ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mentorierte Arbeit in Fachdidaktik Mathematik

Eigengesichter

Oliver Rietmann

Inhalt
In dieser Arbeit wird die Technik der *Eigengesichter* (engl. *eigenfaces*) erklärt. Hierbei handelt es sich um eine rudimentäre Methode zur Gesichtserkennung, welche durch Computer automatisiert werden kann. Unter Gesichtserkennung versteht man klassischerweise das (automatisierte) identifizieren einer Person auf einem Foto. Aber auch damit verwandte Aufgaben werden hier besprochen. Diese Arbeit führt die Leser durch eine Anleitung zur Implementierung eines solchen Programms in Python. Der Fokus liegt dabei auf der zugrundeliegenden Mathematik dieses Verfahrens. Die dazu verwendeten Unterrichtsmethoden sollen zudem aus didaktischer Sicht beleuchtet werden.

Zielpublikum
Fortgeschrittene gymnasiale MittelschülerInnen mit Schwerpunkt Mathematik und StudentInnen einer mathematischen/technischer Fachrichtung.

Voraussetzungen
Vertrautheit mit den Grundbegriffen der linearen Algebra (Vektoren, Matrizen, Basis, Linearkombination, Unterräume von $\mathbb{R}^n$). Zudem werden grundlegende Programmierkenntnisse vorausgesetzt.

Form
Text mit Aufgaben, Lösungen und Lernzielen. Die Python-Codes befinden sich im Anhang. Alternativ können sie online heruntergeladen werden. (Der Link befindet sich in der Arbeit.)

Betreuung
Christian Rüede

Datum
19. Oktober 2021

Inhaltsverzeichnis

I Lernskript: Eigengesichter **2**

1 Datenbank einrichten 3

2 Vom Bild zum Vektor 4

3 Durchschnittsgesicht und Differenzgesichter 8

4 Eigengesichter 13

5 Projektion auf die Eigengesichter 18

6 Bildkompression 22

7 Gesichtserkennung 25

II Didaktische Methoden **29**

1 Interleaved Practice 29

2 Scaffolding 30

3 Lernziele 31

4 Gruppenarbeit: Erstellung der Datenbank 31

5 Problem basiertes Lernen 33

6 Vom $\mathbb{R}^3$ in den $\mathbb{R}^n$ 33

7 Holistic mental model confrontation 34

Teil I
Lernskript: Eigengesichter

Historisch gesehen war die Methode der Eigengesichter das erste durch Computer automatisierbare Verfahren zur Gesichtserkennung. Das Fundament dazu wurde 1987 durch Sirovich and Kirby entwickelt [15]. In ihrer Arbeit verwendeten Sie bereits den Begriff *eigenpictures*. Sie zeigten auf, wie man Fotos von Gesichtern geeignet durch Begriffe der linearen Algebra beschreiben kann. Damit war der Weg frei um die mächtige Maschinerie der Mathematik, insbesondere der linearen Algebra und der Statistik, auf Fotos anzuwenden. Dies geschah schon kurz darauf, nämlich 1991, durch Turk und Pentland [19]. Sie verwendeten bereits den Begriff *eigenfaces* und beschrieben das Verfahren zur Gesichtserkennung, welches auch wir implementieren werden. Modernere Methoden wie zum Beispiel *DeepFace* von Facebook funktionieren allerdings viel besser als unsere rudimentäre Methode und sind so gut wie echte Menschen bei der Gesichtserkennung [17]. Ein Vergleich der (heutzutage) besten Methoden findet man zum Beispiel hier [18].

Alle solchen Programme, auch die modernsten, verwenden eine *Datenbank* von Bildern von Gesichtern, deren Identität bereits bekannt ist. Man hat also eine gewisse Anzahl von Personen. Jeder einzelnen Person sind mehrere Bilder zugeordnet, nämlich die Bilder, welche das Gesicht eben dieser Person zeigen. Man kann sich das als Unterteilung in verschiedene Klassen vorstellen: Zu jeder Person gehört eine Klasse und jede Klasse enthält eine Menge von Bildern. Dies ist in Abbildung 2 veranschaulicht. Aus dieser Datenbank

Klasse (Person)	**Bild 1**	**Bild 2**	**Bild 3**	**Bild 4**	⋯
Adam Sandler					⋯
Emma Watson					⋯
Natalie Portman					⋯
⋮	⋮	⋮	⋮	⋮	⋱

Abbildung 1: Visualisierung einer Datenbank von Bildern von Gesichtern.

