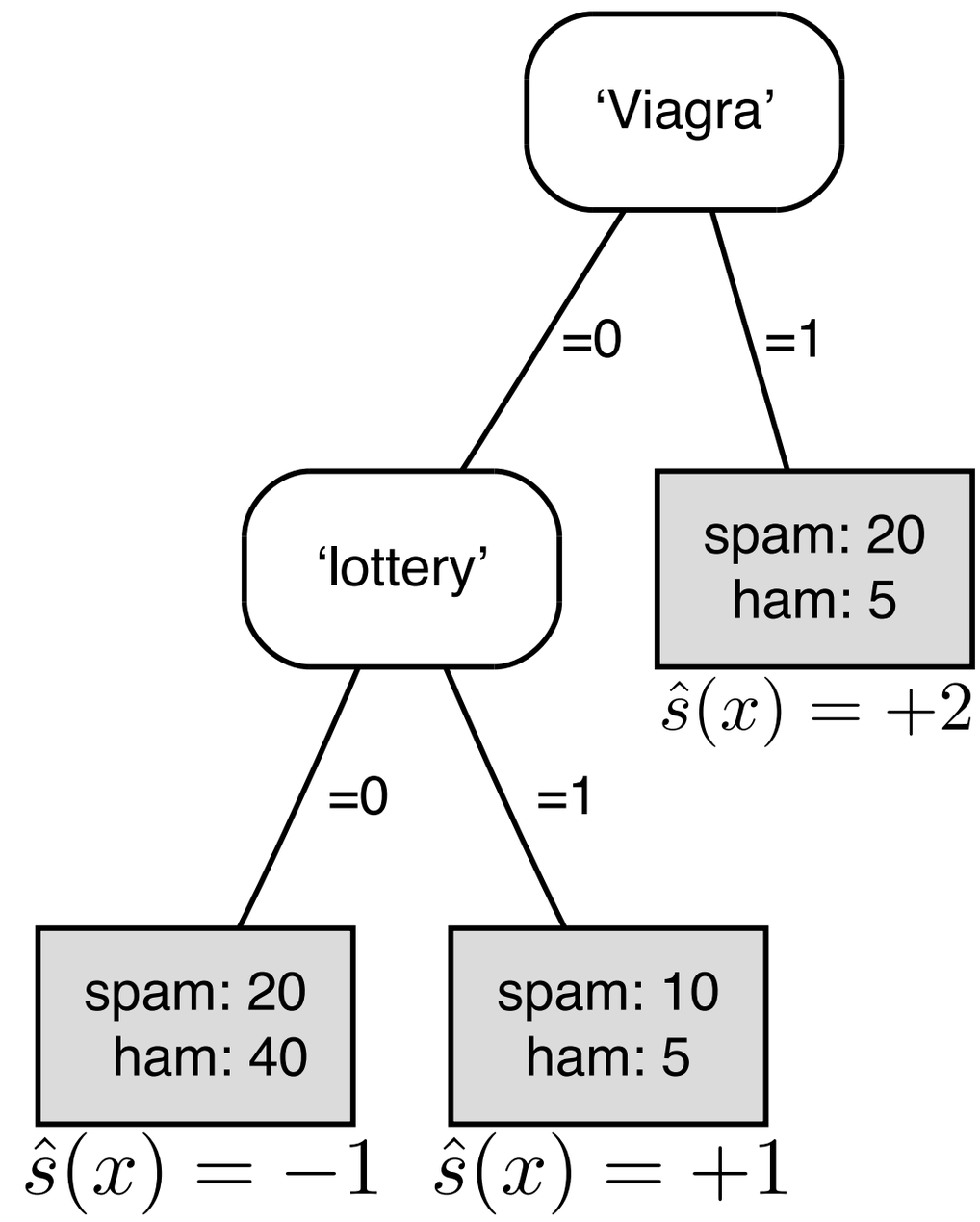
Ranking error examples

$$\text{rank-err}(x_1^+, x_2^+, x_3^-, x_4^+, x_5^+, x_6^-, x_7^-, x_8^-) = \frac{2}{16} = \frac{1}{8}$$

$$\text{rank-err}(x_1^-, x_2^-, x_3^-, x_4^-, x_5^-, x_6^+, x_7^+, x_8^+) = \frac{15}{5 \times 3} = 1$$

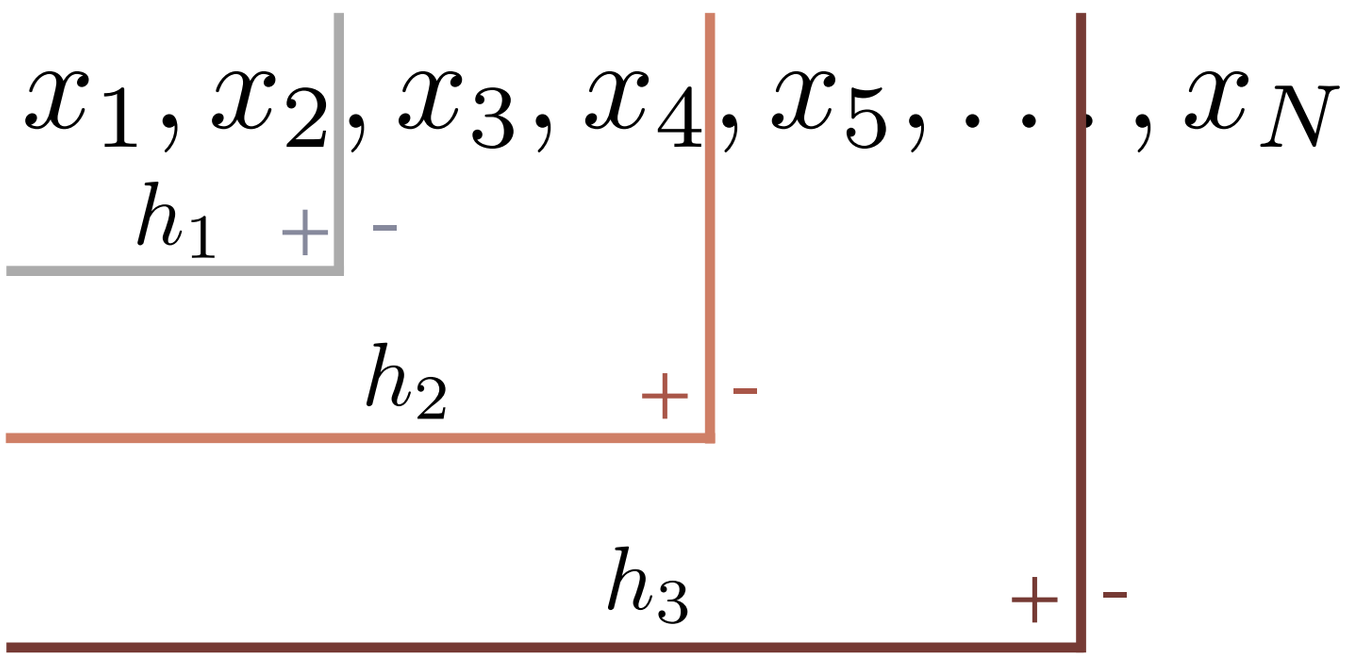
Ranking error example

- The 5 negatives in the right leaf are scored higher than the 10 positives in the middle leaf and the 20 positives in the left leaf, resulting in
  50 + 100 = 150 ranking errors.
- The 5 negatives in the middle leaf are scored higher than the 20 positives in the left leaf, giving a further 100 ranking errors.
- In addition, the left leaf makes 800 half ranking errors (because 20 positives and 40 negatives get the same score), the middle leaf 50 and the right leaf 100.
- In total we have 725 ranking errors out of a possible 50·50=2500, corresponding to a ranking error rate of 29% or a ranking accuracy of 71%.

