02.02_API_LinReg

October 30, 2020

1 Estimator API: Einfache lineare Regression (supervised)

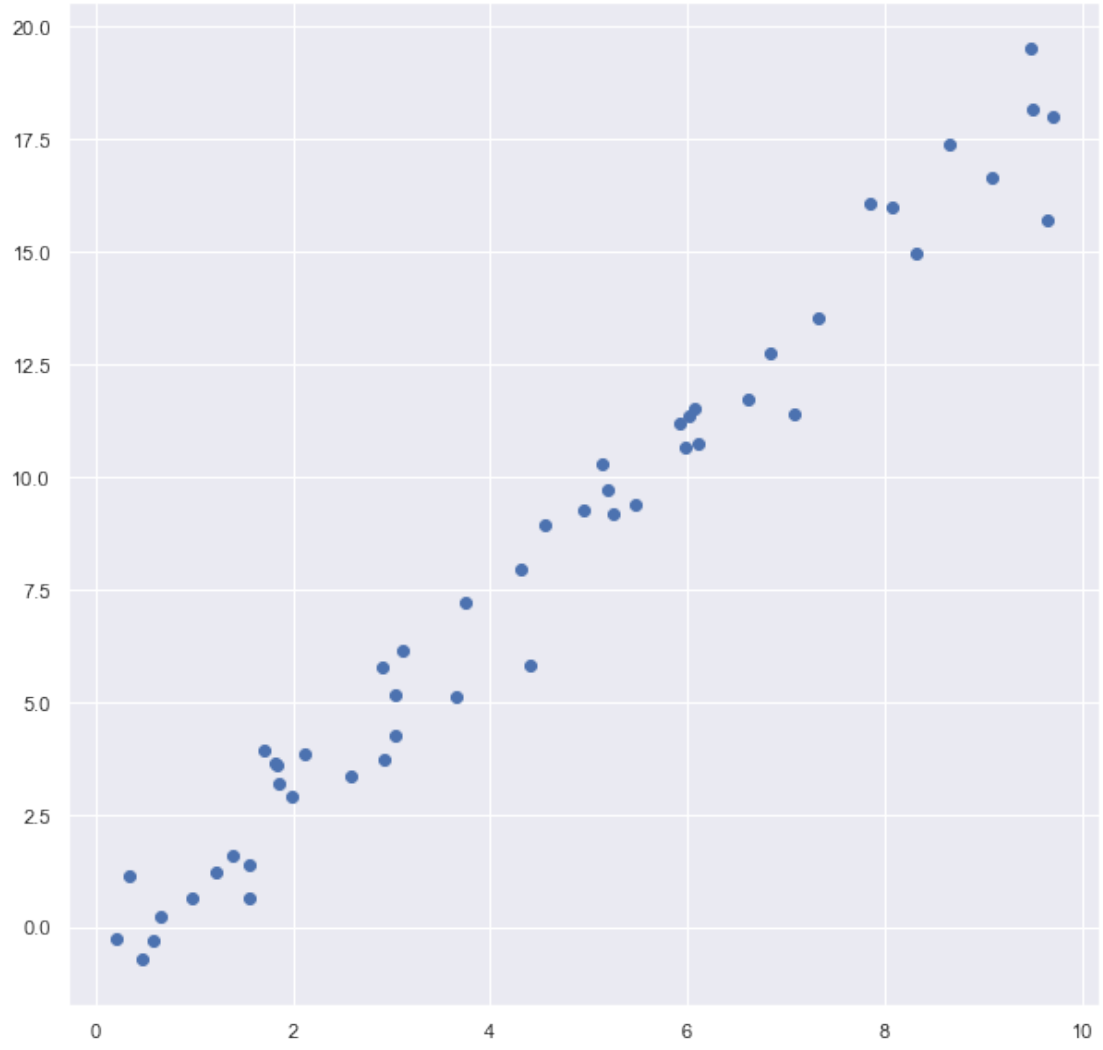
[1]:
```
%matplotlib inline
import seaborn as sns; sns.set()
```

1.1 0. Generiere Daten

Wir generieren 50 zufällige Datenpunkte zwischen 0 und 10, die **ungefähr** auf der Geraden $y = 2x - 1$ liegen.