„lernt“ das Programm, neue Bilder zu klassifizieren, also den Personen aus der Datenbank zuzuordnen. Das Wort „neu“ bedeutet hier, dass dieses Bild nicht notwendigerweise in der Datenbank enthalten ist. Die Person auf dem Bild muss aber in der Datenbank sein! Würde die Datenbank in Abbildung 1 wirklich nur diese drei Personen enthalten, so könnte man zum Beispiel kein Bild von Brad Pitt korrekt klassifizieren, auch wenn eine noch so gute Methode verwendet wird. Die Datenbank und die Methode der Gesichtserkennung sind unabhängig voneinander. Das heisst einerseits, aus der selben Datenbank können verschiedene Methoden lernen. Andererseits kann ein und die selbe Methode verschiedene

Datenbanken nutzen. Wie gut die Gesichtserkennung am Schluss funktioniert hängt nicht nur von der Methode selbst ab, sondern auch von der Datenbank, welche diese verwendet. Grundsätzlich gilt, dass jede Methode umso besser funktioniert, je mehr Bilder pro Person in der Datenbank gespeichert sind, die sie verwendet. Mit „gut funktionieren“ ist gemeint, dass neue Bilder mit hoher Wahrscheinlichkeit richtig klassifiziert werden.

Die in Abbildung 1 gezeigten Bilder stammen aus einer Datenbank von über 10'000 Bildern von über 100 berühmten Persönlichkeiten [5]. Die Bilder sind alle schwarz-weiss und zeigen lediglich die Gesichter der Personen. Genau diese Datenbank werden wir auch für alle nachfolgenden Kapitel verwenden. Sie ist unter folgendem Link zu finden:

`https://people.math.ethz.ch/~rioliver/eigenfaces/datenbank.zip`

Allerdings kann auch eine andere Datenbank verwendet werden, sofern sie in das richtige Format gebracht wird. Eine Anleitung dazu befindet sich in Kapitel 1.

Das Grundgerüst eines Programms zur Gesichtserkennung in Python steht uns schon zur Verfügung. Wir werden dieses in den folgenden Kapiteln zu einem voll funktionsfähigen Programm erweitern. Der gesamte Code befindet sich auf GitHub und kann unter folgendem Link heruntergeladen werden:

`https://github.com/OliverRietmann/eigenfaces`

1 Datenbank einrichten

Bitte laden Sie nun die Datenbank herunter (mit dem Link aus der Einleitung) und entpacken Sie diese. Wir nennen das entstehende Verzeichnis `datenbank`. Laden Sie ebenfalls die Python-Codes herunter. Führen Sie als erstes das Python-Skript `save_eigenfaces.py` über die Kommandozeile (Linux, MacOS) oder über IPython (Windows) aus und übergeben Sie dabei den Pfad zur Datenbank und einen Namen für das resultierende File (z.B. `eigenfaces.dat`), also

```
python3 save_eigenfaces.py datenbank eigenfaces.dat
```

Dies generiert ein File `eigenfaces.dat` welches wir später brauchen um die Aufgaben zu lösen. Damit lassen sich alle nachfolgenden Kapitel bearbeiten. Alternativ kann man eine eigene Datenbank erstellen und stattdessen diese für die Bearbeitung des Lernskriptes verwenden. Dieses Kapitel ist eine Anleitung dazu. Es werden folgende Programme benötigt:

- Ein Bildbearbeitungsprogramm wie zum Beispiel das open source Programm GIMP.
- Ein Tabellen-Kalkulationsprogramm wie zum Beispiel das open source Programm LibreOffice Calc.

Nehmen wir als Beispiel eine Schulklasse bestehend aus 10 Lernenden. Jeder Schüler und jede Schülerin macht einige Fotos von seinem/ihrem Gesicht. Sagen wir, 8 Fotos pro Person. Das Ziel ist ein Verzeichnis `datenbank` anzulegen, welches wie in Abbildung 2 aufgebaut ist. Das heisst, pro Person enthält `datenbank` je ein Verzeichnis mit deren Namen. Diese Verzeichnisse enthalten wiederum die 8 Fotos der jeweiligen Person. Die Dateien mit der Endung `.csv` enthalten jeweils eine Tabelle und können mit einem Tabellenkalkulationsprogramm erstellt/editiert werden.

