Decision-Tree-Klassifikator

Decision Trees haben einige Vorteile gegenüber den beiden schon beschriebenen Klassifikationsmethoden. Man benötigt in der Regel keine so aufwendige Vorverarbeitung (Preprocessing), weil kategorische Input-Variablen nicht als Zahlen codiert werden müssen. Du kannst sogar numerische und kategorische Features simultan benutzen. Zudem kann eine Rangfolge der relevantesten Features nach der Trainingsphase relativ leicht extrahiert werden. Dies ist vor allem wichtig, wenn du die Entscheidungskriterien des Modells nachvollziehen möchtest. Decision Trees tendieren aber auch zu Overfitting. Es gibt einige Techniken zur Regularisierung, aber diese sind in der Praxis oft recht aufwendig. Häufig sind Ergebnisse nicht reproduzierbar, weil kleine Änderungen in den Daten manchmal schon zu gänzlich unterschiedlichen Baumstrukturen führen können.

Als einfaches Beispiel erzeugen und betrachten wir wieder zwei verschiedene Klassen (rot und blau dargestellt in Abbildung 4-24) mit den Features x und y.

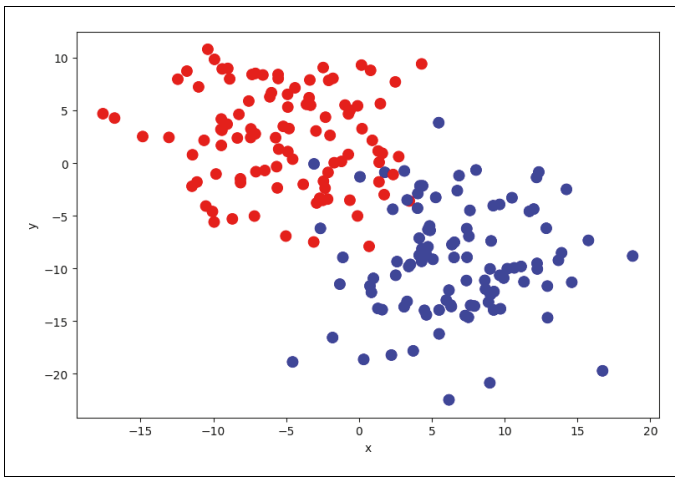

Abbildung 4-24: Decision Tree mit zwei Klassen

Wenn wir den `DecisionTreeClassifier` von Sklearn auf dieses Problem anwenden, können wir die Entscheidung des Klassifikators wieder mithilfe farbiger Entscheidungsflächen visualisieren (siehe Abbildung 4-25):

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(X, y)
print(tree.score(X,y))
> 1.0
```

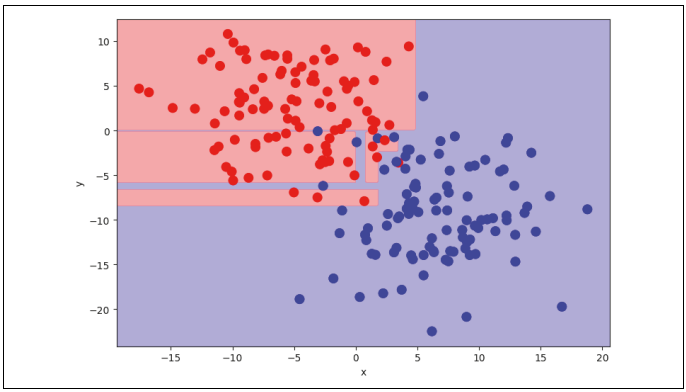

Abbildung 4-25: Entscheidung eines Decision-Tree-Klassifikators

Eine schematische Darstellung des Entscheidungsbaums könnte in etwa aussehen wie in Abbildung 4-26 (in der Abbildung wurden aus Platzgründen die Verzweigungswerte auf ganze Zahlen gerundet).

Auf der ersten Ebene teilt der Algorithmus alle Trainingsbeispiele in zwei Zweige: in die Beispiele mit dem `x`-Wert kleiner als 1,75 (erster Zweig) bzw. größer als 1,75 (zweiter Zweig, siehe Abbildung 4-27). Selbst mit dieser sehr groben Einteilung kommen wir bereits auf einen erstaunlich guten Score. Wir setzen dazu `max_depth` auf 1, um nur die erste Ebene zu nutzen:

```
tree1 = DecisionTreeClassifier(max_depth=1)
tree1.fit(X, y)
print(tree1.score(X,y))
> 0.91
```

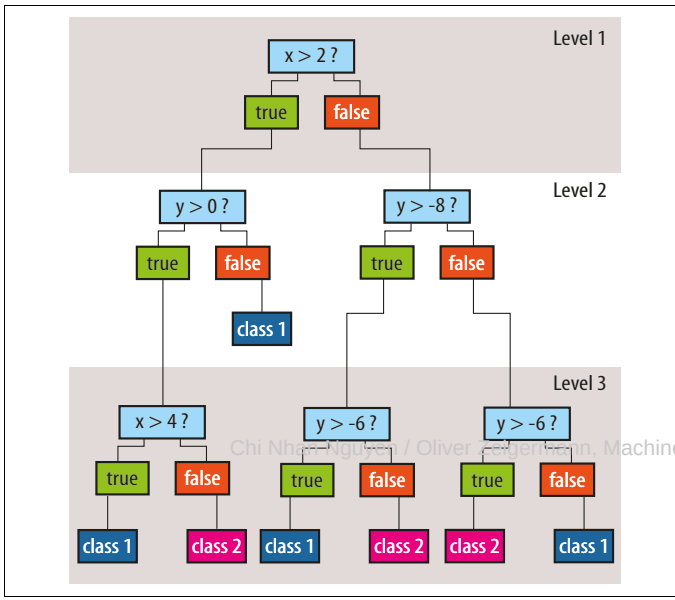

Abbildung 4-26: Schematischer Entscheidungsbaum

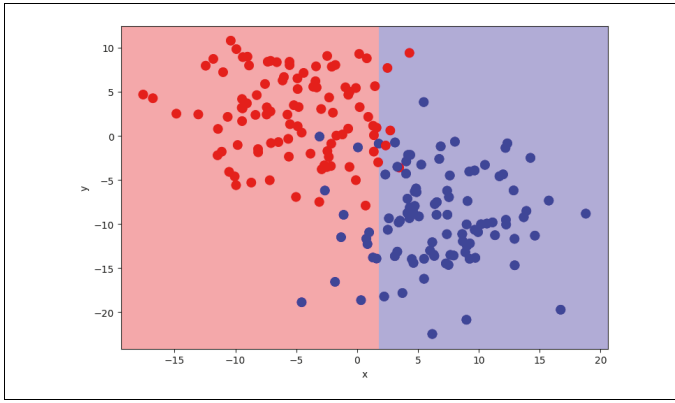

Abbildung 4-27: Level 1 Decision Tree

Auf der zweiten Ebene werden die jeweiligen Zweige wiederum in je zwei weitere Zweige geteilt. Der erste Zweig (x <= 1,75) wird geteilt in die Beispiele mit y < –8,4 und y > –8,4, während der zweite Zweig (x > 1,75) bei etwa y = 0 geteilt wird (siehe Abbildung 4-28):

