Algorithmic Data Analysis

Esther Galbrun

Spring 2024

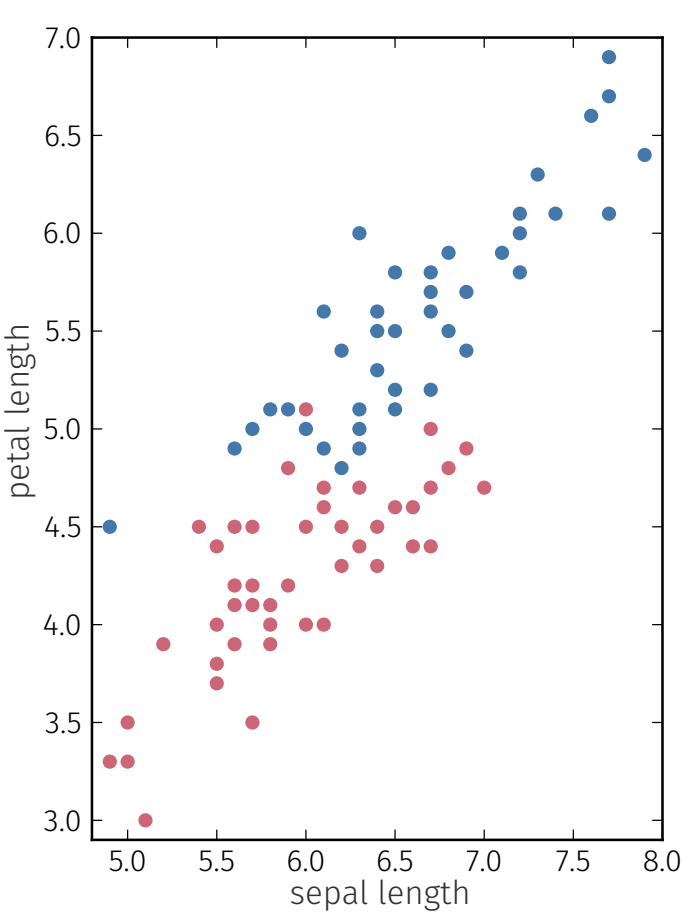
UNIVERSITY OF
EASTERN FINLAND

Last updated on 2023-12-28 11:06

Part I

Classification variants

A simple example



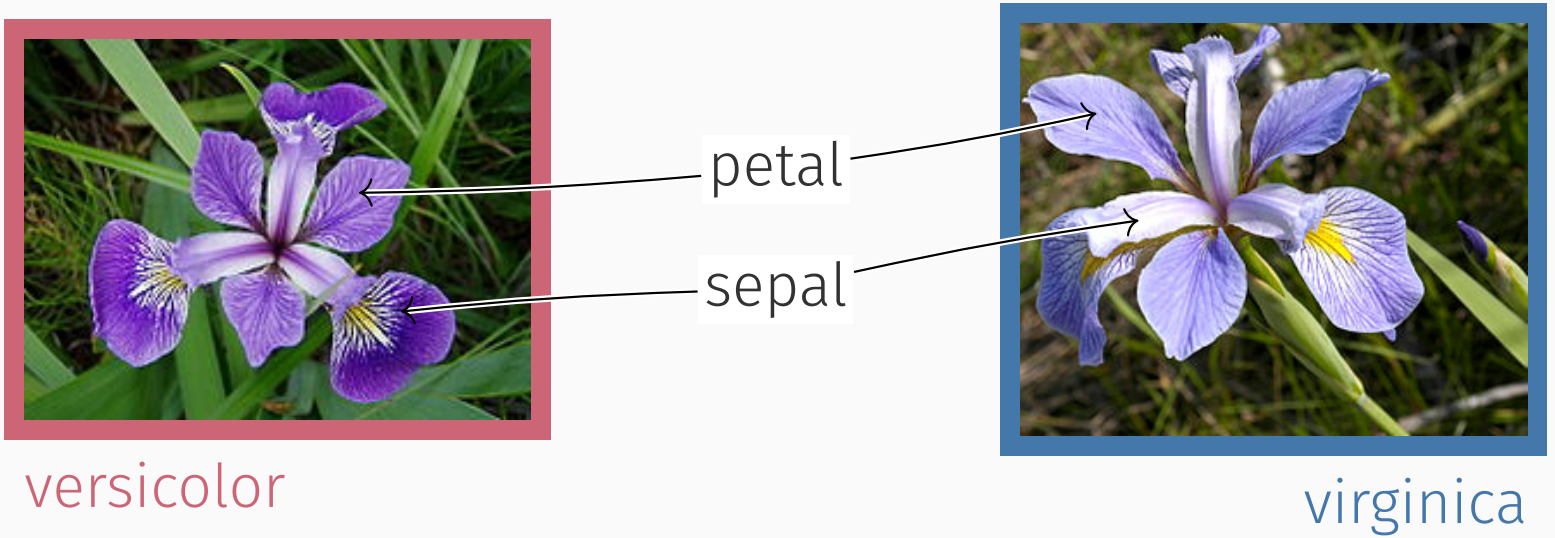
A dataset with two classes

A simple example

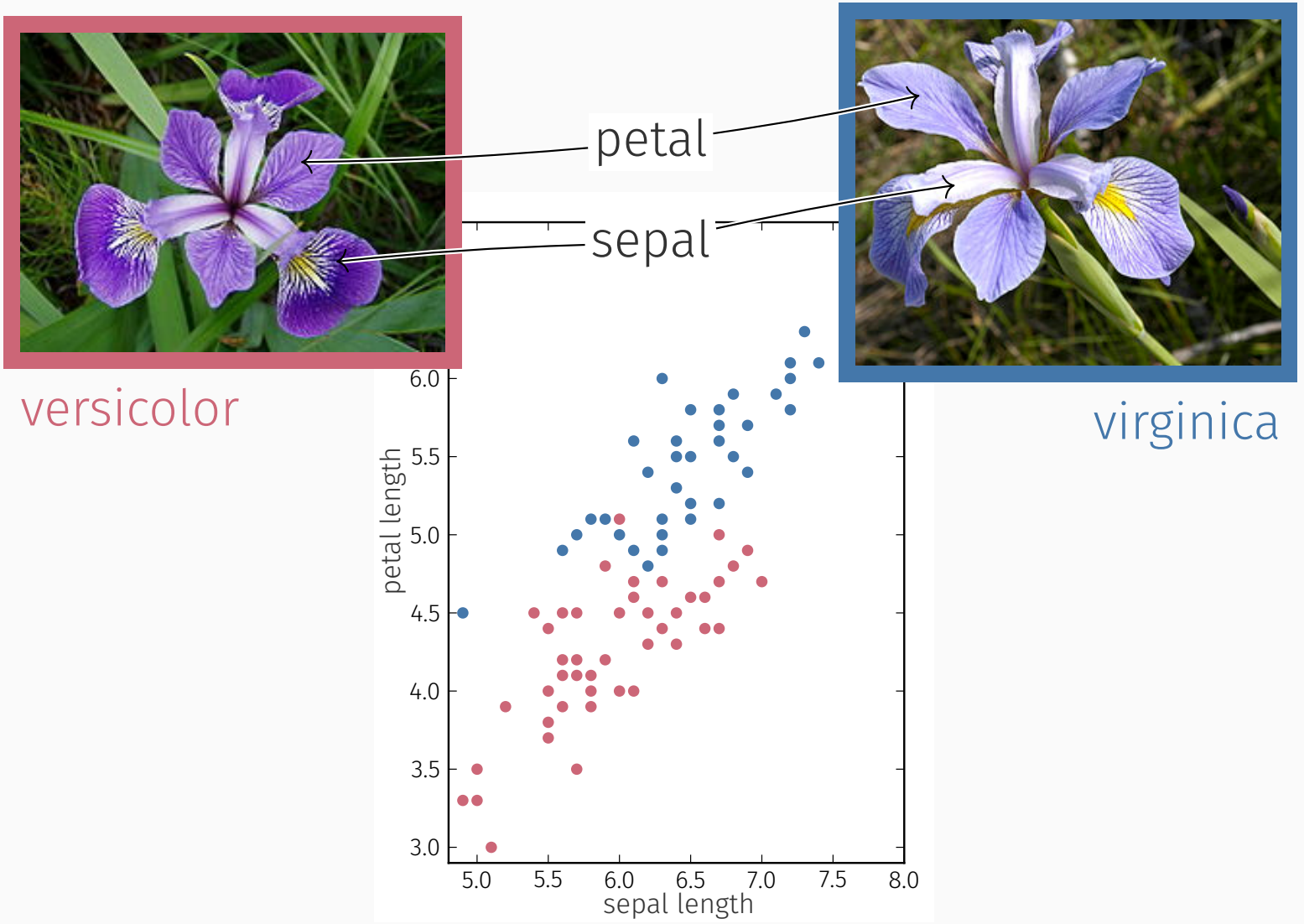
data points: Iris flowers

attributes: physical properties,
length of the petal and length of the sepal in *cm*

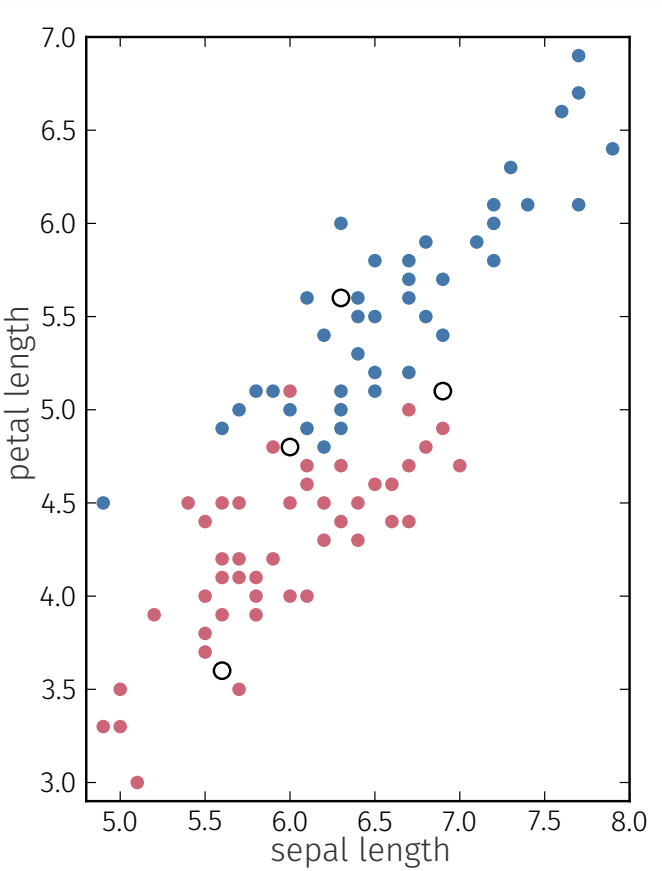
class: species, *versicolor* vs. *virginica*



A simple example



A simple example

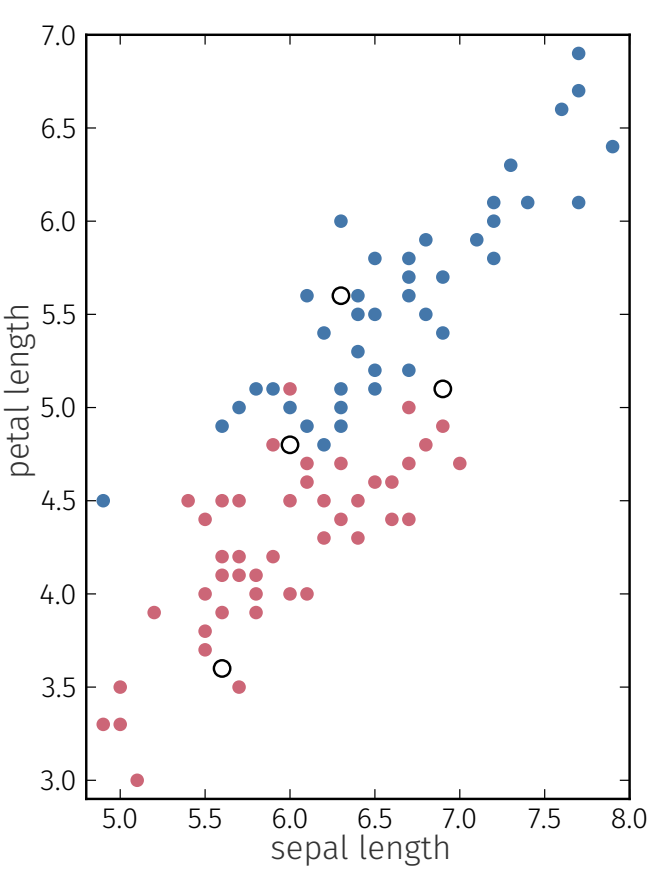


?

Class information, i.e. species, is absent for some points

Can we use the available information to predict it?

