

# **Pattern Recognition und Machine Learning**

Hochschule München

Sommersemester 2023

Prof. Dr.-Ing. Claudius Schnörr

---

## **Blatt 4**

**F. Freter, E. Kirchberger, S. Symhoven & J. Wustl**

# Aufgabe 1/2: Lineare Regression

## 0.0.1 Imports

```
[17]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches
```

## 0.1 Daten erzeugen

```
[3]: x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
                26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
y = np.array([2, 3, 4, 15, 4, 7, 8, 6, np.nan, np.nan, np.nan, 10, 14, 12, np.
nan, 1, 10, 7, 6, 5, 6, 4, 7, 3, 5, 7,
                np.nan, 8, np.nan, 12, 3, 5, 6, 8, np.nan, np.nan, np.nan, np.
nan, np.nan, 14, 2, 7, 5, 10, 12, 15, 8, np.nan, np.nan, 20])
```

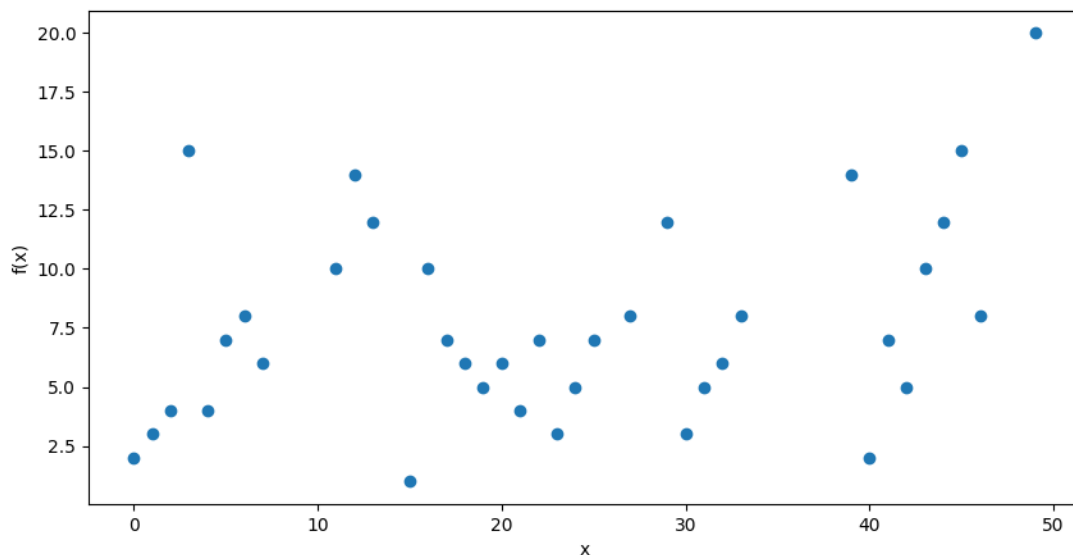
## 0.2 Plot

```
[4]: fig, ax = plt.subplots(figsize=(10, 5))

plt.scatter(x, y)

plt.xlabel('x')
plt.ylabel('f(x)')

plt.show()
```