```
tree2 = DecisionTreeClassifier(max_depth=2)
tree2.fit(X, y)
print(tree2.score(X,y))
> 0.97
```

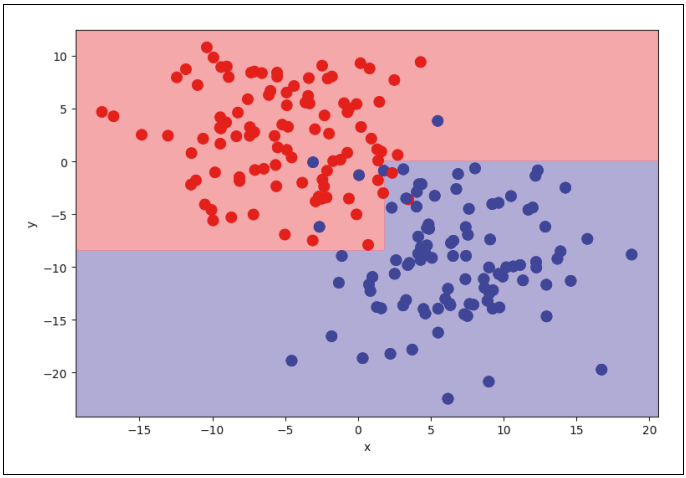

Abbildung 4-28: Level 2 Decision Tree

Auf die gleiche Weise geht es weiter auf der dritten und vierten Ebene etc. (siehe Abbildungen 4-29 bis 4-31), und zwar so lange, bis nur noch Beispiele von lediglich einer Klasse in einem Zweig (Branch) übrig sind. Die Klassifikation für den Zweig ist dann abgeschlossen, und der Zweig wird als Blatt (Leaf) bezeichnet:

```
tree3 = DecisionTreeClassifier(max_depth=3)
tree3.fit(X, y)
```

```
print(tree3.score(X,y))
> 0.975

tree4 = DecisionTreeClassifier(max_depth=4)
tree4.fit(X, y)
print(tree4.score(X,y))
> 0.985

tree5 = DecisionTreeClassifier(max_depth=5)
tree5.fit(X, y)
print(tree5.score(X,y))
> 0.99
```

Auf der vierten und fünften Ebene (siehe Abbildungen 4-29 und 4-30) kannst du aber ein Problem erkennen: Overfitting. Der Algorithmus konzentriert sich zu sehr auf Einzelbeispiele und verliert dadurch an Generalisierungsfähigkeiten. Wir erkennen das intuitiv daran, dass kleine Entscheidungsinseln oder -raster entstehen, die nur durch einzelne wenige Trainingsbeispiele verursacht werden.

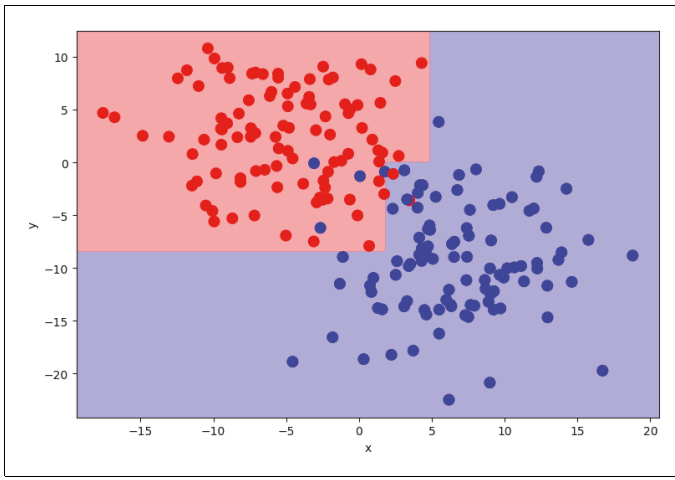

Abbildung 4-29: Level 3 Decision Tree

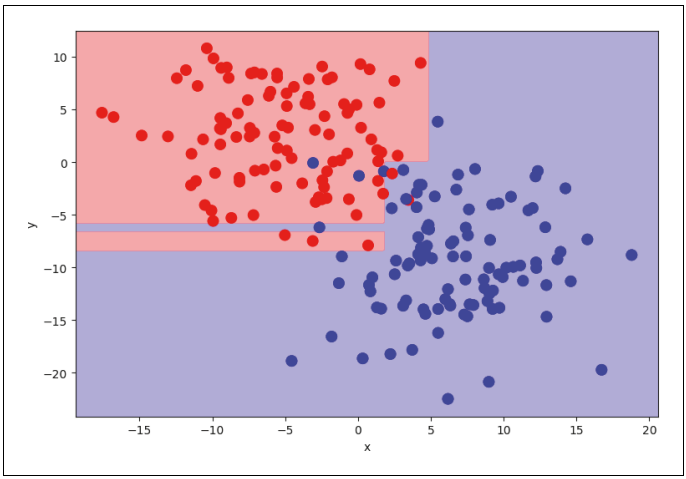

Abbildung 4-30: Level 4 Decision Tree

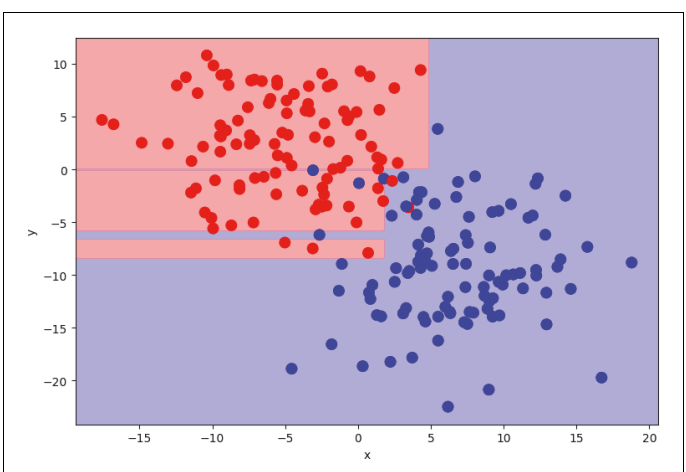

Abbildung 4-31: Level 5 Decision Tree

Wir können Overfitting unterdrücken, wenn wir einige der Voreinstellungen ändern. Eine Möglichkeit ist, die maximale Anzahl der Ebenen anzugeben. Wenn wir in unserem Beispiel diese maximale Anzahl auf 5 festlegen, sehen die Entscheidungsflächen folgendermaßen aus (siehe Abbildung 4-32):

