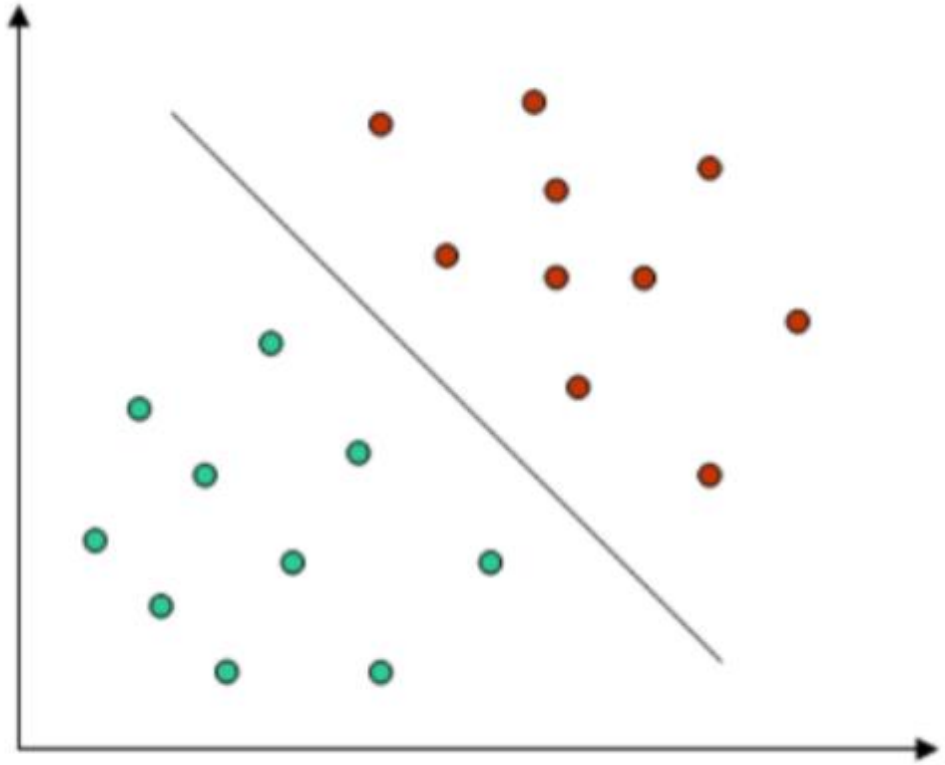
FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Geo.BigData(Science)
Machine Learning (ML)

Supportvektormaschinen, Random Forest, Ensemble Methoden

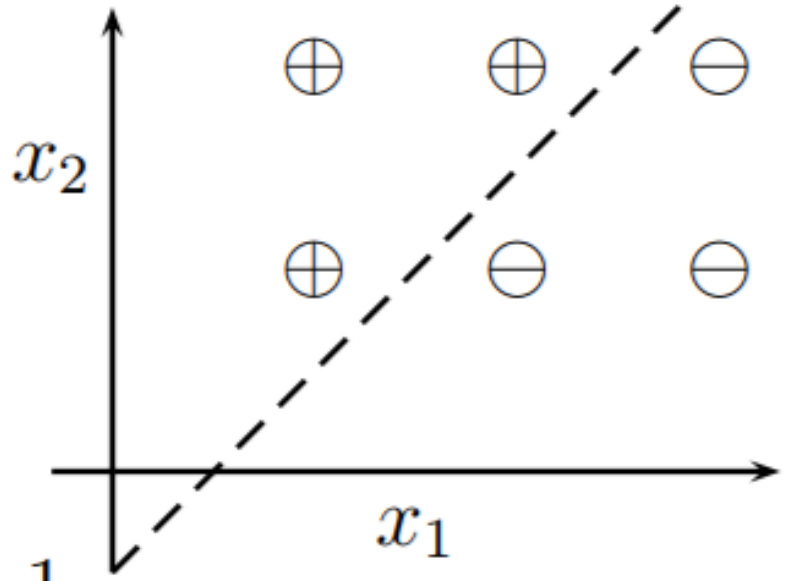
Dr. Joachim Steinwendner

Lineare Separierbarkeit von Klassen



Lineare Separierbarkeit von Klassen

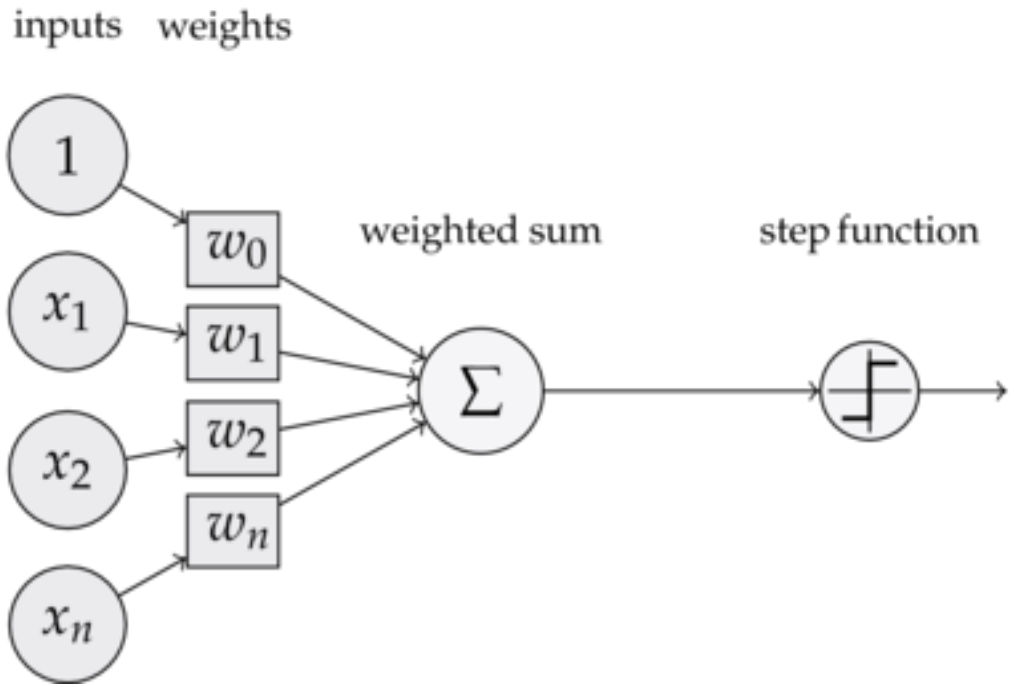
F1	F2	Klasse
1	1	$\oplus$
1	2	$\oplus$
2	1	$\ominus$
2	2	$\oplus$
3	1	$\ominus$
3	2	$\ominus$



$$f(x_1, x_2) = -x_1 + x_2 + \frac{1}{2}$$

Perzeptron

Ein *Neuron – Graphische vs. Mathematische Darstellung*



Perzeptron

Wesentliche Elemente:

Gleichungen: Übliche Aktivierungsfunktionen für Perzeptrons

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

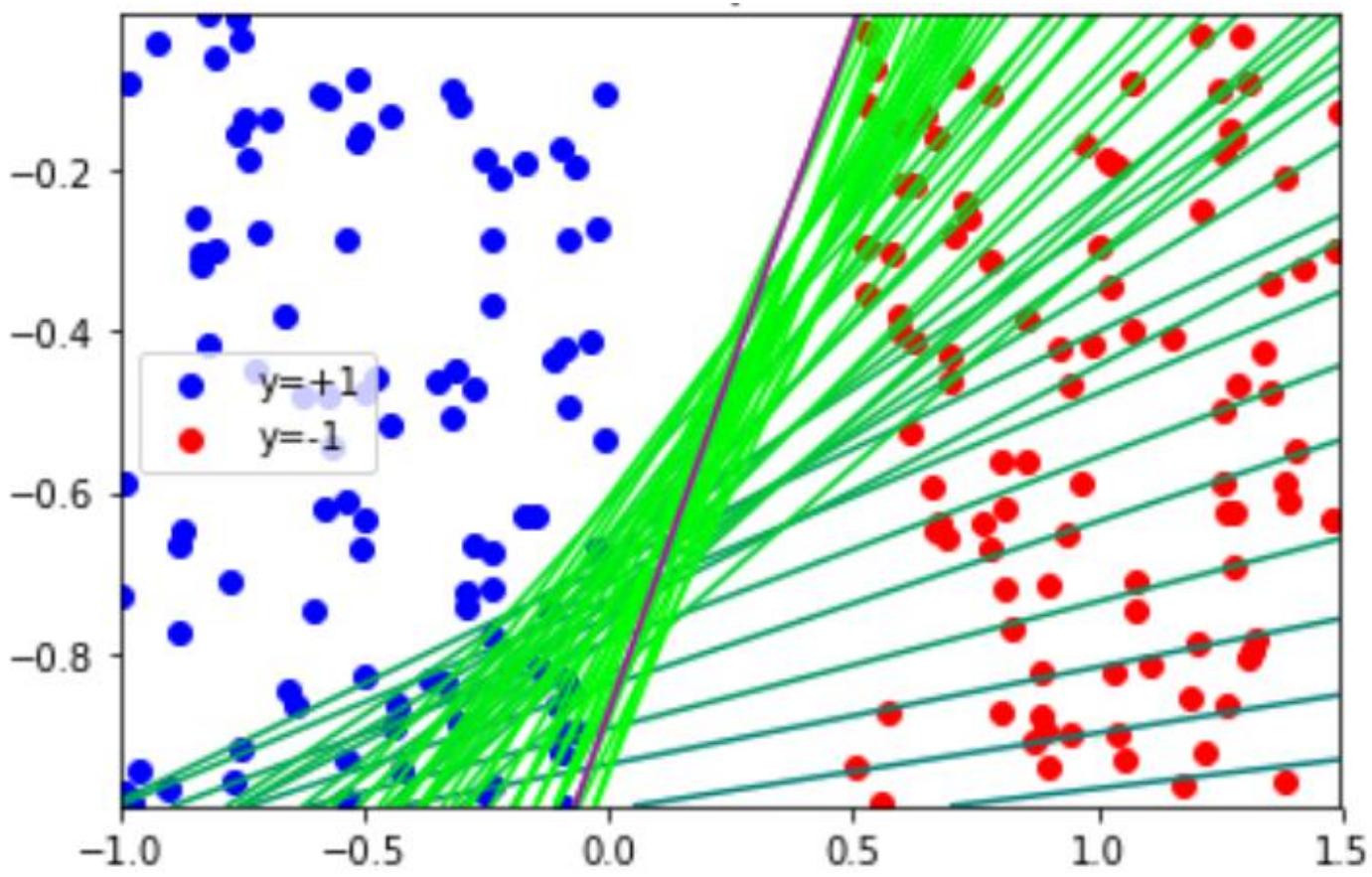
Perzeptron Lernregel (Gewichtsänderung)

$$w_i^{(\text{neu})} = w_i^{(\text{alt})} + \eta(y - \hat{y})x_i$$
$$\vec{w}^{(\text{neu})} = \vec{w}^{(\text{alt})} + \eta(y - \hat{y})\vec{x}$$

wobei

η = Lernrate und $(y - \hat{y})$ = der Fehler (erwarteter abzüglich berechneten Wert)

