Modeling

Supervised Learning: Decision Trees und Random Forest

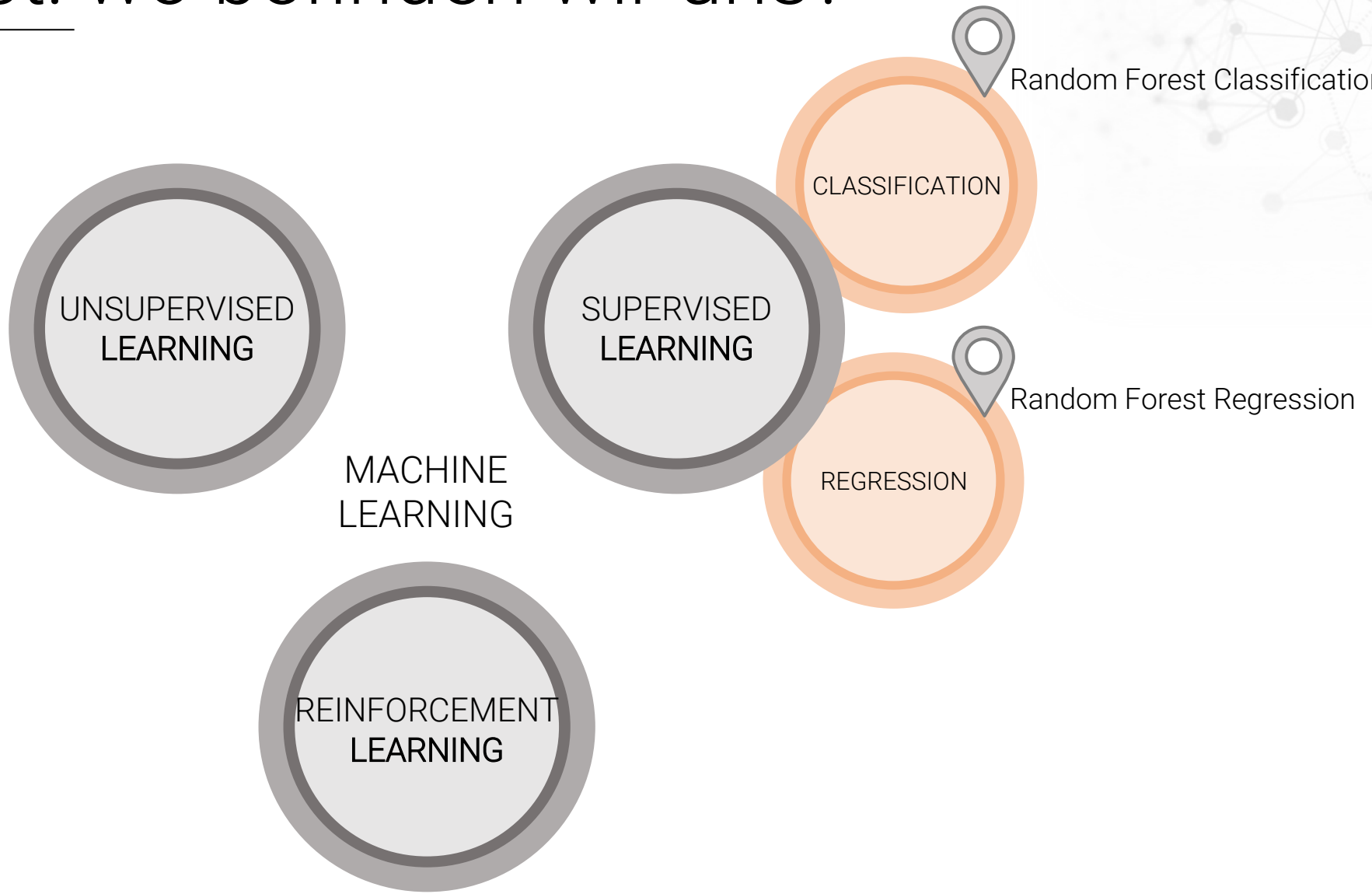
Agenda

- Entscheidungsbäume
- Gini Impurity
- Random Forest
- Bagging und Feature Bagging
- Out-of-Bag Error
- Feature Importance
- Boosting
- Random Forest Regression
- Large Guided Project:
  Random Forest Regression for Remaining Useful Life Prediction

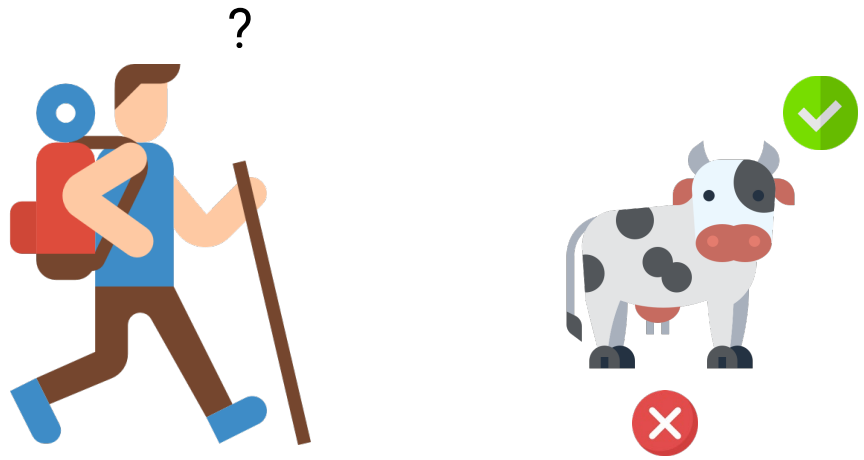
So what?

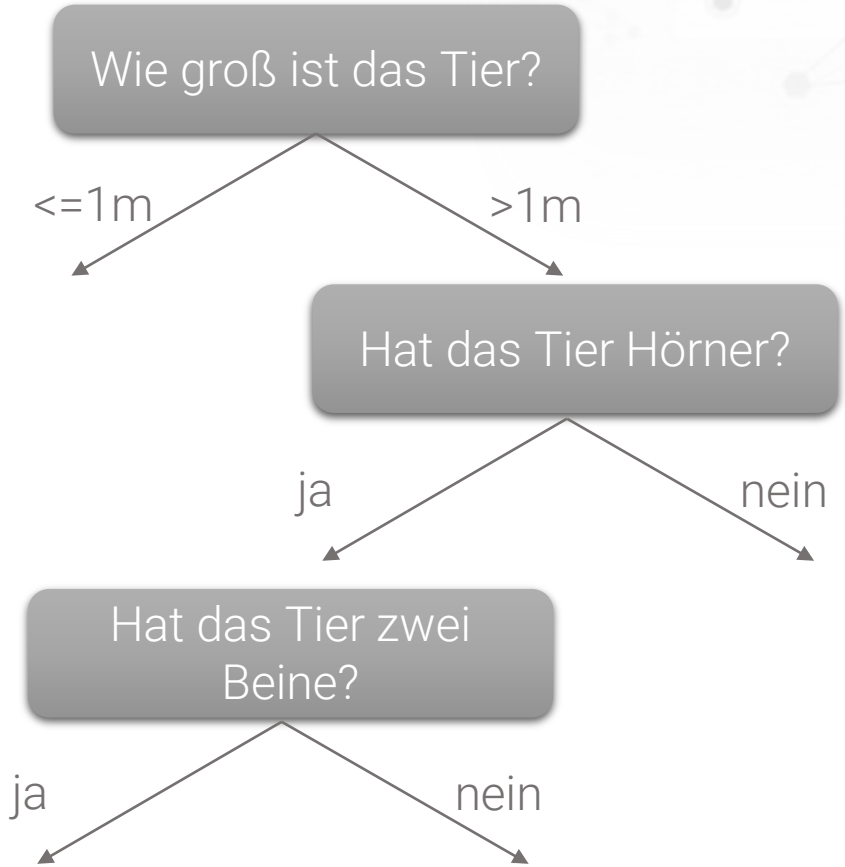
Da es sich bei Random Forest Ansätzen um sog. *Ensemble Methoden* handelt, lernen wir in diesem Abschnitt diese generischen ML-Konzepte im spezifischen Fall der Random Forests kennen.

Random Forest: wo befinden wir uns?



Stellen Sie sich vor Sie gingen wandern…

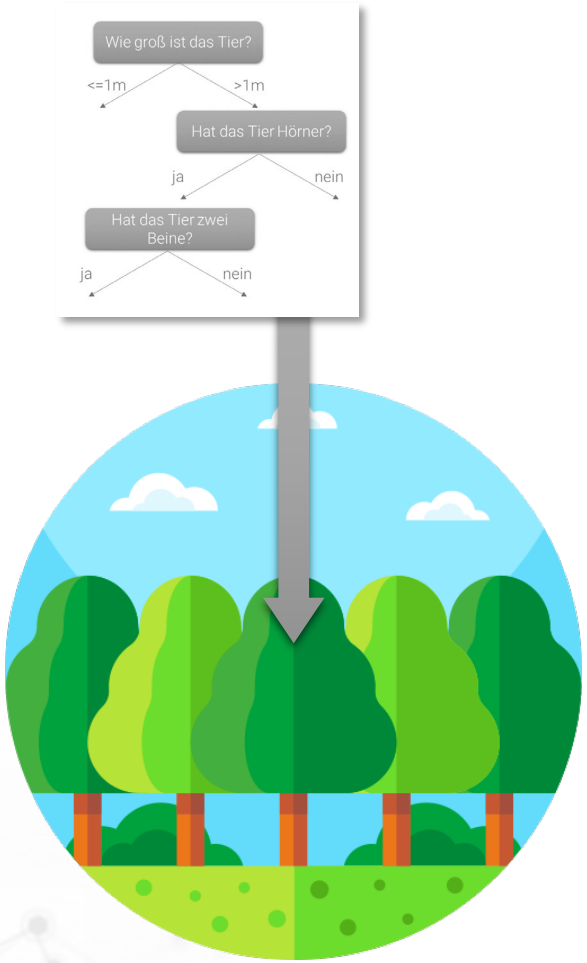




So what?