```
tree = DecisionTreeClassifier(max_depth=5)
tree.fit(X, y)
print(tree.score(X,y))
> 0.97
```

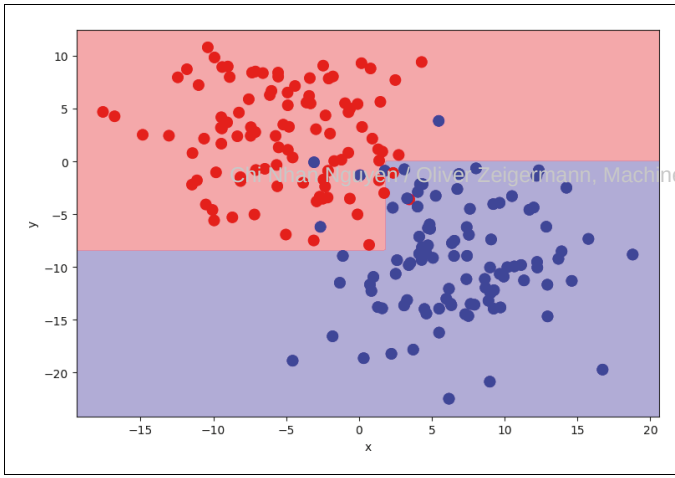

Abbildung 4-32: Regularisierung: maximale Anzahl an Levels

Eine weitere Möglichkeit besteht darin, die Mindestanzahl von 10 Beispielen pro Verzweigung anzugeben (siehe Abbildung 4-33):