### 0.3 Aufgabe 1

```
[5]: def polynomial_regression(x, y, degree):  
    """  
    Berechnet die Koeffizienten einer polynomialen Regression auf den gegebenen  
    ↪ Daten.  
  
    Parameter:  
    x (ndarray): Ein Array von X-Koordinaten.  
    y (ndarray): Ein Array von Y-Koordinaten.  
    degree (int): Der Grad des Polynoms.  
  
    Rückgabe:  
    a (ndarray): Ein Array von Koeffizienten des Polynoms.  
    mse (float): Die mittlere quadratische Abweichung zum Polynom.  
    """  
  
    # Entferne fehlende Werte aus den Daten  
    mask = ~np.isnan(y)  
    x_clean = x[mask]  
    y_clean = y[mask]  
  
    # Erzeuge die Matrix Y aus X und den Potenzen von X bis zum Polynomgrad  
    Y = np.ones_like(x_clean)  
    for i in range(1, degree+1):  
        Y = np.column_stack((Y, x_clean ** i))  
  
    # Berechne die Koeffizienten des Polynoms  
    Y_T = Y.T  
    a = np.linalg.inv(Y_T.dot(Y)).dot(Y_T).dot(y_clean)  
  
    # Berechne die mittlere quadratische Abweichung zum Polynom  
    y_pred = Y.dot(a)  
    mse = np.mean((y_clean - y_pred) ** 2)  
  
    # Gib die Koeffizienten und die mittlere quadratische Abweichung aus  
    return a, mse  
  
def build_poly_func(coeffs):  
    """  
    Erstellt eine Polynomfunktion auf Grundlage der Koeffizienten.  
  
    Parameter:  
    coeffs (tuple): Ein Tuple von Koeffizienten in aufsteigender Reihenfolge.  
  
    Rückgabe:
```

```

    func (function): Eine Funktion, die den Funktionswert des Polynoms für eine
    ↪ gegebene x-Koordinate berechnet.
    """

    return sum([coeff * x ** i for i, coeff in enumerate(coeffs)])

def generate_cost_function(coeff, mse):
    """
    Generiert eine Gütefunktion für die lineare Regression.

    Parameter:
    coeff (ndarray): Ein Array von Koeffizienten der Regressionsgerade.
    mse (float): Die mittlere quadratische Abweichung zur Regressionsgerade.

    Rückgabe:
    a1 (ndarray): Ein Array von Werten für den Koeffizienten a1.
    a0 (ndarray): Ein Array von Werten für den Koeffizienten a0.
    mse_paraboloid (ndarray): Ein Array von Werten für die mittlere
    ↪ quadratische Abweichung zur Regressionsgerade,
        berechnet als Funktion von a1 und a0.

    mse_paraboloid ist eine Approximation der Gütefunktion  $J(a_1, a_0)$  durch
    ↪ eine parabolische Funktion.

    Die Gütefunktion  $J(a_1, a_0)$  ist eine Funktion von  $a_1$  und  $a_0$ , die den
    ↪ mittleren quadratischen Abstand
        zwischen der Regressionsgeraden und den tatsächlichen Datenpunkten angibt.

    Die Idee, eine parabolische Funktion zur Approximation der Gütefunktion zu
    ↪ verwenden, beruht auf der Annahme,
        dass die Gütefunktion in der Nähe des Schätzwertes der Koeffizienten coeff
    ↪ ein Minimum aufweist. Dieses Minimum wird durch die
        parabolische Funktion approximiert.

    Um die parabolische Funktion zu erzeugen, wird zuerst der Schätzwert der
    ↪ Koeffizienten coeff als Schwerpunkt der Parabel verwendet.

    Dann wird um diesen Schwerpunkt eine Spanne von  $\pm 1$  erzeugt und in 100
    ↪ gleich große Schritte unterteilt, um die Werte für  $a_1$  und  $a_0$ 
        zu erhalten. Schließlich wird für jedes Paar von  $a_1$  und  $a_0$  der Wert von
    ↪  $J(a_1, a_0)$  durch die Formel
        
$$\text{mse\_paraboloid} = \text{mse}_0 + (a_1 - a_{1\_0}) ** 2 + (a_0 - a_{0\_0}) ** 2$$

        berechnet.
        Dabei ist  $\text{mse}_0$  die mittlere quadratische Abweichung zur Regressionsgerade
    ↪ an der Stelle coeff,  $a_{1\_0}$  und  $a_{0\_0}$  sind die Schätzwerte
        für  $a_1$  und  $a_0$ , und  $a_1$  und  $a_0$  sind die Werte, die um den Schwerpunkt der
    ↪ Parabel erzeugt wurden.
    """

```

```

a1_0 = coeff[0]
a0_0 = coeff[1]
mse_0 = mse

a1 = np.linspace(a1_0 - 1, a1_0 + 1, 100)
a0 = np.linspace(a0_0 - 1, a0_0 + 1, 100)
a1, a0 = np.meshgrid(a1, a0)

mse_paraboloid = mse_0 + (a1 - a1_0) ** 2 + (a0 - a0_0) ** 2

return a1, a0, mse_paraboloid

