1 Einführung

In dieser Übung wenden wir die logistische Regression, auf zwei unterschiedliche simulierte Datensätze an: Einstellungstest und Qualitätskontrolle. Beide Datensätze haben jeweils Original-Eingangvariablen x_1 und x_2 und eine Ausgangssvariable y, welche nur die Werte $y = 1$ und $y = 0$ annehmen kann. Weil y nur zwei Werte annehmen kann, sprechen wir auch von binärer Klassifikation. Das ist die wichtigste Anwendung der logistischen Regression.

Da für beide Probleme eine Gerade als Entscheidungsschwelle nicht ausreicht, müssen wir die Original-Eingangsvariablen auf Polynommerkmale abbilden, um ein gutes Modell zu erhalten. Beim Datensatz Einstellungstest reicht es, wenn wir polynomiale Merkmale bis zur zweiten Ordnung verwenden. Beim Datensatz Qulitätskontrolle brauchen wir Polynommerkmale deutlich höherer Ordnung. Deshalb ist es hier unbedingt erforderlich Maßnahmen gegen Overfitting zu ergreifen. Dazu setzen wir die L2-Norm-Regularisierung ein.

2 Logistische Regression

Durchführung

Aufgabe D1: Logistische Regression mit den Einstellungstest-Daten

In dieser Aufgabe wollen wir die Parameter $\boldsymbol{\beta}$ der Hypothese für die Einstellungstest-Daten[a] sowohl mit dem Gradientenverfahren als auch mit dem BFGS-Verfahren ermitteln. Hintergrund der Daten ist folgender: Die Personalabteilung einer großen Firma hat viele Daten ihrer bisherigen Bewerber gesammelt. Im Einstellungsverfahren hat sie für jeden Bewerber zwei Einstellungstests durchgeführt. Die Anzahl der Punkte, die ein Bewerber im 1. Test erzielt hat, werden in der Variable x_1 gespeichert. Die Anzahlt der Punkte des zweiten Tests werden in x_2 gespeichert. Außerdem wurde bisher mit jedem Bewerber ein Vorstellungsgespräch durchgeführt, auf dessen Basis dann die Entscheidung getroffen wurde, ober der Bewerber eingestellt wird ($y = 1$) oder nicht ($y = 0$). In Zukunft will die Firma auf das Einstellungsgespräch verzichten. Deshalb ist es unsere Aufgabe, mit Hilfe der bisher gesammelten Daten einen Klassifikationsalgorithmus zu implementieren, der aufgrund der beiden Testergebnisse x_1 und x_2 eine Entscheidung treffen kann.

a) Laden Sie die Daten aus dem File `D1_Training.mat`. In diesem File ist die Matrix `X` mit den Beobachtungen der Eingangsvariablen x_1 und x_2 sowie der Vektor `y` mit den Beobachtungen der Ausgangsvariable y enthalten.

b) Stellen Sie die Daten als Streudiagramme (scatter plots) dar. Verwenden Sie für die horizontale Achse die Variable x_1 und für die vertikale Achse die Variable x_2. Den Wert der Ausgangsvariable y können Sie dadurch darstellen, dass Sie unterschiedliche Farben (und/oder unterschiedliche Marker) für die jeweiligen Datenpunkte verwenden. Um alle Werte mit $y = 1$ und alle Werte mit $y = 0$ zu finden, können Sie den **`find`**-Befehl benutzen. Alternativ können Sie auch die bereits vorgegebene Funktion `zeichneStreudiagramm` benutzen. Das Streudiagramm sollte in etwa so aussehen wie Abbildung 1.

c) Implementieren Sie eine MATLAB-Funktion `sigmoid`, welche die logistische Funktion implementiert. Der Funktionsprototyp soll folgendermaßen aussehen:

```
g = sigmoid(z).
```

Stellen die Sigmoidfunktion im Bereich $-10 \leq z \leq 10$ graphisch dar. Es sollte sich dabei ein Kurve wie in Abbildung 2 ergeben.