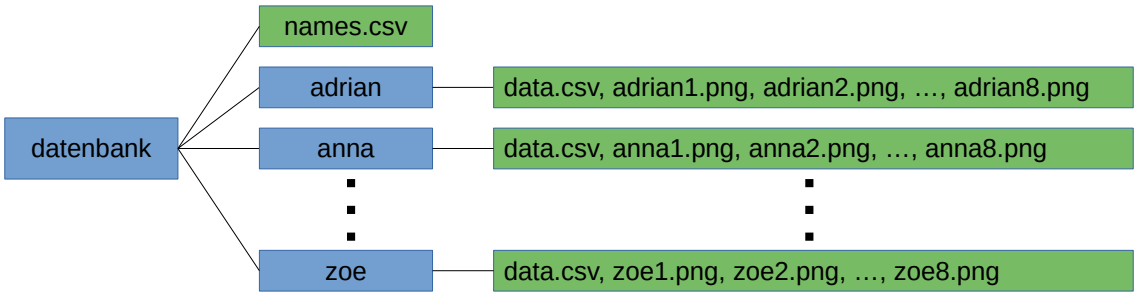


Abbildung 2: Datenbank aus Trainingsbildern. Die Verzeichnisse sind in blau und die Dateien in grün abgebildet.

- `names.csv`: Besteht aus 10 Zeilen, welche die 10 Unterverzeichnisse `adrian, anna, ..., zoe` von `datenbank` auflisten. Unser Programm benötigt dieses File um die Unterverzeichnisse zu finden.
- `data.csv`: Besteht aus 8 Zeilen, welche die Dateinamen der 8 Fotos auflisten. Zum Beispiel enthält `adrian/data.csv` die Zeilen `adrian1.png, adrian2.png, ..., adrian8.png`. Unser Programm benötigt dieses File um die Bilddateien in den Unterverzeichnissen zu finden.

Es fehlt noch ein letzter Schritt. Die Bilder müssen noch in ein geeignetes Format gebracht werden. Dazu verwenden wir ein Bildbearbeitungsprogramm.

- Alle Bilder müssen auf die selbe Auflösung zugeschnitten werden. Eine geeignete Auflösung ist zum Beispiel eine Breite von $N = 144$ Pixel und eine Länge von $M = 180$ Pixel.
- Der vorherige Punkt sollte so bewerkstelligt werden, dass das Gesicht möglichst das ganze Bild ausfüllt.
- Alle Bilder müssen in ein schwarz-weiss Bild konvertiert werden.

Führen Sie dann die Schritte zu Beginn dieses Kapitels für diese Datenbank aus.

2 Vom Bild zum Vektor

Lernziele Kapitel 2

2.1 Die Lernenden können für niedrige Auflösung ($M \cdot N < 10$) die Umrechnungen zwischen Bild, Matrix und Vektor von Hand ausführen.
(Aufgabe 2.1 und 2.2)

2.2 Die Lernenden können in Python die Einträge von Vektoren und Matrizen auslesen und verändern.
(Aufgabe 2.3 und 2.4)

Der erste Schritt besteht darin, Bilder als Vektoren aufzufassen. Das hat zwei Gründe: Erstens können wir diese nur so geeignet in Python darstellen und manipulieren. Zweitens

erlaubt uns das, Bilder in den Kontext der linearen Algebra zu bringen um deren mächtige Methoden anzuwenden. Als Beispiel betrachten wir ein Bild der Auflösung $M = 180$ Pixel (Höhe) mal $N = 144$ Pixel (Breite), wie in Abbildung 3. Jedem Pixel wird nun eine reelle Zahl zwischen 0 und 1 zugeordnet. Dabei bedeutet 0, dass das Pixel schwarz ist und 1 bedeutet, dass es weiss ist. Die reellen Zahlen dazwischen beschreiben die Graustufen. Wir Nummerieren diese Pixel mit zwei Indices (m, n), wobei $1 \leq m \leq M$ und $1 \leq n \leq N$. Zum Beispiel entspricht $(1, N)$ dem Pixel in der oberen rechten Ecke des Bildes. Diesem Pixel wird also eine Zahl p_{mn} zugeordnet, wobei $0 \leq p_{mn} \leq 1$. Das gibt uns eine $M \times N$-Matrix deren Einträge gerade die p_{mn} sind. So können wir also ein schwarz-weiss Bild als Matrix auffassen. Nun schreiben wir die Spalten dieser Matrix in einen Vektor wie in Abbildung 3 gezeigt. Damit erhalten wir eine eindeutige Korrespondenz zwischen schwarz-weiss Bilder der Auflösung $M \times N$ und Vektoren der Länge $M \cdot N$ mit Einträgen zwischen 0 und 1. Für diesen Schritt ist es egal ob das Bild ein Gesicht zeigt oder etwas anderes.