Wenn Sie sich zur *Kategorisierung* solche Fragen stellen und anhand dieser Fragen eine *Klassifikation* durchführen, dann erzeugen Sie implizit einen sog. *Entscheidungsbaum*

Entscheidungsbäume

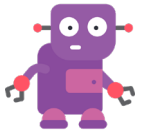


- Entscheidungsbäume sind die Grundlage eines *Random Forest*
- Random Forests sind sog. *Ensemble Learner:* sie setzen sich aus mehreren (schwachen) Learnern bzw. Klassifikatoren (den einzelnen Entscheidungsbäumen) zusammen
- Hier gilt oft: das Ganze ist mehr als die Summe seiner Teile! D.h. die Klassifikation des gesamten Ensembles ist oft besser als die Klassifikation eines einzelnen Klassifikators des Ensembles
- Entscheidungsbäume sind eine sehr intuitive Methode, um Klassifikationen durchzuführen
  → Kern: geschickte Fragen stellen

Was denken Sie?

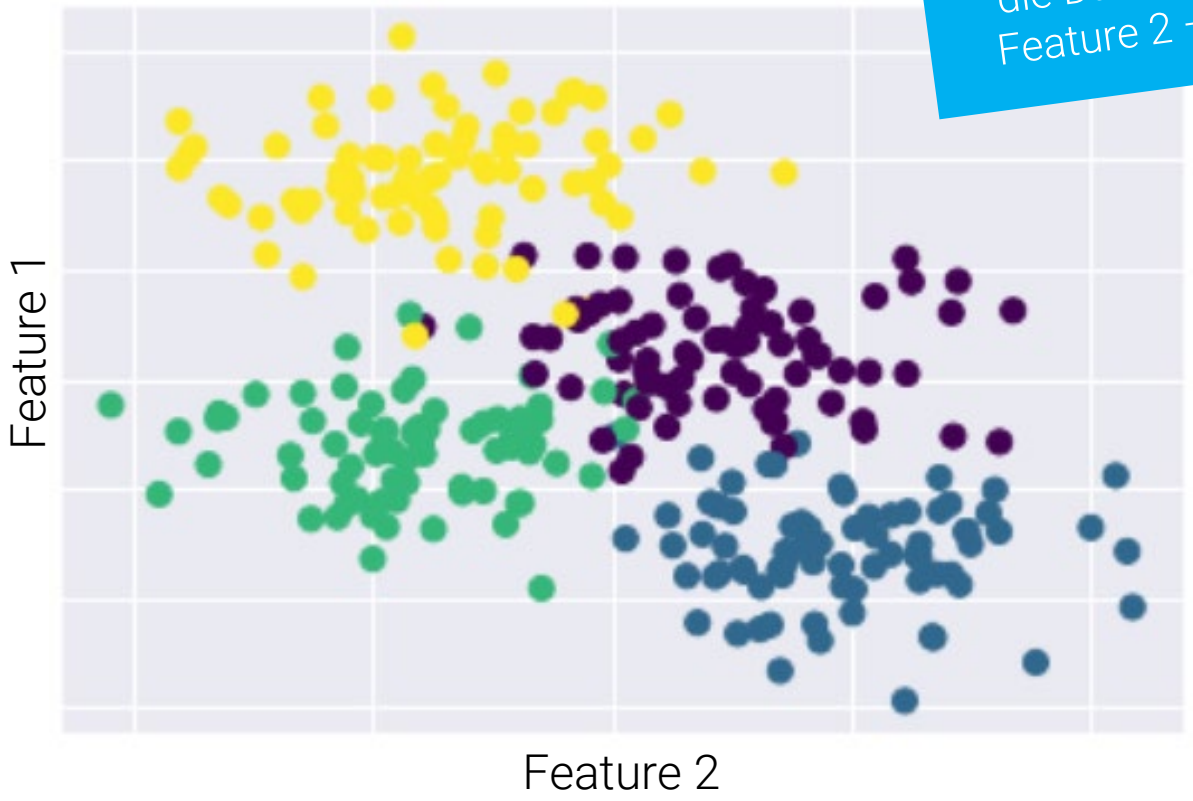
Was entspricht „Fragen stellen“ im Daten- bzw. Feature-Raum?

Wir als Klassifikatoren

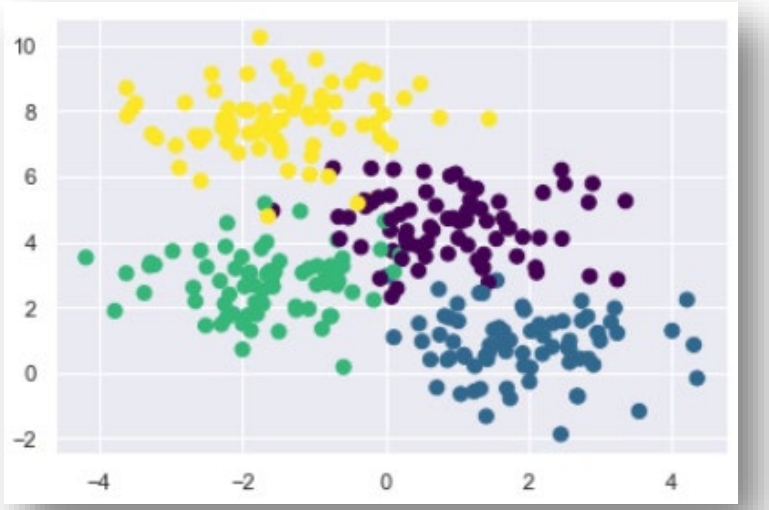


Was denken Sie?

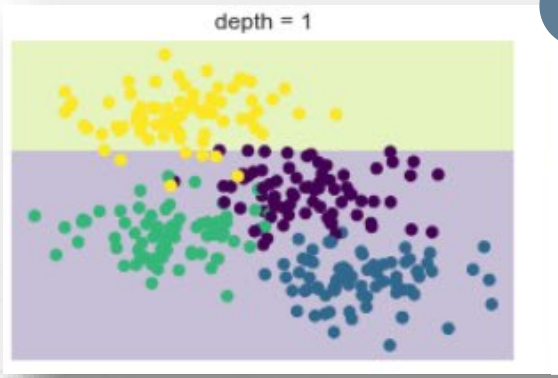
Versuchen Sie zuerst die richtige „Frage“ an die Daten anhand Feature 1 – und dann Feature 2 – zu stellen. (Zoom: zeichnen)



Entscheidungsbäume



- Im Daten- bzw. Feature-Raum bedeutet „geschickte Fragen stellen“ die Daten anhand einer geschickt gewählten Linie, parallel zu einer Achse, aufzuteilen
- D.h. jeder *Knoten* eines Entscheidungsbaumes teilt die Daten in zwei Gruppen auf, mittels eines Cut-off Wertes entlang einer Feature-Achse
- Lassen Sie uns das direkt an Daten ausprobieren!



Was denken Sie?

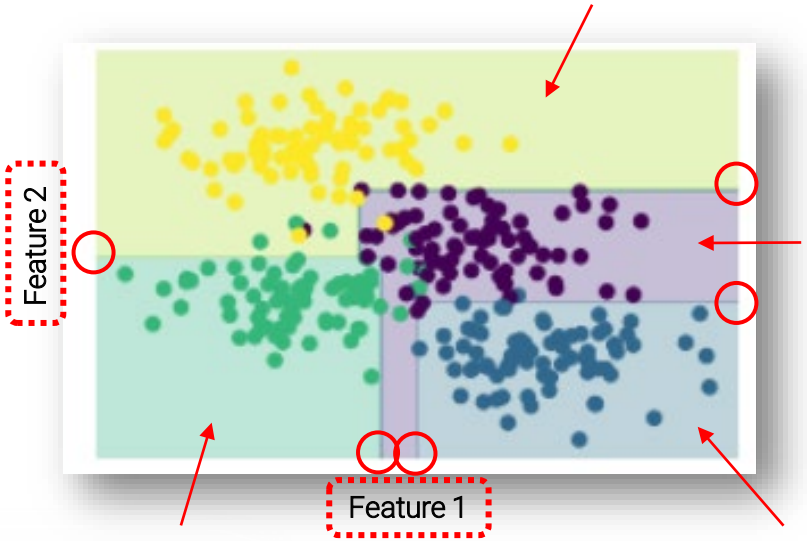
Würden Sie den gelben Bereich nochmal splitten?

Erste „Frage“ an die Daten… …zweite… …dritte… …vierte…

Entscheidungsbäume

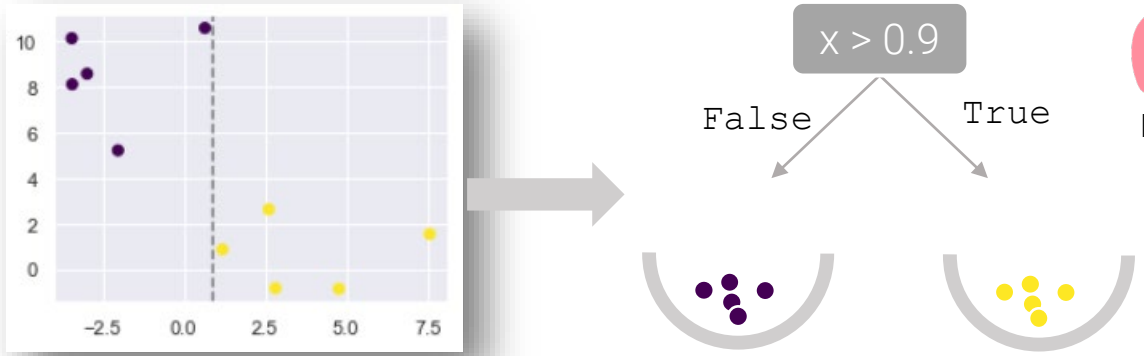
Was denken Sie?

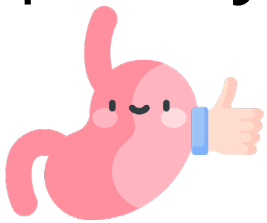
Welche Fragen tauchen jetzt für uns auf?



- Wie werden die zu splittenden Variablen bzw. Features ausgewählt?
  → Quantitatives Qualitätskriterium!
- Wie werden die Cut-off Werte auf den jeweiligen Feature-Achsen gewählt?
  → Quantitatives Qualitätskriterium!
- Wie kommt die Färbung der Bereiche zustande?
  → Majoritätsvotum

Qualitätskriterien: Gini Impurity

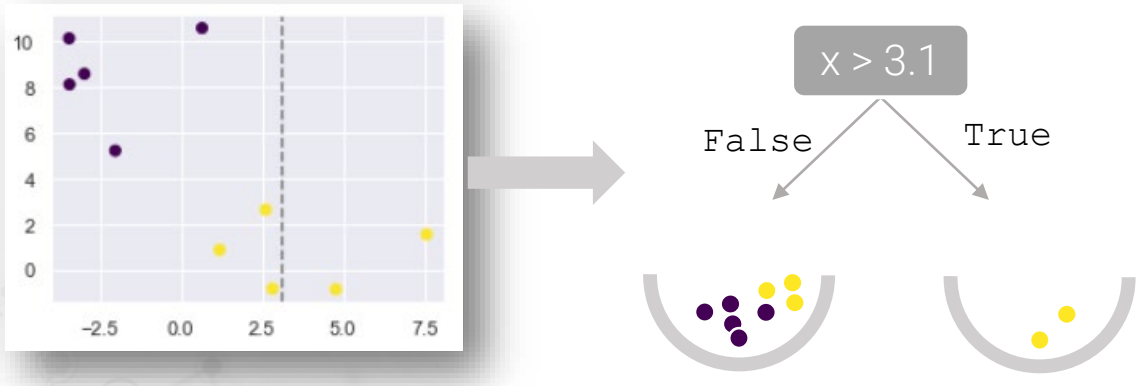




Bauchgefühl – check!

