MATH60026/MATH70026
Methods for Data Science
Lecture 2

Barbara Bravi, Imperial College London

Department of Mathematics, Academic year 2024-2025

IMPERIAL

k-Nearest Neighbours (*k*NN)

Simple idea: Create a prediction model using only the subset of samples in our training set that are closest to the new sample we want to predict.

Used both for regression and classification

*k*NN for Regression

Recall the data: $\{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, \quad y^{(i)}\}_{i=1}^{N}$

Predictors (continuous)

Outcomes (continuous)

*k*NN for Regression

Recall the data: $\{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, \quad y^{(i)}\}_{i=1}^{N}$

Predictors (continuous)

Outcomes (continuous)

Data to use for training

Typically, Euclidean distance

Algorithm: The kNN algorithm is implemented as follows. Given an input $\mathbf{x}^{\text{in}}$:

(1) Compute all distances between $\mathbf{x}^{\text{in}}$ and the samples in the training set:

$$\|\mathbf{x}^{\text{in}} - \mathbf{x}^{(i)}\| \qquad i = 1, \ldots, N^{\text{training}}$$

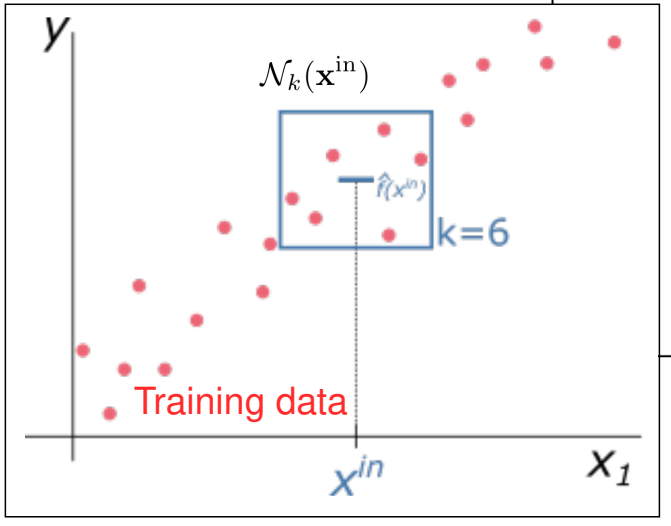
(2) Find the k nearest neighbours to $\mathbf{x}^{\text{in}}$ which define the neighbourhood:

$$\mathcal{N}_k(\mathbf{x}^{\text{in}})$$

Note that k is a choice, i.e., it is a hyperparameter to set via model selection.

(3) The simplest choice for predictor is just averaging over the values of the outcome variables for the samples in the neighbourhood:

$$\hat{f}_{\mathcal{N}(\mathbf{x}^{\text{in}})} = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x}^{\text{in}})} y^{(i)}$$



Final prediction: $\hat{y} = \hat{f}_{\mathcal{N}(\mathbf{x}^{\text{in}})}(\mathbf{x}^{\text{in}})$

Remark 1

*k*NN gives a *local* model:

$$\hat{y} = \hat{f}_{\mathcal{N}(\mathbf{x}^{\text{in}})}(\mathbf{x}^{\text{in}})$$

The predictor function depends on the input

Remark 1

*k*NN gives a *local* model:

$$\hat{y} = \hat{f}_{\mathcal{N}(\mathbf{x}^{\text{in}})}(\mathbf{x}^{\text{in}})$$

The predictor function depends on the input

While methods like Linear Regression give *global* models:

$$\hat{y} = \hat{f}_{\text{Lin}}(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}^\star$$

The predictor function is the same for all inputs

Remark 2

The distance function is a key ingredient, since it expresses the *similarity* between data points we want to leverage for prediction.

Remark 2

The distance function is a key ingredient, since it expresses the *similarity* between data points we want to leverage for prediction.

For continuous predictors, we typically take the Euclidean distance:

$$d(x^{(i)}, x^{(j)}) = \sqrt{\sum_{m=1}^{p} (x_m^{(i)} - x_m^{(j)})^2}$$

in which all features count in the same way.

Remark 2

The distance function is a key ingredient, since it expresses the *similarity* between data points we want to leverage for prediction.

For continuous predictors, we typically take the Euclidean distance:

$$d(x^{(i)}, x^{(j)}) = \sqrt{\sum_{m=1}^{p} (x_m^{(i)} - x_m^{(j)})^2}$$

in which all features count in the same way.

Features should be all relevant and non-redundant, which is difficult to have in high-dim. (i.e., high p) data.

Remark 2

The distance function is a key ingredient, since it expresses the *similarity* between data points we want to leverage for prediction.

For continuous predictors, we typically take the Euclidean distance:

$$d(x^{(i)}, x^{(j)}) = \sqrt{\sum_{m=1}^{p} (x_m^{(i)} - x_m^{(j)})^2}$$

in which all features count in the same way.

Features should be all relevant and non-redundant, which is difficult to have in high-dim. (i.e., high p) data.

High-magnitude features might dominate, so it is particularly important here to *standardise:*

standardised feature

$$z_m^{(i)} = \frac{x_m^{(i)} - \mu_m}{\sigma_m}$$

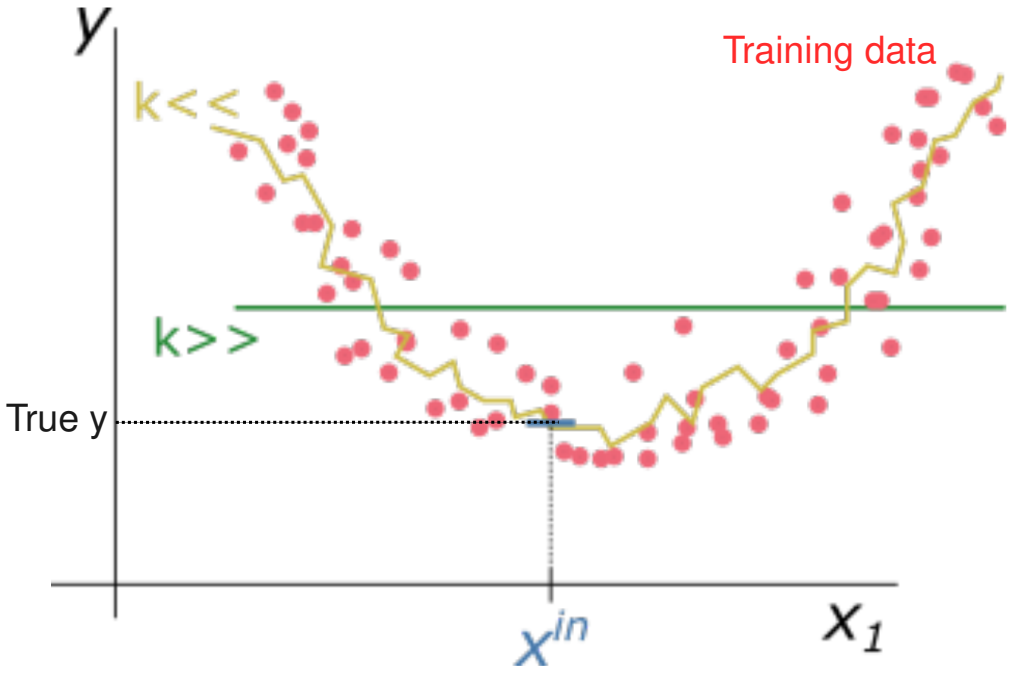
mean

standard deviation

so they are brought to vary on the same scale.

Remark 3

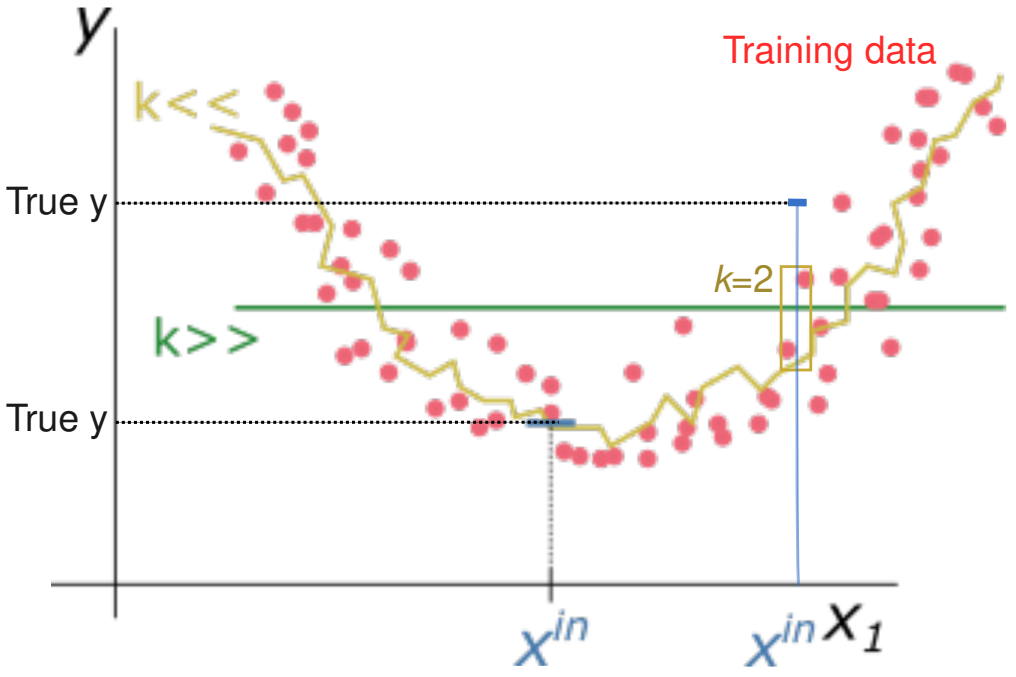
The choice of *k* is crucial, because it sets the size of the data neighbourhood, hence controlling *overfitting*.



If *k* is too big: it is not accurate, even for inputs in the training set.

Remark 3

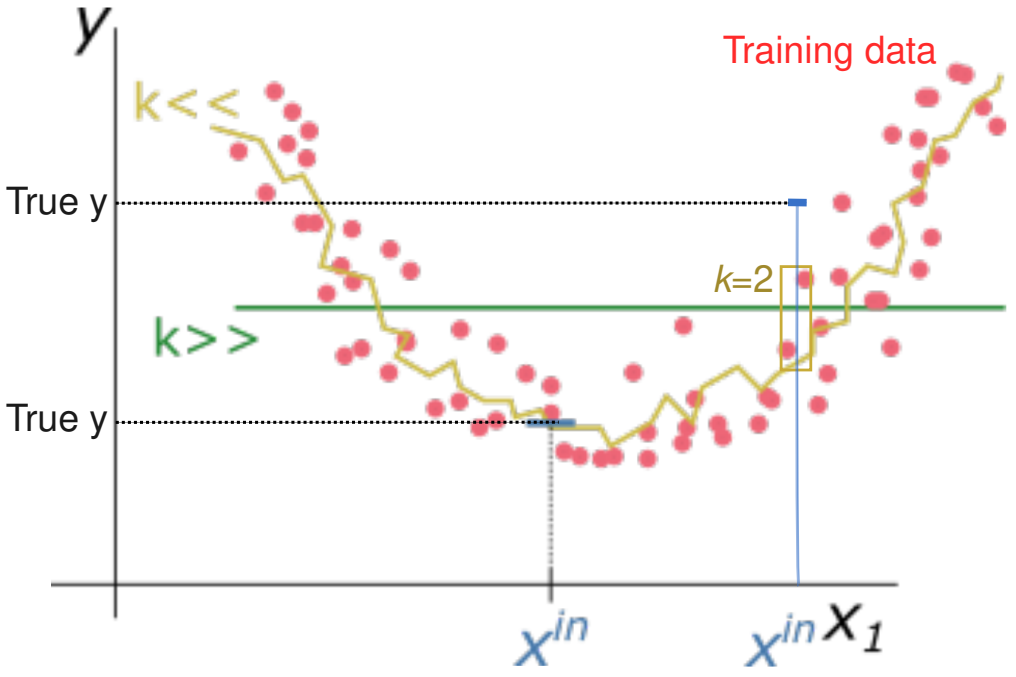
The choice of *k* is crucial, because it sets the size of the data neighbourhood, hence controlling *overfitting*.



If *k* is too small: it adapts too well to the training set (prediction too close to it).

Remark 3

The choice of *k* is crucial, because it sets the size of the data neighbourhood, hence controlling *overfitting*.



If *k* is too small: it adapts too well to the training set (prediction too close to it).

The choice of *k:* it's a hyperparameter to set by cross-validation

T-fold cross-validation

Applied generally across statistical and machine learning methods to set hyperparameter values achieving optimal *generalisability.*

T-fold cross-validation

Applied generally across statistical and machine learning methods to set hyperparameter values achieving optimal *generalisability.*

Divide into training and test sets:

size: N^{training} N^{test}

$$\mathcal{S} = \{\mathbf{x}^{(i)}, y^{(i)}\} \text{ for } i = 1, \ldots, N^{\text{training}}$$

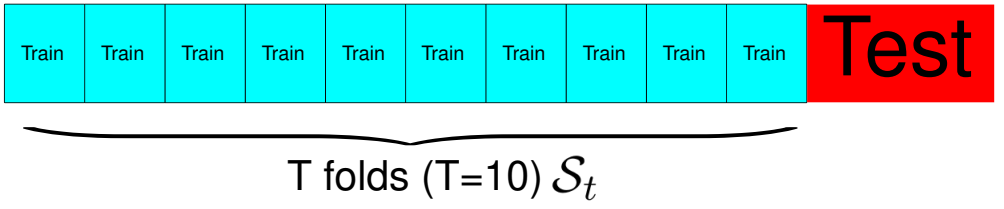
T-fold cross-validation

Applied generally across statistical and machine learning methods to set hyperparameter values achieving optimal *generalisability.*

Divide into training and test sets:

$$\mathcal{S} = \{\mathbf{x}^{(i)}, y^{(i)}\} \text{ for } i = 1, \ldots, N^{\text{training}}$$

Divide the training set into T subsets (folds):

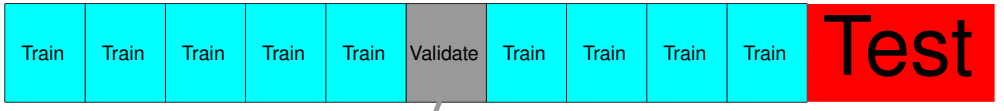


$$\mathcal{S} = \bigcup_{t=1}^{T} \mathcal{S}_t \text{ and } |\mathcal{S}_t| = \frac{|\mathcal{S}|}{T}, \text{ where } |\mathcal{S}| = N^{\text{training}}$$

T-fold cross-validation

Applied generally across statistical and machine learning methods to set hyperparameter values achieving optimal *generalisability.*

Use 9 folds for training, 1 fold for validation:



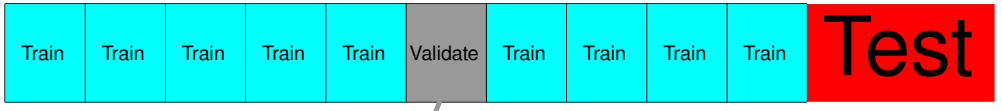
Evaluating the performance metric one the 1 fold for validation:

$$\mathrm{MSE}_t = \frac{1}{|\mathcal{S}_t|} \sum_{i \in \mathcal{S}_t} \left[\hat{f}_{\overline{\mathcal{S}_t}}(\mathbf{x}^{(i)}) - y^{(i)} \right]^2$$

T-fold cross-validation

Applied generally across statistical and machine learning methods to set hyperparameter values achieving optimal *generalisability.*

Use 9 folds for training, 1 fold for validation:



Evaluating the performance metric one the 1 fold for validation:

$$\mathrm{MSE}_t = \frac{1}{|\mathcal{S}_t|} \sum_{i \in \mathcal{S}_t} \left[\hat{f}_{\overline{\mathcal{S}_t}}(\mathbf{x}^{(i)}) - y^{(i)} \right]^2$$

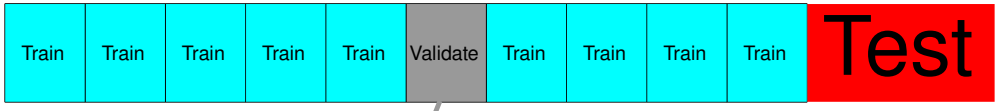
Repeat for all 10 choices of validation set and average the metric over folds:

$$\langle \mathrm{MSE} \rangle = \frac{1}{T} \sum_{t=1}^{T} \mathrm{MSE}_t$$

T-fold cross-validation

Applied generally across statistical and machine learning methods to set hyperparameter values achieving optimal *generalisability.*

Use 9 folds for training, 1 fold for validation:

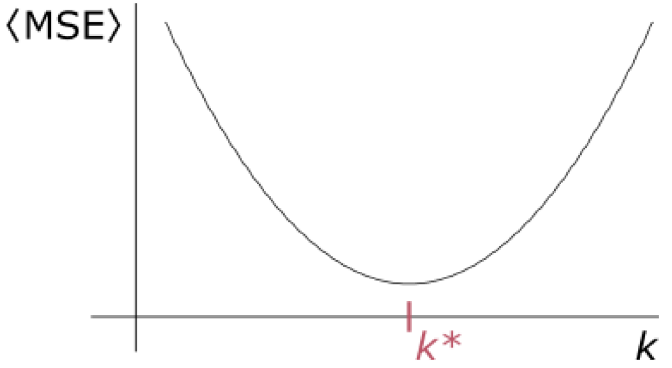


Evaluating the performance metric one the 1 fold for validation:

$$\mathrm{MSE}_t = \frac{1}{|\mathcal{S}_t|} \sum_{i \in \mathcal{S}_t} \left[\hat{f}_{\overline{\mathcal{S}_t}}(\mathbf{x}^{(i)}) - y^{(i)} \right]^2$$

Repeat for all 10 choices of validation set and average the metric over folds:

$$\langle \mathrm{MSE} \rangle = \frac{1}{T} \sum_{t=1}^{T} \mathrm{MSE}_t$$



Cartoon of the expected behaviour

*k*NN with discrete variables

e.g. binary:

$$\mathbf{x}^{(i)} \in \{0,1\}^p, \quad y^{(i)} \in \mathbb{R}$$

with different predictor variables, **one has to adjust the distance**.

