Python scikit-learn - Supervised Learning

Arbeiten mit dem IRIS Datensatz – Classification in python scikit-learn

```
#Import der Bibliotheken und Vorbereitung
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#Einstellen des Plot-Styles
#Welche weiteren styles gibt es ?
plt.style.use('ggplot')
```


Weitere styles sind hier zu finden:

https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html

```
#Laden der sklearn datasets
from sklearn import datasets
```

```
#Laden des IRIS Datensatzes, der in scikit-learn enthalten ist in die
#Variable iris
iris = datasets.load_iris()
```

```
#Überprüfen mit type()
type(iris)
```

```
#Ausgabe der keys()
print(iris.keys())
```

```
#Ausgabe der iris.DESCR mit print()
print(iris.DESCR)
```

```
#Überprüfen des type() von data und target
type(iris.data), type(iris.target)
```

```
#Ausgabe der Struktur
iris['data'].shape
```

```
#Ausgabe der Struktur
iris['target'].shape
```

```
#Ausgabe der Struktur verkettet
iris.data.shape
```

```
#Ausgabe der target_names
iris.target_names
```

```
#Zuweisung der Daten an die Variable X
X = iris.data
```

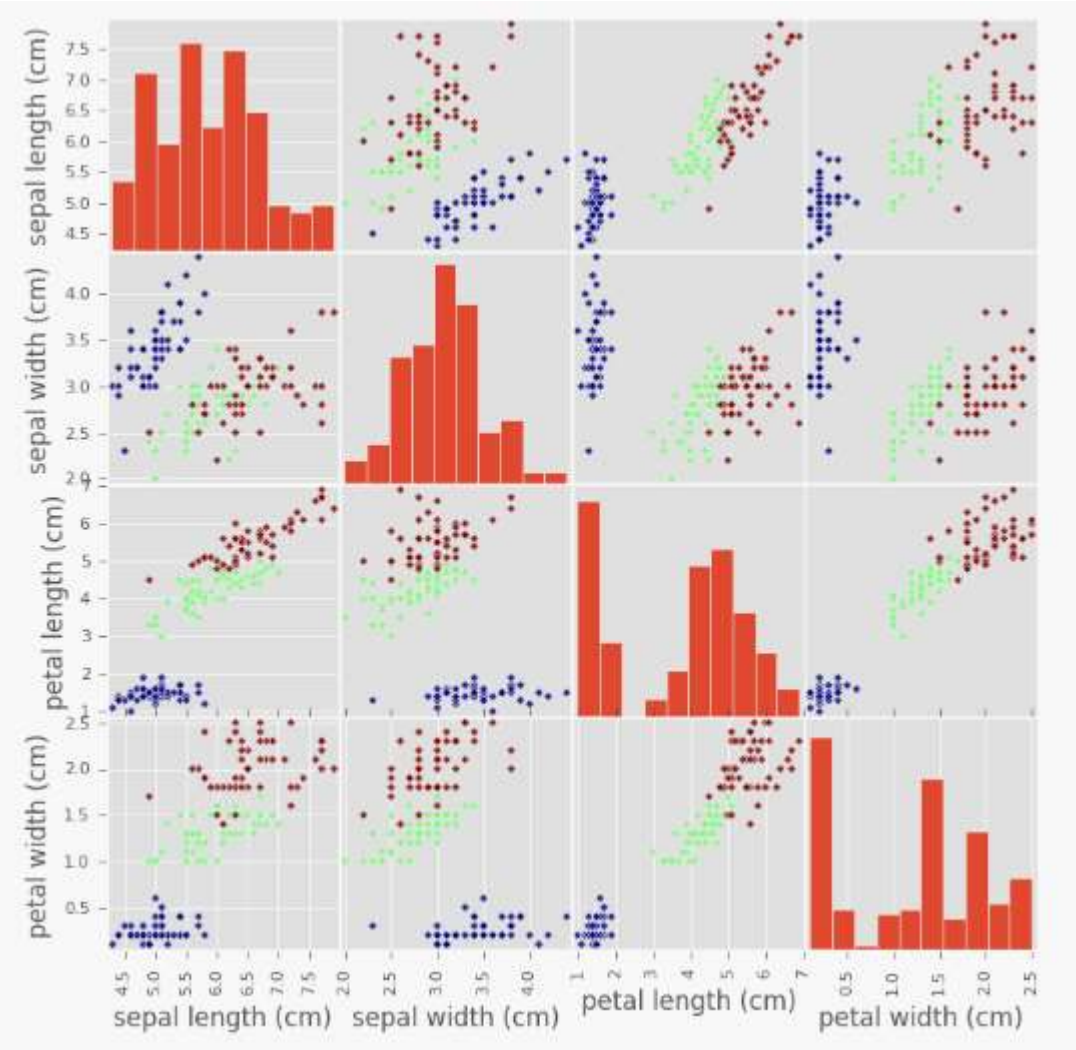
```
Zuweisung der target Daten an die Variable y
y = iris.target
```

```
#Übergabe der Features der Variable X an einen Dataframe df, als Spaltennamen
#werden die feature_names verwendet
df = pd.DataFrame(X, columns=iris.feature_names)
```



```
#Ausgabe der ersten 5 Zeilen aus dem Dataframe df
print(df.head())
```

```
#Erstellen einer Scatter Matrix
_ = pd.scatter_matrix(df, c = y, figsize = [8, 8], s=10, marker = 'D')
```



```
#Verwenden von scikit-learn um einen Classifier zu trainieren
#Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
#Erstellen einer Instanz mit 6 Nachbarn
knn = KNeighborsClassifier(n_neighbors=6)
```

```
#Verwenden von .fit() um das Modell zu trainieren
knn.fit(iris['data'], iris['target'])
```

```
#Zuweisung neuer Werte an die Variable X_new
X_new=np.array([[1.8, 3.1, 2.5, 0.3], [2.6, 3.3, 2.2, 0.2], [5.7, 3.2, 5.6, 0.3]])
```