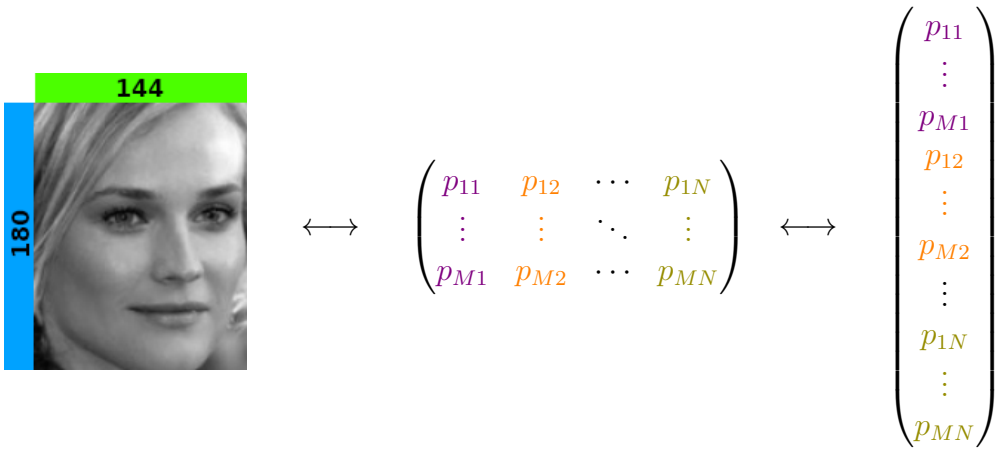


Abbildung 3: Ein schwarz-weiss Bild kann als Matrix oder Vektor aufgefasst werden.

Aufgabe 2.1

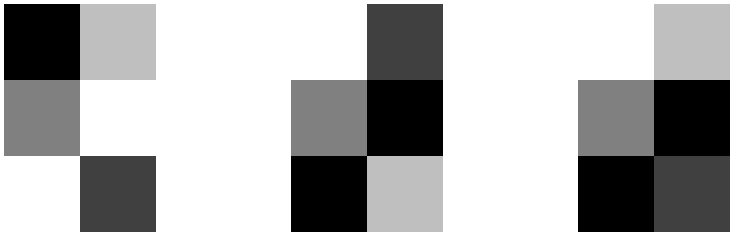
Man betrachte das schwarz-weiss Bild, welches durch folgende Matrix beschrieben ist.

$$\begin{pmatrix} 1 & \frac{1}{4} \\ \frac{1}{2} & 0 \\ 0 & \frac{3}{4} \end{pmatrix}$$

(a) Welche Werte für M und N beschreiben die Auflösung dieses Bildes?

(b) Wie sieht der Vektor aus, der dieses Bild beschreibt?

(c) Welches der folgenden drei Bilder entspricht dieser Matrix?



Lösung

Die Lösung der ersten beiden Teilaufgaben kann von Abbildung 3 abgelesen werden. Für die letzte Teilaufgabe erinnern wir uns, dass die Zahlen zwischen 0 und 1 fliessend den Graustufen von Schwarz (Null) bis Weiss (Eins) entsprechen.

(a) Die Auflösung ist $M = 3$ mal $N = 2$ Pixel.

(b) Der Vektor ist gegeben durch

$$\begin{pmatrix} 1 \\ \frac{1}{2} \\ 0 \\ \frac{1}{4} \\ 0 \\ \frac{3}{4} \end{pmatrix}.$$

(c) Das mittlere Bild entspricht der Matrix.

Aufgabe 2.2

Man betrachte den Vektor

$$\begin{pmatrix} \frac{3}{4} \\ 0 \\ \frac{1}{4} \\ 0 \\ \frac{1}{2} \\ 1 \end{pmatrix}.$$

Dieser soll ein Bild der Auflösung $M = 2$ und $N = 3$ beschreiben.

(a) Schreiben Sie die entsprechende Matrix gemäss Abbildung 3 auf.

(b) Zeichnen Sie das entsprechende schwarz-weiss Bild, analog zu Teil (c) in Aufgabe 2.1.

Lösung

(a) Die Matrix hat nun $M = 2$ Zeilen und $N = 3$ Spalten. Nach Abbildung 3 erhalten wir

$$\begin{pmatrix} \frac{3}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{2} & 1 \end{pmatrix}.$$

(b) Das entsprechende schwarz-weiss Bild lässt sich von obiger Matrix ablesen:

Natürlich können Sie nicht die exakten Graustufen wiedergeben. Aber ihr Bild

sollte vier verschiedene Graustufen enthalten, die richtig auf die Pixel verteilt sind.

