regularisierung

August 21, 2021

1 Regularisierung mit Ridge

Ein Werkzeug gegen Overfitting ist die Regularisierung (Regularisation). Dafür gibt es mehrere Möglichkeiten, zwei davon wollen wir hier kurz ansprechen:

- L2-Regularisierung (Ridge)
- L1-Regularisierung (Lasso - least absolute shrinkage and selection operator)

Die Idee hinter beiden Methoden: Jeder Parameter wird mit einem Strafterm versehen, insbesondere wenn er einen hohen Einfluss auf die vorherzusagende Größe hat. Bei einer Linearen Regression also zum Beispiel die unterschiedlichen Koeffizienten $\beta_0, \beta_1, \beta_2, ..., \beta_n$.

Ziel bei der Regression ist es, den Fehler zu minimieren, also die Koeffizienten $\beta_0, \beta_1, \beta_2, ..., \beta_n$ so anzupassen, dass die Fehlerfunktion einen Minimalwert erreicht. Die Fehlerfunktion (Loss-Function) ist bei der OLS-Methode der Mean-Squared-Error, MSE:

$$MSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Bei der Regularisierung wird nun der Strafterm durch die Berücksichtigung der Koeffizienten ergänzt, sodass Koeffizienten mit einem großen Wert auf einen kleineren Wert reduziert werden. Insbesondere bei der Ridge-Regression werden große Koeffizienten stärker bestraft, da diese Werte quadriert einfließen. Bei der Lasso-Regularisierung wird nur der Betrag der Koeffizienten-Werte berücksichtigt.

Kosten-Funktion bei der Ridge-Regression:

$$MSE + \lambda \sum_{j=1}^{k} \beta_j^2$$

Kosten-Funktion bei der Lasso-Regression:

$$MSE + \lambda \sum_{j=1}^{k} |\beta_j|$$

λ ist ein "Lernparameter", der angibt, wie stark große Koeffizienten bestraft werden sollen. Große Werte für λ bewirken also eine stärke Reduzierung der β-Werte.

Der Unterschied der Lasso-Regularisierung im Vergleich zur Ridge-Variante ist, dass bei Lasso bestimmte Koeffizienten auch auf 0 gesetzt werden können, sodass dieses Argument überhaupt keinen Einfluss auf die zu prognostizierenden Werte hat.

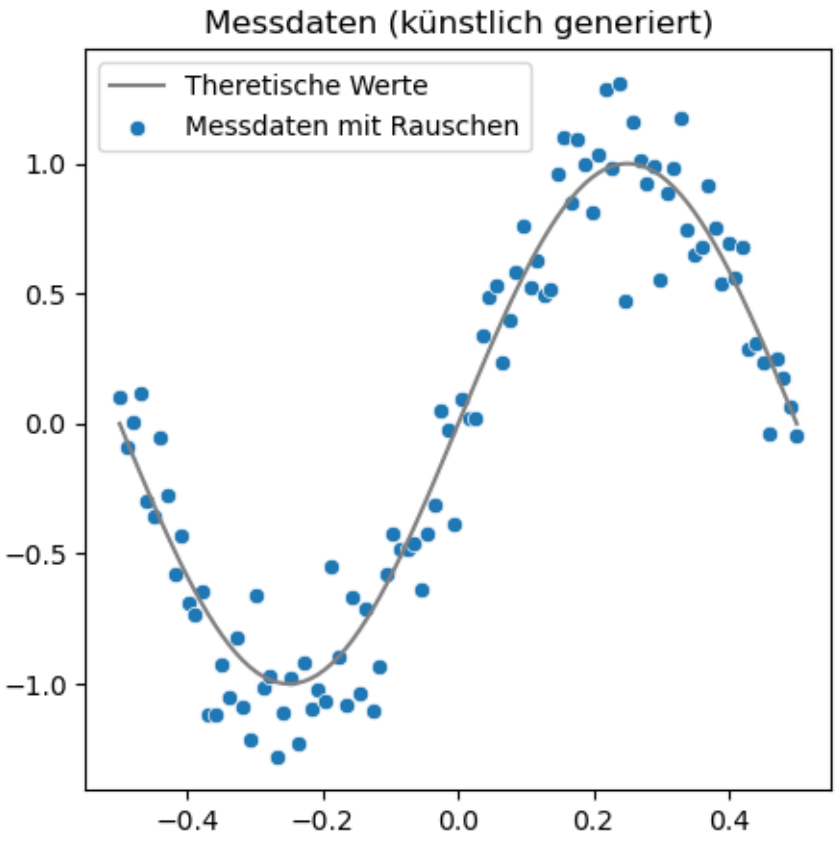
Im Beispiel generieren wir Beispieldaten und erstellen verschiedene Modelle. Da die Daten einer Sinus-Kurve entsprechen, versuchen wir es mit einer Polynomialen Regression mit verschiedenen

Graden. In den erstellen Diagrammen sehen wir, dass bei einem Polynoom 3. Grades ein recht gutes Modell zustande kommt. Darunter haben wir Underfitting - Das Modell ist zu allgemein.

Polynome höheren Graden führen zu Overfitting, was man insbesondere beim einem Grad >= 15 sehr gut erkennt: Das Modell passt sich zu gut an die Trainingsdaten an!

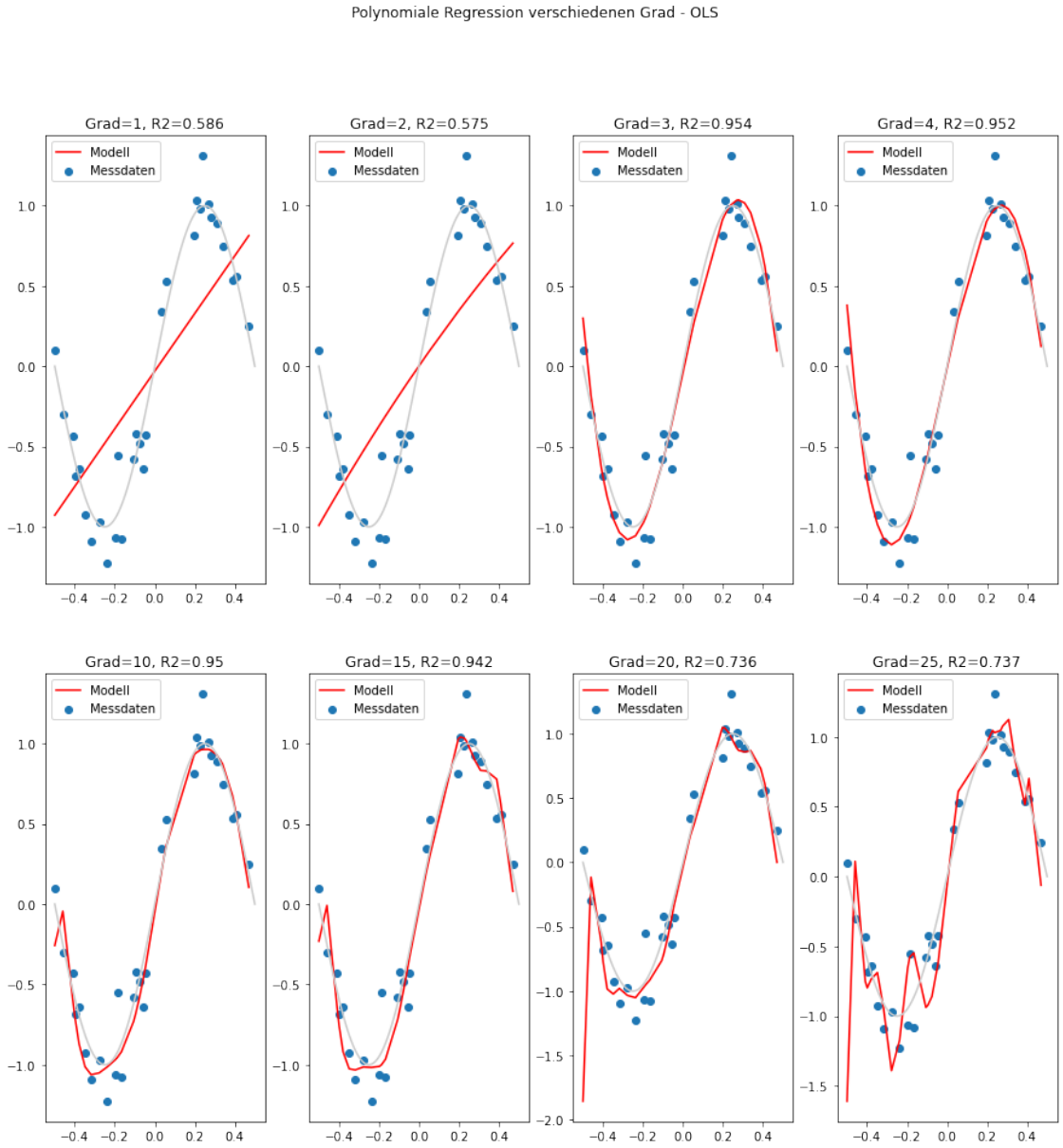
Ein optimaler Wert für λ kann zum Beispiel durch K-Fold-Cross-Validation ermittelt werden.

[1]:
```
# Funktion für Beispieldaten

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams["figure.figsize"] = (5,5)

def f(X):
    return np.sin(2 * np.pi * X)


def messdaten(X, n):
    np.random.seed(42)
    return f(X)+ np.random.normal(loc=0, scale=.2,size=n)

degrees = [1, 2, 3, 4, 10, 15, 20, 25]

n = 100
X = np.linspace(-0.5,.5,n)
y = messdaten(X,n)
sns.scatterplot(x=X, y=messdaten(X, n), label="Messdaten mit Rauschen")
sns.lineplot(x=X, y=f(X), label="Theretische Werte", color="gray")
plt.title("Messdaten (künstlich generiert)")
plt.legend()
plt.show()
```



[2]:
```
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
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,shuffle=True, random_state=42)

plt.rcParams["figure.figsize"] = (15,15)


sort_index = np.argsort(X_test)

X_test = X_test[sort_index]
y_test = y_test[sort_index]

for i, d in enumerate (degrees, start=1):
    pfeatures_train = PolynomialFeatures(degree=d).fit_transform(X_train.
 ↪reshape(-1,1))
```

```
    pfeatures_test  = PolynomialFeatures(degree=d).fit_transform(X_test.
 ↪reshape(-1,1))

    model1 = LinearRegression().fit(pfeatures_train, y_train.reshape(-1,1))
    pred1  = model1.predict(pfeatures_test)
    r2 = model1.score(pfeatures_test, y_test.reshape(-1,1))
    plt.subplot(2, 4, i)
    plt.scatter(X_test, y_test, label="Messdaten")
    plt.plot(X_test, pred1, color="red", label="Modell")
    plt.plot(X,f(X) , color="#CCCCCC")
    plt.title(f"Grad={d}, R2={np.round(r2,3)}")
    plt.legend();
    plt.suptitle("Polynomiale Regression verschiedenen Grad - OLS")

plt.show()
```

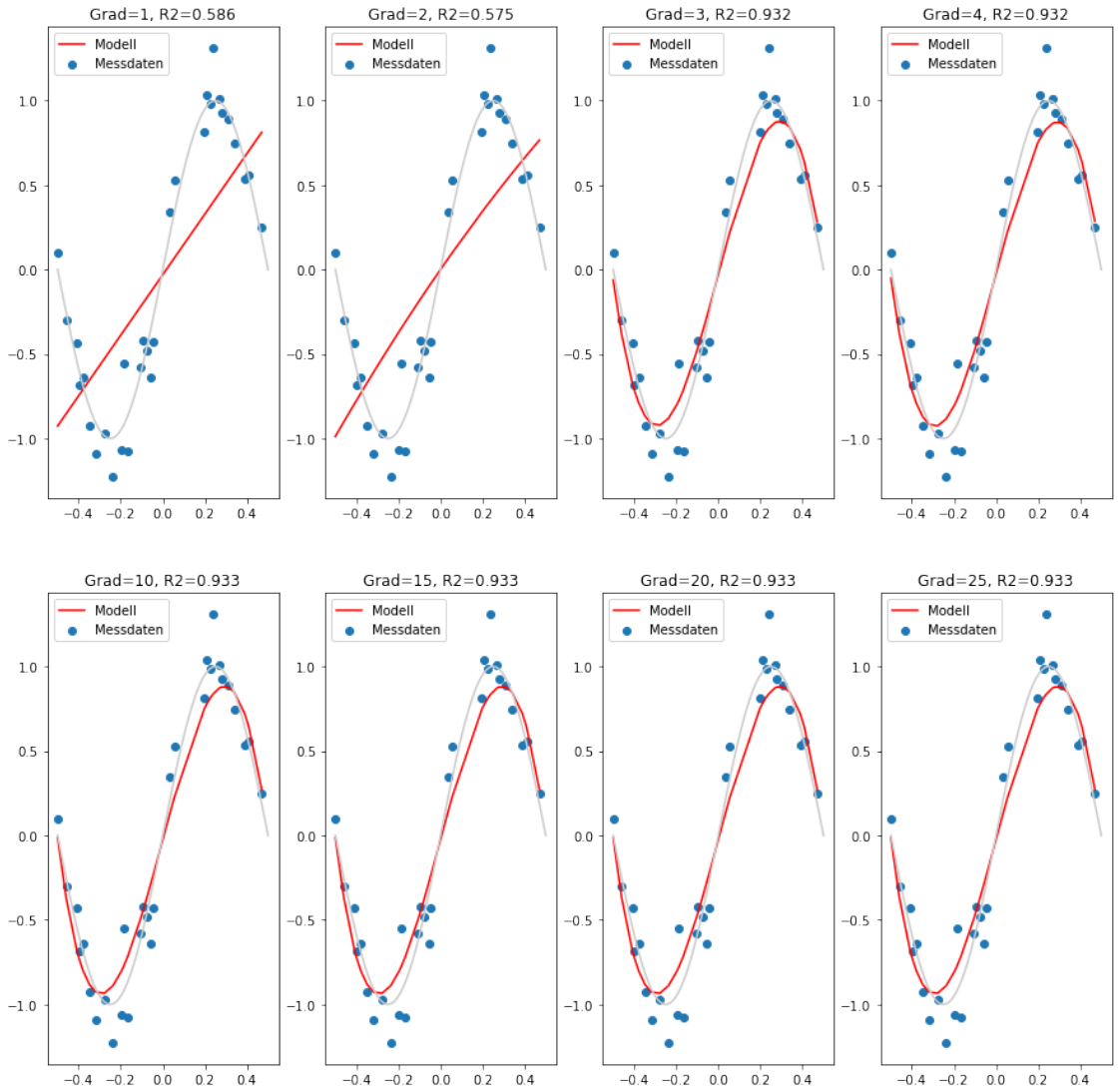


Nun wiederholen wir das Erstellen verschiedener Modelle, verwenden aber nicht OLS sondern die Ridge-Regularisierung. Als λ setzen wir einen Wert für 0,05 ein. In den Diagrammen sehen wir, dass selbst bei einem Polynom vom Grad 25 noch ken Overfitting zu sehen ist

1.1 Mit Ridge-Regularisierung

[3]:
```
%matplotlib inline
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
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,shuffle=True, random_state=42)
plt.rcParams["figure.figsize"] = (15,15)


sort_index = np.argsort(X_test)
X_test = X_test[sort_index]
y_test = y_test[sort_index]


for i, d in enumerate (degrees, start=1):
    pfeatures_train = PolynomialFeatures(degree=d,include_bias=False).
 ↪fit_transform(X_train.reshape(-1,1))
    pfeatures_test  = PolynomialFeatures(degree=d,include_bias=False).
 ↪fit_transform(X_test.reshape(-1,1))

    model2 = Ridge(alpha=0.01).fit(pfeatures_train, y_train[:, np.newaxis])
    pred2  = model2.predict(pfeatures_test)
    r2 = model2.score(pfeatures_test, y_test[:,np.newaxis])
    plt.subplot(2, 4, i)
    plt.scatter(X_test, y_test, label="Messdaten")
    plt.plot(X_test, pred2, color="red", label="Modell")
    plt.plot(X,f(X) , color="#CCCCCC")
    plt.title(f"Grad={d}, R2={np.round(r2,3)}")
    plt.legend()
    plt.suptitle("Polynomiale Regression verschiedenen Grad - Ridge")

plt.show()
```


Polynomiale Regression verschiedenen Grad - Ridge
Grad=1, R2=0.586
Grad=2, R2=0.575
Grad=3, R2=0.932
Grad=4, R2=0.932
Grad=10, R2=0.933
Grad=15, R2=0.933
Grad=20, R2=0.933
Grad=25, R2=0.933
Modell
Messdaten