Einführung in die mathematische Datenanalyse

Jan Heiland

FAU Erlangen-Nürnberg – Sommersemester 2022

Contents

Vorwort

Das ist ein Aufschrieb.

Korrekturen und Wünsche immer gerne als *issues* oder *pull requests* ans github-repo.

Chapter 1

Was ist Data Science?

Data Science umfasst unter anderem folgende Aufgaben:

1. Strukturieren/Aufbereiten (Umgehen mit falschen, korrumpierten, fehlenden, unformatierten Daten)
2. Data Exploration (Daten "verstehen")
3. Data Analysis (quantitative Analysen, Hypothesen aufstellen)
4. Data Visualization (Hypothesen graphisch kommunizieren)
5. Modelle erzeugen/validieren (Regeln/Muster erkennen, Vorhersagen treffen) – *das* ist Machine Learning aber es gibt auch viele andere Ansätze.
6. Daten Reduktion

1.1 Wie passiert die Datenanalyse?

Mit mathematischen Methoden aus den Bereichen der

- linearen Algebra (z.B. Matrizen, Basen, lineare Gleichungssysteme)
- Statistik (z.B. Mittelwerte, Korrelationen, Verteilungen)
- Analysis (Grenzwerte, Abschätzungen)
- …

Dabei hilft Software, z.B.,

- Excel
- **Python**
- Matlab
- R

bei der Berechnung, Automatisierung, Visualisierung.

Python ist ein de-facto Standard in Data Science und Machine Learning.

1.2 Was sind Daten?

Wie sehen Daten aus?

- Numerisch reell, z.B. Temperatur
- Numerisch diskret, z.B. Anzahl
- Ordinal: Element einer festen Menge mit expliziter Ordnung, z.B. {neuwertig, mit Gebrauchsspuren, defekt}
- Binär: Eine von zwei Möglichkeiten, z.B. Wahr/Falsch oder aktiv/inaktiv
- Kategoriell: Element einer festen Menge ohne klare Ordnung, z.B. {Säugetier, Vogel, Fisch}
- sonstige strukturierte Daten, z.B. geographische Daten, Graphen
- reiner Text, z.B. Freitext in Restaurantbewertung

Außerdem können wir noch allgemeine Eigenschaften (Qualitätsmerkmale) von Daten unterscheiden

- strukturiert
- lückenhaft
- fehlerbehaftet (*verrauscht*)
- interpretierbar
- geordnet (oder nicht zu ordnen)

1.3 Beispiele

1.3.1 Tabellendaten – Mietpreise

Hier wären die Aufgaben von Data Science:

- Daten "verstehen", Zusammenhänge zwischen Variablen aufdecken,
- visualisieren.
- Gegebenenfalls fehlende Einträge bei (z.B.) `kaltmiete` vorhersagen

Datenexploration und -analyse für einzelne Variablen 1/3 Wir betrachten eine *numerische* Variable in einem rechteckigen Datensatz, also eine *Spalte* (z.B. `kaltmiete`). Wir bezeichnen den i-ten Eintrag in dieser Spalte mit x_i, wobei $i = 1, \ldots, N$ (N Anzahl der Zeilen).

Folgende *Schätzer/Metriken* können dabei helfen, diese Spalte besser zu verstehen:

- Mittelwert $\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$

	bundesland	stadt	baujahr	etage	hat_kueche	kaltmiete	wohnflaeche	zimmer
0	Nordrhein_Westfalen	Duisburg	1973	1	0	640	105	3
1	Baden_Württemberg	Ulm	1936	1	0	302	43	2
2	Nordrhein_Westfalen	Wuppertal	1986	3	0	150	67	2
3	Rheinland_Pfalz	Rhein_Lahn_Kreis	1970	3	0	315	47	3
4	Sachsen	Chemnitz	1900	2	0	315	63	2
5	Rheinland_Pfalz	Mainz	1989	1	0	1200	96	4
6	Bayern	Aschaffenburg_Kreis	1965	2	0	722	85	3
7	Berlin	Berlin	1952	0	1	706	63	2
8	Sachsen	Mittelsachsen_Kreis	1996	3	1	220	29	1
9	Brandenburg	Potsdam	1985	2	1	400	34	1

Figure 1.1: Abbildung: Tabelle von Wohnungsangeboten

	bundesland	stadt	baujahr	etage	hat_kueche	kaltmiete	wohnflaeche	zimmer
0	Nordrhein_Westfalen	Duisburg	1973	1	0	640	105	3
1	Baden_Württemberg	Ulm	1936	1	0	302	43	2
2	Nordrhein_Westfalen	Wuppertal	1986	3	0	150	67	2
3	Rheinland_Pfalz	Rhein_Lahn_Kreis	1970	3	0	315	47	3
4	Sachsen	Chemnitz	1900	2	0	315	63	2
5	Rheinland_Pfalz	Mainz	1989	1	0	1200	96	4
6	Bayern	Aschaffenburg_Kreis	1965	2	0	722	85	3
7	Berlin	Berlin	1952	0	1	706	63	2
8	Sachsen	Mittelsachsen_Kreis	1996	3	1	220	29	1
9	Brandenburg	Potsdam	1985	2	1	400	34	1