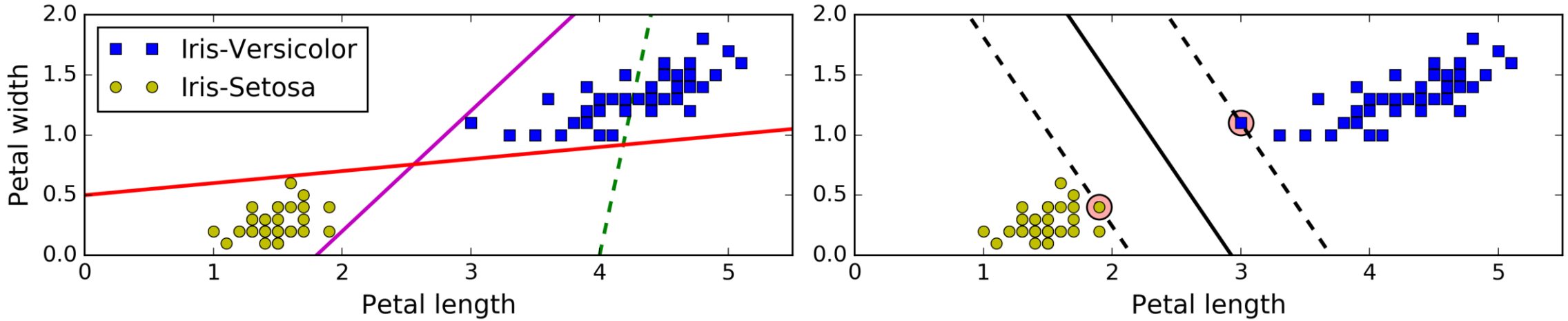
Perzeptron



Perzeptron-Notebook

Supportvektormaschinen

Finde optimale Hyperebene definiert durch Supportvektoren



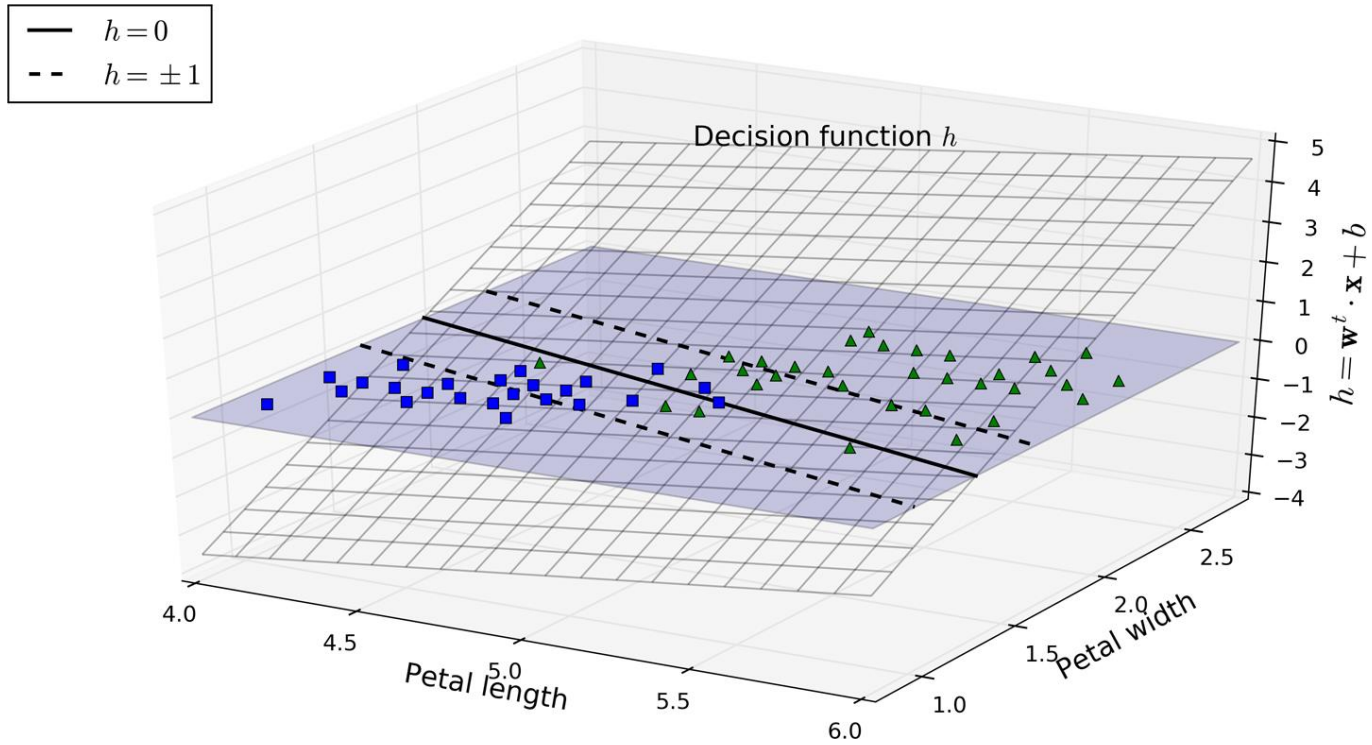
Optimierungsproblem für lineare Supportektormaschinen

Minimiere: $$\vec{w}^*, d^* = \arg\min_{\vec{w},d} |\vec{w}| = \frac{1}{2} \arg\min_{\vec{w},d} |\vec{w}|^2$$

$$\text{mit } y_i(\vec{w} \cdot \vec{x}^i + d) \geq 1 \text{ für alle } x^i$$

Supportvektormaschinen

Finde optimale Hyperebene definiert durch Supportvektoren



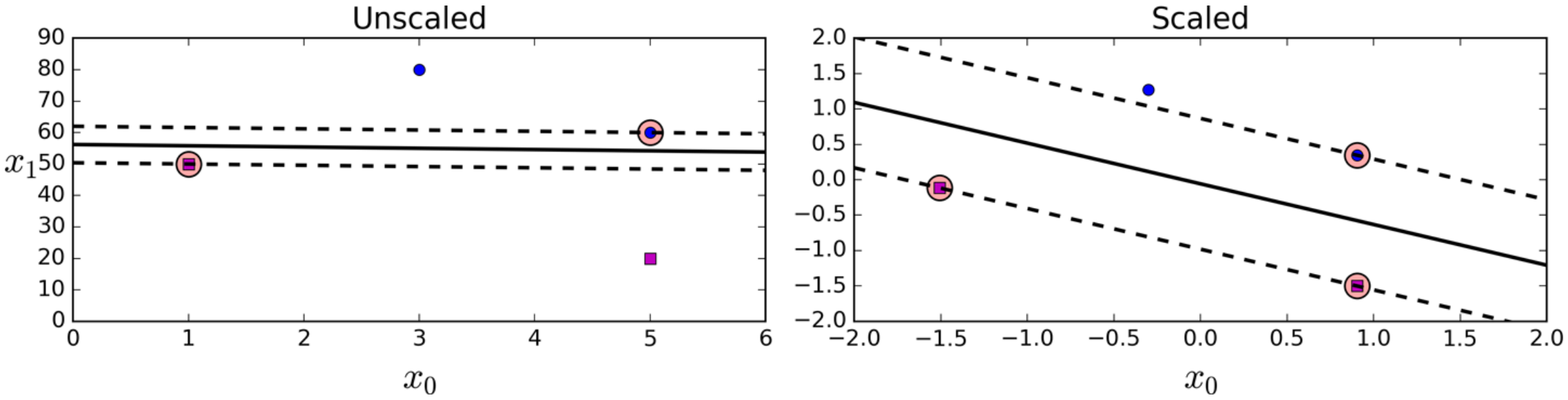
Optimierungsproblem

Finde Werte für w, so dass der Margin möglichst gross wird.

Supportvektormaschinen

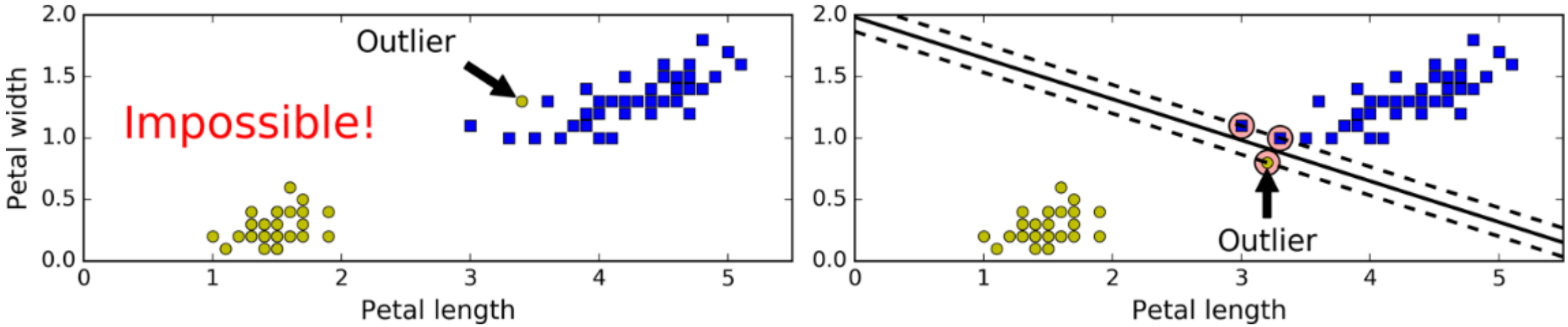
Skalierung der Trainingsdaten

- Supportvektormaschinen sind empfindlich gegenüber Skalierung
- Mit Skalierung wird auch zentralisiert (Linear Supportvektorklassifizierer regularisieren den Bias-Term)