[23]:
```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(42)
x = 10 * rng.rand(50)                # Erzeuge 50 zufällige x-Werte zwischen 0␣
 ↪und 10
y = 2 * x - 1 + rng.randn(50)        # Erzeuge die y-Werte: y = 2x-1 + zufälliges␣
 ↪Rauschen
plt.figure(figsize=(10,10))
plt.scatter(x, y);
```



1.2 1. Wähle Modellklasse

[3]:
```
from sklearn.linear_model import LinearRegression
```

1.3 2. Wähle Modellparameter

Alle Modelle sind ausführlich dokumentiert. Diese kann online eingesehen werden ("sklearn Linearregression") oder durch ein `?` aufgerufen werden.

[4]:
```
LinearRegression?
```

[16]:
```
model = LinearRegression(fit_intercept=True)
```

Bis hier ist noch nichts wesentliches passiert - es wurde nur das Modell angelegt, aber noch nicht auf die Daten angepasst.

1.4 3. Bereite Daten vor

[7]:
```
x
```

[7]:
```
array([3.74540119, 9.50714306, 7.31993942, 5.98658484, 1.5601864 ,
       1.5599452 , 0.58083612, 8.66176146, 6.01115012, 7.08072578,
       0.20584494, 9.69909852, 8.32442641, 2.12339111, 1.81824967,
       1.8340451 , 3.04242243, 5.24756432, 4.31945019, 2.9122914 ,
       6.11852895, 1.39493861, 2.92144649, 3.66361843, 4.56069984,
       7.85175961, 1.99673782, 5.14234438, 5.92414569, 0.46450413,
       6.07544852, 1.70524124, 0.65051593, 9.48885537, 9.65632033,
       8.08397348, 3.04613769, 0.97672114, 6.84233027, 4.40152494,
       1.22038235, 4.9517691 , 0.34388521, 9.09320402, 2.58779982,
       6.62522284, 3.11711076, 5.20068021, 5.46710279, 1.84854456])
```

[8]:
```
y
```

[8]:
```
array([ 7.22926896, 18.18565441, 13.52423055, 10.67206599,  0.64185082,
        1.4000462 , -0.29896653, 17.38064514, 11.36591852, 11.3984114 ,
       -0.26422614, 18.01311476, 14.97193082,  3.8584585 ,  3.66749887,
        3.59937032,  4.24562734,  9.18591626,  7.9701638 ,  5.80012793,
       10.75788366,  1.60421824,  3.736558  ,  5.13103024,  8.93392551,
       16.05975926,  2.92146552, 10.28822167, 11.2099274 , -0.7161115 ,
       11.51229264,  3.94851904,  0.26520582, 19.5423544 , 15.69289556,
       15.98984947,  5.17932245,  0.65443493, 12.77642131,  5.81548096,
        1.22109281,  9.26065077,  1.16566447, 16.66813782,  3.36710603,
       11.74868864,  6.14962364,  9.73011153,  9.40444538,  3.21035654])
```

x ist noch ein Vektor. Wir brauchen aber eine *Features Matrix* der Form `[n_samples, n_features]`.

[24]:
```
X = x[:, np.newaxis]
```

[25]:
```
X.shape
```

[25]:
```
(50, 1)
```

[26]:
```
y.shape
```

[26]:
```
(50,)
```

1.5 4. Passe das Modell an die Daten an

[27]:
```
model.fit(X,y)
```

[27]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Was ist jetzt passiert? Durch modellspezifische Berechnungen wurden modellspezifische Parameter bestimmt; das ist die eigentliche Arbeit. Hier: - Berechnung: Löse Normalengleichung - Ergebnis: Parameter der (affin)linearen Funktion; diese heißen `coef_` und `intercept_`

[28]:
```
model.coef_
```

[28]:
```
array([1.9776566])
```

[29]:
```
model.intercept_
```

[29]:
```
-0.9033107255311164
```

Das passt recht gut zu den Daten, die ja gemäß $y = 2x - 1$ erzeugt wurden und dann ein wenig verrauscht wurden.

1.6 5. Wende das Modell auf neue Daten an

In unserem *superviseden* Fall heißt das: Vorhersage eines Labels für unbekannte Daten (hier: x-Werte).