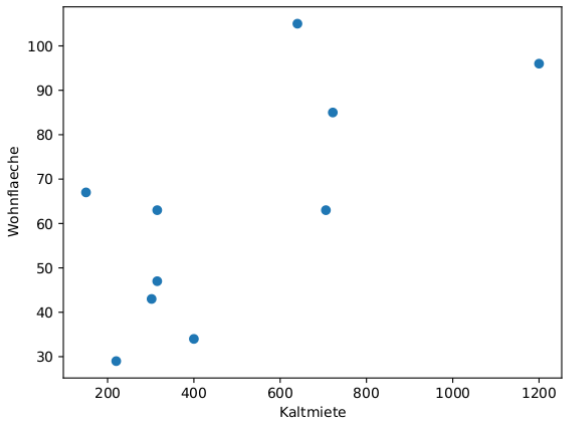
Figure 1.2: Abbildung: Eine Spalte der Tabelle

- gewichteter Mittelwert $\overline{x}_w = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{j=1}^{N} w_j}$, wobei w_i das Gewicht des i-ten Eintrages ist (z.B. eine andere Variable).
- Varianz: $s_x^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})^2$
- Standardabweichung $s = \sqrt{s_x^2}$.
- Median $= \frac{315+400}{2} = 357.5$.

Datenexploration und -analyse für mehrere Variablen Wir betrachten zwei Spalten $x = (x_1, \dots, x_N)$ und $y = (y_1, \dots, y_N)$.

Das Verteilung von zwei Variablen läßt sich im sogenannte **Scatter Plot** visualisieren.

che	kaltmiete	wohnflaeche	zin
0	640	105	
0	302	43	
0	150	67	
0	315	47	
0	315	63	
0	1200	96	
0	722	85	
1	706	63	
1	220	29	
1	400	34	



Datenexploration und -analyse für mehrere Variablen Wir betrachten zwei Spalten $x = (x_1, \dots, x_N)$ und $y = (y_1, \dots, y_N)$.

- Kovarianz $s_{xy} = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y})$
- Korrelation $\rho_{xy} = \frac{s_{xy}}{s_x \cdot s_y} \in [-1, 1]$.
- $\rho \approx 1$: Starke positive Korrelation, wenn x groß ist, ist y auch groß.
- $\rho \approx -1$: Starke negative Korrelation, wenn x groß ist, ist y klein
- $\rho \approx 0$: Wenig/keine Korrelation.

1.3.2 COVID-19 Daten

Vergleiche die Einführung in *Mathematik für Data Science 1* vom letzten Semester.

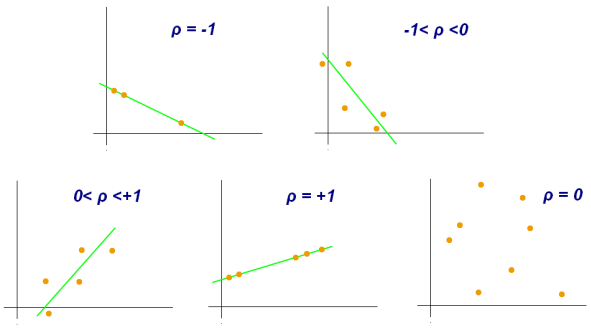


Figure 1.3: Von Kiatdd - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=37108966

1.3.3 Netflix Prize

Hierbei geht es darum, ob aus bekannten Bewertungen von vielen verschiedenen Benutzern für viele verschiedene Filme abgeleitet werden kann, ob ein bestimmter Nutzer einen bestimmten Film mag (also positiv bewerten würde).

Vergleiche auch Wikipedia:Netflix_Prize

Das (Trainings-)Daten bestehen über `480189` Benutzer, die für `17770` Filme insgesamt `100480507` Bewertungen als ganze Zahlen zwischen `1` und `5` verteilten.

Ziel der Datenanalyse war es, für `2817131` "Paare" von Benutzern und Filmen, die Bewertung vorauszusagen. Neben der schieren Masse an Daten kamen noch Einschränkungen hinzu, die ein Mindestmaß an Qualität der Vorhersage sicherstellen sollten.

Das Problem ließe sich wie folgt darstellen.

Benutzer \ Film	`F1`	`F2`	...	`Fn`	...
`B1`	–	3	...	5	...
`B2`	3	4	...	2	...
`B3`	1	2	...	?	...
...	3	4	...	–	...

Gegeben viele (aber bei weitem nicht alle) Einträge in einer riesigen Tabelle. Können wir aus den Zusammenhängen bestimmte fehlende Einträge (z.B. wie findet Nutzer `B3` den Film `Fn`) herleiten?

Die besten Lösungen für dieses Problem basieren durchweg auf *Machine Learning* Ansätzen.

1.4 Python

Die Programmiersprache `python` wird uns durchs Semester begleiten. Einfach weil sie so wichtig ist für *Data Science* aber auch weil sie (meiner Meinung nach) einfach zu erlernen und zu benutzen ist.

1.5 Aufgaben

1.5.1 Python

Bringen sie ihr `python` zum Laufen, installieren sie `numpy`, `scipy` und

```
N = 20
xmax = 2
xmin = 0

xdata = np.linspace(xmin, xmax, N)
ydata = np.exp(xdata)

plt.figure(1)
plt.plot(xdata, ydata, '.')

plt.figure(2)
plt.semilogy(xdata, ydata, '.')
plt.show()
```

1.5.2 Einheitsmatrix

Schreiben sie ein script, dass die `5x5` Einheitsmatrix auf 3 verschiedene Arten erzeugt. (Eine Art könnte die eingebaute `numpy` Funktion `eye` sein).

```
import numpy as np

idfive = np.eye(5)
print(idfive)
```

Hinweis: schauen sie sich mal an wie `numpy`'s `arrays` funktionieren.

1.5.3 Matrizen Multiplikation und Potenz

Schreiben sie ein script, das die Übungsaufgabe aus der Vorlesung (potenzieren der Matrizen M_i, $i = 1, 2, 3, 4$) löst. Zum Beispiel mit

```
import numpy as np
mone = np.array([[0.9, 0.9], [0.9, 0.9]])

mone_ptwo = mone @ mone
print(mone_ptwo)

mone_pfour = mone_ptwo @ mone_ptwo
print(mone_pfour)
```

Oder so:

```
import numpy as np
mone = np.array([[0.9, 0.9], [0.9, 0.9]])
mone_p = np.eye(2)

for k in range(16):
```

```
    mone_p = mone_p @ mone
    if k == 1 or k == 3 or k == 15:
        print('k=', k+1)
        print(mone_p)
```

Achtung:

- bei Matrizen kann auch `*` benutzt werden – das ist aber nicht die richtige Matrizenmultiplikation (sondern die Multiplikation eintragsweise)
- Moegliche Realisierung der Matrizenmultiplikation
  - `np.dot(A, B)` – die klassische Methode
  - `A.dot(B)` – das selbe (manchmal besser, wenn `A` etwas allgemeiner ist (zum Beispiel eine `scipy.sparse` matrix)
  - `A @ B` – convenience Notation

Chapter 2

Lineare Regression

Auch bekannt als

- *lineare Ausgleichsrechnung* oder
- *Methode der kleinsten Quadrate*.

Ein wesentlicher Aspekt von *Data Science* ist die Analyse oder das Verstehen von Daten. Allgemein gesagt, es wird versucht, aus den Daten heraus Aussagen über Trends oder Eigenschaften des Phänomens zu treffen, mit welchem die Daten im Zusammenhang stehen.

Wir kommen nochmal auf das Beispiel aus der Einführungswoche zurück, werfen eine bereits geschärften Blick darauf und gehen das mit verbesserten mathematischen Methoden an.

Gegeben seien die Fallzahlen aus der CoVID Pandemie 2020 für Bayern für den Oktober 2020.

Table 2.1: Anzahl der SARS-CoV-2 Neuinfektionen in Bayern im Oktober 2020.

Tag	1	2	3	4	5	6	7	8	9	10	11
Fälle	352	347	308	151	360	498	664	686	740	418	320

Tag	12	13	14	15	16	17	18	19	20	21
Fälle	681	691	1154	1284	127	984	573	1078	1462	2239

Tag	22	23	24	25	26	27	28	29	30	31
Fälle	2236	2119	1663	1413	2283	2717	3113	2972	3136	2615

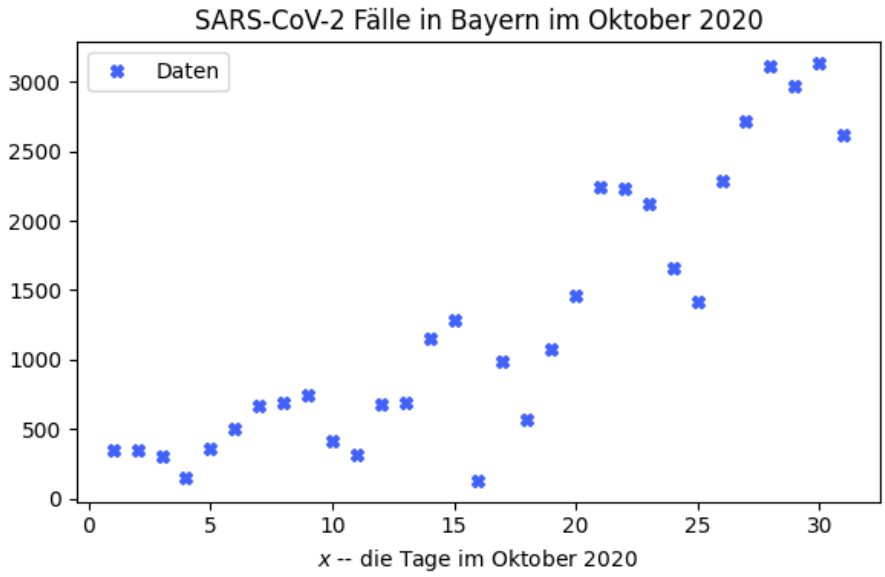


Figure 2.1: Fallzahlen von Sars-CoV-2 in Bayern im Oktober 2020

Wieder stellen wir uns die Frage ob wir **in den Daten einen funktionalen Zusammenhang** feststellen können. Also ob wir die Datenpaare

(Tag x, Infektionen am Tag x)

die wir als

(x_i, y_i)

über eine Funktion f und die Paare

$(x, f(x))$

beschreiben (im Sinne von gut darstellen oder approximieren) können.

2.1 Rauschen und Fitting

Beim obigen Beispiel (und ganz generell bei Daten) ist davon auszugehen, dass die Daten **verrauscht** sind, also einem Trend folgen oder in einem funktionalen Zusammenhang stehen aber zufällige Abweichungen oder Fehler enthalten.

Unter diesem Gesichtspunkt ist eine Funktion, die

$$f(x_i) = y_i$$

erzwingt nicht zielführend. (Wir wollen Trends und größere Zusammenhänge erkennen und nicht kleine Fehler nachzeichnen.) Das zu strenge Anpassen an möglicherweise verrauschte Daten wird **overfitting** genannt.

Vielmehr werden wir nach einer Funktion f suchen, die die Daten näherungsweise nachstellt:

$$f(x_i) \approx y_i$$

Hierbei passen jetzt allerdings auch Funktionen, die vielleicht einfach zu handhaben sind aber die Daten kaum noch repräsentieren. Jan spricht von **underfitting**.

Eine gute Approximation besteht im Kompromiss von *nah an den Daten* aber mit wenig *overfitting*.

2.2 Ansätze für lineare Regression

Um eine solche Funktion f zu finden, trifft Jan als erstes ein paar Modellannahmen. Modellannahmen legen fest, wie das f im Allgemeinen aussehen soll und versuchen dabei

1. die Bestimmung von f zu ermöglichen
2. zu garantieren, dass f auch die gewollten Aussagen liefert
3. und sicherzustellen, dass f zum Problem passt.

Jan bemerke, dass die ersten beiden Annahmen im Spannungsverhältnis zur dritten stehen.

Lineare Regression besteht darin, dass die Funktion als Linearkombination

$$f_w(x) = \sum_{j=1}^{n} w_j b_j(x)$$

von Basisfunktionen geschrieben wird und dann die *Koeffizienten* w_i so bestimmt werden, dass f die Daten bestmöglich annähert.

> Jan bemerke, dass *bestmöglich* wieder *overfitting* bedeuten kann aber auch, bei schlechter Wahl der Basis, wenig aussagekräftig sein kann. Der gute Kompromiss liegt also jetzt in der Wahl der passenden Basisfunktionen und deren Anzahl. (Mehr Basisfunktionen bedeutet möglicherweise bessere Approximation aber auch die Gefahr von *overfitting*.)

Typische Wahlen für die Basis $\{b_1, b_2, \dots, b_n\}$ sind

- Polynome: $\{1, x, x^2, \dots, x^{n-1}\}$ – für $n = 2$ ist der Ansatz *eine Gerade*

- Trigonometrische Funktionen: $\{1, \cos(x), \sin(x), \cos(2x), \sin(2x), \dots\}$
- Splines – Polynome, die abschnittsweise definiert werden
- Wavelets – Verallgemeinerungen von trigonometrischen Funktionen

2.3 Fehlerfunktional und Minimierung

Wir setzen nun also an

$$f_w(x) = \sum_{j=1}^{n} w_j b_j(x)$$

und wollen damit $y_i \approx f_w(x_i)$ *bestmöglich* erreichen (indem wir die Koeffizienten $(w_1, \dots, w_n)$ *optimal* wählen. Bestmöglich und optimal spezifizieren wir über den Mittelwert der quadratischen Abweichungen in der Approximation über alle Datenpunkte

$$\frac{1}{N} \sum_{i=1}^{N} (y_i - f_w(x_i))^2$$

Ein paar Bemerkungen

- jetzt müssen wir die w_i's bestimmen so dass dieser Fehlerterm minimal wird
- das optimale w ist unabhängig von einer Skalierung des Fehlerterms
- des wegen schreiben wir gerne einfach $\frac{1}{2} \sum_{i=1}^{N} (y_i - f_w(x_i))^2$ als das Zielfunktional, das es zu minimieren gilt.

Wie finden wir jetzt die w_i's? Zunächst gilt, dass

$$f_w(x_i) = \sum_{i=j}^{n} w_j b_j(x_i) = \begin{bmatrix} b_1(x_i) & b_2(x_i) & \dots & b_n(x_i) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

und wenn wir alle $f_w(x_i)$, $i = 1, \dots, N$ übereinander in einen Vektor schreiben, dass

$$f_w(\mathbf{x}) := \begin{bmatrix} f_w(x_1) \\ \vdots \\ f_w(x_N) \end{bmatrix} = \begin{bmatrix} b_1(x_1) & \dots & b_n(x_1) \\ \vdots & \ddots & \vdots \\ b_1(x_N) & \dots & b_n(x_N) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} =: \Phi(\mathbf{x}) w$$

Damit, mit $\mathbf{y}$ als den Vektor aller y_i's, und mit der Definition der Vektornorm, können wir unser Minimierungsproblem schreiben als

$$\frac{1}{2} \sum_{i=1}^{N} (y_i - f_w(x_i))^2 = \frac{1}{2} \|\mathbf{y} - \Phi(\mathbf{x}) w\|^2 \to \min .$$

Wir bemerken, dass

- das Fehlerfunktional immer größer und bestenfalls gleich 0 ist
- falls das lineare Gleichungssystem $\Phi(\mathbf{x})w = \mathbf{y}$ eine Lösung w hat, ist das auch eine Lösung unserer Minimierung
- im typischen Falle aber ist allerdings $N \gg n$ und das System überbestimmt ($n = N$ würde ein overfitting bedeuten…) sodass wir keine Lösung des linearen Gleichungssystems erwarten können.
- Das Minimierungsproblems selbst hat allerdings immer eine Lösung.

2.4 Berechnung der Bestlösung

Wir suchen also ein Minimum der Funktion (mit Φ, $\mathbf{x}$, $\mathbf{y}$ gegeben)

$$\begin{aligned} w \mapsto \frac{1}{2}\|\mathbf{y} - \Phi(\mathbf{x})w\|^2 &= \frac{1}{2}(\mathbf{y} - \Phi(\mathbf{x})w)^T(\mathbf{y} - \Phi(\mathbf{x})w) \\ &= \frac{1}{2}[\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\Phi(\mathbf{x})w - w^T\Phi(\mathbf{x})^T\mathbf{y} + w^T\Phi(\mathbf{x})^T\Phi(\mathbf{x})w] \\ &= \frac{1}{2}[\mathbf{y}^T\mathbf{y} - 2w^T\Phi(\mathbf{x})^T\mathbf{y} + w^T\Phi(\mathbf{x})^T\Phi(\mathbf{x})w] \end{aligned}$$

wobei wir die Definition der Norm $\|v\|^2 = v^Tv$ und die Eigenschaft, dass für die skalare Größe $w^T\Phi(\mathbf{x})^T\mathbf{y} = [w^T\Phi(\mathbf{x})^T\mathbf{y}]^T = \mathbf{y}^T\Phi(\mathbf{x})w$ gilt, ausgenutzt haben.