Supportvektormaschinen

Hard- vs. Soft Margin



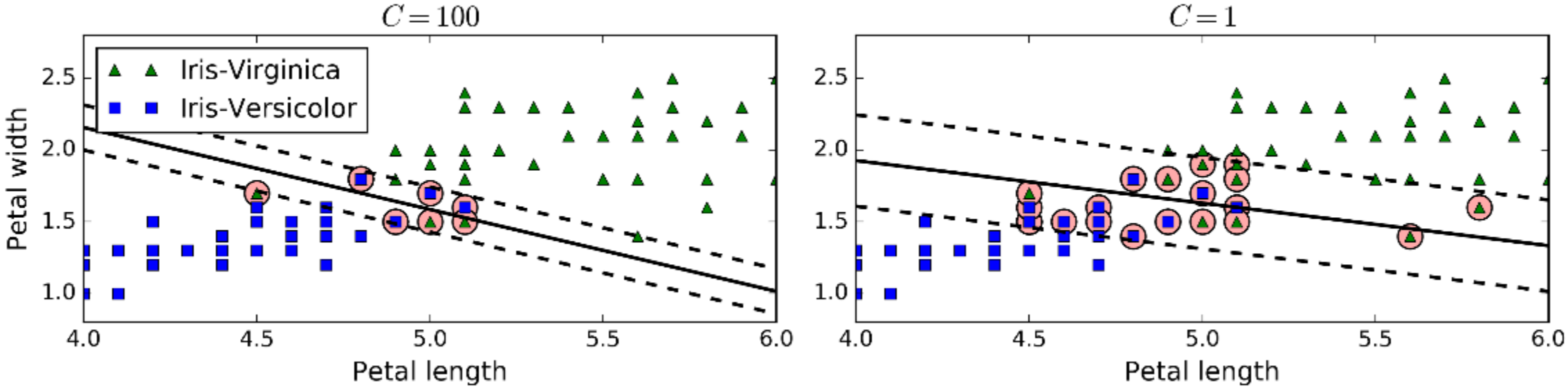
Supportvektormaschinen

Hard- vs. Soft Margin

- Schlupfvariable

Minimiere: $y_i \cdot (\vec{w} \cdot \vec{x}^i + d) \geq 1 - \xi_i$ für alle x^i

mit $|\vec{w}|^2 + C \sum_i \xi_i$



Bei Overfitting -> Reduziere C

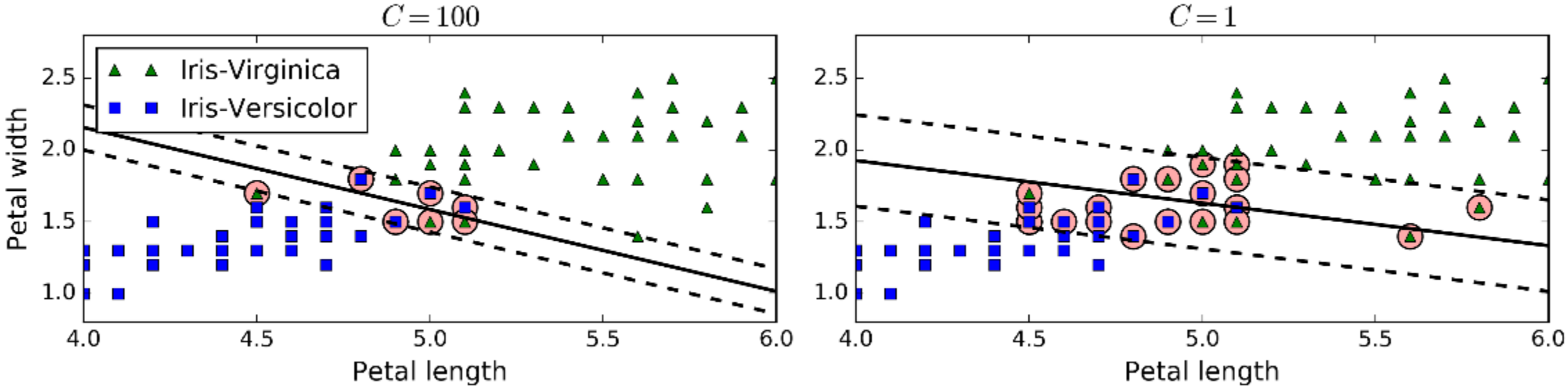
Supportvektormaschinen

Hard- vs. Soft Margin

- Schlupfvariable

$$y_i \cdot (\vec{w} \cdot \vec{x}^i + d) \geq 1 - \xi_i \text{ für alle } x^i$$

$$\text{mit} \quad |\vec{w}|^2 + C \sum_i \xi_i$$



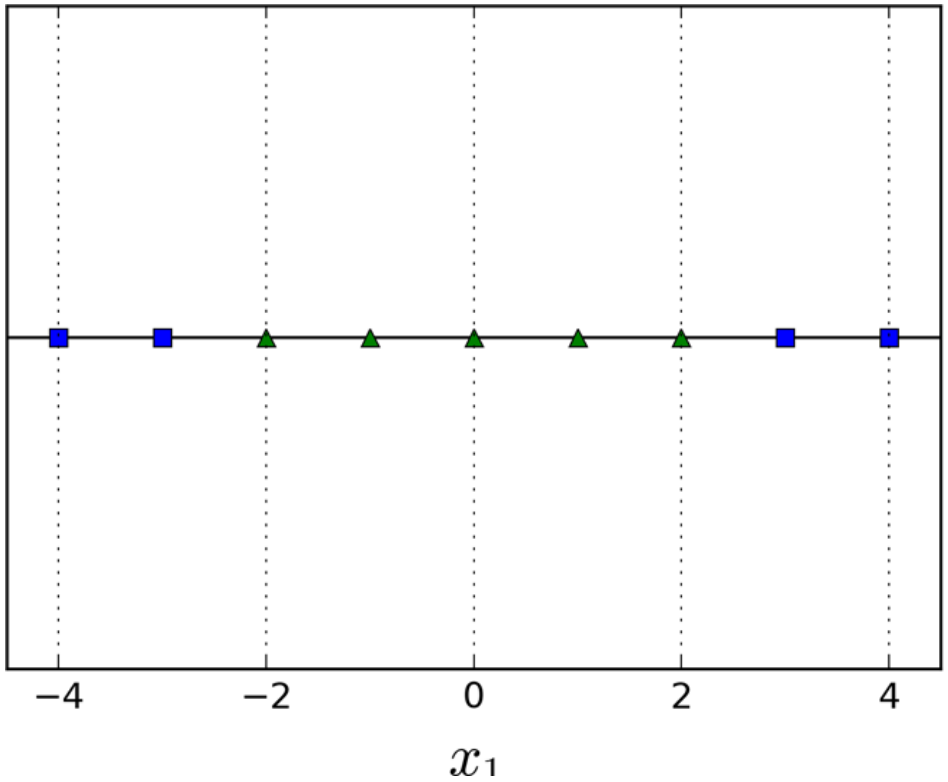
Bei Overfitting -> Reduziere C

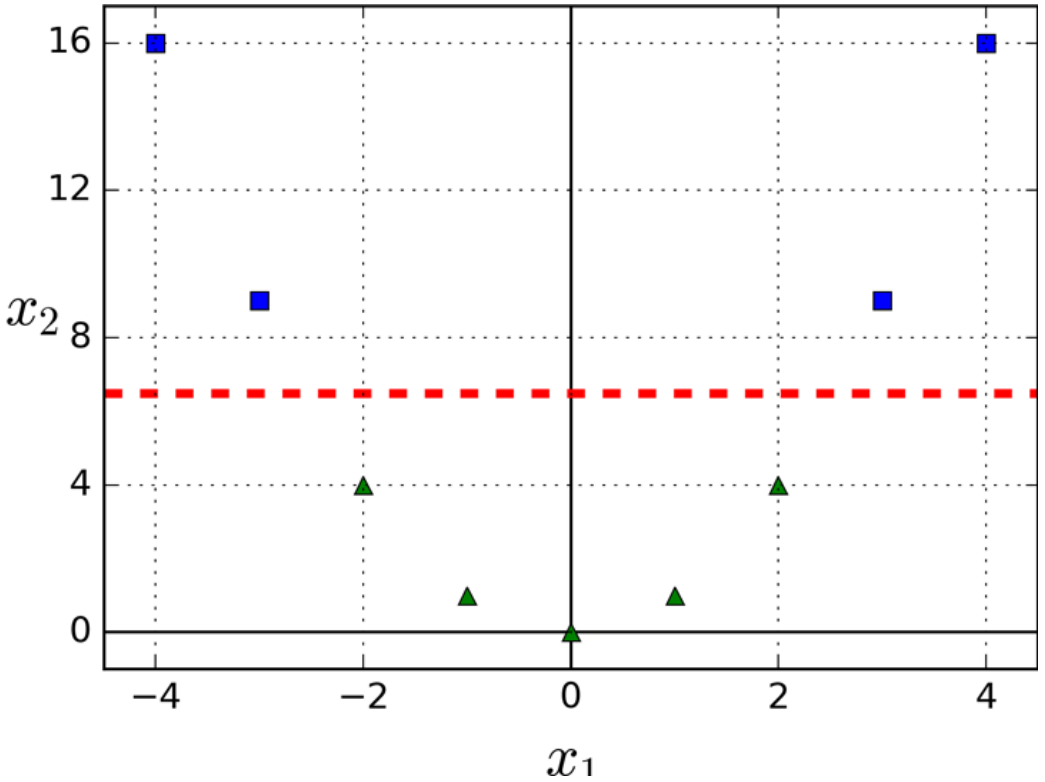
Supportvektormaschinen

Der Kerneltrick

- Features transformieren um einen Datensatz linear trennbar zu machen (PVA2)

- Features hinzufügen um einen Datensatz linear trennbar zu machen





Supportvektormaschinen

Der Kerneltrick

$$k(\vec{x}, \vec{y}) := \Phi(\vec{x}) \cdot \Phi(\vec{y}) \tag{18}$$

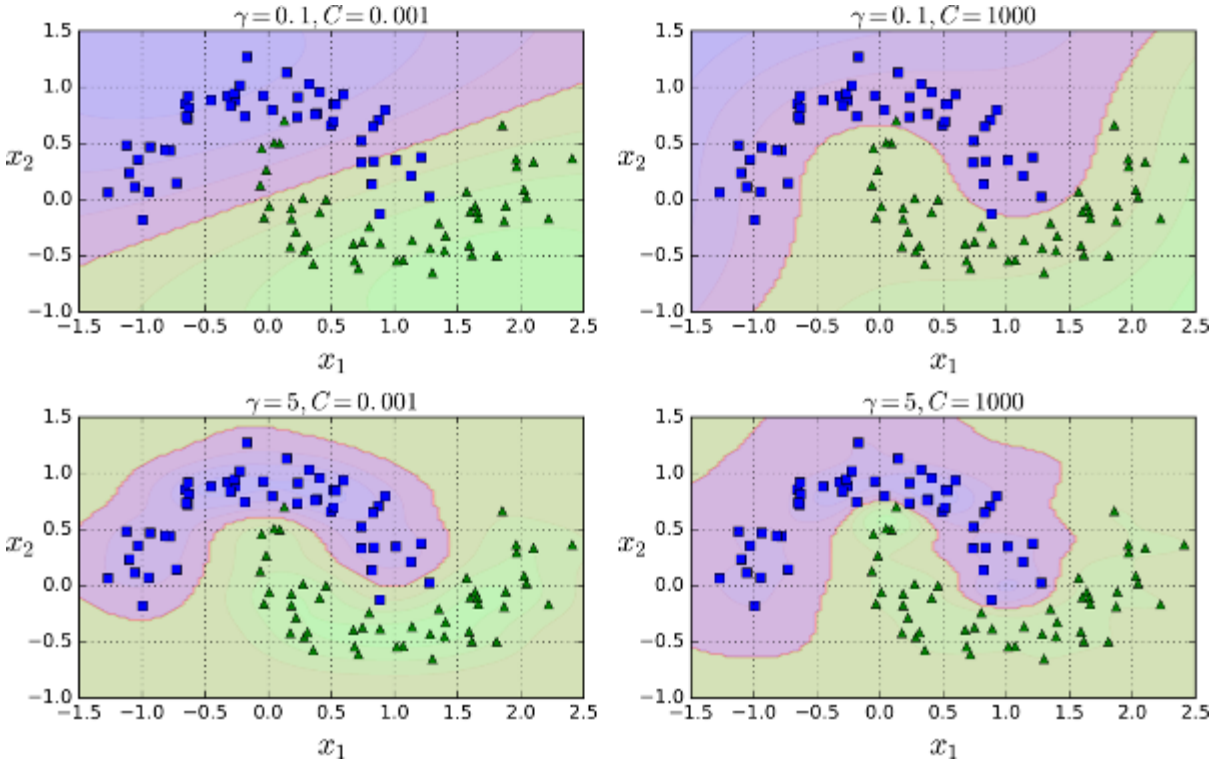
Folgende Kernfunktionen werden häufig verwendet:

- Linearer Kern: Ist eigentlich nur eine Sprechweise dafür, dass die Kernel-Technik nicht verwendet wird. Φ ist die Identität und $k(\vec{x}, \vec{y}) := \vec{x} \cdot \vec{y}$
- Polynomialer Kern: $k(\vec{x}, \vec{y}) := (\vec{x} \cdot \vec{y} + c)^d$
  c und d (Degree) sind Parameter, die spezifiziert werden können.
- Gauß-Kernel oder rbf-Kernel (rbf: radial basis function):
  $k(\vec{x}, \vec{y}) := e^{-\gamma \cdot |\vec{x} - \vec{y}|^2}$
  γ ist ein Parameter für diesen Kernel.

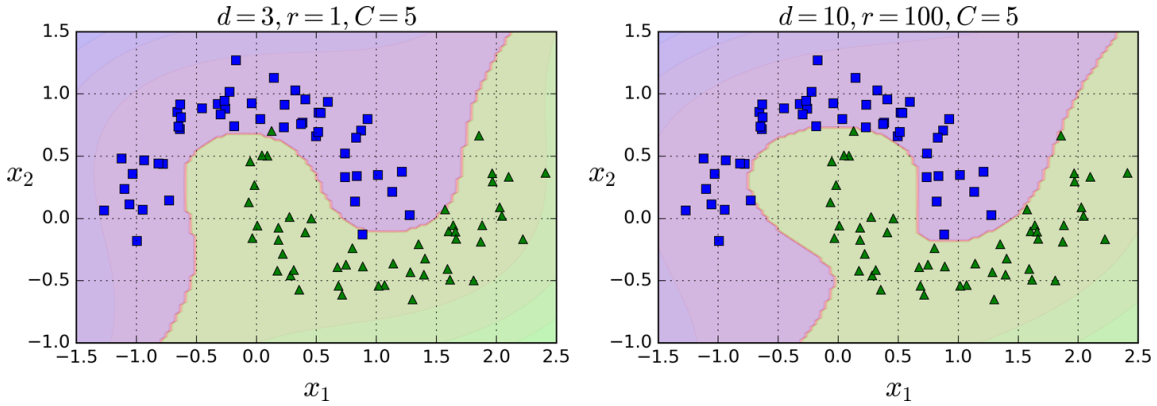
Supportvektormaschinen

Der Kerneltrick

Gausscher RBF Kernel



Polynomialer Kernel



Supportvektormaschinen

Berechnungskomplexität

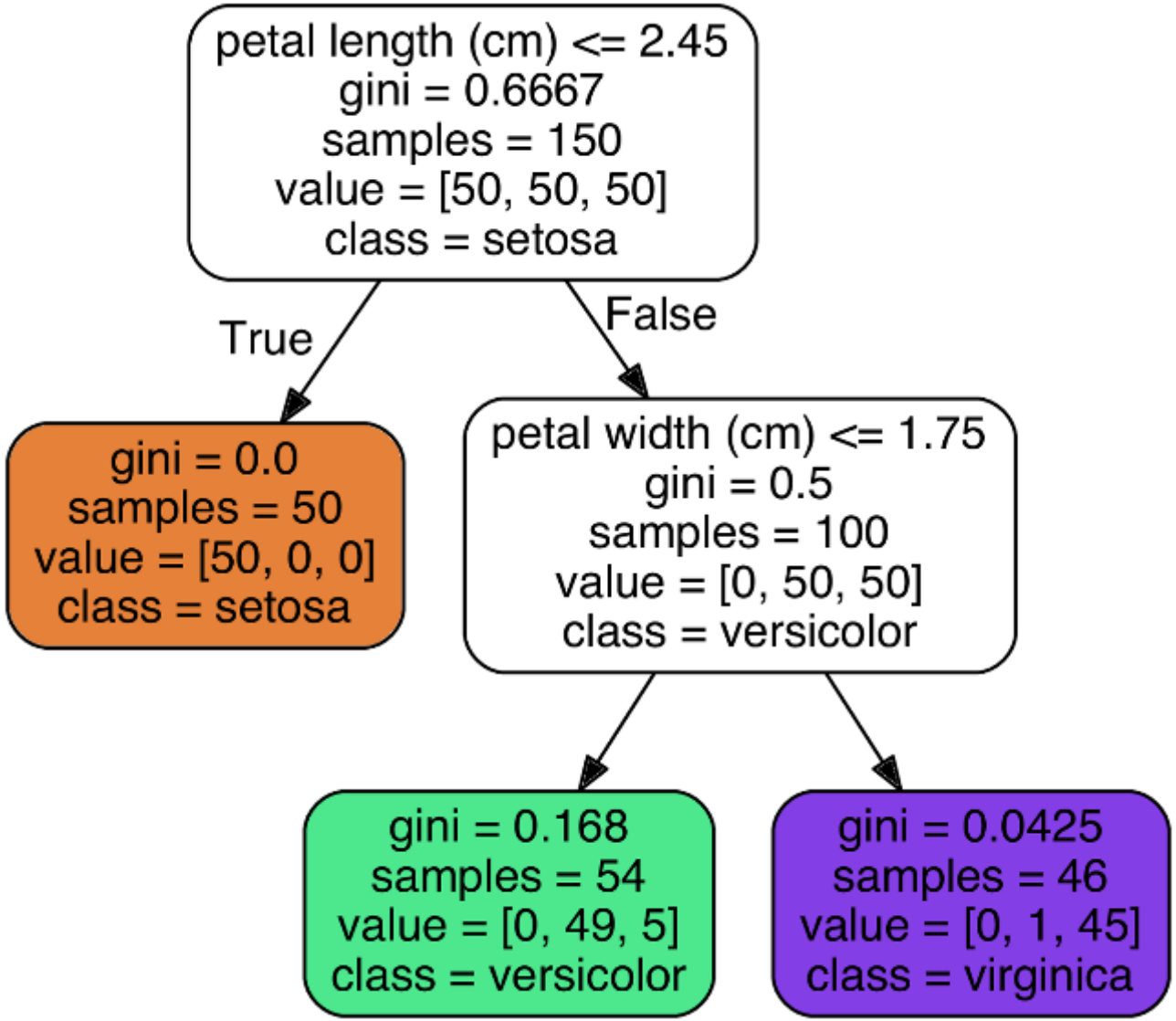
`LinearSVC`	$O(m \times n)$
`SGDClassifier`	$O(m \times n)$
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$

Supportvektormaschinen Notebook

Entscheidungsbäume

Wie funktioniert der Decisiontree Algorithmus?