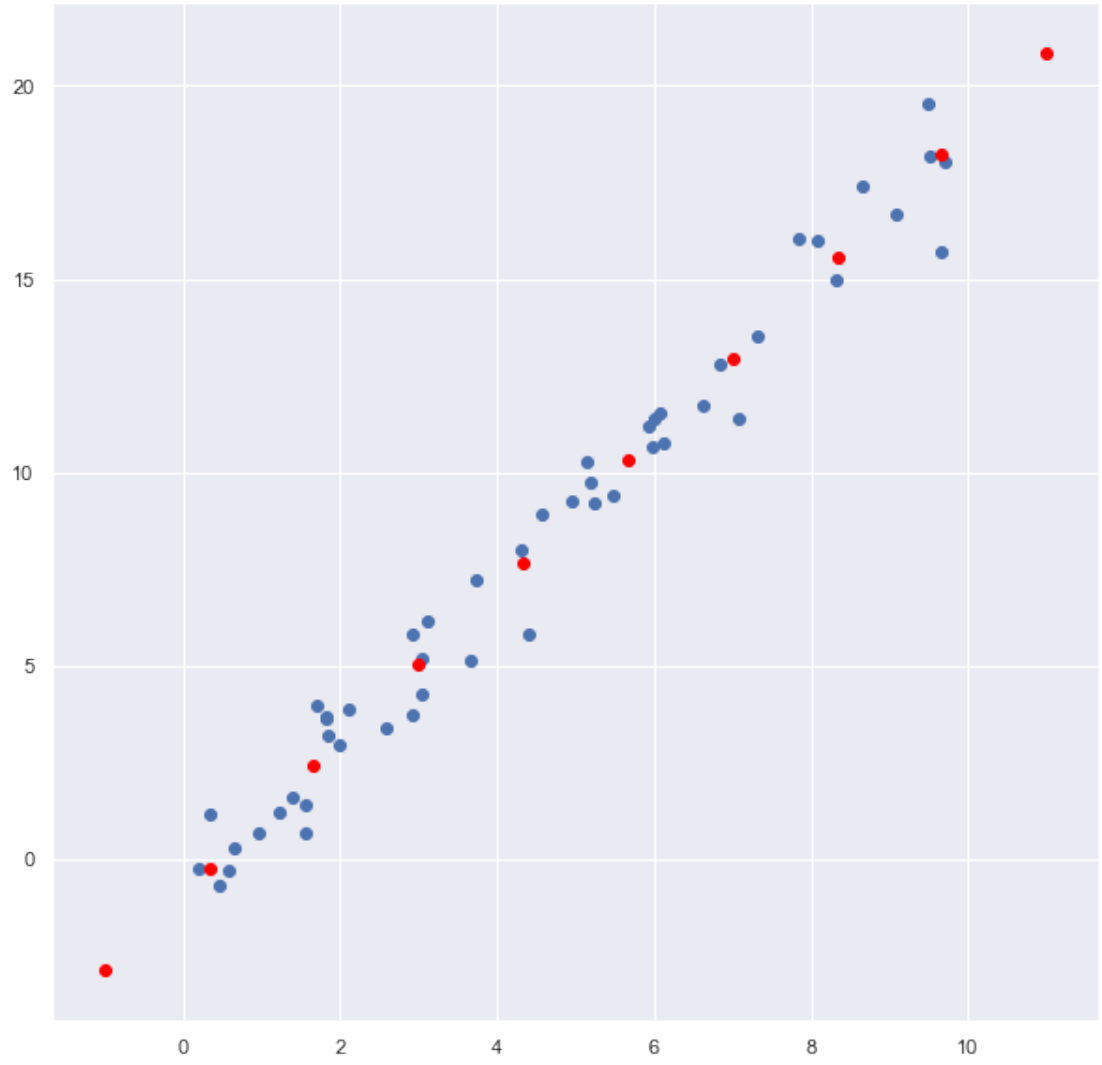
Erzeuge neue x-Daten:

[30]:
```
xfit = np.linspace(-1,11, num=10)      # Erzeugt 10 gleich äquidistante Punkte␣
↪zwischen -1 und 11
Xfit = xfit[:, np.newaxis]     # Wandle Array in Matrix um
```

Nun wird das Modell aktiv: Es berechnet für jeden dieser 10 neuen x-Werte "seine Vorhersage" eines y-Werts:

[31]:
```
yfit = model.predict(Xfit)
```

[34]:
```
plt.figure(figsize=(10,10))
plt.scatter(x,y)
plt.scatter(xfit, yfit, color='red');
#plt.plot(xfit, yfit);
```


20
15
10
5
0
0
2
4
6
8
10

[ ]: