Introduction to Algorithmic Data Analysis

Esther Galbrun

Autumn 2023

UNIVERSITY OF
EASTERN FINLAND

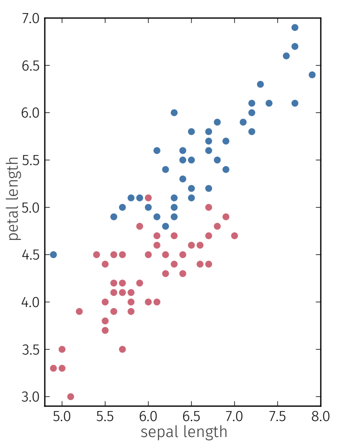
Last updated on 2023-07-04 11:01

Part III

Classification Basics

Problem

A simple example

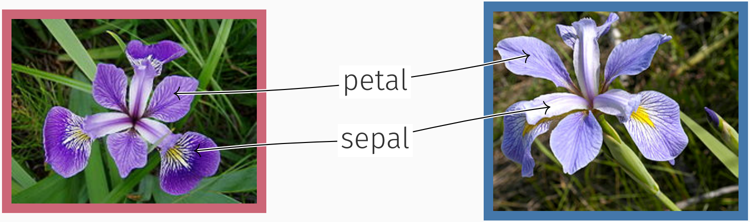


A dataset with two classes

data points: Iris flowers

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

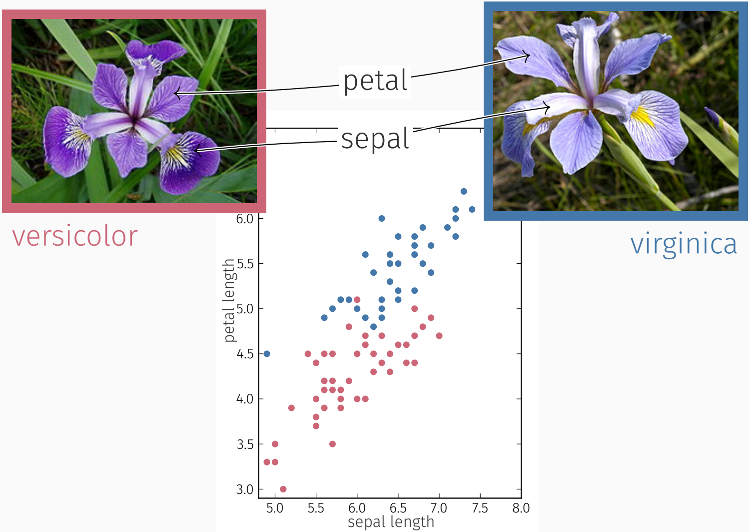
class: species, *versicolor* vs. *virginica*



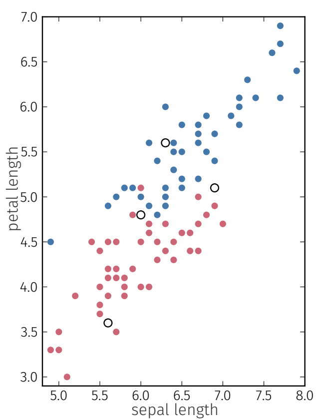
versicolor

virginica

A simple example



A simple example



?

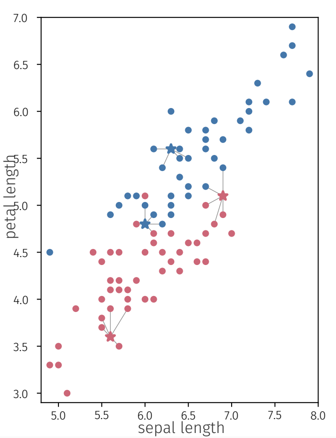
Class information, i.e. species, is absent for some points

Can we use the available information to predict it?

Different methods

Look at the most similar data points

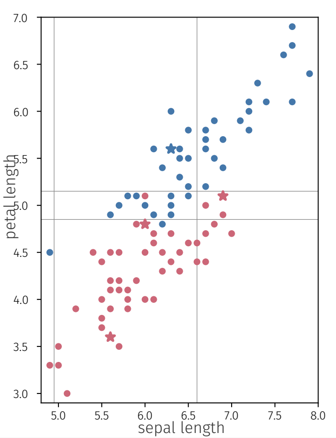
→ k nearest neighbors (k-NN)

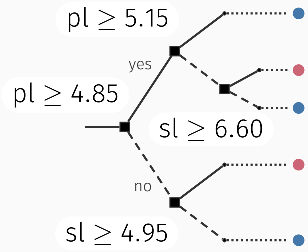


majority class
among k nearest neighbors

Apply a sequence of tests on attributes' values

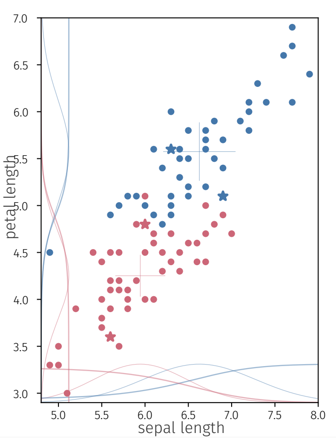
$\longrightarrow$ classification tree





Look at class probabilities conditioned on attributes' values

$\longrightarrow$ Naive bayes

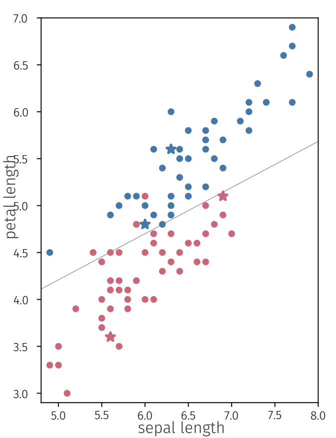


$$P(c \mid \mathrm{sl}, \mathrm{sp}) \propto P(c) \cdot P(\mathrm{sl} \mid c) \cdot P(\mathrm{sp} \mid c)$$

$$P(\bullet \mid \mathrm{sl}, \mathrm{sp}) \; > \; P(\bullet \mid \mathrm{sl}, \mathrm{sp}) \quad \bullet$$

$$P(\bullet \mid \mathrm{sl}, \mathrm{sp}) \; \leq \; P(\bullet \mid \mathrm{sl}, \mathrm{sp}) \quad \bullet$$

Look at the sign of a linear combination of the attributes
→ perceptron



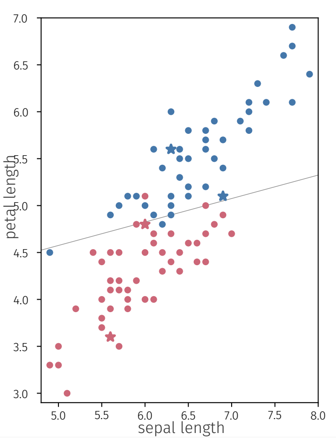
$$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 < 0$$

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

Different methods

Look at the sign of a linear combination of the attributes

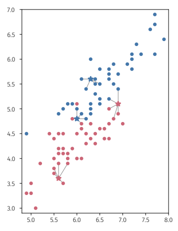
→ support vector machine (SVM)



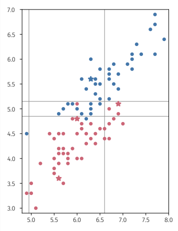
$$\mathrm{sl} - 4 \cdot \mathrm{pl} + 13.3 < 0$$

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

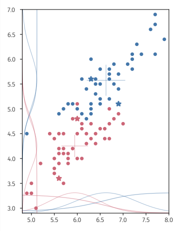
k-NN



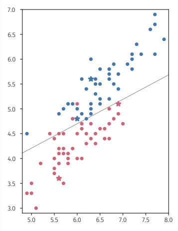
decision tree



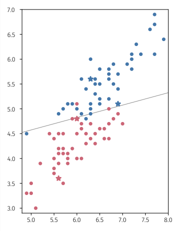
naive Bayes



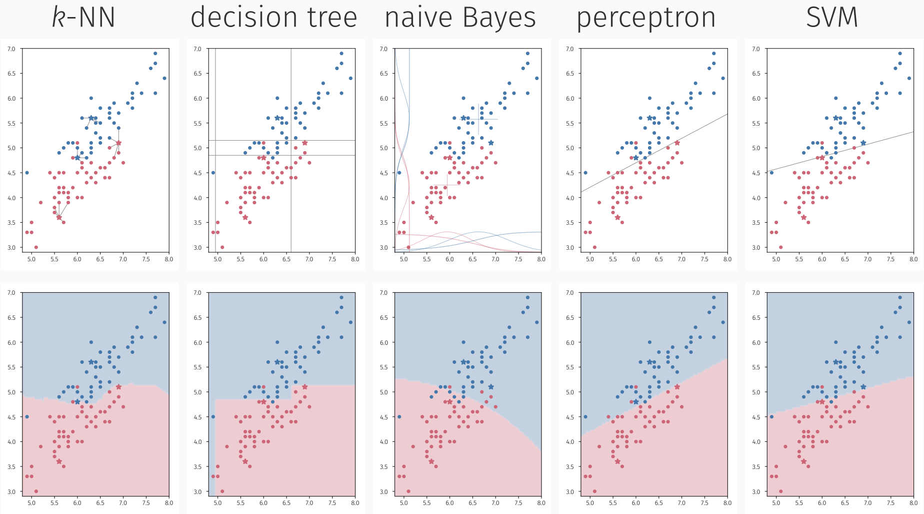
perceptron



SVM



...but it is all about learning a decision boundary



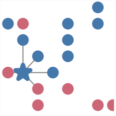
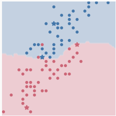
Some notations

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

Methods

Input: data set $\mathcal{D}$, data point x

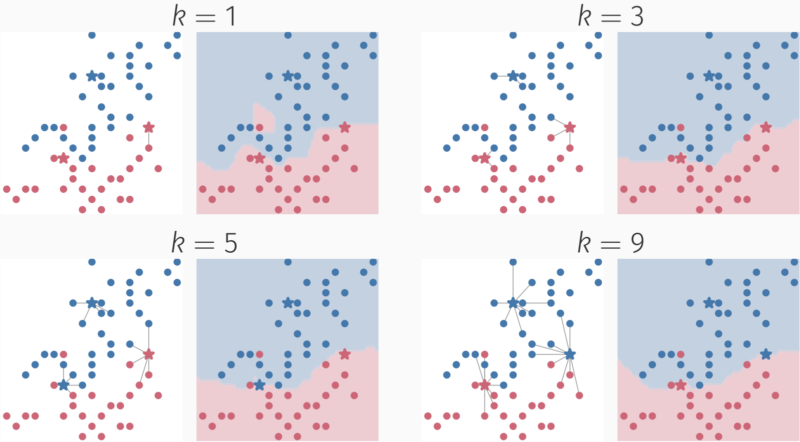
Parameters: distance function d, number of neighbors k

No training

Prediction: return majority class among k points in $\mathcal{D}$ that minimize $\mathsf{d}(x, x')$

k nearest neighbors

$\mathcal{K} \leftarrow \{k \text{ points } \boldsymbol{x}' \in \mathcal{D} \text{ that minimize } \mathrm{d}(\boldsymbol{x}, \boldsymbol{x}')\}$
return majority class in $\mathcal{K}$



k nearest neighbors

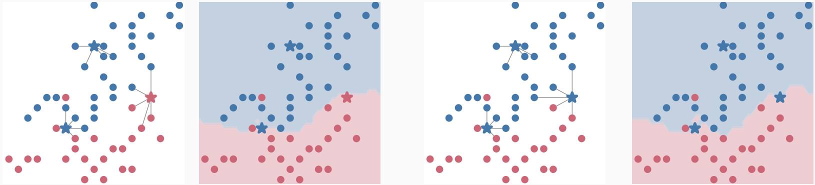
$\mathcal{K} \leftarrow \{k$ points $\mathbf{x}' \in \mathcal{D}$ that minimize $\mathrm{d}(\mathbf{x}, \mathbf{x}')\}$
return majority class in $\mathcal{K}$

Euclidean distance (ℓ_2 norm)

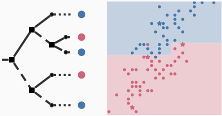
$$\mathrm{d}(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{i=m} (x_i - x_i')^2}$$

Manhattan distance (ℓ_1 norm)

$$\mathrm{d}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{i=m} \left| x_i - x_i' \right|$$



Decision tree



Input: data set $\mathcal{D}$, data point x

Parameters: split evaluation measure, max depth d, min leaf size l

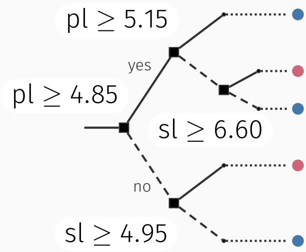
Training: construct tree T by recursively finding tests that yield best splits in $\mathcal{D}$

Prediction: apply sequence of tests from T to x until reaching a leaf, return associated class

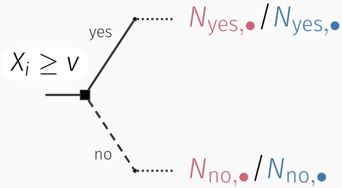
Decision tree: structure representing a succession of tests and possible classification or regression outcomes

Leaf node decision (class/value)

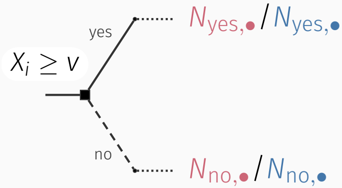
Other node test on an attribute's value



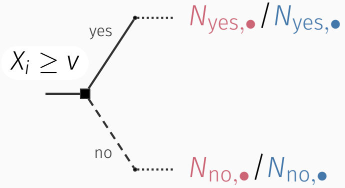
Decision tree: Split evaluation measures



Decision tree: Split evaluation measures



	$\bullet$	$\bullet$	
yes	$N_{\text{yes},\bullet}$	$N_{\text{yes},\bullet}$	N_{yes}
no	$N_{\text{no},\bullet}$	$N_{\text{no},\bullet}$	N_{no}
	$N_{\bullet}$	$N_{\bullet}$	N



	${\color{red}\bullet}$	${\color{blue}\bullet}$	
yes	$N_{\text{yes},{\color{red}\bullet}}$	$N_{\text{yes},{\color{blue}\bullet}}$	N_{yes}
no	$N_{\text{no},{\color{red}\bullet}}$	$N_{\text{no},{\color{blue}\bullet}}$	N_{no}
	$N_{{\color{red}\bullet}}$	$N_{{\color{blue}\bullet}}$	N

error rate

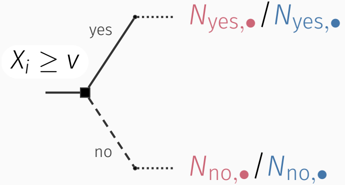
$$\sum_{\text{b}\in\{\text{yes},\text{no}\}} \frac{1}{N}\left(N_{\text{b}} - \max_{c\in\{{\color{red}\bullet},{\color{blue}\bullet}\}} N_{\text{b},c}\right)$$

Gini index

$$\sum_{\text{b}\in\{\text{yes},\text{no}\}} \frac{N_{\text{b}}}{N}\left(1 - \sum_{c\in\{{\color{red}\bullet},{\color{blue}\bullet}\}} \left(\frac{N_{\text{b},c}}{N_{\text{b}}}\right)^2\right)$$

entropy

$$\sum_{\text{b}\in\{\text{yes},\text{no}\}} \frac{N_{\text{b}}}{N} \sum_{c\in\{{\color{red}\bullet},{\color{blue}\bullet}\}} -\frac{N_{\text{b},c}}{N_{\text{b}}} \log_2\left(\frac{N_{\text{b},c}}{N_{\text{b}}}\right)$$

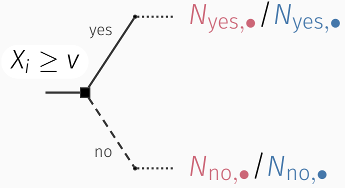


	$\textcolor{red}{\bullet}$	$\textcolor{blue}{\bullet}$	
yes	$N_{\text{yes},\textcolor{red}{\bullet}}$	$N_{\text{yes},\textcolor{blue}{\bullet}}$	N_{yes}
no	$N_{\text{no},\textcolor{red}{\bullet}}$	$N_{\text{no},\textcolor{blue}{\bullet}}$	N_{no}
	$N_{\textcolor{red}{\bullet}}$	$N_{\textcolor{blue}{\bullet}}$	N

information gain

$$\sum_{c \in \{\textcolor{red}{\bullet},\textcolor{blue}{\bullet}\}} -\frac{N_c}{N} \log_2\left(\frac{N_c}{N}\right) - \sum_{b \in \{\text{yes},\text{no}\}} \frac{N_b}{N} \sum_{c \in \{\textcolor{red}{\bullet},\textcolor{blue}{\bullet}\}} -\frac{N_{b,c}}{N_b} \log_2\left(\frac{N_{b,c}}{N_b}\right)$$

Decision tree: Split evaluation measures

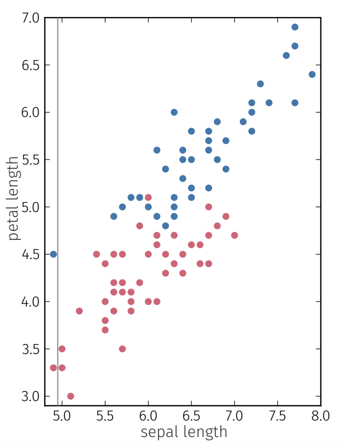


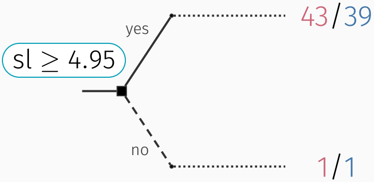
	$\bullet$	$\bullet$	
yes	$N_{yes,\bullet}$	$N_{yes,\bullet}$	N_{yes}
no	$N_{no,\bullet}$	$N_{no,\bullet}$	N_{no}
	$N_{\bullet}$	$N_{\bullet}$	N

	10 0	9 1	7 3	5 5	0 10	5 2	7 2
	0 10	1 9	3 7	5 5	10 0	5 8	3 8
ER	0.000	0.100	0.300	0.500	0.000	0.350	0.250
G	0.000	0.180	0.420	0.500	0.000	0.451	0.374
E	0.000	0.469	0.881	1.000	0.000	0.927	0.809
IG	1.000	0.531	0.119	0.000	1.000	0.073	0.191

Decision tree: Training

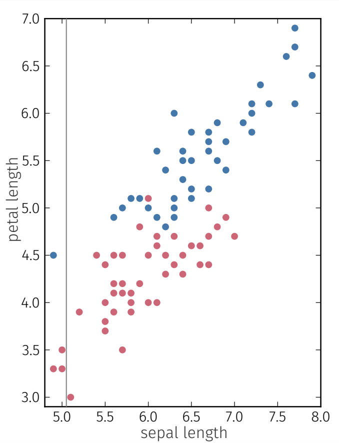
Try splitting on different attributes and values

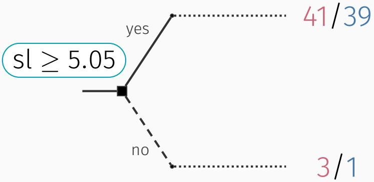




$G = 0.498$

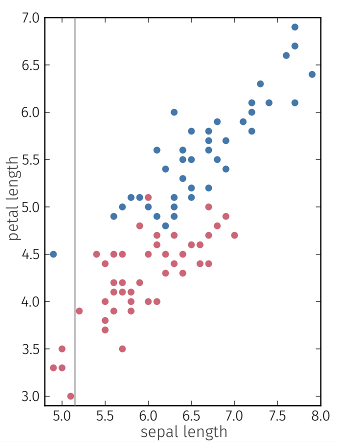
Try splitting on different attributes and values

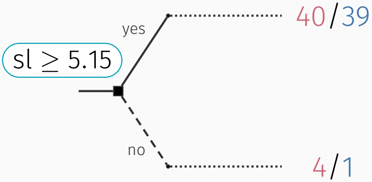




$G = 0.493$

Try splitting on different attributes and values

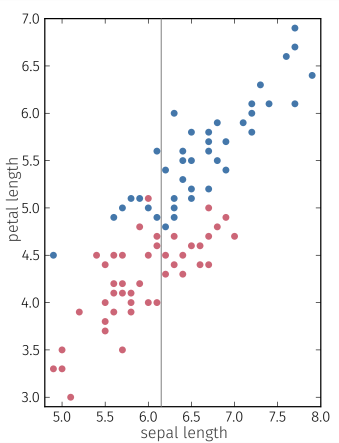


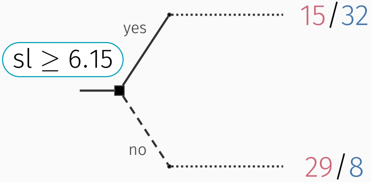


$G = 0.489$

Decision tree: Training

Try splitting on different attributes and values

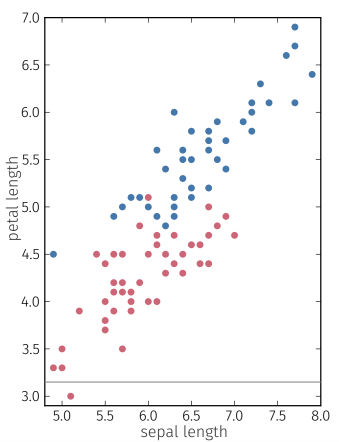


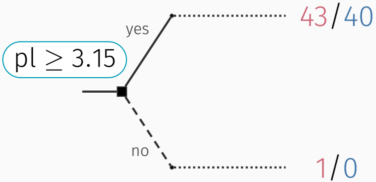


G = 0.392

Decision tree: Training

Try splitting on different attributes and values

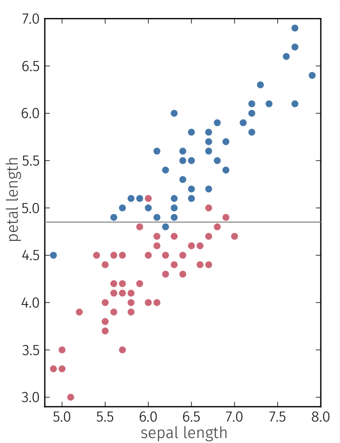


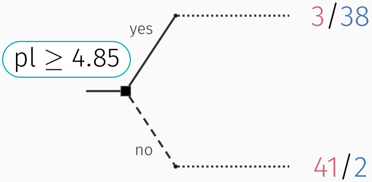


$G = 0.493$

Decision tree: Training

Try splitting on different attributes and values

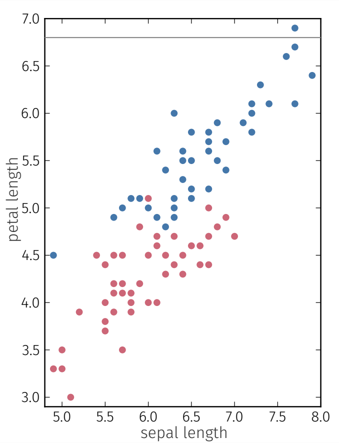


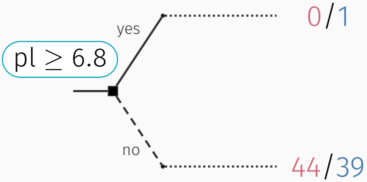


$G = 0.111$

Decision tree: Training

Try splitting on different attributes and values

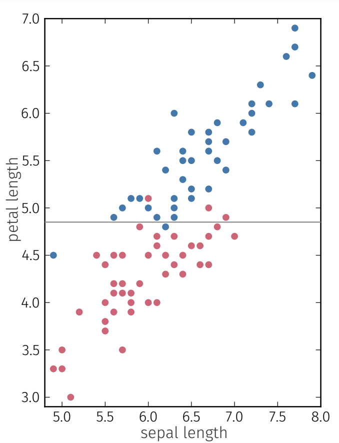


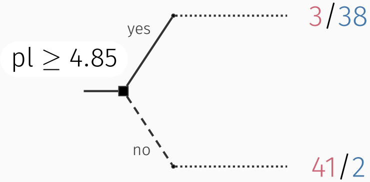


$G = 0.492$

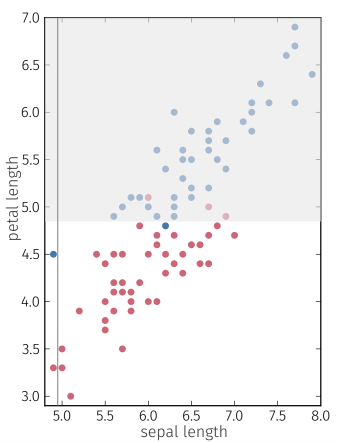
Decision tree: Training

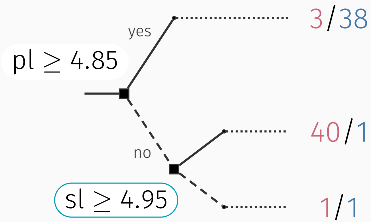
Select best split,
divide the data accordingly and recurse on the subsets





Considering only the 'no' branch,
try splitting on different attributes and values

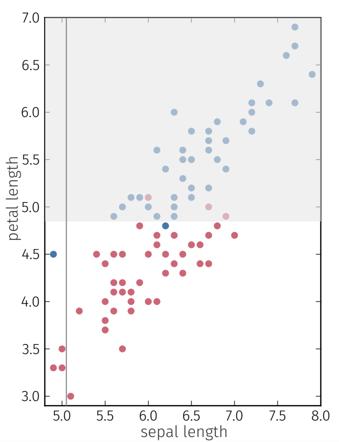


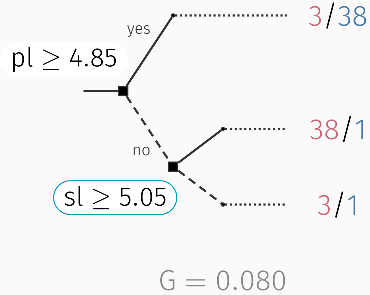


$G = 0.068$

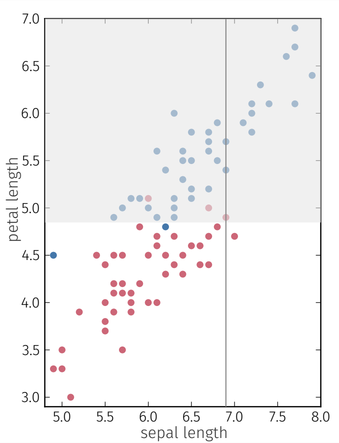
Decision tree: Training

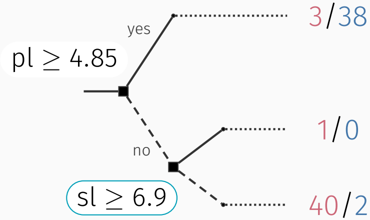
Considering only the ‘no’ branch,
try splitting on different attributes and values





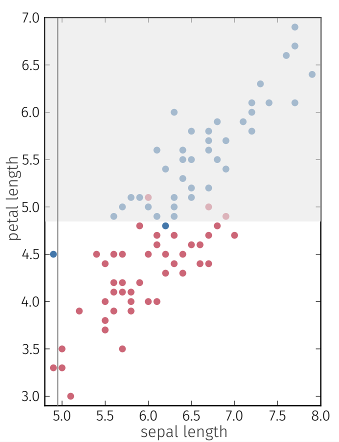
Considering only the ‘no’ branch,
try splitting on different attributes and values

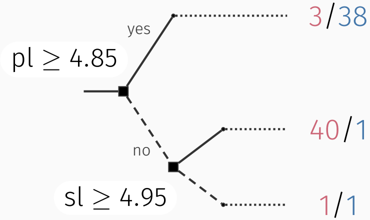




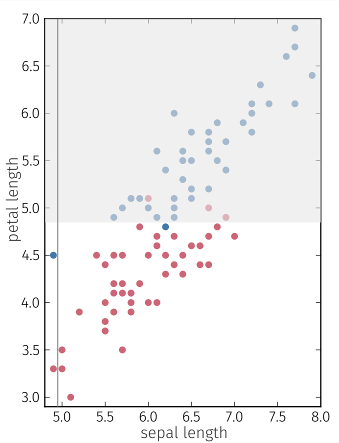
$G = 0.088$

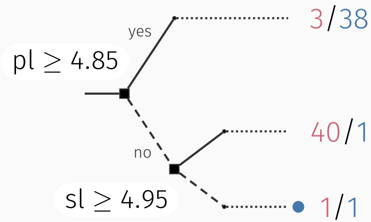
Considering only the 'no' branch,
select best split...





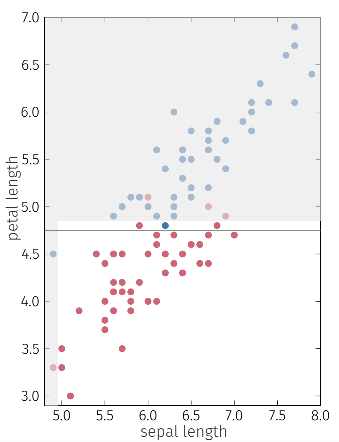
Node is below the minimum size,
add leaf with dominant class as decision

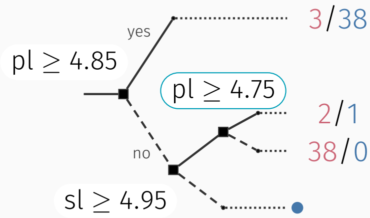




Decision tree: Training

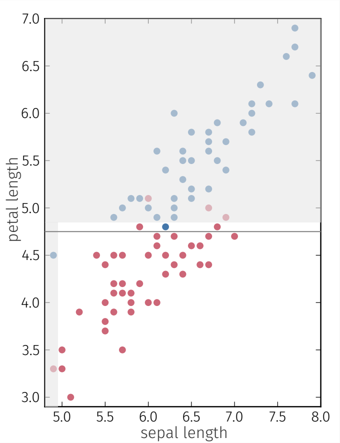
Considering only the current branch,
try splitting on different attributes and values

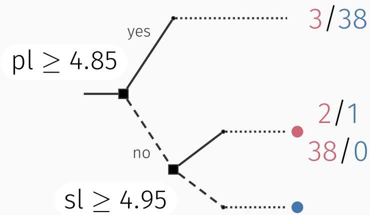




$G = 0.032$

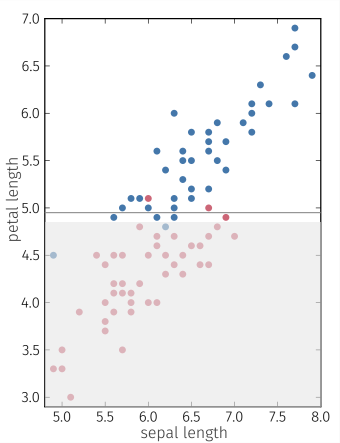
No improving split can be found,
add leaf with dominant class as decision

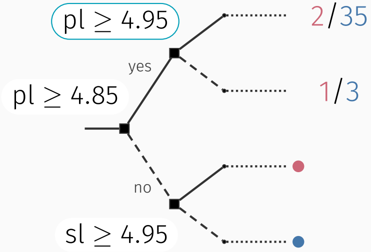




Decision tree: Training

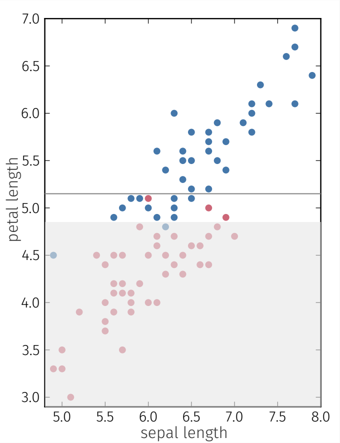
Considering only the 'yes' branch,
try splitting on different attributes and values

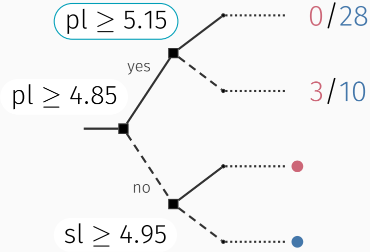




$G = 0.128$

Considering only the 'yes' branch,
try splitting on different attributes and values

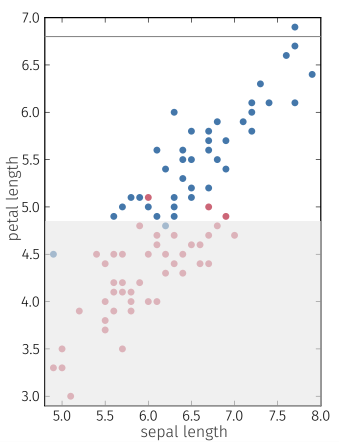


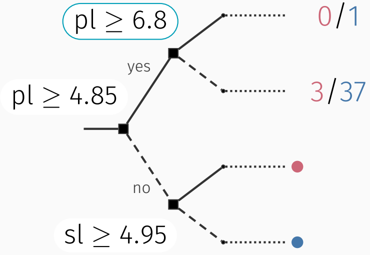


$G = 0.112$

Decision tree: Training

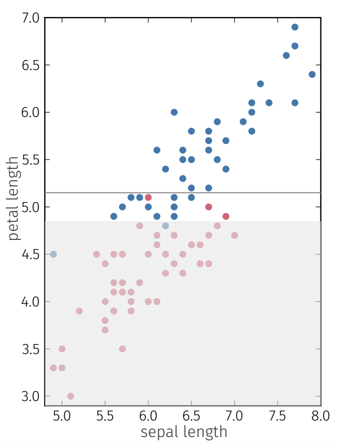
Considering only the ‘yes’ branch,
try splitting on different attributes and values

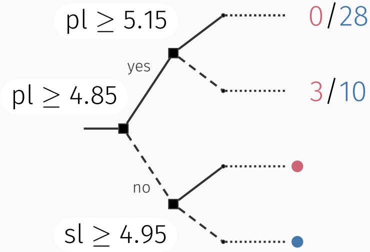




$G = 0.135$

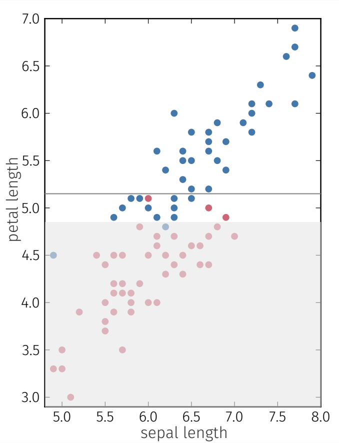
Considering only the ‘yes’ branch, select best split...

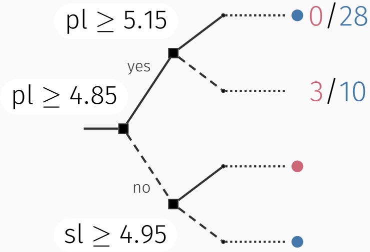




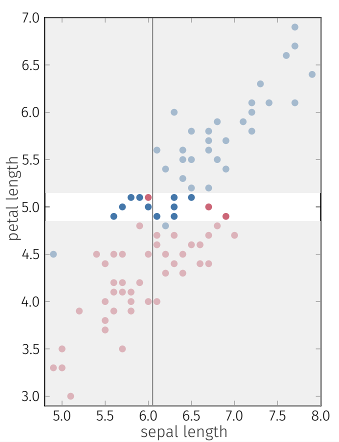
Decision tree: Training

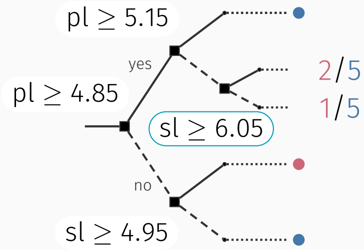
Node is pure,
add leaf with class as decision





Considering only the current branch,
try splitting on different attributes and values

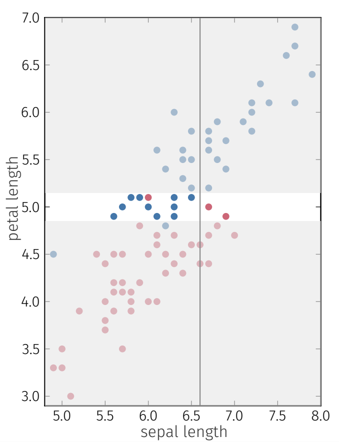


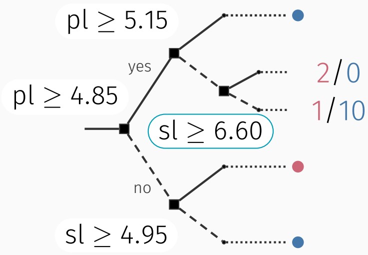


$G = 0.347$

Decision tree: Training

Considering only the current branch,
try splitting on different attributes and values

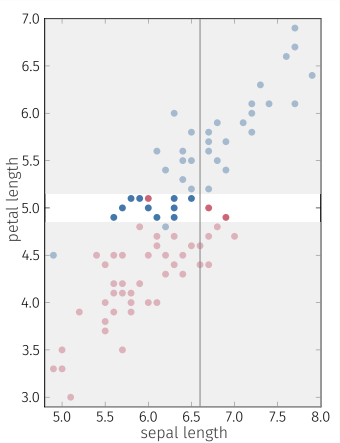


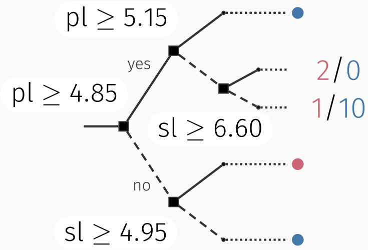


$G = 0.139$

Decision tree: Training

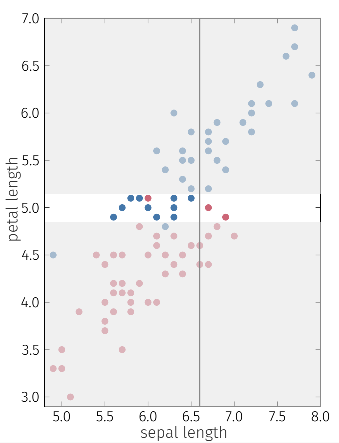
Considering only the current branch,
select best split...

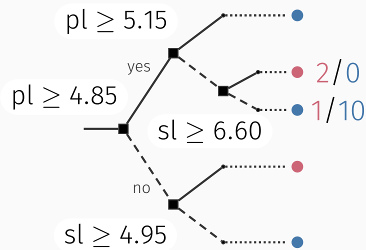




Decision tree: Training

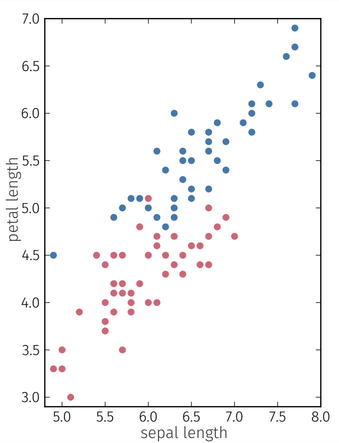
Maximum depth has been reached,
add leaves with dominant classes as decision

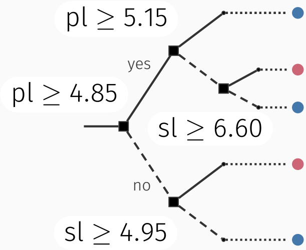




Decision tree: Training

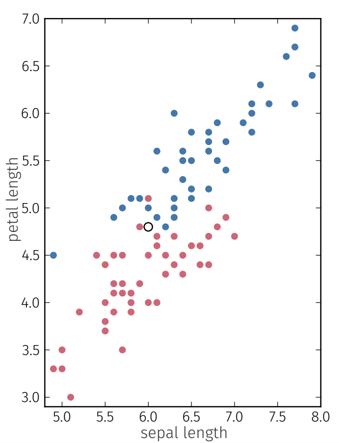
Tree is fully grown...

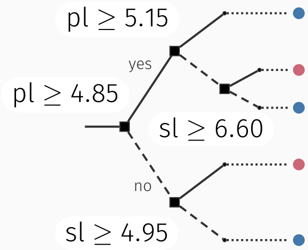




Decision tree: Prediction

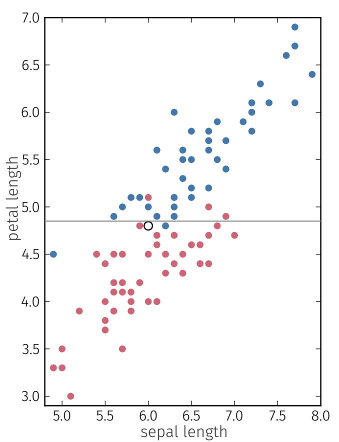
Given a point to classify: $(\mathrm{sl} = 6.0, \mathrm{pl} = 4.8)$

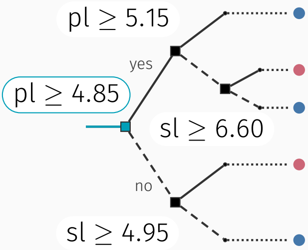




Decision tree: Prediction

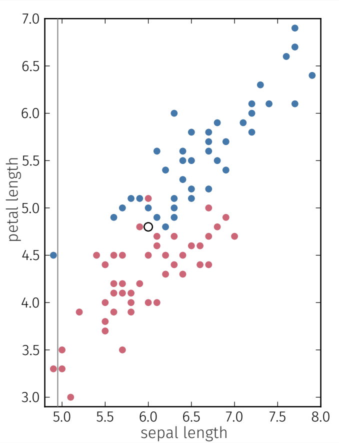
Apply test and follow branch according to the outcome

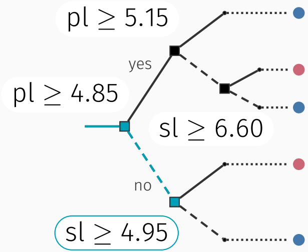




Decision tree: Prediction

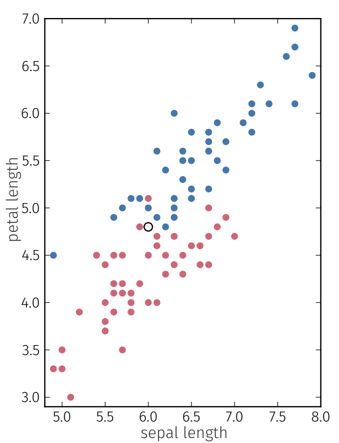
Apply test and follow branch according to the outcome

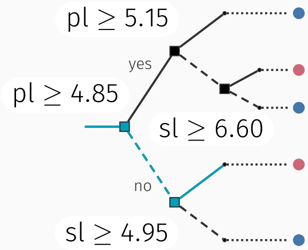




Decision tree: Prediction

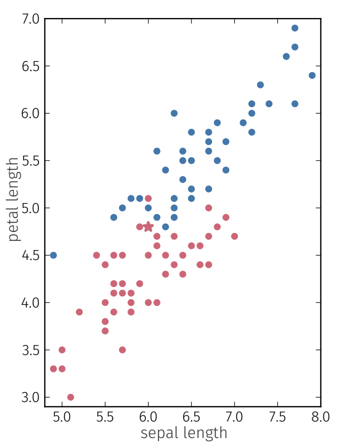
Apply test and follow branch according to the outcome

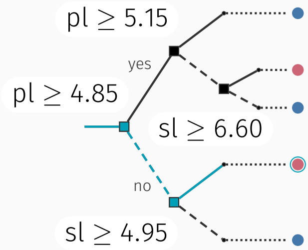




Decision tree: Prediction

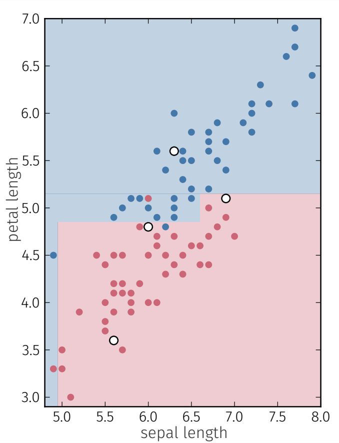
Assign the class associated to the leaf

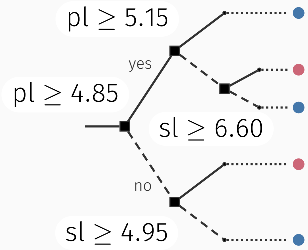




Decision tree: Prediction

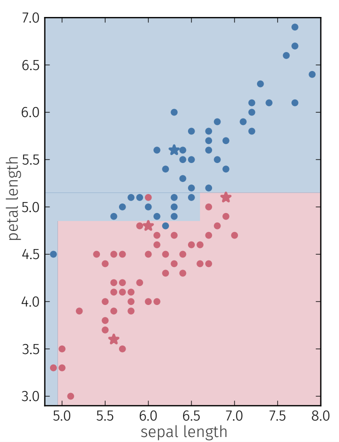
The sequences of tests corresponding to the branches of the tree define decision boundaries

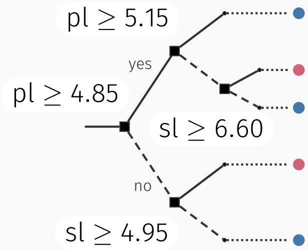




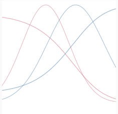
Decision tree: Prediction

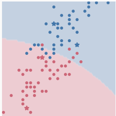
The sequences of tests corresponding to the branches of the tree define decision boundaries





Naive Bayes





Input: data set $\mathcal{D}$ with attributes $X_1, \ldots, X_m$ and class labels Y, data point $\boldsymbol{x}$

Training: estimate the class probabilities $\mathrm{P}(Y)$ and conditional probabilities $\mathrm{P}(X_i \mid Y)$ from $\mathcal{D}$

Prediction: compute conditional probabilities $\mathrm{P}(Y \mid X_1, \ldots, X_m)$ according to Bayes' rule, return class with highest probability

Naive Bayes: Bayes' rule

$$\mathrm{P}(Y \mid X) = \frac{\mathrm{P}(Y) \cdot \mathrm{P}(X \mid Y)}{\mathrm{P}(X)}$$

$$\begin{aligned}
&\mathrm{P}(Y = c \mid X_1 = a_1, \ldots, X_m = a_m) \\
&\qquad = \frac{\mathrm{P}(Y = c) \cdot \mathrm{P}(X_1 = a_1, \ldots, X_m = a_m \mid Y = c)}{\mathrm{P}(X_1 = a_1, \ldots, X_m = a_m)} \\
&\qquad \propto \mathrm{P}(Y = c) \cdot \prod_{i=1}^{i=m} \mathrm{P}(X_i = a_i \mid Y = c)
\end{aligned}$$

Naive Bayes: Training

Estimate $\mathrm{P}(Y = c)$ and $\mathrm{P}(X_i = a_i \mid Y = c)$ from the data, for the different classes c, attributes X_i and values a_i

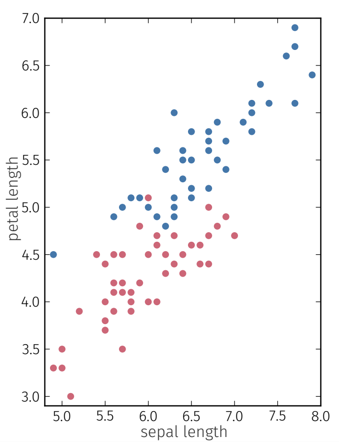
Naive Bayes: Training

$P(Y = c)$ count occurrences of each class in the data

$$P(Y = c) = \frac{\#(c)}{n}$$

Naive Bayes: Training

$P(Y = c)$ count occurrences of each class in the data



$$P(\bullet) = 40/84 = 0.476$$

$$P(\bullet) = 44/84 = 0.524$$

Naive Bayes: Training

$\mathrm{P}(X_i = a_i \mid Y = c)$ count occurrences of each values in the data, for each class and each attribute

$$\mathrm{P}(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}$$

Naive Bayes: Training

$\mathrm{P}(X_i = a_i \mid Y = c)$ count occurrences of each values in the data, for each class and each attribute

$$\mathrm{P}(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}$$

! Needs much data to get reliable values

! How about rare values?

Naive Bayes: Training

$P(X_i = a_i \mid Y = c)$ count occurrences of each values in the data, for each class and each attribute

$$P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}$$

! Needs much data to get reliable values

! How about rare values?

→ use Laplacian smoothing

$$P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c) + \alpha}{\#(c) + \kappa \cdot \alpha}$$

where κ is the number of distinct values of attribute X_i

That is, denoting the domain of X_i as A_i, we let $\kappa = |A_i|$

$$\text{so that} \sum_{a_i \in A_i} P(X_i = a_i \mid Y = c) = 1$$

Naive Bayes: Training

$\mathrm{P}(X_i = a_i \mid Y = c)$ count occurrences of each values in the data, for each class and each attribute

$$\mathrm{P}(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}$$

! Needs much data to get reliable values

! How about continuous domains?

Naive Bayes: Training

$P(X_i = a_i \mid Y = c)$ count occurrences of each values in the data, for each class and each attribute

$$P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}$$

! Needs much data to get reliable values

! How about continuous domains?

$\rightarrow$ model with Gaussian distributions

$$P(X = v \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(v-\mu)^2}{2\sigma^2}}$$

Naive Bayes: Training

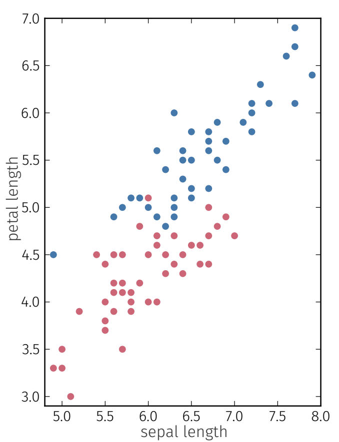
$P(Y = c)$ count occurrences of each class in the data

$P(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data

Naive Bayes: Training

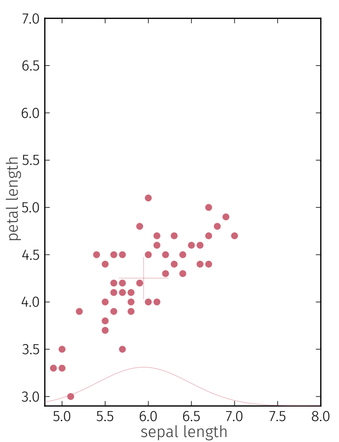
$P(Y = c)$ count occurrences of each class in the data

$P(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$\mathrm{P}(Y = c)$ count occurrences of each class in the data

$\mathrm{P}(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$$\mathrm{P}(v \mid \mu_{\mathrm{sl}}, \sigma_{\mathrm{sl}}) = \frac{1}{\sqrt{2\pi{\sigma_{\mathrm{sl}}}^2}} e^{-\frac{(v-\mu_{\mathrm{sl}})^2}{2{\sigma_{\mathrm{sl}}}^2}}$$

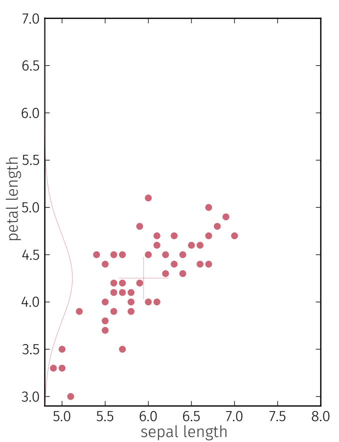
$$\mu_{\mathrm{sl}} = \mathrm{mean}(\mathrm{sl} \mid \bullet) \qquad = 5.945$$

$$ {\sigma_{\mathrm{sl}}}^2 = \mathrm{var}(\mathrm{sl} \mid \bullet) \qquad = 0.285$$

Naive Bayes: Training

$P(Y=c)$ count occurrences of each class in the data

$P(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$$P(v \mid \mu_{pl}, \sigma_{pl}) = \frac{1}{\sqrt{2\pi\sigma_{pl}^2}} e^{-\frac{(v-\mu_{pl})^2}{2\sigma_{pl}^2}}$$

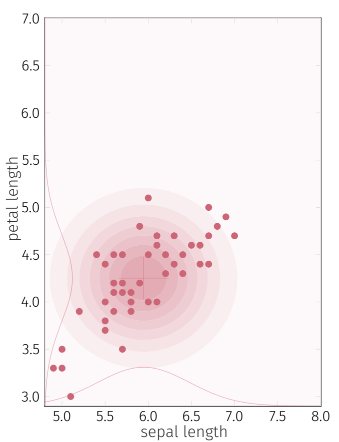
$$\mu_{pl} = \mathrm{mean}(pl \mid \bullet) \qquad = 4.252$$

$$\sigma_{pl}^2 = \mathrm{var}(pl \mid \bullet) \qquad = 0.219$$

Naive Bayes: Training

$\mathrm{P}(Y = c)$ count occurrences of each class in the data

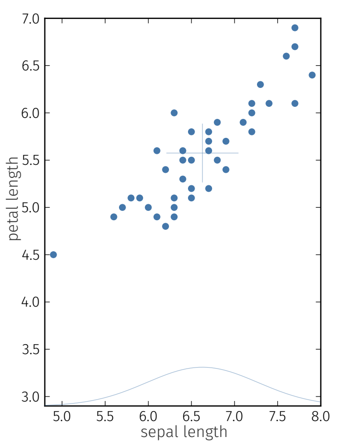
$\mathrm{P}(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$$\mathrm{P}(v \mid \mu_{\mathrm{sl}}, \sigma_{\mathrm{sl}})$$
$$\mathrm{P}(v \mid \mu_{\mathrm{pl}}, \sigma_{\mathrm{pl}})$$

$\mathrm{P}(Y = c)$ count occurrences of each class in the data

$\mathrm{P}(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$$\mathrm{P}(v \mid \mu_{\mathrm{sl}}, \sigma_{\mathrm{sl}}) = \frac{1}{\sqrt{2\pi{\sigma_{\mathrm{sl}}}^2}} e^{-\frac{(v-\mu_{\mathrm{sl}})^2}{2{\sigma_{\mathrm{sl}}}^2}}$$