Was denken Sie?

Was ist „more impure“?

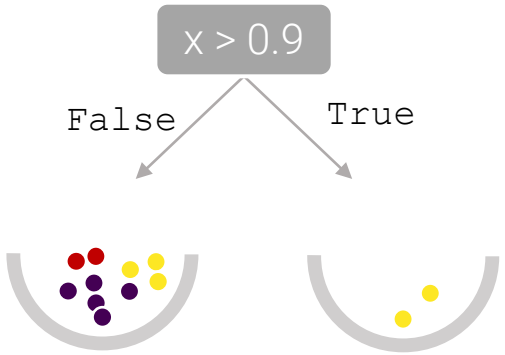


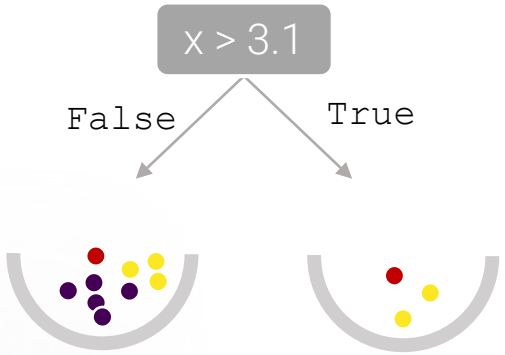
- Um bewerten zu können, ob ein Split „gut“ oder „schlecht“ ist, brauchen wir ein *quantitatives Maß*
  → Gütekriterium
- Ein solches Maß ist die sog. *Gini Impurity*

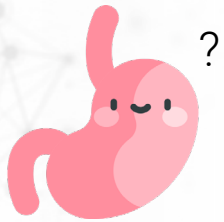
So what?

Ein allgemeiner Faktor: Qualitäts- bzw. Gütekriterien sind zentraler Bestandteil des Machine Learnings

Qualitätskriterien: Gini Impurity







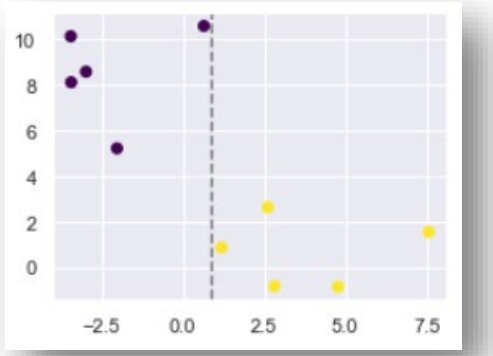
Bauchgefühl – hm?!

- Sobald die Zusammensetzung unserer Datenpunkte innerhalb der *Branches* etwas komplizierter wird, verlässt uns unsere Intuition
- Wir brauchen also ein objektives Maß, um die *Impurity* unserer Aufteilung zu beurteilen

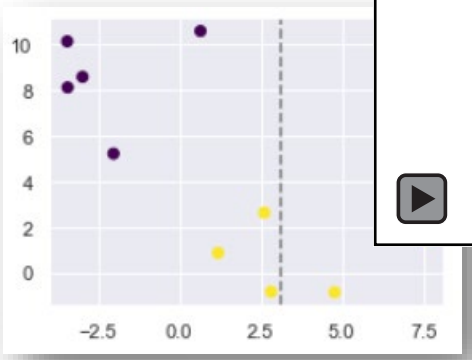
Definition

Die *Gini Impurity* ist die Wahrscheinlichkeit einen zufällig gezogenen Datenpunkt falsch zu klassifizieren, falls man die Klassifizierung anhand der Verteilung der Klassen im Datensatz durchführt

Qualitätskriterien: Gini Impurity



$$G_{left|right} = 0$$



$$G_{left} = 0.46875 \qquad G_{right} = 0$$

→ Gewichten nach Anzahl/Anteil an Datenpunkten, umrühren.. äh … aufsummieren, fertig!
→ Gütemaß zur Bewertung eines Splits

- Zurück zu unserem Ausgangsbeispiel: 2 Klassen mit je 5 Datenpunkten
- Bezogen auf den gesamten Datensatz: falls wir einen zufälligen Datenpunkt ziehen würden erhielten wir zu 50% einen violetten oder gelben Datenpunkt
- Würden wir entsprechend der Verteilung der Klassen in den Daten klassifizieren, würden wir ihm zu 50% die Klasse violett bzw. gelb zuweisen

berechnen wir also eine *Falsche Zuordnung?*
Ziehe violett, klassifiziere violett (25%)
Ziehe violett, klassifiziere gelb (25%) ✗
Ziehe gelb, klassifiziere violett (25%) ✗
Ziehe gelb, klassifiziere gelb (25%)
→ 25% + 25% = 50% → Gini Impurity: 0.5

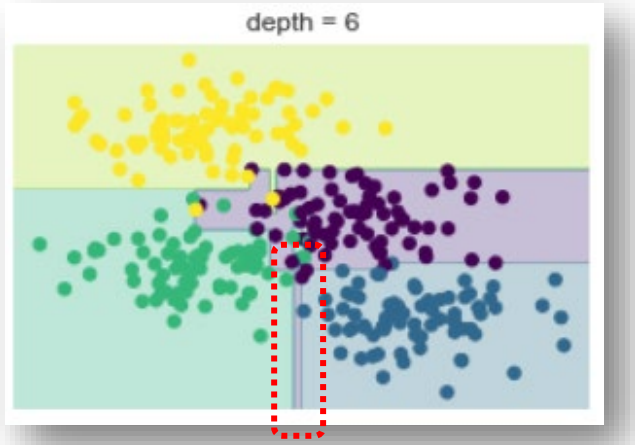
So what?
Grundgedanke: falsche Zuordnung ist wahrscheinlicher, je „durchmischter" die Daten sind

Definition

Wenn uns C Klassen vorliegen und $p(i)$ die Wahrscheinlichkeit ist einen Datenpunkt der Klasse $i \in C$ zu ziehen, dann ist die Gini Impurity definiert als

$$G = \sum_{i=1}^{C} p(i)(1 - p(i))$$

Entscheidungsbäume:



Was denken Sie?

Was ist denn hier passiert?

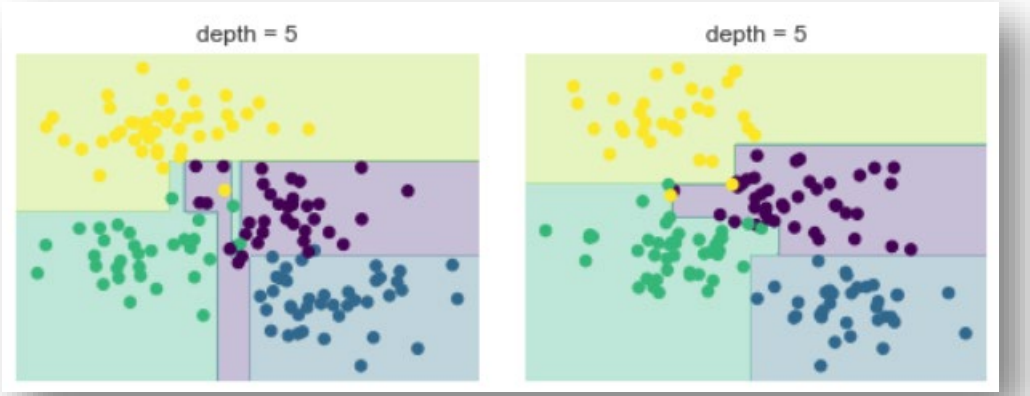
Was denken Sie?

Was also tun?

So what?

Dort, wo die Klassifizierung sicherer ist, erzeugen verschiedene Decision Trees konsistente Resultate – bei unsicheren „Bereichen“ eher inkonsistente.

- Dieser schmale Bereich scheint nicht einer *intrinsischen* Eigenschaft unserer Daten zu entsprechen
  → Der Decision Tree kommt ins Overfitting – und das nach schon 6 Splits
- Diese Tendenz des Overfittings ist eine generelle Eigenschaft von Decision Trees
- Dieses Overfitting kann man auch verdeutlichen, wenn man Decision Trees an verschiedenen Teilmengen unseres Datensatzes trainiert



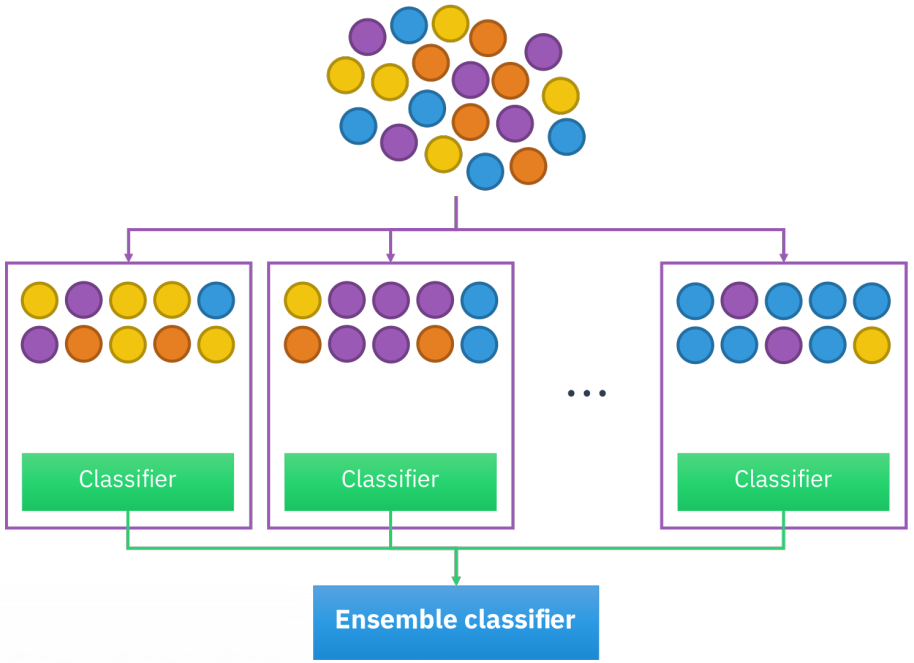
Ensembles: Random Forest

- Um aus diesen sog. *Weak Learners* einen geeigneten, sog. *Strong Learner*, zu konstruieren, kombiniert man die Resultate *mehrerer* Decision Trees zu einem *Random Forest*
- Durch Randomisierung erzeugt man diese Weak Learners – und deren Kombination führt zum Ensemble, das *Random Forest* genannt wird

So what?

