


Entscheidungsbäume und Random Forests I

Hinweis: Damit es zu keinem Missverständnis kommt: Random Forest ist ein Vertreter des *Shallow Learnings* („flaches“ Lernen).

Random Forest ist ein *Machine Learning*-Algorithmus¹, der auf Entscheidungsbäumen basiert. Hierbei werden n-viele Entscheidungsbäume zu einem Ensemble – dem Forest – geformt, ganz nach dem Motto: „Die Weisheit der Crowd“! Man geht also davon aus, dass mehrere Bäume zusammen, was in der Regel auch zutrifft, eine bessere Vorhersagegenauigkeit erzielen als ein einzelner. Wie das generell abläuft und was neben dem bereits erwähnten *Ensembling* noch überaus entscheidend ist, gilt es mithilfe des referenzierten Tutorials herauszufinden.

 14.1 Nehmen Sie das Tutorial *Understanding Random Forest*² durch.

Mit dem Random Forest-Algorithmus kann man also Klassifizierungsaufgaben vornehmen. Genau das wollen wir uns am Beispiel *Iris Dataset* im Detail ansehen. Es ist bekannt, dass das Dataset mit *Iris {setosa, virginica, versicolor}* über drei Klassen mit den Features *Sepal {length, width}* und *Petal {length, width}* verfügt. Über welche Gattung es sich im jeweiligen Fall handelt, gibt die Spalte *species* (siehe Abbildung 1) Auskunft. Wir verfügen also über „Beispiele“ und „Antworten“ – also all das, was wir benötigen, um einen Random Forest-Algorithmus zu trainieren und dessen Vorhersagen zu evaluieren.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Abbildung 1: Iris Dataset

Wir benötigen folgende Imports aus der *Scikit Learn*-Sammlung³:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

#Create a classifier of type "RandomForest"
clf = RandomForestClassifier(max_depth=5, n_estimators=10)
```

¹ Vgl. Abbildung 1 in KI4_B02-MachineLearningGrundlagen.pdf (htl.boxtree.at/lehre)

² <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

³ Natürlich braucht es auch all jene Imports, die im Zusammenhang mit dem Laden und Aufbereiten des Iris Datasets vorgestellt wurden.

im nächsten Schritt gilt es die Daten aufzubereiten, und zwar in die Beispiel x und die dazugehörigen Antworten y :

```
X = # your job
```

```
Y = # your job
```

x ist eine Matrix, die alle Features des Datasets beinhaltet (n Zeilen x 4 Spalten). y ist ebenfalls eine Matrix, die die Antworten bereitstellt (n Zeilen x 1 Spalte).

```
print(X.head())
```

```
print(y.head())
```

```
#Output X.head():
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
#Output y.head():
```

0	0
1	0
2	0
3	0
4	0

```
Name: species, dtype: int32
```

Die Datenaufbereitung ist abgeschlossen. Es gilt den *Classifier*⁴ zu trainieren. Hierzu teilen wir die Daten in Trainings- und Validierungsdaten (Test-Daten) auf, und zwar mit `train_test_split()`⁵. Im Beispielsfall werden 30% für spätere Tests/Validierungen zurückgehalten.

```
# split the data in a "training set" and a "test set"
```

```
# 30% is the size of our "test set" to check the results of the predicted values
```

⁴ <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

⁵ https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?highlight=train_test_split#sklearn.model_selection.train_test_split

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3)


# "put" the data („Beispiele“ + „Antworten“) into the classifier
clf.fit(X_train,y_train)


# store the predicted values in a matrix with one column (y_pred)
y_pred=clf.predict(X_test)
print(y_pred)

#Output y_pred:
array([2 2 2 1 1 2 0 1 2 0 0 2 0 0 1 0 0 0 1 1 1 2 2 0 2 2 2 0 1 2 1 1 2 1
0 0 2 1 0 1 1 1 2 0 1])


```

Die Vorhersagen für die 45 zurückgehaltenen Test-Beispiele sind in `y_pred` enthalten. Die dazugehörigen Antworten in `y_test`.


 14.2 Implementieren Sie den vorgestellten *Classifier* und geben Sie dessen Vorhersagen (`y_pred`) für das Iris Dataset aus. Datenaufbereitung (siehe gelbe Markierungen) nicht vergessen. **Hinweis:** Die erzielten Ergebnisse weichen definitiv von den hier dokumentierten ab. Das ist der Auswahl der „Beispiele“ in `train_test_split()` geschuldet.

 14.3 Im nächsten Schritt ist die Erstellung eines *DataFrames* mit den Spalten „predicted“, „actual“ sowie „correct“ das erklärte Ziel. Es gilt also auszuwerten, ob die Vorhersage (`y_pred`) gleich der vorgegeben „Antwort“ ist. Das Ergebnis ist in der neu zu erstellenden Spalte „correct“ anzuzeigen:

	predicted	actual	correct
0	2	2	1
1	2	2	1
2	2	2	1
3	1	1	1
4	1	2	0

 14.3 Abschließen sind wir natürlich auch noch an der *Accuracy* unseres Classifiers interessiert. Darunter ist jener Zahlenwert zu verstehen, der Auskunft über die Anzahl der richtig klassifizierten Beispiele gibt. Ermitteln Sie diese ohne Zuhilfenahme irgendeiner Scikit Learn-Ressource – also einfach durch die vorhandenen Daten aus 14.3. **Tipp:** Die Spalte „correct“ liefert die entsprechende Grundlage. Im Beispielsfall beläuft sich diese auf ca. 93%:

```
# 0.9333333333333333
```

 14.4 Welche Bedeutung wird den Argumenten `max_depth` und `n_estimators` beim Instanzieren des Classifiers zuteil? Probieren Sie, ob durch Veränderung der vorgeschlagenen Werte eine Verbesserung des Ergebnisses möglich ist.

So, der Classifier ist trainiert und validiert. Jetzt ist es Zeit, „neue“ Wert (Blumen) klassifizieren zu lassen. Ein Beispiel mit zufälligen Werten liefert als Ergebnis „2“, also Versicolor.

```
clf.predict([[6, 3.0, 5.5, 1.8]])
```

Output:

```
array([2])
```