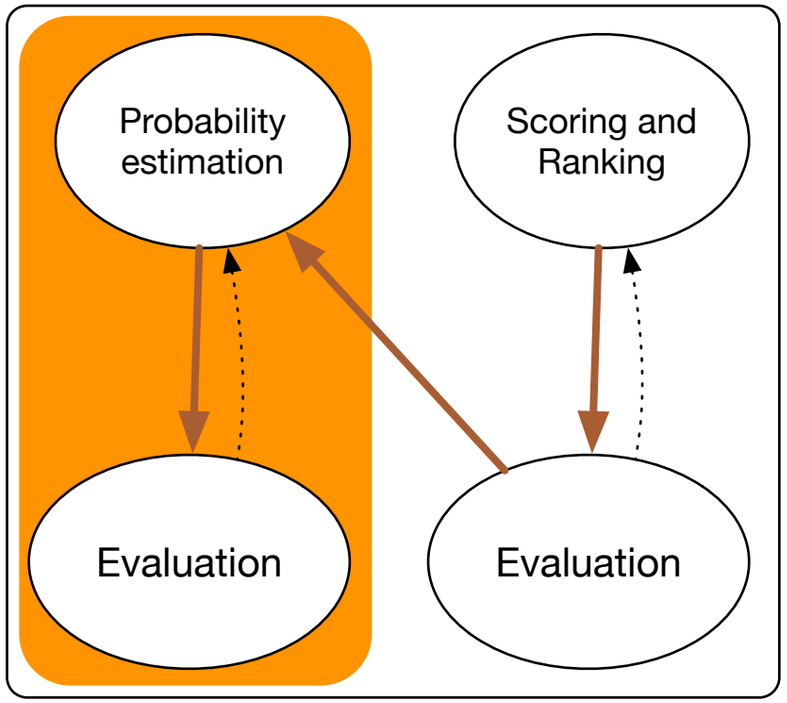


Binary classifiers and Ranking functions

Given a ranking function h one can create different binary classifiers based on h by choosing different thresholds:



Probability estimation



Class probability estimation

A class probability estimator is a scoring classifier that outputs probability vectors over classes, i.e., a mapping:

$$\hat{\mathbf{p}} : \mathscr{X} \rightarrow [0,1]^k$$

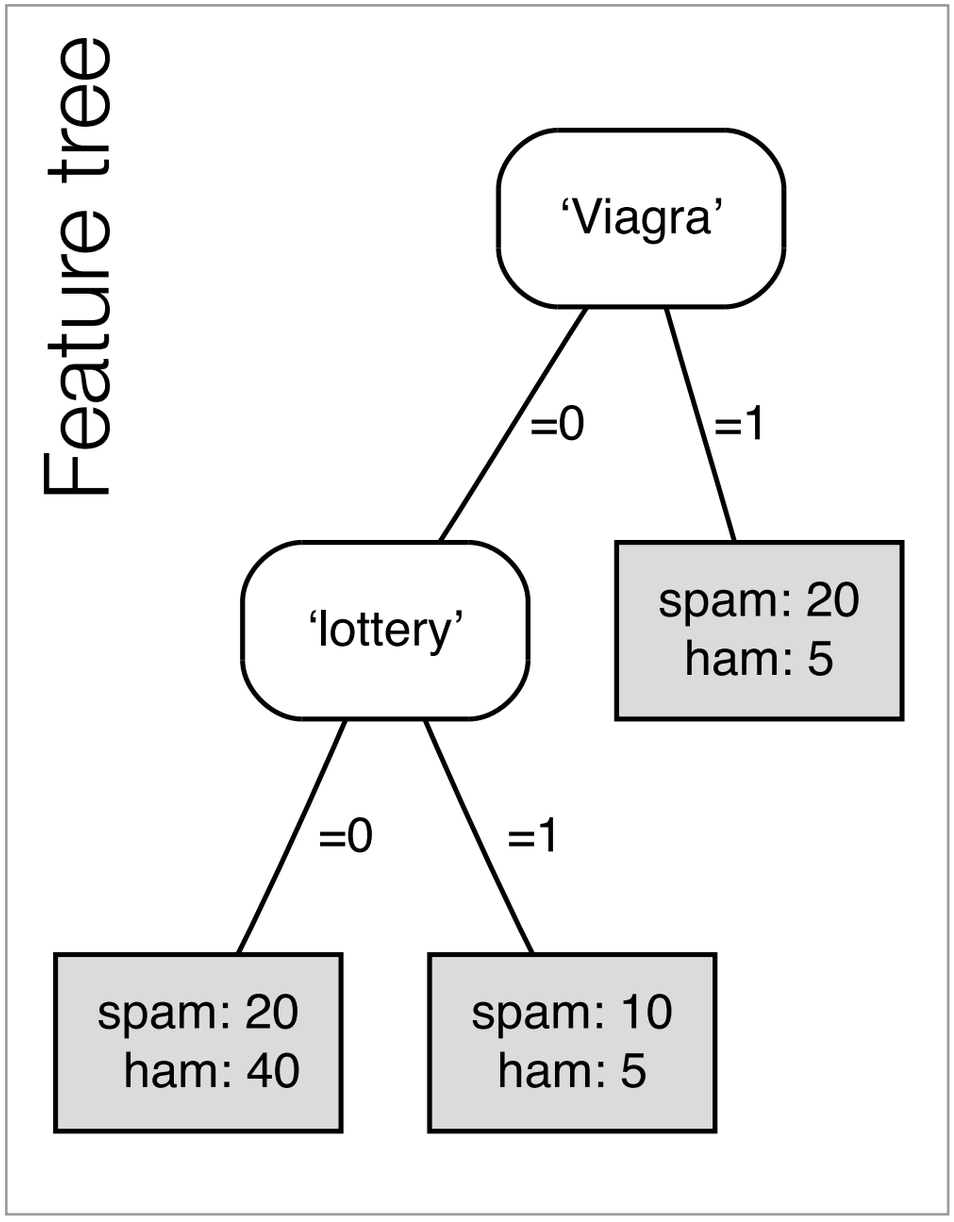
We write:

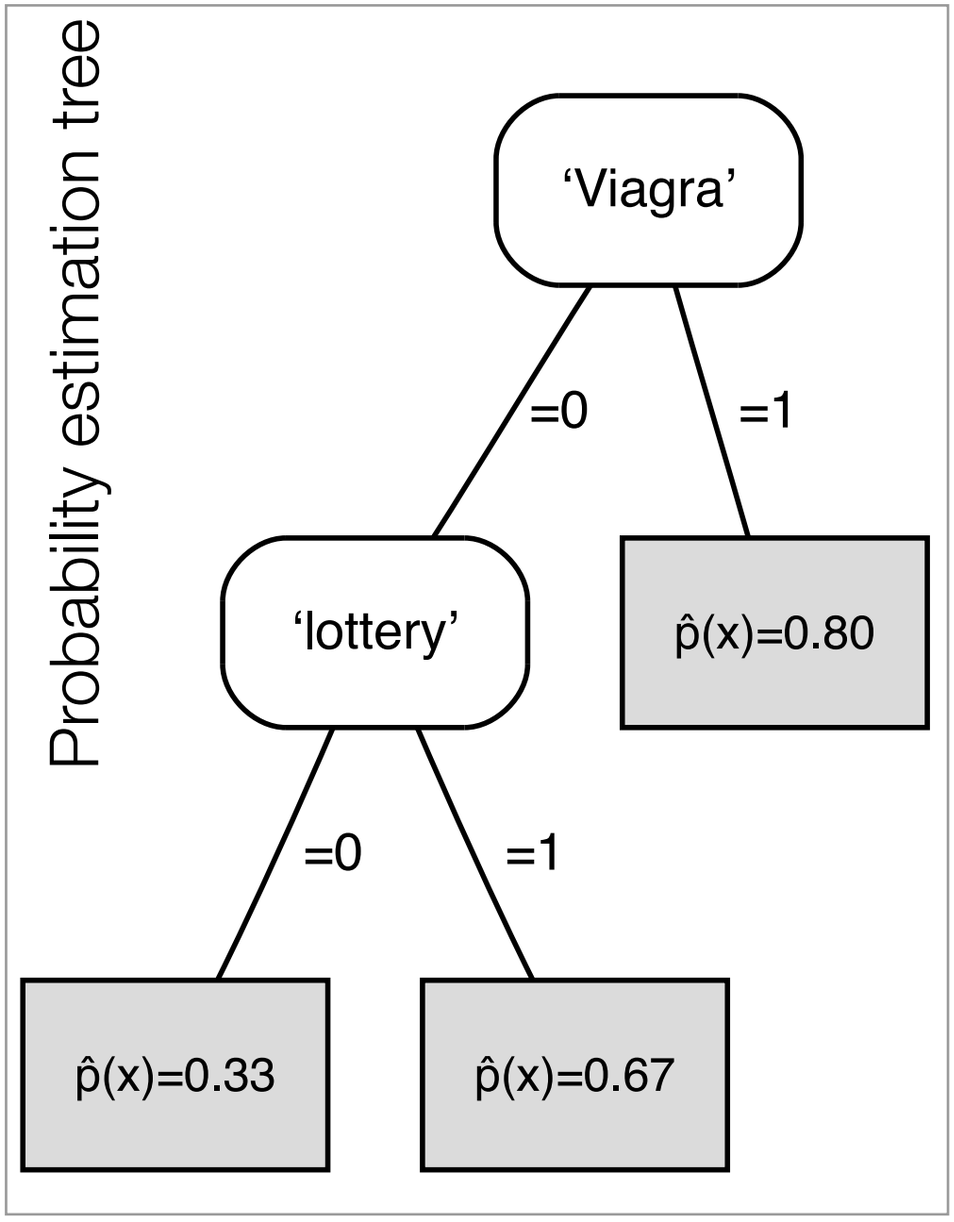
$$\hat{\mathbf{p}}(x) = \left(\hat{p}_1(x), \ldots, \hat{p}_k(x)\right)$$

where the i-th component is the probability assigned to class C_i and $\sum_{i=1}^{k} \hat{p}_i(x) = 1.$

if we have only two classes, then $\hat{p}(x)$ denotes the estimated probability for the positive class.

Probability estimation tree





Mean squared probability error

The **squared error** (SE) of the predicted probability vector on an example x is defined as:

$$\begin{aligned}\mathrm{SE}(x) &= \frac{1}{2}\|\hat{\mathbf{p}}(x) - I_{c(x)}\|_2^2 \\ &= \frac{1}{2}\sum_{i=1}^{k}(\hat{p}_i(x) - I[c(x) = C_i])^2\end{aligned}$$

where $I_{c(x)}$ is a vector having 1 in the position corresponding to label $c(x)$ and 0 in all other positions.

SE example (1)

Let us assume: $\hat{\mathbf{p}}(x) = (0.7, 0.1, 0.2)$

and that $c(x)=C_1$ yielding $I_{c(x)}=(1,0,0)$

$\mathrm{SE}(x)$ would then be evaluated as:

$$\begin{aligned}
\mathrm{SE}(x) &= \frac{\|(0.7, 0.1, 0.2) - (1, 0, 0)\|_2^2}{2} \\
&= \frac{\|(-0.3, 0.1, 0.2)\|_2^2}{2} \\
&= \frac{0.09 + 0.01 + 0.04}{2} \\
&= \frac{0.14}{2} = 0.07
\end{aligned}$$

SE example (2)

Let us assume: $\hat{\mathbf{p}}(x) = (0, 1, 0)$

and that $c(x)=C_1$ yielding $I_{c(x)}=(1,0,0)$

$\mathrm{SE}(x)$ would then be evaluated as:

$$\begin{aligned}
\mathrm{SE}(x) &= \frac{\|(0,1,0) - (1,0,0)\|_2^2}{2} \\
&= \frac{\|(-1,1,0)\|_2^2}{2} \\
&= \frac{1+1+0}{2} \\
&= \frac{2}{2} = 1
\end{aligned}$$

SE example (3)

Consider these cases:

$c(x) = C_1$

$$\hat{\mathbf{p}}(x) = (0.7, 0.1, 0.2) \text{ yielding } \mathrm{SE}(x) = 0.07$$

$$\hat{\mathbf{p}}(x) = (0.99, 0, 0.01) \text{ yielding } \mathrm{SE}(x) = 0.0001$$

$c(x) = C_3$

$$\hat{\mathbf{p}}(x) = (0.7, 0.1, 0.2) \text{ yielding } \mathrm{SE}(x) = 0.57$$

$$\hat{\mathbf{p}}(x) = (0.99, 0, 0.01) \text{ yielding } \mathrm{SE}(x) = 0.98$$

Mean squared error

The mean squared error is simply the average SE over all instances in the test set:

$$\mathrm{MSE}(Te) = \frac{1}{|Te|} \sum_{x \in Te} \mathrm{SE}(x)$$

Exercise: evaluate the squared error of the tree in our running example:

- assuming that estimated probabilities for the three leafs (from left to right) are 0.33,0.67, and 0.80 (solution: ~0.21)
- assuming that estimated probabilities are instead 0.10,0.80, and 0.90 (solution: ~0.24)

Empirical Probabilities

Empirical probabilities are important as they allow us to obtain or finetune probability estimates from classifiers or rankers. If we have a set S of labelled examples, and the number of examples in S of class C_i is denoted n_i , then the empirical probability vector associated with S is:

$$\dot{p}(S) = (n_1/|S|, \ldots, n_k/|S|)$$

Laplace correction

It is almost always a good idea to smooth these relative frequencies. The most common way to do this is by means of the **Laplace correction**:

$$\dot{p}_i(S) = \frac{n_i + 1}{|S| + k}$$

In effect, we are adding uniformly distributed pseudo-counts to each of the k alternatives, reflecting our prior belief that the empirical probabilities will turn out uniform.

We can also apply non-uniform smoothing by setting

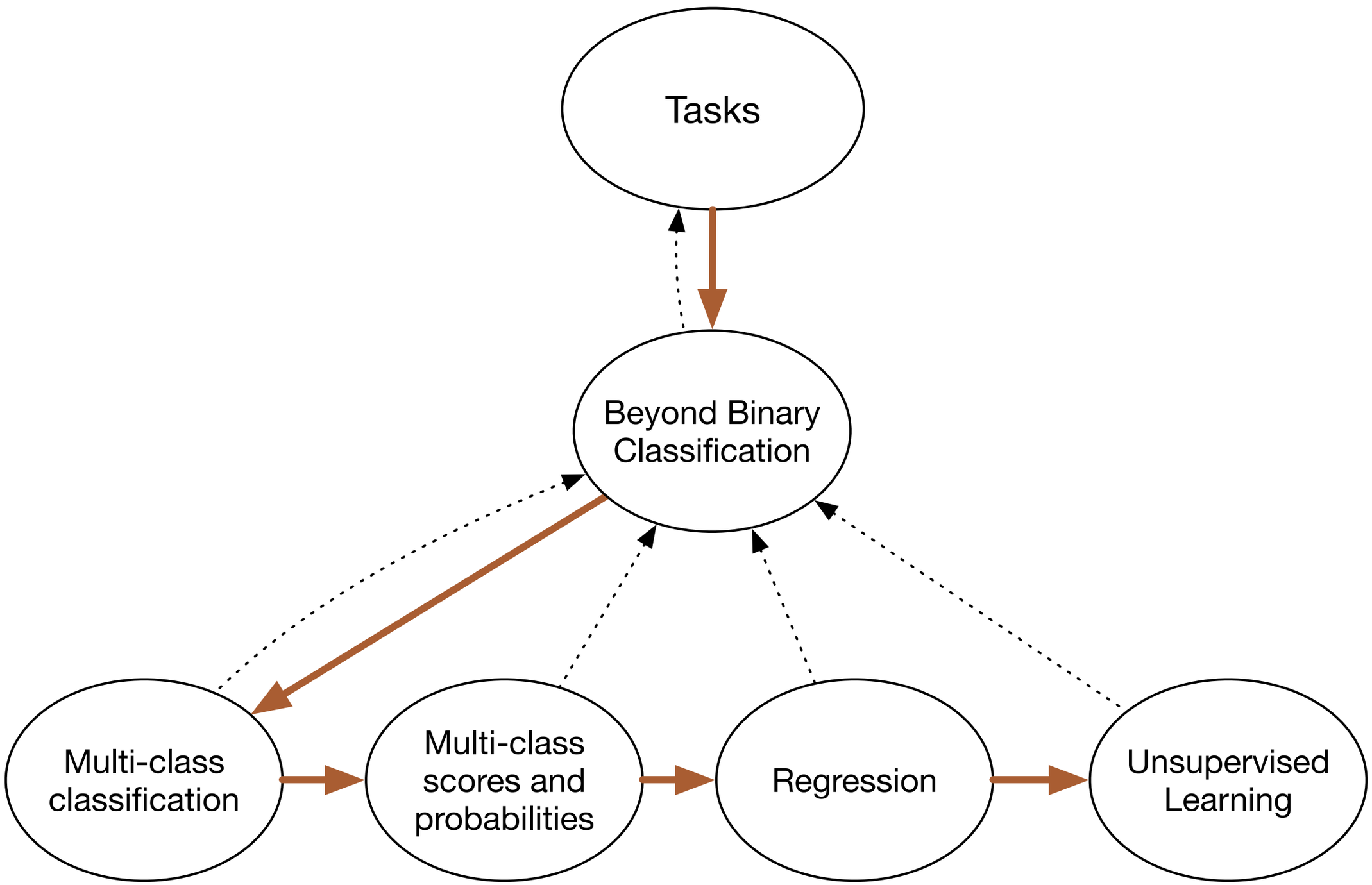
$$\dot{p}_i(S) = \frac{n_i + m \cdot \pi_i}{|S| + m}$$

This smoothing technique, known as the m-estimate, allows the choice of the number of pseudo-counts m as well as the prior probabilities π_i.

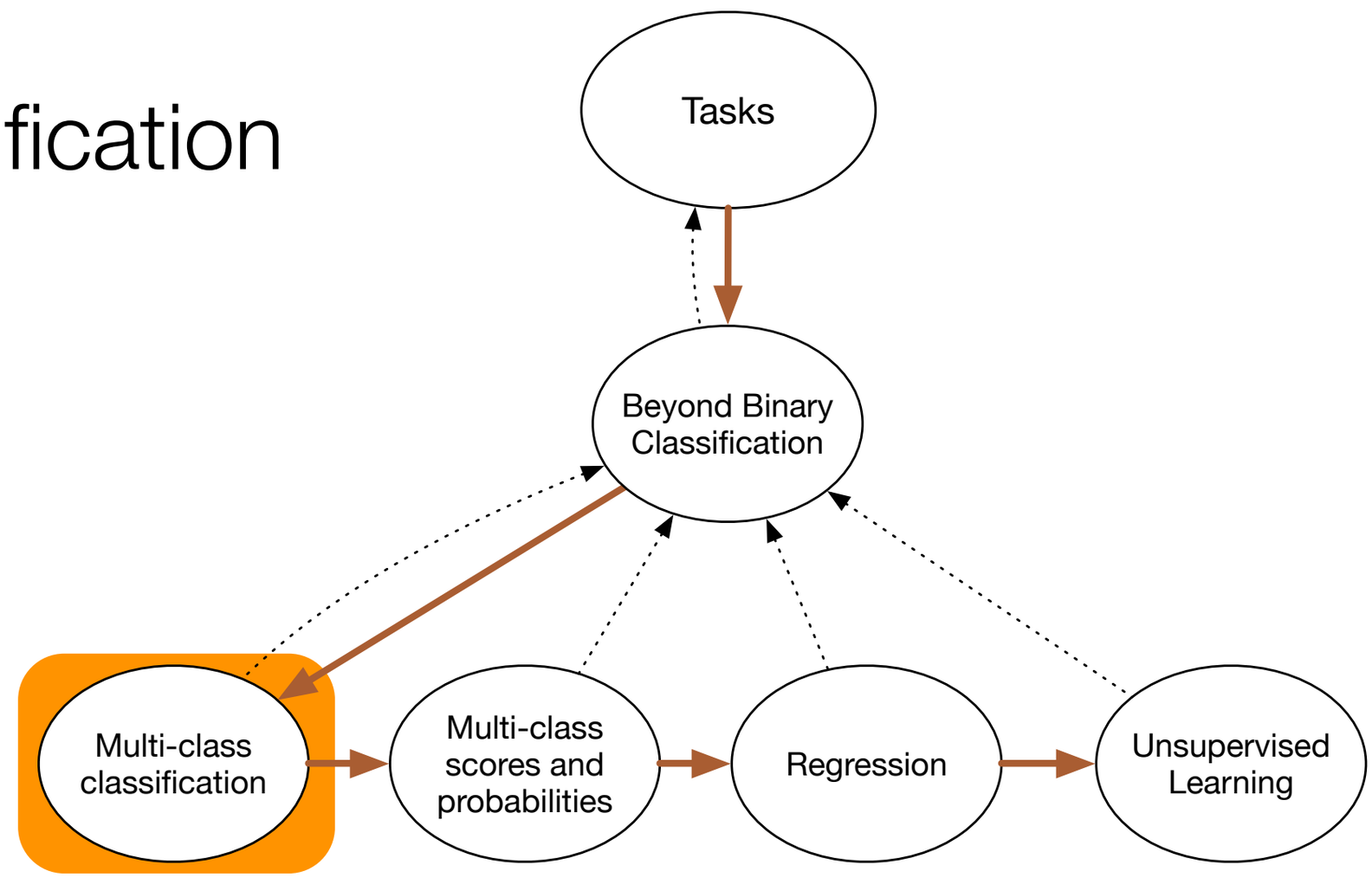
The Laplace correction is a special case of the m-estimate with $m = k$ and $\pi_i = 1/k$.

Beyond binary classification

Beyound binary classifications topics



Multi-class classification



Performance of multi-class classifiers

	Predicted				**per-class recalls**
	15	**2**	**3**	20	**15**/**20**=0.75
Actual	**7**	**15**	**8**	30	**15**/**30**=0.50
	2	**3**	**45**	50	**45**/**50**=0.90
	24	20	56	100	
per-class precisions	**15**/**24**=0.63	**15**/**20**=0.75	**45**/**56**=0.8		accuracy **15**+**15**+**45** / **100** = 0.75

Extending binary classifiers to handle multi-class labelings

Various ways to combine several binary classifiers into a single k-class classifier:

- ✦ one-vs-rest schemes
  - unordered learning
  - fixed-order learning
- ✦ one-vs-one schemes
  - symmetric
  - asymmetric

one-vs-rest (unordered)

Train k classifiers:

initial dataset

x	y
...	1
	3
	2
	1
	3

1 vs {2,3}

x	y
...	1
	-1
	-1
	1
	-1

$\hat{c}_1$

2 vs {1,3}

x	y
...	-1
	-1
	1
	-1
	-1

$\hat{c}_2$

3 vs {1,2}

x	y
...	-1
	1
	-1
	-1
	1

$\hat{c}_3$

output-code matrix for one-vs-rest (unordered)

$$
\begin{array}{c}
\\
C_1 \\
C_2 \\
C_3
\end{array}
\begin{array}{c}
\begin{array}{ccc}
1 & 2 & 3 \\
vs & vs & vs \\
\{2,3\} & \{1,3\} & \{1,2\}
\end{array} \\
\begin{pmatrix}
+1 & -1 & -1 \\
-1 & +1 & -1 \\
-1 & -1 & +1
\end{pmatrix}
\end{array}
$$

output-code matrix for one-vs-rest (fixed-order)

$$
\begin{array}{c}
 \\
C_1 \\
C_2 \\
C_3
\end{array}
\begin{array}{c}
\begin{array}{cc} \textit{1} & \textit{2} \\ \textit{vs} & \textit{vs} \\ \textit{\{2,3\}} & \textit{\{3\}} \end{array} \\
\begin{pmatrix} +1 & 0 \\ -1 & +1 \\ -1 & -1 \end{pmatrix}
\end{array}
$$

output-code matrix for one-vs-one (symmetric)

$$
\begin{array}{c}
\\
C_1 \\
C_2 \\
C_3
\end{array}
\begin{array}{c}
\begin{array}{ccc} 1 & 1 & 2 \\ vs & vs & vs \\ 2 & 3 & 3 \end{array} \\
\begin{pmatrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{pmatrix}
\end{array}
$$

output-code matrix for one-vs-one (asymmetric)

$$
\begin{array}{c}
\\
C_1 \\
C_2 \\
C_3
\end{array}
\begin{array}{c}
\begin{array}{cccccc}
1 & 2 & 1 & 3 & 2 & 3 \\
vs & vs & vs & vs & vs & vs \\
2 & 1 & 3 & 1 & 3 & 2
\end{array} \\
\begin{pmatrix}
+1 & -1 & +1 & -1 & 0 & 0 \\
-1 & +1 & 0 & 0 & +1 & -1 \\
0 & 0 & -1 & +1 & -1 & +1
\end{pmatrix}
\end{array}
$$

Output-code decoding

To classify a new example, a vector w is built containing the output of the learnt classifiers. The output class is the one whose row in the output-code matrix is the *nearest* to w. Distance between code-words and output vectors is defined as: $d(w, c) = \sum_i (1 - c_i w_i)/2$

$$
\begin{array}{c}
 \\ C_1 \\ C_2 \\ C_3
\end{array}
\begin{array}{c}
\begin{array}{ccc} \textit{1 vs \{2,3\}} & \textit{2 vs \{1,3\}} & \textit{3 vs \{1,2\}} \end{array} \\
\begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix}
\end{array}
$$

$w = (+1\ -1\ -1)$

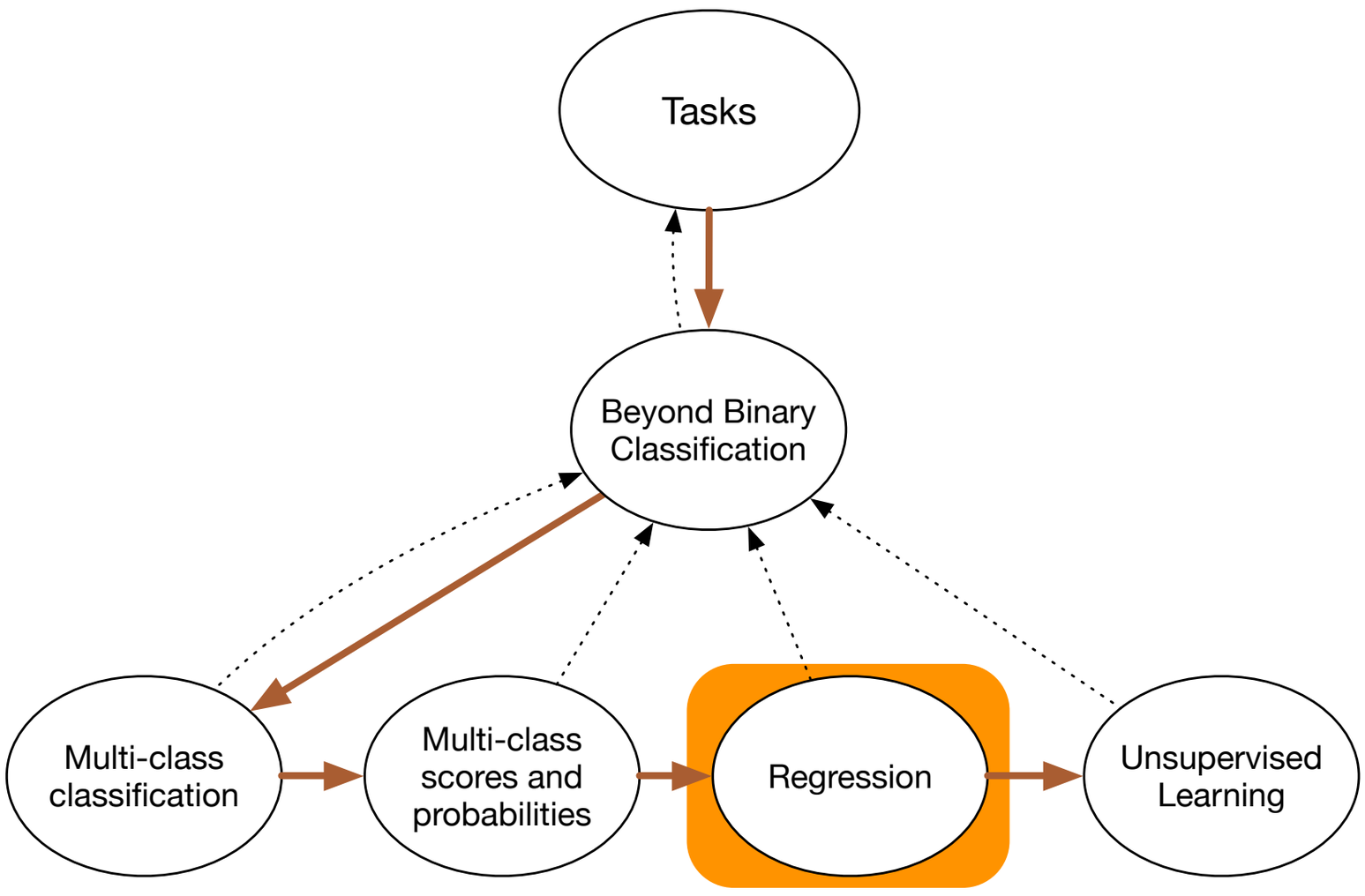
$$d(w, C_1) = 0$$
$$d(w, C_2) = 2$$
$$d(w, C_3) = 2$$

Difficulties in applying one-vs-rest and one-vs-one schemes

In the one-vs-rest scheme the single classifiers usually see highly unbalanced datasets even though the initial dataset is balanced.

In the one-vs-one scheme the above problem is mitigated by assigning the 0 label to examples not belonging to the two labels being assessed. Particularly problematic when data is scarce.

Regression


Tasks
Beyond Binary Classification
Multi-class classification
Multi-class scores and probabilities
Regression
Unsupervised Learning

Real-valued targets

A **function estimator**, also called a regressor, is a mapping

$$\hat{f} : \mathscr{X} \rightarrow \mathbb{R}$$

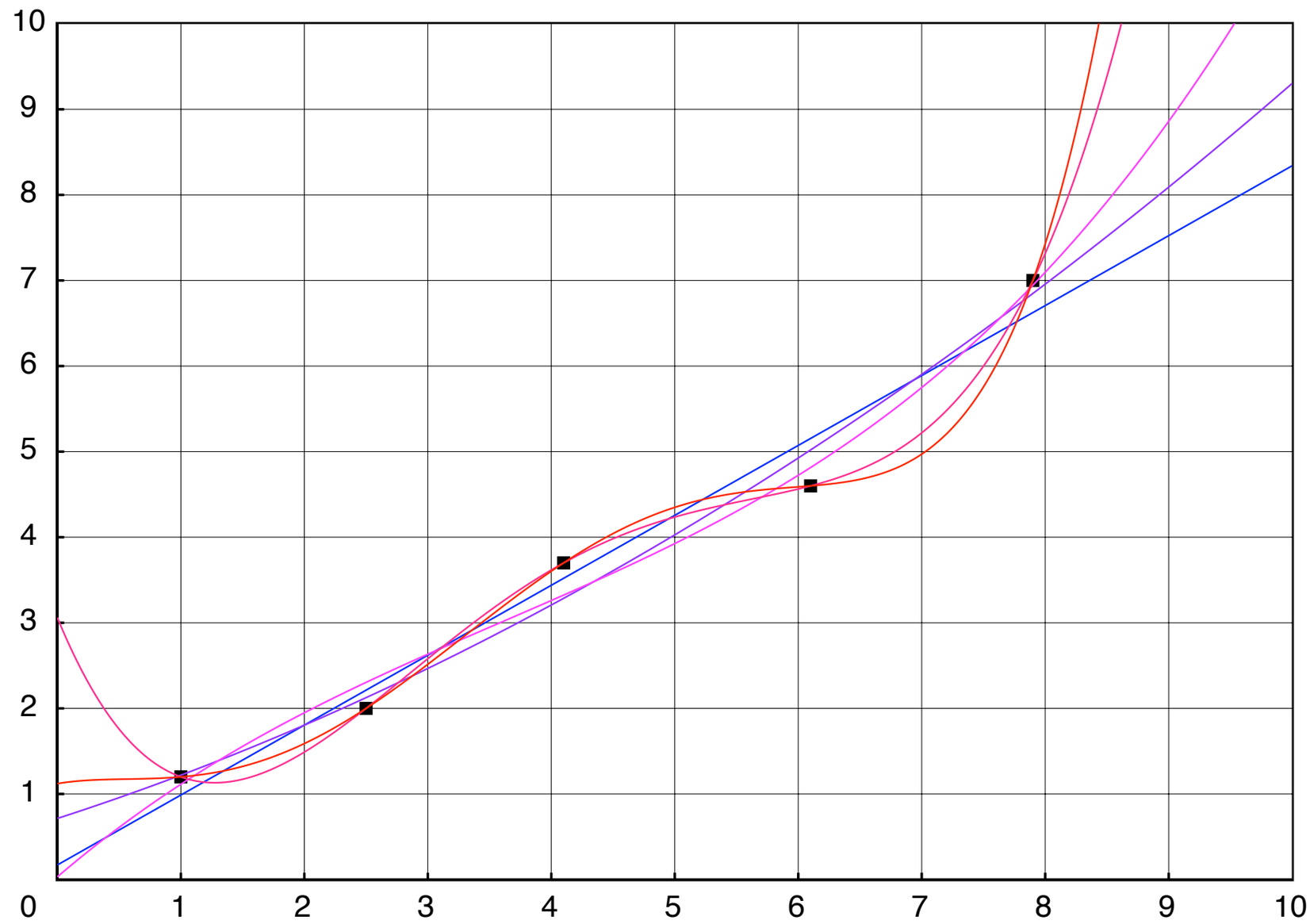
The regression learning problem is to learn a function estimator from examples $(x_i , f(x_i))$.

Note that we switched from a relatively low-resolution target variable to one with infinite resolution. Trying to match this precision in the function estimator will almost certainly lead to overfitting – besides, it is highly likely that some part of the target values in the examples is due to fluctuations that the model is unable to capture.

It is therefore entirely reasonable to assume that the examples are noisy, and that the estimator is only intended to capture the general trend or shape of the function.