```

```

[6]: # Berechne die lineare Regression
coeff, mse = polynomial_regression(x=x, y=y, degree=1)
y_pred = build_poly_func(coeff)

# Erzeuge die parabolische Gütefunktion
a1, a0, mse_paraboloid = generate_cost_function(coeff, mse)

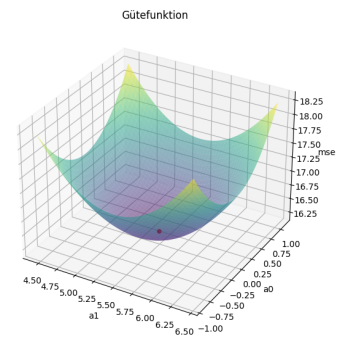
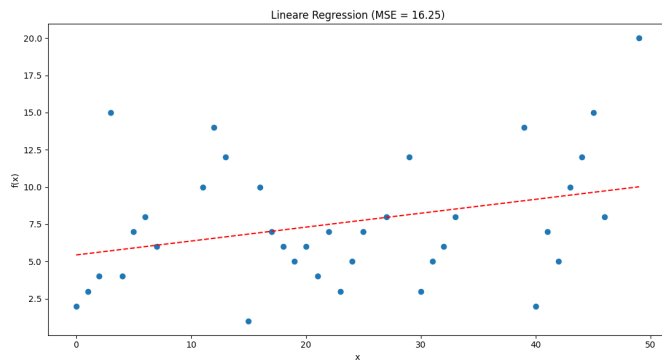
# Erzeuge den Subplot mit beiden Plots
fig = plt.figure(figsize=(20, 6))

# Plot der Daten und Regressionsgerade
ax1 = fig.add_subplot(121)
ax1.scatter(x, y)
ax1.plot(x, y_pred, 'r--')
ax1.set_xlabel('x')
ax1.set_ylabel('f(x)')
ax1.set_title('Lineare Regression (MSE = {:.2f})'.format(mse))

# 3D-Plot der Gütefunktion
ax2 = fig.add_subplot(122, projection='3d')
ax2.plot_surface(a1, a0, mse_paraboloid, cmap='viridis', alpha=0.5)
ax2.scatter(coeff[0], coeff[1], mse, color='red')
ax2.set_xlabel('a1')
ax2.set_ylabel('a0')
ax2.set_zlabel('mse')
ax2.set_title('Gütefunktion')

plt.tight_layout()
plt.show()

```



### 0.3.1 Kommentar zu den Ergebnissen

Aus den generierten Daten lässt sich kein klarer Funktionalzusammenhang erkennen. Sie scheinen zufällig und chaotisch zu sein. Daher ist es schwierig, ein lineares Modell zu finden, das die Beziehung zwischen  $x$  und  $y$  genau beschreibt.

Die durchgeführte lineare Regression liefert dennoch einen Funktionalzusammenhang und somit eine Regressionsgerade, die die Daten beschreibt. Allerdings sind die Datenpunkte sehr weit von der Regressionsgerade entfernt, was darauf hinweist, dass die lineare Regression nicht das beste Modell für diese Daten ist.

Der MSE ist relativ groß im Verhältnis zu dem Bereich der  $y$ -Werte, was ebenfalls darauf hindeutet, dass ein linearer Zusammenhang für diese Daten ungeeignet ist.