d) Implementieren Sie jetzt eine MATLAB-Funktion, welche bei gegebener Matrix $\boldsymbol{X}$, gegebenem Vektor $\boldsymbol{y}$ und gegebenem Parametervektor $\boldsymbol{\beta}$ die Kostenfunktion $C(\boldsymbol{\beta})$ und deren Gradienten $\nabla C(\boldsymbol{\beta})$

berechnet. Der Funktionsaufruf soll folgendermaßen aussehen:

```
[C, gC] = Kostenfunktion(beta, X, y);
```

wobei `C` der Wert der Kostenfunktion und `gC` ihr Gradient ist. **`beta`** ist der aktuelle Parametervektor, `X` ist die Datenmatrix mit den Beobachtungen der Eingangsvariablen und `y` ist der Vektor mit den Beobachtungen der Ausgangsvariable. Für die Implementierung der Kostenfunktion können Sie natürlich die Funktion `sigmoid` verwenden.

e) Testen Sie jetzt die Kostenfunktion mit den gegebenen Trainingsdaten. Fügen Sie dazu in die Matrix `X` noch eine Zeile mit lauter Einsen ein (Hilfsvariable $x_0 = 1$). Wählen Sie als Parametervektor $\boldsymbol{\beta} = \mathbf{0}$. Dann sollte die die Kostenfunktion folgende Werte liefern:

```
C =
    0.6931
gC =
   -0.1600
  -15.2657
  -15.9586
```

f) Bestimmen Sie jetzt den Parametervektor $\hat{\boldsymbol{\beta}}$ mit Hilfe des Gradientenverfahrens und der Funktion aus der voherigen Teilaufgabe. Ermitteln Sie eine geeignete Schrittweite α durch Probieren. Dazu können Sie folgendermaßen vorgehen. Wählen Sie einen beliebigen Wert für die Schrittweite (z.B. $\alpha = 0.01$), führen Sie die Iterationen des Gradientenverfahren aus und speichern Sie für jede Iteration den Wert der Kostenfunktion, der mit dem aktuellen Wert von $\boldsymbol{\beta}$ erreicht wurde. Zeichnen Sie dann den Verlauf der Kostenfunktion über dem Iterationsindex. Wenn sich hier eine oszillierende oder ansteigende Kurve ergibt, ist die Schrittweite zu groß. Wenn sich eine langsam abfallende Kurve ergibt, ist die Schrittweite zu klein. Wenn sich eine schnell abfallende Kurve ergibt, ist die Schrittweite gerade richtig. Zeichnen Sie dann die Entscheidungsschwelle, die sich mit dem Parametervektor $\hat{\boldsymbol{\beta}}$ ergibt, der sich nach Konvergenz des Verfahren eingestellt hat. Dazu können Sie die Funktion `zeichneEntscheidungsschwelle` verwenden. Die Entscheidungsschwelle lässt sich besser interpretieren, wenn Sie auch noch mit der Funktion `zeichneStreudiagramm` die Datenpunkte mit einzeichnen. Passt die Entscheidungsschwelle zu den Datenpunkten? Wie lassen sich etwaige Abweichungen zwischen der Schwelle und den Daten erklären? Abbildung 3 zeigt den Verlauf der Kostenfunktion über dem Iterationsindex für zwei beispielhafte Schrittweiten und Abbildung 4 zeigt die Entscheidungsschwelle für $\alpha = 0.001$.

g) Führen Sie jetzt die Schritte aus der vorhergehenden Aufgabe nochmals durch. Skalieren Sie allerdings vor Anwendung des Gradientenverfahrens alle beteiligten Eingangsvariablen (also x_1 und x_2) so, wie wir das für die lineare Regression besprochen haben. Wenn wir das Gradientenverfahren auf die skalierten Daten anwenden, erhalten wir den Parametervektor $\tilde{\boldsymbol{\beta}}$. Um daraus den Parametervektor $\hat{\boldsymbol{\beta}}$ für das unskalierte Problem zu erhalten, können Sie die gleichen Formeln anwenden wie bei der linearen Regression. Stellen Sie wieder die Kostenfunktion über dem Iterationsindex sowie die Entscheidungsschwelle, die sich bei Konvergenz des Verfahrens ergibt dar. Passt die Entscheidungsschwelle jetzt besser zu den Daten? Gibt es weiteres Verbesserungspotential? Abbildung 5 zeigt die Entscheidungsschwelle für $\alpha = 0.5$.

h) Überprüfen Sie jetzt mit Hilfe der Validierungsdaten aus dem File `D1_Validierung.mat`, wie gut der Klassifikationsalgorithmus für Daten funktioniert, die nicht zum Training verwendet wurden. Ermitteln Sie für die Validierungsdaten die Erkennungsrate

$$ER = \frac{\text{Anzahl der richtig erkannten Beobachtungen}}{\text{Anzahl der gesamten Beobachtungen}}$$