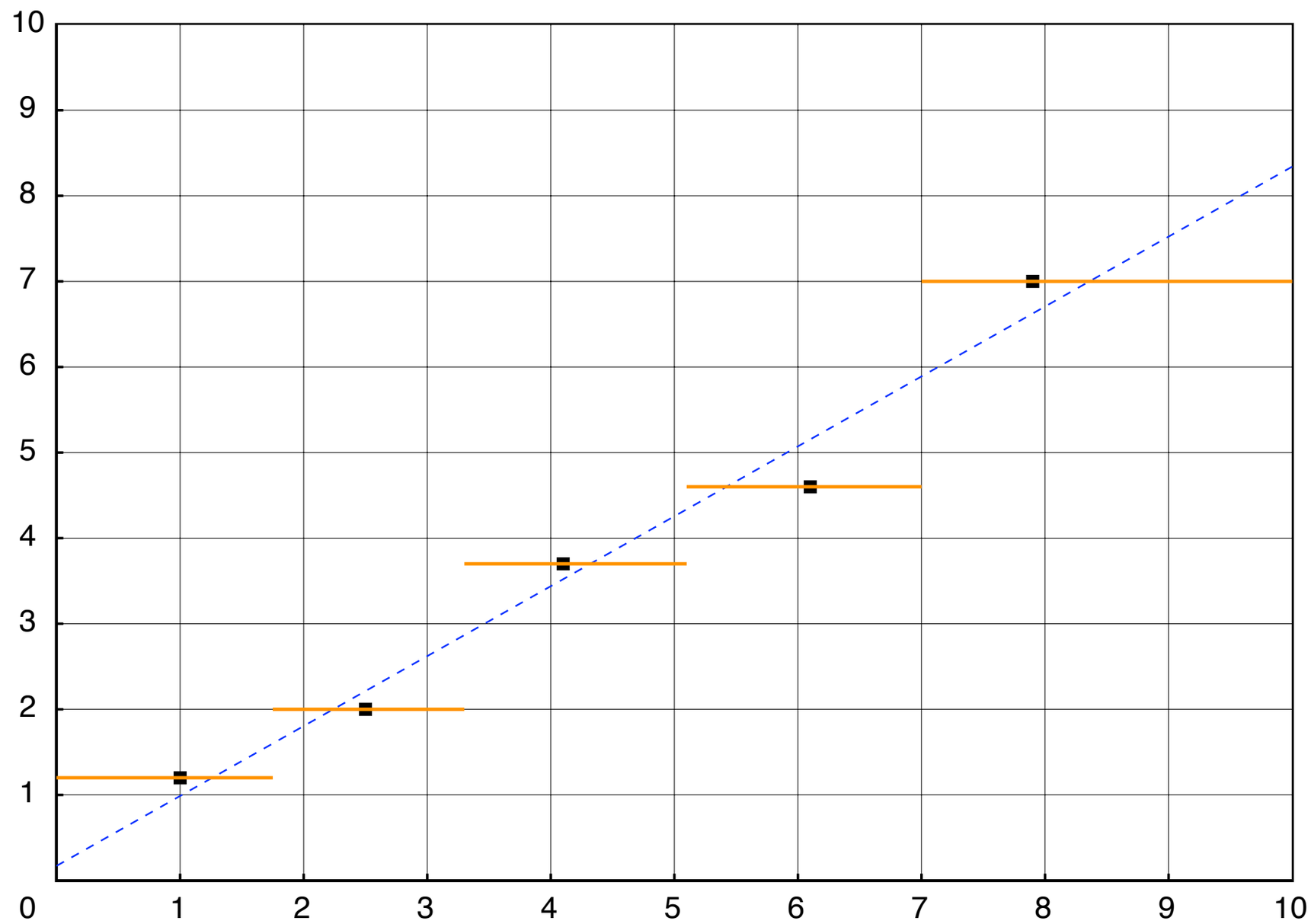
Fitting polynomials to data

x	y
1.0	1.2
2.5	2.0
4.1	3.7
6.1	4.6
7.9	7.0



A piecewise constant function

x	y
1.0	1.2
2.5	2.0
4.1	3.7
6.1	4.6
7.9	7.0



Overfitting

An n-degree polynomial has $n+1$ parameters and it is always able to match up to $n+1$ points.

A piecewise constant model with n segments has $2n-1$ parameters and it is always able to match up to n points.

Usually it is true that the larger the number of parameters, the larger the number of points the model is able to match (independently on how they are positioned in the space).

However: a rule of thumb is that, ***to avoid overfitting, the number of parameters estimated from the data must be considerably less than the number of data points***.

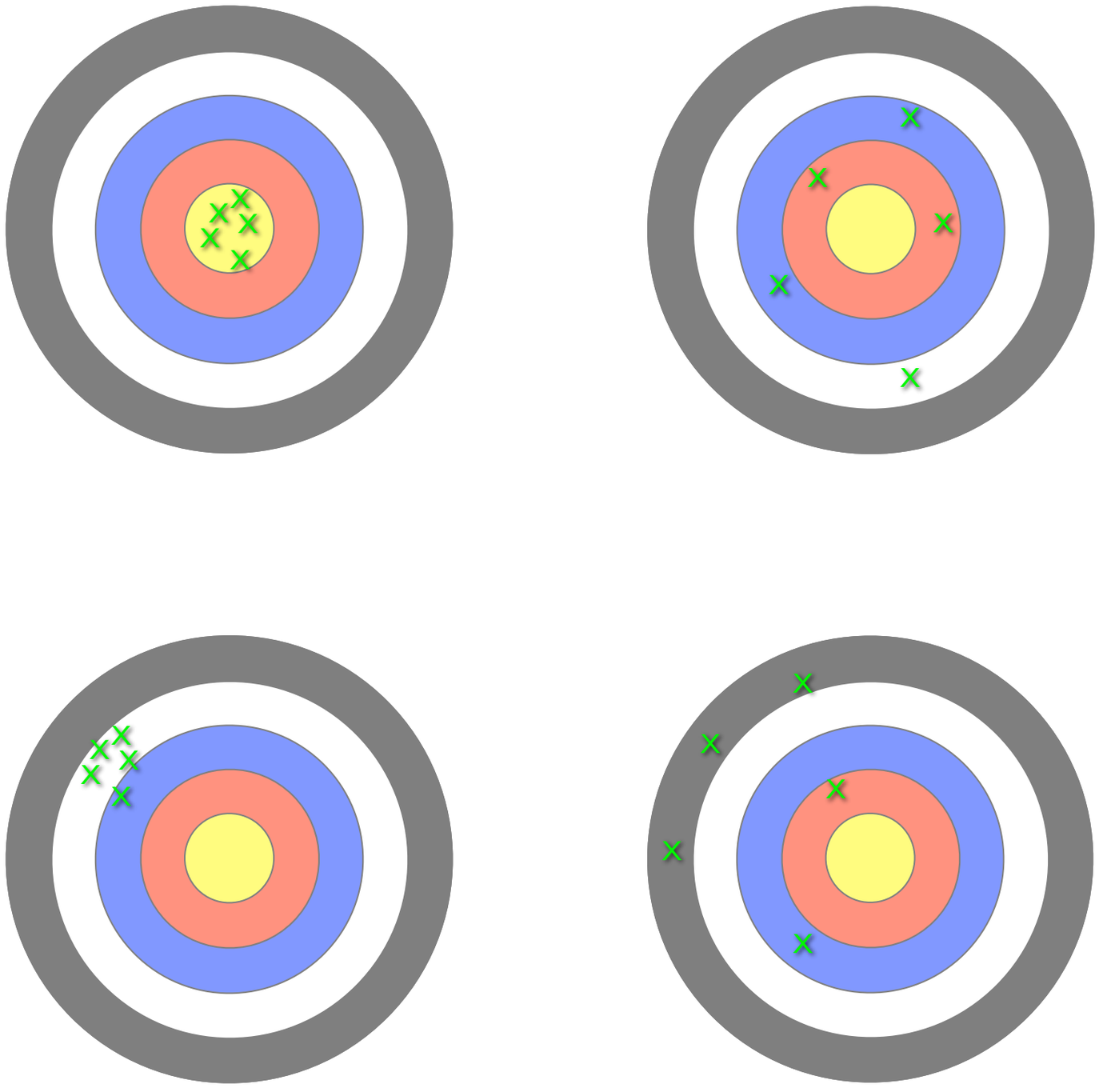
Bias and variance

If we underestimate the number of parameters of the model, we will not be able to decrease the loss to zero, regardless of how much training data we have.

On the other hand, with a larger number of parameters the model will be more dependent on the training sample, and small variations in the training sample can result in a considerably different model.

This is sometimes called the **bias–variance dilemma**: a low-complexity model suffers less from variability due to random variations in the training data, but may introduce a systematic bias that even large amounts of training data can't resolve; on the other hand, a high-complexity model eliminates such bias but can suffer non-systematic errors due to variance.