## 0.4 Aufgabe 2

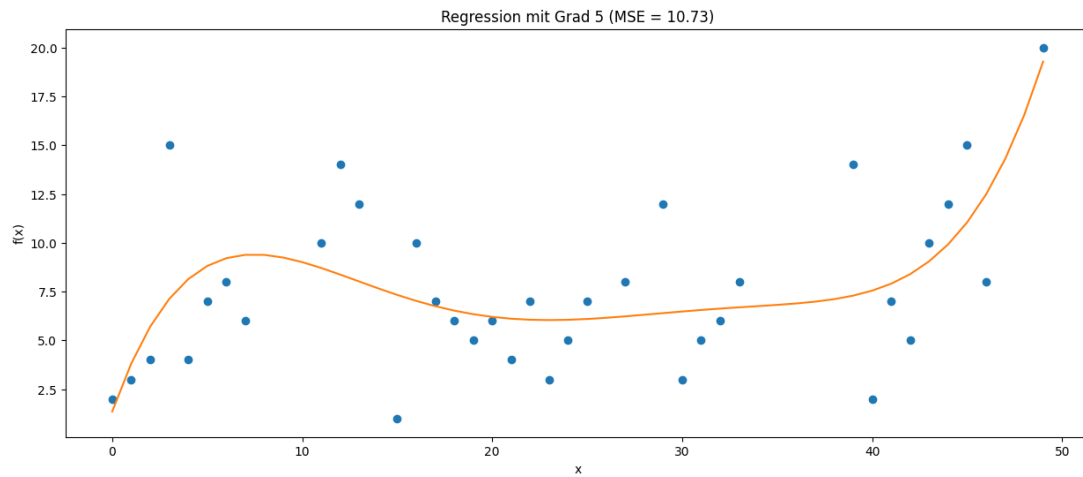
```
[7]: degree = 5

coef, mse = polynomial_regression(x=x, y=y, degree=degree)

poly_func = build_poly_func(coef)

plt.figure(figsize=(15, 6))
plt.plot(x, y, 'o', label='Daten')
plt.plot(x, poly_func, label='Polynom')
plt.xlabel('x')
plt.ylabel('f(x)')

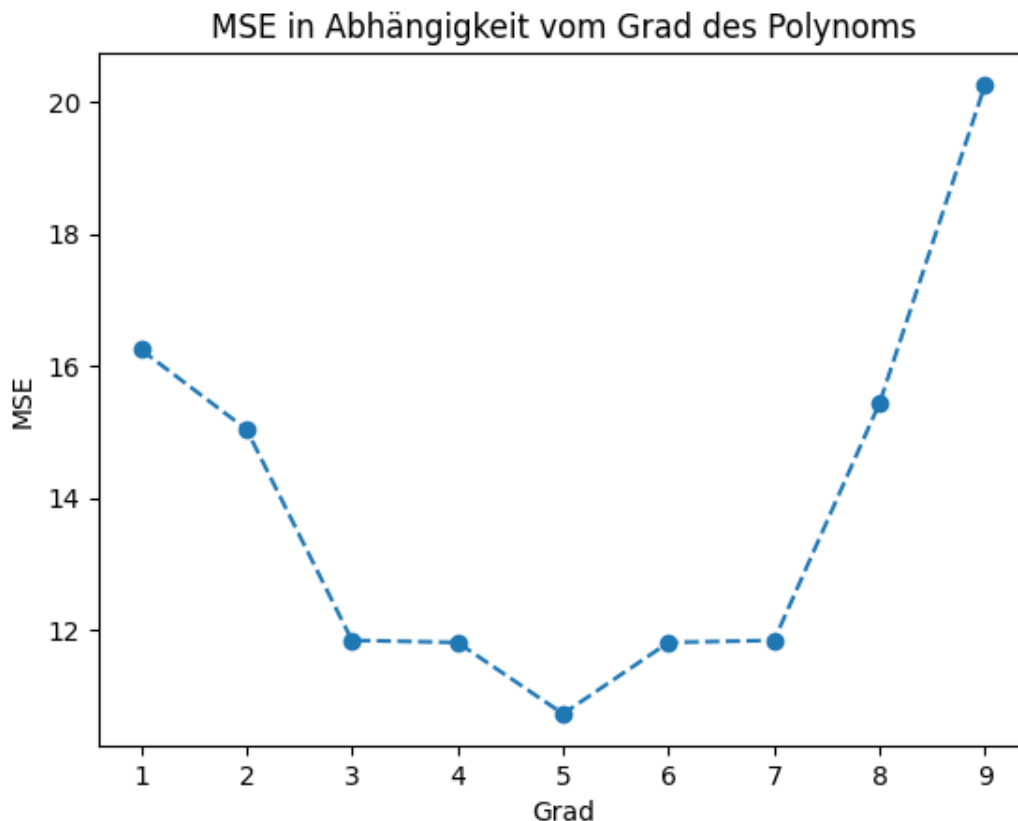
plt.title('Regression mit Grad {} (MSE = {:.2f})'.format(degree, mse))
plt.show()
```



```
[8]: degrees = np.arange(1, 10)
mSES = []

for degree in degrees:
    coef, mse = polynomial_regression(x, y, degree)
    mSES.append(mse)

plt.plot(degrees, mSES, '--o')
plt.xlabel('Grad')
plt.ylabel('MSE')
plt.title('MSE in Abhängigkeit vom Grad des Polynoms')
plt.show()
```



#### 0.4.1 Kommentar zu den Ergebnissen

Generell lässt sich sagen, dass mit zunehmendem Polynomgrad die Anpassungsgenauigkeit an die Datenpunkte steigt. Allerdings besteht auch die Gefahr von Overfitting, wenn das Polynom zu stark auf die Datenpunkte “überanpasst” wird. Es ist wichtig, den geeigneten Polynomgrad zu wählen, der eine gute Balance zwischen Anpassungsgenauigkeit und Modellkomplexität bietet.