Da wir im Folgenden noch öfter eine Klassifikation mit Validierungsdaten durchführen wollen, lohnt es sich für diese Teilaufgabe eine Funktion zu schreiben, deren Funktionskopf beispielsweise wie folgt aussehen könnte:

```
ER = Klassifikation(X, y, beta)
```

Wenn Sie den Parameterverktor $\hat{\boldsymbol{\beta}}$ aus der vorherigen Teilaufgabe verwenden, sollte sich für die Validierungsdaten eine Erkennungsrate von $ER = 0.90$ (also 90 %) ergeben.

i) Als Alternative zum Gradientenverfahren wollen wir jetzt den Parametervektor mit Hilfe des sogenannten BFGS-Verfahrens ermitteln. Das BFGS-Verfahren ist ein sogenanntes Quasi-Newton-Verfahren, welches sich in der Praxis sehr gut bewährt hat. Für dieses Verfahren gibt es in der MATLAB-Optimization-Toolbox bereits die Funktion `fminunc`. Der folgende Ausschnitt aus einem MATLAB-Code, der im File `Aufruf_fminunc.m` zur Verfügung steht, zeigt, wie man `fminunc` benutzt.

```
%Initialisierung
n=2;
initial_beta = zeros(n+1,1);

%  Setzen von Optionen fuer fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);

%  Ausfuehren von fminunc um den Optimalwert fuer beta zu erhalten
[beta, cost] = ...
  fminunc(@(t)(Kostenfunktion(t, X, y)), initial_beta, options);
```

Bestimmen Sie auch die Erkennungsrate, die sich mit den Validierungsdaten für $\hat{\boldsymbol{\beta}}$ ergibt. Die Ausgabe durch `fminunc` sollte in etwa wie folgt aussehen.

```
fminunc stopped because the final change in function value relative to its initial value
is less than the default value of the function tolerance.
<stopping criteria details>
Kostenfunktion beim Optimalwert: 0.231969
beta:
 -9.268987
 0.108780
 0.110020
```

Der Wert für $\hat{\boldsymbol{\beta}}$, den wir mit `fminunc` erhalten, ist sehr ähnlich (aber nicht exakt gleich) wie der Wert für $\hat{\boldsymbol{\beta}}$, den wir mit dem Gradientenverfahren mit skalierten Daten erhalten haben. Die Erkennungsrate beträgt hier $ER = 0.899$, also fast identisch mit der Erkennungsrate aus der vorherigen Teilaufgabe.

j) Jetzt wollen wir unseren Klassifikator auf Polynommerkmale zweiten Grades erweitern. Dazu müssen wir zunächst die Polynommerkmale, die von den Originalmerkmalen abgeleitet werden definieren. Dafür steht die bereits vorgegebene Funktion `PolynomMerkmale` zur Verfügung. Wenn wir diese Funktion wie folgt aufrufen

```
Xq = PolynomMerkmale(X(:,2),X(:,3),2);
```

wird aus der 2. und 3. Spalte der bisherigen Datenmatrix eine neue Datenmatrix `Xq` gebildet, welche neben den Originalmerkmalen x_1 und x_2 (und natürlich $x_0 = 1$) auch noch die abgeleiteten Merkmale $x_3 = (x_1)^2$, $x_4 = (x_2)^2$ und $x_5 = x_1 \cdot x_2$ enthält. Mit der Matrix `Xq` und dem Vektor `y` können wir jetzt wie in der vorherigen Aufgabe mit Hifle von `fminunc` den optimalen Parametervektor $\hat{\boldsymbol{\beta}}$ ermitteln. Stellen Sie jetzt mit den Funktionen `zeichneStreudiagramm` und `zeichneEntscheidungsschwelle` die Entscheidungsschwelle und die Datenpunkte in einem gemeinsamen Koordinatensystem dar. Passt die Entscheidungsschwelle mit den Polynommerkmalen besser als die Gerade?

k) Führen Sie mit dem Polynommodell der vorherigen Aufgabe eine Klassifikation der Validierungsdaten durch und ermitteln Sie die zugehörige Erkennungsrate. Hat sich die Erkennungsrate gegenüber den Originalmerkmalen verbessert? Die Erkennungsrate mit den Polynommerkmalen sollte $ER = 0.9465$ betragen.

[a] Diese Daten sind simuliert.