Wären w und $\mathbf{y}$ keine Vektoren sondern einfach reelle Zahlen, wäre das hier eine Parabelgleichung $aw^2 + bw + c$ mit $a > 0$, die immer eine Minimalstelle hat.

Tatsächlich gilt hier alles ganz analog. Insbesondere ist $\Phi(\mathbf{x})^T\Phi(\mathbf{x})$ in der Regel "größer 0" (was heißt das wohl bei quadratischen Matrizen?). Und mittels "Nullsetzen" der ersten Ableitung können wir das Minimum bestimmen. In diesem Fall ist die erste Ableitung (nach w)

$$\nabla_w(\frac{1}{2}\|\mathbf{y} - \Phi(\mathbf{x})\|^2) = \Phi(\mathbf{x})^T\Phi(\mathbf{x})w - \Phi(\mathbf{x})^T\mathbf{y},$$

(den *Gradienten* ∇_w als Ableitung von Funktionen mit mehreren Veränderlichen werden wir noch genauer behandeln) was uns als Lösung, die Lösung des linearen Gleichungssystems

$$\Phi(\mathbf{x})^T\Phi(\mathbf{x})w = \Phi(\mathbf{x})^T\mathbf{y}$$

definiert.

Letzte Frage: Wann hat dieses Gleichungssystems eine eindeutige Lösung? Mit $N > n$ (also $\Phi(\mathbf{x})$ hat mehr Zeilen als Spalten) gelten die Äquivalenzen:

- $\Phi(\mathbf{x})^T\Phi(\mathbf{x})w = \Phi(\mathbf{x})^T\mathbf{y}$ hat eine eindeutige Lösung
- die Matrix $\Phi(\mathbf{x})^T\Phi(\mathbf{x})$ ist regulär
- die Spalten von $\Phi(\mathbf{x})$ sind linear unabhängig
- die Vektoren $b_i(\mathbf{x})$ sind linear unabhängig.

Praktischerweise tritt genau diese Situation im Allgemeinen ein.

- $N > n$ (mehr Datenpunkte als Parameter)
- b_i's werden als *linear unabhängig* (im Sinne ihres Funktionenraums) gewählt, was die lineare unabhängigket der $b_i(\mathbf{x})$ impliziert.

2.5 Beispiel

Unsere Covid-Zahlen "mit einer Geraden angenähert":

- $f_w(x) = w_1 + w_2 x$ – das heißt $n = 2$ und Basisfunktionen $b_1(x) \equiv 1$ und $b_2(x) = x$
- $\mathbf{x} = (1, 2, 3, \dots, 31)$ – die Tage im Februar, das heißt $N = 31$
- $\mathbf{y} = (352, 347, \dots, 2615)$ – die Fallzahlen

Wir bekommen

$$\Phi(\mathbf{x}) = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ \vdots & \vdots \\ 1 & 31 \end{bmatrix}$$

(die Spalten sind linear unabhängig) und müssen "nur" das 2x2 System

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & 31 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ \vdots & \\ 1 & 31 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & 31 \end{bmatrix} \begin{bmatrix} 352 \\ 347 \\ 308 \\ \vdots \\ 2615 \end{bmatrix}$$

lösen um die Approximation f_w zu bestimmen.

Und noch als letzte Bemerkung. Egal wie die Basisfunktionen b_i gewählt werden, die Parameterabhängigkeit von w ist immer linear. Deswegen der Name **lineare Ausgleichsrechnung**.

Chapter 3

Matrix-Zerlegungen

Die Lösung w des Problems der *linearen Ausgleichsrechnung* war entweder als Lösung eines Optimierungsproblems

$$\min_w \|Aw - y\|^2$$

oder als Lösung des linearen Gleichungssystems

$$A^T Aw = y$$

gegeben. Hierbei steht nun $A \in \mathbb{R}^{N \times n}$ für die Matrix $\Phi(\mathbf{x})$ der Daten und Basisfunktionen. Wir hatten uns überlegt, dass in den meisten Fällen

- die Matrix mehr Zeilen als Spalten hat ($N > n$) und
- die Spalten linear unabhängig sind.

3.1 QR Zerlegung

Wir betrachten nochmal das Optimierungsproblem $\min_w \|Aw - y\|^2$. Gäbe es eine Lösung des Systems $Aw = y$, wäre das sofort eine Lösung des Optimierungsproblems. Da A aber mehr Zeilen als Spalten hat, ist das System $Aw = y$ überbestimmt und eine Lösung in der Regel nicht gegeben.

Die Überlegung ist nun, die Gleichung $Aw = y$ *so gut wie möglich* zu erfüllen, indem wir die relevanten Gleichungen indentifizieren und wenigstens diese lösen. Ein systematischer (und wie wir später sehen werden auch zum Optimierungsproblem passender) Zugang bietet die QR Zerlegung.

Theorem 3.1 (QR Zerlegung)**.** *Sei $A \in \mathbb{R}^{m \times n}$, $m > n$. Dann existiert eine orthonormale Matrix $Q \in \mathbb{R}^{m \times m}$ und eine obere Dreiecksmatrix $\hat{R} \in \mathbb{R}^{n \times n}$ derart dass*

$$A = QR =: Q \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}.$$

Hat A vollen (Spalten)Rang, dann ist $\hat{R}$ invertierbar.