In unserem Python Code ist die Funktion, welche eine $M \times N$ Matrix auf diese Weise in einen Vektor der Länge $M \cdot N$ überführt, bereits implementiert. Sie befindet sich im File `chapter2.py` und heisst `matrix_to_vector`. Wir betrachten diese nun etwas genauer, um die Manipulation von Matrizen und Vektoren in Python zu lernen.

```
def vector_to_matrix(v, M, N):
    P = np.zeros((M, N))
    for m in range(M):
        for n in range(N):
            P[m, n] = v[n + N * m]
    return P
```

./codes/solution/chapter2.py

Das Argument `P` ist eine `M` mal `N` Matrix und besteht aus den Einträgen p_{mn} wie oben. Auf die Einträge von Vektoren und Matrizen kann über die eckigen Klammern [...] zugegriffen werden. Wir brauchen aber auch die Umkehrung dieser Operation. Das ist der Zweck folgender Übung.

Aufgabe 2.3

Ergänzen Sie im File `chapter2.py` die Funktion `vector_to_matrix(v, M, N)`. Dabei ist `v` ein Vektor der Länge `M * N` wie oben. Die Funktion soll die zu `v` gehörende Matrix zurück geben. Sie können die ihre Lösung überprüfen indem Sie das Skript `chapter2.py` laufen lassen.

Lösung

Bei einer richtigen Lösung sollte das Skript `chapter2.py` das Foto aus Abbildung 3 generieren. Die Lösung könnte zum Beispiel so aussehen:

```
def matrix_to_vector(P, M, N):
    v = np.zeros(M * N)
    for m in range(M):
        for n in range(N):
            v[n + N * m] = P[m, n]
    return v
```

./codes/solution/chapter2.py

Man kann gewisse Effekte in einem Bild erzeugen, indem man den zugehörigen Vektor manipuliert und anschliessend wieder als Bild darstellt. Zum Beispiel kann man das Negativ eines gegebenen schwarz-weiss Bildes generieren. Dazu nimmt man den Vektor $\vec{p}$ der Länge $M \cdot N$, welcher dieses Bild darstellt, und definiert damit einen neuen Vektor wie folgt

$$\vec{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{M \cdot N} \end{pmatrix} \quad \longrightarrow \quad \begin{pmatrix} 1 - p_1 \\ 1 - p_2 \\ \vdots \\ 1 - p_{M \cdot N} \end{pmatrix}.$$

Der Vektor auf der rechten Seite entspricht dem Negativ des ursprünglichen Bildes. Wie dieses Bild genau aussieht, sehen wir in der nächsten Aufgabe.

Aufgabe 2.4

Ergänzen Sie im Skript `chapter2.py` die Funktion `get_negative.py`. Diese soll zu einem gegeben Vektor `p` mit Einträgen zwischen 0 und 1 den Vektor des entsprechenden Negativs zurückgeben, wie oben beschrieben. Lassen Sie das Skript `chapter2.py` laufen um das entsprechende Bild auszugeben und um Ihre Lösung zu überprüfen.

Lösung

Links ist eine mögliche Implementierung gezeigt. Rechts ist ein Bild und dessen Negativ.

```python
def get_negative(p):
    MN = len(p)
    for i in range(MN):
        p[i] = 1.0 - p[i]
    return p
```

./codes/solution/chapter2.py

3 Durchschnittsgesicht und Differenzgesichter

Lernziele Kapitel 3

3.1 Die Lernenden können den Durchschnitt einer Familie von Vektoren von Hand berechnen (für konkrete Zahlenbeispiele).
(Aufgaben 3.1 und 3.2)

3.2 Die Lernenden können die Addition und Subtraktion von Vektoren und deren Multiplikation mit einem Skalar in Python ausführen.
(Aufgaben 3.3 und 3.4)

3.3 Die Lernenden können die Begriffe Durchschnittsgesicht und Differenzgesicht in Worten und Bildern erklären.
(Aufgaben 3.5 und 3.6)

Wir haben im letzten Kapitel gesehen, dass man Bilder der Auflösung $M \times N$ als Vektoren der Länge $M \cdot N$ auffassen kann. Diese Vektoren können wir wiederum als Punkte in einem Raum der Dimension $M \cdot N$ auffassen. In diesem Kapitel werden wir diese Sichtweise ausarbeiten.