Mit zunehmendem Grad des Polynoms wird eine höhere Flexibilität des Modells erreicht, um die Datenpunkte genauer anzupassen. Bei niedrigeren Polynomgraden, wie Grad 1 oder 2, passt sich das Polynom grob der Tendenz der Daten an, aber es gibt immer noch sichtbare Abweichungen.

Bei höheren Polynomgraden, wie Grad 3 oder 4, kann das Modell die Datenkurven genauer abbilden und passt sich den Datenpunkten besser an. Es werden detailliertere Muster erfasst und die Kurven folgen den Daten enger.

Allerdings gibt es einen Punkt, an dem die Zugabe höherer Polynomgrade keine wesentlichen Verbesserungen mehr bringt. Ab einem gewissen Grad (je nach Datenverteilung und Rauschniveau) führt die Erhöhung des Polynomgrads zu einer Überanpassung der Datenpunkte und zu einer komplexeren Modellstruktur. Dies kann zu einem erhöhten MSE führen und das Modell wird empfindlich gegenüber Rauschen und Ausreißern.

In unserem Fall scheint der Grad 5 das höchste Niveau zu sein, bei dem die Polynomregression



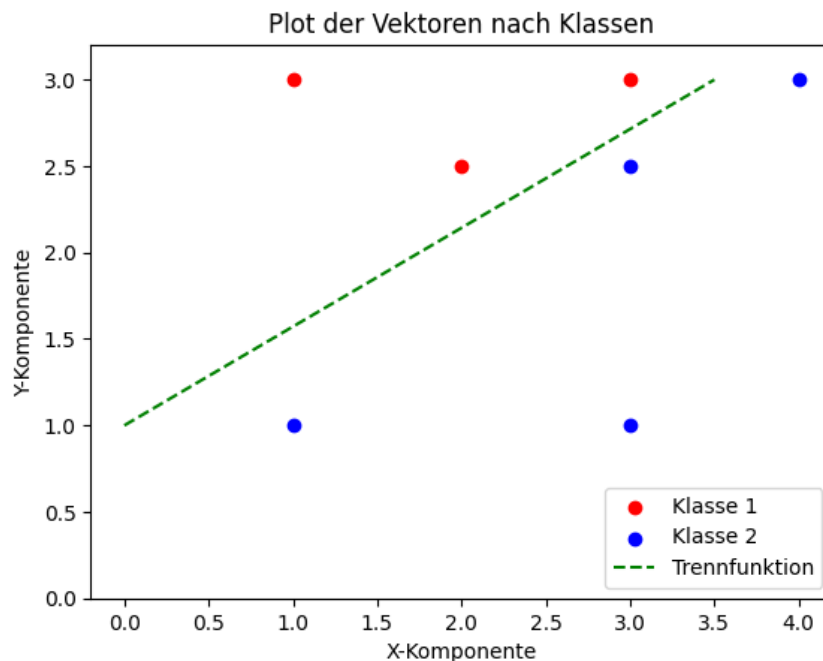
eine sichtbare Verbesserung bringt. Eine weitere Erhöhung des Polynomgrads führt zu minimalen oder keinen Veränderungen an der Regressionskurve. Es ist wichtig, den geeigneten Polynomgrad zu wählen, der die Datenstruktur angemessen erfasst, ohne das Modell zu komplex zu machen.

## Aufgabe 3: Support Vektor Maschinen

### SVM-Klassifikation:

1. Der Vektor  $y$  repräsentiert die Klassenzuordnungen der Trainingsdaten in einem SVM-Klassifikator. In diesem Fall werden Muster der Klasse 1 mit dem Wert 1 gekennzeichnet und Muster der Klasse 2 mit dem Wert -1 gekennzeichnet. Die Reihenfolge der Elemente im Vektor  $y$  entspricht der Reihenfolge der entsprechenden Muster in den Datenvektoren  $x_1$  bis  $x_7$ .

$$y = (-1 \quad 1 \quad 1 \quad -1 \quad 1 \quad -1 \quad -1)^T$$



2. Ja, es handelt sich dabei um eine linear trennbare Problemstellung. Diese kann über zwei Punkte  $a = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$  und  $b = \begin{pmatrix} 3.5 \\ 3 \end{pmatrix}$  definiert werden und maximiert den Rand zwischen den beiden Klassen.
3. Um den Gewichtungsvektor  $w$  zu berechnen, verwenden wir normalerweise den Ansatz der Support Vector Machine und die Methode der Lagrange-Multiplikatoren. Der Gewichtungsvektor  $w$  wird während des Trainingsprozesses der SVM bestimmt und hängt von den Datenpunkten und den Lagrange-Multiplikatoren ab.