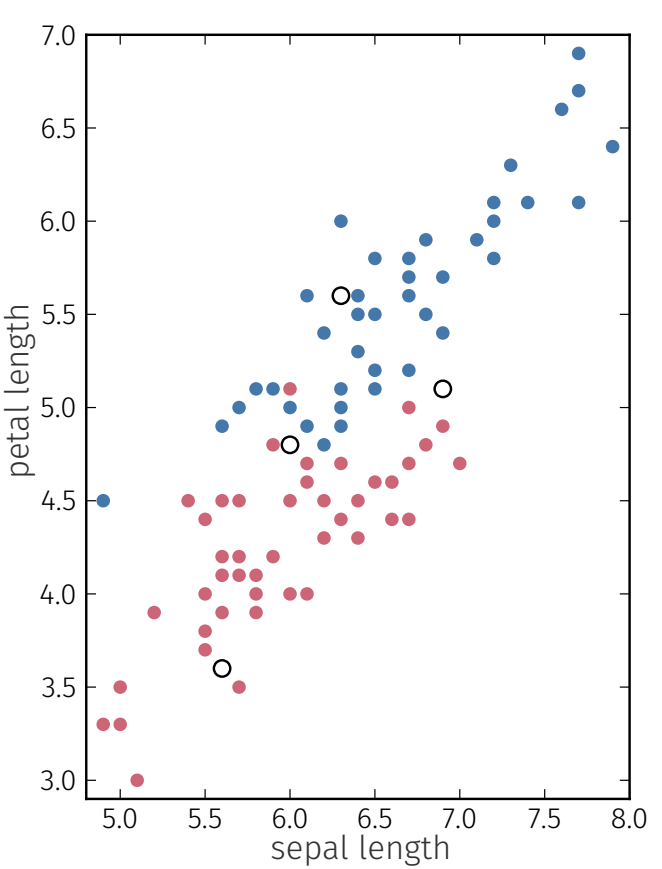
A simple example



?

classification aim to assign a class label to each instance

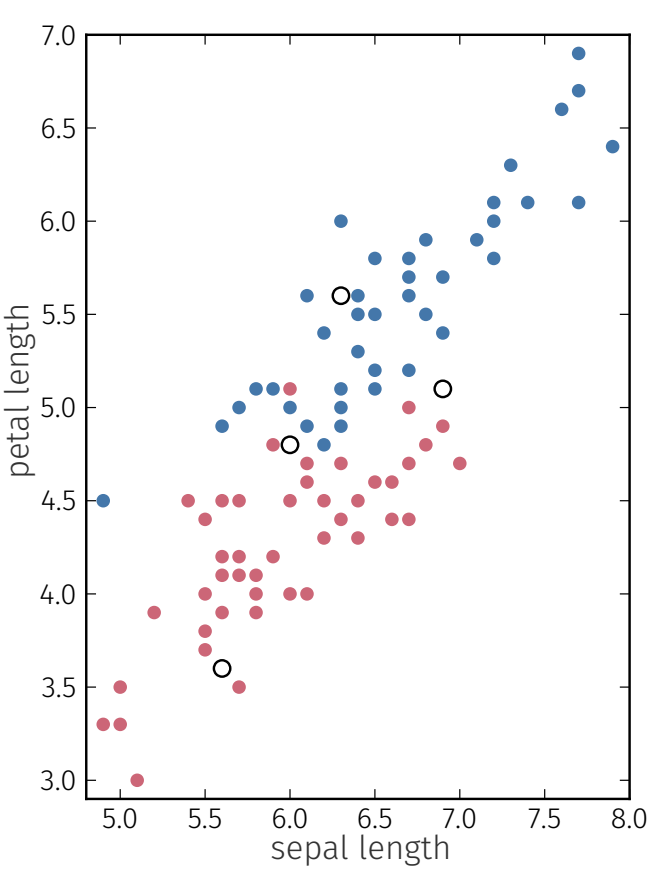
A simple example



?

binary there are two classes to choose from

A simple example



?

supervised labelled training instances are available

A simple example

supervised labelled training instances are available

binary there are two classes to choose from

classification aim to assign a class label to each instance

A typical supervised binary classification problem

Some notations

The data set, denoted as $\mathcal{D}$, contains n data points and m attributes, i.e. it is a $n \times m$ matrix

A data point is a m-dimensional vector $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_m \rangle$
We denote $\boldsymbol{x}^{(j)}$ the j^{th} data point of $\mathcal{D}$, i.e. the j^{th} row
Data points are sometimes called *instances* or *examples*

Class labels are arranged into a n-dimensional vector
$\boldsymbol{y} = \langle y_1, y_2, \ldots, y_n \rangle \in \mathcal{L}^n$, where $l = |\mathcal{L}|$ is the number of classes
That is, y_j is the class label associated with data point $\boldsymbol{x}^{(j)}$
In binary classification, class labels take value -1 or $+1$ (sometimes 0 or 1 instead), i.e. $\mathcal{L} = \{-1, +1\}$ (respectively $\mathcal{L} = \{0, 1\}$) and the two classes might be referred to as negative and positive, respectively

Different methods

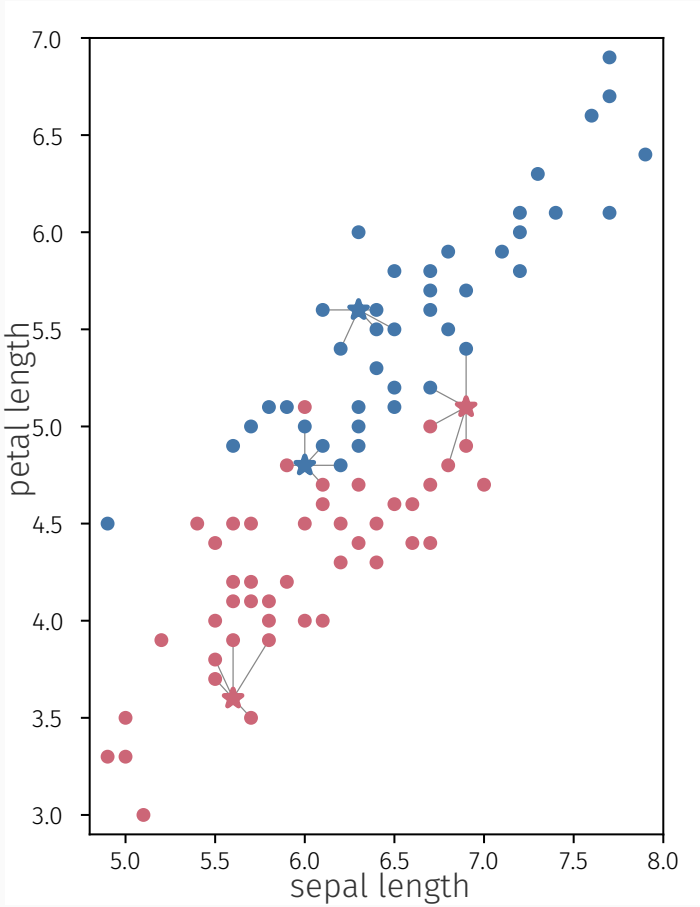
A typical supervised binary classification problem

Various classification methods are available to tackle it

Different methods

Look at the most similar data points

→ k nearest neighbors (k-NN)



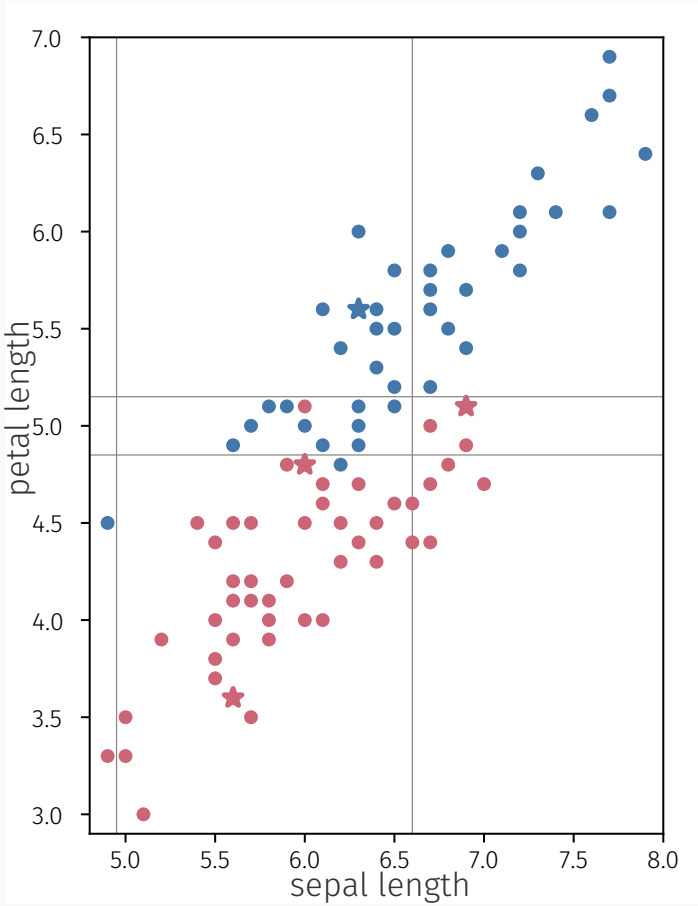
majority class

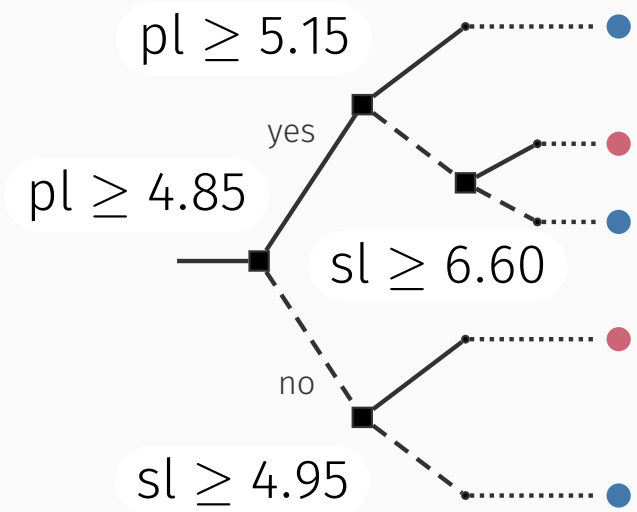
among k nearest neighbors

Different methods

Apply a sequence of tests on attributes' values

→ classification tree

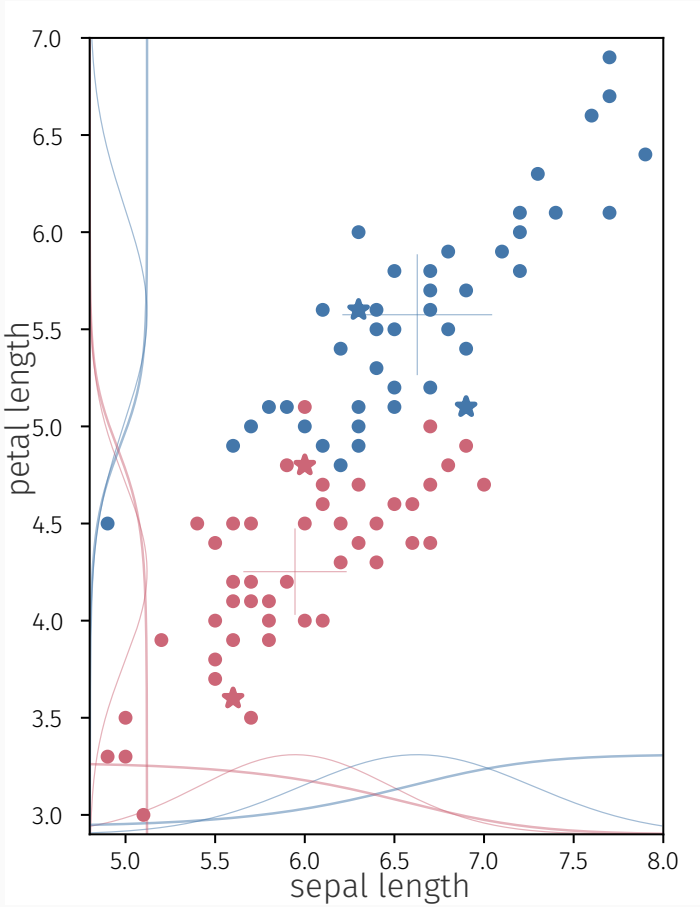




Different methods

Look at class probabilities conditioned on attributes' values

→ Naive bayes



$$P(c \mid \text{sl}, \text{sp}) \propto P(c) \cdot P(\text{sl} \mid c) \cdot P(\text{sp} \mid c)$$

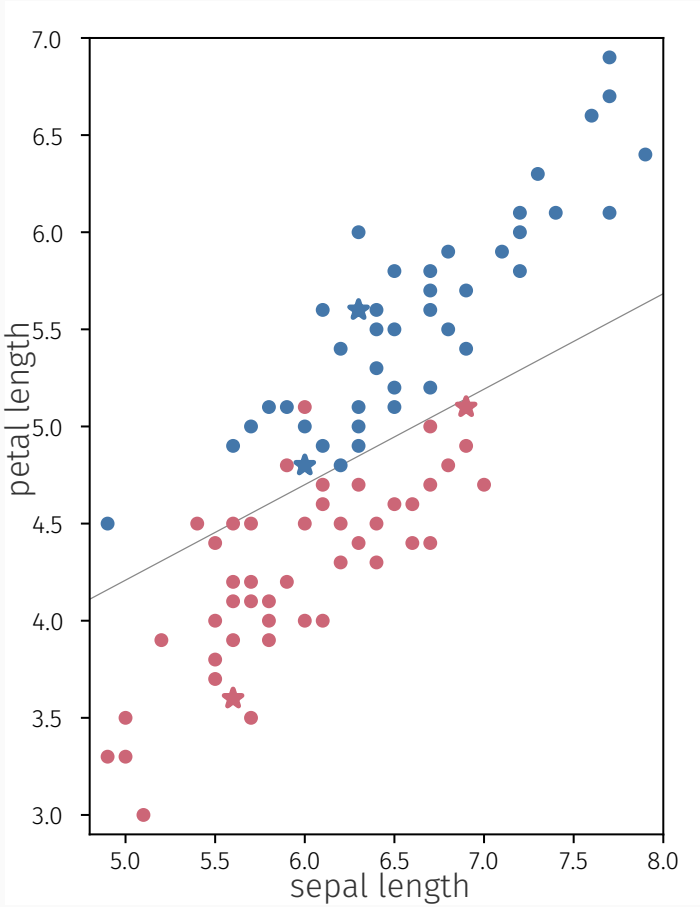
$$P(\bullet \mid \text{sl}, \text{sp}) > P(\bullet \mid \text{sl}, \text{sp})$$

$$P(\bullet \mid \text{sl}, \text{sp}) \leq P(\bullet \mid \text{sl}, \text{sp})$$

Different methods

Look at the sign of a linear combination of the attributes

→ perceptron



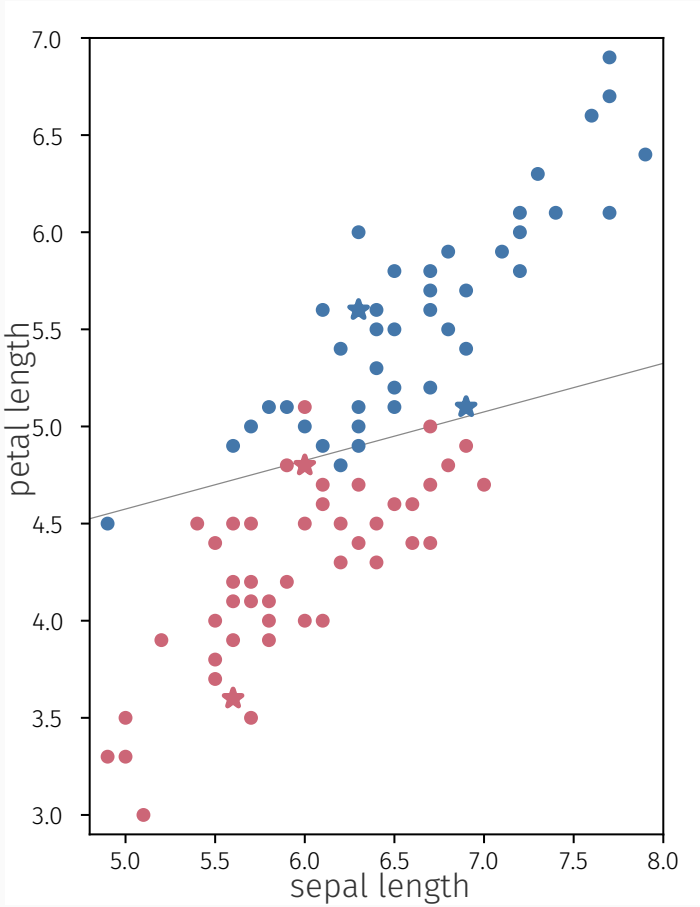
$$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 < 0$$

$$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 \geq 0$$

Different methods

Look at the sign of a linear combination of the attributes

→ support vector machine (SVM)



$$\text{sl} - 4 \cdot \text{pl} + 13.3 < 0$$

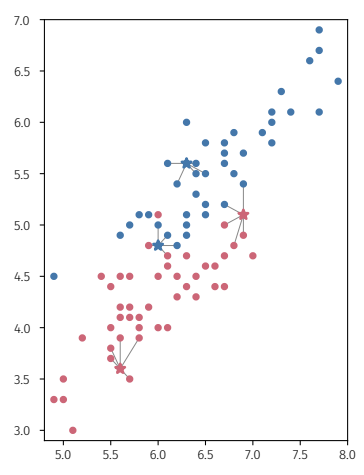
$$\text{sl} - 4 \cdot \text{pl} + 13.3 \geq 0$$

Different methods

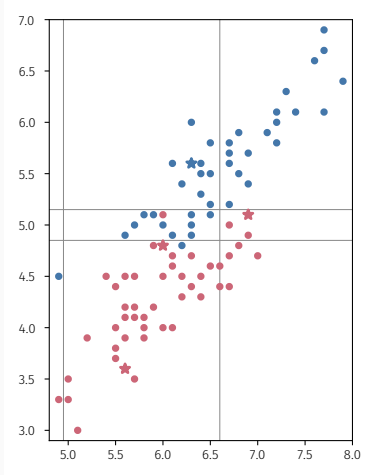
A typical supervised binary classification problem

Various classification methods are available to tackle it

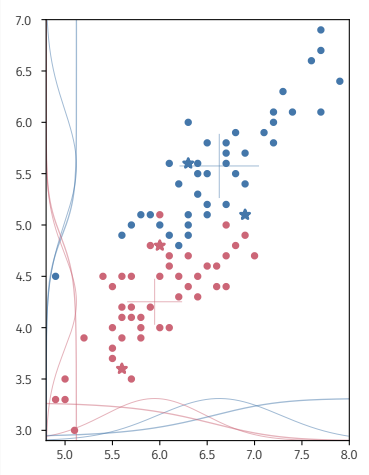
k-NN



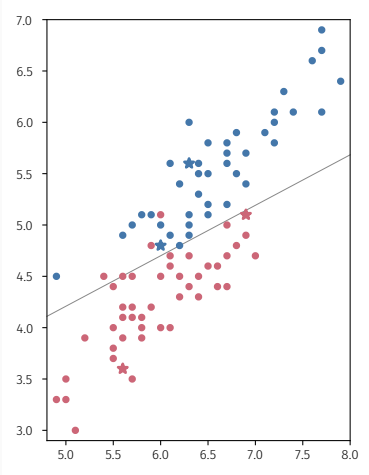
decision tree



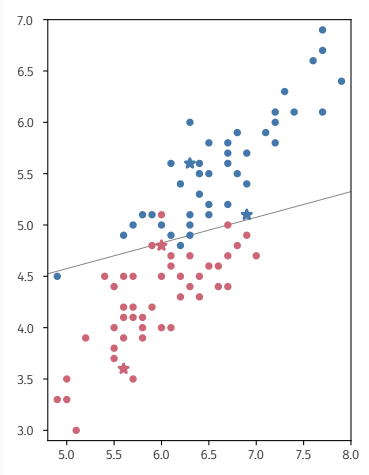
naive Bayes



perceptron



SVM



A simple example

A typical supervised binary classification problem
Various classification methods are available to tackle it

Problem variants

- What if there are more than two classes?
  → Multi-class learning
- What if the two classes are not equally represented?
  → Rare-class learning

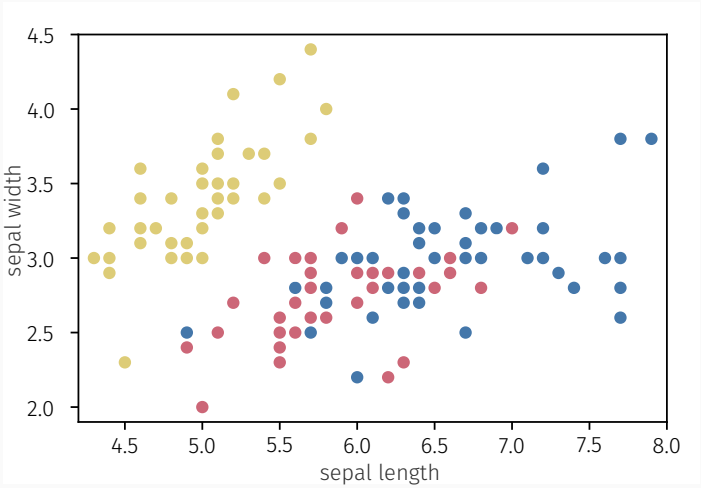
Methods

- How about combining multiple classifiers?
  → Ensemble methods

Multi-class learning

More irises

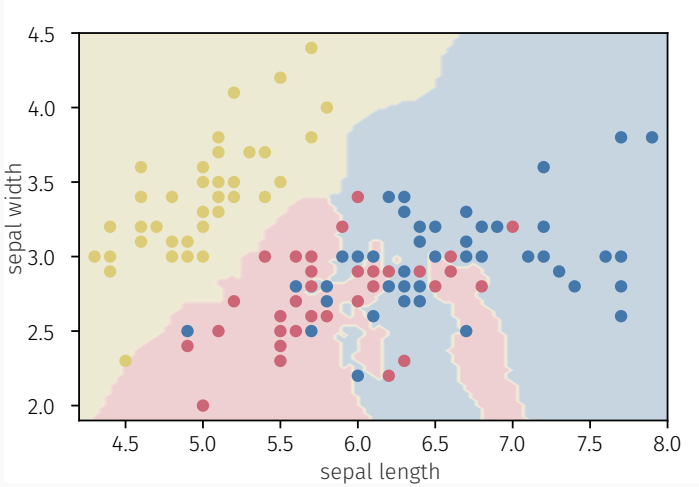
How about telling apart three species of irises?

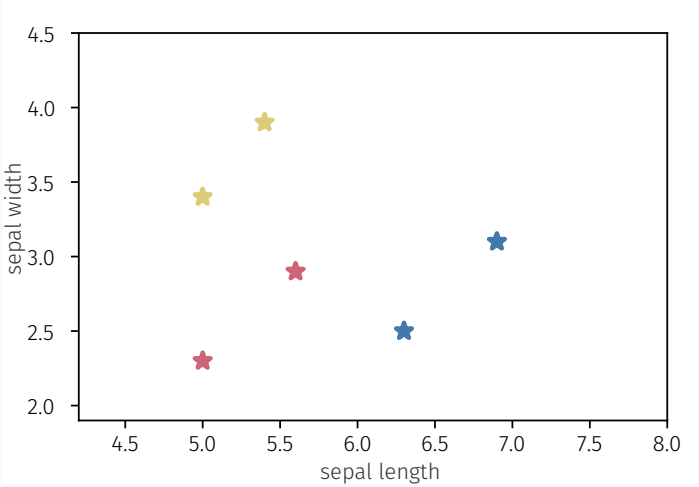


No adaptation needed

Some methods can handle multiple classes

→ k nearest neighbors (k-NN)

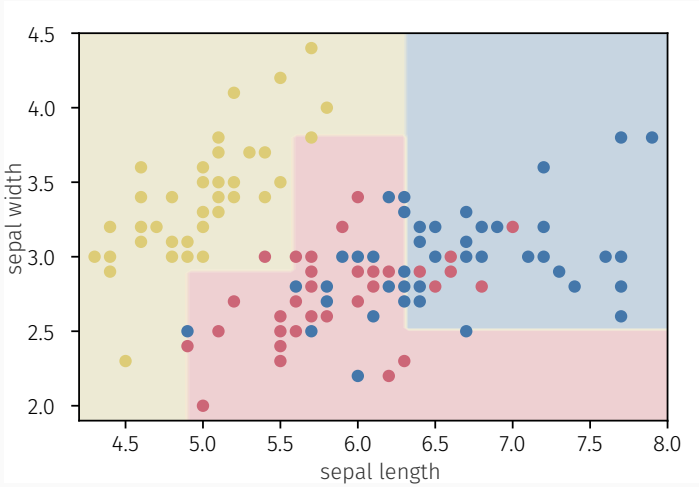


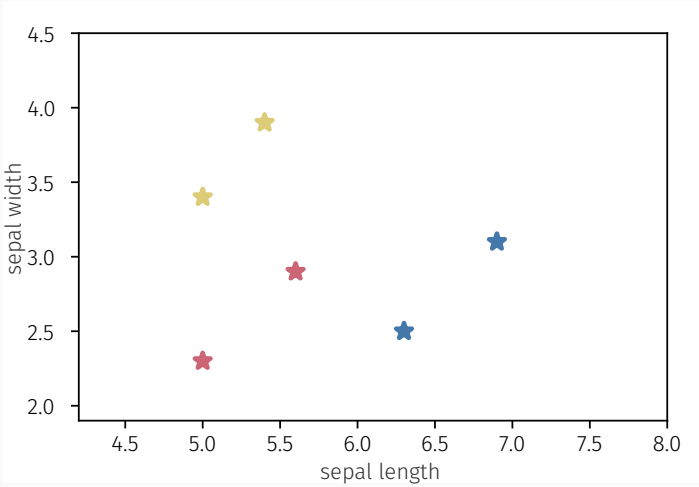


No adaptation needed

Some methods can handle multiple classes

→ classification tree

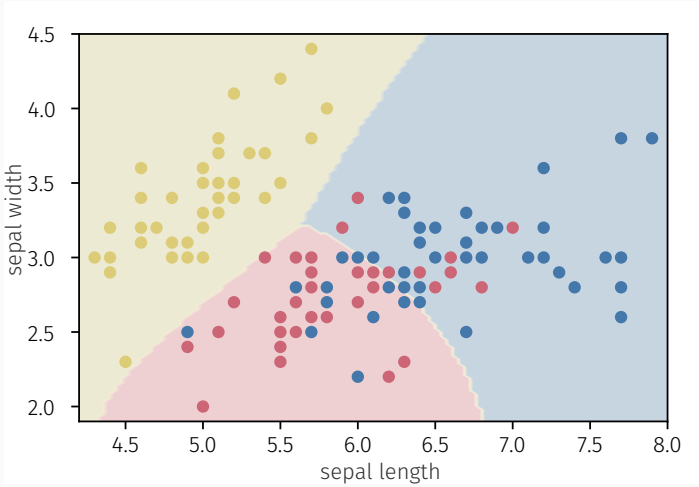


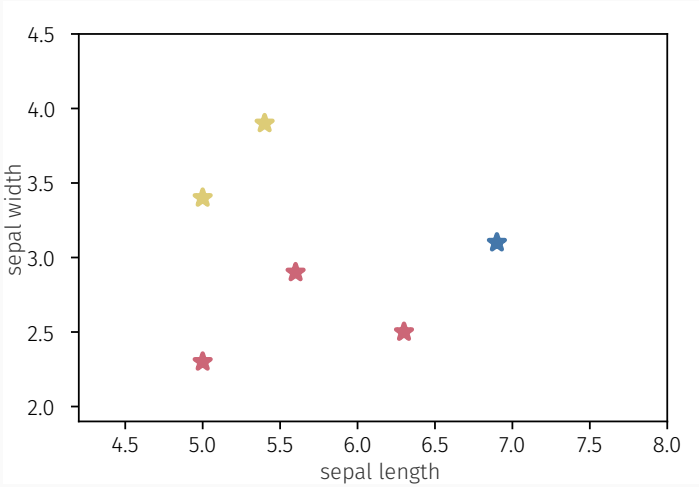


No adaptation needed

Some methods can handle multiple classes

→ Naive bayes





Adaptations needed

Other methods, like the Perceptron and SVMs are naturally designed for the binary scenario

Method-specific adaptations to the multi-class scenario exist

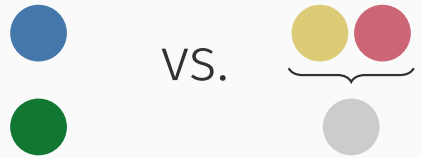
Generic, method-agnostic, meta-frameworks are helpful

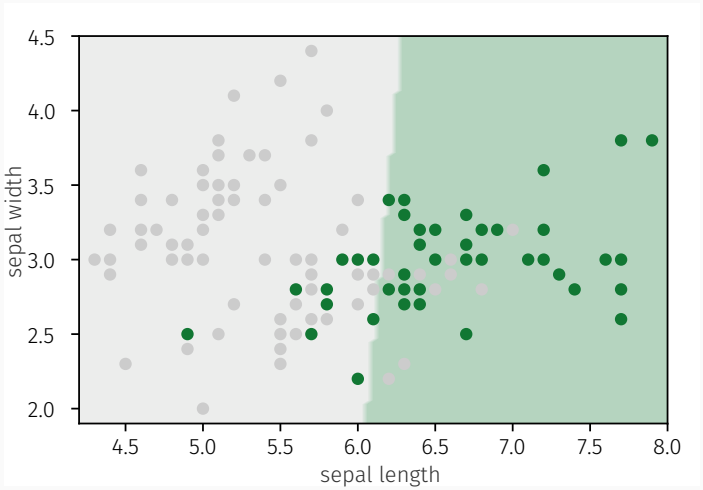
Two main strategies

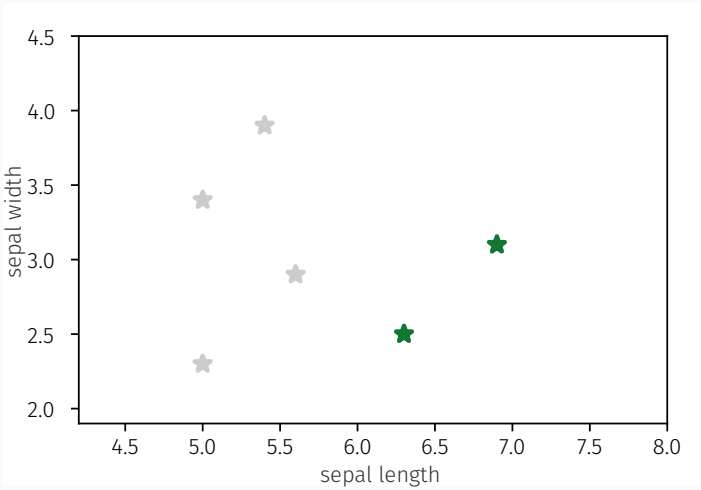
one-against-rest and one-against-one

One-against-rest

Create a new binary classification problem for each class: examples from that class are constitute positive examples the rest are negative examples

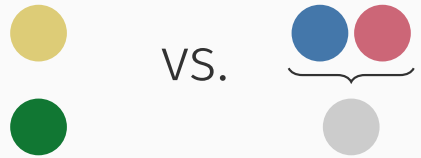


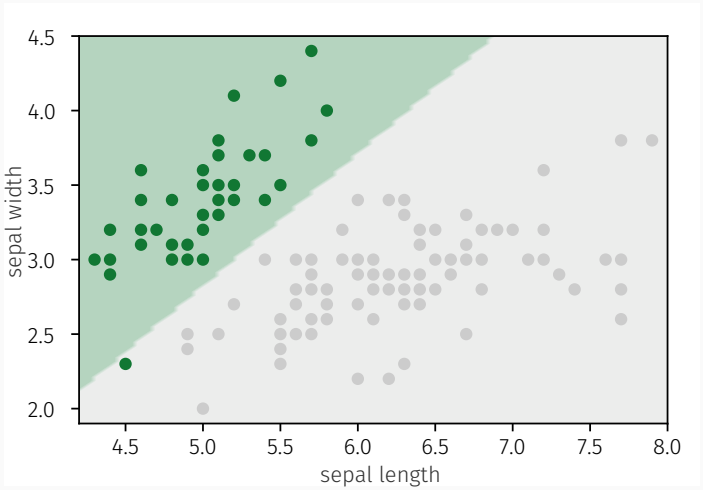


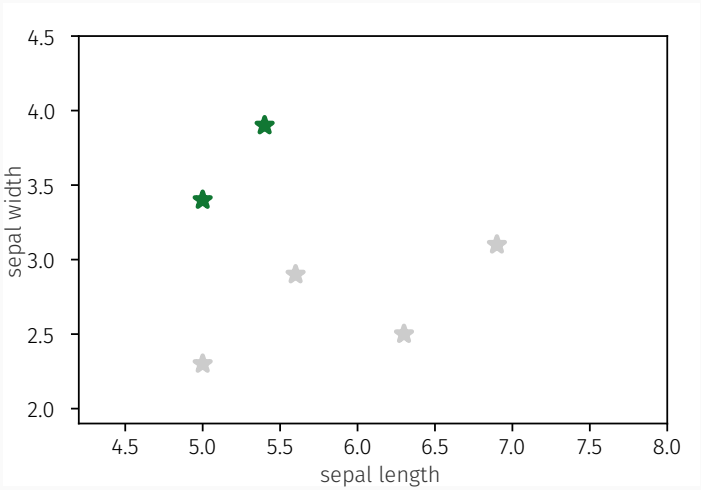


One-against-rest

Create a new binary classification problem for each class: examples from that class are constitute positive examples the rest are negative examples

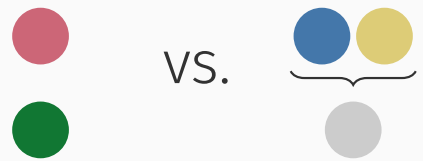


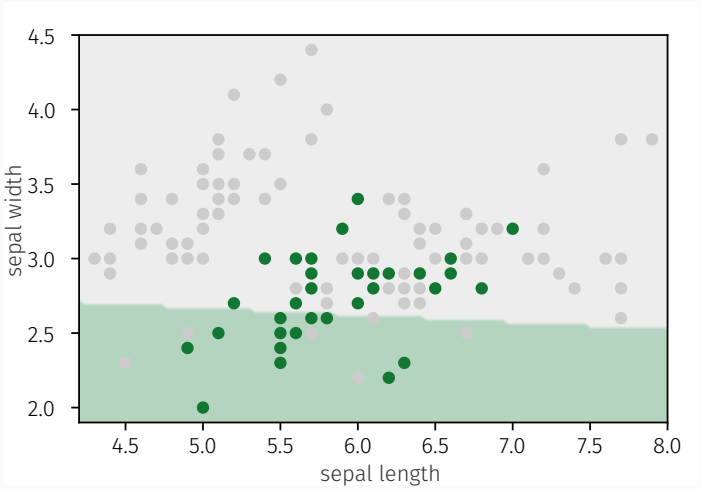


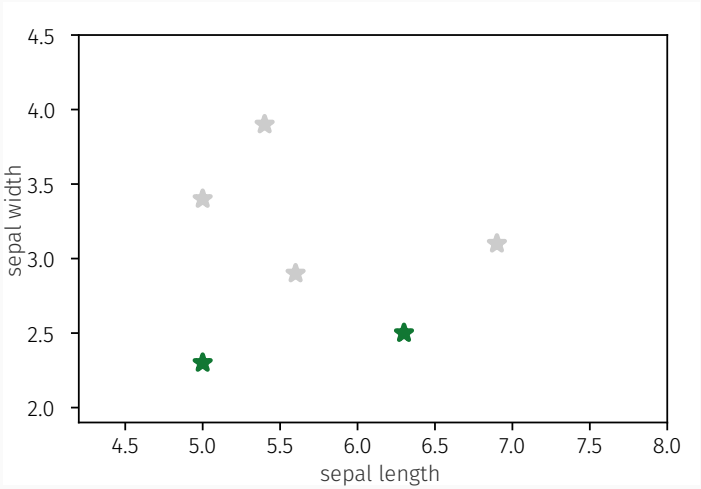


One-against-rest

Create a new binary classification problem for each class: examples from that class are constitute positive examples the rest are negative examples

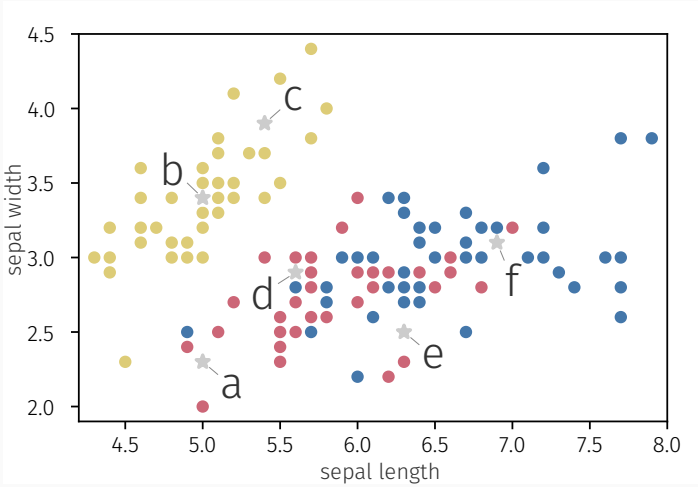






One-against-rest

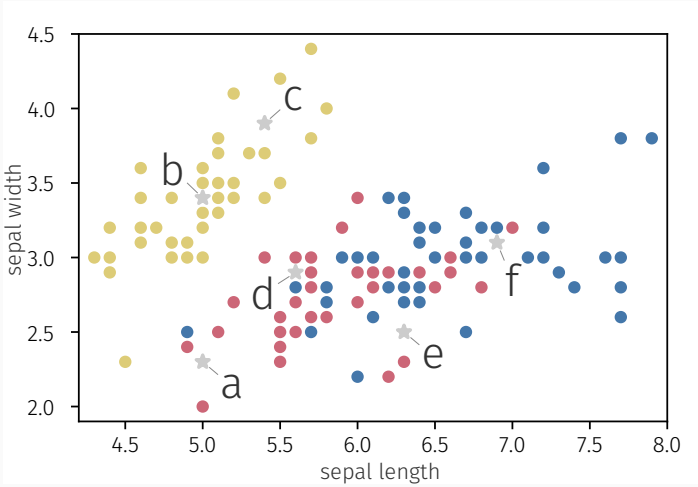
Predictions from the different problems are then combined