Hier heißt *orthonormale Matrix* Q, dass die Spalten von Q paarweise orthogonal sind. Insbesondere gilt

$$Q^T Q = I.$$

Für unser zu lösendes Problem ergibt sich dadurch die Umformung

$$Aw = y \quad \Leftrightarrow \quad QRw = y \quad \Leftrightarrow \quad Q^T QRw = Q^T y \quad \Leftrightarrow \quad Rw = Q^T y$$

oder auch

$$\begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} w = Q^T y = \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} y$$

(wobei wir die $Q_1 \in \mathbb{R}^{m \times n}$ die Matrix der ersten n Spalten von Q ist) und als *Kompromiss* der Vorschlag, das Teilsystem

$$\hat{R}w = Q_1^T y$$

nach w zu lösen und in Kauf zu nehmen, dass der Rest, nämlich das $Q_2^T y$, nicht notwendigerweise gleich null ist.

Wir halten zunächst mal fest, dass

- Obwohl Q eine reguläre Matrix ist, bedarf der Übergang von $Aw = y$ zu $Q^T Aw = Q^T y$ einer genaueren Analyse.
- Wir bemerken, dass für eine hypothetische komplette Lösung $Aw - y$, diese Transformation keine Rolle spielt.
- Für die Kompromisslösung jedoch schon, weil beispielsweise verschiedene Konstruktionen eines invertierbaren Teils, verschiedene Residuen bedeuten und somit Optimalität im Sinne von $\min_w \|Aw - y\|^2$ nicht garantiert ist.

Allerdings, wie Sie als Übungsaufgabe nachweisen werden, löst dieser Ansatz tatsächlich das Optimierungsproblem.

3.2 Singulärwertzerlegung

Eine weitere Matrix Zerlegung, die eng mit der Lösung von Optimierungsproblemen oder überbestimmten Gleichungssystemen zusammenhängt ist die *Singulärwertzerlegung* (SVD – von english: *Singular Value Decomposition*).

Theorem 3.2 (Singulärwertzerlegung)**.** *Sei $A \in \mathbb{C}^{m \times n}$, $m \geq n$. Dann existieren orthogonale Matrizen $U \in \mathbb{C}^{m \times m}$ und $V \in \mathbb{C}^{n \times n}$ und eine Matrix*

$\Sigma \in \mathbb{R}^{m\times n}$ *der Form*

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \sigma_n \\ 0 & 0 & \dots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

mit reellen sogenannten Singulärwerten

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$$

sodass gilt

$$A = U\Sigma V^*$$

wobei gilt $V^* = \overline{V^T}$ *(transponiert und komplex konjugiert).*

Ein paar Bemerkungen.

- Ist A reell, können auch U und V reell gewählt werden.
- Die Annahme $m \geq n$ war nur nötig um für die Matrix Σ keine Fallunterscheidung zu machen. (Für $m \leq n$ "steht der Nullblock rechts von den Singulärwerten"). Insbesondere gilt $A^* = V\Sigma U^*$ ist eine SVD von A^*.
- Eine Illustration der Zerlegung ist in Abbildung 3.1 zu sehen.

Wir machen einige Überlegungen im Hinblick auf große Matrizen. Sei dazu $m > n$, $A \in \mathbb{C}^{m\times n}$ und $A = U\Sigma V^*$ eine SVD wie in Theorem @ref{thm:SVD}. Sei nun

$$U = [U_1 \quad U_2]$$

partitioniert sodass U_1 die ersten n Spalten von U enthält.

Dann gilt (nach der Matrix-Multiplikations Regel *Zeile mal Spalte* die Teile U_2 und V_2 immer mit dem Nullblock in Σ multipliziert werden) dass

$$A = U\Sigma V = [U_1 \quad U_2] \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix} V^* = U_1 \hat{\Sigma} V^*$$

Es genügt also nur die ersten m Spalten von U zu berechnen. Das ist die sogenannte **slim SVD**.

Hat, darüberhinaus, die Matrix A keinen vollen Rang, also $\mathrm{Rg}(A) = r < n$, dann:

- ist $\sigma_i = 0$, für alle $i = r+1, \dots, n$, (wir erinnern uns, dass die Singulärwerte nach Größe sortiert sind)
- die Matrix $\hat{\Sigma}$ hat $n - r$ Nullzeilen
- für die Zerlegung sind nur die ersten r Spalten von U und V relevant – die sogenannte **Kompakte SVD**.

In der Datenapproximation ist außerdem die **truncated SVD** von Interesse. Dazu sei $\hat{r} < r$ ein beliebig gewählter Index. Dann werden alle Singulärwerte, $\sigma_i = 0$, für alle $i = \hat{r}+1, \dots, n$, abgeschnitten – das heißt null gesetzt und die entsprechende *kompakte SVD* berechnet.

Hier gilt nun nicht mehr die Gleichheit in der Zerlegung. Vielmehr gilt

$$A \approx A_{\hat{r}}$$

wobei $A_{\hat{r}}$ aus der *truncated SVD* von A erzeugt wurde. Allerdings ist diese Approximation von A durch optimal in dem Sinne, dass es keine Matrix vom Rang $\hat{r} \geq r = \mathrm{Rg}(A)$ gibt, die A (in der *induzierten* euklidischen Norm[1]) besser approximiert. Es gilt

$$\min_{B \in \mathbb{C}^{m \times n}, \mathrm{Rg}(B) = \hat{r}} \|A - B\|_2 = \|A - A_{\hat{r}}\|_2 = \sigma_{\hat{r}+1};$$

vgl. Bollhoefer/Mehrmann Satz 14.15.