In unserer zeichnerischen Lösung basierend auf den Punkten  $a$  und  $b$ , haben wir bereits eine Trennfunktion definiert. Die Koeffizienten der Geradengleichung, die die beiden Punkte  $a$  und  $b$  verbindet, bestimmen die Parameter der linearen Trennfunktion:

Der Schwellenwert  $b$  ist der y-Achsenabschnitt und somit die zweite Komponente von dem Punkt  $a$ .

$$b = 1$$

Der Gewichtungsvektor  $w$  der linearen Trennfunktion wird berechnet als:

$$w = b - a = \begin{pmatrix} 3.5 \\ 3 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3.5 \\ 2 \end{pmatrix}$$

Wir setzen den Vektor  $w$  auf die Länge 1 (Normalenvektor), indem wir diesen durch seine Norm  $\|w\|$  teilen:

$$\|w\| = \sqrt{3.5^2 + 2^2} = \sqrt{16.25}$$

Daraus folgt:

$$w = \frac{\begin{pmatrix} 3.5 & 2 \end{pmatrix}^T}{\|w\|} \approx \begin{pmatrix} 0.868 \\ 0.496 \end{pmatrix}$$

4. Daraus ergeben sich die Support-Vektoren  $x_2$ ,  $x_5$ ,  $x_6$  und  $x_7$ .

5.

$$\begin{aligned} \langle w, x \rangle &= \sum_{x_i \in \text{SVs}} \alpha_i y_i K(x, x_i) \\ &= 8.23 \cdot K(x, x_2) + 0.95 \cdot K(x, x_5) - 8.23 \cdot K(x, x_6) - 0.95 \cdot K(x, x_7) \\ &= 8.23 \cdot \left( (x_1 \ x_2) \cdot \begin{pmatrix} 3 \\ 3 \end{pmatrix} + 1 \right) + 0.95 \cdot \left( (x_1 \ x_2) \cdot \begin{pmatrix} 2 \\ 2.5 \end{pmatrix} + 1 \right) \\ &\quad - 8.23 \cdot \left( (x_1 \ x_2) \cdot \begin{pmatrix} 3 \\ 2.5 \end{pmatrix} + 1 \right) - 0.95 \cdot \left( (x_1 \ x_2) \cdot \begin{pmatrix} 4 \\ 3 \end{pmatrix} + 1 \right) \\ &= 8.23 \cdot (3x_1 + 3x_2 + 1) + 0.95 \cdot (2x_1 + 2.5x_2 + 1) \\ &\quad - 8.23 \cdot (3x_1 + 2.5x_2 + 1) - 0.95 \cdot (4x_1 + 3x_2 + 1) \\ &= -1.9x_1 + 3.64x_2 \end{aligned}$$

$$\begin{aligned} w^* &= \sum_{i=1}^7 \alpha_i^* y_i x_i \\ &= (8.23 \cdot 1 \cdot x_2) + (0.95 \cdot 1 \cdot x_5) + (8.23 \cdot (-1) \cdot x_6) + (0.95 \cdot (-1) \cdot x_7) \\ &= (8.23 \cdot 1 \cdot \begin{pmatrix} 3 \\ 3 \end{pmatrix}) + (0.95 \cdot 1 \cdot \begin{pmatrix} 2 \\ 2.5 \end{pmatrix}) + (8.23 \cdot (-1) \cdot \begin{pmatrix} 3 \\ 2.5 \end{pmatrix}) + (0.95 \cdot (-1) \cdot \begin{pmatrix} 4 \\ 3 \end{pmatrix}) \\ &= 8.23 \cdot \begin{pmatrix} 3 \\ 3 \end{pmatrix} + 0.95 \cdot \begin{pmatrix} 2 \\ 2.5 \end{pmatrix} - 8.23 \cdot \begin{pmatrix} 3 \\ 2.5 \end{pmatrix} - 0.95 \cdot \begin{pmatrix} 4 \\ 3 \end{pmatrix} \\ &= \begin{pmatrix} -1.9 \\ 3.64 \end{pmatrix} \end{aligned}$$