Am Beispiel des Iris-Datensatzes unter Verwendung on Petal Länge und Breite

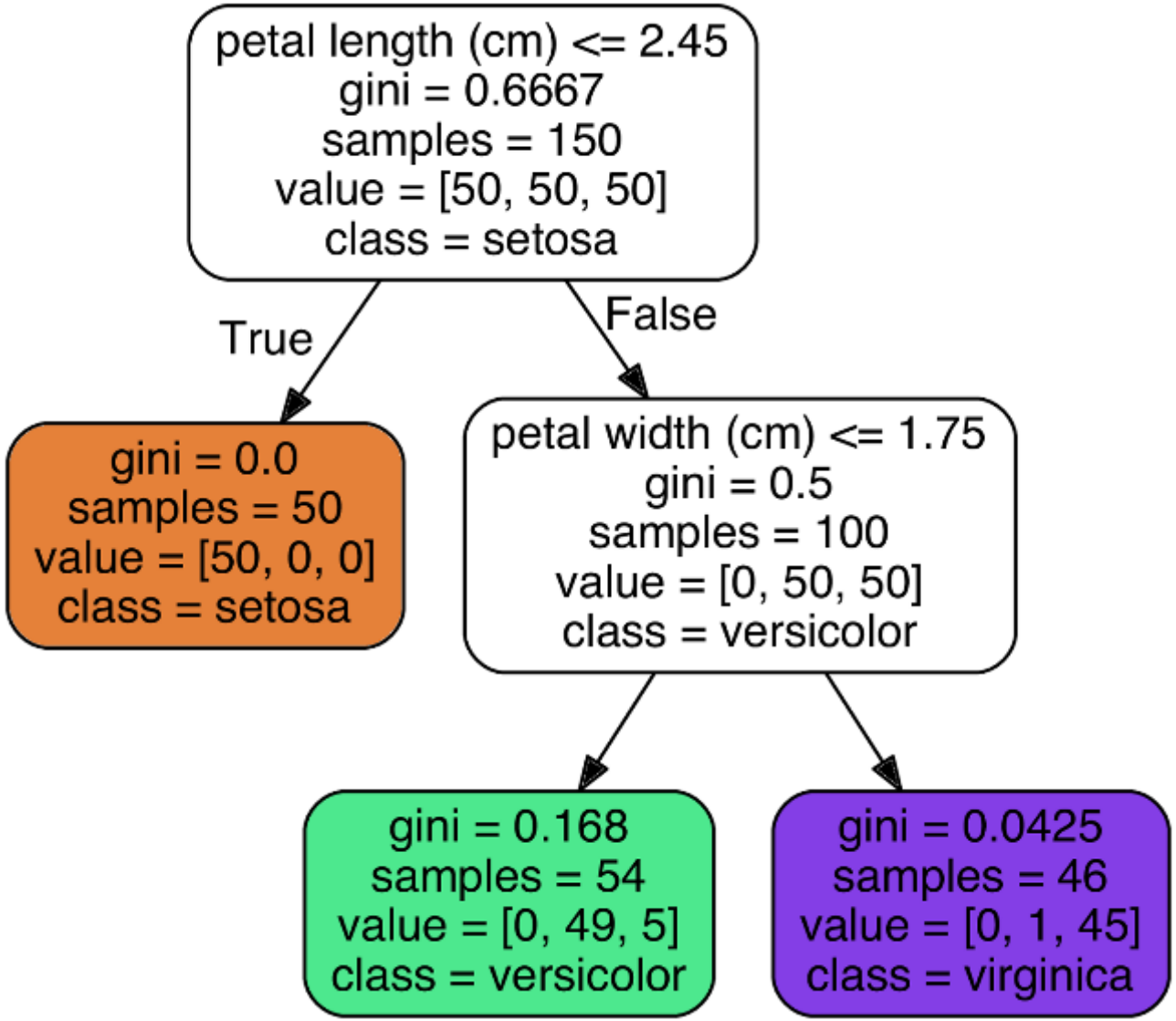


Entscheidungsbäume

Gini impurity

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

$p_{i,k}$ = Verhältnis der Anzahl der Klasse k Instanzen zu den Trainingsinstanzen im i^{ten}Knoten

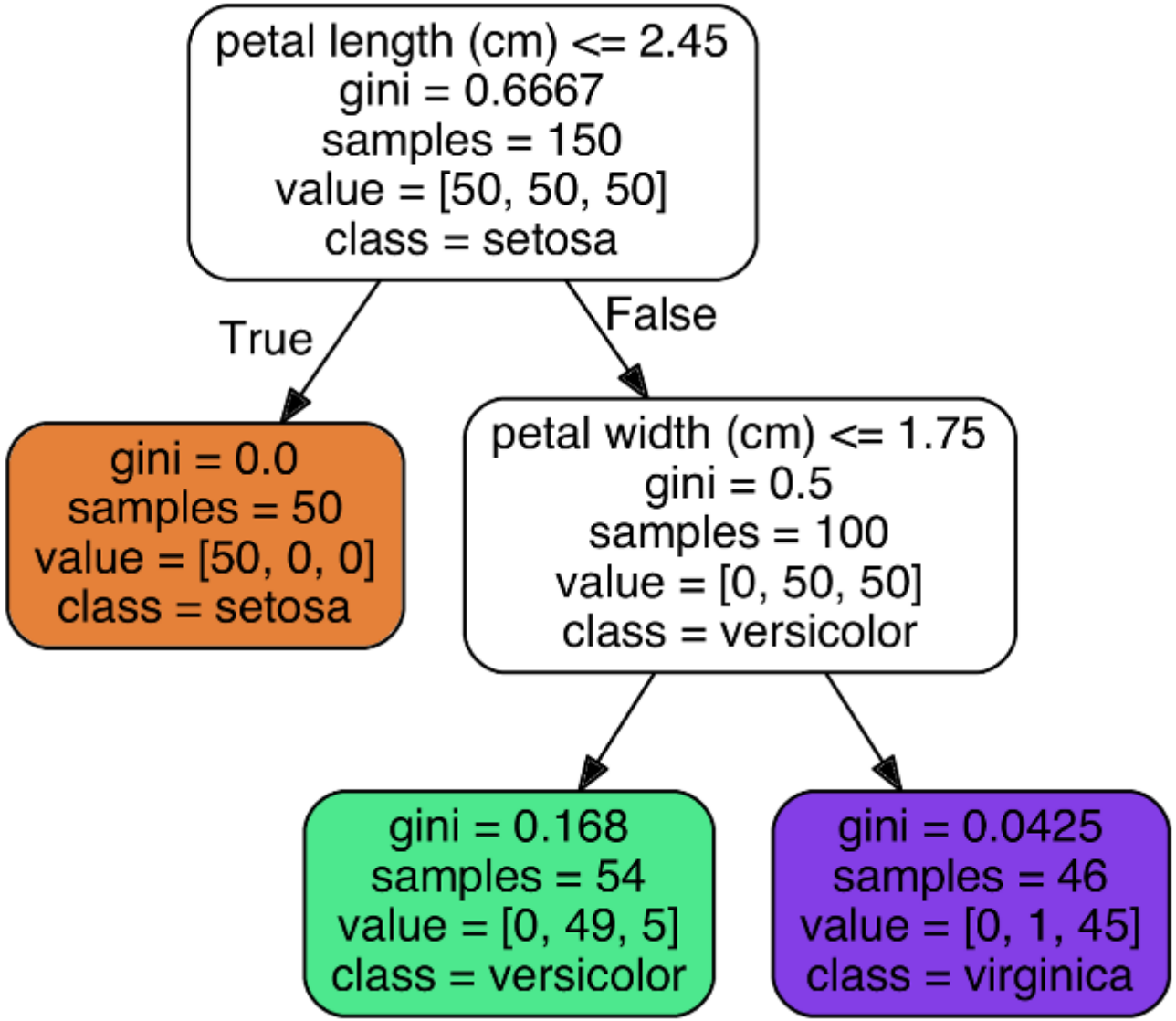


Entscheidungsbäume

Entropie

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log\left(p_{i,k}\right)$$

$p_{i,k}$ = Verhältnis der Anzahl der Klasse k Instanzen zu den Trainingsinstanzen im i^{ten}Knoten



Entscheidungsbäume

Der Trainingsalgorithmus

1. Splitte das Trainingsset in zwei Subsets unter Verwendung eines Features k und einer Grenze (threshold) t_k

2. Wähle jenes Paar (k, t_k), dass die «reinsten (most pure)» Subsets liefert, d.h. minimiere folgende Kostenfunktion

$$J(k, t_k) = \frac{m_{\text{links}}}{m} G_{\text{links}} + \frac{m_{\text{rechts}}}{m} G_{\text{rechts}}$$

$$\text{wo} \begin{cases} G_{\text{links/rechts}} \text{ misst die impurity des linken/rechten Subsets,} \\ m_{\text{links/rechts}} \text{ ist die Anzahl von Instanzen des linken/rechten Subsets.} \end{cases}$$

3. Wiederhole Schritt 1 und 2 für die beiden Subsets

4. Stoppe wenn max_depth erreicht oder es gibt keinen split der die Impurity verringert.

Rekursiver Algorithmus!

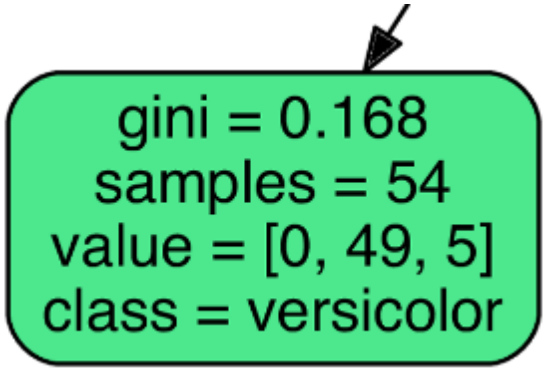
Entscheidungsbäume

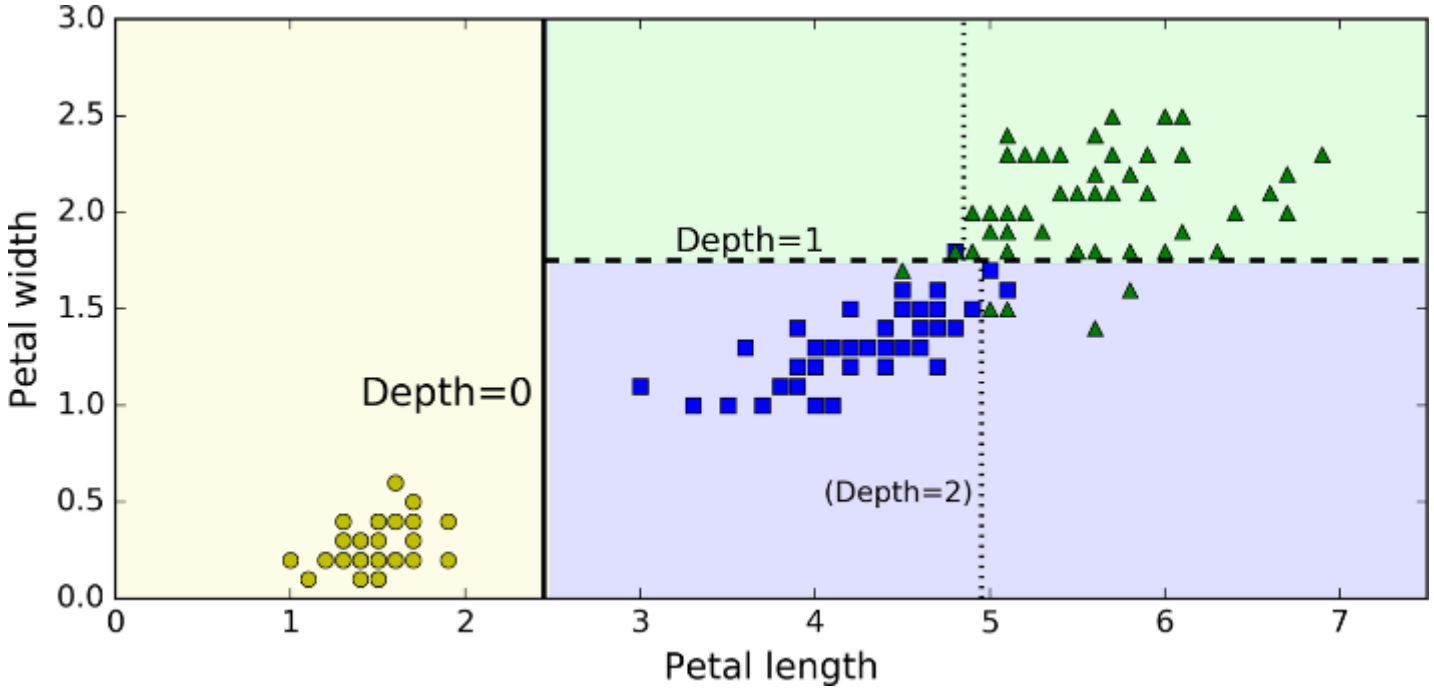
Berechnungskomplexität

Training	$O(e^m)$ für optimalen Baum ->schlecht, weil *NP-Complete,*
	algorithmische Veränderung und suboptimalen Baum $O(n * m \log(m))$
Klassifikation	$O(log_2(m))$