Sei K die Anzahl aller Bilder unserer Datenbank. Jedes Bild soll dabei die Auflösung $M \times N$ haben. Weiter seien $\vec{b}_1, \ldots, \vec{b}_K$ die Vektoren dieser Bilder. Diese Darstellung erlaubt uns, das *Durchschnittsgesicht*, wir nennen es $\vec{m}$, zu definieren

$$\vec{m} = \frac{1}{K}\left(\vec{b}_1 + \ldots + \vec{b}_K\right).$$

Dies ist einfach die Summe der Vektoren dividiert durch deren Anzahl. Wegen dieser Analogie zum arithmetischen Mittel, nennen wir dies das Durchschnittsgesicht. Damit

können wir die sogenannten *Differenzgesichter* $\vec{a}_1, \ldots, \vec{a}_K$ bilden. Diese sind definiert als als die Verschiebung der Gesichter aus der Datenbank um $-\vec{m}$, also

$$\vec{a}_k = \vec{b}_k - \vec{m}, \quad k = 1, \ldots, K.$$

Um den Durchschnitt und die Verschiebung von Vektoren geometrisch besser zu verstehen, schauen wir das zuerst in der Ebene an.

Aufgabe 3.1

Betrachten Sie die folgenden drei Vektoren

$$\vec{b}_1 = \begin{pmatrix} -4 \\ -5 \end{pmatrix}, \quad \vec{b}_2 = \begin{pmatrix} 9 \\ -5 \end{pmatrix}, \quad \vec{b}_3 = \begin{pmatrix} 1 \\ 7 \end{pmatrix}.$$

(a) Berechnen Sie den Durchschnitt $\vec{m}$ der Vektoren $\vec{b}_1, \vec{b}_2, \vec{b}_3$.

(b) Berechnen Sie die um $-\vec{m}$ verschobenen Vektoren

$$\vec{a}_1 = \vec{b}_1 - \vec{m}, \quad \vec{a}_2 = \vec{b}_2 - \vec{m}, \quad \vec{a}_3 = \vec{b}_3 - \vec{m}.$$

(c) Zeichnen Sie alle Vektoren $\vec{a}_1, \vec{a}_2, \vec{a}_3$ und $\vec{b}_1, \vec{b}_2, \vec{b}_3$, sowie $\vec{m}$ in ein Koordinatensystem.

Lösung

(a) Wie im skalaren Fall ist der Durchschnitt definiert als die Summe geteilt durch dir Anzahl der Summanden, also

$$\vec{m} = \frac{1}{3}\left(\vec{b}_1 + \vec{b}_2 + \vec{b}_3\right) = \frac{1}{3}\left(\begin{pmatrix} -4 \\ -5 \end{pmatrix} + \begin{pmatrix} 9 \\ -5 \end{pmatrix} + \begin{pmatrix} 1 \\ 7 \end{pmatrix}\right) = \begin{pmatrix} 2 \\ -1 \end{pmatrix}.$$

(b) Für die um $-\vec{m}$ verschobenen Vektoren erhalten wir folglich

$$\vec{a}_1 = \begin{pmatrix} -6 \\ -4 \end{pmatrix}, \quad \vec{a}_2 = \begin{pmatrix} 7 \\ -4 \end{pmatrix}, \quad \vec{a}_3 = \begin{pmatrix} -1 \\ 8 \end{pmatrix}.$$

(c) Die Skizze ist in Abbildung 4 dargestellt. Gezeichnet sind die Vektoren aufgefasst als Punkte, also

$$\vec{m} = \overrightarrow{OM}, \qquad \vec{a}_k = \overrightarrow{OA_k}, \qquad \vec{b}_k = \overrightarrow{OB_k}, \qquad k = 1, 2, 3.$$

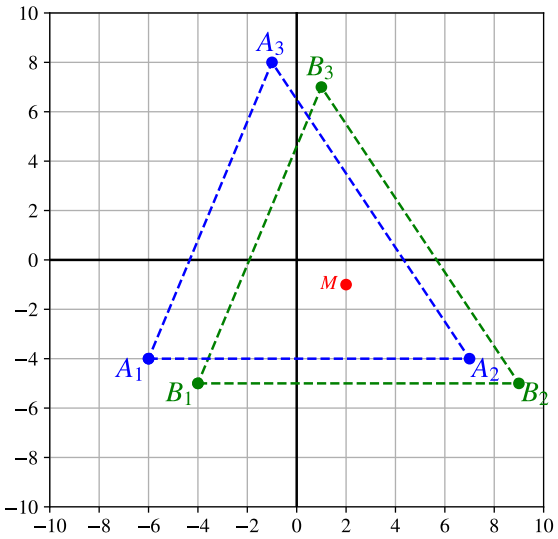


Abbildung 4: Durchschnitt $\vec{m}$ der Vektoren $\vec{b}_1, \vec{b}_2, \vec{b}_3$ sowie deren Translationen $\vec{a}_1, \vec{a}_2, \vec{a}_3$ um $-\vec{m}$, alle aufgefasst als Punkte in der Ebene.