Zum Abschluss noch der Zusammenhang zum Optimierungsproblem. Ist $A = U\Sigma V^*$ "SV-zerlegt", dann gilt

$$A^*Aw = V\Sigma^*U^*U\Sigma V^*w = V\hat{\Sigma}^2V^*$$

und damit

$$A^*Aw = A^*y \quad \Leftrightarrow \quad V\hat{\Sigma}^2V^* = V\Sigma^*U^*y \quad \Leftrightarrow \quad w = V(\Sigma^+)^*U^*y$$

wobei

$$\Sigma^+ = \begin{bmatrix} \hat{\Sigma}^{-1} \\ 0_{m-n \times n} \end{bmatrix}$$

aus $\Sigma\hat{\Sigma}^{-1}\hat{\Sigma}^{-1}$ herrührt.

Bemerkung: Σ^+ kann auch definiert werden, wenn $\hat{\Sigma}$ nicht invertierbar ist (weil manche Diagonaleinträge null sind). Dann wird $\hat{\Sigma}^+$ betrachtet, bei welcher nur die $\sigma_i > 0$ invertiert werden und die anderen $\sigma_i = 0$ belassen werden. Das definiert eine sogenannte *verallgemeinerte Inverse* und löst auch das Optimierungsproblem falls A keinen vollen Rang hat.

3.3 Aufgaben

Erklärung: (T) heißt theoretische Aufgabe, (P) heißt programmieren.

3.3.1 Norm und Orthogonale Transformation (T)

Sei $Q \in \mathbb{R}^{n \times n}$ eine orthogonale Matrix und sei $y \in \mathbb{R}^n$. Zeigen Sie, dass

$$\|y\|^2 = \|Qy\|^2$$

gilt.

[1] Auf Matrixnormen kommen wir noch in der Vorlesung zu sprechen.

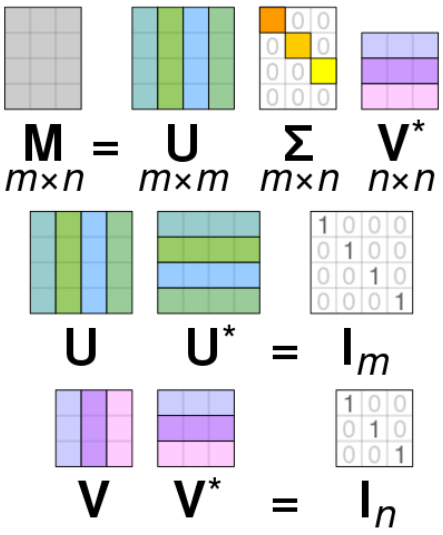


Figure 3.1: Illustration der SVD. Bitte beachten der $*$ bedeutet hier transponiert und komplex konjugiert. By Cmglee - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=67853297

3.3.2 Kleinste Quadrate und Mittelwert

Zeigen sie, dass der *kleinste Quadrate* Ansatz zur Approximation einer Datenwolke

$$(x_i, y_i), \quad i = 1, 2, \ldots, N,$$

mittels einer konstanten Funktion $f(x) = w_1$ auf w_1 auf den Mittelwert der y_i führt.

3.3.3 QR Zerlegung und Kleinstes Quadrate Problem (T)

Sei $A \in \mathbb{R}^{m,n}$, $m > n$, A hat vollen Rank und sei

$$[Q_1 \quad Q_2] \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} = A$$

eine QR-Zerlegung von A. Zeigen sie, dass die Lösung von

$$\hat{R}w = Q_1^T y$$

ein kritischer Punkt (d.h. der Gradient ∇_w verschwindet) von

$$w \mapsto \|Aw - y\|^2$$

ist. **Hinweis**: Die Formel für den Gradienten wurde in der Vorlesung 02 hergeleitet.

3.3.4 Eigenwerte Symmetrischer Matrizen (T)

Zeigen Sie, dass Eigenwerte symmetrischer reeller Matrizen $A \in \mathbb{R}^{n \times n}$ immer reell sind.

3.3.5 Singulärwertzerlegung und Eigenwerte (T)

Zeigen Sie, dass die quadrierten Singulärwerte einer Matrix $A \in \mathbb{R}^{m \times n}$, $m > n$, genau die Eigenwerte der Matrix $A^T A$ sind und in welcher Beziehung sie mit den Eigenwerten von AA^T stehen. **Hinweis**: hier ist "$m > n$" wichtig.

3.3.6 Python – Laden und Speichern von Arrays

Speichern sie die Covid-Daten aus obiger Tabelle zur späteren Verwendung als ein `numpy.array`. Beispielsweise so:

```
import numpy as np
import matplotlib.pyplot as plt

days = [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]
case = [352, 347, 308, 151, 360, 498, 664, 686, 740, 418, 320,
        681, 691, 1154, 1284,  127, 984, 573, 1078,  1462,   2239,
        2236,  2119,  1663,   1413,    2283,
        2717,  3113,   2972,    3136,    2615]

data = np.vstack([days, case])
print('Shape of data: ', data.shape)

datafilestr = 'coviddata.npy'
np.save(datafilestr, data)

lddata = np.load(datafilestr)

plt.figure(1)
plt.plot(lddata[0, :], lddata[1, :], 's', label='Cases/Day')
plt.title('Covid Faelle in Bayern im Oktober 2020')
plt.legend()
plt.show()
```