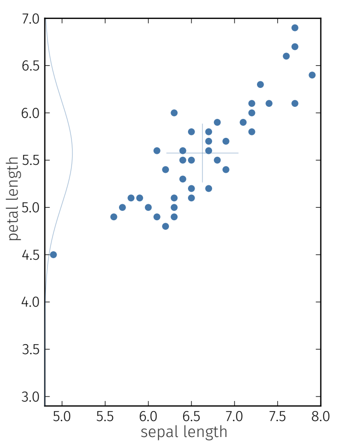
$$\mu_{\mathrm{sl}} = \mathrm{mean}(\mathrm{sl} \mid \bullet) = 6.628$$

$$\sigma_{\mathrm{sl}}{}^2 = \mathrm{var}(\mathrm{sl} \mid \bullet) = 0.414$$

Naive Bayes: Training

$P(Y = c)$ count occurrences of each class in the data

$P(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$$P(v \mid \mu_{pl}, \sigma_{pl}) = \frac{1}{\sqrt{2\pi{\sigma_{pl}}^2}} e^{-\frac{(v-\mu_{pl})^2}{2{\sigma_{pl}}^2}}$$

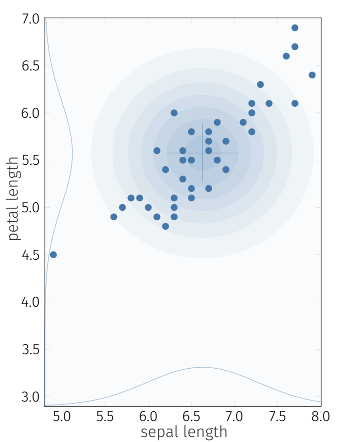
$$\mu_{pl} = \mathrm{mean}(pl \mid \bullet) \qquad = 5.575$$

$$\sigma_{pl}{}^2 = \mathrm{var}(pl \mid \bullet) \qquad = 0.308$$

Naive Bayes: Training

$\mathrm{P}(Y = c)$ count occurrences of each class in the data

$\mathrm{P}(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



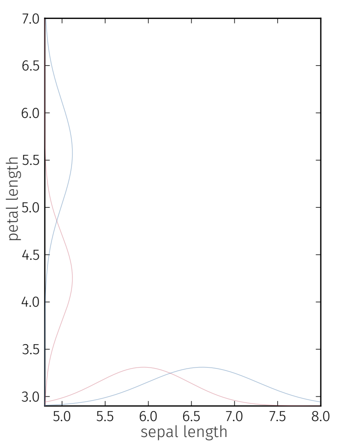
$$\mathrm{P}(v \mid \mu_{\mathrm{sl}}, \sigma_{\mathrm{sl}})$$

$$\mathrm{P}(v \mid \mu_{\mathrm{pl}}, \sigma_{\mathrm{pl}})$$

Naive Bayes: Training

$P(Y = c)$ count occurrences of each class in the data

$P(X_i = a_i \mid Y = c)$ model with Gaussian distributions, estimating the parameters from the data



$$P(\bullet) \qquad P(\bullet)$$

$$P(v \mid \mu_{\text{sl}}, \sigma_{\text{sl}}) \qquad P(v \mid \mu_{\text{sl}}, \sigma_{\text{sl}})$$

$$P(v \mid \mu_{\text{pl}}, \sigma_{\text{pl}}) \qquad P(v \mid \mu_{\text{pl}}, \sigma_{\text{pl}})$$

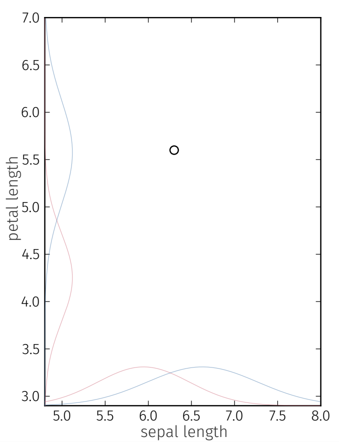
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes' rule

$$\mathrm{P}(Y = c \mid X_1 = a_1, \ldots, X_m = a_m) \propto \mathrm{P}(Y = c) \cdot \prod_{i=1}^{i=m} \mathrm{P}(X_i = a_i \mid Y = c)$$

Naive Bayes: Prediction

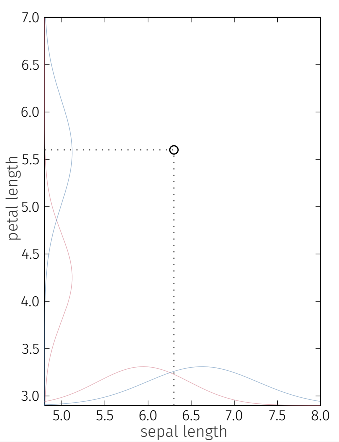
Compute the conditional probability of each class according to Bayes' rule



$$P(\bullet \mid (5.6, 3.6)) \overset{?}{\gtreqless} P(\bullet \mid (5.6, 3.6))$$

Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes' rule

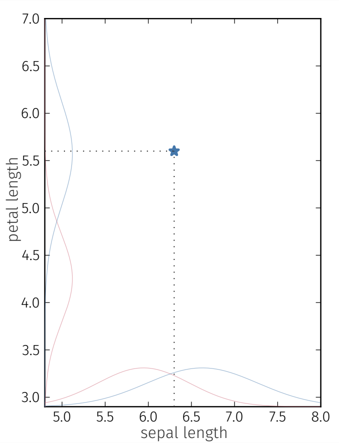


$$P(\bullet \mid (5.6, 3.6)) < P(\bullet \mid (5.6, 3.6))$$

$$0.022 < 0.978$$

Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes' rule

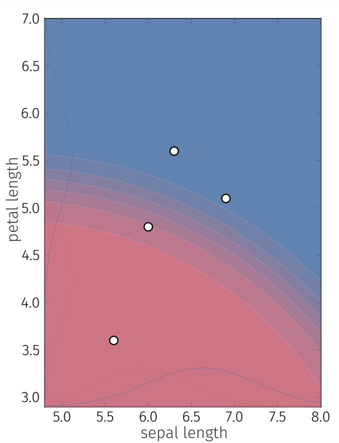


$$P(\textcolor{red}{\bullet} \mid (5.6, 3.6)) < P(\textcolor{blue}{\bullet} \mid (5.6, 3.6))$$

$$0.022 < 0.978$$

Naive Bayes: Prediction

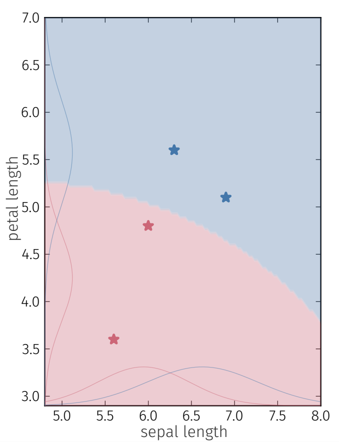
Compute the conditional probability of each class according to Bayes' rule



P(● | x) and P(● | x) can be computed for every point

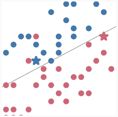
Naive Bayes: Prediction

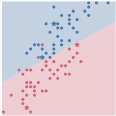
Compute the conditional probability of each class according to Bayes' rule



P(● | x) and P(● | x) can be computed for every point

The decision boundary is the line P(● | x) = P(● | x) = .5





Input: data set $\mathcal{D}$, data point $\boldsymbol{x}$

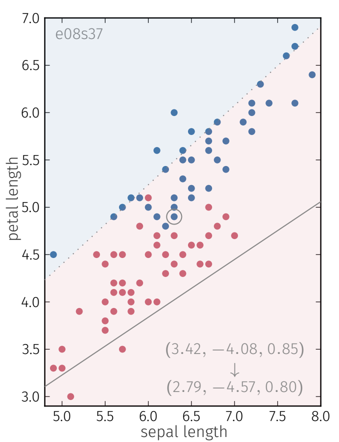
Parameters: learning rate η

Training: initialize weights vector $\boldsymbol{w}$ and bias b,
iterate among points in $\mathcal{D}$ and adjust $\boldsymbol{w}$ and b

Prediction: return class according to sign of $\boldsymbol{w} \cdot \boldsymbol{x} + b$

Perceptron: Training

Iterate among points $x^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights w and bias b



At step t,

compute current prediction

$$z_i = \mathrm{sign}(w^{(t)} \cdot x^{(j)} + b^{(t)})$$

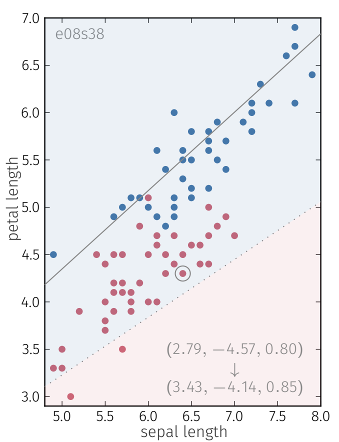
update

$$w^{(t+1)} = w^{(t)} + \eta(y_j - z_j)x^{(j)}$$

$$b^{(t+1)} = b^{(t)} + \eta(y_j - z_j)$$

Perceptron: Training

Iterate among points $x^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights w and bias b



At step t,
compute current prediction

$$z_j = \mathrm{sign}(w^{(t)} \cdot x^{(j)} + b^{(t)})$$

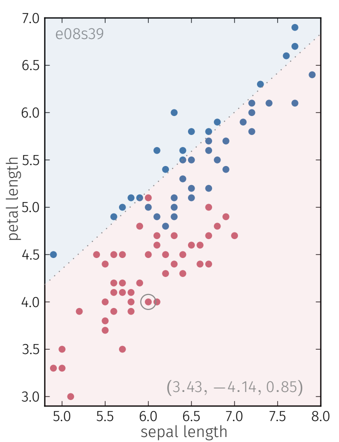
update

$$w^{(t+1)} = w^{(t)} + \eta(y_j - z_j)x^{(j)}$$

$$b^{(t+1)} = b^{(t)} + \eta(y_j - z_j)$$

Perceptron: Training

Iterate among points $\boldsymbol{x}^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights $\boldsymbol{w}$ and bias b



At step t,

compute current prediction

$$z_i = \mathrm{sign}(\boldsymbol{w}^{(t)} \cdot \boldsymbol{x}^{(j)} + b^{(t)})$$

update

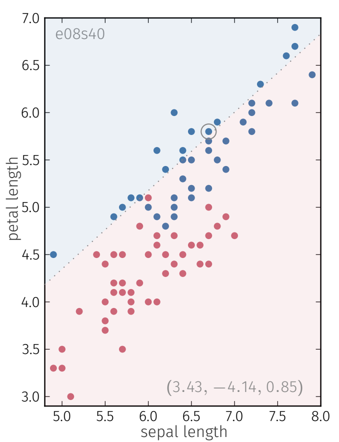
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta(y_j - z_j)\boldsymbol{x}^{(j)}$$

$$b^{(t+1)} = b^{(t)} + \eta(y_j - z_j)$$

Note that if prediction is correct, $\boldsymbol{w}$ and b are unchanged

Perceptron: Training

Iterate among points $\mathbf{x}^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights $\mathbf{w}$ and bias b



At step t,
compute current prediction

$$z_i = \operatorname{sign}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(j)} + b^{(t)})$$

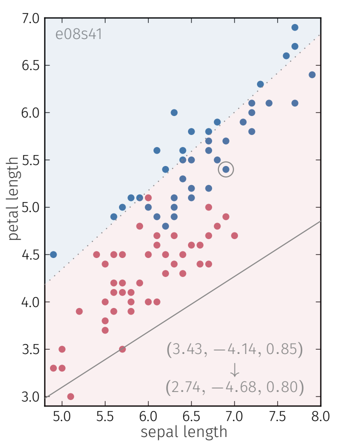
update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta(y_j - z_j)\mathbf{x}^{(j)}$$

$$b^{(t+1)} = b^{(t)} + \eta(y_j - z_j)$$

Note that if prediction is correct, $\mathbf{w}$ and b are unchanged

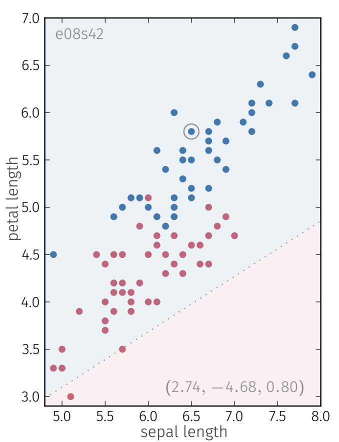
Iterate among points $\boldsymbol{x}^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights $\boldsymbol{w}$ and bias b



Might cycle several times through all points of the training data

Each such cycle is called an *epoch*

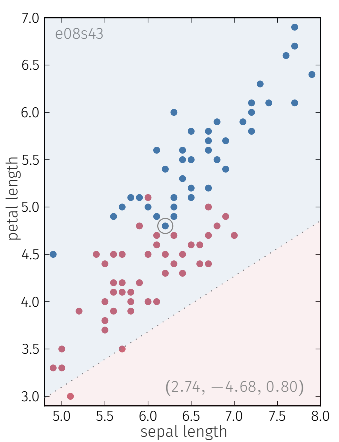
Iterate among points $x^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights w and bias b



If the data is linearly separable, convergence on *some* solution is guaranteed

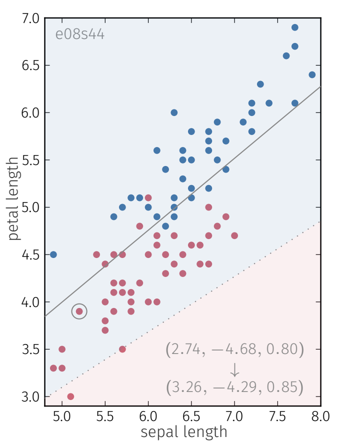
Perceptron: Training

Iterate among points $x^{(j)}$ from $\mathcal{D}$ in a random order
and adjust weights w and bias b



If the data is not linearly separable, learning will fail

Iterate among points $x^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights w and bias b

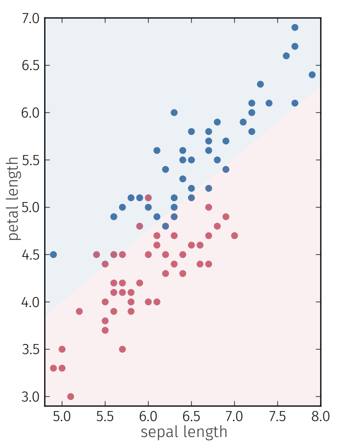


If the data is not linearly separable, learning will fail

The algorithm will not even approach an approximate solution

Perceptron: Training

Iterate among points $x^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights w and bias b

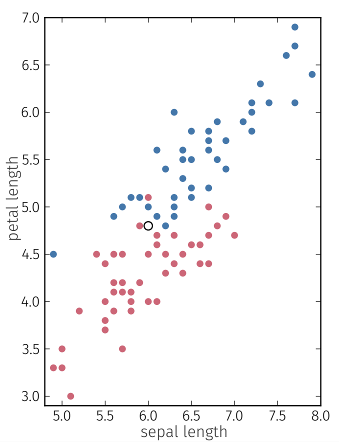


If the data is not linearly separable, learning will fail

A quick fix:
store best solution encountered and return it after chosen maximum number of epochs

Perceptron: Prediction

Return class according to $\mathbf{sign}(w \cdot x + b)$



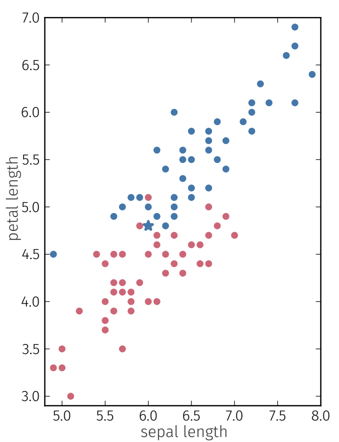
$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 < 0$

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

$$0.671 \cdot 6.0$$
$$-1.365 \cdot 4.8 + 2.39 = -0.136$$

Perceptron: Prediction

Return class according to $\mathrm{sign}(w \cdot x + b)$



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

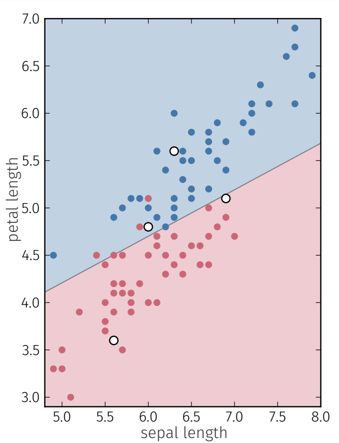
$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 \geq 0$

$$0.671 \cdot 6.0$$
$$-1.365 \cdot 4.8 + 2.39 = -0.136$$

predict

Perceptron: Prediction

Return class according to $\mathrm{sign}(w \cdot x + b)$

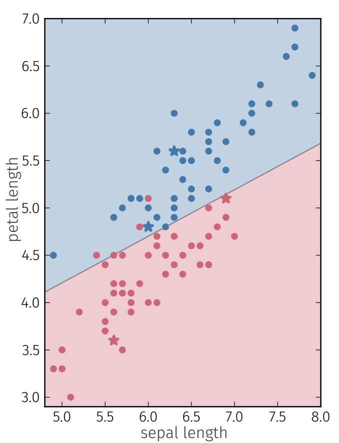


$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 < 0$

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

Perceptron: Prediction

Return class according to $\mathsf{sign}(w \cdot x + b)$



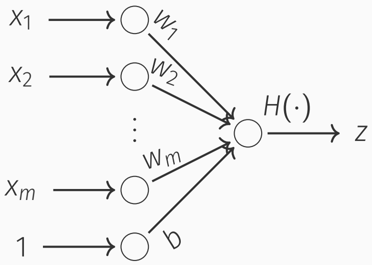
$$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 < 0$$

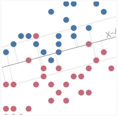
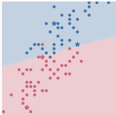
$$0.671 \cdot \mathrm{sl} - 1.365 \cdot \mathrm{pl} + 2.39 \geq 0$$

Perceptron

The Perceptron can be seen as the simplest neural network, a single-layer neural network

One *input node* for each data attribute
One *output node* computing the *activation function*



Input: data set $\mathcal{D}$, data point $\boldsymbol{x}$

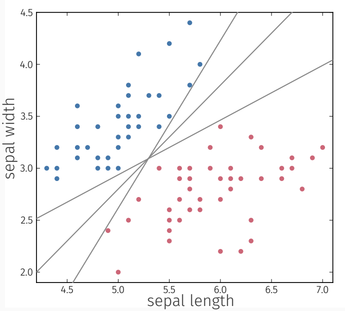
Parameters: penalty coefficient C

Training: solve for vector $\boldsymbol{w}$ and bias b that define a hyperplane separating points in $\mathcal{D}$ from the two classes with largest margin

Prediction: return class according to sign of $\boldsymbol{w} \cdot \boldsymbol{x} + b$

When the data is linearly separable,
there might be multiple different separating hyperplanes

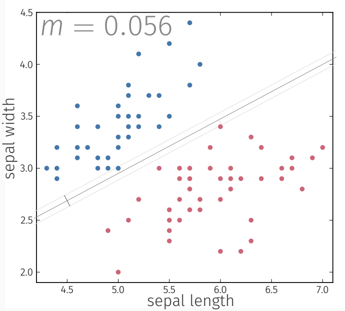
which one to choose?