E.g., for binary variables one can use the *Hamming Distance*:

$$d_H(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_{m=1}^{p} I(x_m^{(i)} \neq x_m^{(j)})$$

Indicator function

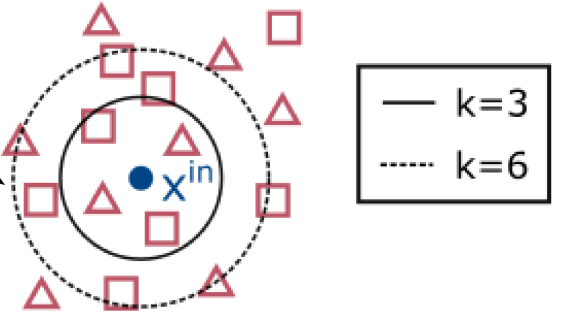
$$I(b) = \begin{cases} 1, & \text{if } b \text{ is true} \\ 0, & \text{if } b \text{ is false} \end{cases}$$

*k*NN with discrete outputs

i.e. prediction = assignment to a class (classification):

$$y^{(i)} \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$$

Class labels

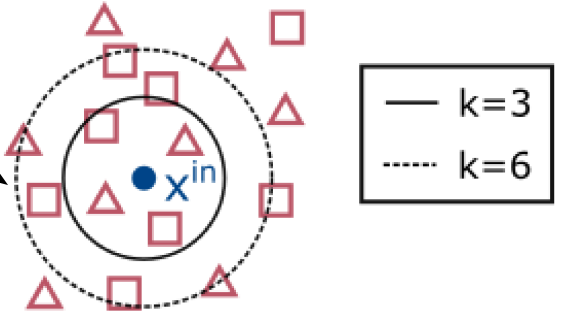


*k*NN with discrete outputs

i.e. prediction = assignment to a class (classification):

$$y^{(i)} \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$$

Class labels



Here, **one has to adjust the predictor function**.

(1) Choose k – same as before;

(2) For $\mathbf{x}^{\text{in}}$, at fixed k, find the neighbourhood $\mathcal{N}_k(\mathbf{x}^{\text{in}})$ – same as before;

(3) The final prediction is different here: $\hat{y} = \hat{f}(\mathbf{x}^{\text{in}}) \in \mathcal{C}_Q$ by the majority rule.

Assign the input point to the class to which the majority of training data in the neighbourhood belong.

Soft vs hard classification

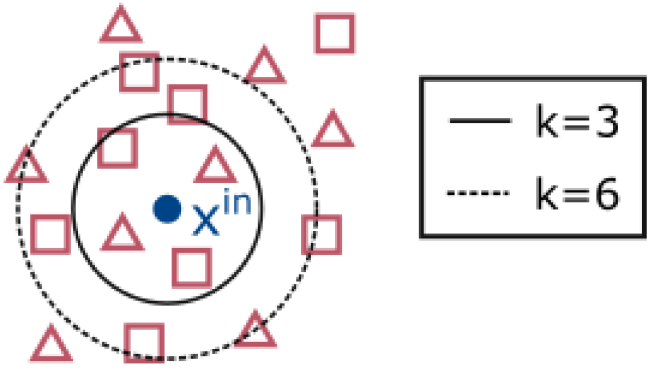
Majority rule: "hard" assignment to one class (hard classification).
It can be the result of a *discretisation* of a probabilistic assignment,
where we predict the probability of belonging to each class ("soft").

Soft vs hard classification

Majority rule: "hard" assignment to one class (hard classification).
It can be the result of a *discretisation* of a probabilistic assignment, where we predict the probability of belonging to each class ("soft").

Soft classification:

$$\hat{f}_{\text{Prob-kNN}}(\mathbf{x}^{\text{in}}) = \frac{1}{k} \begin{pmatrix} \sum_{i \in \mathcal{N}_k(\mathbf{x}^{\text{in}})} I\left(y^{(i)} \in c_1\right) \\ \vdots \\ \sum_{i \in \mathcal{N}_k(\mathbf{x}^{\text{in}})} I\left(y^{(i)} \in c_Q\right) \end{pmatrix}$$



$$= \frac{1}{k} \begin{pmatrix} \text{number of neighbour points in } c_1 \\ \vdots \\ \text{number of neighbour points in } c_Q \end{pmatrix}$$

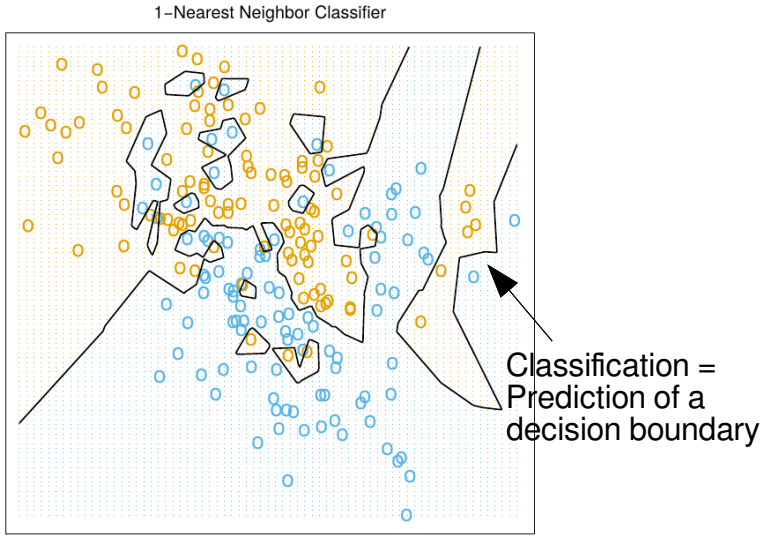
It preserves information on the uncertainty of prediction

Hard classification:

$$\hat{f}_{\text{MajRule-kNN}}(\mathbf{x}^{\text{in}}) = \underset{q}{\text{argmax}}\, \hat{f}_{\text{Prob-kNN}}(\mathbf{x}^{\text{in}})$$

k-Nearest Neighbours classifiers:

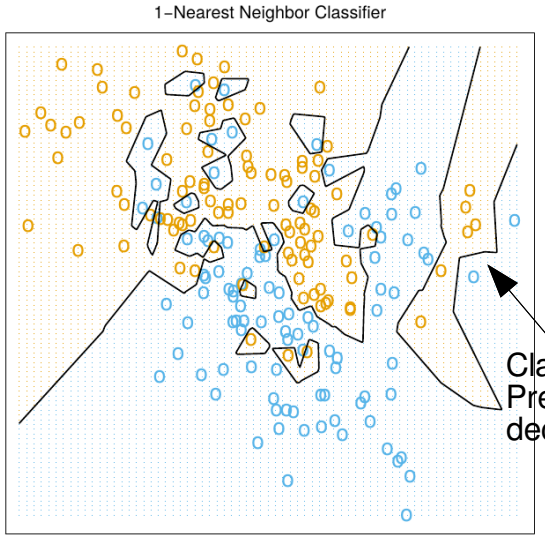
Example with binary classification
+ two continuous predictors
(distance = Euclidean distance)



Simulated data, see Hastie, Tibshirani, Friedman,
Elements of statistical learning, chap. 2, 13

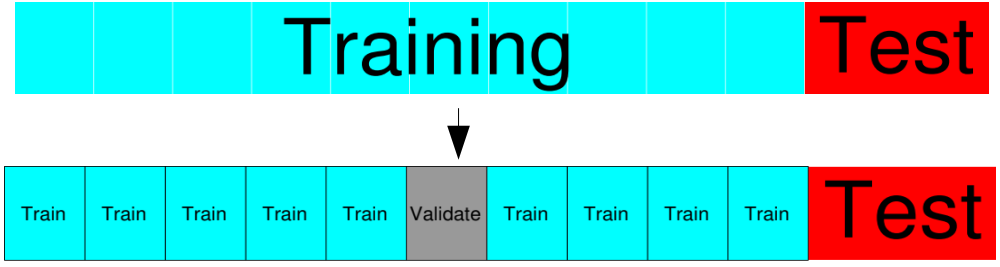
k-Nearest Neighbours classifiers:

Example with binary classification
+ two continuous predictors
(distance = Euclidean distance)



Simulated data, see Hastie, Tibshirani, Friedman,
Elements of statistical learning, chap. 2, 13

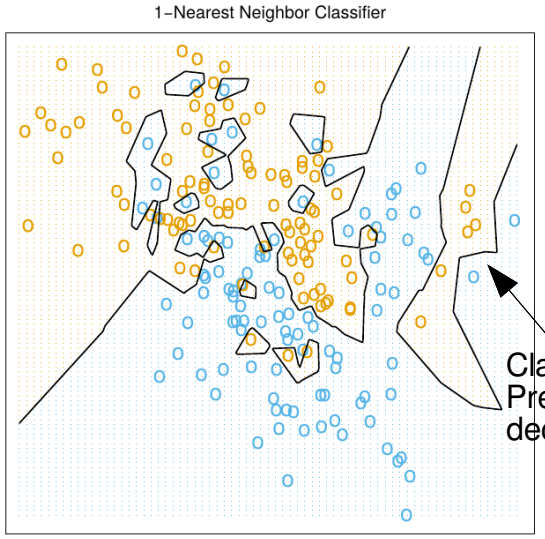
k is set by 10-fold cross-validation:



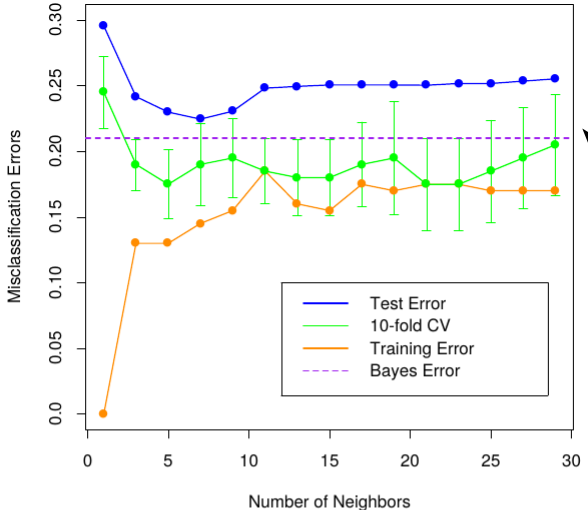
$$\mathcal{S} = \bigcup_{t=1}^{T} \mathcal{S}_t$$

k-Nearest Neighbours classifiers:

Example with binary classification
+ two continuous predictors
(distance = Euclidean distance)



Classification =
Prediction of a
decision boundary



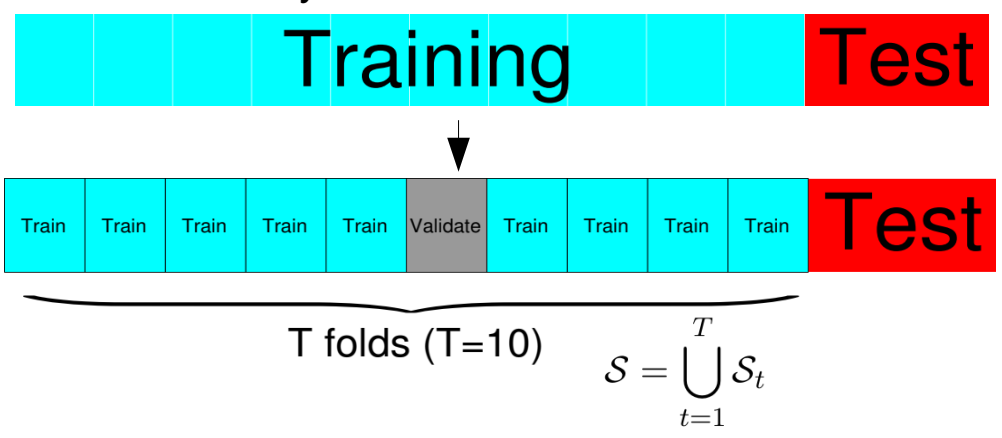
Misclassification error (ME) = 1 - accuracy

Average ME across folds (bars = standard dev):

$$\langle \mathrm{ME} \rangle = \frac{1}{T} \sum_{t=1}^{T} \mathrm{ME}_t$$

on the validation fold!

k is set by 10-fold cross-validation:

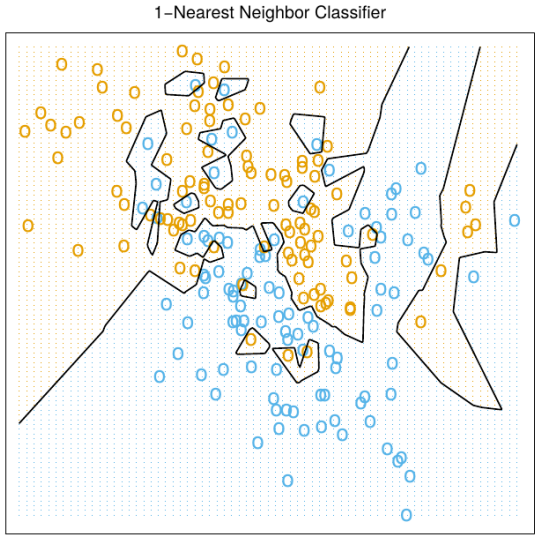


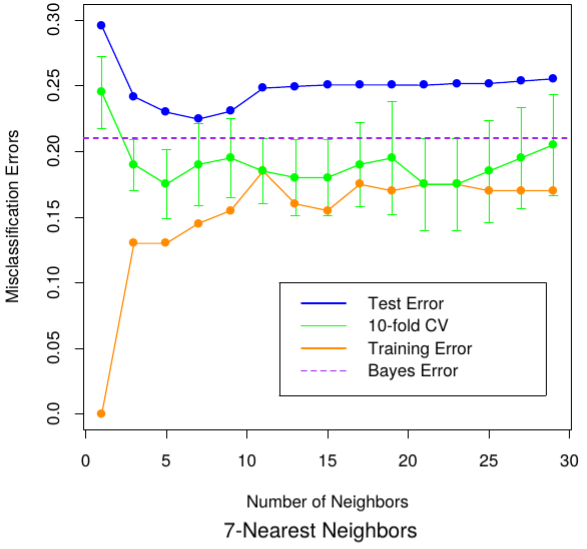
$$\mathcal{S} = \bigcup_{t=1}^{T} \mathcal{S}_t$$

Simulated data, see Hastie, Tibshirani, Friedman,
Elements of statistical learning, chap. 2, 13

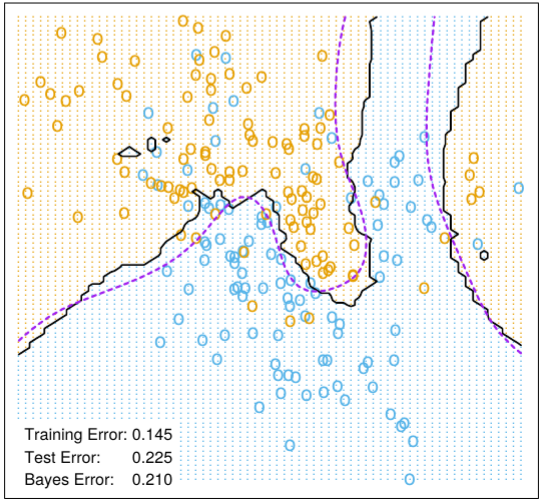
k-Nearest Neighbours classifiers:

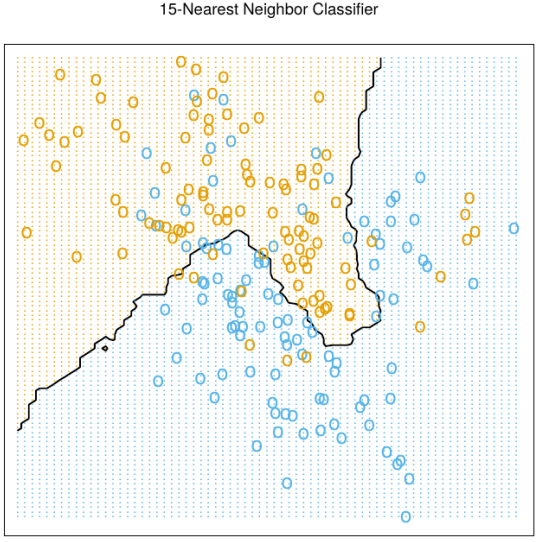
Example with binary classification
+ two continuous predictors
(distance = Euclidean distance)





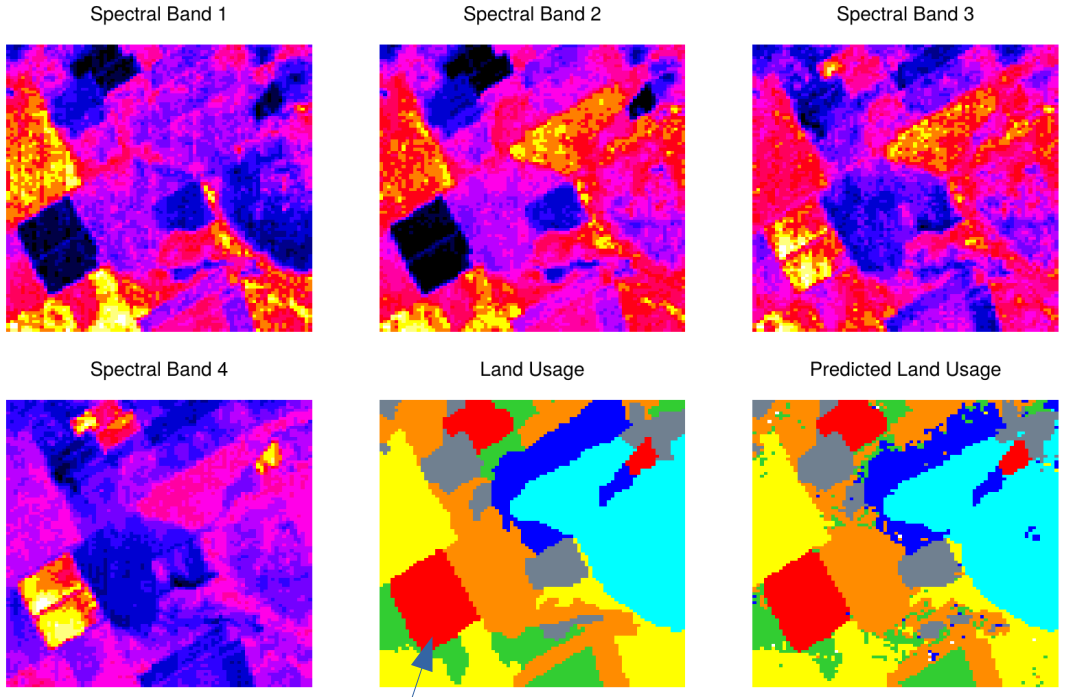
Misclassification error (ME) = 1 - accuracy





Simulated data, see Hastie, Tibshirani, Friedman,
Elements of statistical learning, chap. 2, 13

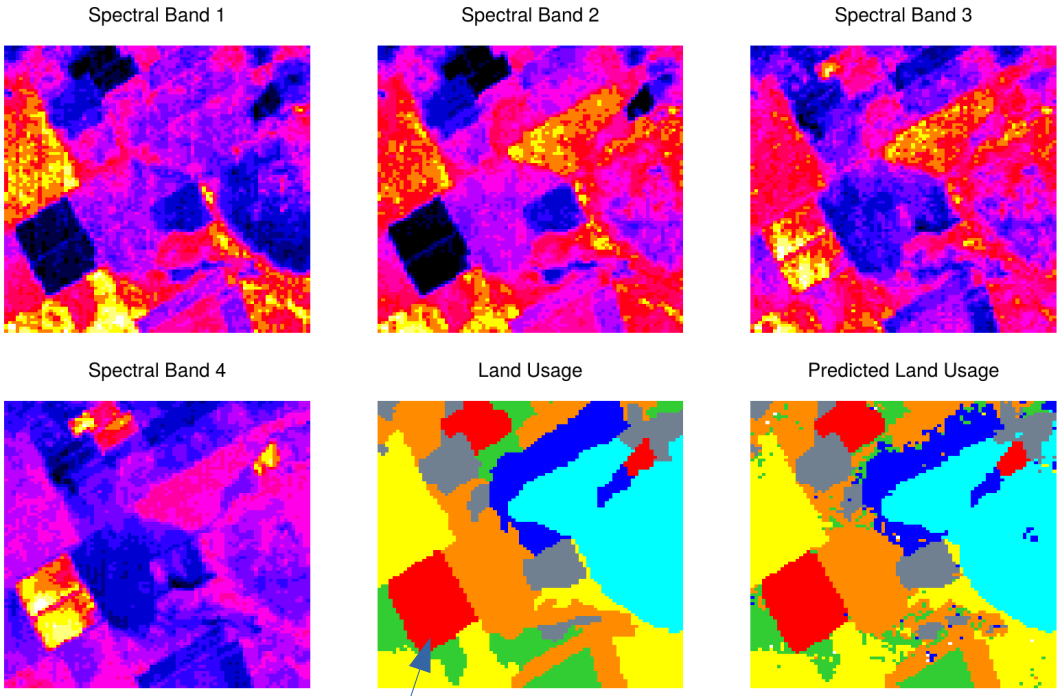
k-Nearest Neighbours to classify land usage



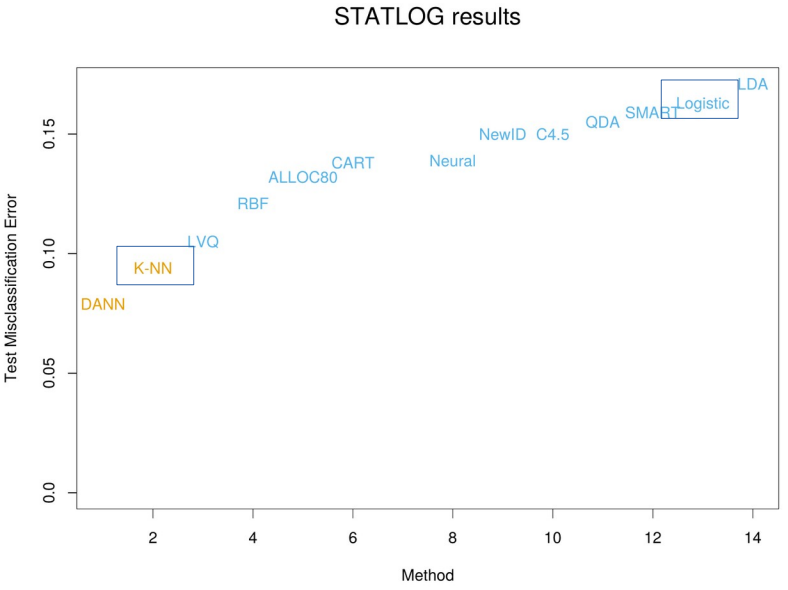
Each pixel belongs to 1 of 7 classes: *red soil, cotton, vegetation mixture* ...

LANDSAT dataset, analyzed in STATLOG benchmark project (King et al. 1995)
see also Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 13

k-Nearest Neighbours to classify land usage



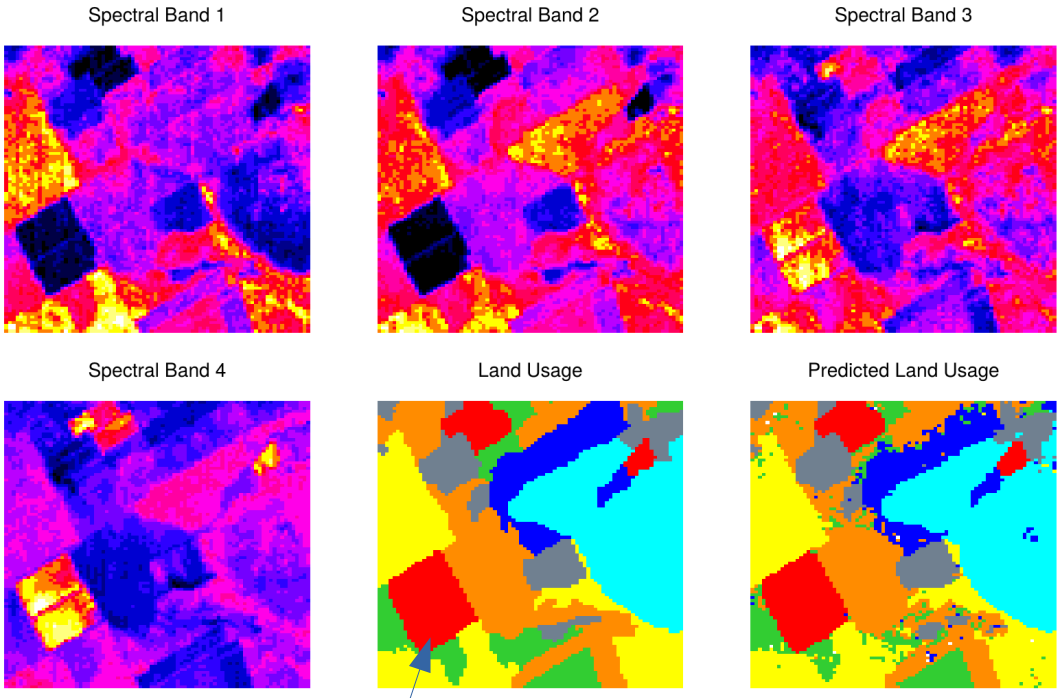
Each pixel belongs to 1 of 7 classes: *red soil, cotton, vegetation mixture* ...



Misclassification error = 1 - accuracy

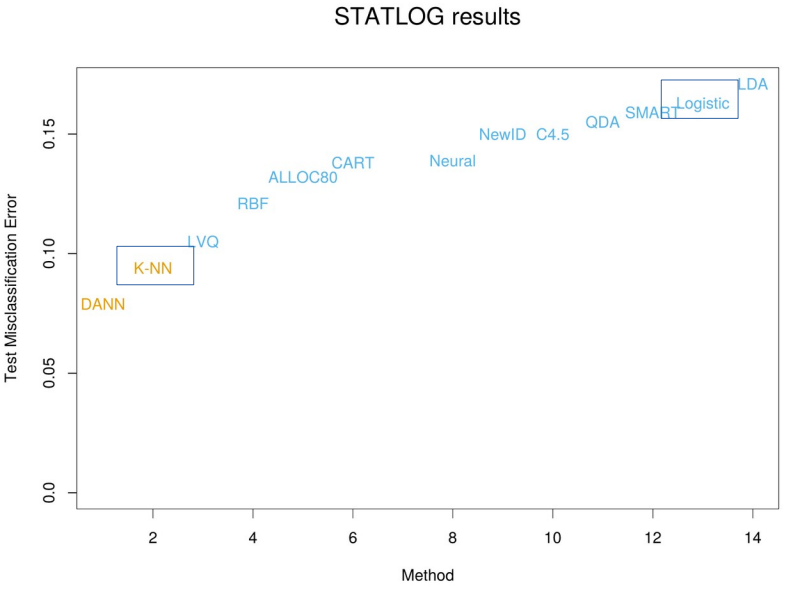
LANDSAT dataset, analyzed in STATLOG benchmark project (King et al. 1995)
see also Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 13

k-Nearest Neighbours to classify land usage



Each pixel belongs to 1 of 7 classes: *red soil, cotton, vegetation mixture* ...

LANDSAT dataset, analyzed in STATLOG benchmark project (King et al. 1995)
see also Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 13



Misclassification error = 1 - accuracy

<u>*k*-NN simple but powerful method</u>

- Only 1 hyper-parameter (*k*) to calibrate

- Baseline prediction accuracy purely based on similarity (for comparison of more sophisticated ML methods)

Logistic Regression

One of the simplest classifiers, as is a *linear* classifier (and global), in the sense that it predicts a **linear decision boundary**.

Logistic Regression

One of the simplest classifiers, as is a *linear* classifier (and global), in the sense that it predicts a **linear decision boundary**.

We study it in the binary (2-class) case:

$$\mathbf{x}^{(i)} \in \mathbb{R}^p; \quad y^{(i)} \in \{0,1\} \qquad i = 1, \ldots, N$$

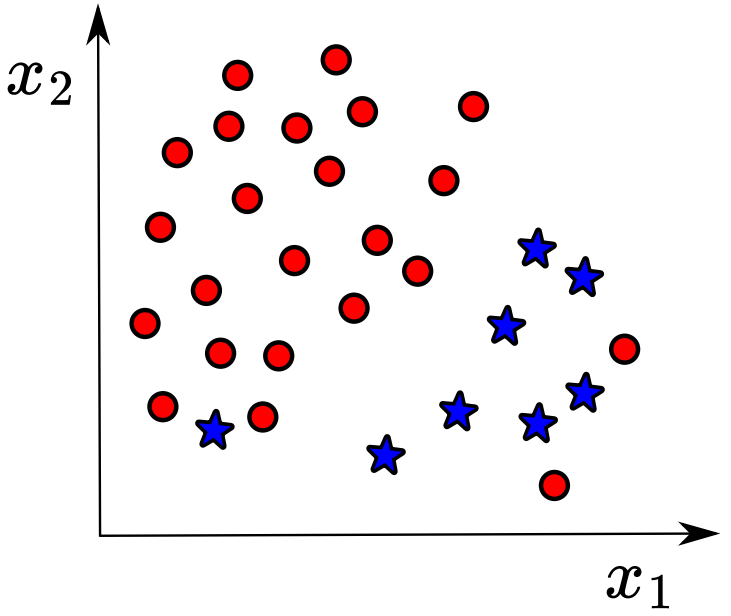
Logistic regression predicts the probability of belonging to each class by setting:

Linear combination

Logit function (log-odds)

$$\mathbf{x}^T \boldsymbol{\beta} = \log \frac{P(y=1|\mathbf{x})}{1 - P(y=1|\mathbf{x})}$$

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 & \beta_1 & \cdots & \beta_p \end{pmatrix}^T \qquad \mathbf{x}^T = (1, x_1, \ldots, x_p)$$



Logistic Regression

One of the simplest classifiers, as is a *linear* classifier (and global), in the sense that it predicts a **linear decision boundary**.