Vergleich von  $w^*$  mit  $w$  aus Teilaufgabe 3:

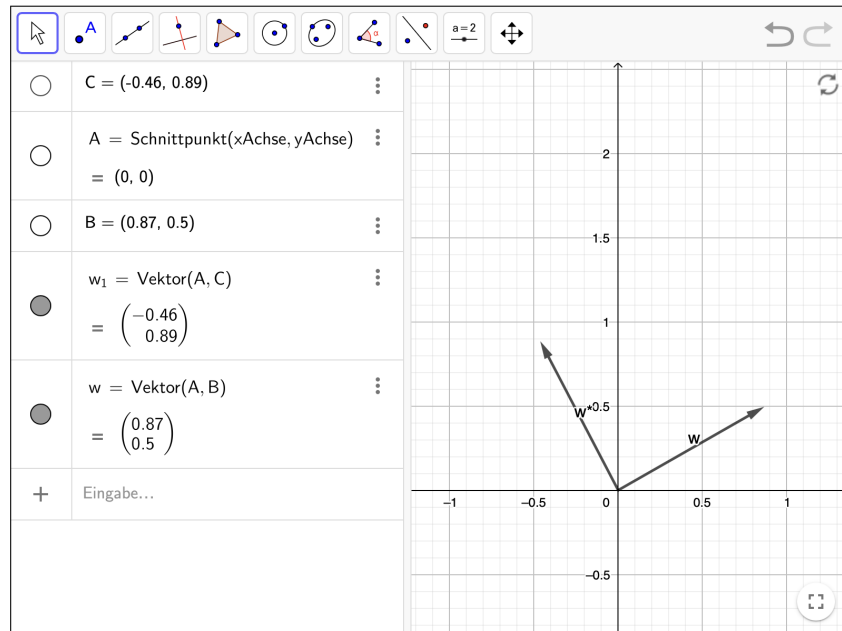
Wir teilen  $w^*$  ebenfalls durch seine Norm und erhalten:

$$\|w\| = \sqrt{-1.9^2 + 3.64^2}$$

und erhalten:

$$w^* = \frac{(-1.9 \quad 3.64)^T}{\|w\|} \approx \begin{pmatrix} -0.463 \\ 0.886 \end{pmatrix}$$

Es fällt auf, dass  $w^*$  und  $w$  orthogonal zu einander sind:



Dies entspricht unserer Erwartung, da es sich bei  $w$  um die Trennlinie bzw. Trennhyperebene handelt wohingegen  $w^*$  den resultierenden Gewichtsvektor für die optimale Trennung der Klassen darstellt.

#### 6. Aus der Komplementaritätsbedingung

$$\alpha_i[(\langle w, x_i \rangle + b)y_i - 1] = 0$$

folgt

$$b = \frac{1}{y_i} - \langle w, x_i \rangle$$

$i = 2$ :

$$\begin{aligned} b_2 &= \frac{1}{1} - (-0.463 \cdot 3 + 0.886 \cdot 3) \\ b_2 &= -0.269 \end{aligned}$$

$i = 5$ :

$$\begin{aligned} b_5 &= \frac{1}{1} - (-0.463 \cdot 2 + 0.886 \cdot 2.5) \\ b_5 &= -0.289 \end{aligned}$$

$i = 6$ :

$$b_6 = \frac{1}{-1} - (-0.463 \cdot 3 + 0.886 \cdot 2.5)$$

$$b_6 = -1.826$$

$i = 7$ :

$$b_7 = \frac{1}{-1} - (-0.463 \cdot 4 + 0.886 \cdot 3)$$

$$b_7 = -1.806$$

Gemittelt über alle Support-Vektoren ist  $b$ :

$$b = \frac{1}{4} \left( -0.262 + (-0.289) + (-1.826) + (-1.806) \right) = -1.04575$$

7.

$$\begin{aligned} f(x_s) &= \text{sign}(\langle w, x_s \rangle + b) \\ &= \text{sign}(-0.463 \cdot 2 + 0.886 \cdot 2 - 1.04575) \\ &= \text{sign}(-0.19975) \\ &= -1 \end{aligned}$$

Der Punkt  $x_s$  ist somit als Klasse 2 zu klassifizieren.