SVM: linearly separable case

When the data is linearly separable,
there might be multiple different separating hyperplanes

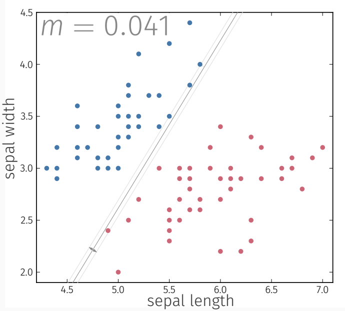
which one to choose?



SVM: linearly separable case

When the data is linearly separable,
there might be multiple different separating hyperplanes

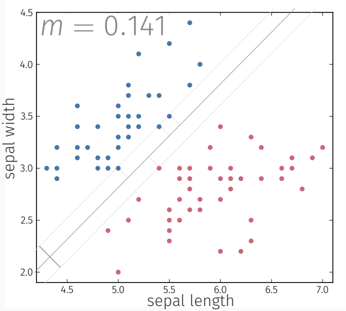
which one to choose?



SVM: linearly separable case

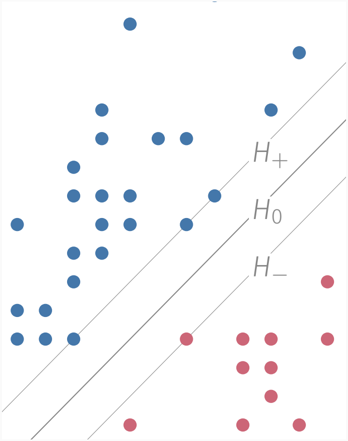
When the data is linearly separable,
there might be multiple different separating hyperplanes

which one to choose? Larger margin provides more stability



Hard-margin SVM

Determining the maximum margin hyperplane



we have two hyperplanes,
such that (under suitable scaling)

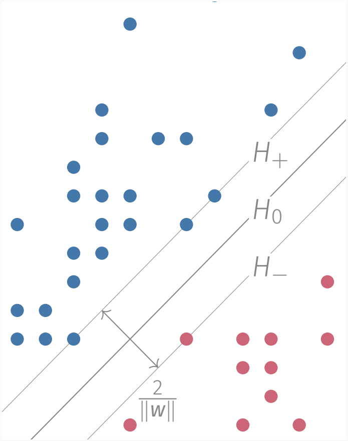
$$(H_+) \quad \mathbf{w} \cdot \mathbf{x}^{(j)} + b \geq +1 \; \forall j, y_j = +1$$
$$(H_-) \quad \mathbf{w} \cdot \mathbf{x}^{(j)} + b \leq -1 \; \forall j, y_j = -1$$

In short

$$y_j(\mathbf{w} \cdot \mathbf{x}^{(j)} + b) \geq 1 \; \forall j$$

Hard-margin SVM

Determining the maximum margin hyperplane



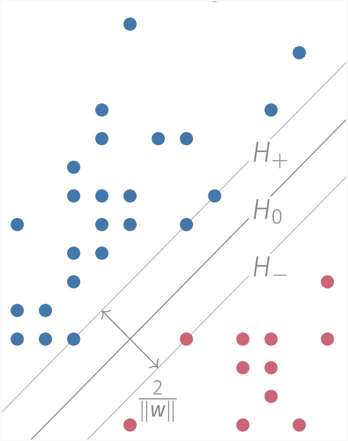
we have two hyperplanes,
such that (under suitable scaling)

$$(H_+) \quad w \cdot x^{(j)} + b \geq +1 \; \forall j, y_j = +1$$
$$(H_-) \quad w \cdot x^{(j)} + b \leq -1 \; \forall j, y_j = -1$$

The distance between H_+ and H_- equals $2/\|w\|$

Hard-margin SVM

Determining the maximum margin hyperplane

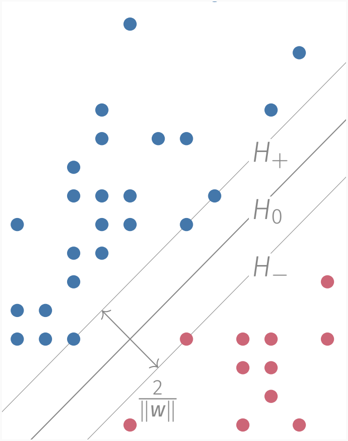


the problem can be formulated as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_j(\mathbf{w}\cdot\mathbf{x}^{(j)} + b) \geq 1 \;\forall j \end{aligned}$$

Hard-margin SVM

Determining the maximum margin hyperplane



the problem can be formulated as

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{s.t.} \quad y_j(\boldsymbol{w} \cdot \boldsymbol{x}^{(j)} + b) \geq 1 \; \forall j$$

this is a quadratic constrained optimization problem

can be solved by the Lagrangian multiplier method

Hard-margin SVM

Primal problem

$$\min L_P = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{j=1}^{j=n} a_j\left(1 - y_j(\mathbf{w} \cdot \mathbf{x}^{(j)} + b)\right) \text{ s.t. } 0 \leq a_j \; \forall j$$

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2}\sum_{j=1}^{j=n}\sum_{i=1}^{i=n} a_j a_i y_j y_i \mathbf{x}^{(j)} \cdot \mathbf{x}^{(i)}$$

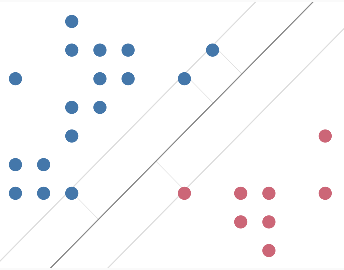
$$\text{s.t. } 0 \leq a_j \text{ and } \sum_{j=1}^{j=n} a_j y_j = 0 \; \forall j$$

variables a_j are defined in such a way that $\mathbf{w} = \sum_{j=1}^{j=n} a_j y_j \mathbf{x}^{(j)}$

Hard-margin SVM: Training

Training the SVM means solving for the a_j by differentiating the dual problem and setting it to zero

Most of the a_j will have value zero
Training points associated to non-zero a_j are called support vectors, since they actually define the separating hyperplane



$$\mathbf{w} = \sum_{j=1}^{j=n} a_j y_j \mathbf{x}^{(j)}$$

Support vectors satisfy
$y_j(\mathbf{w} \cdot \mathbf{x}^{(j)} + b) = 1$
which allows to compute b

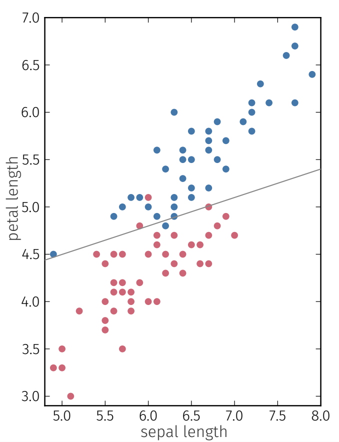
Hard-margin SVM: Prediction

Return class according to $\mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b)$, where

$$\mathbf{w} \cdot \mathbf{x} + b = b + \sum_{j=1}^{j=n} a_j y_j \mathbf{x}^{(j)} \cdot \mathbf{x}$$

SVM: non linearly separable case

What if the data is not linearly separable?



No hyperplane such that constraint

$$y_j(\mathbf{w} \cdot \mathbf{x}^{(j)} + b) \geq 1$$

would be satisfied by all points

Some points inside the margin, or even on the wrong side of the separating hyperplane

SVM: non linearly separable case

What if the data is not linearly separable?

Use the *hinge loss* and introduce a new variable for each point

$$\xi_j = \max\left(0, 1 - y_j(\mathbf{w} \cdot \mathbf{x}^{(j)} + b)\right)$$

the value is

zero if the point satisfies the margin constraint

proportional to the distance to the margin otherwise

SVM: non linearly separable case

What if the data is not linearly separable?

Hard-margin problem

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{s.t.} \quad y_j(\boldsymbol{w} \cdot \boldsymbol{x}^{(j)} + b) \geq 1 \; \forall j$$

Soft-margin problem

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{j=1}^{j=n} \xi_j$$
$$\text{s.t.} \quad y_j(\boldsymbol{w} \cdot \boldsymbol{x}^{(j)} + b) \geq 1 - \xi_j \text{ and } 0 \leq \xi_j \; \forall j$$

Soft-margin SVM

Primal problem

$$\min L_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{j=1}^{j=n}\xi_j + \sum_{j=1}^{j=n} a_j\left(1 - \xi_j - y_j(\mathbf{w}\cdot\mathbf{x}^{(j)} + b)\right) - \sum_{j=1}^{j=n}\mu_j\xi_j$$

$$\text{s.t.} \;\; 0 \le a_j \text{ and } 0 \le \mu_j \;\forall j$$

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2}\sum_{j=1}^{j=n}\sum_{i=1}^{i=n} a_j a_i y_j y_i \mathbf{x}^{(j)}\cdot\mathbf{x}^{(i)}$$

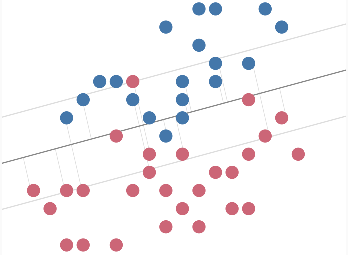
$$\text{s.t.} \;\; 0 \le a_j \le C \text{ and } \sum_{j=1}^{j=n} a_j y_j = 0 \;\forall j$$

variables a_j are defined in such a way that $\mathbf{w} = \sum_{j=1}^{j=n} a_j y_j \mathbf{x}^{(j)}$

Soft-margin SVM: Training

Training the SVM means solving for the a_j by differentiating the dual problem and setting it to zero

Most of the a_j will have value zero
Training points associated to non-zero a_j are called support vectors, since they actually define the separating hyperplane



For every support vector

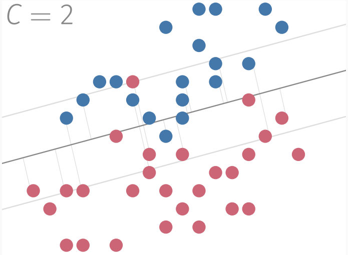
$$y_j(\mathbf{w} \cdot \mathbf{x}^{(j)} + b) = 1 - \xi_j$$

if $\xi_j = 0$, the point is on the margin
otherwise, it is within the margin
or even on the wrong side

Soft-margin SVM: Training

Training the SVM means solving for the a_j by differentiating the dual problem and setting it to zero

Most of the a_j will have value zero
Training points associated to non-zero a_j are called support vectors, since they actually define the separating hyperplane

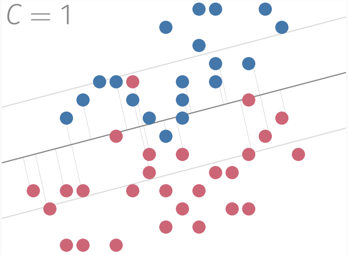


Parameter C allows to adjust the trade-off between width of the margin and constraint violations

Soft-margin SVM: Training

Training the SVM means solving for the a_j by differentiating the dual problem and setting it to zero

Most of the a_j will have value zero
Training points associated to non-zero a_j are called support vectors, since they actually define the separating hyperplane

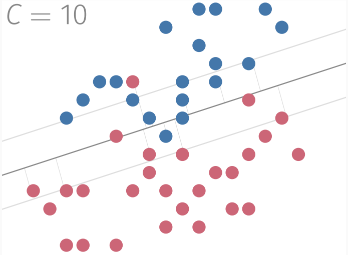


Parameter C allows to adjust the trade-off between width of the margin and constraint violations

Soft-margin SVM: Training

Training the SVM means solving for the a_j by differentiating the dual problem and setting it to zero

Most of the a_j will have value zero
Training points associated to non-zero a_j are called support vectors, since they actually define the separating hyperplane



Parameter C allows to adjust the trade-off between width of the margin and constraint violations

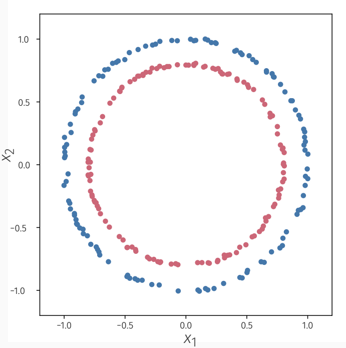
Soft-margin SVM: Prediction

Return class according to $\mathrm{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$, where

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = b + \sum_{j=1}^{j=n} a_j y_j \boldsymbol{x}^{(j)} \cdot \boldsymbol{x}$$

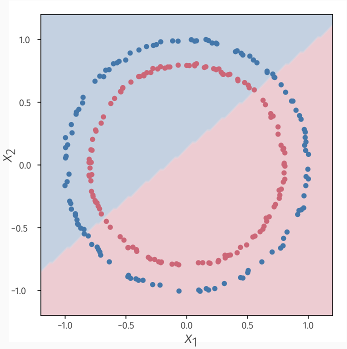
Kernelized SVM

What if a linear decision boundary is not the right option?



Kernelized SVM

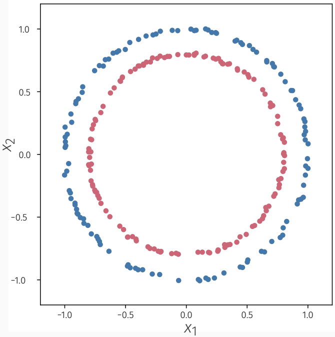
What if a linear decision boundary is not the right option?

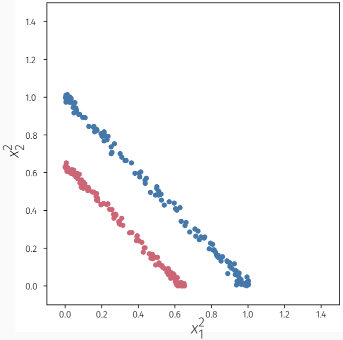


Kernelized SVM

What if a linear decision boundary is not the right option?

Project the data to a different space



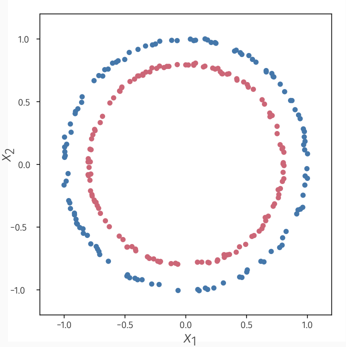


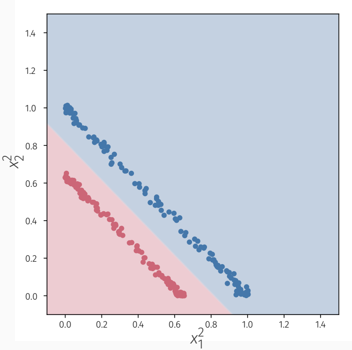
Kernelized SVM

What if a linear decision boundary is not the right option?

Train the SVM in the projected space

$$\varphi_S(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle$$



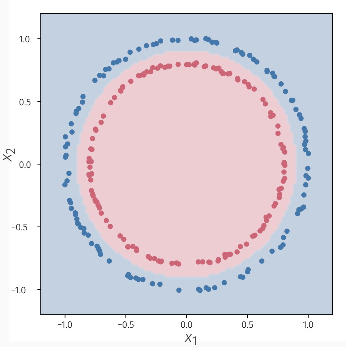


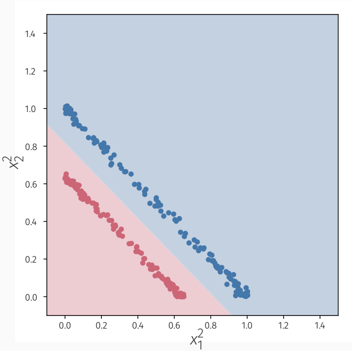
Kernelized SVM

What if a linear decision boundary is not the right option?

Train the SVM in the projected space

$$\varphi_S(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle$$





Kernelized SVM

If the data cannot be separated in the original space

1. find a projection φ to a space where the data can be separated
2. apply the SVM method to the transformed dataset

If the data cannot be separated in the original space

1. find a projection φ to a space where the data can be separated
2. apply the SVM method to the transformed dataset

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i \, y_j y_i \; \varphi(\mathbf{x}^{(j)}) \cdot \varphi(\mathbf{x}^{(i)})$$

$$\text{s.t.} \;\; 0 \leq a_j \leq C \text{ and } \sum_{j=1}^{j=n} a_j y_j = 0 \; \forall j$$

Prediction $\quad \text{sign}\left(b + \sum_{j=1}^{j=n} a_j y_j \, \varphi(\mathbf{x}^{(j)}) \cdot \varphi(\mathbf{x}^{(i)})\right)$

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i \, y_j y_i \, \varphi(\mathbf{x}^{(j)}) \cdot \varphi(\mathbf{x}^{(i)})$$

$$\text{s.t.} \;\; 0 \leq a_j \leq C \text{ and } \sum_{j=1}^{j=n} a_j y_j = 0 \;\forall j$$

Prediction $\quad \text{sign}\left(b + \sum_{j=1}^{j=n} a_j y_j \, \varphi(\mathbf{x}^{(j)}) \cdot \varphi(\mathbf{x}^{(i)})\right)$

The transformed values $\varphi(\mathbf{x})$ appear only in dot products

The transformed values $\varphi(x)$ appear only in dot products

If the dot product in the transformed space can be replaced by a function

$$K(x, x') = \varphi(x) \cdot \varphi(x')$$

we can avoid performing the transformation explicitely

Kernelized SVM

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2}\sum_{j=1}^{j=n}\sum_{i=1}^{i=n} a_j a_i\, y_j y_i \; \mathrm{K}(\mathbf{x}^{(j)}, \mathbf{x}^{(i)})$$

$$\text{s.t. } \; 0 \leq a_j \leq C \text{ and } \sum_{j=1}^{j=n} a_j y_j = 0 \; \forall j$$

Prediction $\quad \text{sign}\left(b + \sum_{j=1}^{j=n} a_j y_j \, \mathrm{K}(\mathbf{x}^{(j)}, \mathbf{x}^{(i)})\right)$

The Kernel Trick

By replacing the dot product in the transformed space by a function

$$K(x, x') = \varphi(x) \cdot \varphi(x')$$

we can avoid performing the transformation explicitly

The feature map φ does not need to be explicitly defined

It is enough that K be expressible as an inner product

Mercer's theorem gives the conditions for K to be a valid kernel function

In particular, the similarity matrix (a.k.a. Gram matrix) $S_{ij} = K(\mathbf{v}_i, \mathbf{v}_j)$ for a finite input space $\langle \mathbf{v}_1, \ldots, \mathbf{v}_l \rangle$ must be positive semi-definite (PSD)

The Kernel Trick

Improving the separability of data points typically means projecting into a high dimensional space
computing with high dimensional vectors is costly
the kernel function operates in the original space

The kernel trick provides the benefits of high-dimensionality without the costs

Kernels are useful beyond SVMs, in other methods where dot products, i.e. similarity computations, are involved

Common kernel functions

Polynomial kernel

$$K(x, x') = (x \cdot x' + c)^h$$

Sigmoid kernel

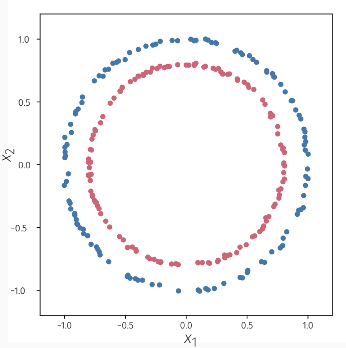
$$K(x, x') = \tanh(\kappa\, x \cdot x' - \delta)$$

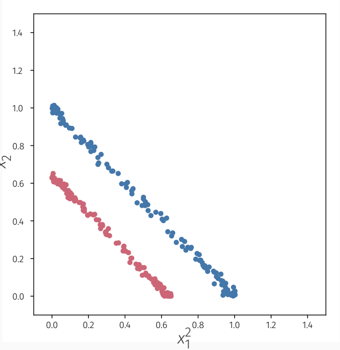
Gaussian radial basis kernel

$$K(x, x') = e^{\|x - x'\|^2 / 2\sigma^2}$$

Kernelized SVM

$$\varphi_s(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle$$





Kernelized SVM

$$\varphi_s(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle$$

Consider the polynomial kernel of degree two

$$\begin{aligned} \mathrm{K}(\mathbf{x}, \mathbf{x}') &= (\mathbf{x} \cdot \mathbf{x}')^2 = (x_1 {x'}_1 + x_2 {x'}_2)^2 \\ &= x_1^2 {x'}_1^2 + x_2^2 {x'}_2^2 + 2 x_1 {x'}_1 x_2 {x'}_2 \\ &= \langle x_1^2, x_2^2, \sqrt{2} x_1 x_2 \rangle \cdot \langle {x'}_1^2, {x'}_2^2, \sqrt{2} {x'}_1 {x'}_2 \rangle \\ &= \varphi_p(\mathbf{x}) \cdot \varphi_p(\mathbf{x}') \quad \text{where } \varphi_p(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2, \sqrt{2} x_1 x_2 \rangle \end{aligned}$$

The terms of the feature map φ_s is a subset of those of φ_p

Using the polynomial kernel

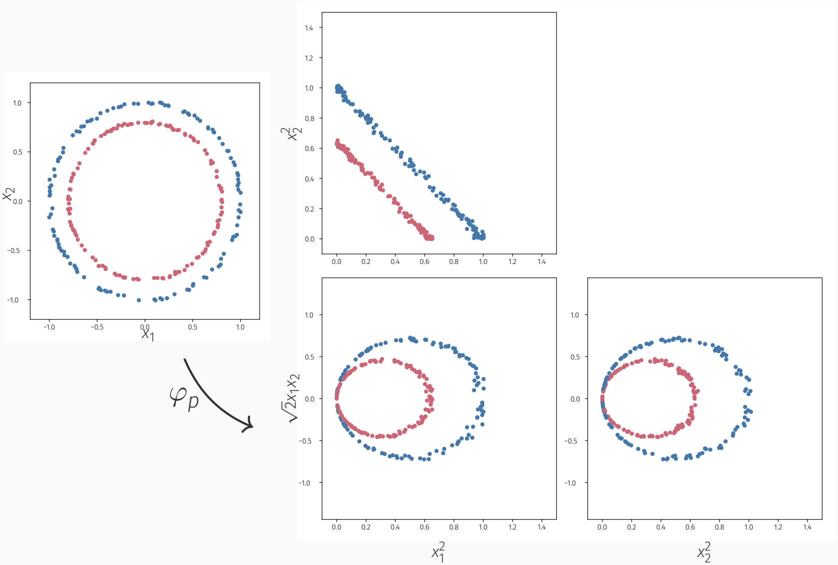
$$K(x, x') = (x \cdot x')^2$$

for the two-dimensional example dataset corresponds to projecting the points into three dimensional space

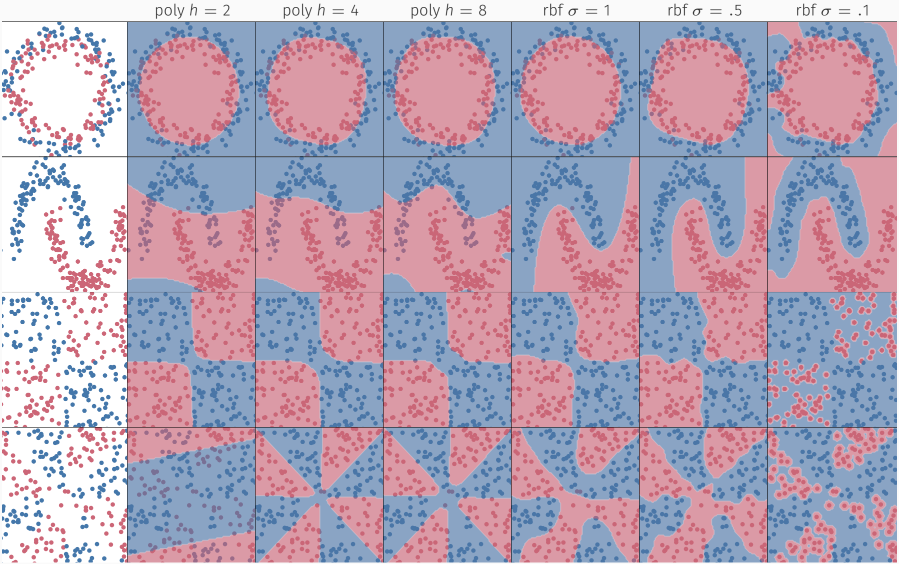
$$\varphi_p(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2, \sqrt{2}x_1x_2 \rangle$$

before training the SVM

Kernelized SVM



Different kernels



Evaluation

Model evaluation

Given a dataset, we can build a number of models, that come with different variants and parameter settings

We need to quantify the accuracy of models, in order to

- measure the effectiveness and tune the parameters of a particular model
- compare, select, combine various models

Evaluation measures

To evaluate the performance of a model we compare the known labels of the instances, representing the *ground truth*, to the predicted labels

Let y and z denote respectively the true and predicted labels

If there are l distinct classes, there are $l \cdot l$ distinct possible outcomes for a given instance

The instance might belong to c_1 *and be predicted as* c_1
The instance might belong to c_1 *and be predicted as* c_2

$$\vdots$$

The instance might belong to c_l *and be predicted as* c_1

$$\vdots$$

The instance might belong to c_l *and be predicted as* c_l

Evaluation measures

Let y and z denote respectively the true and predicted labels

The outcome of the classification of a set of instances across l classes can be summarized in a $l \times l$ *contingency matrix*

		Ground truth		
		$y = c_1$	$\dots$ $y = c_i$ $\dots$	$y = c_l$
Prediction	$z = c_1$	$\#(z = c_1, y = c_1)$	$\#(z = c_1, y = c_i)$	$\#(z = c_1, y = c_l)$
	$\vdots$			
	$z = c_i$	$\#(z = c_i, y = c_1)$	$\#(z = c_i, y = c_i)$	$\#(z = c_i, y = c_l)$
	$\vdots$			
	$z = c_l$	$\#(z = c_l, y = c_1)$	$\#(z = c_l, y = c_i)$	$\#(z = c_l, y = c_l)$

Evaluation measures

Let y and z denote respectively the true and predicted labels

The outcome of the classification of a set of instances across l classes can be summarized in a $l \times l$ *contingency matrix*

The **accuracy** is the fraction of correctly classified instances

		Ground truth				
		$y = c_1$	$\dots$	$y = c_i$	$\dots$	$y = c_l$
Prediction	$z = c_1$	$\#(z = c_1, y = c_1)$		$\#(z = c_1, y = c_i)$		$\#(z = c_1, y = c_l)$
	$\vdots$					
	$z = c_i$	$\#(z = c_i, y = c_1)$		$\#(z = c_i, y = c_i)$		$\#(z = c_i, y = c_l)$
	$\vdots$					
	$z = c_l$	$\#(z = c_l, y = c_1)$		$\#(z = c_l, y = c_i)$		$\#(z = c_l, y = c_l)$

Evaluation measures

Binary classification is a special case with specific terminology

Two classes: *positive* and *negative*

There are four possible outcomes for a given instance

The 2×2 *contingency matrix* is called *confusion matrix*

		Ground truth	
		$y = 0$	$y = 1$
Prediction	$z = 0$	True negative	False negative
	$z = 1$	False positive	True positive

Evaluation measures

Binary classification is a special case with specific terminology

Two classes: *positive* and *negative*

There are four possible outcomes for a given instance

The 2×2 *contingency matrix* is called *confusion matrix*

	$y = 0$	$y = 1$
$z = 0$	True negative	False negative
$z = 1$	False positive	True positive

False positive: type I error (a.k.a. false discovery, false alarm)

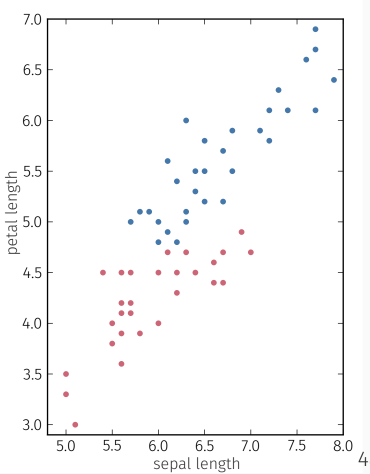
False negative: type II error (a.k.a. miss)

Evaluation measures

A binary classifier is trained on a portion of data (*training data*)

For example,
consider a linear SVM

Solve for vector w and bias b
defining a separating hyperplane

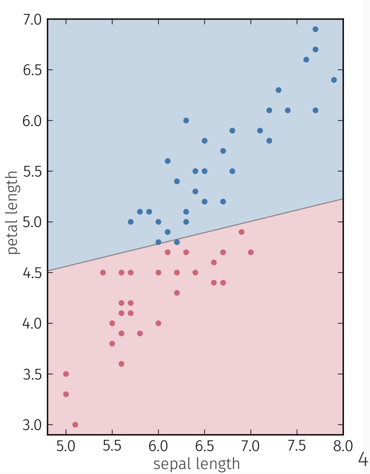


Evaluation measures

A binary classifier is trained on a portion of data (*training data*)

For example,
consider a linear SVM

Solve for vector $\boldsymbol{w}$ and bias b
defining a separating hyperplane

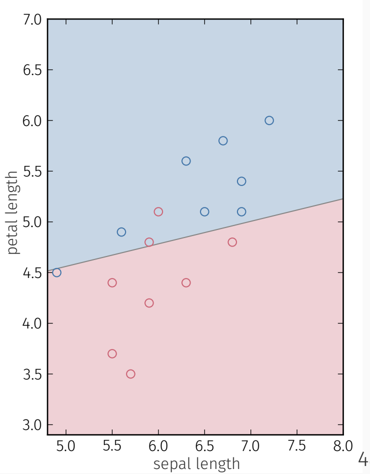


Evaluation measures

A binary classifier is trained on a portion of data (*training data*), and applied on another portion for which the ground-truth is also known but hidden from the classifier (*test data*)

For example,
consider a linear SVM

For instance $\boldsymbol{x}$, predict class according to sign of $\boldsymbol{w} \cdot \boldsymbol{x} + b$

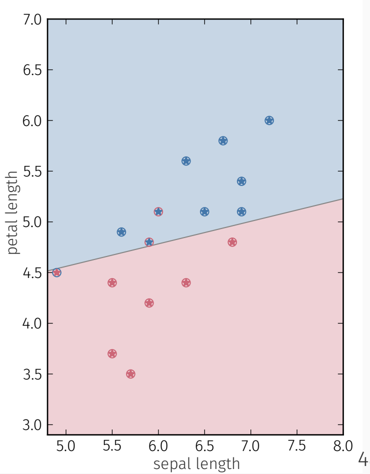


Evaluation measures

A binary classifier is trained on a portion of data (*training data*), and applied on another portion for which the ground-truth is also known but hidden from the classifier (*test data*)

For example,
consider a linear SVM

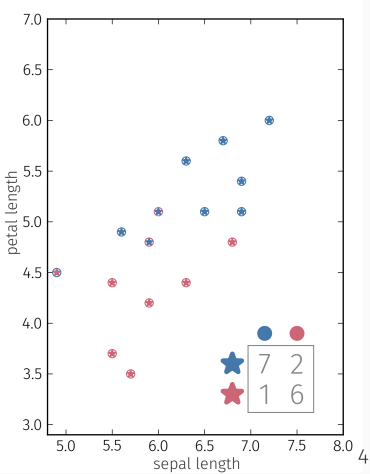
For instance $\boldsymbol{x}$, predict class according to sign of $\boldsymbol{w} \cdot \boldsymbol{x} + b$



Evaluation measures

The outcome of the binary classification of a collection of N instances can be summarized in a confusion matrix

$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$



Evaluation measures

The outcome of the binary classification of a collection of N instances can be summarized in a confusion matrix

$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$

	$y = 0$	$y = 1$
$z = 0$	True negative	False negative
$z = 1$	False positive	True positive

Evaluation measures

$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$

	$y = 0$	$y = 1$
$z = 0$	True negative	False negative
$z = 1$	False positive	True positive

Various measures can be computed from this matrix

precision $\frac{N_{TP}}{N_{TP}+N_{FP}}$ (a.k.a. positive predictive value)

recall $\frac{N_{TP}}{N_{TP}+N_{FN}}$ (a.k.a. sensitivity, true positive rate)

specificity $\frac{N_{TN}}{N_{TN}+N_{FP}}$ (a.k.a. selectivity, true negative rate)

false positive rate $\frac{N_{FP}}{N_{FP}+N_{TN}}$

false negative rate $\frac{N_{FN}}{N_{FN}+N_{TP}}$

Evaluation measures

$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$

	$y = 0$	$y = 1$
$z = 0$	True negative	False negative
$z = 1$	False positive	True positive

Various measures can be computed from this matrix

precision $\frac{N_{TP}}{N_{TP}+N_{FP}}$ (a.k.a. positive predictive value)

recall $\frac{N_{TP}}{N_{TP}+N_{FN}}$ (a.k.a. sensitivity, true positive rate)

F1 score harmonic mean of recall and precision

$$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2N_{TP}}{2N_{TP} + N_{FP} + N_{FN}}$$

Evaluation measures

$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$

	$y = 0$	$y = 1$
$z = 0$	True negative	False negative
$z = 1$	False positive	True positive

Various measures can be computed from this matrix

accuracy fraction of instances in which the predicted label matches the ground truth

$$\frac{N_{TP} + N_{TN}}{N}$$

Evaluation measures

accuracy fraction of instances in which the predicted label matches the ground truth

$$\frac{\sum_{c \in C} \#(y = c, z = c)}{\sum_{c \in C} \#(y = c)}$$

In some cases, not all classes are equally important

misclassification in one class incurs a higher cost than misclassification in the other class

reflected by weight w_c assigned to each class

weighted accuracy (a.k.a. cost-sensitive accuracy)

$$\frac{\sum_{c \in C} w_c \cdot \#(y = c, z = c)}{\sum_{c \in C} w_c \cdot \#(y = c)}$$

Evaluation measures

The outcome of the binary classification of a collection of N instances can be summarized in a confusion matrix

Various measures can be computed from this matrix

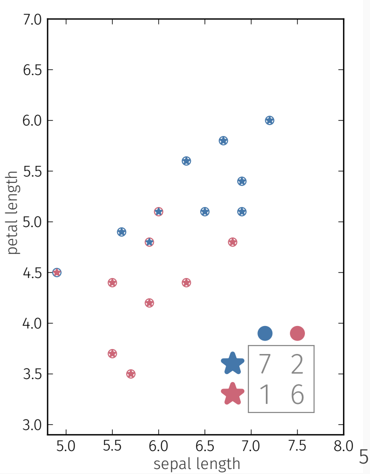
$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$

$$\mathrm{acc} = 0.812$$

$$\mathrm{TPR} = 0.750$$

$$\mathrm{FPR} = 0.125$$

$$\dots$$



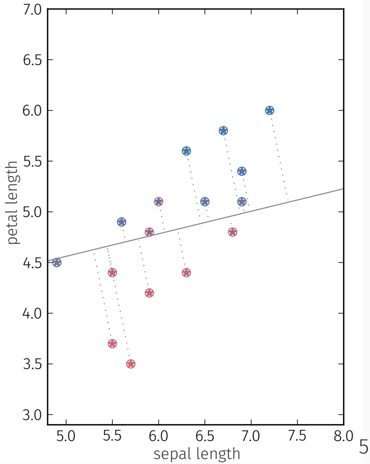
Evaluation measures

Instead of crisp class assignments we might consider a numerical score reflecting the confidence of the classifier

Class probabilities, distance from the decision boundary, etc.

For example,
consider a linear SVM

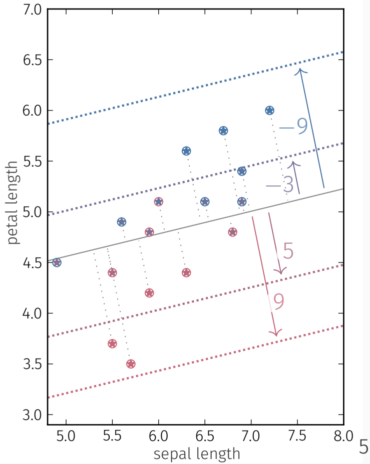
Use $\boldsymbol{w} \cdot \boldsymbol{x} + b$
as score for instance $\boldsymbol{x}$



Evaluation measures

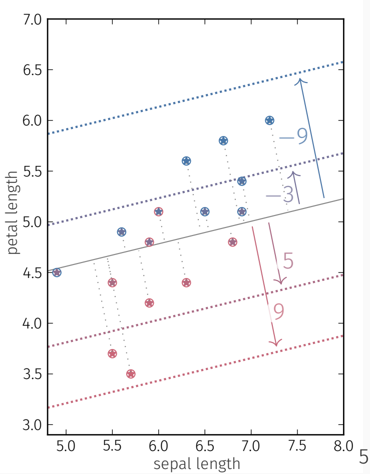
Varying the score above which an instance is assigned to the positive class means moving the decision boundary

For instance $\boldsymbol{x}$, predict class according to the outcome of test $\boldsymbol{w} \cdot \boldsymbol{x} + b \geq \theta$ with $\theta = -9$, $\theta = -3$, $\theta = 5$, or $\theta = 9$, for example



Let's look at what happens if we modify the level of confidence required to assign an instance to the positive class

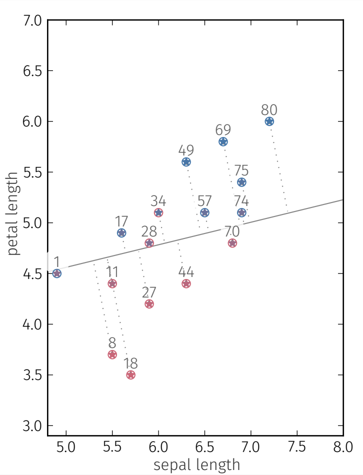
From almost certain that it does not belong to the negative class ($\theta \to -\infty$)
to almost certain that it belongs to the positive class ($\theta \to +\infty$)
and cases in between



Evaluation

Consider a linear SVM, use $\mathbf{w} \cdot \mathbf{x} + b$ as score for instance $\mathbf{x}$

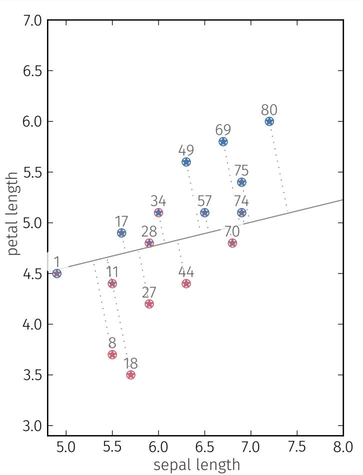
Collect the test instances with their scores



○	1	0.259
○	8	6.481
○	11	1.815
○	17	−1.370
○	18	8.111
○	27	3.741
○	28	−0.259
○	34	−2.111
○	44	3.000
○	49	−5.000
○	57	−1.370
○	69	−5.741
○	70	1.074
○	74	−0.778
○	75	−2.778
○	80	−6.333

Consider a linear SVM, use $w \cdot x + b$ as score for instance x

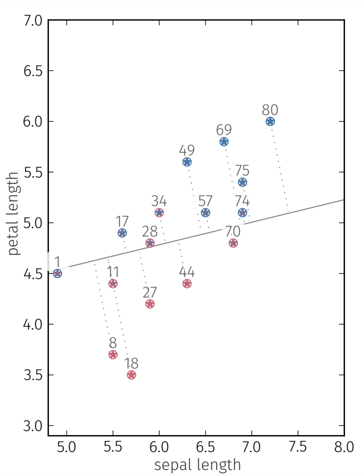
Sort the test instances by increasing values of the score



80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Evaluation

As thresholds we can use mid-point values between successive distinct score values among the test instances

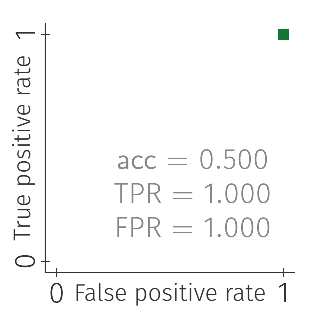


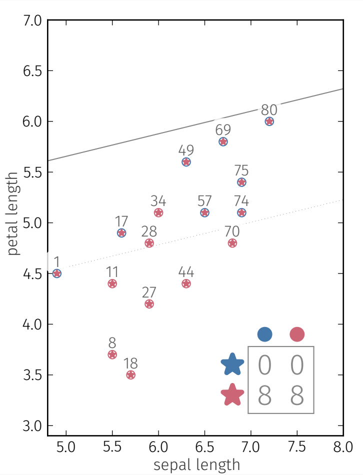
80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

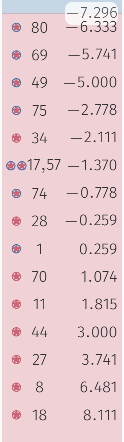
Evaluation

Set the threshold to the minimum value

Record the corresponding FPR and TPR

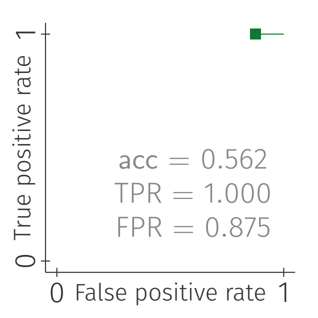


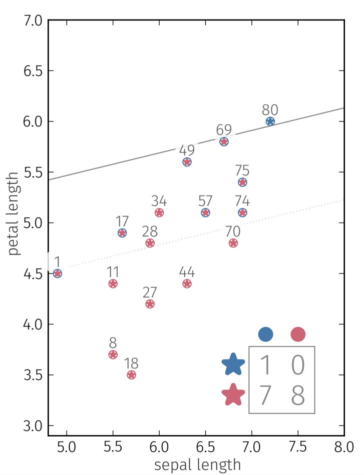




Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR



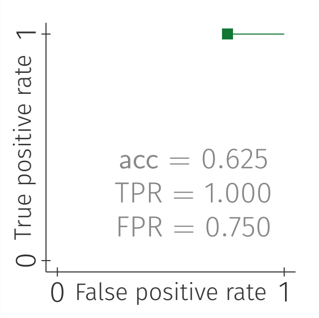


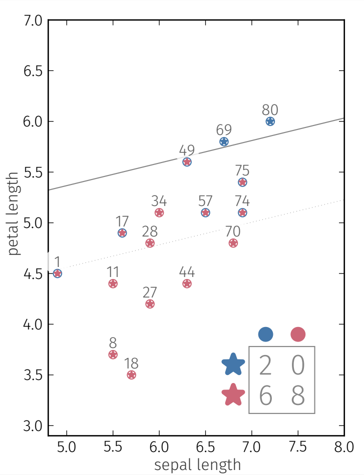
80	−6.333
	−6.037
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Evaluation

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

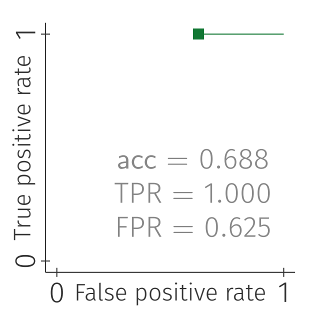


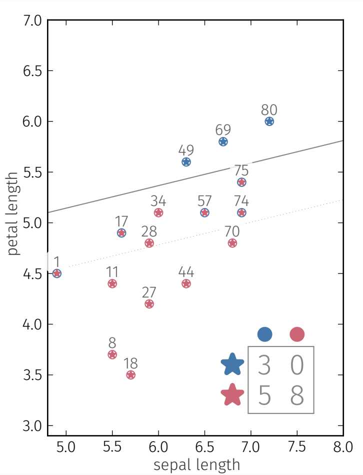


80	−6.333
69	−5.741
	−5.370
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

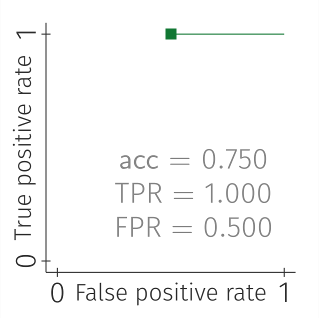
Record the corresponding FPR and TPR

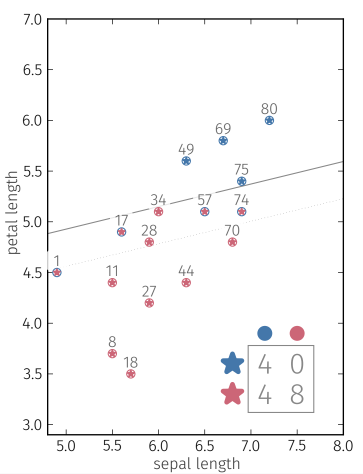




80	−6.333
69	−5.741
49	−5.000
	−3.889
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

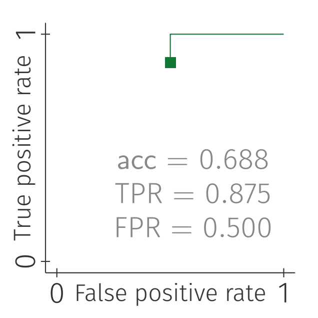


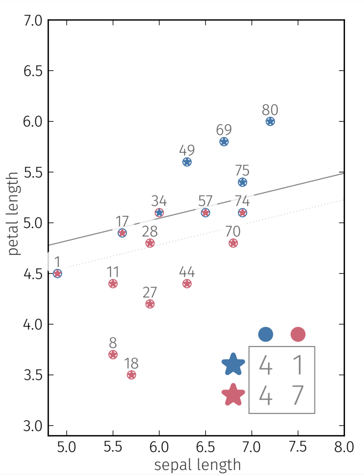


80	−6.333
69	−5.741
49	−5.000
75	−2.778
	−2.445
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

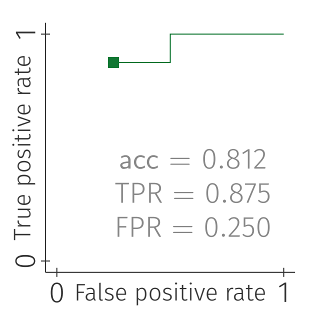


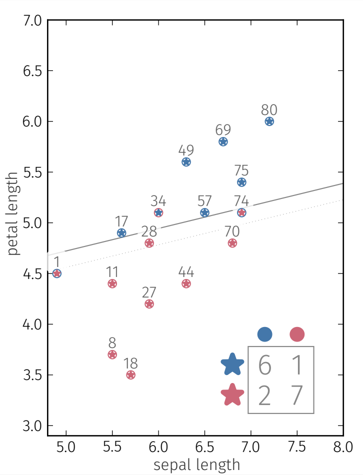


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
	−1.741
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

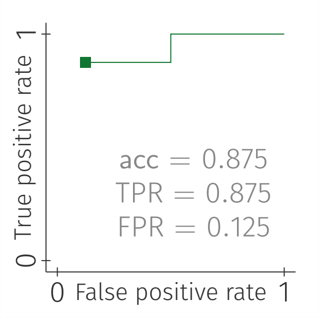


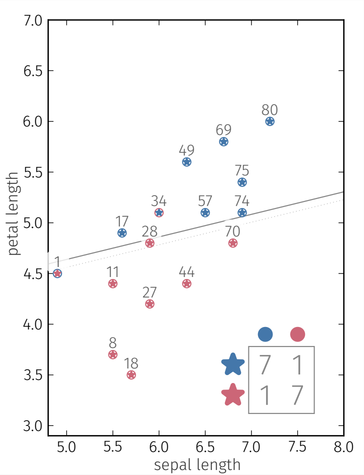


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
	−1.074
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

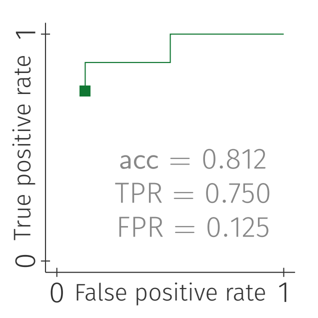


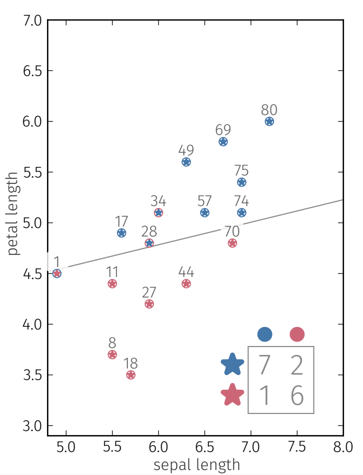


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
	−0.518
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

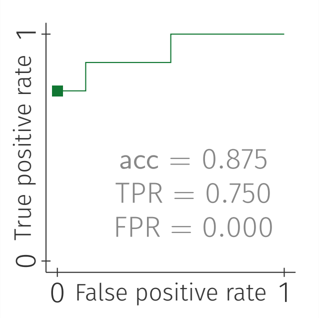


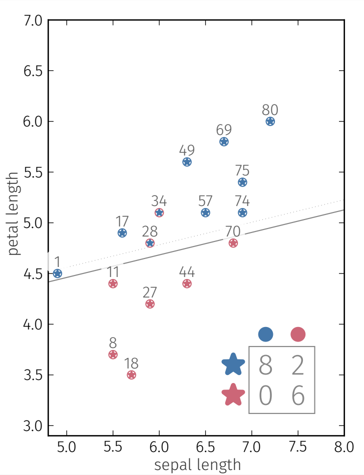


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
	0.000
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

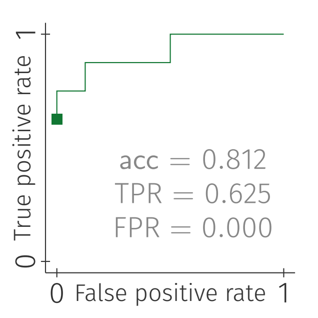


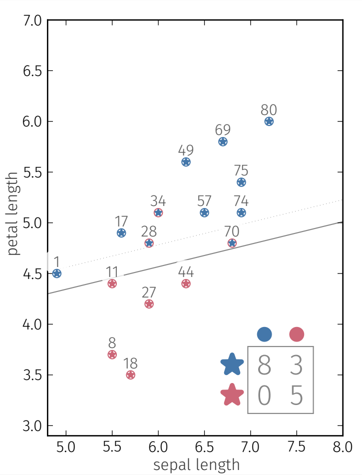


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
	0.667
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

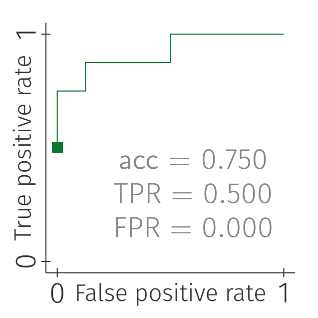


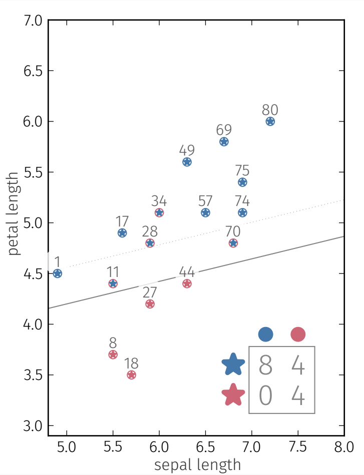


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
	1.445
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Evaluation

Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

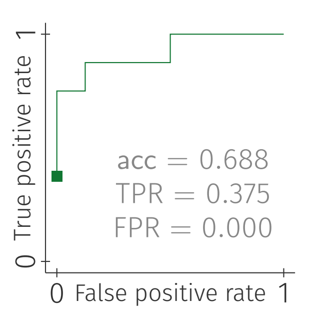


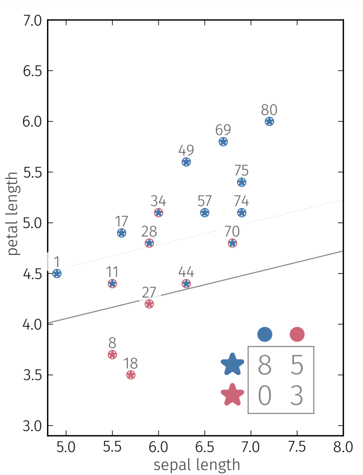


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
	2.407
44	3.000
27	3.741
8	6.481
18	8.111

Set the threshold to the minimum value, raise it progressively

Record the corresponding FPR and TPR

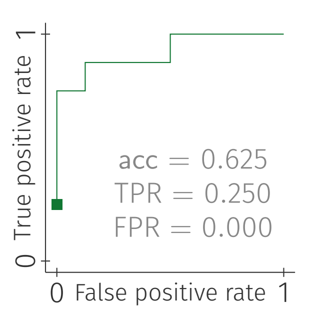


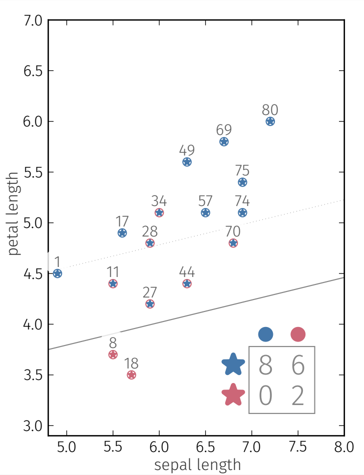


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
	3.370
27	3.741
8	6.481
18	8.111