Entscheidungsbäume

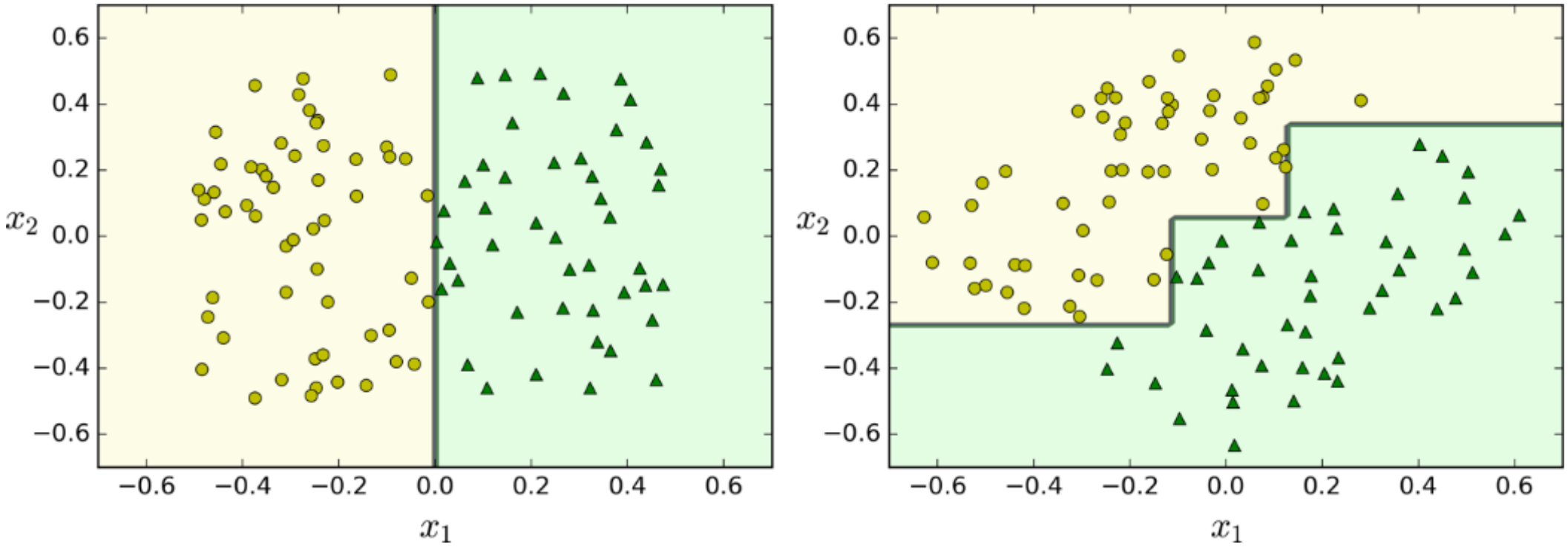
Schätzen der Klassenwahrscheinlichkeiten





Entscheidungsbäume

Instabilität – Sensibel auf einfache Trainingsdatenänderungen



Entscheidungsbaum Notebook

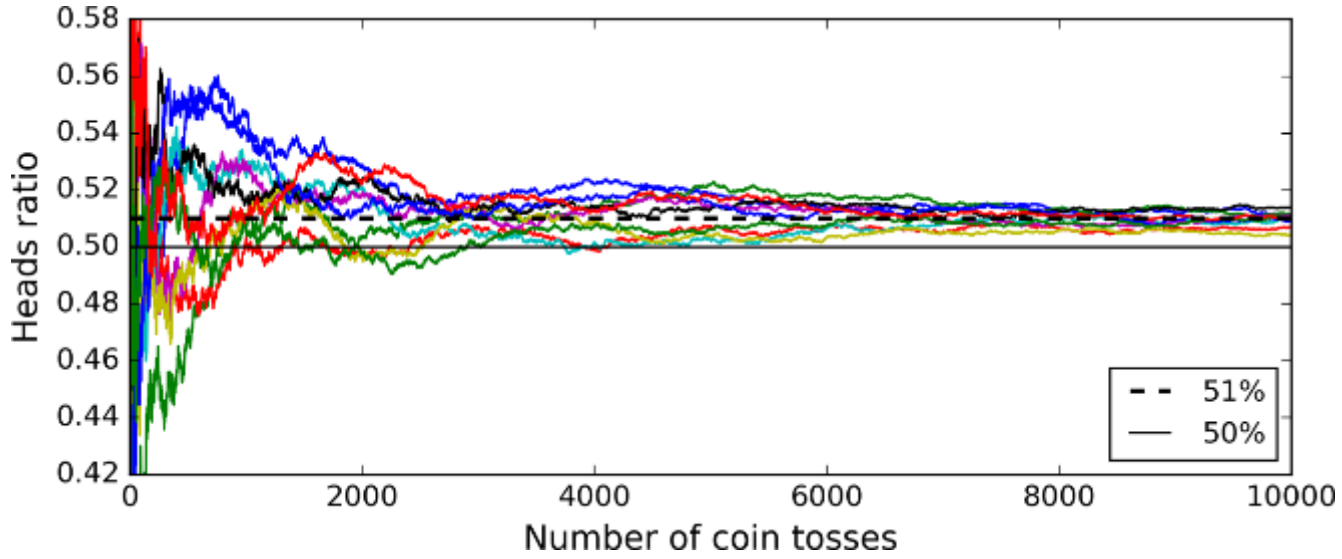
Ensemble Methoden - Übersicht

Eine Gruppe von Klassifikatoren wird als Ensemble bezeichnet.

- No-free-lunch Theorem

Warum funktioniert das? Ist die Kombination mehrerer Klassifikatoren besser als EIN Klassifikator?

- Gesetz der grossen Zahlen (oder wisdom of the crowd)



Ensemble Methoden - Übersicht

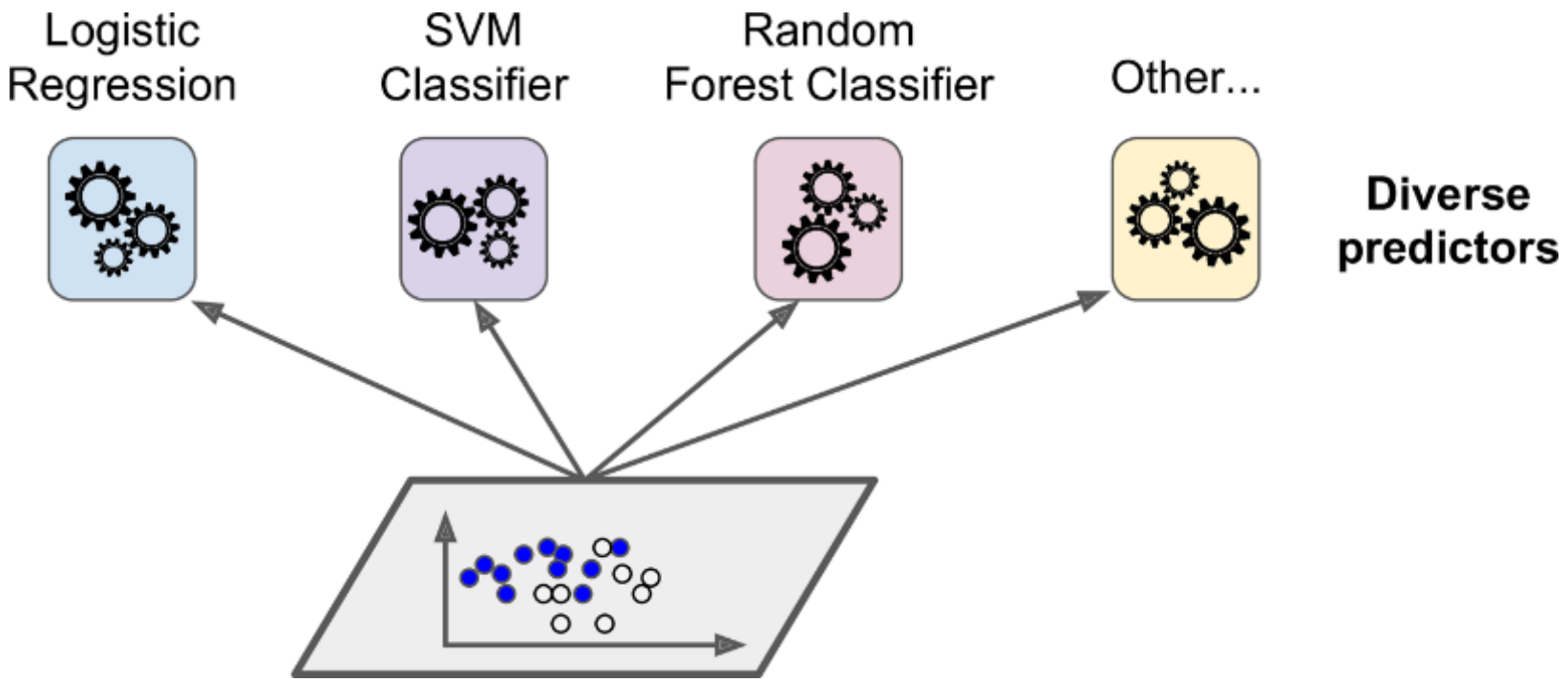
Mögliche Methoden

- Voting
- Random Forests
- Bagging and Pasting
- Boosting
- Stacking

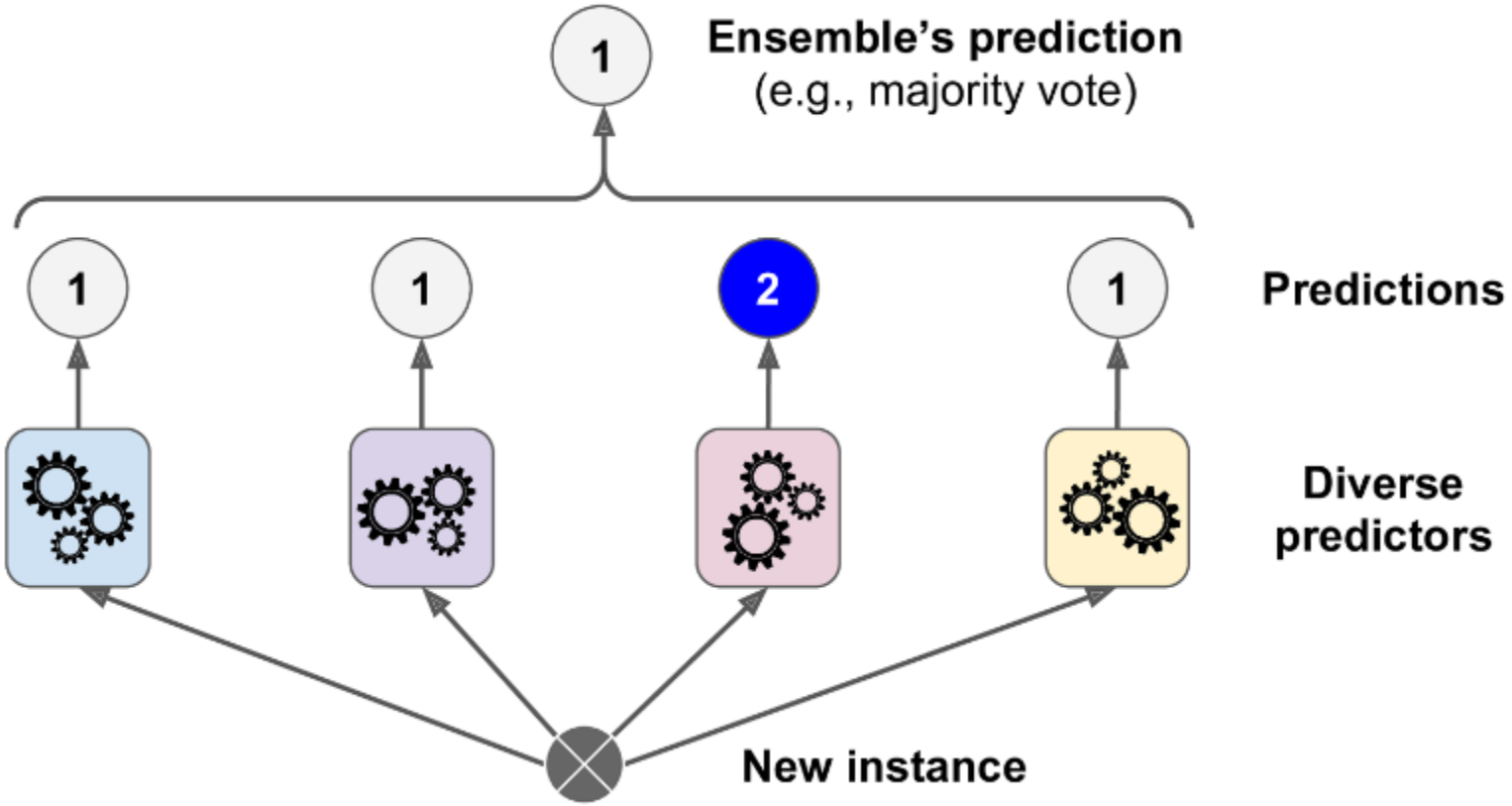
Ensemble Methoden arbeiten am besten, wenn die Klassifizierer voneinander so unabhängig wie möglich sind. Folgende Möglichkeiten das annähernd zu erreichen:

1. Training mit unterschiedlichen Algorithmen/Modellen
2. Training eines Basisklassifkators mit unterschiedlichen Trainingsdaten