Achtung: doppelte eckige Klammern !

```
#Vorhersage mit neuen Daten ausführen
prediction = knn.predict(X_new)
```

```
#Ausgabe der Struktur von X_new
X_new.shape
```

```
#Ausgabe der Vorhersage
print('Prediction {}'.format(prediction))
```

Verwenden von train/test split aus scikit-learn

```
# Import notwendiger Module
#Import des Classifiers
from sklearn.neighbors import KNeighborsClassifier
#Import von train_test_split von sklearn.model_selection
from sklearn.model_selection import train_test_split
```

```
#Splitten der Daten
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
...: random_state=21, stratify=y)
```

Wirken sich unterschiedliche test_sizes auf das Ergebnis aus ? Warum wird stratify hier auf y gesetzt?

Welche Bedeutung hat random_state=21 ?

```
#Erstellen einer Instanz mit 8 Nachbarn
knn = KNeighborsClassifier(n_neighbors=8)
```

```
#Verwenden von .fit() um das Modell auf den Trainingsdaten zu trainieren
knn.fit(X_train, y_train)
```

```
#Verwenden von .predict() um auf den Testdateneine Vorhersage zu erzeugen
y_pred = knn.predict(X_test)
```

```
#Ausgabe der Test set Vorhersagen
print("Test set predictions:\n {}".format(y_pred))
Test set predictions:
```

```
#Ausgabe der accuracy von k-NN mit knn.score()
knn.score(X_test, y_test)
```

Arbeiten mit dem MNIST Datensatz – Classification in python mit scikit-learn

```
# Import der notwendigen Module und Vorbereitung
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```



```
#Laden der sklearn datasets
from sklearn import datasets
```

```
# Laden des digits Datensatzes in die Variable digits
digits = datasets.load_digits()
```

```
# Ausgabe mit print() der keys und der DESCR dieses Datensatzes
print(digits.keys())
print(digits.DESCR)
```

```
# Ausgabe mit print der Struktur shape of the images und data keys
print(digits.images.shape)
print(digits.data.shape)
```

```
# Ausgabe digit 1010
plt.imshow(digits.images[1010], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```

```
# Ausgabe digit 1121
plt.imshow(digits.images[1010], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```

```
# Import notwendiger Module
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
# feature and target arrays erzeugen
X = digits.data
y = digits.target
```