Bias and Variance



Bias and variance

Interestingly this idea can be captured formally. In fact the expected loss of the regressor over example x can be decomposed as:

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

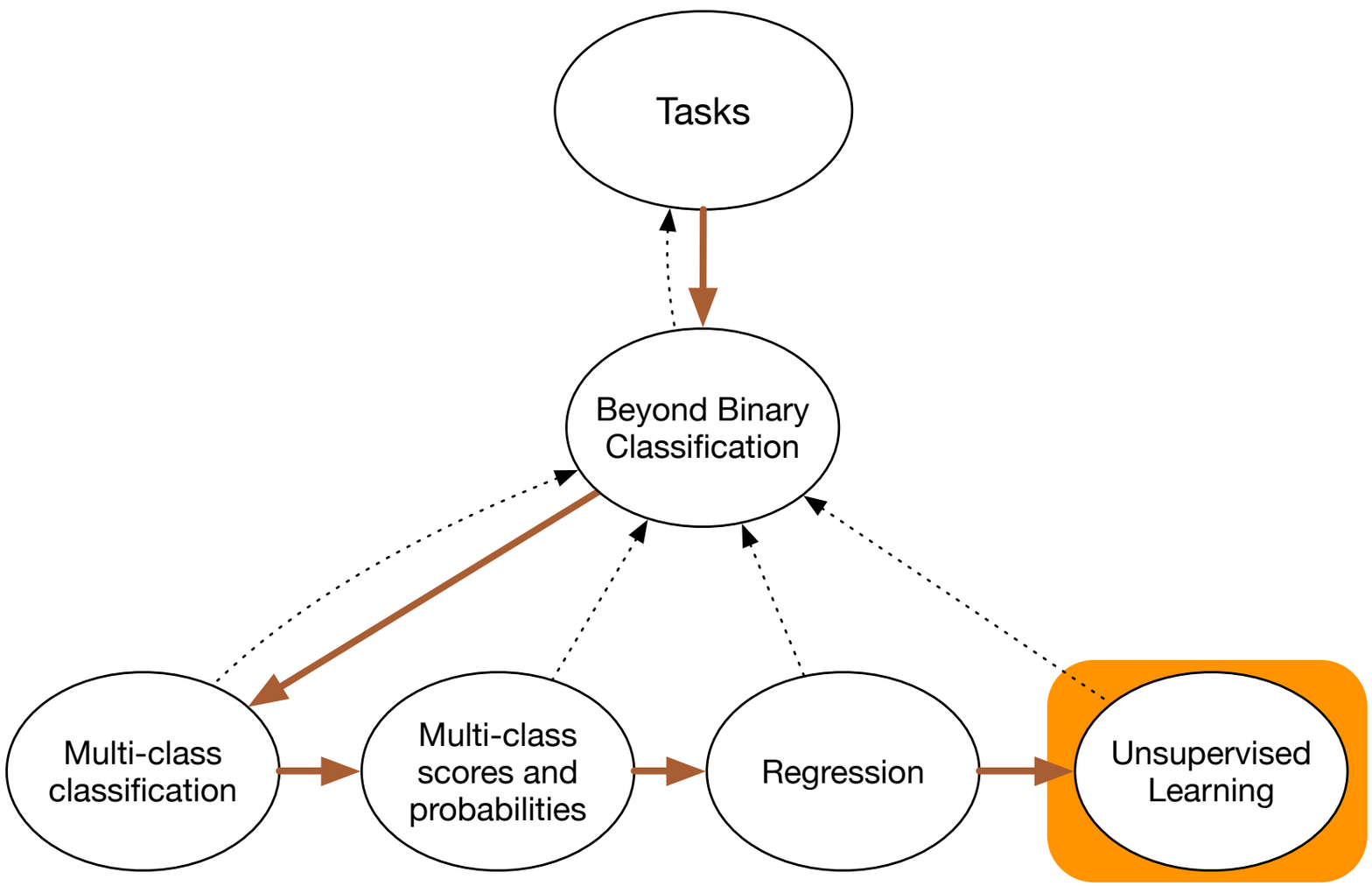
$\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2$ zero if the regressor is correct on average, otherwise it exhibit a systematic **bias**

$\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right]$ error due to fluctuations around the average (i.e., the **variance** of the error)

Bias and variance

$$
\begin{aligned}
E[(f(x) - \hat{f}(x)^2] &= E[f(x)^2 - 2f(x)\hat{f}(x) + \hat{f}(x)^2] \\
&= E[f(x)^2] - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)^2] \\
&= E[f(x)^2] - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + E[\hat{f}(x)]^2 \\
&= E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + f(x)^2 - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)]^2 \\
&= E[(\hat{f}(x) - E[\hat{f}(x)])^2] + (f(x) - E[\hat{f}(x)])^2 \\
&= \mathrm{Var}(\hat{f}(x)) + \mathrm{Bias}^2(\hat{f}(x))
\end{aligned}
$$

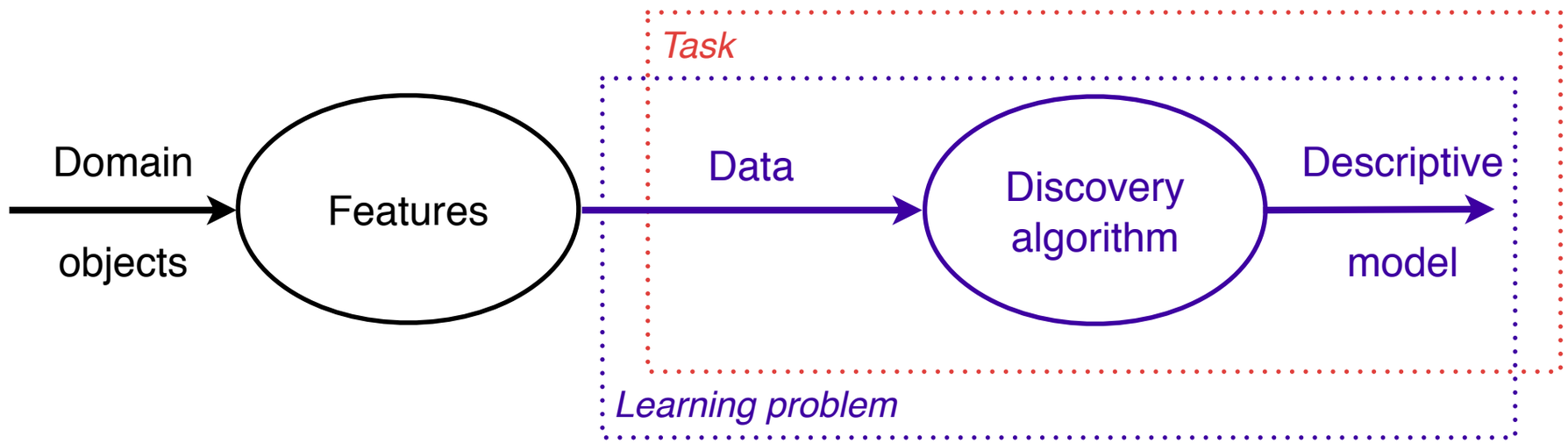
Unsupervised and descriptive learning



Machine learning tasks

	Predictive model	Descriptive model
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

Predictive vs **Descriptive** learning



In **descriptive** learning the task and learning problem coincide: we do not have a separate training set, and the task is to produce a descriptive model of the data.

Summarising:

- in *predictive* learning the task is to learn a model to label unseen examples.
- in *descriptive* learning the task is to learn a model describing the data.

Unsupervised learning

In unsupervised learning there is no labels associated to the data. The tasks involve figure out regularities in the data using only the data itself.

Machine learning tasks

	Predictive model	Descriptive model
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

Clustering

Clustering can be thought of as the task of making sense of data by finding homogeneous groups inside it.

Predictive clustering vs Descriptive clustering

Predictive clustering can be understood as the process of learning a new labelling function from unlabelled data. A 'clusterer' is then a mapping