Voting

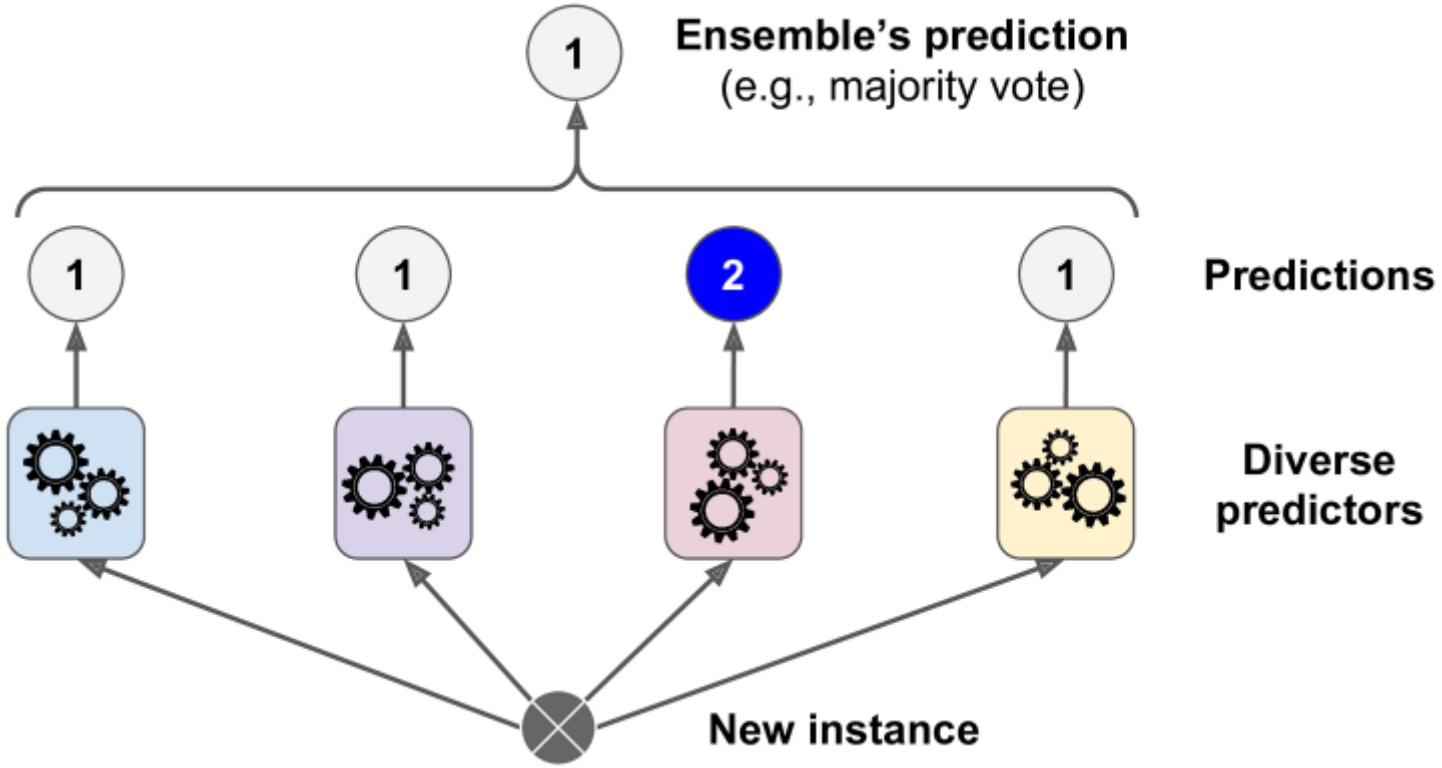

Logistic Regression
SVM Classifier
Random Forest Classifier
Other...
Diverse predictors

Voting


1
Ensemble's prediction
(e.g., majority vote)
1
1
2
1
Predictions
Diverse
predictors
New instance

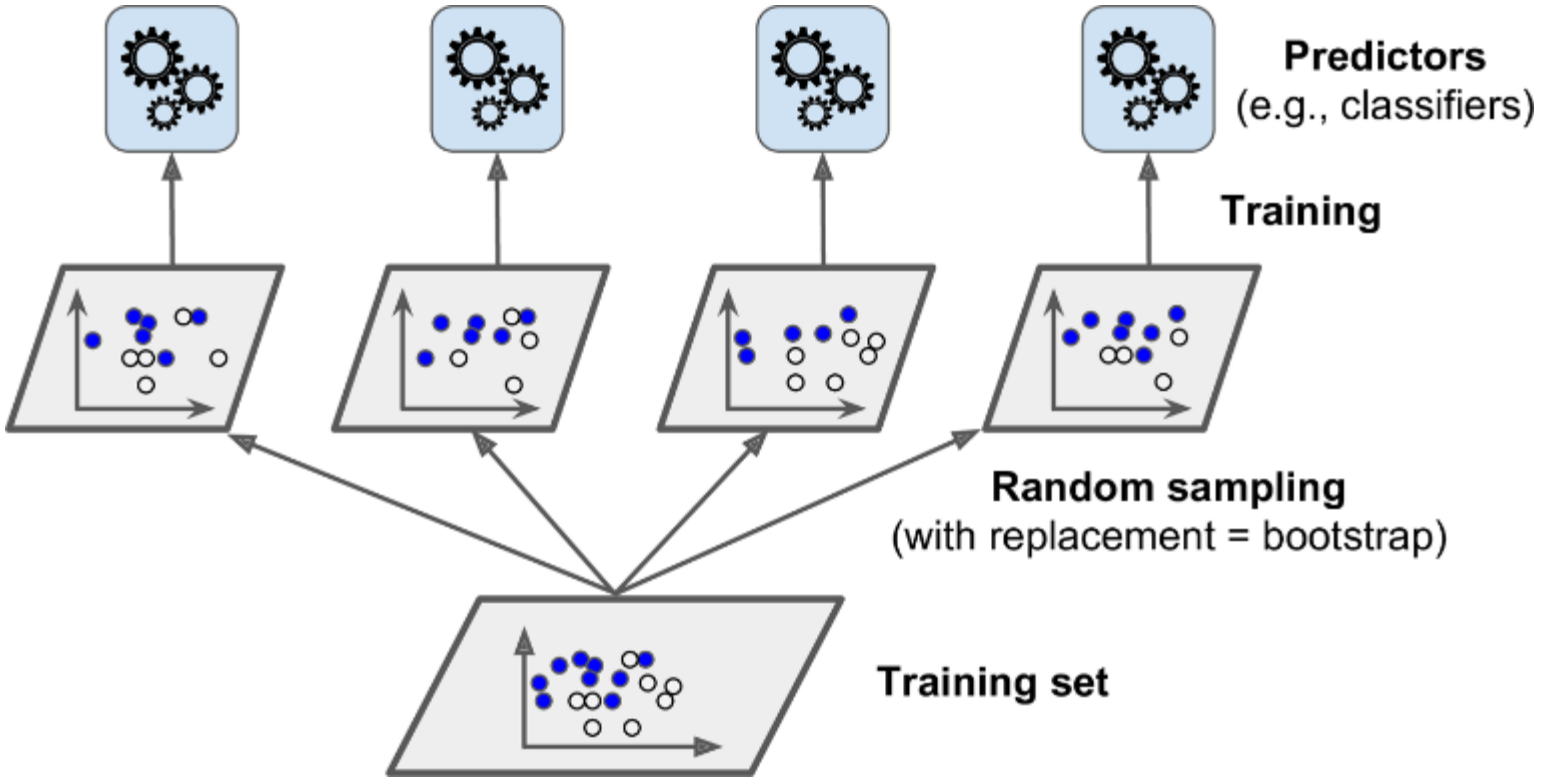
Voting

- Hard Voting (Mehrheit der Klassifikation aller Klassifikatoren) vs.
  Soft Voting (Klassifiziere auf Basis der höchsten Klassenwahrscheinlichkeit – MW aller Klassifikatoren)



Bagging (Bootstrap Aggregating) und Pasting

Basisklassifizierer und zufällige Auswahl von Samples



Bootstrapping

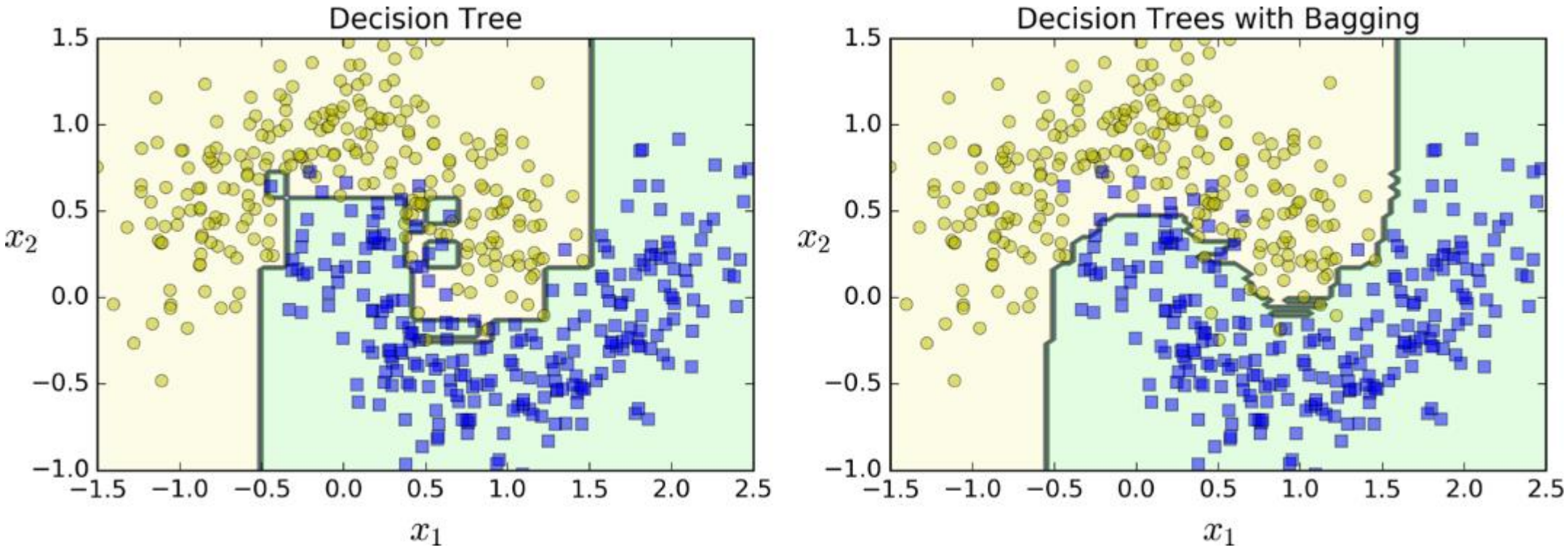
Zufällige Auswahl von Samples MIT Ersetzen

Pasting

Zufällige Auswahl von Samples OHNE Ersetzen

Bagging and Pasting

Vergleich



Bagging and Pasting

- **Out-of-Bag Evaluation**

- **Random Patches** (zufällige Auswahl von Samples und Features)
  Zufälliges Auswählen der Features für den Klassifizierer
  z.B. bootstrap=True, max_features=50 und bootstrap_features=true
  Sinnvoll bei hochdimensionalen Datensätzen (z.B. Bilder)
- **Random Subspace** (Gesamttrainingsdatensatz nur zufällige Auswahl von Features)
  z.B. bootstrap=False und max_samples=1.0, bootstrap_features=True und max_features<=1.0
  Sinnvoll bei hochdimensionalen Datensätzen (z.B. Bilder)

Random Forests

= Bagging mit Decision Trees!

Vergleiche:

```
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)

bag_clf = BaggingClassifier(DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),
                n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1)
```

Random Forests

Extremely Randomized Trees (Extra-Trees)

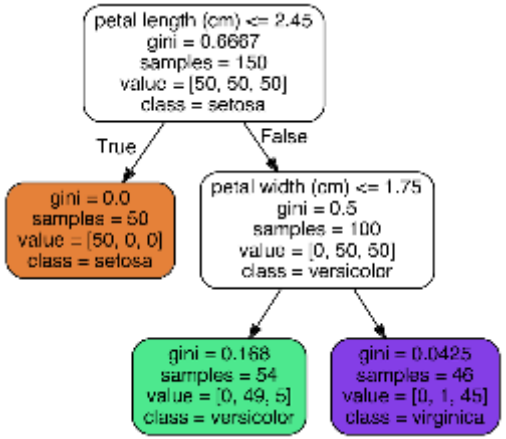
Um die Bäume noch "zufälliger" zu machen, nimmt man zufällige Grenzen (thresholds) für jedes Feature statt die bestmögliche Grenze zu suchen!