```
tree = DecisionTreeClassifier(max_depth=5,
                              min_samples_split=10)
tree.fit(X, y)
print(tree.score(X,y))
> 0.98
```

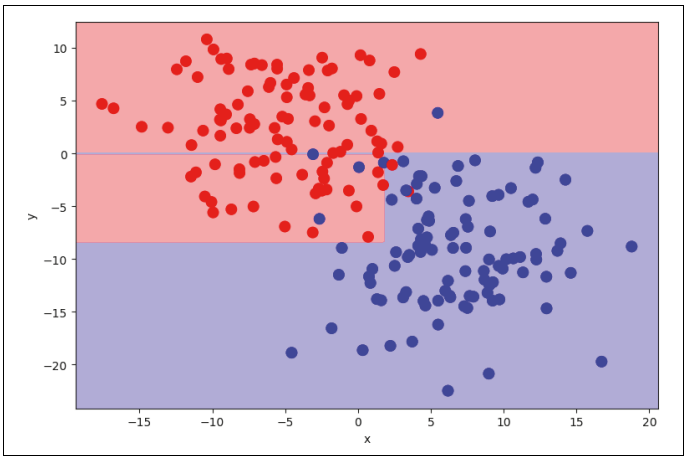

Abbildung 4-33: Regularisierung: Mindestanzahl von Beispielen pro Branch-Splitting

Die Festlegung der maximalen Anzahl von Blättern auf 8 führt zu folgendem Ergebnis (siehe Abbildung 4-34):

```
tree = DecisionTreeClassifier(max_depth=5,
                              min_samples_leaf=1,
                              max_leaf_nodes=8)
tree.fit(X, y)
print(tree.score(X,y))
> 0.98
```

Zusammenfassend können wir nun den Decision-Tree-Algorithmus mit allen besprochenen Regularisierungen auf den Irisdatensatz anwenden und erhalten folgende Ergebnisse:

```
X = iris.data
y = iris.target

from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=5,
                              min_samples_leaf=1,
                              min_samples_split=2,
                              max_leaf_nodes=8)
```