We study it in the binary (2-class) case:

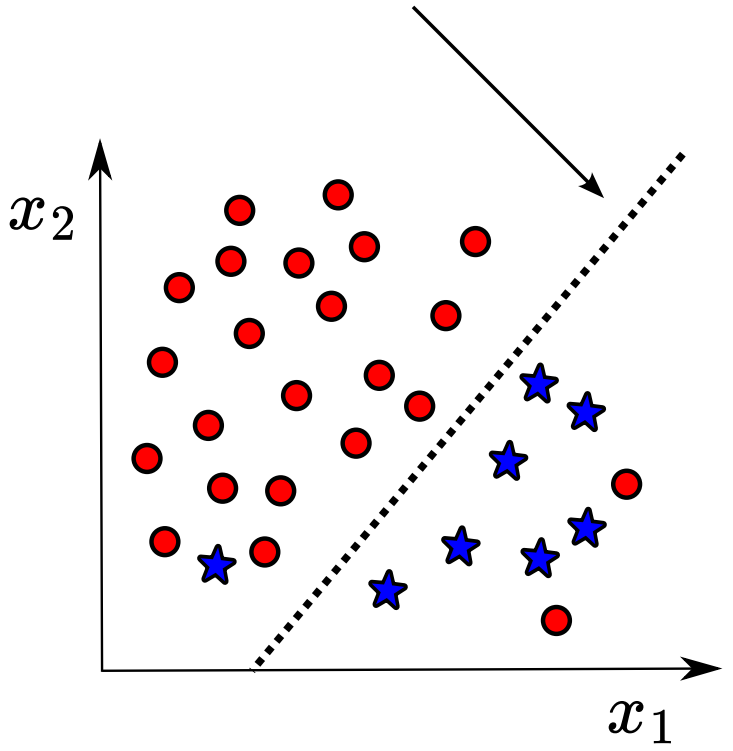
$$\mathbf{x}^{(i)} \in \mathbb{R}^p; \quad y^{(i)} \in \{0,1\} \qquad i = 1, \ldots, N$$

Logistic regression predicts the probability of belonging to each class by setting:

Linear combination

Logit function (log-odds)

$$\mathbf{x}^T \boldsymbol{\beta} = \log \frac{P(y=1|\mathbf{x})}{1 - P(y=1|\mathbf{x})}$$



$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 & \beta_1 & \cdots & \beta_p \end{pmatrix}^T \qquad \mathbf{x}^T = (1, x_1, \ldots, x_p)$$

Hence, the decision boundary is a hyperplane, typically: $\mathbf{x}^T \boldsymbol{\beta} = 0$

Logistic Regression

It is thus a *probabilistic* classifier, and its probabilistic assumption is:

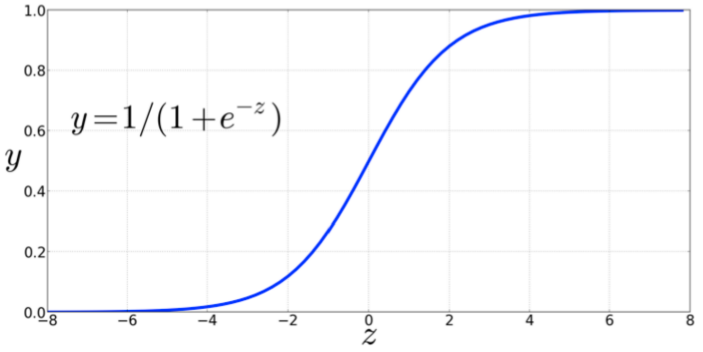
$$P(Y = y \mid \mathbf{x}, \boldsymbol{\beta}) = \left(h_{\boldsymbol{\beta}}(\mathbf{x})\right)^{y} \left(1 - h_{\boldsymbol{\beta}}(\mathbf{x})\right)^{1-y}$$

i.e., the probability of the output conditional on the data features is **Bernoulli**, with:

Probability of success

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}^T \boldsymbol{\beta}}} = h\left(\mathbf{x}^T \boldsymbol{\beta}\right) =: h_{\boldsymbol{\beta}}(\mathbf{x})$$

Logistic function



Depends on beta parameters to learn!

Learning the parameters

Thanks to this probabilistic formulation, I can write a likelihood:

$$P\left(\{y^{(i)}\} \mid \{x^{(i)}\}, \boldsymbol{\beta}\right) = \prod_{i=1}^{N} \left(h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)^{1-y^{(i)}}$$

and I can appeal to the **maximum likelihood** criterion: learn the parameters that maximise the probability of the data under the model specified by those parameters.

Learning the parameters

Thanks to this probabilistic formulation, I can write a likelihood:

$$P\left(\{y^{(i)}\} \mid \{x^{(i)}\}, \boldsymbol{\beta}\right) = \prod_{i=1}^{N} \left(h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)^{1-y^{(i)}}$$

and I can appeal to the **maximum likelihood** criterion: learn the parameters that maximise the probability of the data under the model specified by those parameters.

It's convenient to work with the log-likelihood:

$$\mathcal{L} = \sum_{i=1}^{N} y^{(i)} \log h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)$$

To maximise the log-likelihood (see lecture notes):

$$\nabla_{\boldsymbol{\beta}} \mathcal{L}|_{\boldsymbol{\beta}^{\star}_{\log}} = \sum_{i=1}^{N} \left(y^{(i)} - h_{\boldsymbol{\beta}^{\star}_{\log}}(\mathbf{x}^{(i)})\right) \mathbf{x}^{(i)} = \mathbf{0}$$

Learning the parameters

Thanks to this probabilistic formulation, I can write a likelihood:

$$P\left(\{y^{(i)}\} \mid \{x^{(i)}\}, \boldsymbol{\beta}\right) = \prod_{i=1}^{N} \left(h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)^{1-y^{(i)}}$$

and I can appeal to the **maximum likelihood** criterion: learn the parameters that maximise the probability of the data under the model specified by those parameters.

It's convenient to work with the log-likelihood:

$$\mathcal{L} = \sum_{i=1}^{N} y^{(i)} \log h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - h_{\boldsymbol{\beta}}(\mathbf{x}^{(i)})\right)$$

To maximise the log-likelihood (see lecture notes):

$$\nabla_{\boldsymbol{\beta}} \mathcal{L}|_{\boldsymbol{\beta}^{\star}_{\log}} = \sum_{i=1}^{N} \left(y^{(i)} - h_{\boldsymbol{\beta}^{\star}_{\log}}(\mathbf{x}^{(i)})\right) \mathbf{x}^{(i)} = \mathbf{0}$$

Set of p+1 equations to solve, but no analytical formula

Yet, $\mathcal{L}$ is **concave**, thus *gradient ascent* leads to a global maximum $\boldsymbol{\beta}^{\star}_{\log}$

Yet, $\mathcal{L}$ is **concave**, thus *gradient ascent* leads to a global maximum $\boldsymbol{\beta}^{\star}_{\log}$

Once the parameters are learnt, passing through the **logistic function** gives:

$$P(y = 1|\mathbf{x}^{\text{in}}) = h_{\boldsymbol{\beta}^{\star}_{\log}}(\mathbf{x}^{\text{in}}) \in [0, 1]$$

that is, a probabilistic prediction of belonging to e.g. the class with label 1 (*soft* classification).

Yet, $\mathcal{L}$ is **concave**, thus *gradient ascent* leads to a global maximum $\boldsymbol{\beta}^{\star}_{\log}$

Once the parameters are learnt, passing through the **logistic function** gives:

$$P(y = 1|\mathbf{x}^{\text{in}}) = h_{\boldsymbol{\beta}^{\star}_{\log}}(\mathbf{x}^{\text{in}}) \in [0, 1]$$

that is, a probabilistic prediction of belonging to e.g. the class with label 1 (*soft* classification).

To obtain a hard classification, one fixes a threshold τ such that:

$$P(y = 1 \mid \mathbf{x}^{\text{in}}) > \tau \implies \mathbf{x}^{\text{in}} \in \{1\}$$

$\tau = \frac{1}{2}$ clearly gives the argmax discretisation seen for *k*NN.

Yet, $\mathcal{L}$ is **concave**, thus *gradient ascent* leads to a global maximum $\boldsymbol{\beta}^{\star}_{\log}$

Once the parameters are learnt, passing through the **logistic function** gives:

$$P(y = 1|\mathbf{x}^{\text{in}}) = h_{\boldsymbol{\beta}^{\star}_{\log}}(\mathbf{x}^{\text{in}}) \in [0, 1]$$

that is, a probabilistic prediction of belonging to e.g. the class with label 1 (*soft* classification).

To obtain a hard classification, one fixes a threshold τ such that:

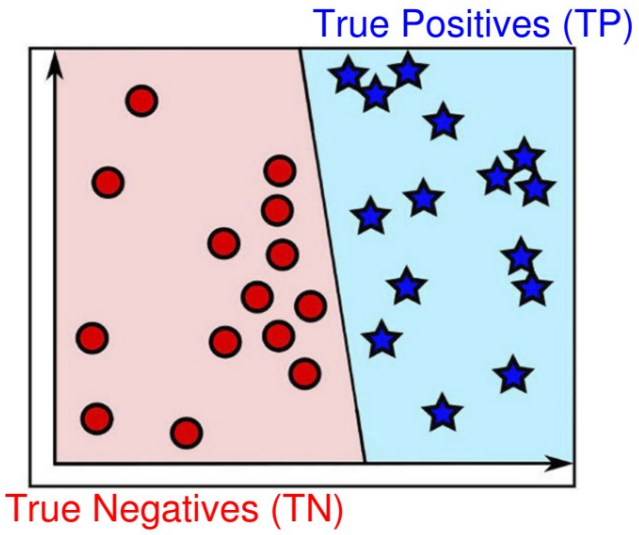
$$P(y = 1 \mid \mathbf{x}^{\text{in}}) > \tau \implies \mathbf{x}^{\text{in}} \in \{1\}$$

$\tau = \frac{1}{2}$ clearly gives the argmax discretisation seen for *k*NN.

How does one evaluate the performance of (hard or soft) classification?

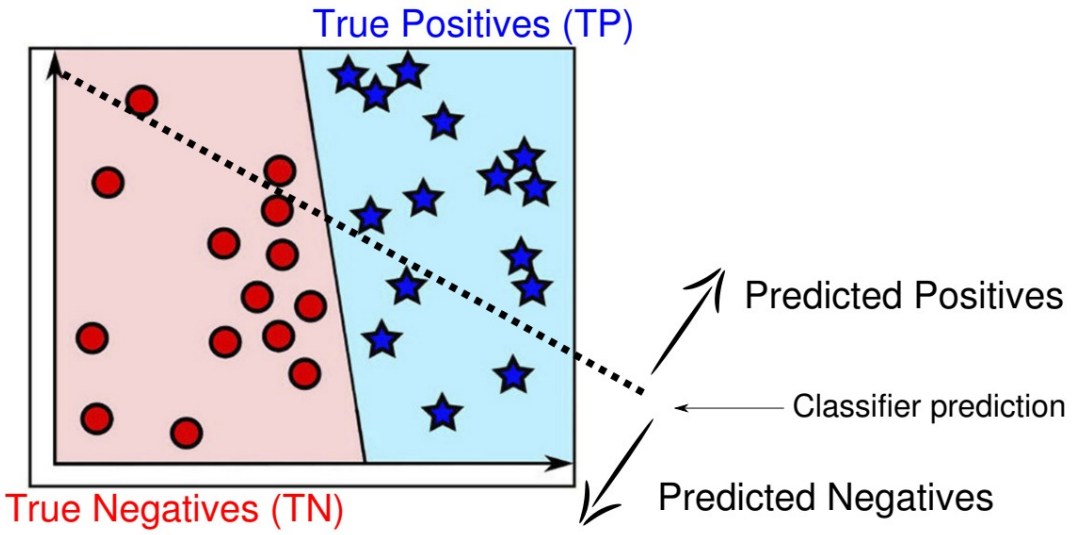
Classification performance

Cartoon example with 2 classes:



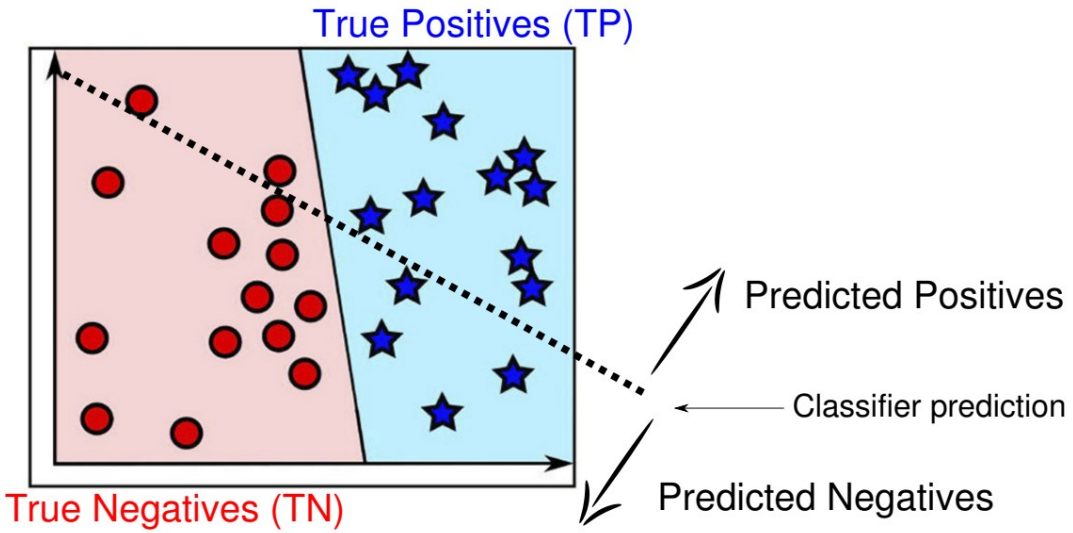
Classification performance

Cartoon example with 2 classes:



Classification performance

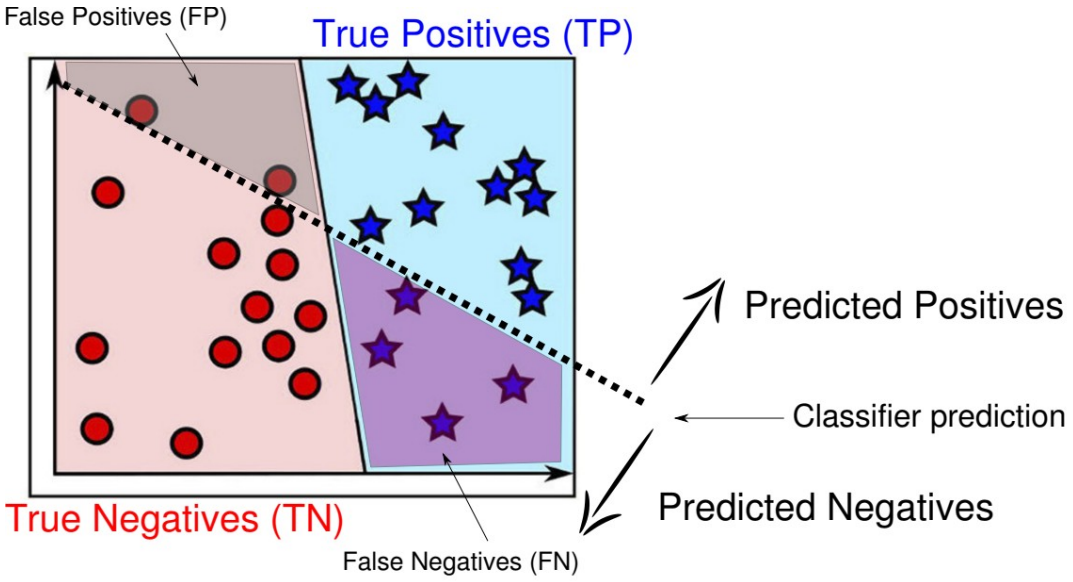
Cartoon example with 2 classes:

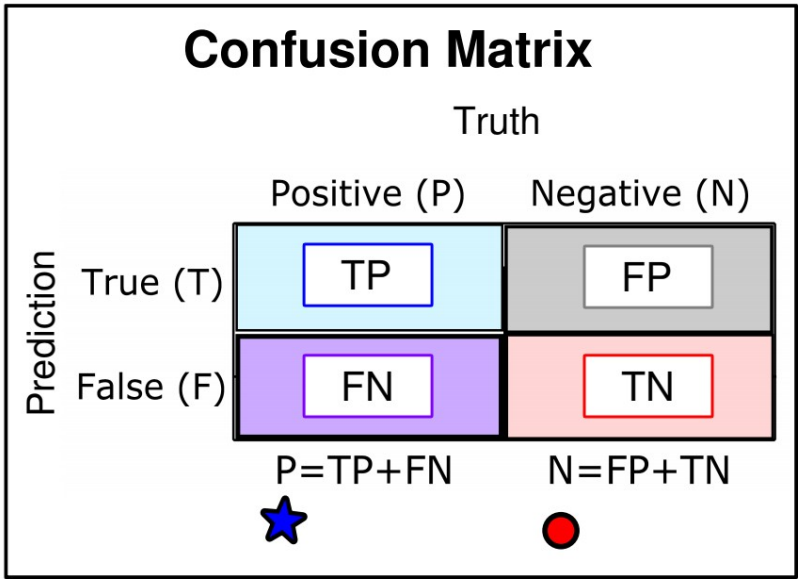


For example, in logistic regression, if $P(x \in$ ★ $) > \tau$ then x is predicted as a Positive

Classification performance

Cartoon example with 2 classes:





TP = True Positives; FP = False Positives
FN = False Negatives; TN = True Negatives

Classification performance

Metrics of classification performance:

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{TN}{N} \equiv TNR$$

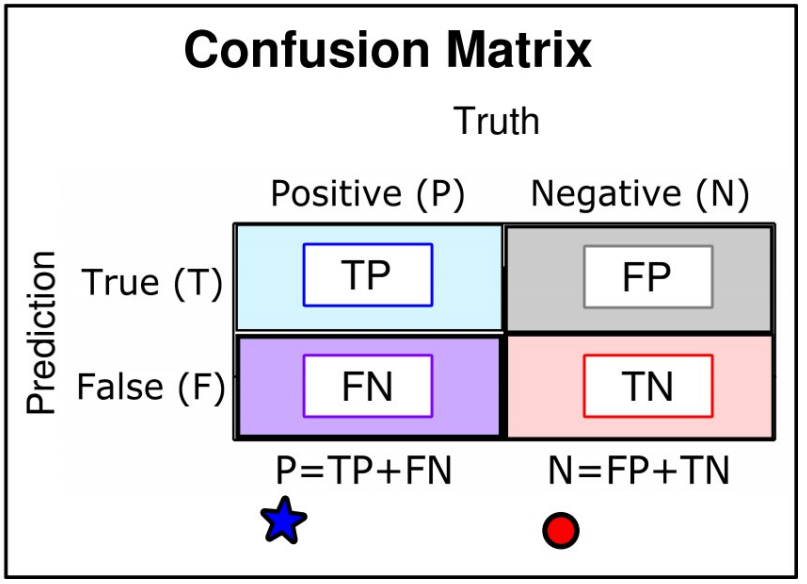
True Negative Rate

$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{TP}{P} \equiv TPR$$

True Positive Rate

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$



Classification performance

Metrics of classification performance:

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{TN}{N} \equiv TNR$$

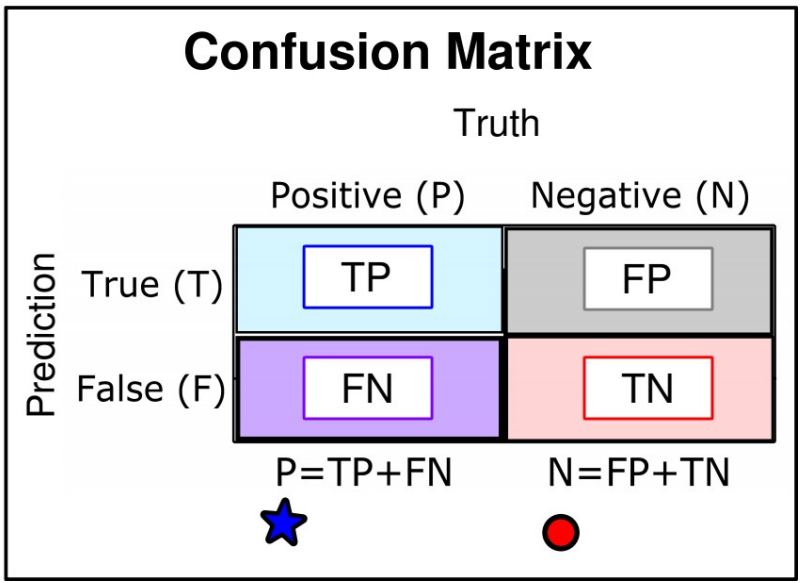
True Negative Rate

$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{TP}{P} \equiv TPR$$

True Positive Rate

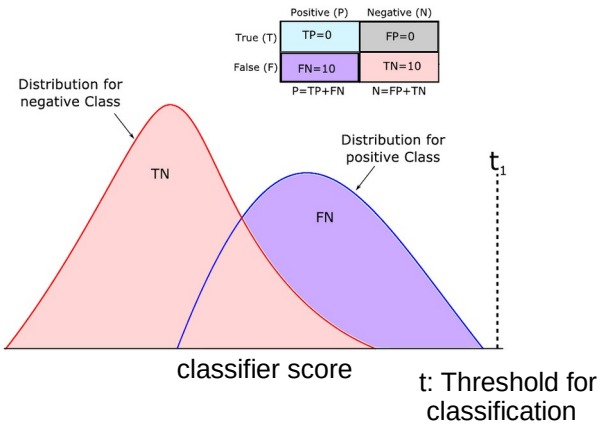
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$



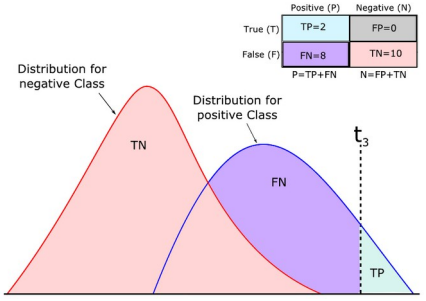
But we can also assess the ability of the classifier to discriminate classes more globally,
without fixing one defined threshold

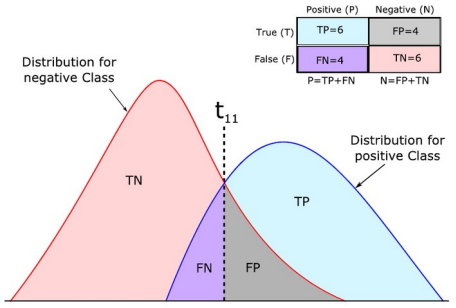
Receiver Operating Characteristics:

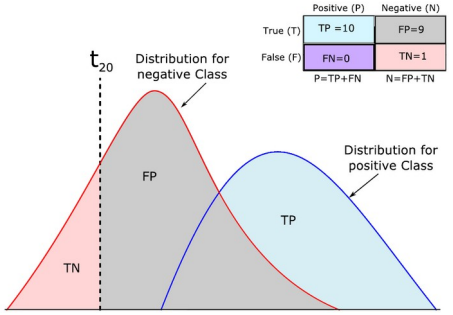


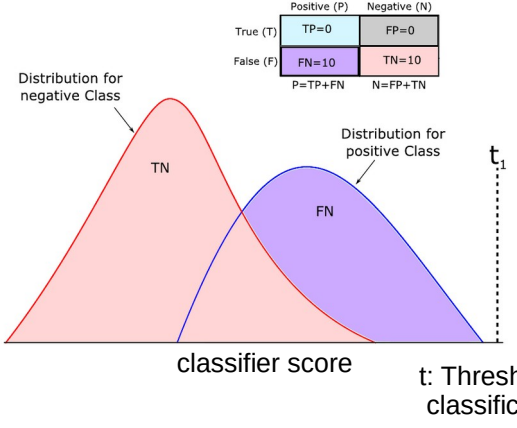
Receiver Operating Characteristics:

We progressively increase the threshold for classification





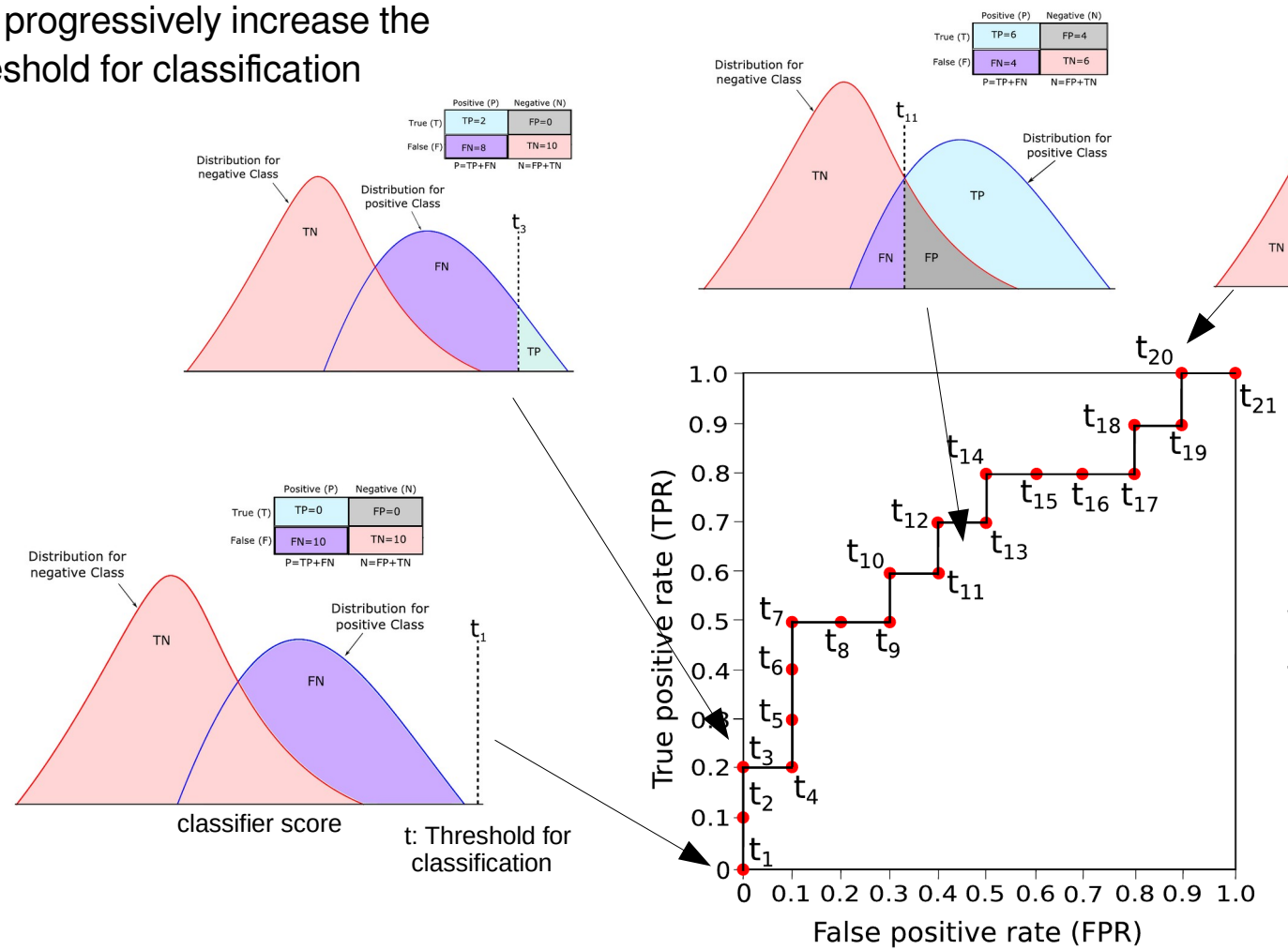




t: Threshold for classification

Receiver Operating Characteristics:

We progressively increase the threshold for classification

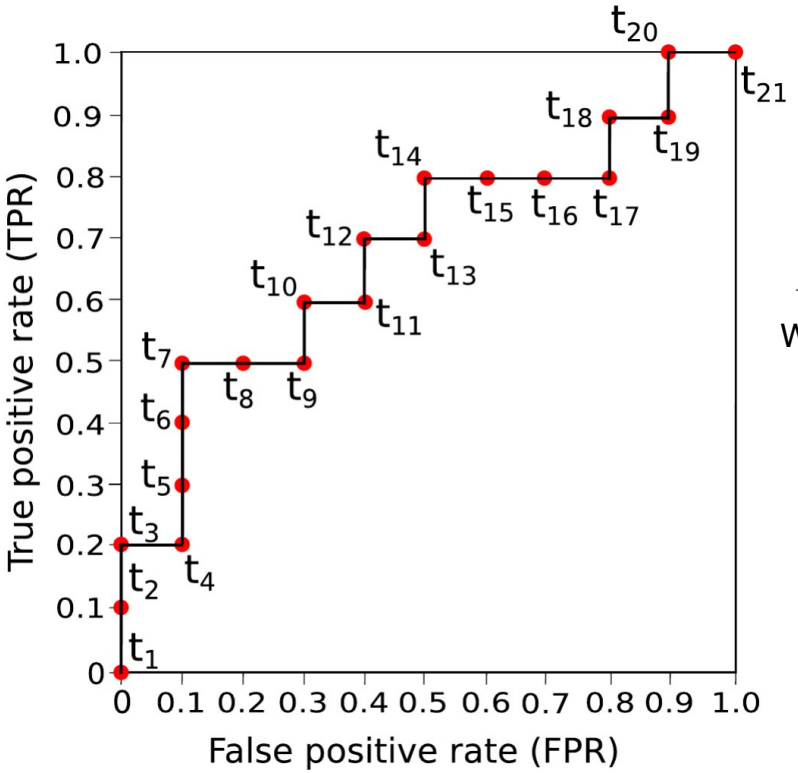


We plot TPR vs FPR varying the threshold for classification

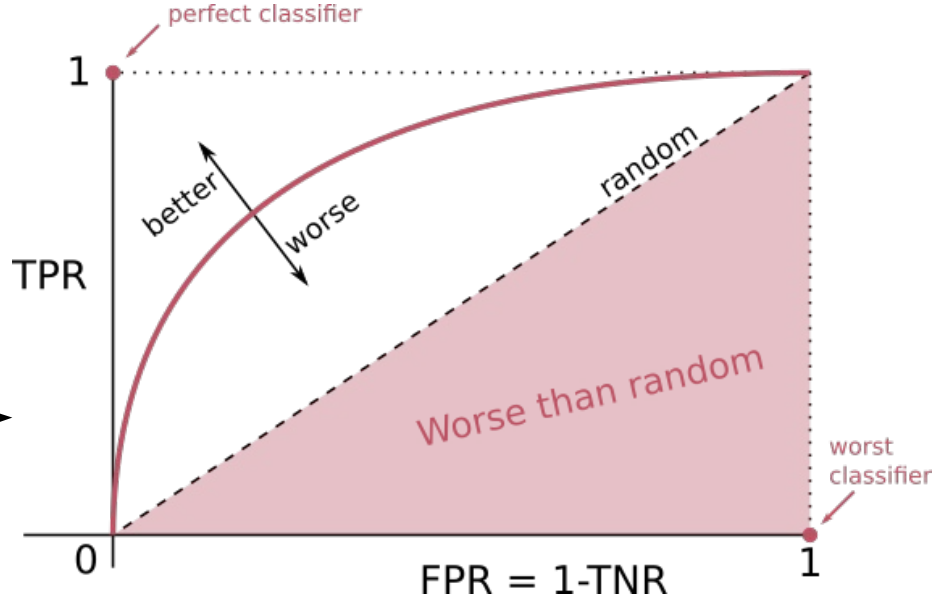
$$TPR = \frac{TP}{P} \qquad FPR = \frac{FP}{N}$$

Figures adapted from Tharwat 2018

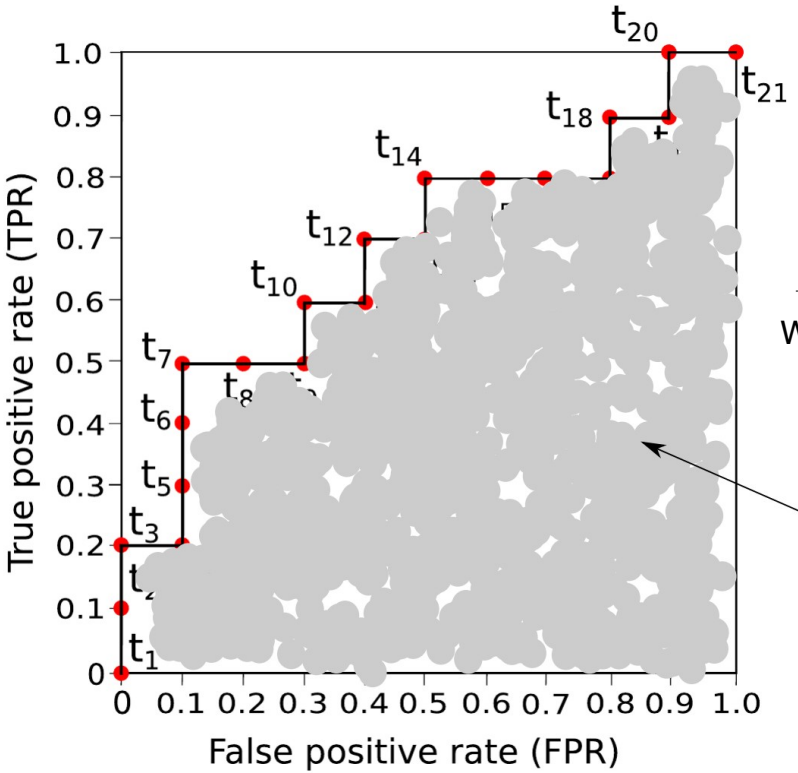
Receiver Operating Characteristics:



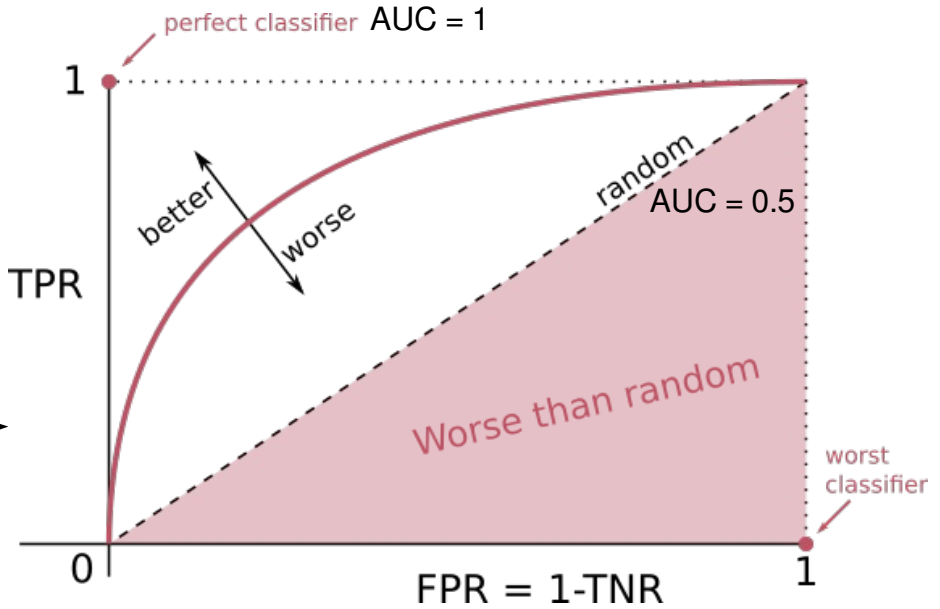
With many data points



Receiver Operating Characteristics:

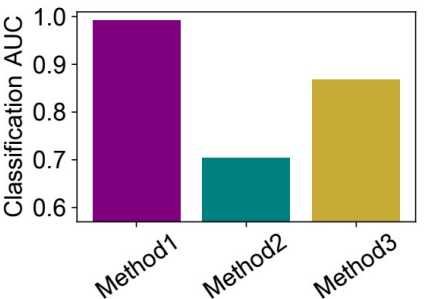


With many data points



Area Under the Curve (AUC)

Summary metric to compare classifiers

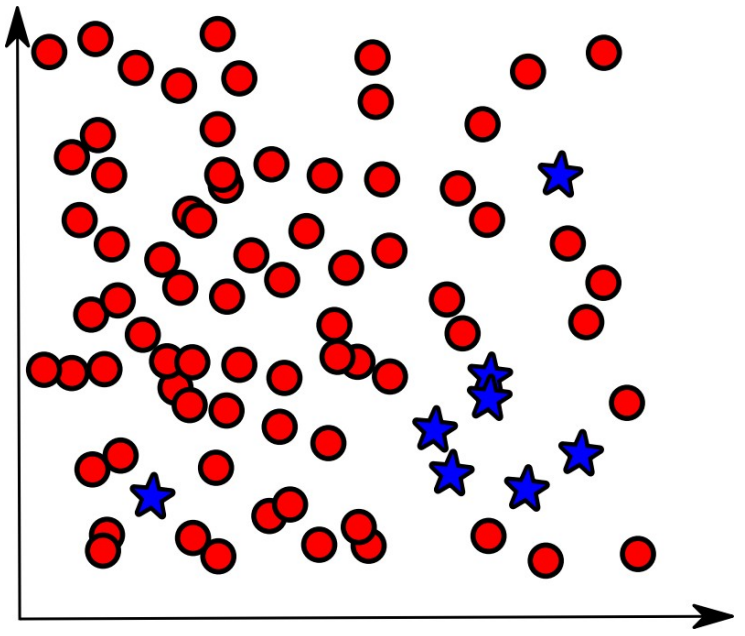


Imbalanced data

I.e., the size of classes can be very different:
majority class *vs* a minority class

Often, we can be interested in identifying the minority class (**rare-class learning problems**):

- Medical diagnosis
- Anomaly detection (detection of frauds, risks, errors in texts)

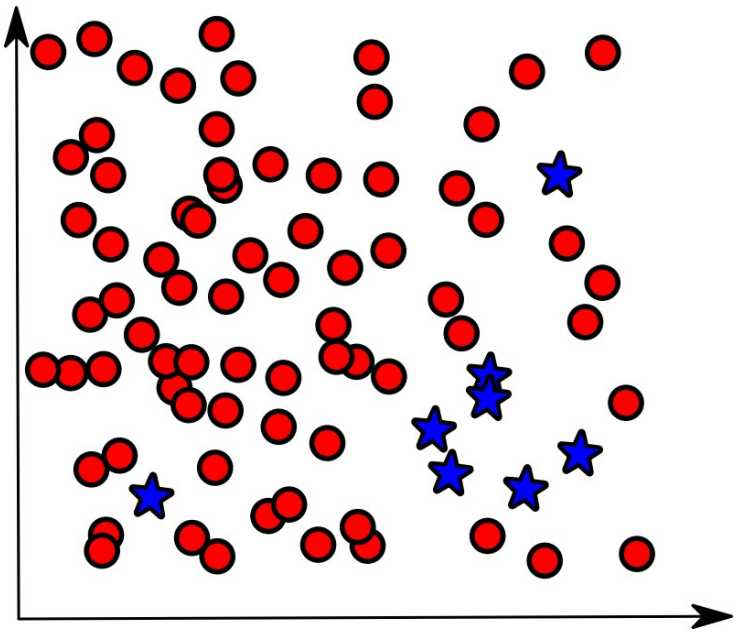


Imbalanced data

I.e., the size of classes can be very different:
majority class *vs* a minority class

Often, we can be interested in identifying the minority class (**rare-class learning problems**):

- Medical diagnosis
- Anomaly detection (detection of frauds, risks, errors in texts)



What are the right measures of classification performance?

Classification performance

Metrics of classification performance:

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{TN}{N} \equiv TNR$$

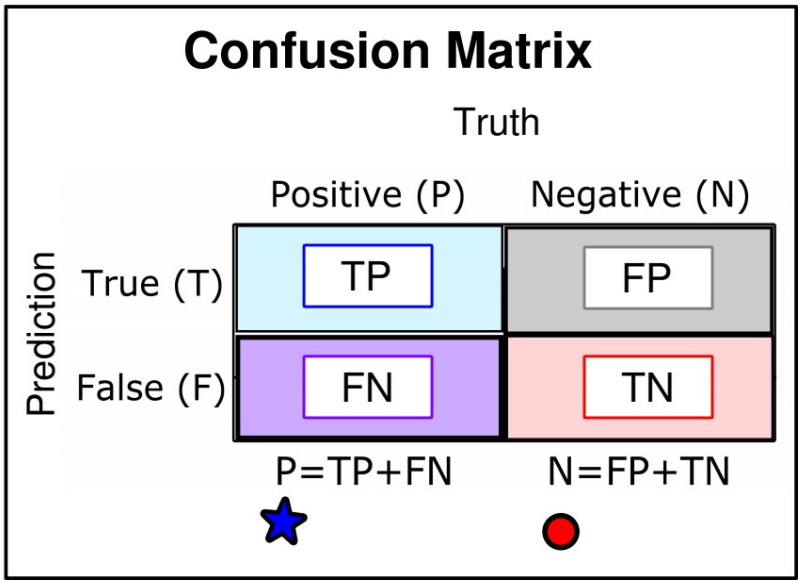
$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{TP}{P} \equiv TPR$$

Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

Positive predictive value

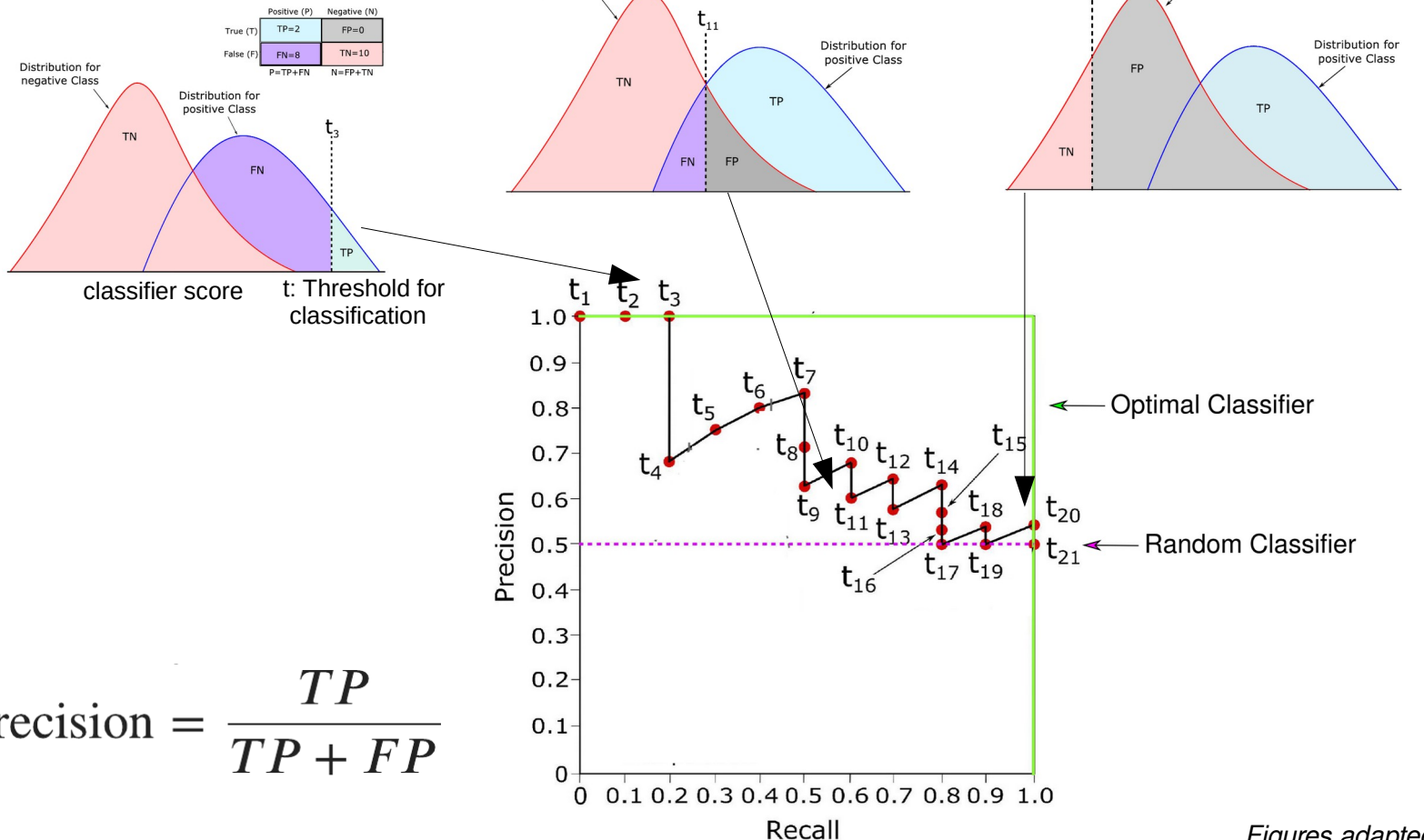
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$



F-score $\mathrm{F} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$ *Don't depend on TN*

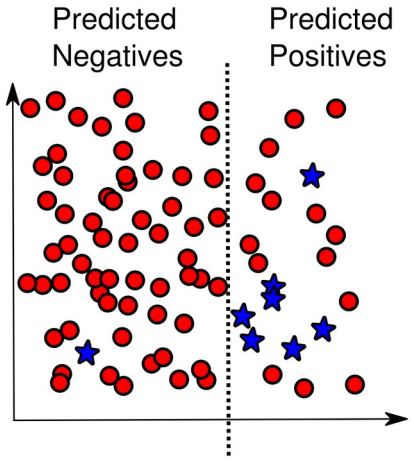
Precision Recall curve:

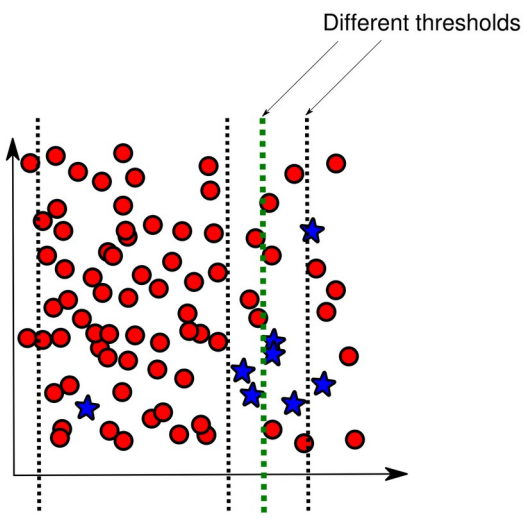
We progressively increase the threshold for classification

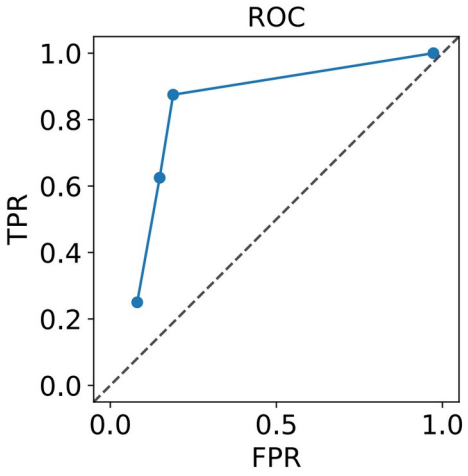


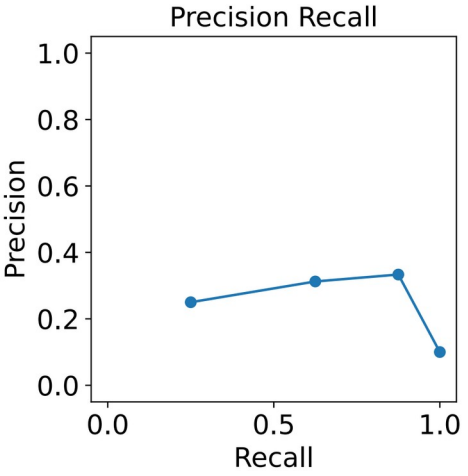
$$\text{Precision} = \frac{TP}{TP + FP}$$