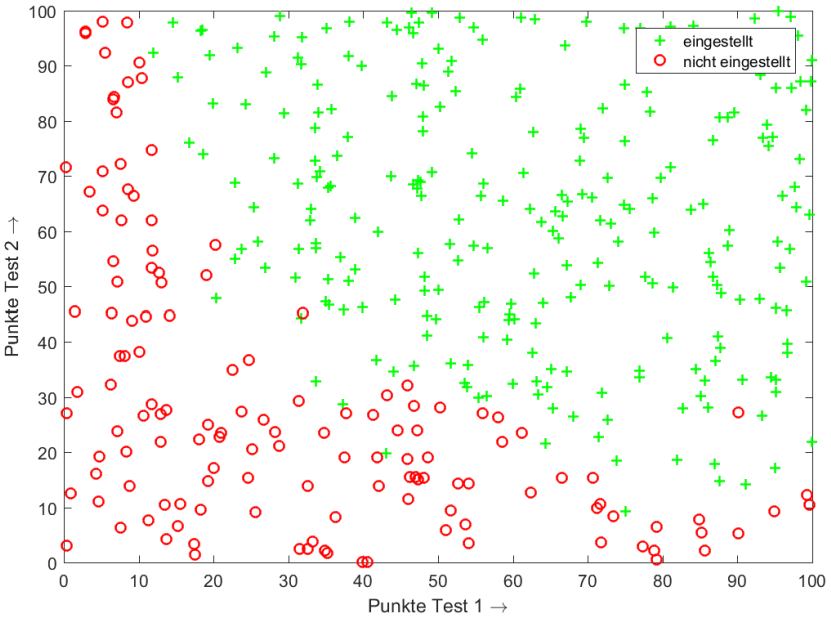

Abbildung 1: Streudiagramm für die Einstellungstest-Trainingsdaten.

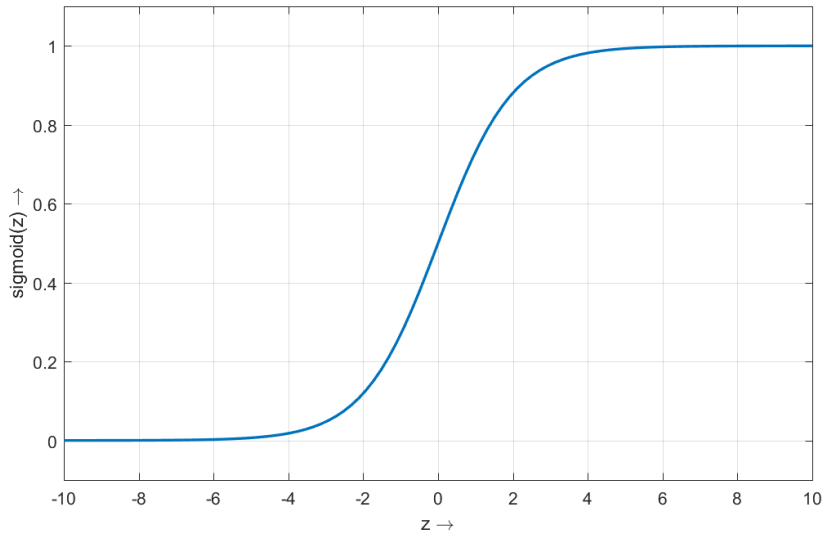

Abbildung 2: Sigmoidfunktion.