3.3.7 Lineare Regression für Covid Daten (P)

Führen sie auf den Covid-daten eine lineare Regression zum Fitten

- einer konstanten Funktion $f(x) = c$
- einer linearen Funktion $f(x) = ax + b$

- einer quadratischen Funktion $f(x) = w_1 + w_2 x + w_3 x^2$

durch. Berechnen Sie die mittlere quadratische Abweichung $\frac{1}{N}\sum_{i=1}^{N} \|y_i - f(x_i)\|^2$ für alle drei Approximationen und plotten Sie die Covid-Daten zusammen mit der aus der Regression erhaltenen Funktion. Beispielsweise so:

```
import numpy as np
import matplotlib.pyplot as plt

datafilestr = 'coviddata.npy'
cvdata = np.load(datafilestr)

# ## Definieren der Basis Funktion(en)


def b_zero(x):
    ''' eine konstante Funktion '''
    return 1.


ybf = cvdata[1, :]  # die y-werte
xbf = cvdata[0, :]  # die x-werte

N = xbf.size   # Anzahl Datenpunkte
ybf = ybf.reshape((N, 1))  # reshape als Spaltenvektor

bzx = [b_zero(x) for x in xbf]  # eine Liste mit Funktionswerten
bzx = np.array(bzx).reshape((N, 1))  # ein Spalten Vektor

Phix = bzx  # hier nur eine Spalte

# Das LGS AtA w = At y
rhs = Phix.T @ ybf
AtA = Phix.T @ Phix

# ACHTUNG: das hier geht nur weil AtA keine Matrix ist in diesem Fall
w = 1./AtA * rhs
# ACHTUNG: das hier ging nur weil AtA keine Matrix ist in diesem Fall


def get_regfunc(weights, basfunlist=[]):
    ''' Eine Funktion, die eine Funktion erzeugt

    Eingang: die Gewichte, eine Liste von Basisfunktionen
    Ausgang: die entsprechende Funktion `f = w_1b_1 + ... `
    '''
```

```
    def regfunc(x):
        fval = 0
        for kkk, basfun in enumerate(basfunlist):
            fval = fval + weights[kkk]*basfun(x)
        return fval

    return regfunc


const_regfunc = get_regfunc([w], [b_zero])

const_approx = [const_regfunc(x) for x in xbf]
const_approx = np.array(const_approx).reshape((N, 1))

plt.figure(1)
plt.plot(cvdata[0, :], cvdata[1, :], 's', label='Cases/Day')
plt.plot(cvdata[0, :], const_approx, '-', label='constant fit')
plt.legend()
plt.show()
```

Hinweise:

- `liste = [f(x) for x in iterable]` ist sehr bequem um Vektoren von Funktionswerten zu erzeugen aber nicht sehr *pythonesque* (und auch im Zweifel nicht effizient). Besser ist es Funktionen zu schreiben, die *vektorisiert* sind. Zum Beispiel können die meisten *built-in* Funktionen wie `np.exp` ein `array` als Eingang direkt in ein `array` der Funktionswerte umsetzen.
- *Eine Funktion, die eine Funktion erzeugt* finde ich sehr hilfreich für viele Anwendungen (ist aber manchmal nicht so gut nachvollziehbar).

3.3.8 Truncated SVD (P+T)

1. **(P)** Berechnen und plotten sie die Singulärwerte einer 4000×1000 Matrix mit zufälligen Einträgen und die einer Matrix mit "echten" Daten (hier Simulationsdaten einer Stroemungssimulation)[2]. Berechnen sie den Fehler der *truncated SVD* $\|A - A_{\hat{r}}\|$ für $\hat{r} = 10, 20, 40$ für beide Matrizen.
2. **(T)** Was lässt sich bezüglich einer Kompression der Daten mittels SVD für die beiden Matrizen sagen. (Vergleichen sie die plots der Singulärwerte und beziehen sie sich auf die gegebene Formel für die Differenz).
3. **(P+T)** Für die "echten" Daten: Speichern sie die Faktoren der bei $\hat{r} = 40$ abgeschnittenen SVD und vergleichen Sie den Speicherbedarf der Faktoren und der eigentlichen Matrix.

Beispielcode:

[2] Download bitte hier – Achtung das sind 370MB

```
import numpy as np
import scipy.linalg as spla
import matplotlib.pyplot as plt

randmat = np.random.randn(4000, 1000)

rndU, rndS, rndV = spla.svd(randmat)

print('U-dims: ', rndU.shape)
print('V-dims: ', rndV.shape)
print('S-dims: ', rndS.shape)

plt.figure(1)
plt.semilogy(rndS, '.', label='Singulaerwerte (random Matrix)')

realdatamat = np.load('velfielddata.npy')

# # Das hier ist eine aufwaendige Operation
rlU, rlS, rlV = spla.svd(realdatamat, full_matrices=False)
# # auf keinen Fall `full_matrices=False` vergessen

print('U-dims: ', rlU.shape)
print('V-dims: ', rlV.shape)
print('S-dims: ', rlS.shape)

plt.figure(1)
plt.semilogy(rlS, '.', label='Singulaerwerte (Daten Matrix)')

plt.legend()
plt.show()
```

Hinweise:

- Es gibt viele verschiedene Normen für Vektoren und Matrizen. Sie dürfen einfach mit `np.linalg.norm` arbeiten. Gerne aber mal in die Dokumentation schauen *welche* Norm berechnet wird.
- Die (T) Abschnitte hier bitte mit den anderen (T) Aufgaben oder als Bildschirmausgabe im Programm.