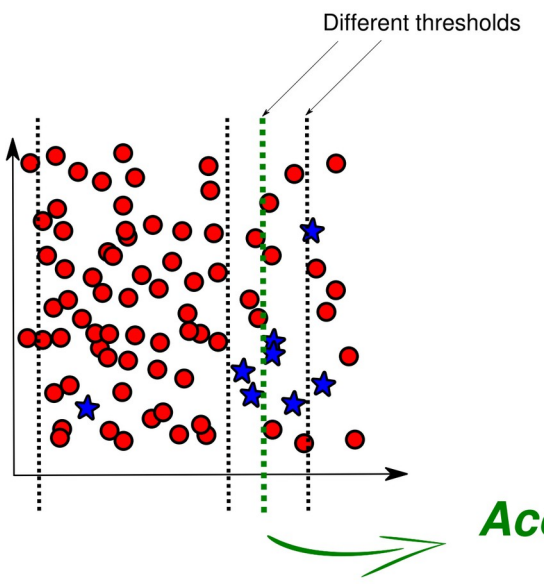
Figures adapted from Tharwat 2018

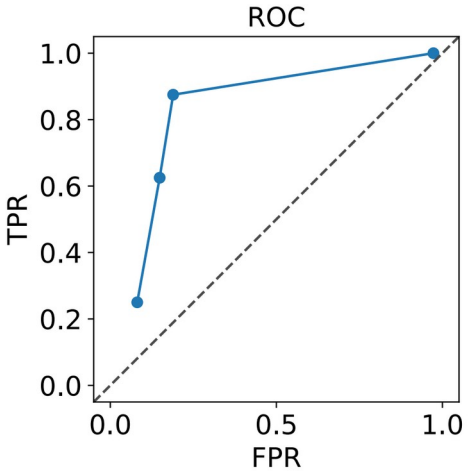

Predicted
Negatives
Predicted
Positives

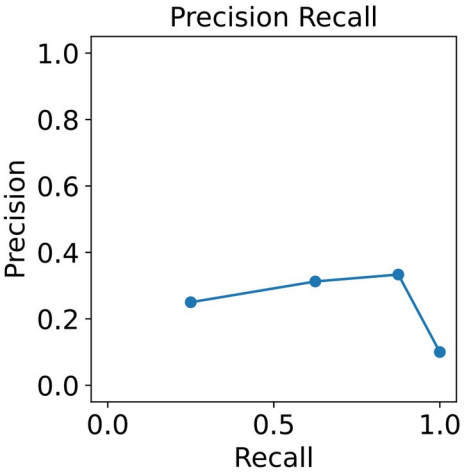

Different thresholds


ROC
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0

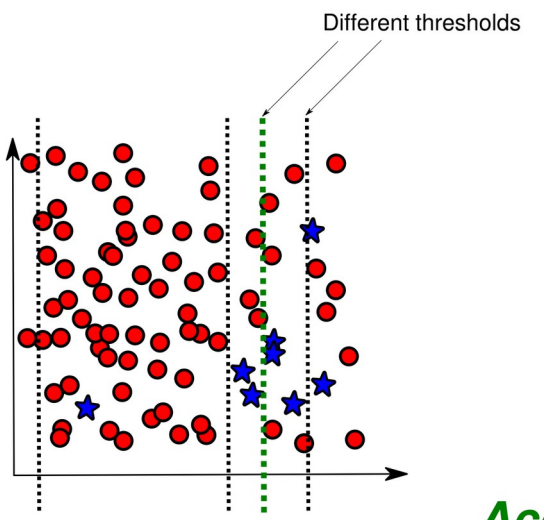

Precision Recall
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0

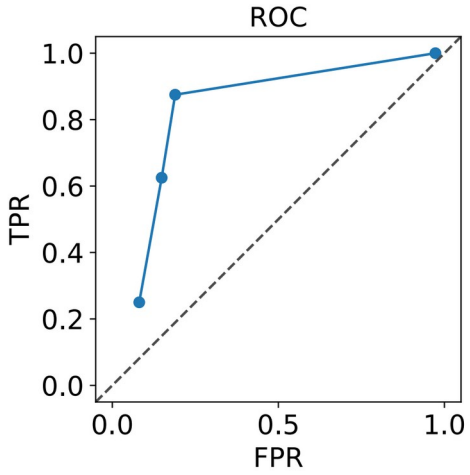


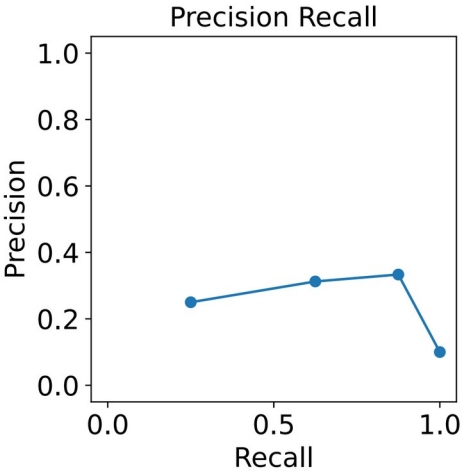




Accuracy = 0.83 ***F-score = 0.42***

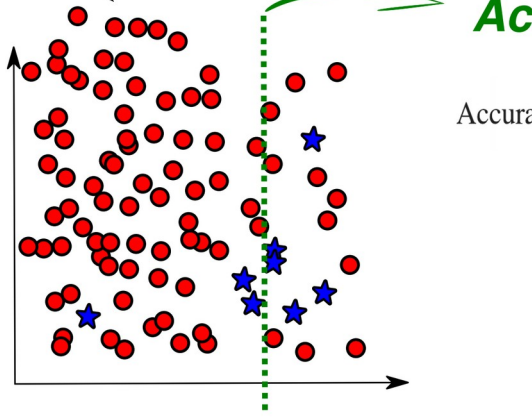






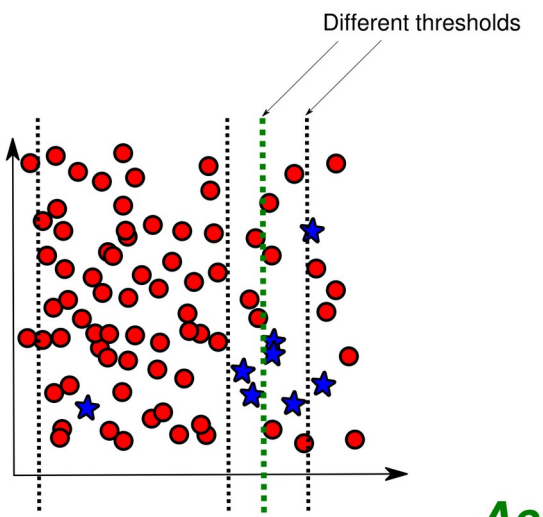
Accuracy = 0.83 ***F-score = 0.42***

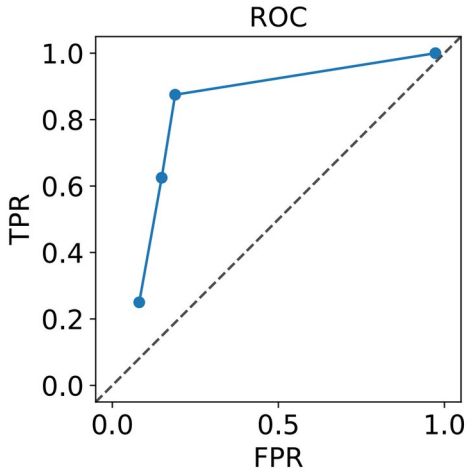
If we add TN:

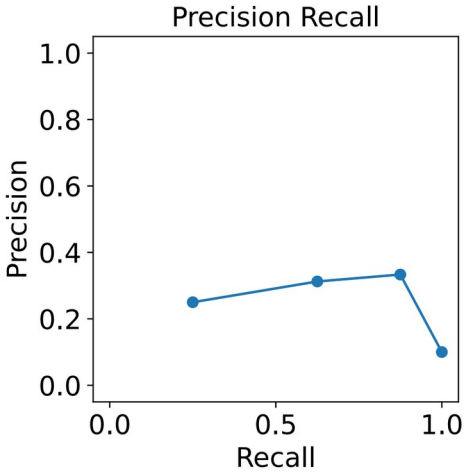


Accuracy = 0.85 ***F-score = 0.42***

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

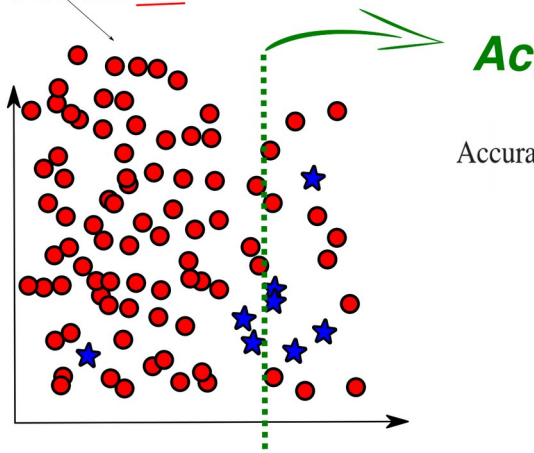






Accuracy = 0.83 ***F-score = 0.42***

If we add TN:



Accuracy = 0.85 ***F-score = 0.42***

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

In summary:

- **Accuracy and AUC can indicate misleadingly high performance** (still, they are informative about the overall discrimination ability and can be used for comparing different classifiers on the same data);

- If we are interested in how well the **minority class** (positives) is predicted out of the negatives, **it's important to look also at Precision, Recall, F-score**

Remarks:

1. Imbalanced data:

- There exist also ***ad hoc* adjustments of classification measures** (e.g. Balanced Accuracy)

Remarks:

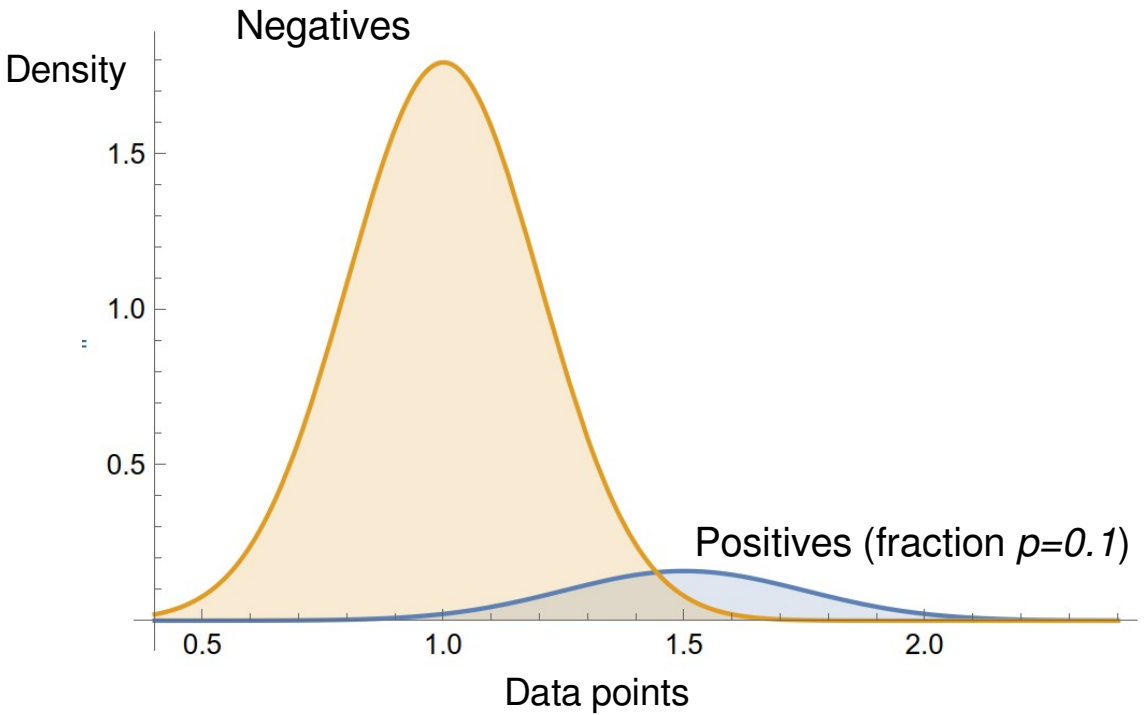
1. Imbalanced data:

- There exist also ***ad hoc* adjustments of classification measures** (e.g. Balanced Accuracy)
- There exist **strategies at the algorithm- and data-level** (e.g. under/over-sampling the majority/minority class, reweighing)

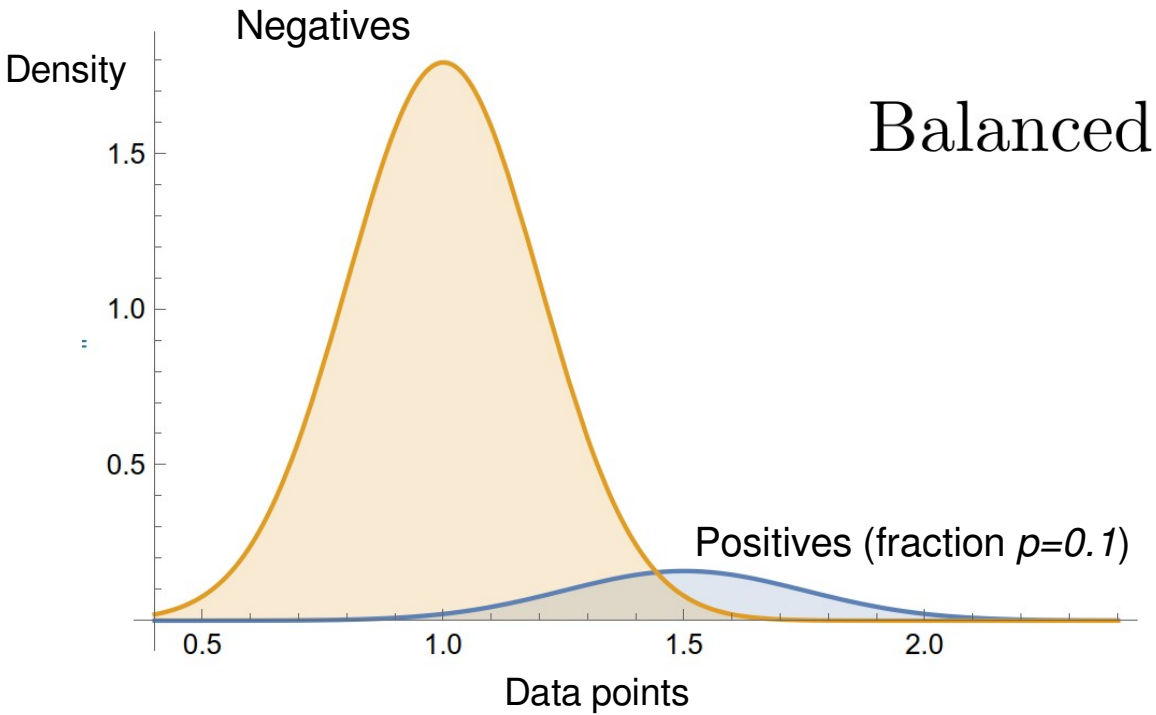
Strategies to deal with imbalanced data

Adjusted classification metrics:

Key step: assign weights to data points in test set according to the prevalence of their true class.



Strategies to deal with imbalanced data



Adjusted classification metrics:

Key step: assign weights to data points in test set according to the prevalence of their true class. Example (see Grandini 2020):

$$\text{Balanced Accuracy} = \frac{\frac{\text{TP}}{\text{P}} + \frac{\text{TN}}{\text{N}}}{2}$$

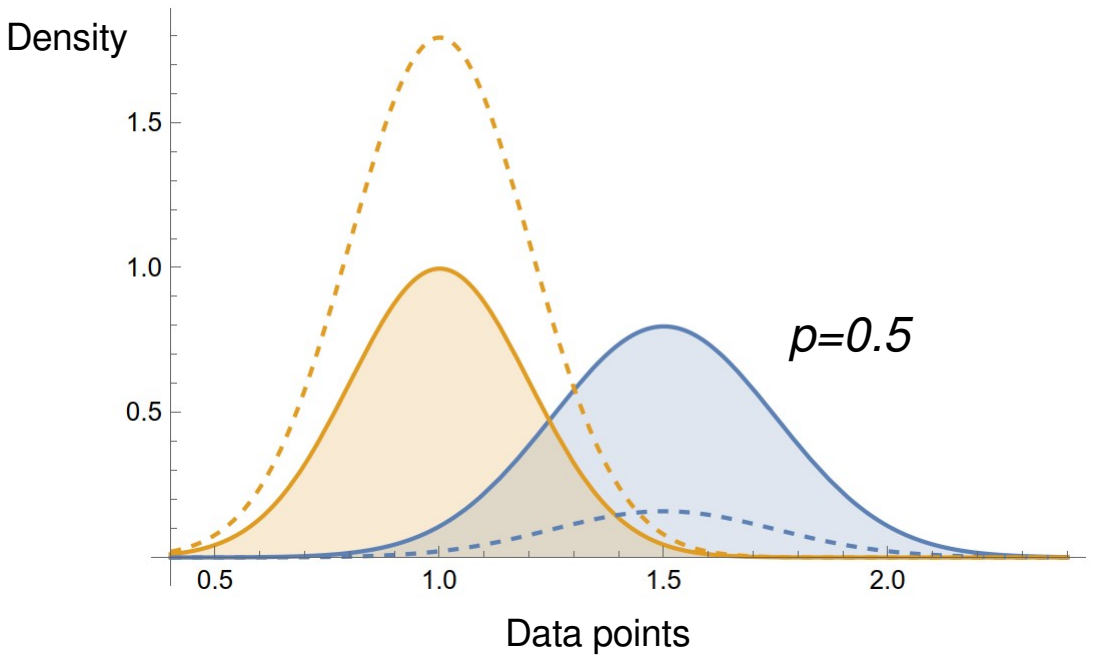
classes

TP weighted by 1/P

TN weighted by 1/N

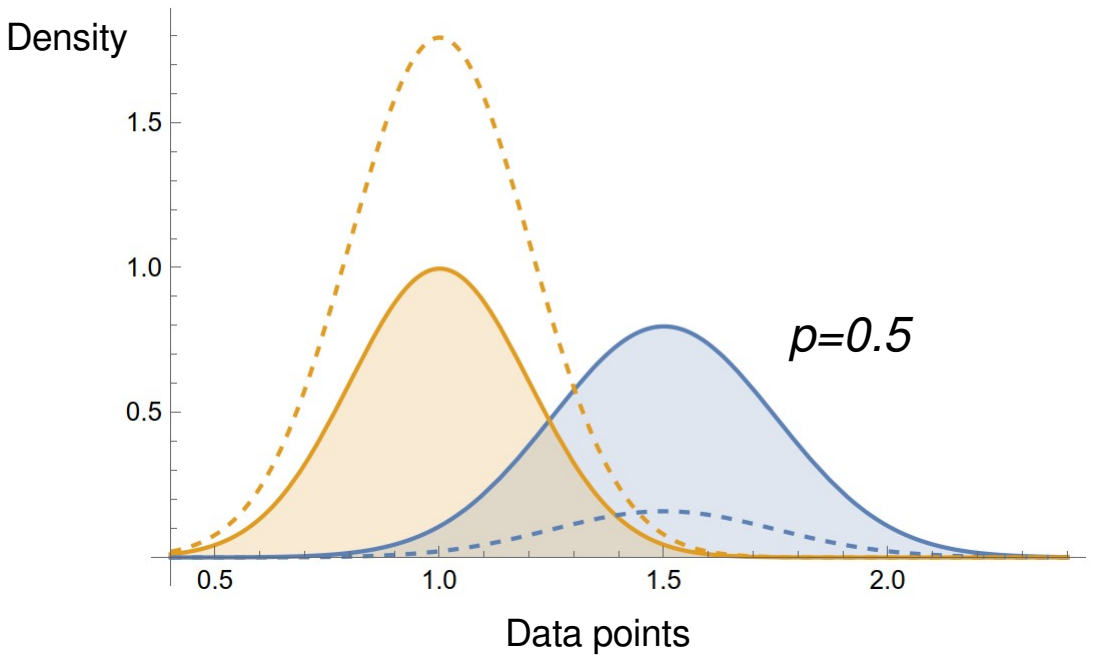
Weighted re-sampling

Re-sampling data using **bootstrap** (see next lecture!) with different weights for P and N produces a balanced dataset (to use as training set):



Weighted re-sampling

Re-sampling data using **bootstrap** (see next lecture!) with different weights for P and N produces a balanced dataset (to use as training set):



→ Advantage: it works on existing implementations

Weighted loss function

Key idea: ***train more*** on the minority class. *Example: weighted logistic regression*

CW1 2022/2023. Classification task: predict diagnosis from clinical descriptors

	patient_number	cholesterol	glucose	hdl_chol	chol_hdl_ratio	age	height	weight	bmi	systolic_bp	diastolic_bp	waist	hip	waist_hip_ratio	diabetes
0	115	224	85	30	7,5	36	69	205	30,3	150	99	37	41	0,9	No diabetes
1	318	194	95	36	5,4	63	58	210	43,9	140	100	44	53	0,83	No diabetes
2	73	207	75	44	4,7	30	72	180	24,4	118	62	35	41	0,85	No diabetes
3	69	144	81	28	5,1	30	72	165	22,4	118	78	31	38	0,82	No diabetes
4	326	181	177	24	7,5	64	71	225	31,4	130	66	44	47	0,94	Diabetes

Weighted loss function

Key idea: ***train more*** on the minority class. *Example: weighted logistic regression*

CW1 2022/2023. Classification task: predict diagnosis from clinical descriptors

	patient_number	cholesterol	glucose	hdl_chol	chol_hdl_ratio	age	height	weight	bmi	systolic_bp	diastolic_bp	waist	hip	waist_hip_ratio	diabetes
0	115	224	85	30	7,5	36	69	205	30,3	150	99	37	41	0,9	No diabetes
1	318	194	95	36	5,4	63	58	210	43,9	140	100	44	53	0,83	No diabetes
2	73	207	75	44	4,7	30	72	180	24,4	118	62	35	41	0,85	No diabetes
3	69	144	81	28	5,1	30	72	165	22,4	118	78	31	38	0,82	No diabetes
4	326	181	177	24	7,5	64	71	225	31,4	130	66	44	47	0,94	Diabetes

```
print(f"Frequency of 'No Diabetes' label: {freq_0}")
print(f"Frequency of 'Diabetes' label: {freq_1}")
```

```
Frequency of 'No Diabetes' label: 0.8388278388278388
Frequency of 'Diabetes' label: 0.16117216117216118
```

Diagnosis ‘Diabetes’ is the minority class

Weighted loss function

Key idea: ***train more*** on the minority class. *Example: weighted logistic regression*

CW1 2022/2023. Classification task: predict diagnosis from clinical descriptors

	patient_number	cholesterol	glucose	hdl_chol	chol_hdl_ratio	age	height	weight	bmi	systolic_bp	diastolic_bp	waist	hip	waist_hip_ratio	diabetes
0	115	224	85	30	7,5	36	69	205	30,3	150	99	37	41	0,9	No diabetes
1	318	194	95	36	5,4	63	58	210	43,9	140	100	44	53	0,83	No diabetes
2	73	207	75	44	4,7	30	72	180	24,4	118	62	35	41	0,85	No diabetes
3	69	144	81	28	5,1	30	72	165	22,4	118	78	31	38	0,82	No diabetes
4	326	181	177	24	7,5	64	71	225	31,4	130	66	44	47	0,94	Diabetes

```
print(f"Frequency of 'No Diabetes' label: {freq_0}")
print(f"Frequency of 'Diabetes' label: {freq_1}")
```

```
Frequency of 'No Diabetes' label: 0.8388278388278388
Frequency of 'Diabetes' label: 0.16117216117216118
```

Diagnosis ‘Diabetes’ is the minority class

Introduce a weight for each data point in the loss function

$$E(L) = -\frac{1}{N}\sum_{i=1}^{N} w^{(i)}(y^{(i)} \log h_\beta(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_\beta(\boldsymbol{x}^{(i)})))$$

Weight: ?

Weighted loss function

Key idea: ***train more*** on the minority class. *Example: weighted logistic regression*

CW1 2022/2023. Classification task: predict diagnosis from clinical descriptors

	patient_number	cholesterol	glucose	hdl_chol	chol_hdl_ratio	age	height	weight	bmi	systolic_bp	diastolic_bp	waist	hip	waist_hip_ratio	diabetes
0	115	224	85	30	7,5	36	69	205	30,3	150	99	37	41	0,9	No diabetes
1	318	194	95	36	5,4	63	58	210	43,9	140	100	44	53	0,83	No diabetes
2	73	207	75	44	4,7	30	72	180	24,4	118	62	35	41	0,85	No diabetes
3	69	144	81	28	5,1	30	72	165	22,4	118	78	31	38	0,82	No diabetes
4	326	181	177	24	7,5	64	71	225	31,4	130	66	44	47	0,94	Diabetes

```
print(f"Frequency of 'No Diabetes' label: {freq_0}")
print(f"Frequency of 'Diabetes' label: {freq_1}")
```

```
Frequency of 'No Diabetes' label: 0.8388278388278388
Frequency of 'Diabetes' label: 0.16117216117216118
```

Diagnosis ‘Diabetes’ is the minority class

Introduce a weight for each data point in the loss function

$$E(L) = -\frac{1}{N}\sum_{i=1}^{N} w^{(i)}(y^{(i)} \log h_\beta(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_\beta(\boldsymbol{x}^{(i)})))$$

Weight: the frequency of the other class
In the sum, both classes have same weight

Weighted loss function

Key idea: ***train more*** on the minority class. *Example: weighted logistic regression*

CW1 2022/2023. Classification task: predict diagnosis from clinical descriptors

	patient_number	cholesterol	glucose	hdl_chol	chol_hdl_ratio	age	height	weight	bmi	systolic_bp	diastolic_bp	waist	hip	waist_hip_ratio	diabetes
0	115	224	85	30	7,5	36	69	205	30,3	150	99	37	41	0,9	No diabetes
1	318	194	95	36	5,4	63	58	210	43,9	140	100	44	53	0,83	No diabetes
2	73	207	75	44	4,7	30	72	180	24,4	118	62	35	41	0,85	No diabetes
3	69	144	81	28	5,1	30	72	165	22,4	118	78	31	38	0,82	No diabetes
4	326	181	177	24	7,5	64	71	225	31,4	130	66	44	47	0,94	Diabetes

```
print(f"Frequency of 'No Diabetes' label: {freq_0}")
print(f"Frequency of 'Diabetes' label: {freq_1}")
```

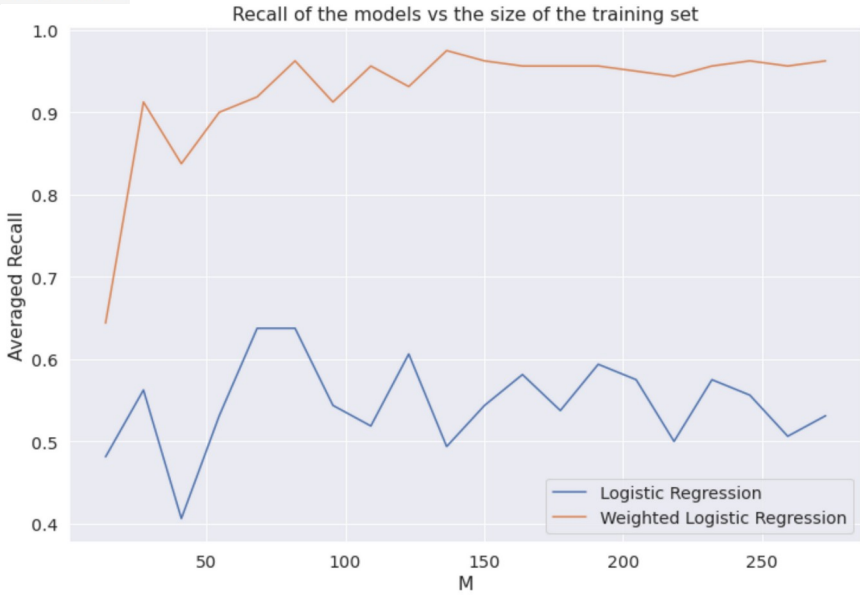
```
Frequency of 'No Diabetes' label: 0.8388278388278388
Frequency of 'Diabetes' label: 0.16117216117216118
```

Diagnosis ‘Diabetes’ is the minority class

Introduce a weight for each data point in the loss function

$$E(L) = -\frac{1}{N}\sum_{i=1}^{N} w^{(i)}(y^{(i)} \log h_\beta(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_\beta(\boldsymbol{x}^{(i)})))$$

Weight: the frequency of the other class
In the sum, both classes have same weight



Remarks:

1. Imbalanced data:

- There exist also ***ad hoc* adjustments of classification measures** (e.g. Balanced Accuracy)
- There exist **strategies at the algorithm- and data-level** (e.g. under/over-sampling the majority/minority class, reweighing)

2. Multi-class classification:

		PREDICTED classification				
	Classes	a	b	c	d	Total
ACTUAL classification	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	**52**

From Grandini et al. 2020

Remarks:

1. Imbalanced data:

- There exist also ***ad hoc* adjustments of classification measures** (e.g. Balanced Accuracy)
- There exist **strategies at the algorithm- and data-level** (e.g. under/over-sampling the majority/minority class, reweighing)

2. Multi-class classification:

- Some measures are easily generalisable:

$$Acc = \frac{\text{Sum of correct predictions}}{\text{Sum of all entries}}$$

		PREDICTED classification				
	Classes	a	b	c	d	Total
ACTUAL classification	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	**52**

From Grandini et al. 2020

Remarks:

1. Imbalanced data:

- There exist also ***ad hoc* adjustments of classification measures** (e.g. Balanced Accuracy)
- There exist **strategies at the algorithm- and data-level** (e.g. under/over-sampling the majority/minority class, reweighing)

2. Multi-class classification:

- Some measures are easily generalisable:

$$Acc = \frac{\text{Sum of correct predictions}}{\text{Sum of all entries}}$$

		PREDICTED classification				
	Classes	a	b	c	d	Total
ACTUAL classification	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	**52**

From Grandini et al. 2020

- In general: micro/macro averages of metrics over classes

Micro/macro averages for multi-class classification

ACTUAL classification \ PREDICTED classification	a	b	c	d
a	TN	FP	TN	TN
b	FN	TP	FN	FN
c	TN	FP	TN	TN
d	TN	FP	TN	TN

Reference class b

From Grandini et al. 2020

ACTUAL classification \ PREDICTED classification	a	b	c	d	Total
a	50	37	24	39	150
b	10	480	5	3	498
c	14	10	765	1	790
d	0	2	9	101	112
Total	74	529	803	144	**1550**

Macro: measure performance for each class and then average. E.g., for precision:

For a class *k*:

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \qquad MacroAveragePrecision = \frac{\sum_{k=1}^{K} Precision_k}{K}$$

Micro/macro averages for multi-class classification

ACTUAL classification \ PREDICTED classification	a	b	c	d
a	TN	FP	TN	TN
b	FN	TP	FN	FN
c	TN	FP	TN	TN
d	TN	FP	TN	TN

Reference class b

From Grandini et al. 2020

ACTUAL classification \ PREDICTED classification	a	b	c	d	Total
a	50	37	24	39	150
b	10	480	5	3	498
c	14	10	765	1	790
d	0	2	9	101	112
Total	74	529	803	144	**1550**

Macro: measure performance for each class and then average. E.g., for precision:

For a class *k*:

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \qquad MacroAveragePrecision = \frac{\sum_{k=1}^{K} Precision_k}{K}$$

Micro: aggregate and measure performance:

$$Micro\ Average\ Precision = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} Total\ Column_k} = \frac{\sum_{k=1}^{K} TP_k}{Grand\ Total}$$

Micro/macro averages for multi-class classification

ACTUAL classification \ PREDICTED classification	a	b	c	d
a	TN	FP	TN	TN
b	FN	TP	FN	FN
c	TN	FP	TN	TN
d	TN	FP	TN	TN

Reference class b

From Grandini et al. 2020

ACTUAL classification \ PREDICTED classification	a	b	c	d	Total
a	50	37	24	39	150
b	10	480	5	3	498
c	14	10	765	1	790
d	0	2	9	101	112
Total	74	529	803	144	**1550**

Macro: measure performance for each class and then average. E.g., for precision:

For a class *k*:

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \qquad MacroAveragePrecision = \frac{\sum_{k=1}^{K} Precision_k}{K}$$

Equal weight to each class (fairer representation of minority classes)

Micro: aggregate and measure performance:

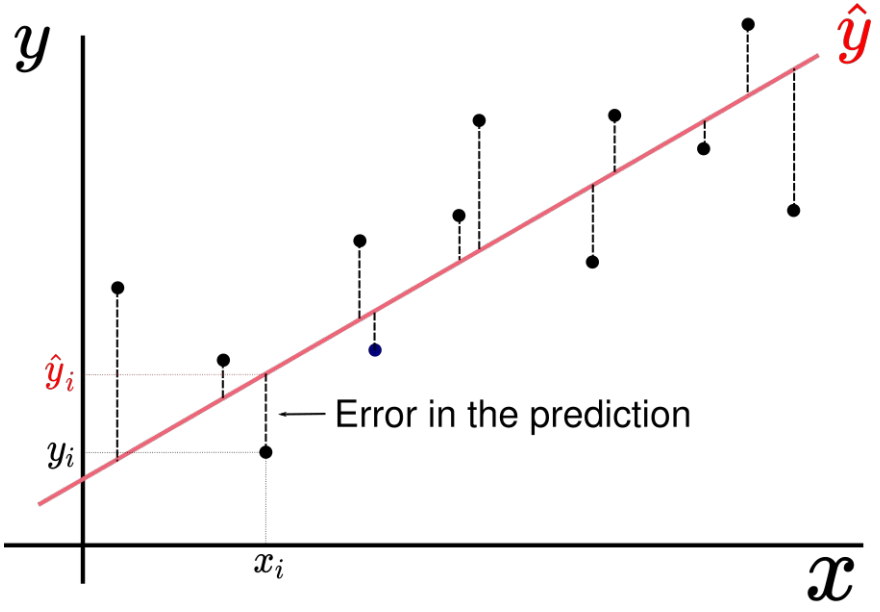
$$Micro\ Average\ Precision = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} Total\ Column_k} = \frac{\sum_{k=1}^{K} TP_k}{Grand\ Total}$$

Equal weight to each sample

Measuring the quality of regression

1. Mean Squared Error

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



E.g. MSE (Training): 9.75 MSE (Test): 10.02

Measuring the quality of regression

2. R^2 score: it measures the variation in the data explained by the fitted relationship

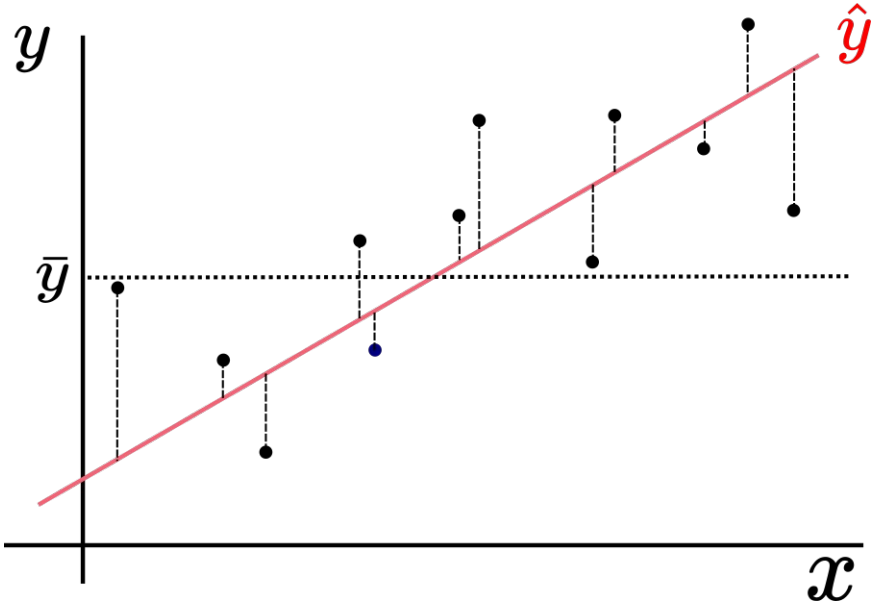
Variation around the line
(prediction error)

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

Variation around the mean

Mean

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$



$R^2 = 0$	$R^2 = 1$
Poor prediction	Perfect prediction

Measuring the quality of regression

2. R^2 score: it measures the variation in the data explained by the fitted relationship

Variation around the line
(prediction error)

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

Variation around the mean

Mean

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Quality of the prediction:

- Quality of the solution
- Appropriate formulation of the regression problem