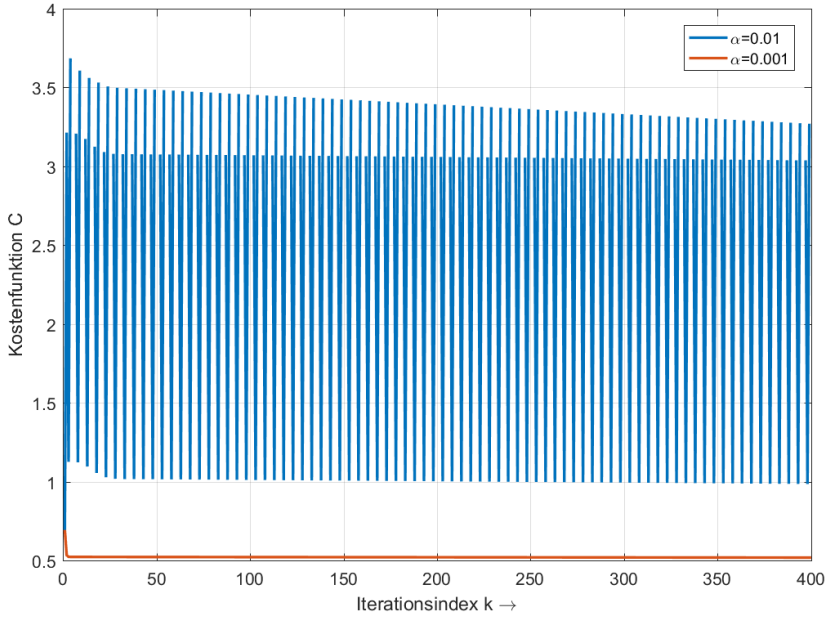

Abbildung 3: Verlauf der Kostenfunktion für zwei unterschiedliche Schrittweitenn, nämlich $\alpha = 0.01$ (Divergenz) und $\alpha = 0.001$ (vermeintliche Konvergenz, für den Parameter β_0 konvergiert das Verfahren äußerst langsam).

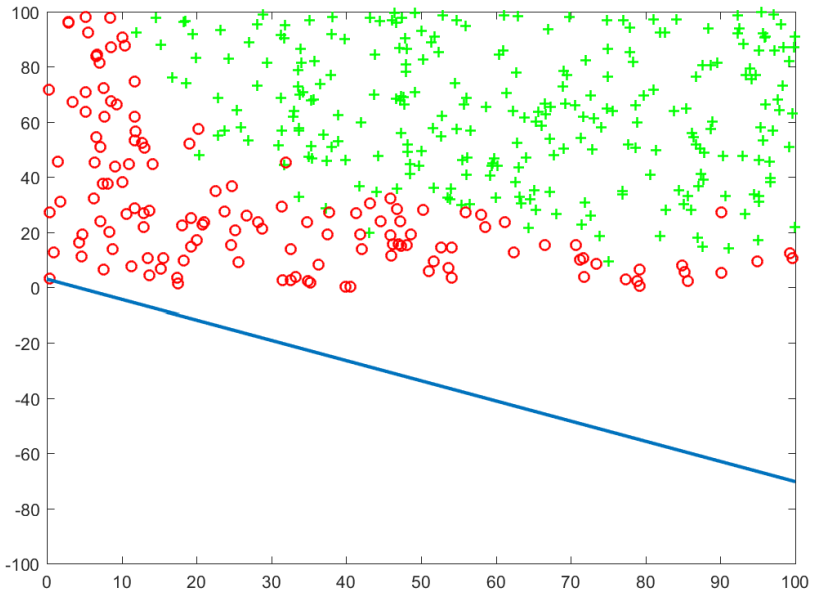

Abbildung 4: Entscheidungsschwelle mit dem Parametervektor, der nach 400 Iterationen mit $\alpha = 0.001$ und unskalierten Eingangsvariablen erreicht wird. Aufgrund der schlechten Konvergenz für β_0 ist der Achsenabschnitt der Entscheidungsschwelle noch völlig falsch.

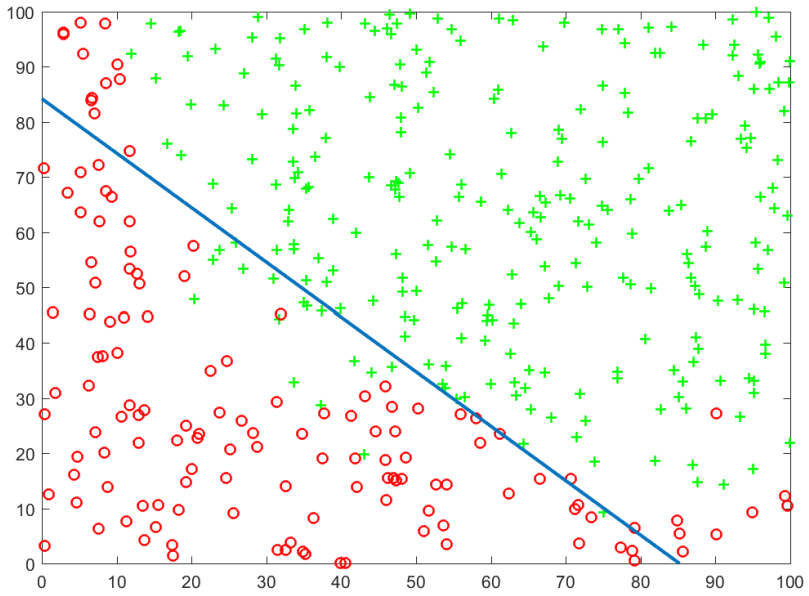

Abbildung 5: Entscheidungsschwelle mit dem Parametervektor, der nach 400 Iterationen mit $\alpha = 0.5$ und skalierten Eingangsvariablen erreicht wird.

3 Logistische Regression mit Regularisierung

Durchführung

Aufgabe D2: Logisitsche Regression mit den Qualitätskontrolle-Daten

In dieser Aufgabe wenden wir die logistische Regression auf Daten einer industriellen Qualitätskontrolle an.[a] In der Produktionanlage eines Halbleiterherstellers werden zur Beurteilung eines Leistungs-MOSFETs zwei einfache Messungen durchgeführt, die im Folgenden als die beiden Eingangsvariablen x_1 und x_2 verwendet werden. Um beurteilen zu können, ob diese beiden Messungen ausreichen, um zu entscheiden, ob der MOSFET gut (d.h. er erfüllt die Spezifikationen und kann verkauft werden) oder schlecht (d.h. er erfüllt die Spezifikationen nicht und ist somit Ausschuss) ist, wird eine Auswahl von MOSFETs genau untersucht. Von dieser Auswahl kennen wir somit exakt (wir nennen das auch "ground truth") die Ausgangsvariable y, die angibt oder der MOSFET gut ($y = 1$) oder schlecht ($y = 0$) ist. Wir benutzen einen Teil der Auswahl an Bauteilen als Trainingsdaten und den restlichen Teil als Validierungsdaten.

a) Laden Sie die Daten aus dem File `D2_Training.mat`. In diesem File ist die Matrix `X` mit den Beobachtungen der beiden Eingangsvariablen x_1 und x_2 sowie der Vektor `y` mit den Beobachtungen der Ausgangsvariable y enthalten.

b) Stellen Sie die Daten als Streudiagramme (scatter plots) dar. Dazu können Sie Ihre eigene Funktion aus der letzten Aufgabe oder auch die bereits vorgegebene Funktion `zeichneStreudiagramm` benutzen.

c) Aus den relativ wenigen Beobachtungen der Trainingsdaten lässt sich erahnen, dass der Zusammenhang zwischen x_1, x_2 und y relativ kompliziert ist. Schätzen Sie mit Hilfe der Funktionen aus der vorherigen Aufgabe den Parametervektor $\hat{\boldsymbol{\beta}}$ für die logistische Regression (benutzen Sie `fminunc`, weil das einfacher ist als das Gradientenverfahren) und stellen Sie dann die Entscheidungsschwelle zusammen mit den Trainingsdaten dar. Führen Sie die logistische Regression für unterschiedliche Polynomgrade a durch (eine gute Wahl für die Werte a ist z.B. $a \in \{2, 3, 5, 20\}$). Anhand der jeweils resultierenden Entscheidungsschwelle werden Sie feststellen, dass mit $a \leq 2$ keine Entscheidungsschwelle, welche die Daten genau genug beschreibt, erreicht werden kann. Für $a \geq 3$ treten bereits Anzeichen von Overfitting auf. Abbildung 6 zeigt die Entscheidungsschwellen für 4 unterschiedliche Polynomgrade.

d) Um den Parametervektor für die logistische Regression in unserem Szenario auch für hohe Polynomgrade zuverlässig bestimmen zu können, setzen wir die L2-Norm-Regularisierung ein. Implementieren Sie dazu eine MATLAB-Funktion, welche bei gegebener Matrix $\boldsymbol{X}$, gegebenem Vektor $\boldsymbol{y}$, gegebenem Parametervektor $\boldsymbol{\beta}$ und gegebener Regularisierungskonstante λ die Kostenfunktion $C_R(\boldsymbol{\beta})$ und deren Gradienten $\nabla C_R(\boldsymbol{\beta})$ berechnet. Der Funktionsaufruf soll folgendermaßen aussehen:

```
[C, gC] = KostenfunktionReg(beta, X, y,lambda);
```

wobei `C` der Wert der Kostenfunktion $C_R(\boldsymbol{\beta})$ und `gC` ihr Gradient ist. **`beta`** ist der aktuelle Parametervektor, `X` ist die Datenmatrix mit den Beobachtungen der Eingangsvariablen, `y` ist der Vektor mit den Beobachtungen der Ausgangsvariable und `lambda` ist die Regularisierungskonstante. Für die Implementierung der neuen Kostenfunktion können Sie die Kostenfunktion aus der vorherigen Aufgabe um die Regularisierung erweitern.

e) Testen Sie jetzt die `KostenfunktionReg` mit den gegebenen Trainingsdaten. Fügen Sie dazu in die Matrix `X` noch eine Zeile mit lauter Einsen ein (Hilfsvariable $x_0 = 1$). Wählen Sie als Parametervektor $\boldsymbol{\beta} = \mathbf{0}$. Dann sollte die Kostenfunktion folgende Werte liefern:

```
C =
```

```
    0.6931
gC =
   -0.1050
    0.0204
    0.0032
```

f) Schätzen Sie mit Hilfe der neuen Kostenfunktion `KostenfunktionReg` und mit `fminunc` den Parametervektor $\hat{\boldsymbol{\beta}}_R$ für die logistische Regression mit Regularisierung und stellen Sie dann die Entscheidungsschwelle zusammen mit den Trainingsdaten dar. Führen Sie die logistische Regression für den Polynomgrad $a = 20$ und unterschiedlichen Werte von λ durch. Berechnen Sie für jeden Parametervektor $\hat{\boldsymbol{\beta}}_R$, den Sie abhängig von λ erhalten, auch die Erkennungsrate für die Validierungsdaten (File `D2_Validierung.mat`). Abbildung 7 zeigt die Entscheidungsschwelle für unterschiedliche Werte von λ. Für die Erkennungsrate mit den Validierungsdaten ergeben sich folgende Werte:

```
lambda =
   1.0000e-03
ER =
    0.7895
lambda =
    0.0100
ER =
    0.8875
lambda =
    0.1000
ER =
    0.9200
lambda =
     1
ER =
    0.9215
lambda =
    10
ER =
    0.8825
lambda =
    100
ER =
    0.8365
```

Demzufolge ist für $a = 20$ der Wert $\lambda = 1$ am besten und $\lambda = 0.1$ ist ähnlich gut geeignet. Für $\lambda < 0.1$ tritt Overfitting auf und die Erkennungsrate mit den Validierungsdaten ist kleiner. Für $\lambda > 1$ tritt Underfitting auf. Auch hier ist die Erkennungsrate kleiner als für gute Werte von λ.

[a] Diese Daten sind simuliert.

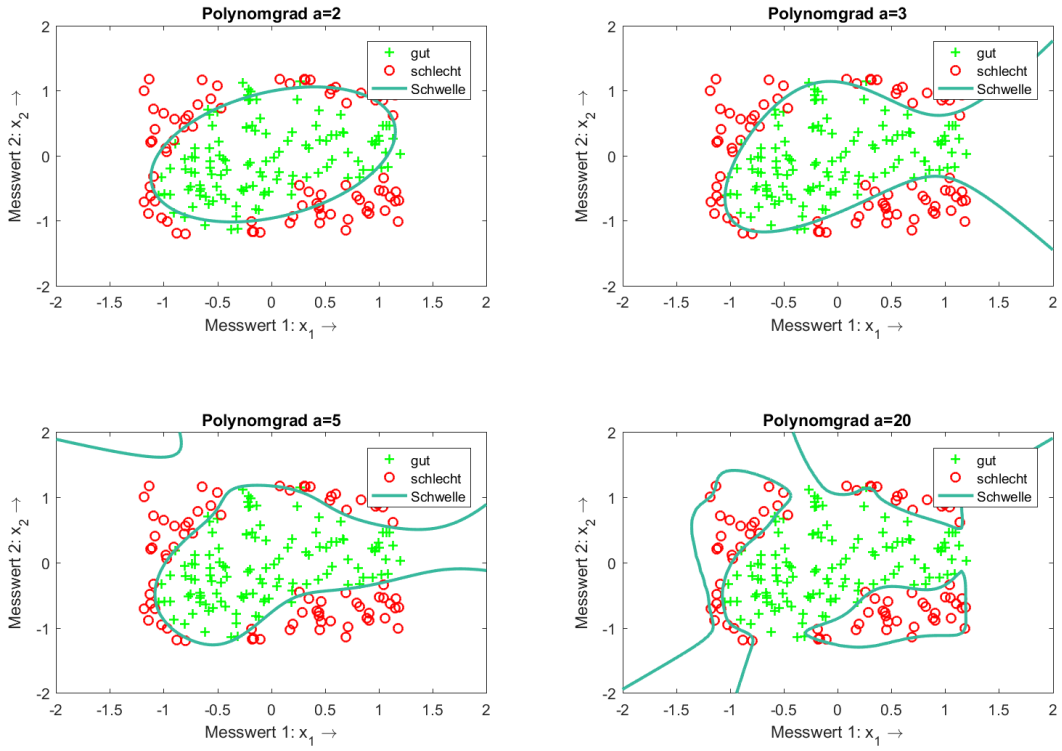

Abbildung 6: Entscheidungsschwelle für unterschiedliche Polynomgrade a ohne Regularisierung.