## SVM-Regression:

```
[73]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.svm import SVR
```

### 0.0.1 SVM-Regression

```
[74]: df = pd.read_excel('zinsen.xlsx', sheet_name="GTDEM10Y Govt", header=5,
    parse_dates=True, index_col='Dates')
```

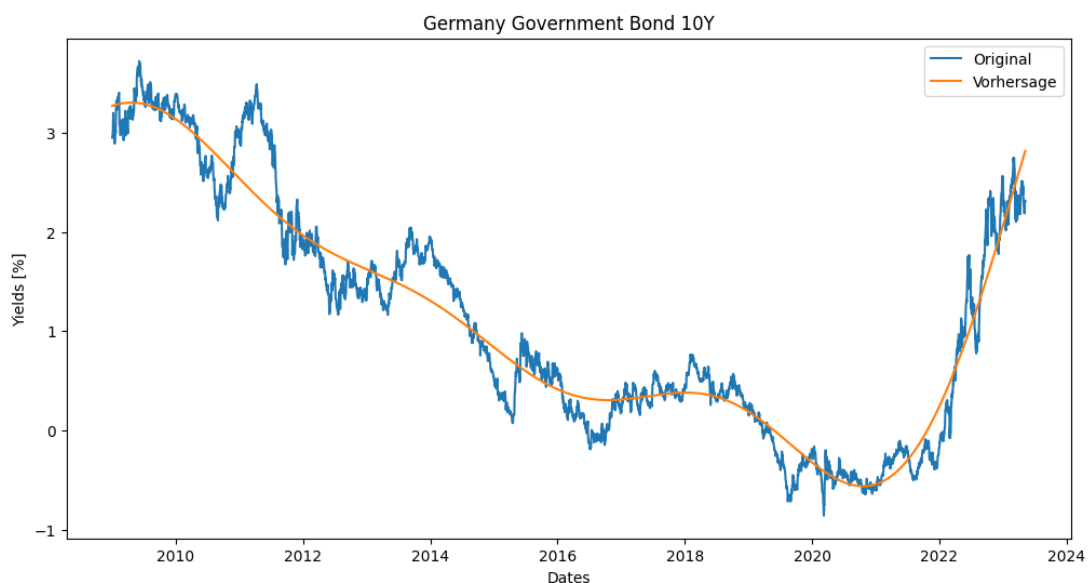
```
[75]: df['Numeric_Date'] = pd.to_numeric(df.index)

x = df['Numeric_Date'].values.reshape(-1, 1)
y = df['PX_LAST'].values

svm_reg = SVR(kernel='rbf', C=1.0, gamma='scale', epsilon=0.1)
svm_reg.fit(x, y)

x_pred = df['Numeric_Date'].values.reshape(-1, 1)
y_pred = svm_reg.predict(x_pred)

plt.figure(figsize=(12, 6))
plt.plot(df.index, y, label='Original')
plt.plot(df.index, y_pred, label='Vorhersage')
plt.xlabel('Dates')
plt.ylabel('Yields [%]')
plt.title('Germany Government Bond 10Y')
plt.legend()
plt.show()
```



Parameter  $\mathcal{C}$ :

Der Parameter  $\mathcal{C}$  steuert die Straffunktion in der SVM-Regression. Ein kleinerer Wert von  $\mathcal{C}$  führt zu einer größeren Strafe für Datenpunkte, die gegen den Rand der Entscheidungsgrenze verstoßen. Ein größerer Wert von  $\mathcal{C}$  erlaubt eine größere Flexibilität der Entscheidungsgrenze, um mehr Datenpunkte zu umfassen.

Parameter  $\sigma$ :

Der Parameter  $\sigma$  steuert die Reichweite des Einflusses jedes einzelnen Trainingspunkts. Ein kleinerer Wert von  $\sigma$  führt zu einer größeren Reichweite und einer glatteren Entscheidungsgrenze. Ein größerer Wert von  $\sigma$  führt zu einer geringeren Reichweite und einer stärkeren Anpassung an einzelne Trainingspunkte.

Parameter  $\varepsilon$ :

Der Parameter  $\varepsilon$  bestimmt die Breite des Epsilon-Schlauchs, der die Toleranz für die Vorhersageabweichung von den Trainingsdaten darstellt. Ein kleinerer Wert von  $\varepsilon$  erlaubt weniger Toleranz und erfordert eine genauere Anpassung an die Trainingsdaten. Ein größerer Wert von  $\varepsilon$  erlaubt eine größere Toleranz und ermöglicht eine gewisse Flexibilität in der Vorhersage.