$$\hat{q} : \mathscr{X} \rightarrow \mathscr{C}$$

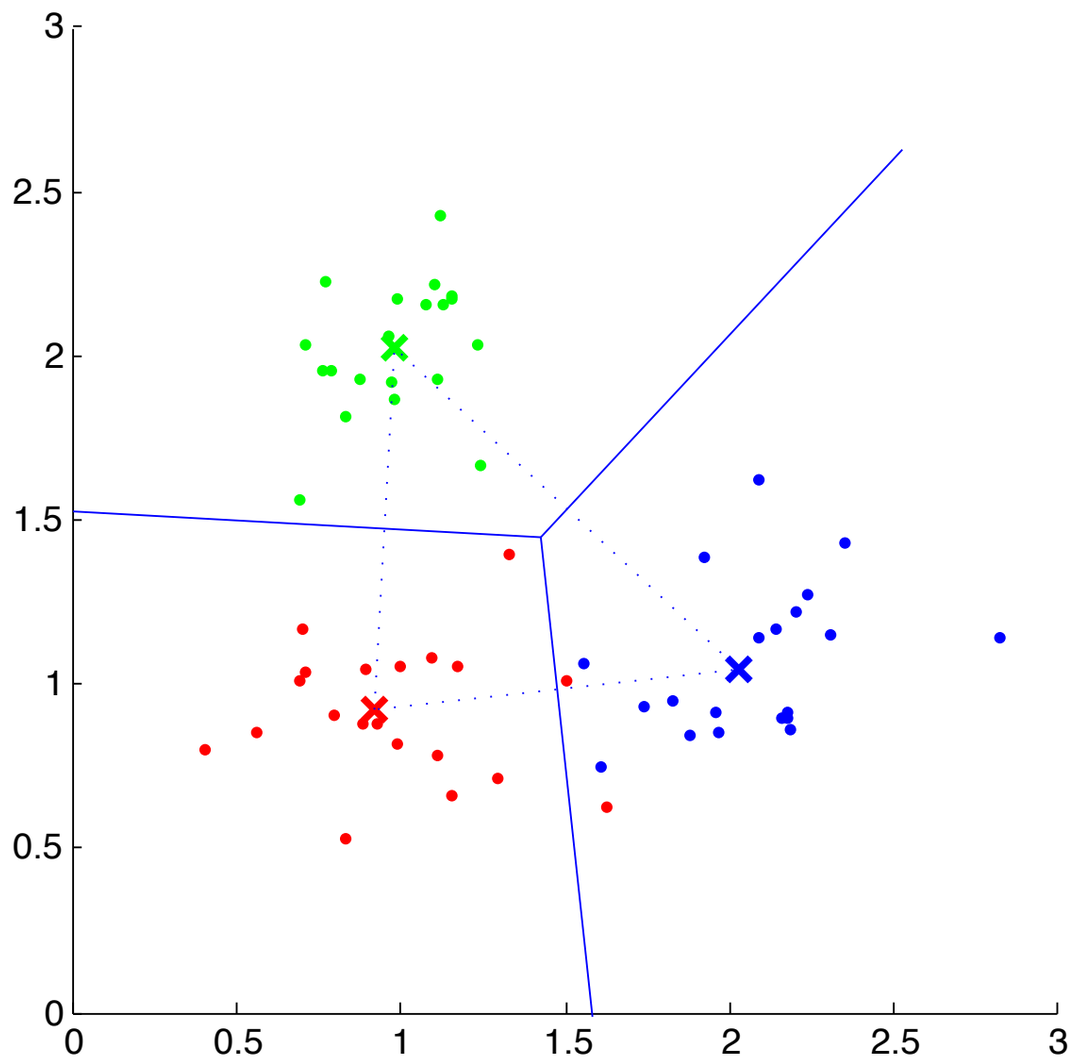
where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ is a set of new labels.

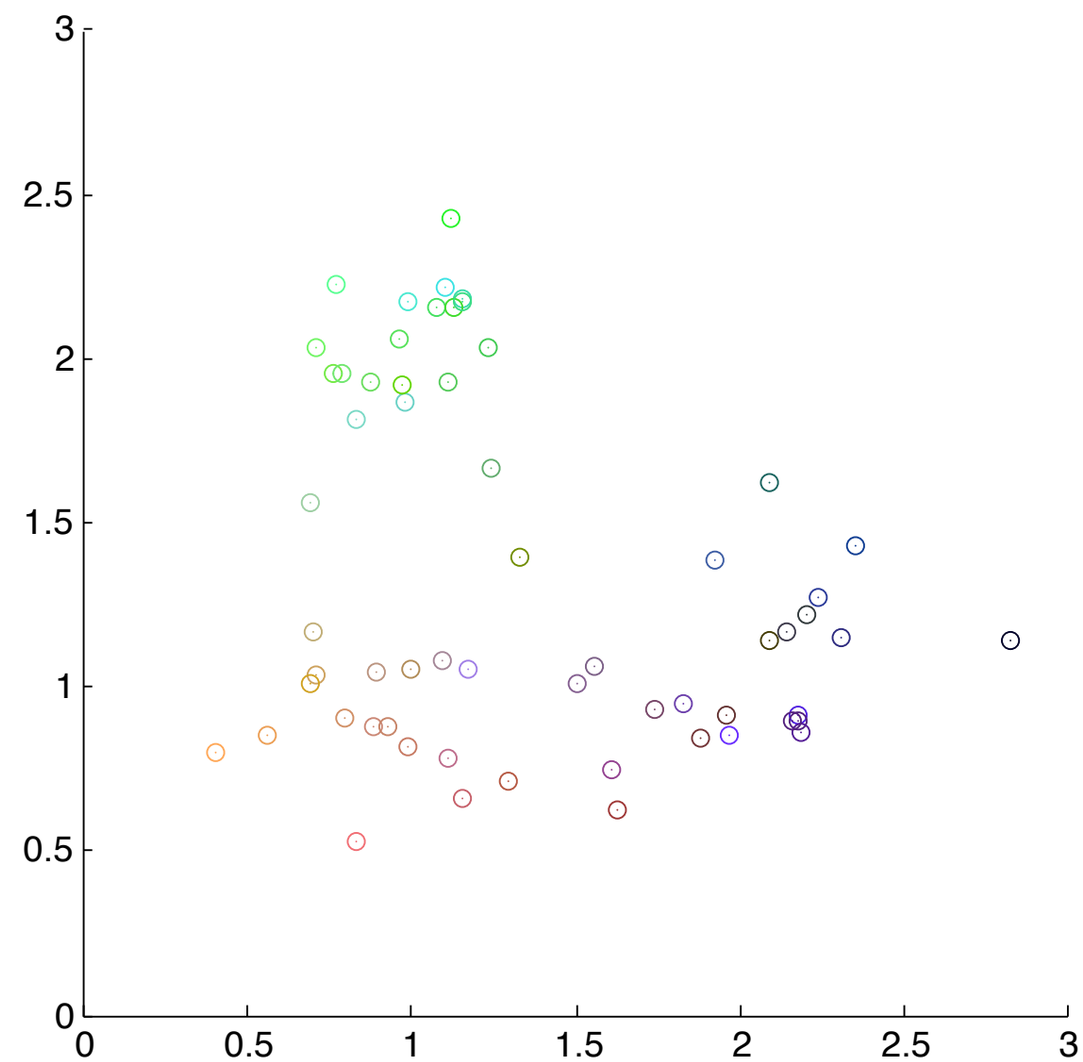
In **descriptive clustering** the model learned from the data would instead be a mapping

$$\hat{q} : D \rightarrow \mathscr{C}$$

where D is the used dataset.

Predictive clustering example





Machine learning tasks

	Predictive model	Descriptive model
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

Supervised subgroup-discovery

Task definition:

given a labelled dataset *{(x, l(x))}*, find a function:

$$\hat{g} : D \rightarrow \{true, false\}$$

such that $G = \{x \in D | \hat{g}(x) = true\}$ has a class distribution markedly different from the original population.

G is said to be the *extension* of the subgroup.

Example: in a sales dataset, find a subgroup of people whose propensity to buy a given product is markedly higher than that of the whole population.

Supervised sub-group discovery

In general sub-group discovery algorithms are guided by an evaluation measure. Many different ones exist, but most of them share the following characteristics:

- they prefer larger sub-groups
- they are usually symmetric, i.e., they report the same value for the sub-group and for its complement (what does this imply?)

Machine learning tasks

	Predictive model	Descriptive model
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

Association rules

Task description:

Given an unlabelled dataset D find a set of **rules** $\{ b \rightarrow h \}$ such that the ***itemset*** $b \cup h$ is frequent and that h is likely to hold whenever b holds.

Here b and h are sets of attribute/value pairs.

Association rules

To mine association rules, we first need to identify frequent itemsets (the ones with high support). Once we know that itemset $\{i_1, i_2, i_3\}$ is frequent, then all the following rules are frequent:

$i_1 \rightarrow i_2, i_3$	$i_2 \rightarrow i_1, i_3$	$i_3 \rightarrow i_1, i_2$
$i_1, i_2 \rightarrow i_3$	$i_1, i_3 \rightarrow i_2$	$i_2, i_3 \rightarrow i_1$

Among those rules, we want to select those satisfying some measure. For instance, if we use “confidence” then we would select those for which $\mathrm{supp}(b \cup h)/\mathrm{supp}(b)$ is high.

Example

If we set 0.6 as our support threshold (i.e., an itemset is frequent whenever it appears in 60% of the transactions). The following frequent itemsets can be extracted:

{Bread} (supp:0.8), {Milk} (supp: 0.6),
{Water} (supp:0.6), {Bread, Milk} (supp:0.6)

allowing one to generate the following rules:

Bread → Milk (conf: 0.6/0.8=0.75)
Milk → Bread (conf: 0.6/0.6=1)

id	Product
1	Bread
1	Milk
1	Water
2	Bread
2	Milk
3	Water
3	Bread
3	Ham
4	Water
4	Eggs
5	Bread
5	Milk

Problems

The brute force approach to this problem would generate all possible subsets of available items and calculate the support for each of them. This would require exponential time (why?).

Much more efficient algorithms exist. Main property that efficient algorithms exploit is the fact that if an itemset is unfrequent so are all its supersets.