```
# Spliten der Daten
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state=42, stratify=y)
```

```
# Erzeugen eines k-NN classifiers mit 7 Nachbarn: knn
knn = KNeighborsClassifier(n_neighbors=7)
```

```
# Trainieren des Classifiers
knn.fit(X_train, y_train)
```

```
# Ausgabe der accuracy
print(knn.score(X_test, y_test))
```

Overfitting and underfitting einmal überprüfen

Jetzt berechnen und zeichnen wir die Trainings- und Testgenauigkeitswerte für eine Reihe verschiedener Nachbarwerte. Wenn wir beobachten, wie sich die Genauigkeitswerte für Trainings- und Testdaten mit unterschiedlichen Werten von k unterscheiden, entwickeln wir eine Intuition für Über- und Unterfitting.

Die Trainings- und Testdaten stehen ja bereits im Arbeitsbereich als X_train, X_test, y_train, y_test zur Verfügung. Außerdem wurde KNeighborsClassifier von sklearn.neighbors importiert.

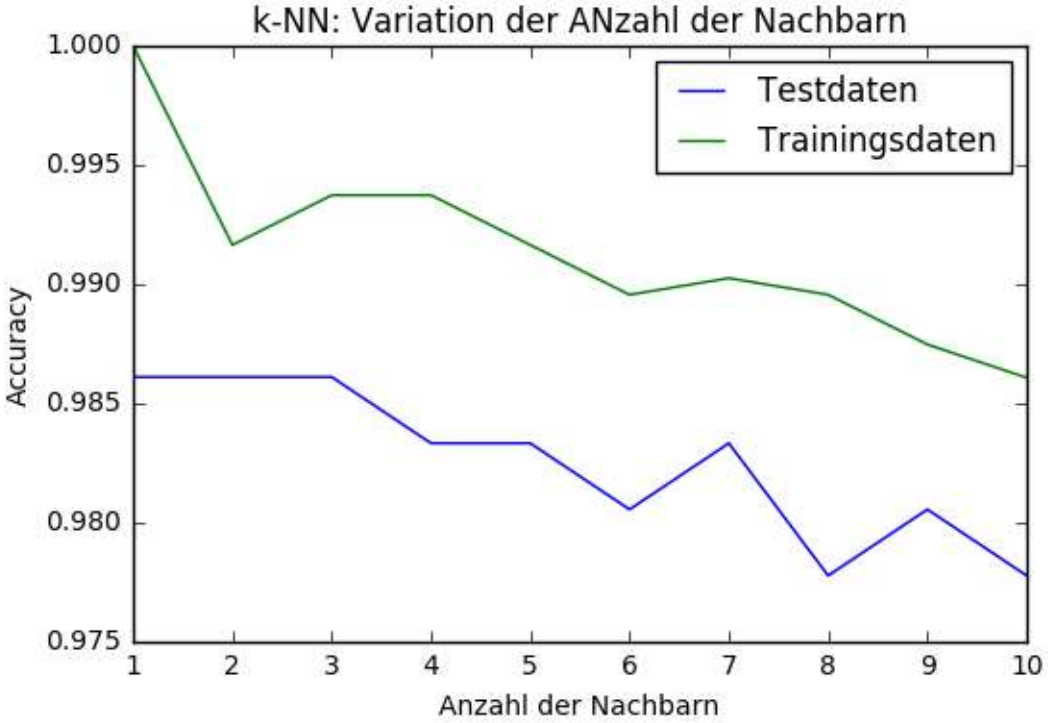
```
#Wir setzen arrays auf um die Ergebnisse für verschiedene Neighbors (1-11) zu
#überprüfen
neighbors = np.arange(1, 11)
#Ausgabe des arrays mit print()
print(neigbors)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
# Schleife über verschiedene Werte von k
for i, k in enumerate(neighbors):
    # Erzeugen eines k-NN Classifiers mit k Nachbarn: knn
    knn = KNeighborsClassifier(n_neighbors=k)
    #Trainieren des Classifiers mit .fit()
    knn.fit(X_train, y_train)
    #Berechne accuracy auf den Trainingsdaten
    train_accuracy[i] = knn.score(X_train, y_train)
    #Berechne accuracy auf den Testdaten
    test_accuracy[i] = knn.score(X_test, y_test)
```

Was macht enumerate()?

https://docs.python.org/2/library/functions.html