	●	●	●
a	☆	☆	★
b	☆	★	☆
c	☆	★	☆
d	☆	☆	☆
e	★	☆	★
f	★	☆	☆

One-against-rest

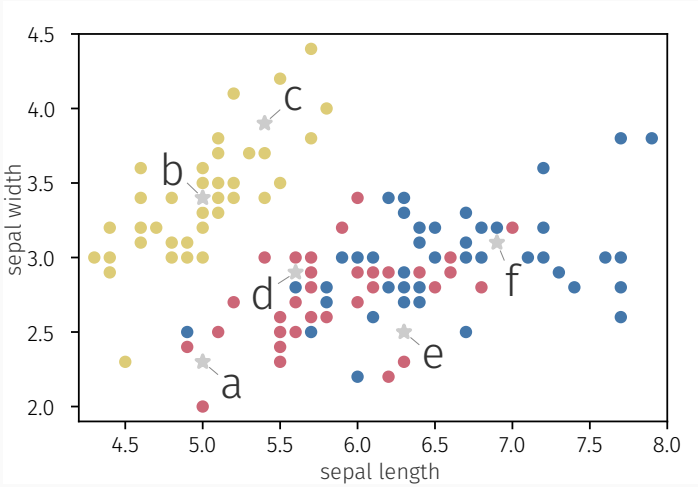
Predictions from the different problems are then combined

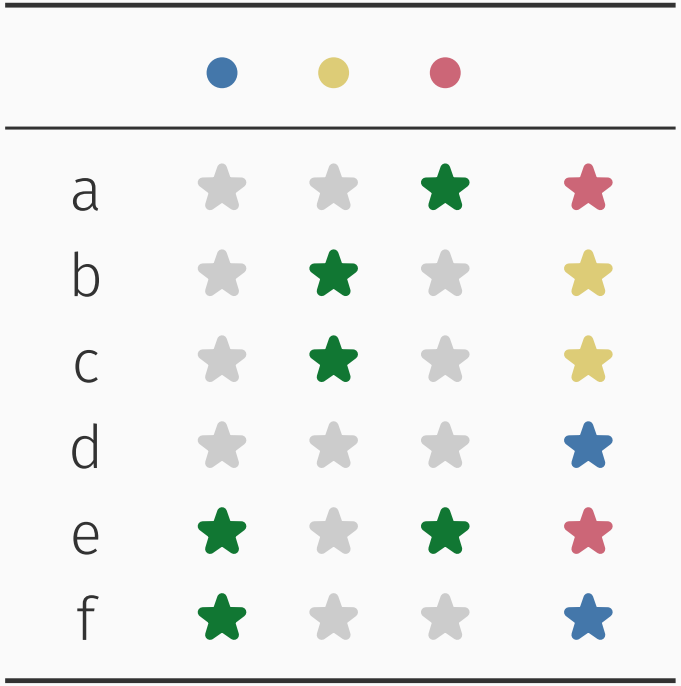


	●	●	●	
a	★	★	★	★
b	★	★	★	★
c	★	★	★	★
d	★	★	★	?
e	★	★	★	?
f	★	★	★	★

One-against-rest

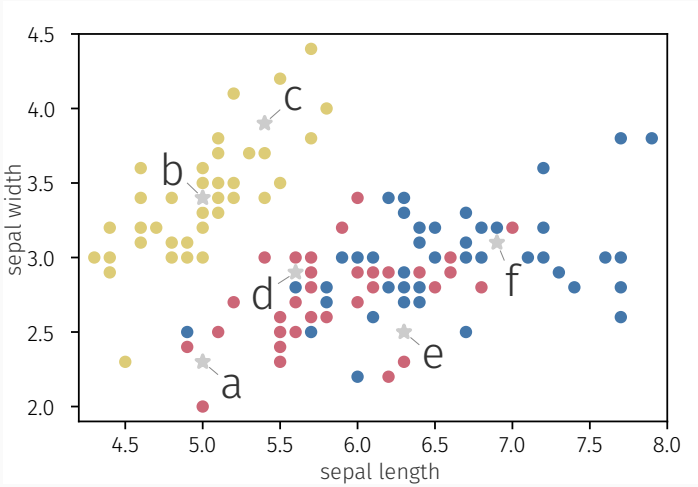
Predictions from the different problems are then combined
Might require tie-breaking,
using weighted rather than crisp votes can help





One-against-rest

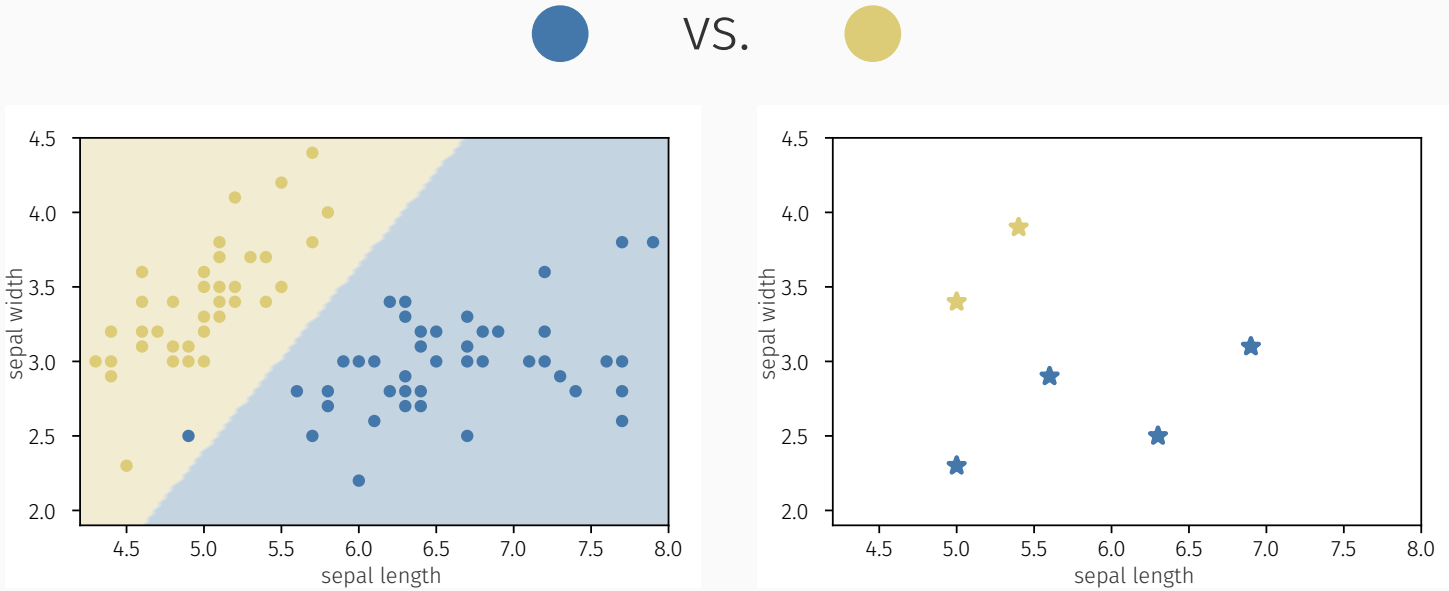
A k class problem maps to k binary models



	●	●	●	
a	★	★	★	★
b	★	★	★	★
c	★	★	★	★
d	★	★	★	?
e	★	★	★	?
f	★	★	★	★

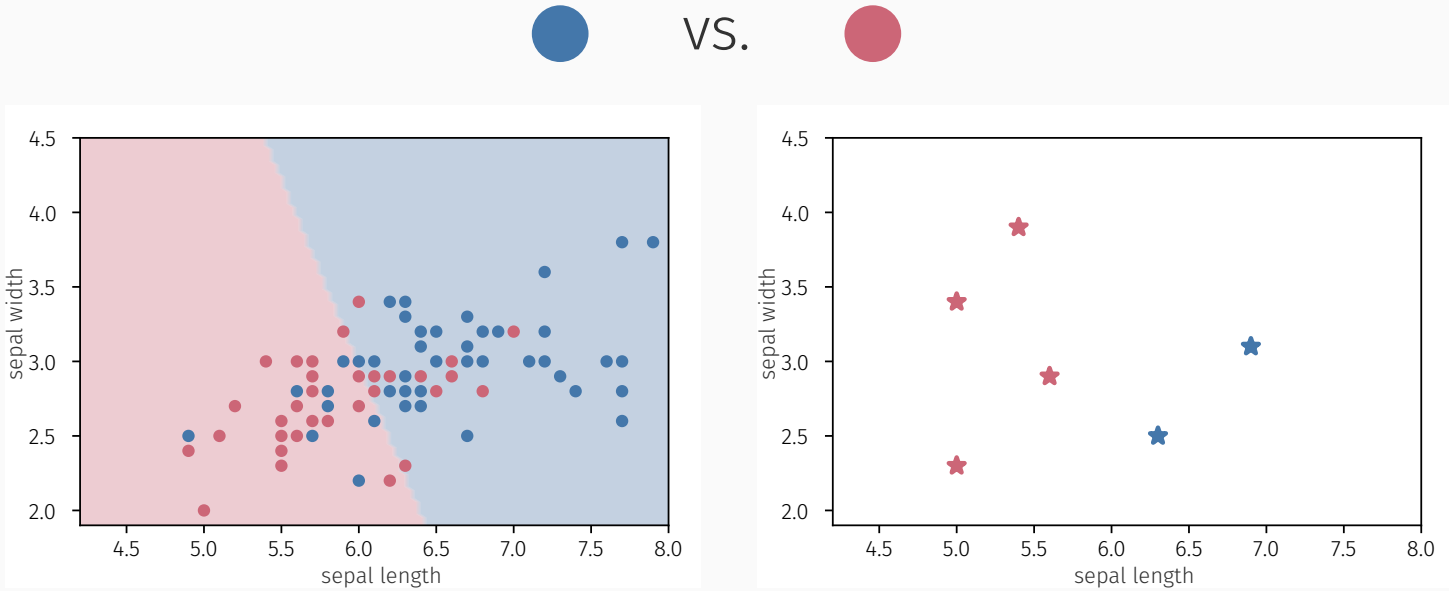
One-against-one

Create a new binary classification problem for each pair of classes, considering only examples from these two classes



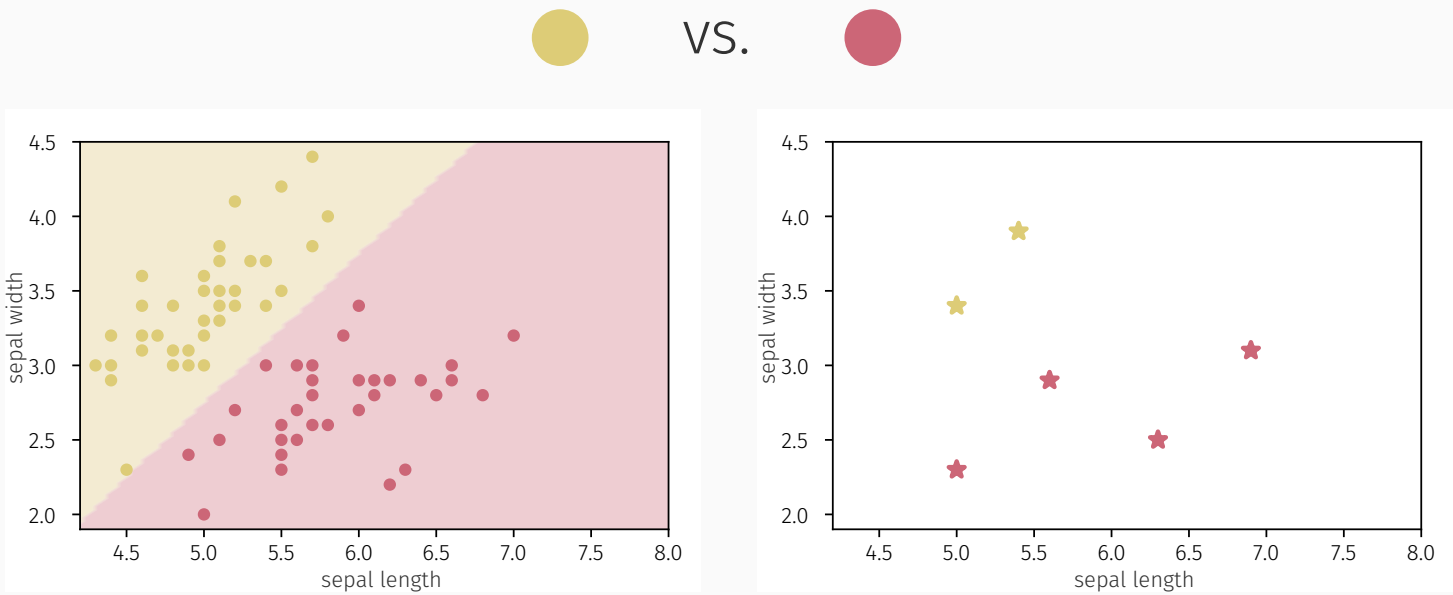
One-against-one

Create a new binary classification problem for each pair of classes, considering only examples from these two classes