Evaluation

FPR and TPR depend on where the ranking is split between classes, not the specific threshold value

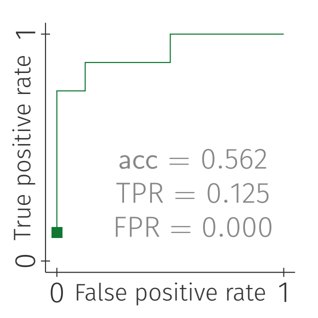


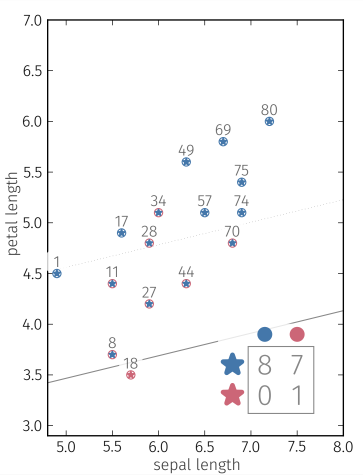


80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
	5.111
8	6.481
18	8.111

Evaluation

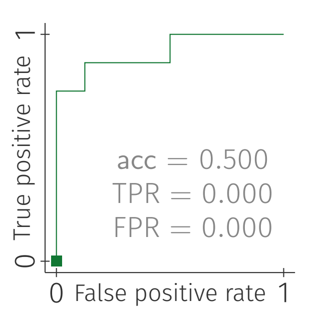
Hence, successive FPR and TPR can be computed directly from the ranked instances

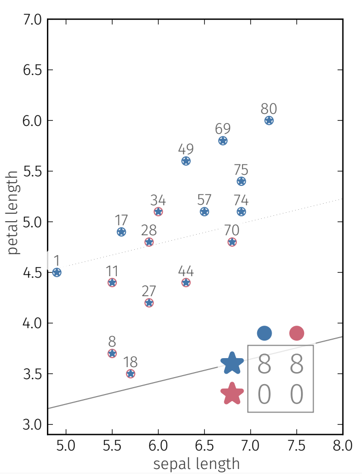




80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
	7.296
18	8.111

The curve FPR vs. TPR is commonly refered to as **receiver operating characteristic (ROC) curve**

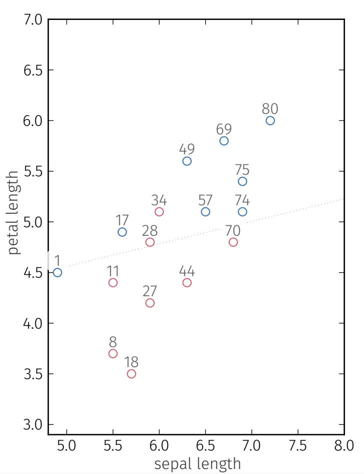




80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111
	9.074

The curve FPR vs. TPR is commonly refered to as **(ROC) curve**

The **area under the curve (AUC)** summarizes the ROC curve in a single number



80	−6.333
69	−5.741
49	−5.000
75	−2.778
34	−2.111
17,57	−1.370
74	−0.778
28	−0.259
1	0.259
70	1.074
11	1.815
44	3.000
27	3.741
8	6.481
18	8.111

Model evaluation

! The goal is not to best mimic the labels of the training data

For evaluation, we need labelled data points not seen during training

Compromise: the more labelled data for training the better but some labelled examples need to be hold out for evaluation

Hold-out

Divide the labelled data into two disjoint sets

training data used to train the model
typically ca. 2/3 – 3/4 of data

test data used to evaluate the model
typically ca. 1/3 – 1/4 of data

Hold-out

Divide the labelled data into two disjoint sets

training data used to train the model
typically ca. 2/3 – 3/4 of data

test data used to evaluate the model
typically ca. 1/3 – 1/4 of data

! only a fraction of data used for training

! error estimates are pessimistic

Hold-out

Divide the labelled data into two disjoint sets

training data used to train the model
typically ca. 2/3 – 3/4 of data

test data used to evaluate the model
typically ca. 1/3 – 1/4 of data

! only a fraction of data used for training
! error estimates are pessimistic

Repeat this process over several different hold-out samples

- improve the error estimate
- measure variance and compute statistical confidence intervals on the error

Cross-validation

Divide the labelled data into ℓ disjoint sets of equal size ℓ/n
use one set as test data, remaining $\ell - 1$ as training data
repeat with each set as test data

In each of ℓ rounds, the training data has size $n(\ell - 1)/\ell$
This is called **ℓ-fold cross-validation**
e.g. 10-fold cross-validation is common

Cross-validation

Divide the labelled data into ℓ disjoint sets of equal size ℓ/n
use one set as test data, remaining $\ell - 1$ as training data
repeat with each set as test data

In each of ℓ rounds, the training data has size $n(\ell - 1)/\ell$
This is called **ℓ-fold cross-validation**

For larger values of ℓ,
the training data comprises more examples
 $\rightarrow$ better error estimation (still pessimistic)
more rounds are needed
 $\rightarrow$ higher computational cost

Cross-validation

Divide the labelled data into ℓ disjoint sets of equal size ℓ/n
use one set as test data, remaining $\ell - 1$ as training data
repeat with each set as test data

In each of ℓ rounds, the training data has size $n(\ell - 1)/\ell$
This is called ℓ-**fold cross-validation**

Extreme case: $\ell = n$
i.e. ℓ rounds with $n - 1$ training examples and 1 test example
This is called **leave-one-out** cross-validation

Bootstrap

Sample training data of size n from the labelled data uniformly with replacement

The training data has the same size as the original data but some original data examples might be duplicated while others might be missing

The probability that a point is not included in the sample is $(1 - 1/n)^n$, which tends towards $1/e$ as n increases

Sample training data of size n from the labelled data uniformly with replacement

Use the full original labelled data as test data

The large overlap between training and test data means the error estimate is highly optimistic

Repeating this process over several different bootstrap samples allows to compute the mean and variance of the error estimate

Leave-one-out bootstrap

Generate ℓ bootstrap samples and use each such sample to train one classifier

For labelled example $\boldsymbol{x}$, evaluate $\mathrm{acc}(\boldsymbol{x})$ the performance on $\boldsymbol{x}$ of the classifiers trained on samples that do not contain $\boldsymbol{x}$

Average $acc(\boldsymbol{x})$ over all labelled examples

Validation

A step of parameter tuning and model selection, called **validation**, might be needed

Validation and test should not be carried out on the same set, since knowledge of the test set has been implicitly used while building the model

A portion of the data is used to train different models and select one
The performance of the selected model should be evaluated on a distinct, so far unseen, portion of the data

Comparing models

The statistical robustness of models is important differences in accuracy between models might be due to random variations

Assume we have obtained ℓ estimates of the accuracy of two models $\mathcal{M}_A$ and $\mathcal{M}_B$ on different randomly sampled subsets of data (e.g. through repeated hold-out or bootstrap procedures) $\{\mathrm{acc}(\mathcal{M}_A, 1), \ldots, \mathrm{acc}(\mathcal{M}_A, \ell)\}$ and $\{\mathrm{acc}(\mathcal{M}_B, 1), \ldots, \mathrm{acc}(\mathcal{M}_B, \ell)\}$

Comparing models

Assume we have obtained ℓ estimates of the accuracy of two models $\mathcal{M}_A$ and $\mathcal{M}_B$ on different randomly sampled subsets of data (e.g. through repeated hold-out or bootstrap procedures) $\{\text{acc}(\mathcal{M}_A, 1), \ldots, \text{acc}(\mathcal{M}_A, \ell)\}$ and $\{\text{acc}(\mathcal{M}_B, 1), \ldots, \text{acc}(\mathcal{M}_B, \ell)\}$

Let δ_i be the difference in accuracy in round i, i.e.

$$\delta_i = \text{acc}(\mathcal{M}_A, i) - \text{acc}(\mathcal{M}_B, i)$$

the average difference in accuracy is $\Delta = \sum_{i=1}^{i=\ell} \delta_i / \ell$
and the standard deviation of the difference in accuracy is

$$\sigma = \sqrt{\frac{\sum_{i=1}^{i=\ell} (\delta_i - \Delta)^2}{\ell - 1}}$$

Comparing models

We assume that δ_i are sampled from a normal distribution with estimated mean and standard deviation Δ and σ, respectively According to the central limit theorem, the standard deviation of the estimated mean accuracy difference Δ is $\sigma/\sqrt{\ell}$

The number of standard deviations by which Δ is different from the break-even value of 0 is $\sqrt{\ell}\,|\Delta - 0|\,/\sigma$

For sufficiently large number of rounds ℓ, the probability that one model is truly better than the other can be quantified using the standard normal distribution

Comparing models

For sufficiently large number of rounds ℓ, the probability that one model is truly better than the other can be quantified using the standard normal distribution

It is generally too computationally expensive to run sufficiently many rounds to robustly estimate σ so the Student's t-distribution with $\ell - 1$ degrees of freedom is used instead of the normal distribution

Diagnostic

Classification can be seen as the problem of learning a function between the data attributes and the class label

$$y = \mathrm{g}(x) + \epsilon$$

g represents the true, unknown, relationship between data attributes and class label
ϵ represents the intrinsic error in the data, the noise, cannot be modelled

Diagnostic

Classification can be seen as the problem of learning a function between the data attributes and the class label

$$y = \mathrm{g}(x) + \epsilon$$

g represents the true, unknown, relationship between data attributes and class label, even the form of g is unknown
ϵ represents the intrinsic error in the data, the noise, cannot be modelled

Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

$$z = \mathrm{f}_{\mathcal{D}}(x)$$

Diagnostic

Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

$$z = \mathrm{f}_{\mathcal{D}}(x)$$

The function might be defined algorithmically or in closed form
The parameters of the function are estimated from the data

choice of approach	*Family*
choice of setup	*Species*
estimation from $\mathcal{D}$	*Individual*

Diagnostic

Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

$$z = \mathrm{f}_{\mathcal{D}}(x)$$

The function might be defined algorithmically or in closed form
The parameters of the function are estimated from the data

choice of approach	*Family*	decision tree
choice of setup	*Species*	max depth, max leaf size
estimation from $\mathcal{D}$	*Individual*	intermediate tests, decisions

$\mathrm{f}_{\mathcal{D}}(x)$ is defined algorithmically

Diagnostic

Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

$$z = \mathrm{f}_{\mathcal{D}}(x)$$

The function might be defined algorithmically or in closed form
The parameters of the function are estimated from the data

choice of approach	*Family*	linear SVM
choice of setup	*Species*	
estimation from $\mathcal{D}$	*Individual*	$a_1, \dots, a_n, b$

$$\mathrm{f}_{\mathcal{D}}(x) = \mathrm{sign}\left(b + \sum_{j=1}^{j=n} a_j y_j x^{(j)} \cdot x\right)$$

Diagnostic

Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

$$z = \mathrm{f}_{\mathcal{D}}(x)$$

The function might be defined algorithmically or in closed form
The parameters of the function are estimated from the data

choice of approach	*Family*	kernelized SVM
choice of setup	*Species*	kernel function K
estimation from $\mathcal{D}$	*Individual*	$a_1, \ldots, a_n, b$

$$\mathrm{f}_{\mathcal{D}}(x) = \mathbf{sign}\left(b + \sum_{j=1}^{j=n} a_j y_j \, \mathrm{K}(x^{(j)}, x^{(i)})\right)$$

Diagnostic

Classification algorithms

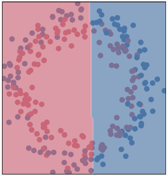
- rely on modeling assumptions
- estimate the parameters from the data

Diagnostic

Classification algorithms

- rely on modeling assumptions
- estimate the parameters from the data

Assumptions may not reflect the true form of the relationship



Oversimplifying assumptions do not allow to capture the underlying structure of the data
$\rightarrow$ Underfitting

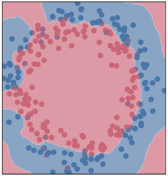
Diagnostic

Classification algorithms

- rely on modeling assumptions
- estimate the parameters from the data

Even with correct modeling assumptions, the true parameters cannot be estimated exactly from the training data

With increasingly complex models, i.e. more parameters, the model might fit too closely to the training data



Captures the structure of the data but also noise

Does not generalize to unseen data

$\rightarrow$ Overfitting

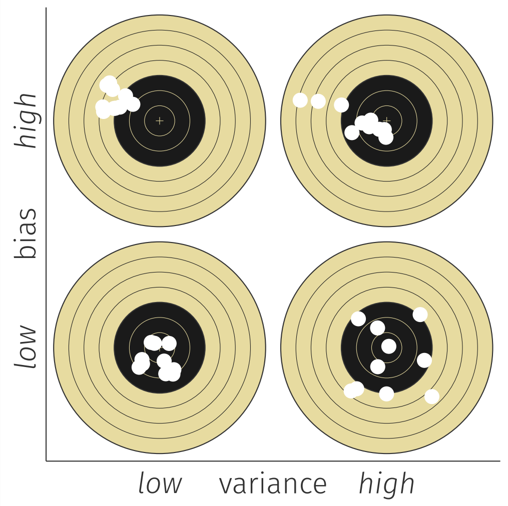
Diagnostic

Imagine we had a very large dataset and could repeat the whole model training many times, we could estimate the expected prediction $E_{\mathcal{D}}[\mathrm{f}_{\mathcal{D}}(x)]$

Because of differences between the assumed model and the true model, $\mathrm{g}(x)$ and $E_{\mathcal{D}}[\mathrm{f}_{\mathcal{D}}(x)]$ would differ
$\rightarrow$ Bias

For a fixed test instance $\boldsymbol{x}$, the value $\mathrm{f}_{\mathcal{D}}(x)$ would vary for different instanciations of the training data $\mathcal{D}$
$\rightarrow$ Variance

Bias and variance



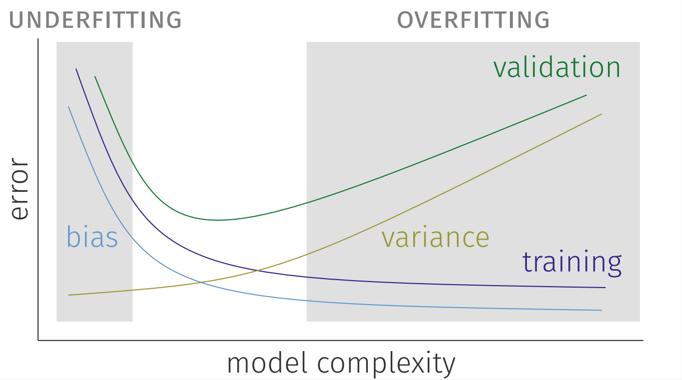
Diagnostic

The expected mean squared error of the prediction for test data points $\{(x_1, y_1), \ldots, (x_l, y_l)\}$ can be written as

$$E_{\mathcal{D}}[\mathrm{MSE}] =$$

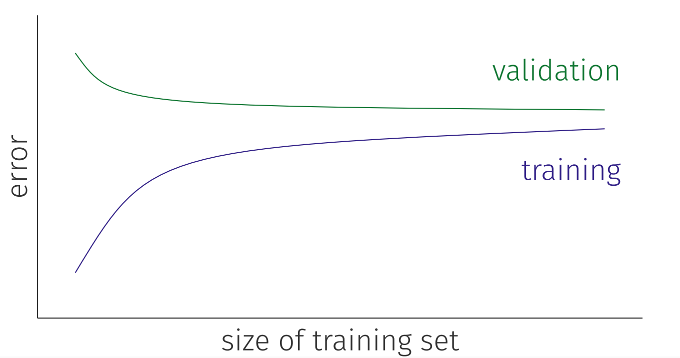
$$\frac{1}{l}\sum_{i=1}^{i=l}\underbrace{(\mathrm{g}(x) - E_{\mathcal{D}}[\mathrm{f}_{\mathcal{D}}(x)])^2}_{\text{bias}} + \underbrace{E_{\mathcal{D}}[(\mathrm{f}_{\mathcal{D}}(x_i) - E_{\mathcal{D}}[\mathrm{f}_{\mathcal{D}}(x_i)])^2]}_{\text{variance}} + \underbrace{(y_i - \mathrm{g}(x_i))^2}_{\text{noise}}$$

Diagnostic


UNDERFITTING
OVERFITTING
validation
error
bias
variance
training
model complexity

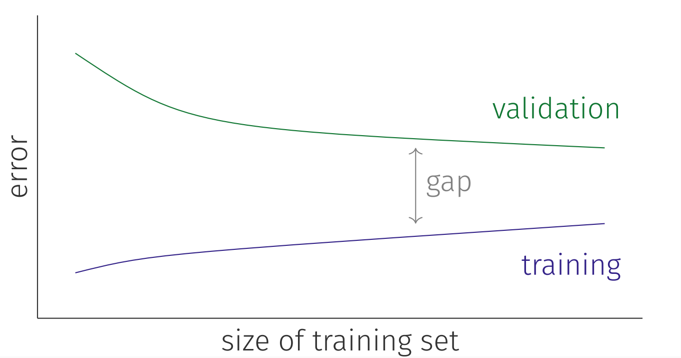
High bias model—Underfitting

- Quick convergence to high error on training and validation sets
- More training data brings little improvement
- Increase model complexity to allow better fit



High variance model–Overfitting

- Performance gap between training and validation sets
- More training data can bring improvement
- Limit model complexity by using e.g. regularization, pruning, early-stopping



Bias vs. variance

higher bias lower variance	lower bias higher variance
lower model complexity	higher model complexity
shallow decision tree	deep decision tree
k-NN with many neighbors	k-NN with few neighbors
linear SVM	kernel SVM
SVM RBF kernel large σ	SVM RBF kernel small σ