Ensembles an Modellen ist etwas Generisches im Bereich des Machine Learnings. Sie können verschiedene Modelle kombinieren, um eine Klassifikation bzw. Vorhersage zu erhalten
→ Ensemble Ansatz

Bagging



Original Data

Bootstrapping

Aggregating

Bagging

- Um ein solches Ensemble aus Weak Learnern zu konstruieren, bedient man sich der Methode des *Bagging (Bootstrap Aggregating)*
- Hierbei *zieht* man aus dem gesamten Datensatz *mit Zurücklegen (Bootstrapping)* Stichproben *(Bootstrap Samples)*
  → Samples können mehrfach in den Stichproben vorkommen!
- Auf jede Stichprobe wird ein Modell trainiert und deren Output zu einem Ensemble Classifier bzw. Strong Learner *aggregiert*
  → Z.B. Mittelwert, Majoritätsvotum (Modus)

Was denken Sie?

Warum zieht man mit Zurücklegen?

Was denken Sie?

Ist Cross Validation Bootstrapping?

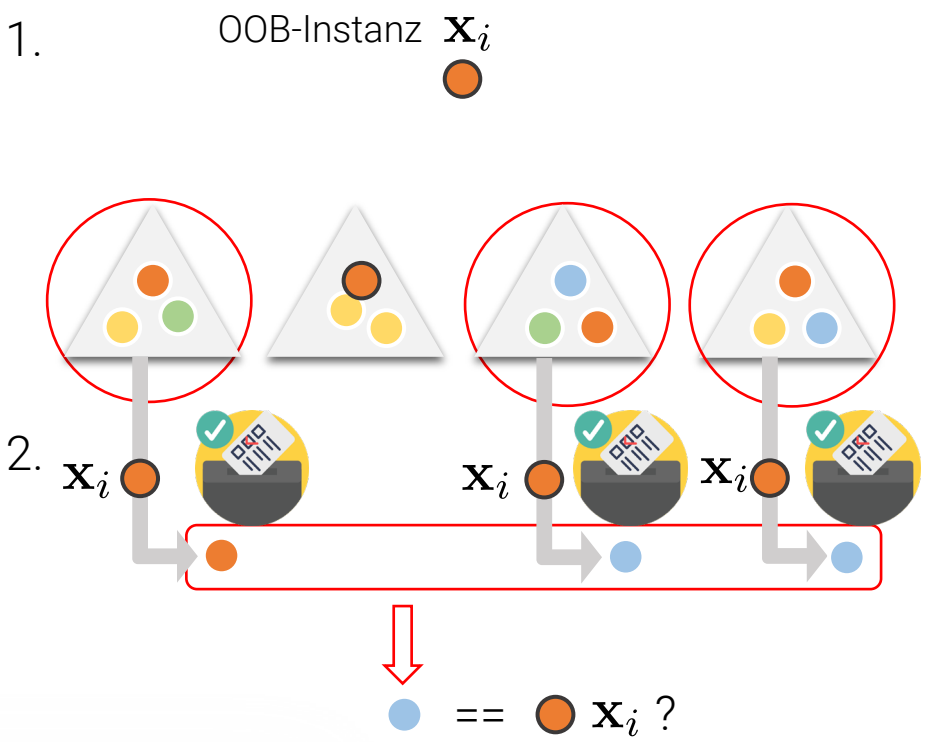
Was denken Sie?

Wie würden Sie aggregieren?

So what?

Warum tut man das alles? Stabilität und Genauigkeit werden verbessert – Overfitting wird reduziert

Out of Bag Error



- Da beim Bagging eine Stichprobe – also nur ein Teil – der originalen Daten gezogen wird, verbleibt ein Teil außerhalb des „Bag“
- Diesen Teil kann man dann nutzen, um die Performance des Modells zu bewerten
- Man berechnet den sog. *Out of Bag (OOB) Error*:
  1. Finde alle Decision Trees, die die OOB-Instanz $\mathbf{x}_i$ nicht in ihrem Trainingsdatensatz hatten
  2. Bestimme das Majoritätsvotum dieser Trees und vergleiche mit dem wahren Label von $\mathbf{x}_i$
  3. Führe 1. und 2. für alle möglichen OOB-Instanzen durch und berechne die Genauigkeit bzw. den Fehler

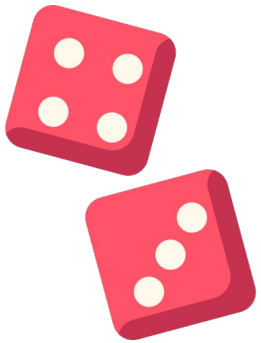
So what?

Cross Validation und OOB-Error sind unterschiedliche Methoden, um die Performance eines Modells zu berechnen. Beide Methoden konvergieren zu ähnlichem Ergebnis. OOB hat den Vorteil, dass es komputativ anspruchsloser ist und man quasi während des Trainings testen kann.

Randomisierte Auswahl an Features

Was denken Sie?

Warum tut man das?



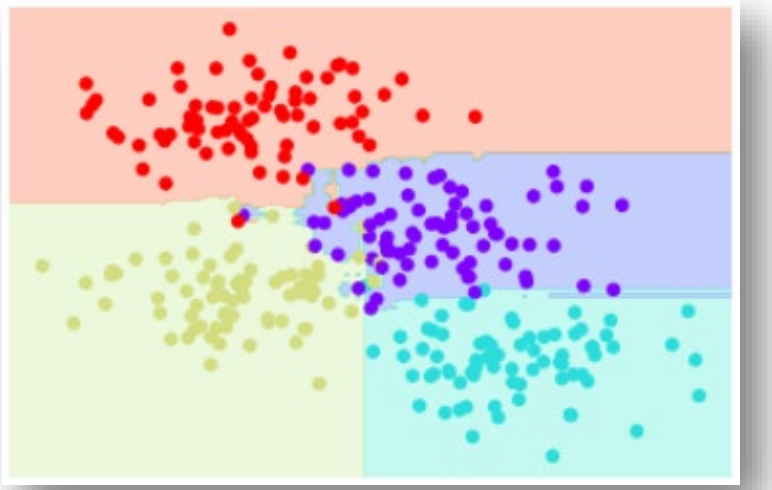
- Zusätzlich zum Bagging – also der Erzeugung von Subsets unseres Datensatzes – wird bei Random Forests noch das sog. *Feature Bagging* durchgeführt
- Diese Modifizierung des Trainings eines Decision Trees besteht aus der *zufälligen Wahl eines Subsets der Features*, an denen der Split durchgeführt werden kann
- Dies wird wiederum durchgeführt, um die Decision Trees zu dekorrelieren
  → Erhöhung der Varianz – und somit Vermeidung von Overfitting

So what?

Auch hier handelt es sich wiederum um einen generischen Ansatz im Machine Learning: dieser wird *Random Subspace Method* genannt

Random Forest in `sklearn`

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=0)
```



So what?
Die Kombination aus den randomisierten Decision Trees führt zu einem Klassifikator, der die zugrundeliegende Struktur der Daten besser modelliert als einzelne Trees.

- In `sklearn` gibt es für einen Random Forest eine Klasse

  `RandomForestClassifier()`

- Interessante Eingabeargumente:
  - `n_estimators`: Anzahl an Decision Trees im Wald
  - `criterion`: Gütemaß zur Auswahl eines Splits. Default ist `gini`, weitere Möglichkeit `entropy`
  - `max_depth`: Maximale Tiefe der Trees
  - `bootstrap`: `True` oder `False` (== nutze gesamten Datensatz)

Was denken Sie?

Bedeuten mehr Decision Trees pauschal höhere Genauigkeiten?

Was denken Sie?

Warum ist es bei Random Forests einfach(er) die Ungewissheit der Vorhersage zu berechnen?

Beispiel generisch: Eignungstestklassifikation - White Box

Wir sind wieder bei unserem Beispiel "Eignungstest - Mini-SVM". Darin war es unsere Aufgabe Ergebnisse von Eignungstests anhand einer SVM klassifizieren zu lassen. Nun würden wir gerne genau sehen können, was unser Machine Learning Modell gelernt hat und **wie** es Entscheidungen trifft - **White-Box-Modelle**. Hierzu trainieren wir einen kleinen Random Forest `n_estimators=5` auf unsere Daten `X_y` und lassen uns mittels der `plot_tree` -Funktion aus `sklearn` den zweiten Decision Tree des Random Forest visualisieren - und das alles als *One-Liner*.

Random Forest in `sklearn`