```
# Ausgabe erzeugen
plt.title('k-NN: Variation der Anzahl an Nachbarn')
plt.plot(neighbors, test_accuracy, label = 'Testdaten Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Trainingsdaten Accuracy')
plt.legend()
plt.xlabel('Anzahl von Nachbarn')
plt.ylabel('Accuracy')
plt.show()
plt.savefig('mnist-accuracy-knn-10.png')
```



Weitere Metriken einsetzen

```
# Import der Module
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
# Splitten der Daten
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.4,
random_state=42)
# k-NN classifier: knn
knn = KNeighborsClassifier(n_neighbors = 3)
# Trainieren
knn = knn.fit(X_train,y_train)
# Vorhersage der labels der testdaten: y_pred
y_pred = knn.predict(X_test)
# Erzeuge die confusion matrix und den  classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Kreuz Validierung

```
from sklearn.model_selection import cross_val_score
knn = KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(knn, iris.data, iris.target, cv=5)
scores
np.mean(scores)
```

Arbeiten mit den Breast Cancer Daten

Classification

Entscheidungsbäume

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline
```



```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale, StandardScaler
```

```
#Vorbereiten des Drucks
plt.rcParams["figure.dpi"] = 200
np.set_printoptions(precision=3)
```

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

```
#Ausgabe der DESCR mit print()
print(cancer.DESCR)
```

```
#Zuweisen der Daten an X
X = cancer.data
```

```
#Zuweisen der Daten an y
y = cancer.target
```

```
#Splitten der Daten test_size = 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
stratify=y, random_state=21)
```

```
#Laden des Classifiers
from sklearn.tree import DecisionTreeClassifier
```

```
#Instanzieren des Classifiers zur Variablen tree
tree = DecisionTreeClassifier(max_depth=4, random_state=21)
```

```
#Trainieren des Classifiers
tree.fit(X_train, y_train)
```

```
#Ausgabe der Accuracy
tree.score(X_test, y_test)
```

```
#Arbeiten mit unterschiedlichen max_depth Werten
depth_range = range(1,11)
```

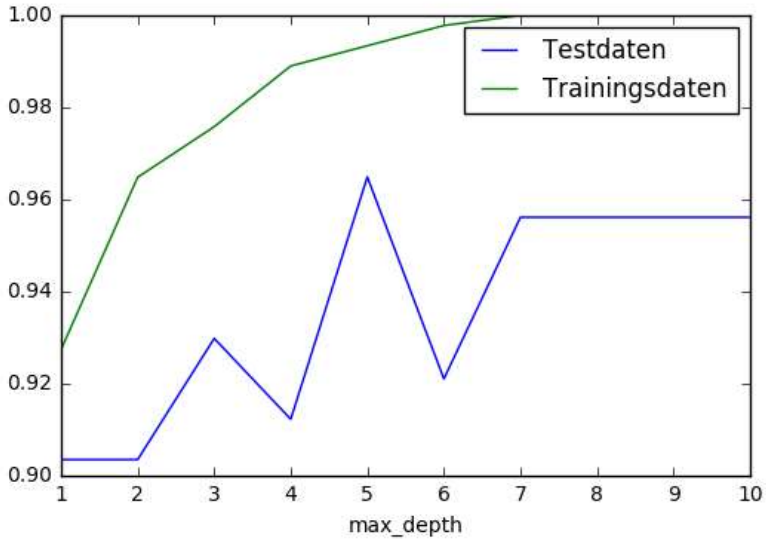
```
#Ausgabe
print(depth_range)
```



```
#Erzeugen der Listen
train_scores = []
test_scores = []
```

```
for k in depth_range:
    tree = DecisionTreeClassifier(max_depth=k, random_state=21)
    tree.fit(X_train, y_train)
    train_scores.append(tree.score(X_train,y_train))
    test_scores.append(tree.score(X_test,y_test))
```

```
#Ausgabe mit plot()
plt.plot(depth_range, test_scores, label = "Testdaten")
plt.plot(depth_range, train_scores, label = "Trainingsdaten")
plt.ylabel = ("Accuracy")
plt.xlabel("max_depth")
plt.legend()
plt.savefig("breast-cancer-tree-max-depth-10.png")
plt.show()
```



Arbeiten mit den Brest Cancer Daten – python scikit-learn

Random Forests

```
#Laden und vorbereiten der Bibliotheken
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline
```

```
#Setzen einiger Plot Parameter
plt.rcParams["figure.dpi"] = 200
np.set_printoptions(precision=3)
```



```
# Laden wichtiger sklearn Werkzeuge
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale, StandardScaler
```

```
# Laden der breast-cancer-daten und Übergabe an die Variable cancer
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

```
# Splitten der Daten
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=0)
```

```
# Laden des Random Forest Classifiers
from sklearn.ensemble import RandomForestClassifier
```

```
#Initiieren der Listen für train_scores und test_scores
train_scores = []
test_scores = []
```

```
# Instanz des RandomForestClassifiers an Variable rf übergeben
rf = RandomForestClassifier(warm_start=True)
```

```
#Festlegen der estimator_range
estimator_range = range(1, 100, 5)
```

```
#Berechnen der train_scores und der test_scores über der estimator_range
for n_estimators in estimator_range:
    rf.n_estimators = n_estimators
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))
```

```
#Ausgabe mit plot()
plt.plot(estimator_range, test_scores, label="test scores")
plt.plot(estimator_range, train_scores, label="train scores")
plt.ylabel("accuracy")
plt.xlabel("n_estimators")
plt.legend()
plt.show()
```

Out of Bag Vorhersagen


```
train_scores = []
test_scores = []
oob_scores = []
```

```
feature_range = range(1, 64, 5)
```

```
for max_features in feature_range:
    rf = RandomForestClassifier(max_features=max_features, oob_score=True,
n_estimators=200, random_state=0)
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))
    oob_scores.append(rf.oob_score_)
```

```
plt.plot(feature_range, test_scores, label="test scores")
plt.plot(feature_range, oob_scores, label="oob_scores scores")
plt.plot(feature_range, train_scores, label="train scores")
plt.legend()
plt.ylabel("accuracy")
plt.xlabel("max_features")
```

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=1)
```

```
rf = RandomForestClassifier(n_estimators=100).fit(X_train, y_train)
```

```
rf.feature_importances_
```

```
pd.Series(rf.feature_importances_, index=cancer.feature_names).plot(kind="barh")
```

Random Forest mit dem MNIST Datensatz durchführen

```
#Laden und vorbereiten der Bibliotheken
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline
```

```
#Setzen einiger Plot Parameter
plt.rcParams["figure.dpi"] = 200
np.set_printoptions(precision=3)
```



```
#Laden der MNIST Daten
from sklearn.datasets import load_digits
digits = load_digits()
```

```
#Splitten der Daten
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, stratify=digits.target, random_state=0)
```

```
# Laden des Random Forest Classifiers
from sklearn.ensemble import RandomForestClassifier
```

```
#Initiieren der Listen für train_scores und test_scores
train_scores = []
test_scores = []
```

```
# Instanz des RandomForestClassifiers an Variable rf übergeben
rf = RandomForestClassifier(warm_start=True)
```

```
#Festlegen der estimator_range
estimator_range = range(1, 100, 5)
```

```
#Berechnen der train_scores und der test_scores über der estimator_range
for n_estimators in estimator_range:
    rf.n_estimators = n_estimators
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))
```

```
#Ausgabe mit plot()
plt.plot(estimator_range, test_scores, label="test scores")
plt.plot(estimator_range, train_scores, label="train scores")
plt.ylabel("accuracy")
plt.xlabel("n_estimators")
plt.legend()
plt.show()
```

Arbeiten mit den Boston housing Daten

```
#Laden der Boston housing data
boston = pd.read_csv('boston.csv')
```

```
#Ausgabe der ersten 5 Datenreihen mit .head()
print(boston.head())
```



```
#Erzeugen der feature and target arrays
X = boston.drop('MEDV', axis=1).values
y = boston['MEDV'].values
```