Aufgabe 3.2

Betrachten Sie die Vektoren

$$\vec{b}_1 = \begin{pmatrix} -2 \\ -1 \\ 1 \end{pmatrix}, \qquad \vec{b}_2 = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}.$$

(a) Berechnen Sie den Durchschnitt $\vec{m}$ der Vektoren $\vec{b}_1$ und $\vec{b}_2$.

(b) Berechnen Sie die um $-\vec{m}$ verschobenen Vektoren

$$\vec{a}_1 = \vec{b}_1 - \vec{m} \qquad \text{und} \qquad \vec{a}_2 = \vec{b}_2 - \vec{m}.$$

(c) Was ist der Durchschnitt der Vektoren $\vec{a}_1$ und $\vec{a}_2$? Was ist der Durchschnitt der Vektoren $\vec{a}_1, \vec{a}_2, \vec{a}_3$ aus Teilaufgabe 3.1 (b)?

Lösung

(a) Wir berechnen die Summe der beiden Vektoren und dividieren durch deren Anzahl

$$\vec{m} = \frac{1}{2}\left(\vec{b}_1 + \vec{b}_2\right) = \frac{1}{2}\begin{pmatrix} -2 \\ -1 \\ 1 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}.$$

(b) Für die um $-\vec{m}$ verschobenen Vektoren erhalten wir folglich

$$\vec{a}_1 = \begin{pmatrix} -3 \\ 0 \\ 0 \end{pmatrix}, \qquad \vec{a}_2 = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}.$$

(c) Der Durchschnitt der verschobenen Vektoren ist

$$\frac{1}{2}\left(\vec{a}_1 + \vec{a}_2\right) = \frac{1}{2}\begin{pmatrix}-3\\0\\0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}3\\0\\0\end{pmatrix} = \begin{pmatrix}0\\0\\0\end{pmatrix}.$$

Wenn man eine Familie von Vektoren um ihren Durchschnitt verschiebt, haben die resultierenden Vektoren immer den Nullvektor als Durchschnitt (warum?). Folglich gilt das auch für die Vektoren aus Teilaufgabe 3.1 (b).

Nun wollen wir das Durchschnittsgesicht

$$\vec{m} = \frac{1}{K}\left(\vec{b}_1 + \ldots + \vec{b}_K\right)$$

der Bilder aus der Datenbank berechnen und wieder als Bild ausgeben. Wie sieht so ein Durchschnittsgesicht aus? Das werden wir in folgender Übung herausfinden.

Aufgabe 3.3

Ergänzen Sie im File `chapter3.py` die Funktion `meanface(b_list)`. Dabei ist `b_list` die Liste der Länge K der Vektoren $\vec{b}_1, \ldots, \vec{b}_K$. Der Rückgabewert soll das Durchschnittsgesicht $\vec{m}$ sein. Sie können die ihre Lösung überprüfen indem Sie das Skript `chapter3.py` laufen lassen. Dabei müssen Sie auf der Kommandozeile den Pfad zur Datenbank übergeben. *Hinweis:* Die Python Funktionen `len(...) und sum(...)` können nützlich sein.

Lösung

Hier ist eine mögliche Lösung und das davon mit `chapter3.py` generierte Durchschnittsgesicht.

```
1 def meanface(b_list):
2     K = len(b_list)
3     return sum(b_list) / K
```

./codes/solution/chapter3.py

Durchschnittsgesicht:

Nun berechnen wir die *Differenzgesichter* $\vec{a}_1, \ldots, \vec{a}_K$, also die Gesichter der Datenbank, welche um $-\vec{m}$ verschoben wurden

$$\vec{a}_k = \vec{b}_k - \vec{m}, \quad k = 1, \ldots, K.$$

Aufgabe 3.4

Ergänzen Sie im File `chapter3.py` die Funktion `diffface(b_list)`. Dabei ist `b_list` die Liste der Länge K der Vektoren $\vec{b}_1, \ldots, \vec{b}_K$. Der Rückgabewert soll eine Liste der Länge K sein, welche die Differenzgesichter $\vec{a}_1, \ldots, \vec{a}_K$ gemäss obiger Formel enthält. Sie können die ihre Lösung überprüfen indem Sie das Skript `chapter3.py` laufen lassen. Dabei müssen Sie auf der Kommandozeile den Pfad zur Datenbank übergeben.