```python
# Import
from sklearn.model_selection import train_test_split

# Split and fit
X_train, X_test, y_train, y_test = train_test_split(X, y)
model.fit(X_train, y_train)

# Calculate class probability
model.predict_proba(X_test)
```

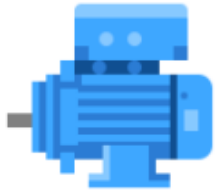
```
array([[0.02, 0.96, 0.02, 0.  ],
       [0.04, 0.9 , 0.06, 0.  ],
       [0.  , 0.03, 0.97, 0.  ],
       [1.  , 0.  , 0.  , 0.  ],
       [0.96, 0.04, 0.  , 0.  ],
       [0.  , 0.  , 0.03, 0.97],
       [0.  , 0.  , 0.01, 0.99],
       [0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.98, 0.02],
       [0.83, 0.  , 0.05, 0.12],
       [0.  , 0.  , 1.  , 0.  ],
       [0.14, 0.  , 0.86, 0.  ],
       [0.  , 0.  , 0.01, 0.99],
       [0.  , 0.  , 1.  , 0.  ],
```

Was denken Sie?

Mit dem Wissen, dass ein Forest aus vielen Trees besteht: können Sie erahnen, was die .predict_proba()-Methode berechnet?

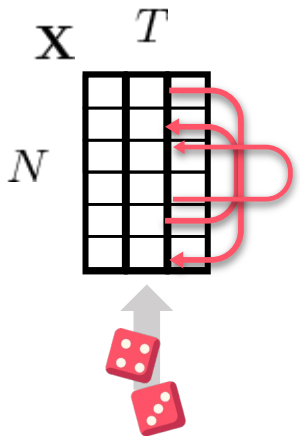
- Der `RandomForestClassifier` hat – neben den gewohnten Methoden `.fit()` etc. – eine weitere interessante Methode
  → Die `.predict_proba()`-Methode
- Mit dieser Methode kann man sich die Wahrscheinlichkeit eines Datenpunktes zur Zugehörigkeit einer Klasse ausgeben lassen
- Das ist nur möglich, da unser Modell aus vielen „kleinen“ Modellen besteht – und jedes dieser Modelle eine Vorhersage machen kann

Beispiel: Servo Engine Fault Classification



Stellen Sie sich vor Sie würden in einem Maschinenbauunternehmen arbeiten und Ihnen liegen Sensordaten `(temperature, current, vibration)` von 200 Servomotoren einer hochautomatisierten Produktionsanlage vor. Es handelt sich nicht um Zeitseriendaten, sondern um Mittelwerte über jeweils eine Woche. Von jedem Servomotor wissen Sie, ob er sich in einem guten `healthy` , mittleren `do_inspection` oder schlechten `bad` Zustand befunden hat. Aus Erfahrung wissen Sie, dass diese Zustände oft schwierig voneinander zu unterscheidn sind. Dies würden Sie nun gerne quantifizieren und einen Random Forest hierzu auf den Datensatz trainieren. Sie wollen - nach dem Training - die `.predict_proba()` -Methode nutzen, um quantitativ aussagen zu können, wie gut sich diese Klassen wirklich voneinander trennen lassen. Ist sich der Klassifikator für alle drei Zustände gleich sicher bzw. unsicher? Visualisieren Sie diese Unsicherheit mittels eines Boxplots. Die identifizierten Punkte können nun von Fachexperten neu beurteilt werden.

Feature Importance: Permutation



- Eine wichtige Frage, die man sich im L… Projektes fragt: **welche Features haben … Vorhersage?**
- Das Konzept, das diese Frage beantwor… *…eature Importance*
- Wir sehen uns einen Spezialfall an: die *Permutation Importance*
- Permutation Importance wird berechnet **nachdem** ein Modell trainiert wurde
- **Zentrale Frage**: Falls die Werte **eines** Features in den **Testdaten** randomisiert werden, wie würde dies die Vorhersagegenauigkeit beeinflussen?
- Die Modellgenauigkeit nimmt natürlich ab – jedoch **stärker** für **wichtigere** Features
- **Ablauf**:
  1. Trainiere Modell
  2. Randomisiere eine Spalte der Feature-Matrix des Testdatensatzes und erzeuge Predictions → Performance-Rückgang ist Maß für Importance
  3. Führe 2. für alle Spalten mehrfach durch

Was denken Sie?

Da Sie die Zeilen einer Spalte randomisieren: auf was müssen Sie achten?

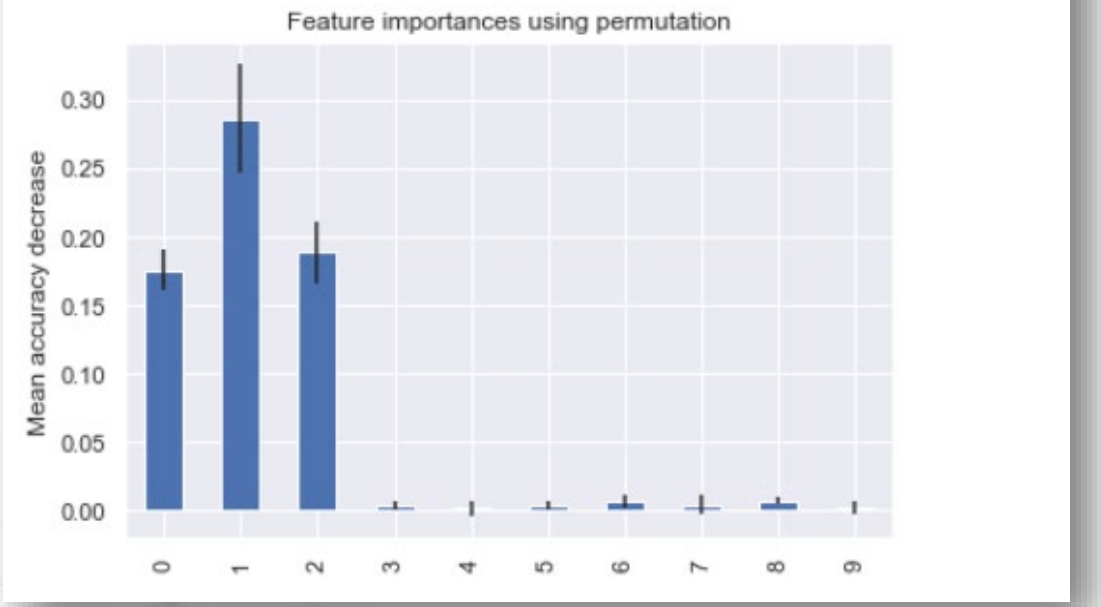
So what?

Feature Importance ist nicht spezifisch für Random Forest, sondern ein generisches Konzept im Machine Learning

https://www.kaggle.com/dansbecker/permutation-importance

Feature Importance: Permutation

```python
fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=result.importances_std, ax=ax)
ax.set_title("Feature importances using permutation")
ax.set_ylabel("Mean accuracy decrease")
fig.tight_layout()
plt.show()
```




```python
from sklearn.inspection import permutation_importance

result = permutation_importance(
    model, X_test, y_test, n_repeats=10, random_state=42, n_jobs=2)
forest_importances = pd.Series(result.importances_mean)
```

- In `sklearn` gibt es die Funktion `permutation_importance`, um die Feature-Importance anhand Permutation zu berechnen

  `permutation_importance(<model>, X_test, y_test, n_repeats)`

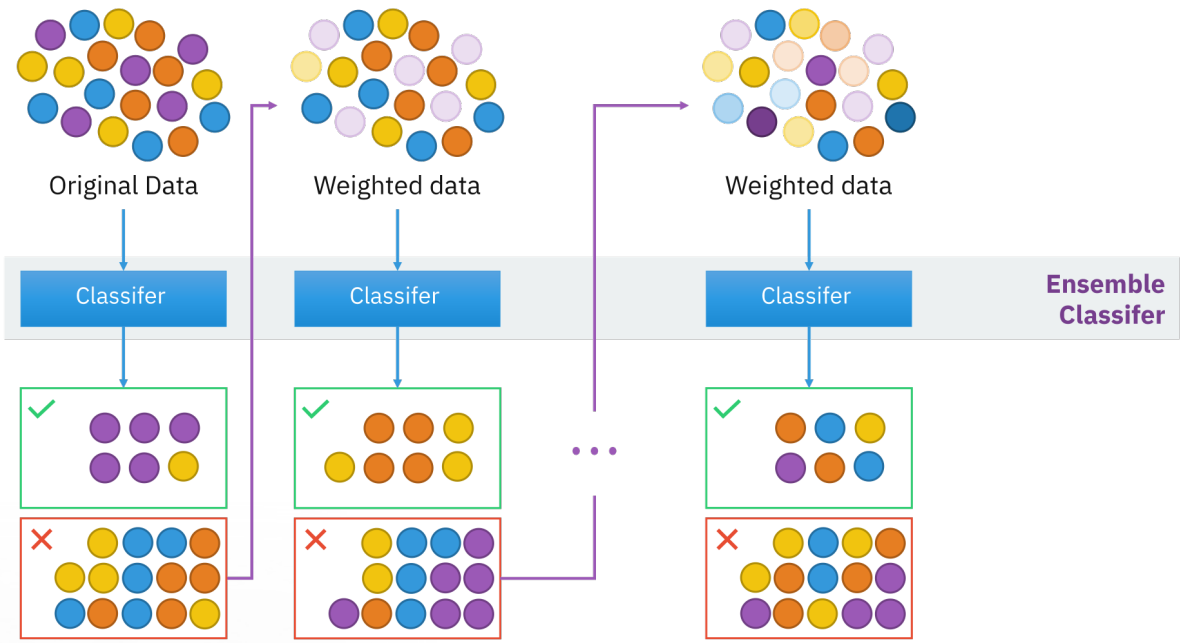
- Dadurch hat man auch mittels z.B. Bar-Plots die Möglichkeit sich die Feature-Importance zu visualisieren

Beispiel: Servo Engine Fault Classification - Sensors to roll-out, Sensors to discard

Wir bleiben bei unserem vorherigen Servomotorbeispiel. Nun liegen uns neue Messungen derselben Motoren vor - jedoch mit weiteren Sensoren `(temperature_1, vibration_1, temperature_2, vibration_2, temperature_3, vibration_3, current, rpm)`. Temperatur- und Vibrationssensoren sind an jeweils drei Positionen der Motoren angebracht. Bei den 300 Motoren handelt es sich um eine Stichprobe, anhand derer entschieden werden soll, ob bzw. welche Sensoren in Zukunft an allen Motoren angebracht werden sollen (Sensoren können - vor allem bei großen Stückzahlen - ordentlich "ins Geld gehen"). Als Angestellte im Maschinenbauunternehmen haben Sie nun die Aufgabe mittels dieser Messungen festzustellen,

1. welche **Sensorenart** und
2. welche **Sensorenposition** für die Schadensklassifikation am *wichtigsten* sind. Darauf aufbauend sollen Sie eine Empfehlung an die Geschäftsleitung geben, welche Sensoren in Zukunft verbaut werden sollen und welche nicht.

Boosting

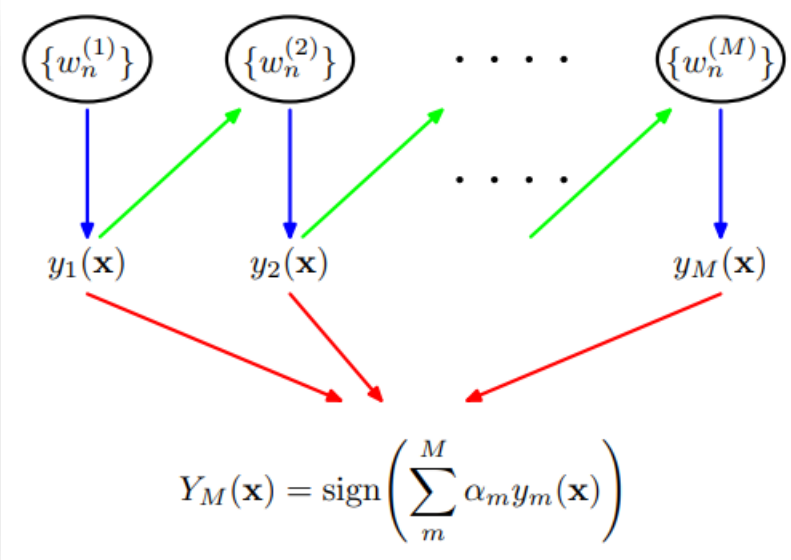


- Wichtigster Unterschied zu Bagging vorweg: bei Boosting werden die Weak Learner *seriell* und nicht parallel trainiert
- Jeder Weak Learner wird mit einer gewichteten Version des Datensatzes trainiert
- Diese Gewichte werden von der Performance des vorhergehenden Modells an diesen Datenpunkten abgeleitet
  → Missklassifikation führt zu höheren Gewichten
- Nachfolgend trainierte Weak Learner sollen dadurch den Fokus auf problematische Datenpunkte legen
- Die gewichtete Summe aller Weak Lerner ergibt dann den Strong Learner

Quellen: Bishop, 2006. S. 657-59

Deep Dive: AdaBoost

AdaBoost



1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.
   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right). \tag{14.19}$$

Bishop, 2006. S. 658

AdaBoost

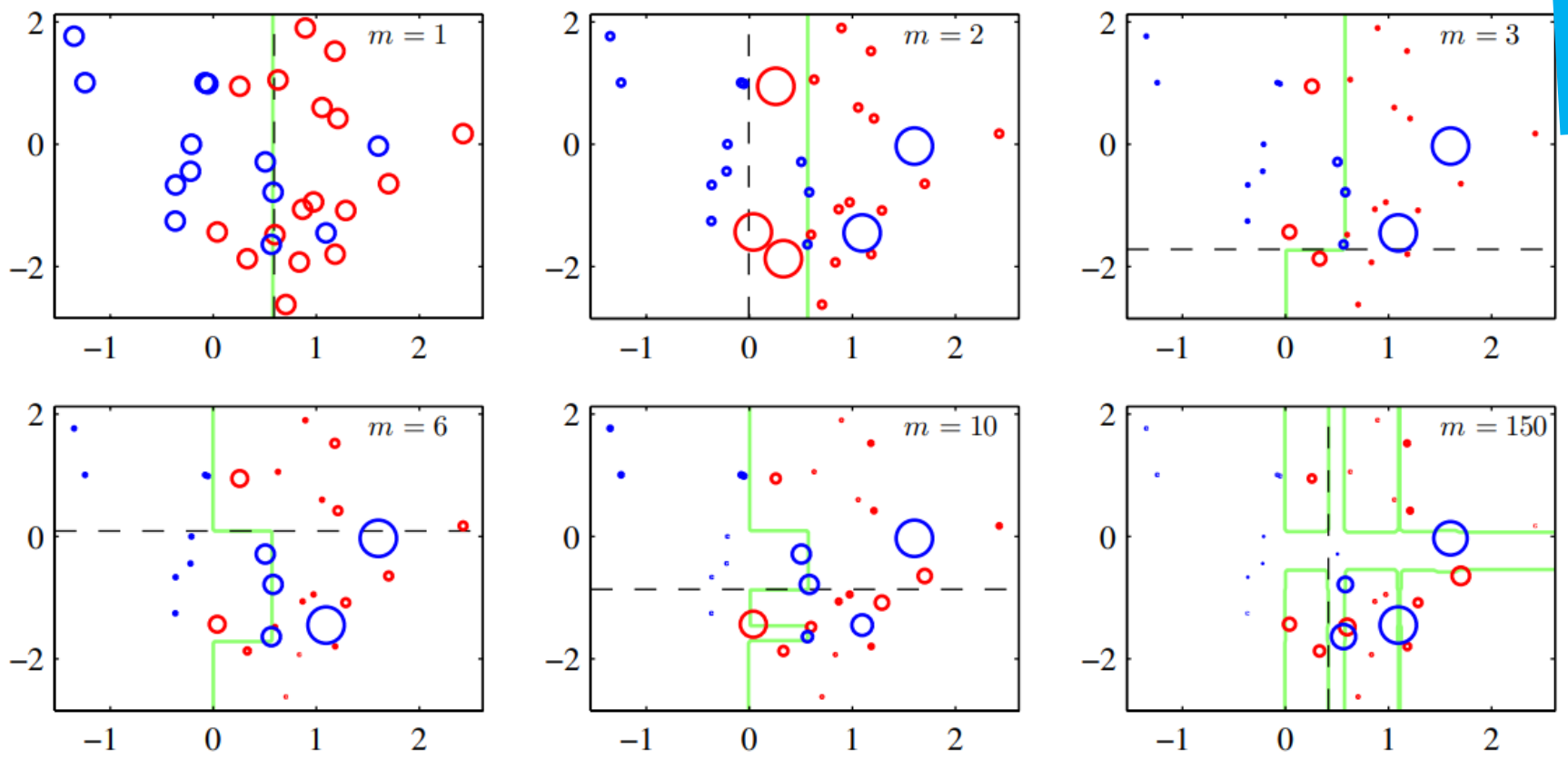


Figure 14.2 Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training the $m = 2$ base learner.

(Bishop, 2006)

Was denken Sie?

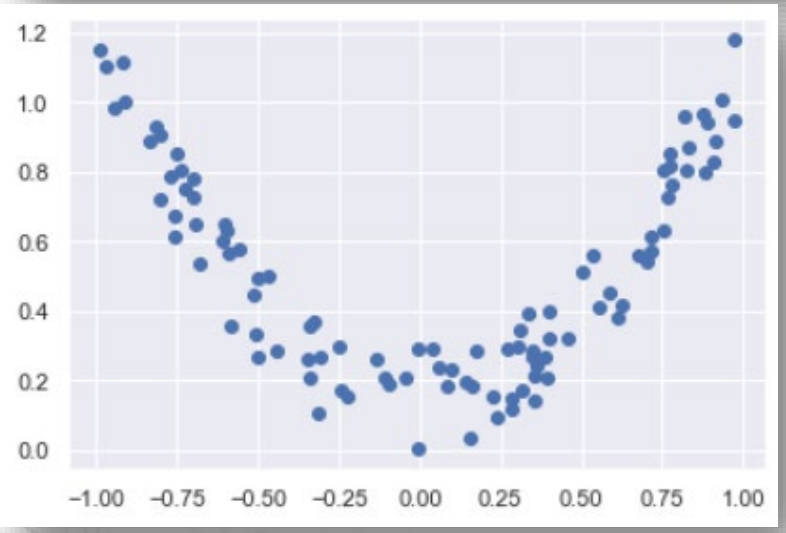
Wo wird im Falle des Random Forest die Gewichtung der Datenpunkte eingehen?

So what?

Hier: Kombination aus Single Split Trees durch Boosting führt zu einem Strong Learner

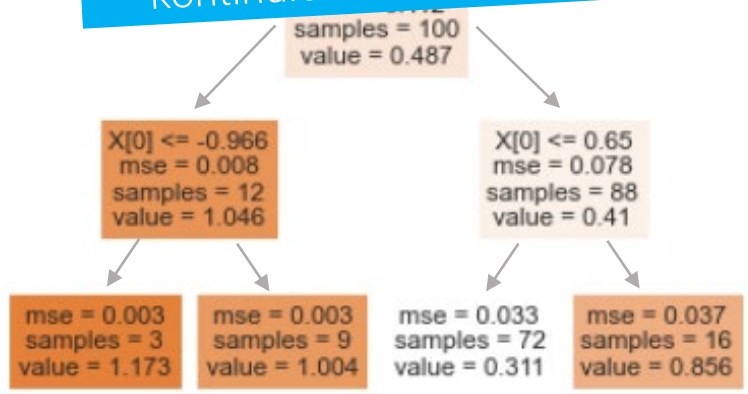
Random Forest Regression

```python
# Import
from sklearn.tree import DecisionTreeRegressor, plot_tree

# Make quadratic data
index = np.random.uniform(-1, 1, size=(100, 1))
data = index ** 2 + np.random.rand(len(index), 1) * .3