```
tree.fit(X, y)
print(tree.score(X,y))
> 0.99333333333
```

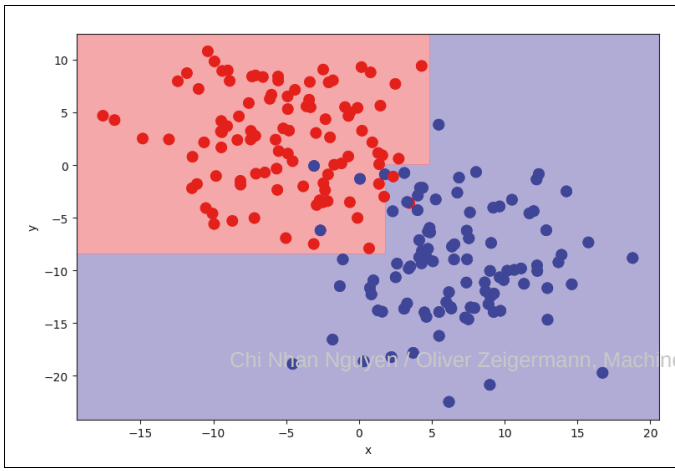

Abbildung 4-34: Regularisierung: maximale Anzahl von Leaf-Knoten

Das Resultat ist eine dreidimensionale Hyperebene im vierdimensionalen Feature-Raum. Diese können wir natürlich wieder nicht darstellen, deshalb zeigen wir dir die paarweisen zweidimensionalen Entscheidungsflächen in Abbildung 4-35. In diesem Fall gibt es sechs solcher Flächen. Sie stellen die Projektionen der dreidimensionalen Hyperfläche auf die jeweilige zweidimensionale Ebene dar. So bekommst du zumindest einen gewissen Eindruck der resultierenden Hyperebene.

Wie schon in der Einleitung angedeutet, können Decision Trees sehr effektiv sein, aber sie tendieren leicht zu Overfitting. Eine Regularisierung mithilfe der vielen Parameter ist möglich, in der Praxis aber oft zu aufwendig. Weiter unten werden wir dir im Rahmen der Ensemble-Methoden auch die Random-Forest-Methode vorstellen, die einige dieser Probleme löst.

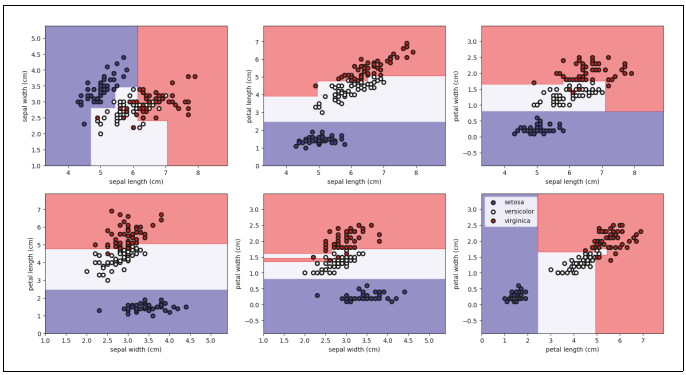

Abbildung 4-35: Paarweise Entscheidungsflächen des Decision-Tree-Klassifikators für die Irisdaten

Split-Kriterium: Gini Impurity vs. Entropy

Vorher aber sehen wir uns für ein besseres Verständnis in einem kleinen Exkurs an, wie auf jeder Ebene die Entscheidung getroffen wird, auf welche Weise die Zweige aufgespaltet werden. Als Voreinstellung wird das Gini-Kriterium angewendet. Dieses Kriterium ist ein Maß für die Klasseninhomogenität (engl. Impurity) der Beispiele, die in einen Zweig fallen:

$$Gini = 1 - \sum p(i)^2$$

Dabei bezeichnen *i* die Klasse und *p* die Wahrscheinlichkeit einer Klasse (approximiert durch den relativen Anteil der Beispiele in Klasse *i*). Je kleiner der Gini-Wert, desto größer ist die Klassenhomogenität des Splits. Teilt der Split die Beispiele für einen Zweig nur in eine Klasse *i* auf, wird die Wahrscheinlichkeit für Klasse *i* 1 und 0 für alle anderen. Die *Gini Impurity* ist dann null. Sind aber gleich viele Beispiele für alle Klassen *i* in dem Zweig, werden alle Wahrscheinlichkeiten klein und die Gini Impurity maximal.

Alternativ kannst du *Entropy* als Kriterium anwenden. Die Entropy ist definiert als

$$Entropy = -\sum p(i) \cdot logp(i)$$

Sie ist ebenfalls ein Maß für die Homogenität der Klassen. Sie hat ähnliche Eigenschaften wie die Gini Impurity, und man erhält in der Praxis sehr ähnliche Ergebnisse mit beiden. Das Gini-Kriterium ist bei der Berechnung etwas schneller, da kein Logarithmus angewendet wird, und wird deshalb etwas häufiger benutzt. Details dazu findest du wieder im Notebook zu diesem Kapitel.

Random-Forest-Klassifikator

Ein großer Nachteil von Decision-Tree-Klassifikatoren ist das Overfitting. Wie wir im vorigen Abschnitt gesehen haben, kann man Overfitting unterdrücken, indem man verschiedene Regularisierungsparameter manuell einstellt. Die optimalen Parameter hängen aber nicht nur von den Features ab, sondern auch von der Anzahl der Trainingsbeispiele. Die Suche nach den optimalen Parametern oder eine Automatisierung ist in der Praxis häufig sehr umständlich und aufwendig.

Die Bagging-Methode des Random Forest Classifier ist ein Beispiel für eine Ensemble-Methode, die Overfitting reduzieren kann. Beim Random-Forest-Klassifikator wird der Datensatz zunächst in kleinere zufällige Subsamples aufgeteilt (wobei Duplikate ausdrücklich erlaubt sind). Für jedes dieser Subsamples wird wiederum ein Decision Tree Classifier mit einem zufälligen Subsample der Features generiert. Die jeweiligen Ergebnisse werden am Ende mit gleicher Gewichtung zusammengefasst. Es wird bei Regression also der Durchschnitt aller Einzelentscheidungen für die Gesamtentscheidung genommen bzw. bei der Klassifikation die Mehrheit. Diese Methode reduziert in der Regel die Varianz des Modells zu dem Preis einer geringen Erhöhung des Bias.