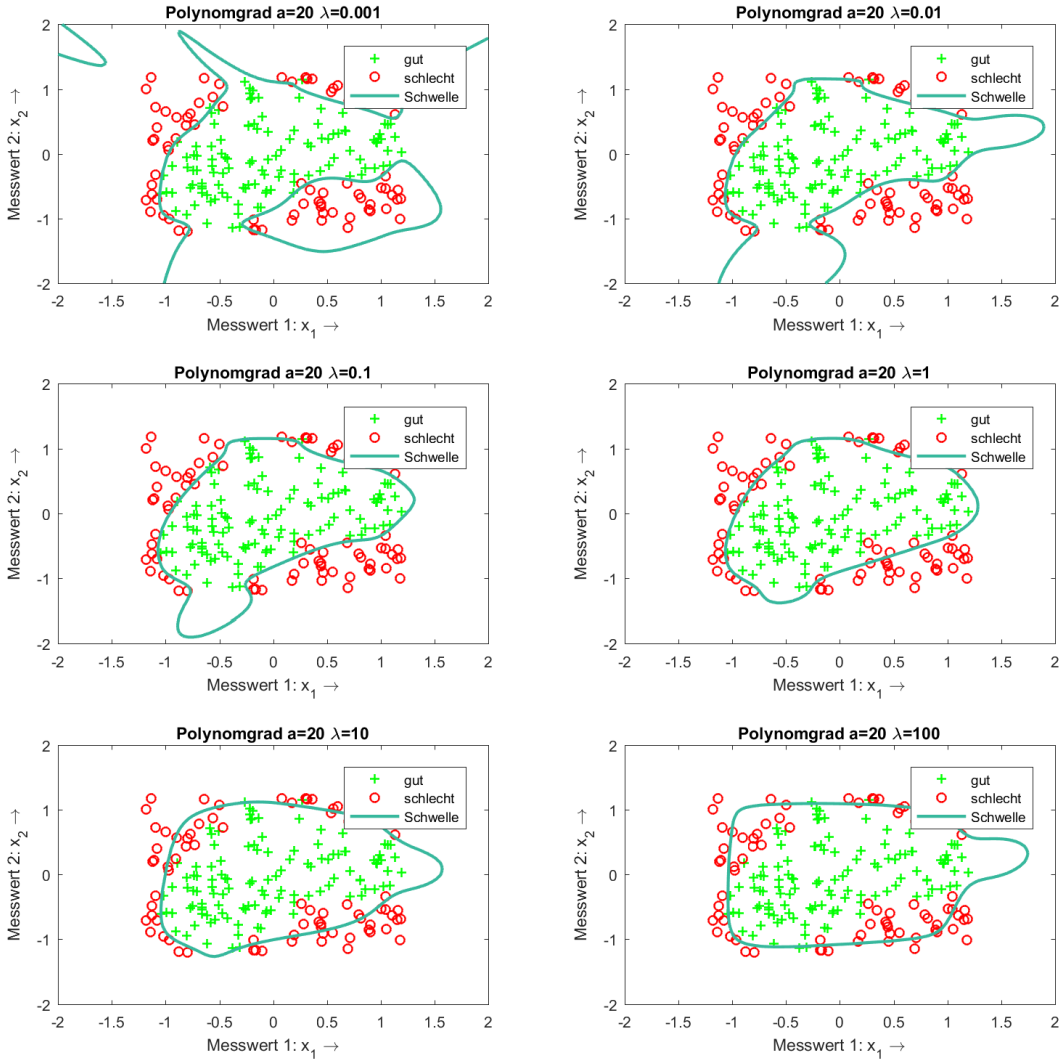

Abbildung 7: Entscheidungsschwelle für unterschiedliche Regularisierungsfaktoren λ bei einem Polynomgrad von $a = 20$.

Weiterführende Literatur

[1] G. James, D. Witten, T. Hastie, and R.Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

[2] T. Hastie, R.Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.