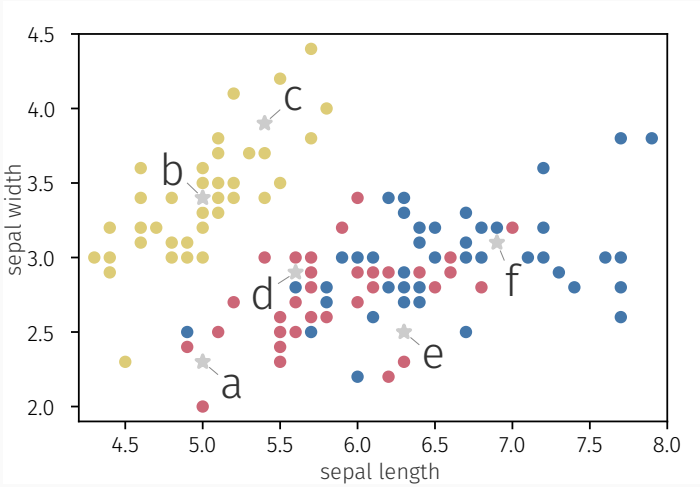
One-against-one

Create a new binary classification problem for each pair of classes, considering only examples from these two classes



One-against-one

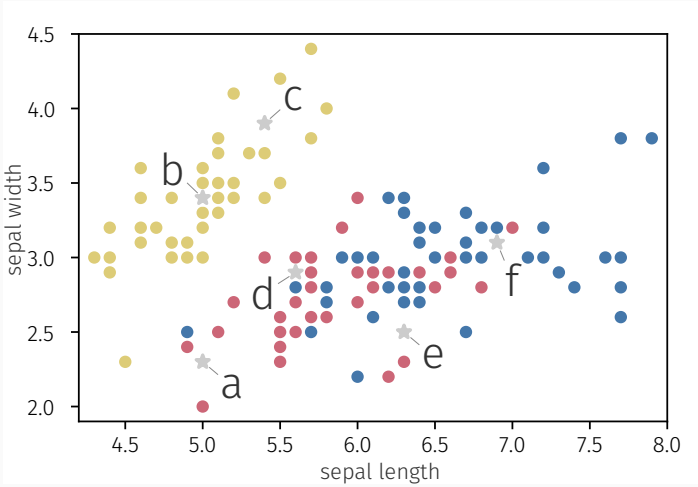
Predictions from the different problems are then combined



	●/●	●/●	●/●
a	★	★	★
b	★	★	★
c	★	★	★
d	★	★	★
e	★	★	★
f	★	★	★

One-against-one

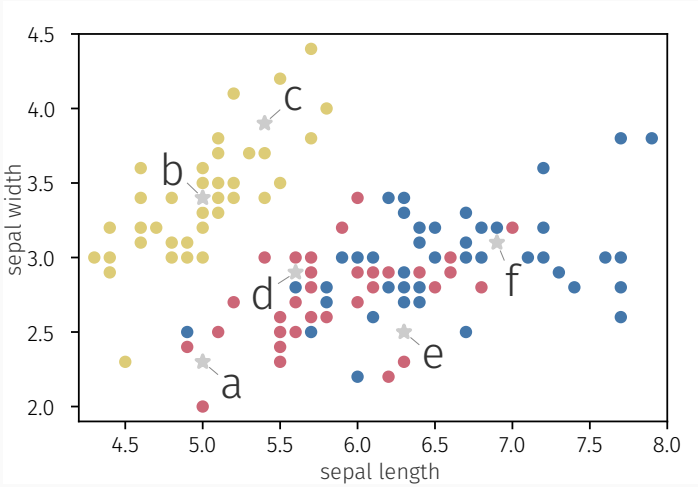
Predictions from the different problems are then combined



	●/●	●/●	●/●	
a	★	★	★	★
b	★	★	★	★
c	★	★	★	★
d	★	★	★	★
e	★	★	★	★
f	★	★	★	★

One-against-one

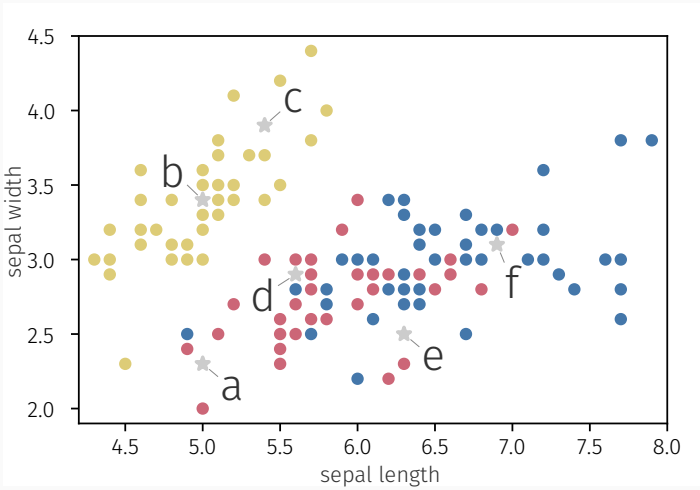
Predictions from the different problems are then combined
Might require tie-breaking,
using weighted rather than crisp votes can help



	●/●	●/●	●/●	
a	★	★	★	★
b	★	★	★	★
c	★	★	★	★
d	★	★	★	★
e	★	★	★	★
f	★	★	★	★

One-against-one

A k class problem maps to $\binom{k}{2} = k(k-1)/2$ binary models

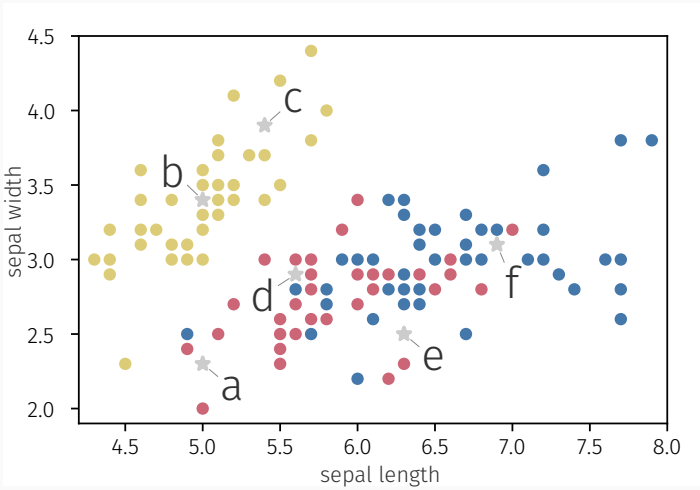


	●/●	●/●	●/●	
a	★	★	★	★
b	★	★	★	★
c	★	★	★	★
d	★	★	★	★
e	★	★	★	★
f	★	★	★	★

One-against-one

A k class problem maps to $\binom{k}{2} = k(k-1)/2$ binary models

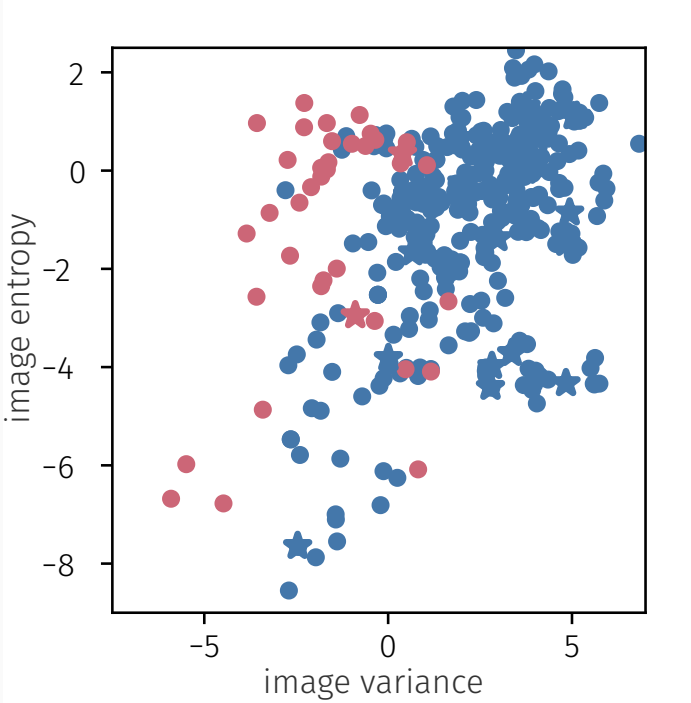
More problems than one-against-rest, but smaller



	●/●	●/●	●/●
a	★	★	★
b	★	★	★
c	★	★	★
d	★	★	★
e	★	★	★
f	★	★	★

Rare-class learning

Fraudulent banknotes detection



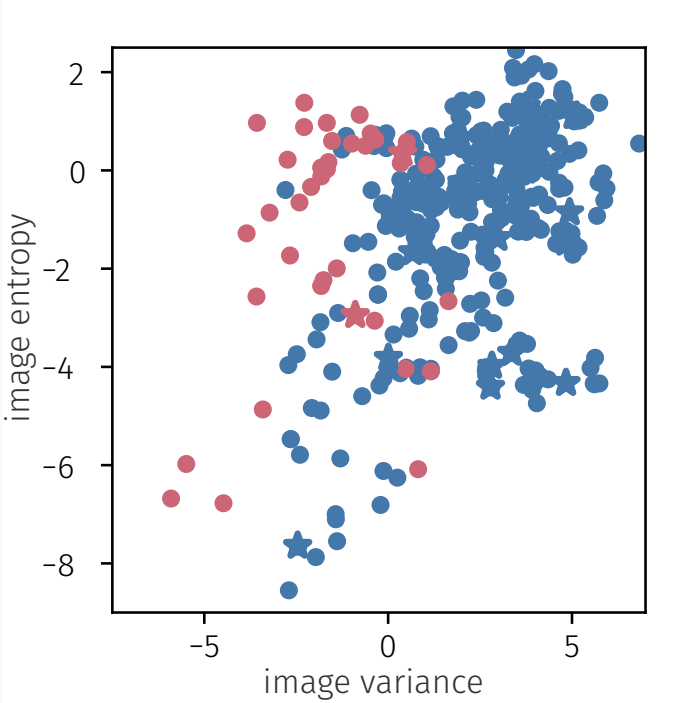
Normal banknotes
are much more common than
fraudulent banknotes
(343 to 37)

Under such class ratio in the test data, trivially predicting everything as normal yields 90% accuracy

False negatives have higher consequences than false positives

Need to emphasize the greater importance of the rare class

Rare-class scenario

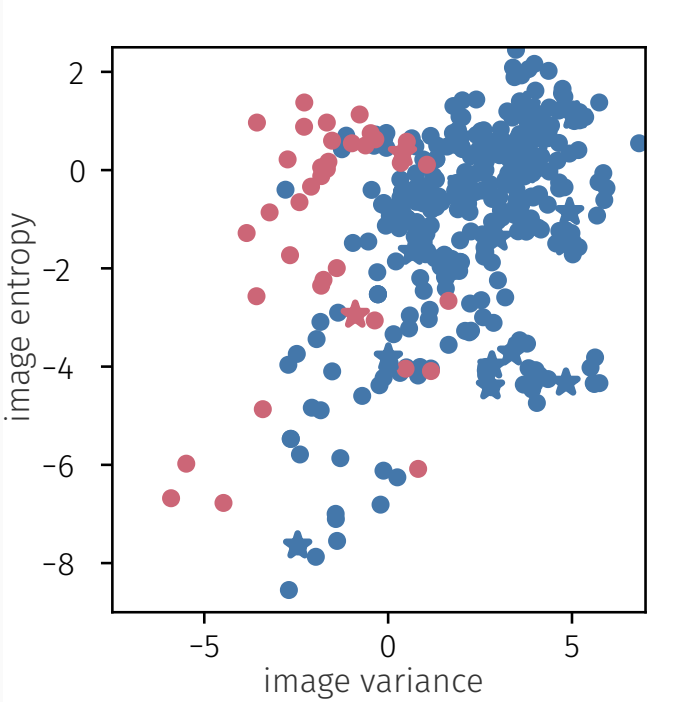


Normal banknotes
are much more common than
fraudulent banknotes
(343 to 37)

It is important to achieve high accuracy on the rare class, at the cost of reduced accuracy on the normal class

Associate different weights to the classes and try to maximize the **weighted accuracy**

Rare-class scenario



Normal banknotes
are much more common than
fraudulent banknotes
(343 to 37)

Two main strategies
example reweighting and example resampling

Rare-class scenario

Example reweighting

- weights are associated to training examples according to their missclassification cost
- algorithms require adaptations to handle these weights

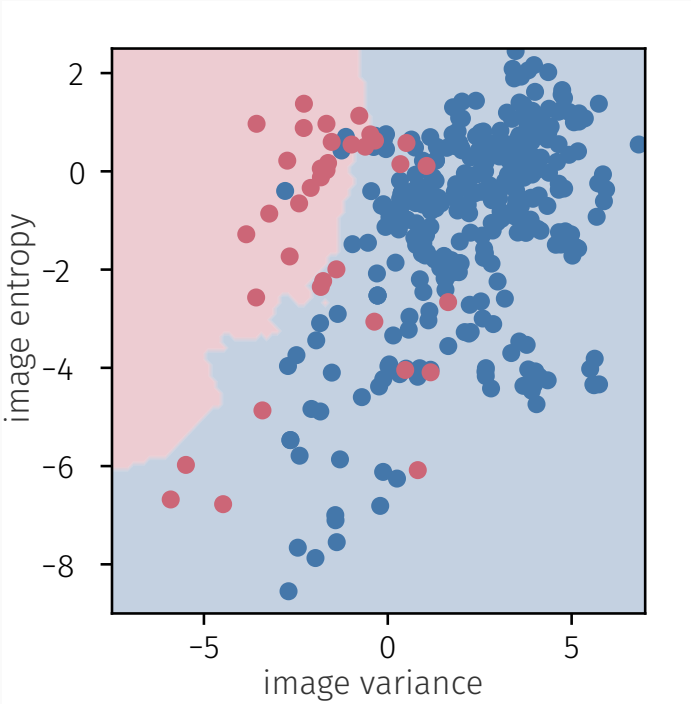
Example resampling

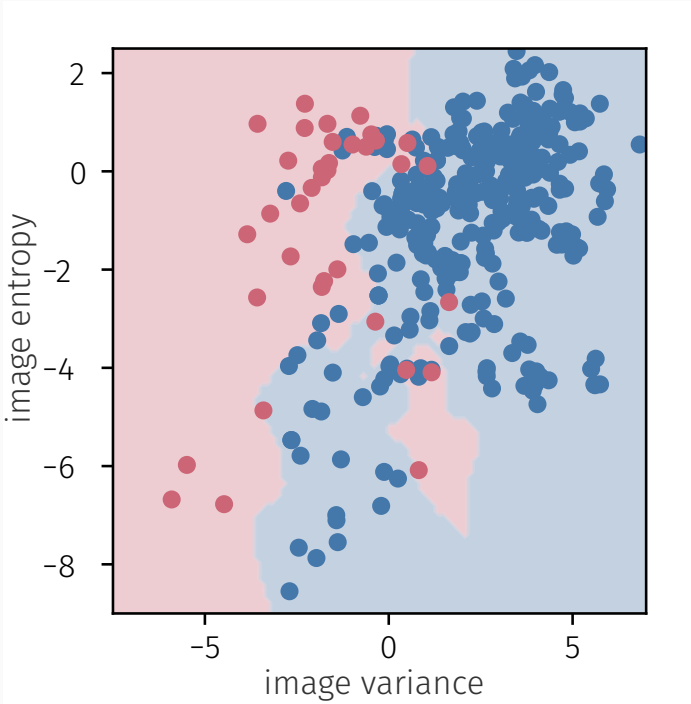
- examples from rare class might be oversampled,
  or examples from normal class be undersampled,
  or a combination of both
- algorithms do not require any adaptation

Rare-class scenario

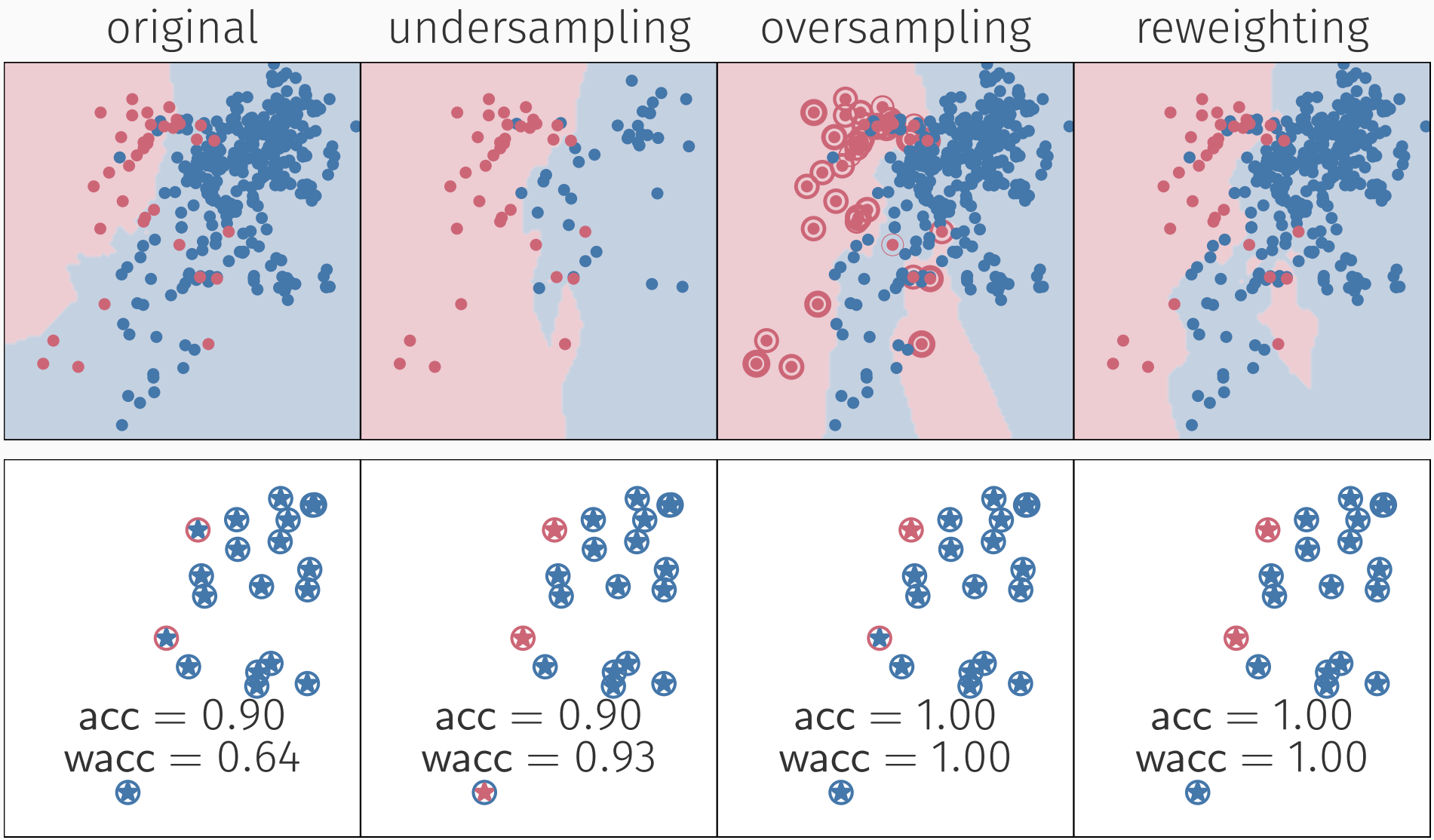
Example reweighting with k nearest neighbors

Identify the k nearest neighbors, assign weights according to their class when deciding majority





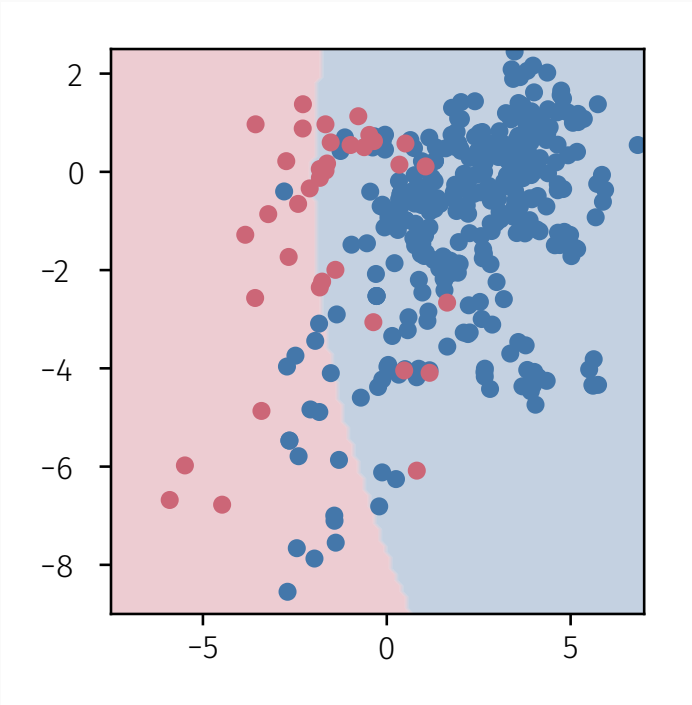
Rare-class scenario with k nearest neighbors

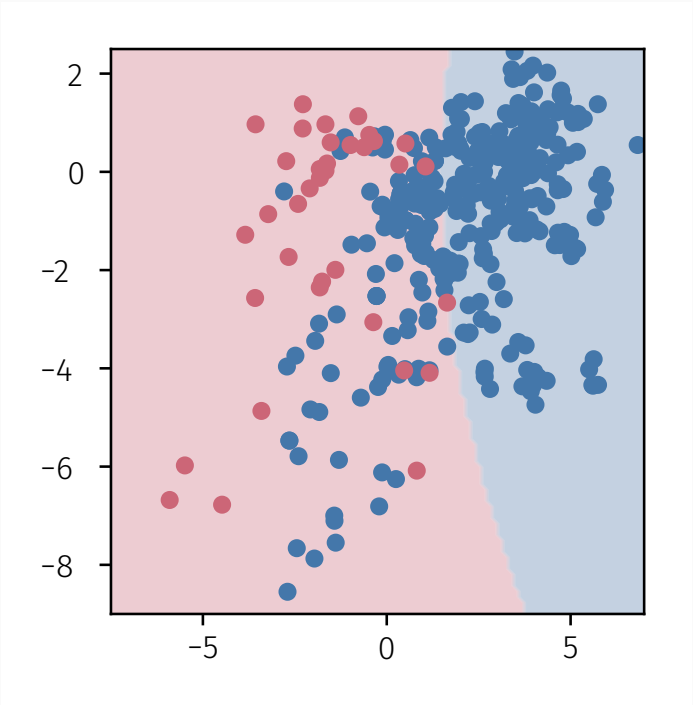


Rare-class scenario

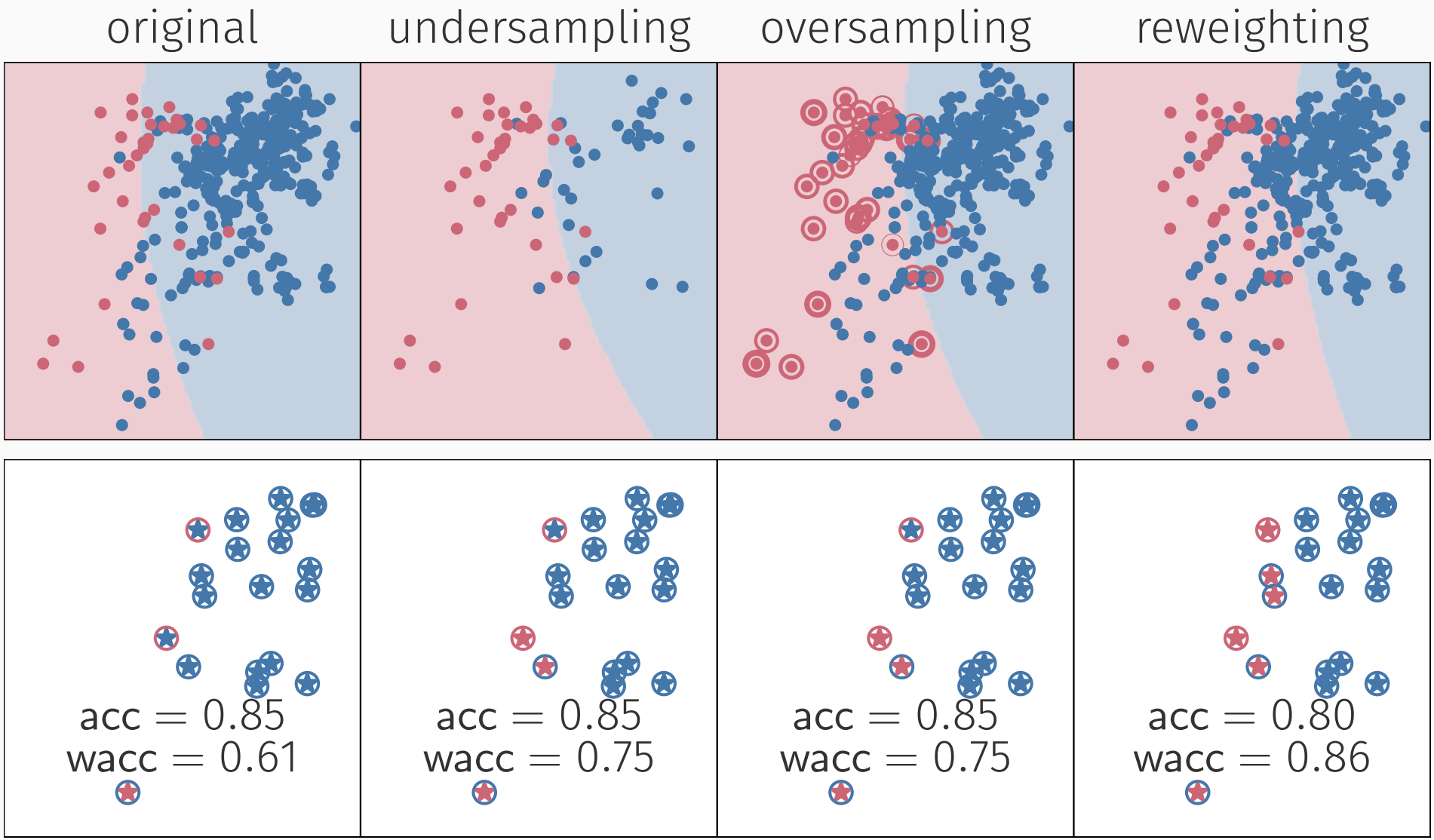
Example reweighting with naives Bayes

Assign weights to instances when computing the classes prior probabilities





Rare-class scenario with Naive Bayes



Rare-class scenario

In effect, **resampling** and **reweighting** are almost equivalent *resampling* can be understood as sampling examples in proportion to their *weights* then treating them equally

Resampling is easier to combine with other approaches

Undersampling is more efficient (smaller datasets)

Resampling has greater randomness
Reweighting is more reliable

Ensemble methods

Ensemble methods

Different classifiers might make different predictions on the same data point due to their specific characteristics or their sensitivity to random artifact in the training data

The aim of ensemble methods is to increase prediction accuracy by combining the results of multiple classifiers

Ensemble methods

For $i = 1, \ldots, \ell$, train model $\mathcal{M}^{(i)}$ on dataset $\mathcal{D}^{(i)}$
Combine the predictions of the different models into a single robust prediction

Data-centered ensembles use a single algorithm on different derivative datasets

Model-centered ensembles use different algorithms or different parameter settings of the same algorithm on a single dataset

Bucket of models

It is often difficult to know beforehand which classifier will work well on a particular dataset

The training dataset is divided into two subsets $\mathcal{D}_A$ and $\mathcal{D}_B$
$\mathcal{D}_A$ is used to train different models
$\mathcal{D}_B$ is used to evaluate their performance
the best model is selected and retrained on the full dataset

Cross-validation can be used for evaluation instead of *hold-out*

The models can correspond to different algorithms or to different parameter settings of the same algorithm

Bucket of models

The performance of the bucket of models is only as good as the best model in the bucket for a particular dataset
Over multiple datasets the approach is able to select the model that is best suited to each case

Bagging

If the variance of a single prediction is σ, the variance of the average of ℓ independent and identically distributed (i.i.d.) such predictions is reduced to σ^2/ℓ

Derivative datasets are created using **bootstrap sampling**
$\mathcal{D}^{(i)}$ is a subset of data points sampled uniformly with replacement from $\mathcal{D}$ to approximately the same size as $\mathcal{D}$

Report the majority vote among the predictions of the models as the ensemble's prediction

Bagging (a.k.a. bootstrapped aggregating) helps **reduce variance** through aggregation
Individual models should be designed so as to reduce bias as much as possible, even at the expense of variance

Bagging

If the variance of a single prediction is σ, the variance of the average of ℓ independent and identically distributed (i.i.d.) such predictions is reduced to σ^2/ℓ

If the predictors have pairwise correlation of ρ between them, the variance of the average prediction is $\rho \cdot \sigma^2 + (1 - \rho)\sigma^2/\ell$ where $\rho \cdot \sigma^2$ is invariant to the number of components in the ensemble and limits the performance gains

Bagging

If the variance of a single prediction is σ, the variance of the average of ℓ independent and identically distributed (i.i.d.) such predictions is reduced to σ^2/ℓ

If the predictors have pairwise correlation of ρ between them, the variance of the average prediction is $\rho \cdot \sigma^2 + (1-\rho)\sigma^2/\ell$ where $\rho \cdot \sigma^2$ is invariant to the number of components in the ensemble and limits the performance gains

! When using **bagging** with decision trees, split choices at the top levels likely remain invariant to bootstrapped sampling
$\rightarrow$ resulting decision trees are correlated
$\rightarrow$ error reduction from aggregation is curtailed

Random forests

A random forest is an ensemble of decision trees where randomness is added explicitely at the split selection to reduce correlation between the components

During tree construction, each split selection is preceded by the random selection of q attributes, among which the split criterion is then chosen, rather than from the entire set of m attributes

Random forests

During tree construction, each split selection is preceded by the random selection of q attributes, among which the split criterion is then chosen, rather than from the entire set of m attributes

Parameter q regulates the amount of randomness

small q leads to more randomness, less correlations across components and more efficient tree growth

large q leads to more accurate individual components

$q = \log_2(m) + 1$ has been show to achieve good trade-off

$q = 1$ (i.e. totally random trees) can achieve good accuracy in aggregation but requires a large number of components

Random forests

During tree construction, each split selection is preceded by the random selection of q attributes, among which the split criterion is then chosen, rather than from the entire set of m attributes
This approach based on *random input selection* is referred to as **Forest-RI**

When m is small this approach does not work well
Instead, generate a subset of q linear combinations of attributes with random coefficients in $[-1, 1]$
This approach based on *random linear combinations* is referred to as **Forest-RC**

Random forests

During tree construction, each split selection is preceded by the random selection of q attributes, among which the split criterion is then chosen, rather than from the entire set of m attributes

Each tree is grown without pruning, on a bootstrapped sample

Restricted split selection increases bias of individual components and leads to problems when the fraction of informative attributes is small

Aggregation provides variance reduction

Random forests are quite resistant to noise and outliers

Boosting

weak learner a classifier that is only slightly correlated with the ground truth, i.e. one that performs only slightly better than random guessing

strong learner a classifier that is arbitrarily well correlated with the ground truth, i.e. one of arbitrarily high accuracy

Hypothesis boosting aims to turn a weak learner into a strong learner

Boosting

Successive models $\mathcal{M}^{(t)}$ are built by applying the same algorithm to weighted variants $\mathcal{D}^{(t)}$ of the dataset

Weights associated to every training instance are adjusted so that the model will focus more on previously missclassified instances

The prediction of the ensemble is a weighted combination of all the models' predictions

Many boosting algorithms have been proposed
AdaBoost (short for Adaptive Boosting) is most popular

AdaBoost

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n,\ i = 1, \ldots, n$

repeat

Train model $\mathcal{M}^{(t)}$ on $\mathcal{D}$ weighted by $\boldsymbol{w}^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} \text{ otherwise} \end{cases}$$

$t \leftarrow t + 1; \quad \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ **or** $\epsilon_{t-1} = 0$ **or** $\epsilon_{t-1} \geq 0.5$

AdaBoost

The algorithm starts with equal weights for all instances

$t \leftarrow 1$; $\quad w_i^{(t)} \leftarrow 1/n,\ i = 1, \ldots, n$

repeat

Train model $\mathcal{M}^{(t)}$ on $\mathcal{D}$ weighted by $\boldsymbol{w}^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} \text{ otherwise} \end{cases}$$

$t \leftarrow t + 1$; $\quad \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ **or** $\epsilon_{t-1} = 0$ **or** $\epsilon_{t-1} \geq 0.5$

AdaBoost

Weights can be incorporated directly to the algorithm or via sampling

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n,\ i = 1, \ldots, n$

repeat

Train model $\mathcal{M}^{(t)}$ on $\mathcal{D}$ weighted by $\boldsymbol{w}^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} \text{ otherwise} \end{cases}$$

$t \leftarrow t + 1; \quad \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ **or** $\epsilon_{t-1} = 0$ **or** $\epsilon_{t-1} \geq 0.5$

AdaBoost

ϵ_t is the fraction of training instances missclassified by $\mathcal{M}^{(t)}$

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n,\ i = 1, \ldots, n$

repeat

Train model $\mathcal{M}^{(t)}$ on $\mathcal{D}$ weighted by $\boldsymbol{w}^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} \text{ otherwise} \end{cases}$$

$t \leftarrow t + 1; \quad \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ **or** $\epsilon_{t-1} = 0$ **or** $\epsilon_{t-1} \geq 0.5$

AdaBoost

The weights of missclassified instances are increased

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n,\ i = 1, \ldots, n$

repeat

Train model $\mathcal{M}^{(t)}$ on $\mathcal{D}$ weighted by $\boldsymbol{w}^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} \text{ otherwise} \end{cases}$$

$t \leftarrow t + 1; \quad \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ **or** $\epsilon_{t-1} = 0$ **or** $\epsilon_{t-1} \geq 0.5$

AdaBoost

The algorithm stops if perfect accuracy is achieved ($\epsilon_t = 0$)
or accuracy is worse than random guessing ($\epsilon_t = 0.5$)
or maximum number of iterations T has been reached

$t \leftarrow 1$; $\quad w_i^{(t)} \leftarrow 1/n$, $i = 1, \ldots, n$

repeat

Train model $\mathcal{M}^{(t)}$ on $\mathcal{D}$ weighted by $\boldsymbol{w}^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} \text{ otherwise} \end{cases}$$

$t \leftarrow t + 1$; $\quad \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ **or** $\epsilon_{t-1} = 0$ **or** $\epsilon_{t-1} \geq 0.5$

AdaBoost

The label for given test instance $\boldsymbol{x}$ is predicted according to

$$\text{sign}\left(\sum_t \alpha_t \, \mathsf{f}_{\mathcal{M}_t}(\boldsymbol{x})\right)$$

i.e. aggregates the weighted predictions of all the models

In some versions of the algorithm, weights are reset to $1/n$ whenever $\epsilon_t \geq 0.5$

In other versions, ϵ_t is allowed to increase beyond 0.5 but the predictions of the corresponding models are effectively inverted by applying negative weights

Boosting

Boosting primarily focuses on **reducing bias**
It aims to combine many weak learners into a strong learner
The approach should be used with simple models having high bias but low variance

When re-weighting is done via sampling, it can also help reduce variance

The approach is vulnerable to noise
It assumes that error is caused by bias, in the presence of noise it will overtrain on low-quality portions of the data

Typically superior to bagging when noise is not excessive

Stacking

The training dataset is divided into two subsets $\mathcal{D}_A$ and $\mathcal{D}_B$
$\mathcal{D}_A$ is used to train ℓ models, the ensemble components
$\mathcal{D}_B$ is used to train a second-level classifier that combines the predictions of the ensemble components

Stacking

$\mathcal{D}_B$ is mapped to a ℓ-dimensional space where each dimension represents the predictions of one ensemble component

original feature space	transformed feature space
training data	
$\mathcal{D}_B$, $n \times m$ matrix	$\mathcal{D}'_B$, $n \times \ell$ matrix
training instance	
$(\mathbf{x}_i, y_i)$	$(\langle \mathrm{f}_{\mathcal{M}_1}(\mathbf{x}_i), \ldots, \mathrm{f}_{\mathcal{M}_\ell}(\mathbf{x}_i) \rangle, y_i)$

A second-level classifier is trained on the transformed training data $\mathcal{D}'_B$, learning to predict class labels from the predictions of the ensemble components

Stacking

The ensemble components can be obtained in various ways, e.g. using ℓ bootstrapped samples $\mathcal{D}_A$ (bagging), ℓ rounds of boosting on $\mathcal{D}_A$, a bucket of ℓ models trained on $\mathcal{D}_A$, etc.

Class probabilities can be used as features instead of the predictions from the ensemble components
Original attributes are often retained in the transformed data

Stacking

Stacking can be combined with m-fold cross-validation

A new representation is obtained for each instance of the training data, where the features are obtained from the ℓ first-level classifiers trained on the $(m-1)$ folds that do not contain that instance

The second-level classifier is trained on this dataset representing *all* training instances

The first-level classifiers are re-trained on the full training data

Stacking

By learning from the errors of the ensemble components stacking allows to reduce both bias and variance

The power of stacking comes from the flexible learning approach of the combiner
Many other ensemble methods can be seen as special cases using less flexible, data-independent, combination procedures such as voting