# Fit decision tree regressor
model = DecisionTreeRegressor(max_depth=2, random_state=42)
model.fit(index, data)

# Plot tree
plot_tree(model, filled=True);
```



Was denken Sie?

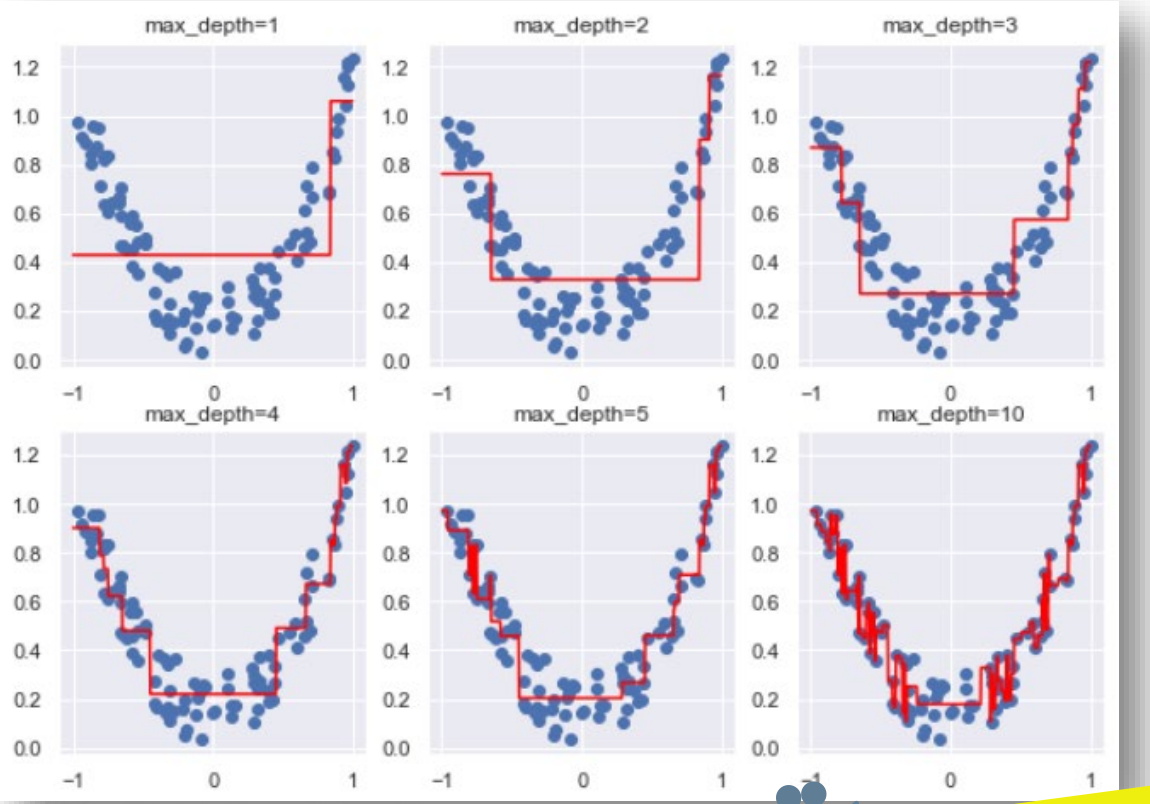
Jetzt wissen Sie wie Klassifikation mit RFs funktioniert: wie kann man nun kontinuierliche Werte vorhersagen?



- Anstatt eines kategorialen Labels können wir auch *kontinuierliche* Werte vorhersagen → Regression
- Der vorhergesagte kontinuierliche Wert ergibt sich dann aus dem Mittelwert der Samples, die in den jeweiligen Bereich des Splits fallen
- Wir brauchen für die Entscheidung, wo gesplitted werden soll, ein anderes Maß als Impurity
  → MSE
- Bei der Random Forest Regression werden also Decision Trees erzeugt, bei denen die Splits zu einer *Minimierung des MSE* führen

Géron, 2019. S. 183-84

Random Forest Regression

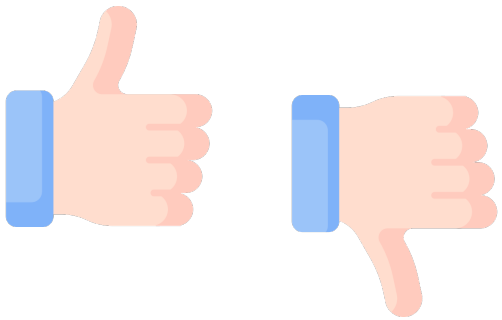



```python
# Import
from sklearn.tree import DecisionTreeRegressor

# Make quadratic data
index = np.random.uniform(-1, 1, size=(100, 1))
data = index ** 2 + np.random.rand(len(index), 1) * .3

# Fit decision tree regressors
index_predictions = np.linspace(-1, 1, 500).reshape(-1, 1)
fig, axes = plt.subplots(2, 3, figsize=(10, 7))
axes = axes.flat
for n, max_depth in enumerate([1, 2, 3, 4, 5, 10]):
    model = DecisionTreeRegressor(max_depth=max_depth)
    model.fit(index, data)

    # Plot predictions
    predictions = model.predict(index_predictions)
    axes[n].scatter(index, data)
    axes[n].plot(index_predictions, predictions, color='red')
    axes[n].set_title(f'max_depth={max_depth}')
```

Demo
Wir gehen den Code im JupyterNotebook durch

Géron, 2019. S. 183-84

Vorteile, Nachteile, Besonderheiten: Random Forest



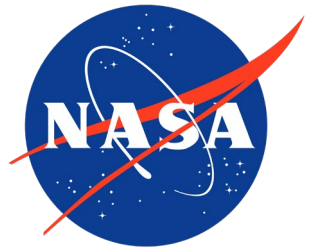
Pros:

- Schnelles Training (einzelner Decision Trees → Trainingszeit linear mit der Anzahl der Bäume)
- Schnelle Evaluierung auf jedem Baum einzeln → parallelisierbar
- Sehr effizient bei großen Datenmengen (viele Klassen, viele Samples, viele Features)
- `.predict_proba()` möglich!
- Kann mit numerischen und kategorischen Variablen umgehen
- Standardisierung nicht nötig
- Eines der mächtigsten Verfahren aus dem „klassischen“ Machine Learning

Cons:

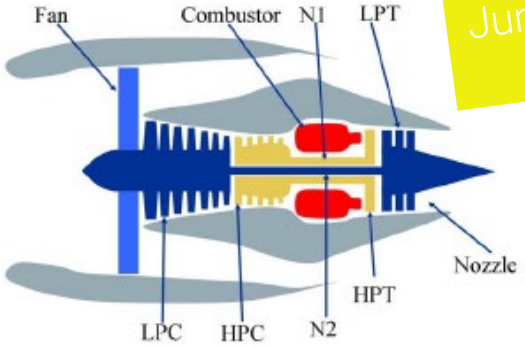
- Im Vergleich zu einzelnem Decision Tree → Black-Box
- Besteht „nur“ eine lineare Abhängigkeit zwischen Features und Label, dann führt ein Random Forest evtl. zu keinem Genauigkeitszugewinn im Vergleich zu einem einzelnen Decision Tree
- Bei mehreren kategorischen Features im Datensatz führt ein Ensemble evtl. auch zu keinem Zugewinn

Large Guided Example



Predictive Maintenance on Turbofan Engines:
Random Forest Regression for Remaining Useful Life Prediction

Random Forest Regression: RUL Prediction



Demo
Jump to Jupyter Notebook

- Wir wollen uns nun einen Use Case ansehen, in dem wir die RUL einer Turbofan Engine modellieren

- Hierzu nutzen wir den NASA-Datensatz „*Turbofan Engine Degradation Simulation Data Set*“

Allgemeine Datensatzbeschreibung:

- Verschlechterung des Motorzustands eines Turbofans wurde mittels *Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)* simuliert
- Jede Zeitserie entspringt einem anderen Motor → Flotte an Motoren des gleichen Typs
- Trainingsdaten: 100 Zeitserien, Testdaten: 100 Zeitpunkte
- Alle Motoren arbeiten **„normal“ zu Beginn** und entwickeln dann **zu bestimmter Zeit Schäden**, die kontinuierlich bis zum Maschinenausfall fortschreiten

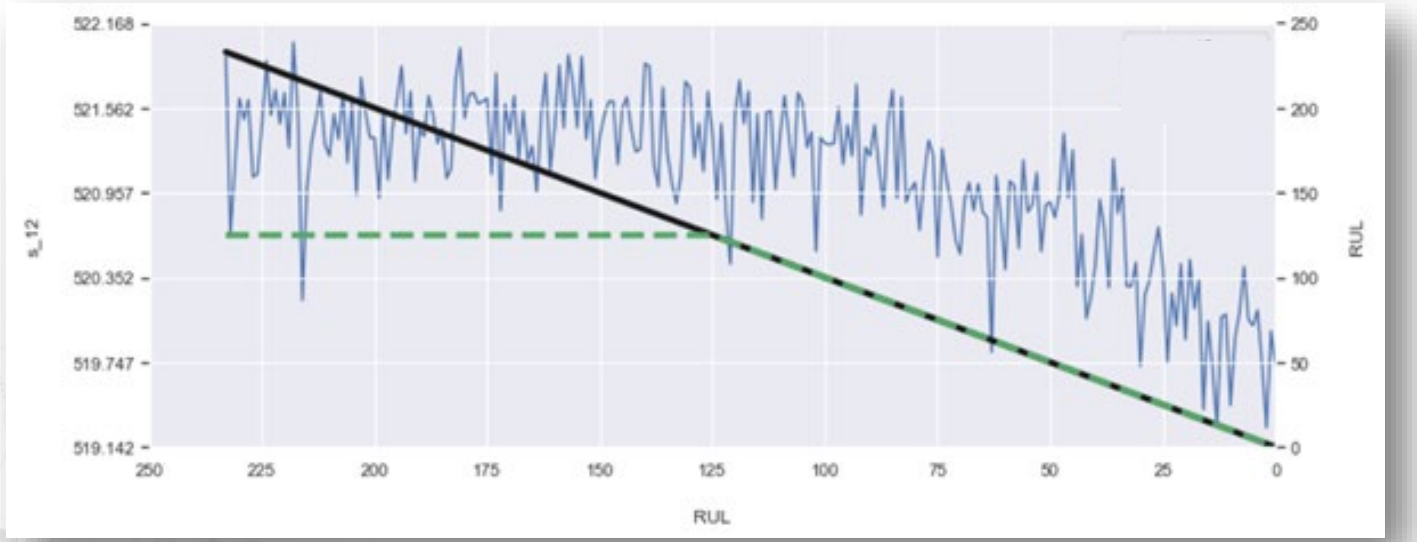
Aufgabe: Vorhersage des verbleibenden Abnutzungsvorrats (RUL Prediction)

So what?
Keine Domäneninformation vorhanden! Wir wissen nur, *dass* es Sensoren gibt, die etwas aufzeichnen. Ein „gutes“ Modell wird somit umso wichtiger!

https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/

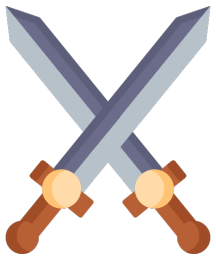
Crude Approximation: RUL at begin normal, decrease begins at some point

- Im Datensatz wird angenommen, dass die RUL zu Beginn „normal“ also noch „voll“ ist und ab einem bestimmten Punkt (125 Zeiteinheiten vor Schaden) beginnt abzunehmen
- Das Modell wird dadurch lernen sich auf die kritische Phase vor dem Engine-Ausfall zu konzentrieren und die normalen Zeiträume eher außer Acht zu lassen
- RUL vs. TTF: würden wir versuchen die TTF vorherzusagen wäre die schwarze Linie unser Target, bei unserer RUL handelt es sich um die gestrichelte grüne Linie



Demo
Back to Jupyter Notebook

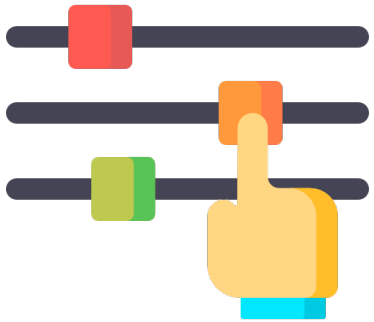
Baseline Models



- Um Machine Learning Modelle bewerten zu können, führt man häufig sog. *Baseline Models* ein
- Darunter versteht man einfache bzw. nicht so „mächtige“ Modelle, wie das Modell im Fokus
- Typische Beispiele:
  - Lineare Regressionen
  - Moving-Average
  - Exponential Smoothing
  - Konstante Vorhersage
  - etc.
- Diese Baseline Models müssen signifikant in der Vorhersagegenauigkeit übertroffen werden, um den zusätzlichen Aufwand bzw. Sprung in der Komplexität mittels Machine Learning Modellen zu rechtfertigen

Demo
Back to Jupyter Notebook

Hyperparameter Sweep: sophisticated guesses

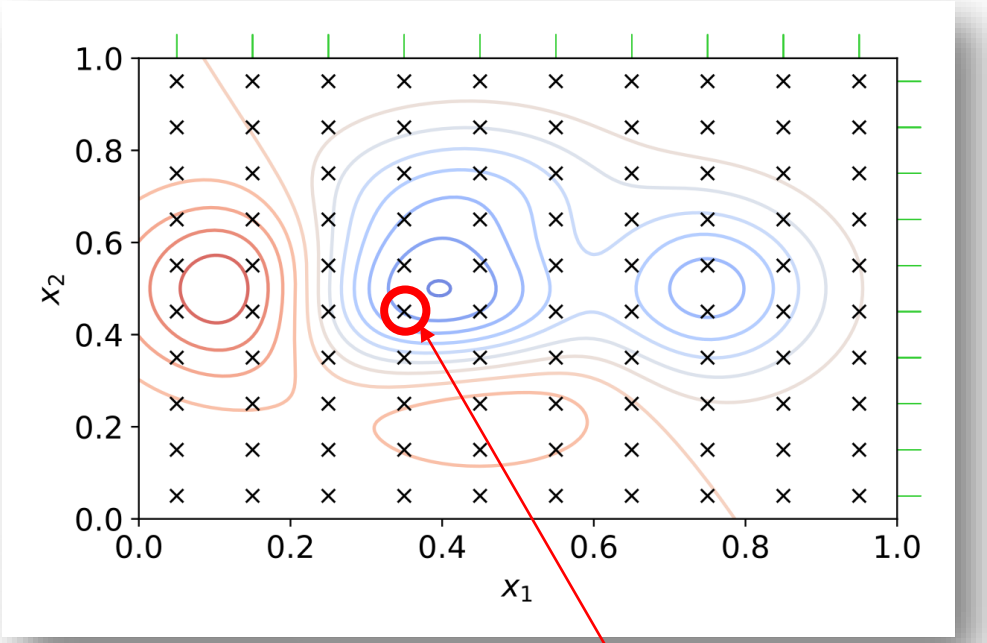


- „Sophisticated Guesses“ == Datengetriebene Einschränkung der Hyperparameterintervalle
- Um nicht „blind“ einen Grid Search in irgendwelchen Intervallen von Hyperparameterwerten suchen zu lassen, versucht man durch Eigenschaften von Daten oder ersten Modellen diese Intervalle einzuschränken
- Mögliche Herangehensweisen:
  - **Eigenschaften erster Modelle**: wie sehen die Hyperparameterwerte erster Versuche aus? Können wir daraus etwas ableiten?
  - **Einschränkung durch Dateneigenschaften**: legen uns unsere Daten sinnvolle Einschränkungen auf?
  - **Wertebereiche der verschiedenen Hyperparameter**: kann man für den vorliegenden Use Case schon auf sinnvolle und weniger sinnvolle Bereiche schließen?

Beyond Grid Search

Was denken Sie?

Was könnten wir denn noch tun außer Grid Search?



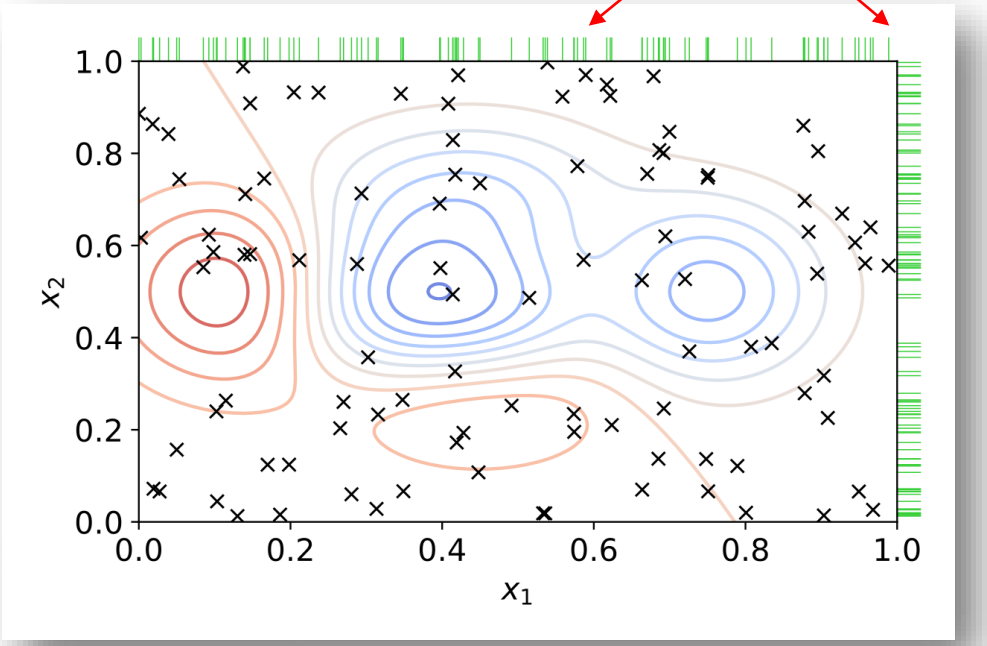
Optimale Parameterkombination aus Grid Search

- Grid Search ist ein sog. *Exhaustive Searching (Brute Force)* im aufgespannten Parameterraum
- Je nach Dimensionalität des Hyperparameterraums können Trainingsvorgänge eines Modells beliebig lang dauern
- Daher gibt es „smartere“ Versionen einer Hyperparameter-Optimierung:
  - Random Search
  - Bayesian Optimization
  - Gradient-based Optimization
  - Simulated Annealing
  - Etc.

https://en.wikipedia.org/wiki/Hyperparameter_optimization

Beyond Grid Search: Random Search

Erhöhte Anzahl an individuellen Werten



- Anstatt der Brute-Force Methode des Grid Search kann man die Parameterkombinationen auch zufällig auswählen
- Hierzu werden aus den Parameterintervallen zufällig Werte (aus einer Verteilung) gezogen
- Für jede Kombination wird die Kostenfunktion berechnet und das Optimum verwendet
- Interessant: im Vergleich zum Grid Search resultiert diese Methode in einer höheren Anzahl an individuellen Werten auf den einzelnen Parameterachsen

Was denken Sie?

Vergleichen Sie die grünen Striche hier mit denen beim Grid Search: was fällt auf?

Demo
Back to Jupyter Notebook

So what?
Man kann zeigen, dass schon 60 zufällig gezogene und eindeutige Parameterkombinationen zu 95% an das Optimum heran bringen.

https://en.wikipedia.org/wiki/Hyperparameter_optimization

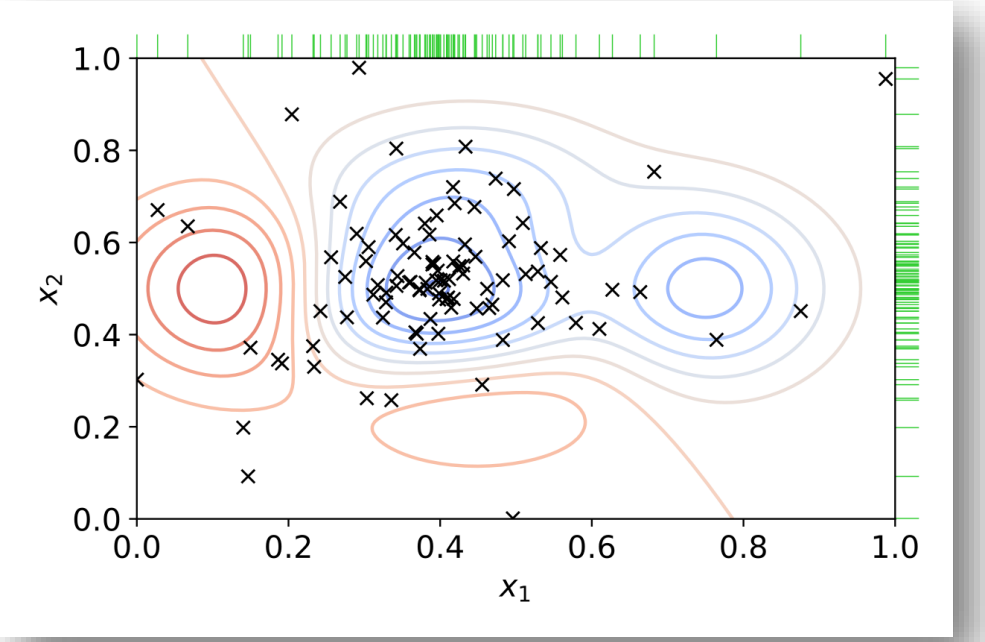
Beyond Grid Search: Random Search: `sklearn`

sklearn.model_selection.RandomizedSearchCV

class `sklearn.model_selection.` **`RandomizedSearchCV`**(*estimator, param_distributions, *, n_iter=10, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score=nan, return_train_score=False*) [source]

- Wie für den Grid Search – gibt es auch für den Random Search eine Klasse in `sklearn`
- Die wichtigsten Parameter sind:
  - `estimator`: das Modell, das optimiert werden soll
  - `param_distribution`: die möglichen Parameterintervalle als z.B. Dictionary
  - `n_iter`: die Anzahl an gezogenen Parameterkombinationen
  - `cv`: Cross-Validation Splitting Strategy wird hiermit festgelegt – Integer für Anzahl an Splits oder *cross-validation generator*

Beyond Grid Search: Bayesian Optimization



- Bayesian Optimization erzeugt iterativ ein Modell der Kostenfunktion
- Basierend auf diesem Modell werden vielversprechende Parameterkombinationen evaluiert und dann das Modell aktualisiert
- Dadurch können „intelligent“ Schritte in Richtung Optimum gegangen werden
- Bayesian Optimization führt in weniger Schritten zu besseren Resultaten als Grid und Random Search

Demo
Back to Jupyter Notebook

So what?
Bayesian Optimization ist ein gutes Beispiel für einen Algorithmus, der versucht *Exploration und Exploitation* auszubalancieren

Was denken Sie?

Können Sie sich vorstellen was Exploration and Exploitation bedeutet?

https://en.wikipedia.org/wiki/Hyperparameter_optimization