- wie immer bei Ensemble Methoden: etwas mehr Bias aber geringere Varianz
- viel schnelleres Trainieren, da keine optimale Grenze für jedes Feature gefunden werden muss

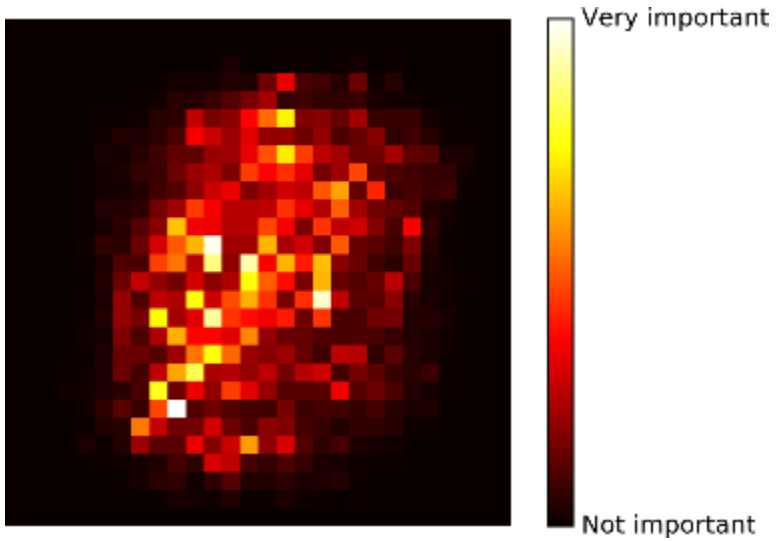
Random Forests

Feature importance

- Bei einfachen Entscheidungsbaum (white-box estimator)

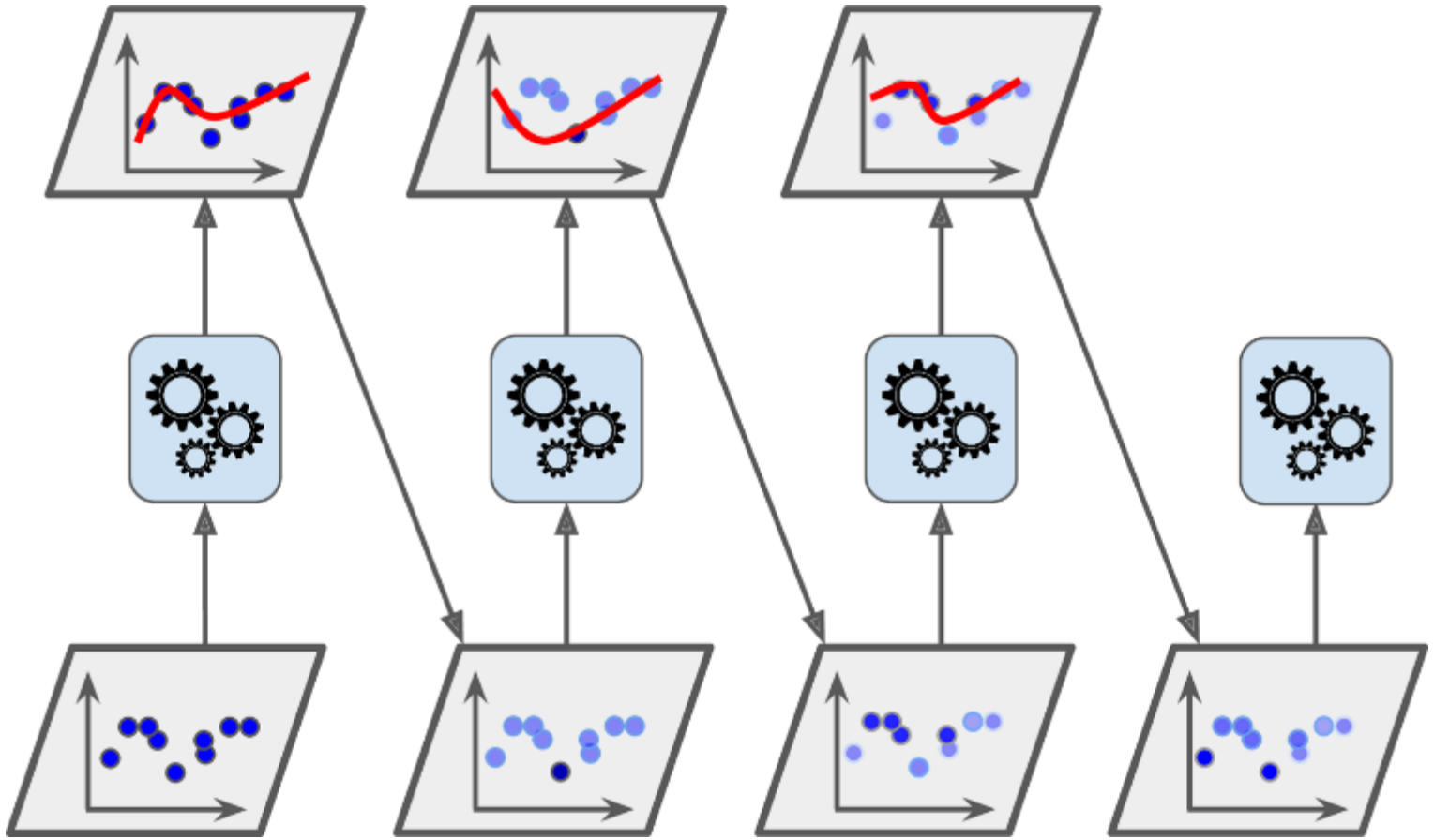


- Random Forests (black-box estimator) aber:



Boosting

Ursprünlich genannt: Hypothesis Boosting



Boosting

- Initialisiere Gewichte: $w_j = 1/N : j = 1, \ldots, N$
- Für $l = 1, \ldots, L$
  - Trainiere einen Klassifikator $f_l(\mathbf{x})$ mit Datengewichten w_j durch Minimierung der gewichteten Missklassifikationsrate $\sum_{j \in E_l} w_j$, wobei $E_l = \{j : f_l(\mathbf{x_j}) \neq y_j\}$
  - Berechne den normalisierten Fehler

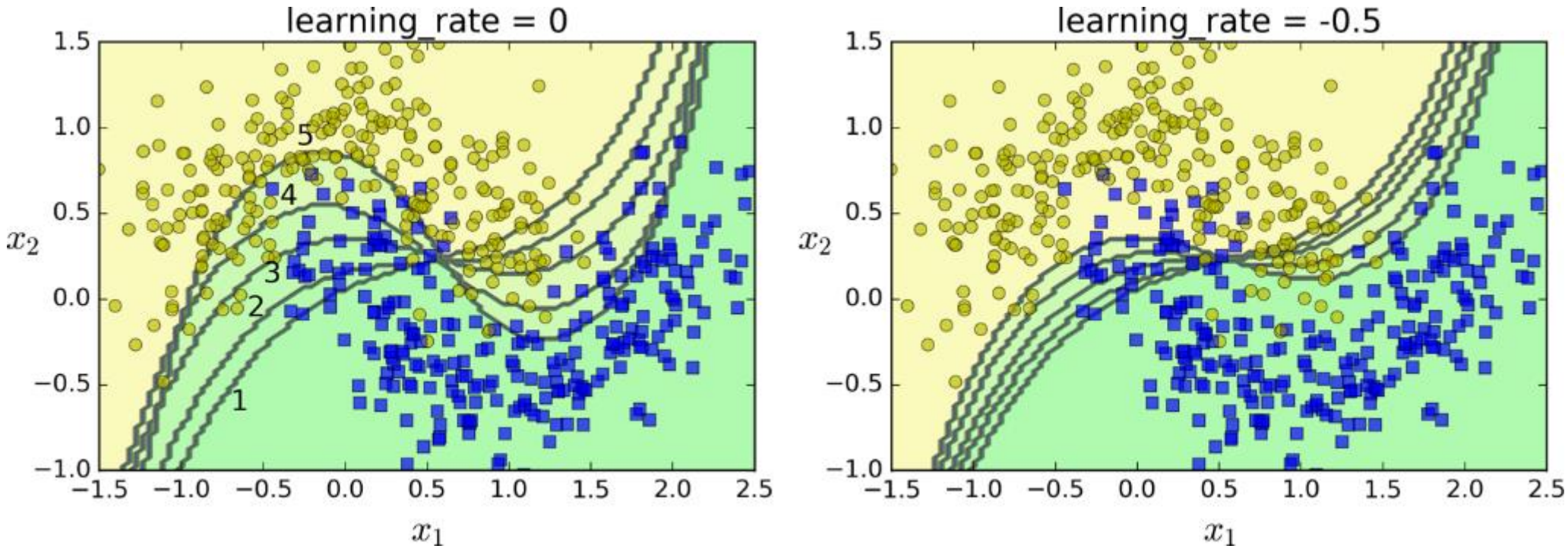
$$\text{err}_l = \frac{\sum_{j \in E_l} w_j}{\sum_{j=1}^{N} w_j}$$

  - Berechne $\alpha_l = \log \frac{1-\text{err}_l}{\text{err}_l}$ (Beachte: $\text{err}_l \leq 0.5$)
  - Berechne neue Datengewichte für alle $\mathbf{x}_j$ mit $j \in E_l$: $w_j \leftarrow w_j e^{\alpha_l}$

Abschließend:

$$f(\mathbf{x}) = \text{sign}\left[\sum_{l=1}^{L} \alpha_l f_l(\mathbf{x})\right]$$

Boosting (adaBoost)



Boosting (adaBoost)

- Gewichtete Fehlerrate von Klassifikator/Predictor *j*

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction for the } i^{\text{th}} \text{ instance.}$$

- Klassifikator Gewichtung a_j

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

Boosting (adaBoost)

- Update der Gewichte der Instanzen/Samples (*m=Anzahl Instanzen)*

$$\text{for } i = 1, 2, \cdots, m$$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

Boosting (adaBoost)

- AdaBoost Klassifizierung (Mehrheit der gewichteten Votes)

$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^{N} \alpha_j \quad \text{where } N \text{ is the number of predictors.}$$

Boosting (Gradient Boosting)

- Anstatt die Instanzengewichte zu verändern, wird bei jedem Schritt der *residual error* des vorgängigen Klassifizieres trainiert!

Boosting (Gradient Boosting)

- Gradient Boosting am Beispiel eines DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