```
# Vorhersage der Hauspreise mit einem einzelnen Feature, Anzahl der Räume, Spalte
6
X_rooms = X[:,5]
```

```
#Überprüfen des type
type(X_rooms), type(y)
```

```
#Überprüfen von shape für X und y
y.shape
X.shape
```

```
#reshape von y und X
y = y.reshape(-1, 1)
X_rooms = X_rooms.reshape(-1, 1)
```

```
#Plotting house value vs. number of rooms
plt.scatter(X_rooms, y)
plt.ylabel('Value of house /1000 ($)')
plt.xlabel('Number of rooms')
plt.show()
```

```
#Trainieren eines regression models auf der einen Variablen
import numpy as np
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(X_rooms, y)
```

```
#
prediction_space = np.linspace(min(X_rooms), max(X_rooms)).reshape(-1, 1)
```

```
#Plotten der Ergebnisse
plt.scatter(X_rooms, y, color='blue')
plt.plot(prediction_space, reg.predict(prediction_space),color='black',
linewidth=3)
```

```
plt.show()
```

```
#Lineare Regression mit allen features des Boston Datensatzes
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.3,
random_state=42)
reg_all = linear_model.LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)
reg_all.score(X_test, y_test)
```



```
#Cross-validation in scikit-learn
from sklearn.model_selection import cross_val_score
reg = linear_model.LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=5)
print(cv_results)
np.mean(cv_results)
```

Ridge Regression anwenden - Beispiel Boston Datensätze

```
Ridge regression in scikit-learn
from sklearn.linear_model import Ridge
```

```
#Splitten der Dateien
X_train, X_test, y_train, y_test = train_test_split(X, y,
...: test_size = 0.3, random_state=42)
```

```
#Instanz der Ridge Regression erzeugen
ridge = Ridge(alpha=0.1, normalize=True)
```

```
#Trainieren
ridge.fit(X_train, y_train)
```

```
#Vorhersage
ridge_pred = ridge.predict(X_test)
```

```
#Beurteilung der accuracy
ridge.score(X_test, y_test)
```

Lasso regression in scikit-learn – Beispiel Boston Datensätze

```
# Lasso Regression laden
from sklearn.linear_model import Lasso
```

```
#Splitten der Daten
X_train, X_test, y_train, y_test = train_test_split(X, y,
...: test_size = 0.3, random_state=42)
```

```
#Erzeugen der lasso Instanz
lasso = Lasso(alpha=0.1, normalize=True)
```



```
#Trainieren
lasso.fit(X_train, y_train)
```

```
#Vorhersage
lasso_pred = lasso.predict(X_test)
```

```
#Auswerten der accuracy
lasso.score(X_test, y_test)
```

Lasso Regression für feature selection in scikit-learn - Beispiel Boston Datensätze

```
from sklearn.linear_model import Lasso
names = boston.drop('MEDV', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
_ = plt.plot(range(len(names)), lasso_coef)
_ = plt.xticks(range(len(names)), names, rotation=60)
_ = plt.ylabel('Coefficients')
plt.show()
```

Boston Housing – Regression mit pipeline

```
# Aufsetzen der pipeline steps: steps
steps = [('imputation', Imputer(missing_values='NaN', strategy='mean', axis=0)),
         ('scaler', StandardScaler()),
         ('elasticnet', ElasticNet())]
```

```
# Erzeugen der pipeline: pipeline
pipeline = Pipeline(steps)
```



```
# Festlegen des hyperparameter space
parameters = {'elasticnet__l1_ratio':np.linspace(0,1,30)}
```

```
# Splitten der Daten
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=42)
```

```
# Erzeugen des  GridSearchCV object: gm_cv
gm_cv = GridSearchCV(pipeline, parameters)
```

```
# Trainieren der Dtaen
gm_cv.fit(X_train, y_train)
```

```
# Berechnung und Ausgabe der Metriken
r2 = gm_cv.score(X_test, y_test)
print("Tuned ElasticNet Alpha: {}".format(gm_cv.best_params_))
print("Tuned ElasticNet R squared: {}".format(r2))
```