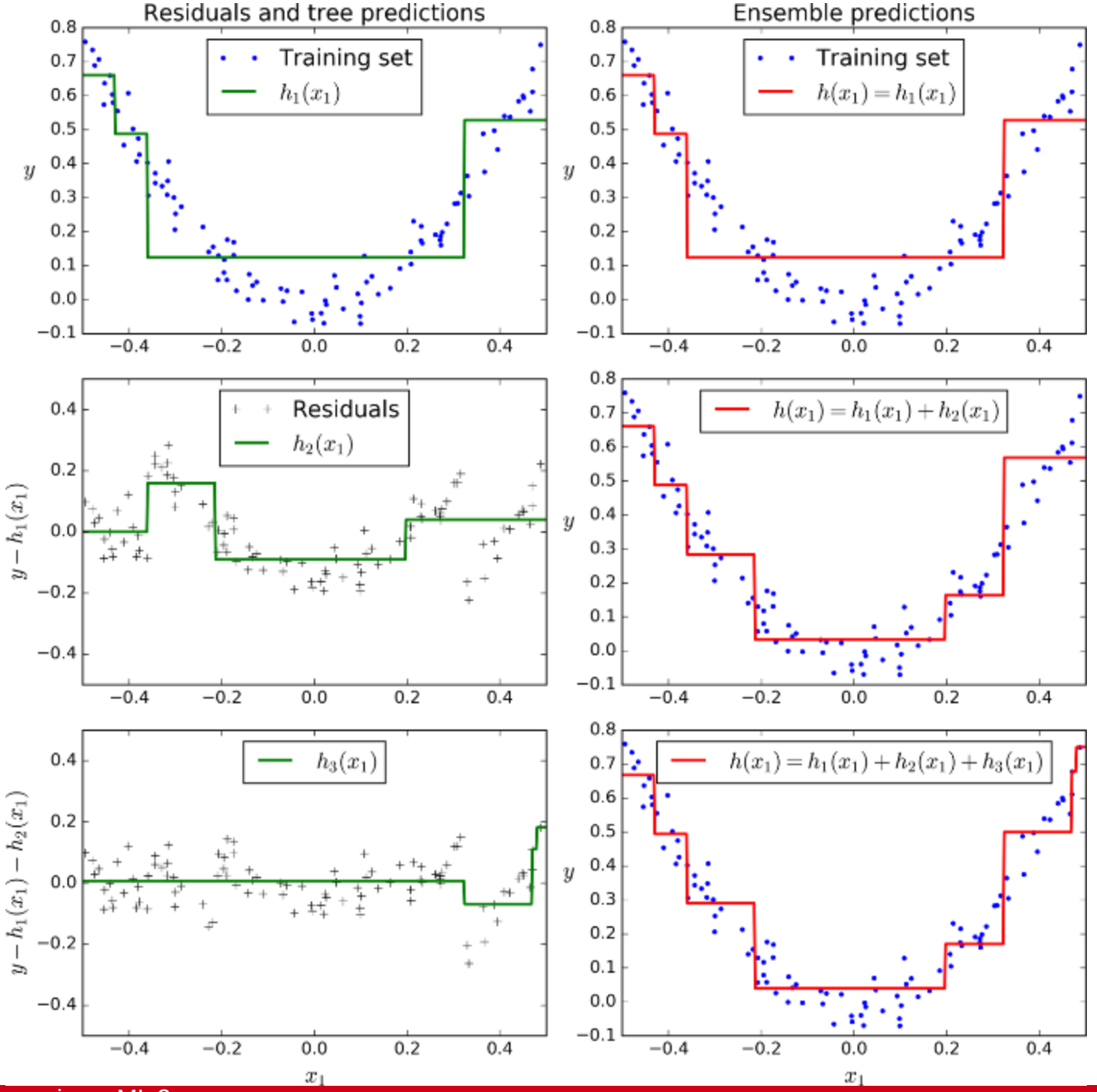
```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

Boosting (Gradient Boosting)

- Gradient Boosting am Beispiel eines DecisionTreeRegressor



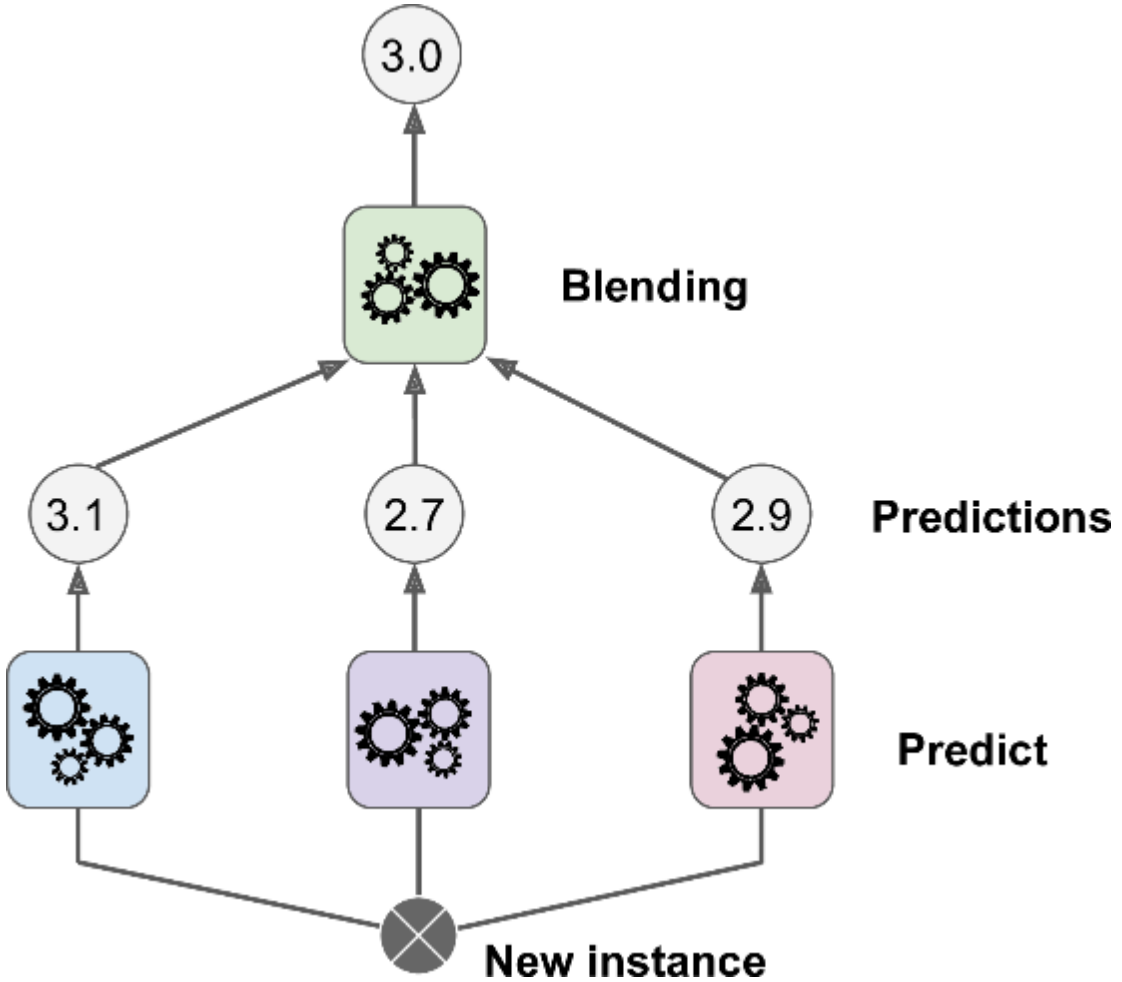
Boosting (xgBoost=eXtreme Gradient Boosting)

Eckpunkte von XGBoost

- Regularisierung
  - Standard GradientBoosting (GBM) hat keine Regularisierung, XGBoost schon!
- Parallel Processing
  - XGBoost ist sehr viel schneller
  - Parallelisierung des Entscheidungsbaums
  - Unterstützt die Verwendung von Hadoop
- Tree Pruning
  - GBM stoppt das Teilen von Knoten bei einem negativem Impurity, XGBoost splittet bis max_depth und startet "pruning" durch Entfernen von Nodes/Splits, wo es keine positive Impurity gibt.

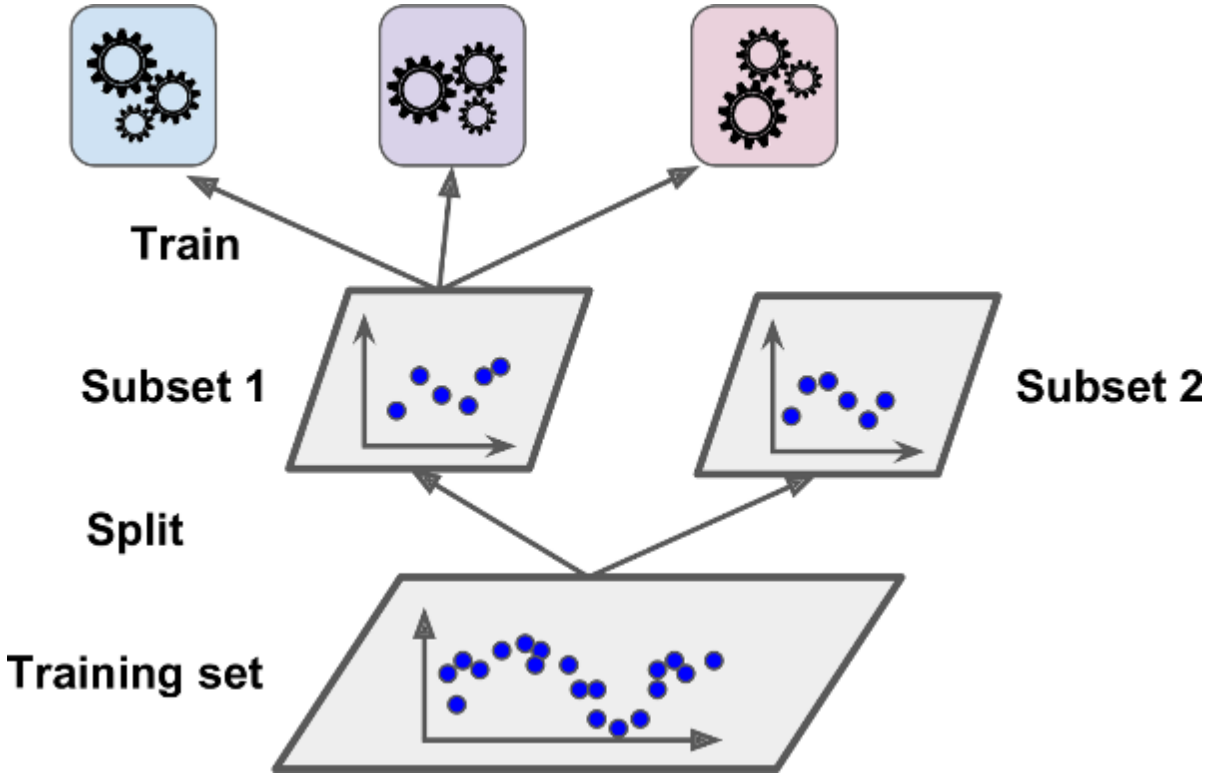
Stacking

- Blending Klassifikator

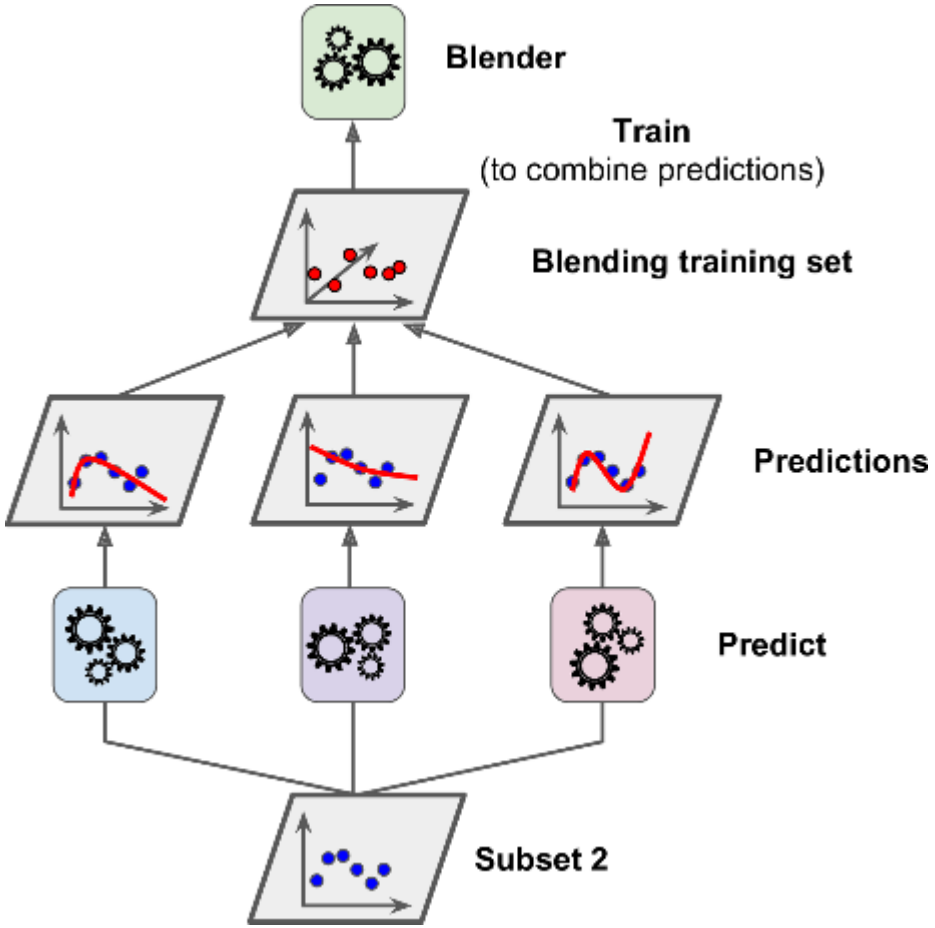


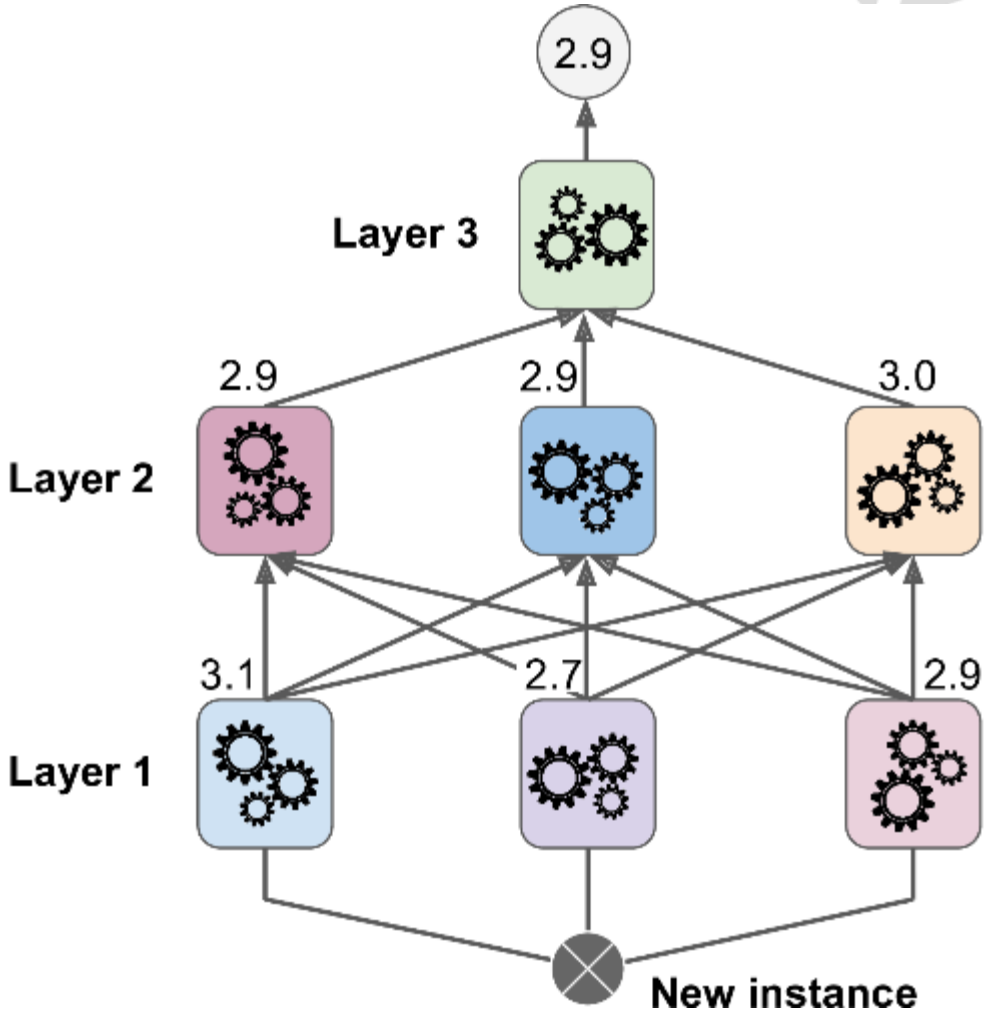
Stacking

1. Teile den Trainingsdatensatz (Training:subset1 und Hold-out-Set:subset2)
2. Trainiere die Klassifizierer mit subset1
3. Klassifiziere das subset2
4. Die Ergebnisse der Klassifizierer bilden das Trainingsset für den Blender



Stacking





Ensemble Notebook