Hinweis: Verwenden Sie die Funktion `meanface(...)` aus der vorherigen Aufgabe.

Lösung

Eine korrekte Lösung könnte so aussehen.

```python
def difffaces(b_list):
    a_list = np.zeros_like(b_list)
    m = meanface(b_list)
    K = len(b_list)
    for k in range(K):
        a_list[k] = b_list[k] - m
    return a_list
```

./codes/solution/chapter3.py

Die eben eingeführten Begriffe sind in Abbildung 5 für Vektoren mit zwei Komponenten visualisiert. Die Quintessenz ist, dass sich diese Vektoren (und damit die Bilder) als Punkte in einem Raum der Dimension $M \cdot N$ auffassen lassen. Wir bezeichnen diese Punkte mit Grossbuchstaben, also (O bezeichnet hier den Ursprung)

$$\vec{m} = \overrightarrow{OM}, \qquad \vec{a}_k = \overrightarrow{OA_k}, \qquad \vec{b}_k = \overrightarrow{OB_k}, \qquad k = 1, \ldots, K.$$

Die Bilder der Datenbank bilden dann eine „Punktewolke“ in diesem Raum. Durch Subtraktion von $\vec{m}$ wir diese Punktewolke um den Ursprung zentriert. Das Resultat dieser Subtraktion (oder Verschiebung) sind die Differenzgesichter. Das ist in Abbildung 5 gezeigt.

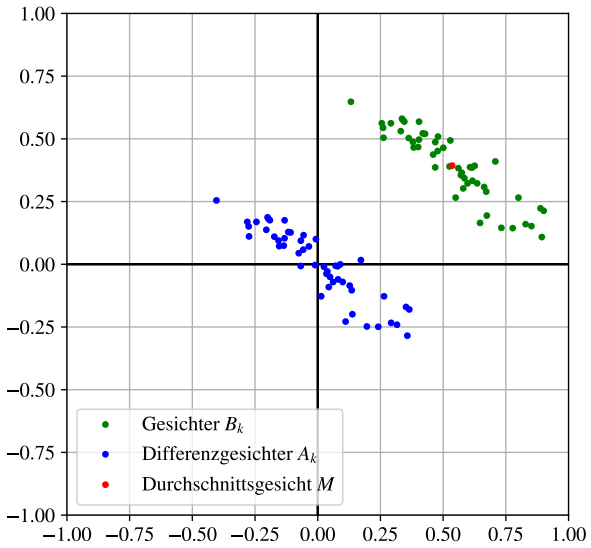


Abbildung 5: Die Gesichter werden um den Ursprung zentriert indem man das Durchschnittsgesicht subtrahiert.

Aufgabe 3.5

Wir haben in Aufgabe 3.3 das Durchschnittsgesicht wieder als Bild dargestellt. Könn-

te man das auch mit den Differenzgesichtern machen? Begründen Sie.

Lösung

Nein, die Differenzgesichter lassen sich nicht wieder als Bilder darstellen, weil die Einträge dieser Vektoren nicht zwischen 0 und 1 liegen. Dies sieht man zum Beispiel in Abbildung 5. Man kann also diesen Vektor-Einträgen keine Graustufe zuordnen.

Aufgabe 3.6

Nennen Sie einen Unterschied und eine Gemeinsamkeit der vereinfachten Darstellung in Abbildung 5 zu unserer tatsächlichen Situation mit Bildern von Gesichtern. Gehen Sie davon aus, dass unsere Bilder eine Auflösung von $M = 180$ und $N = 144$ haben, wie im letzten Kapitel.

Lösung

Gesichter können wir als Punkte in einem Raum der Dimension $M \cdot N$ auffassen. Für $M = 180$ und $N = 144$ wäre diese Dimension 25920 und nicht etwa 2 wie in der Abbildung. Anders ausgedrückt zeigt die Abbildung den Spezialfall $M \cdot N = 2$. Das entspricht Bilder die nur aus zwei Pixeln bestehen. Andererseits wird in der Abbildung korrekt gezeigt, dass die Komponenten der Gesichts-Vektoren $\vec{b}_k$ nur Werte zwischen 0 und 1 annehmen. Zudem sind die Differenzgesichter richtigerweise genau als Verschiebung der Gesichts-Vektoren um $-\vec{m}$ dargestellt.

Die Eigengesichter werden nun aus den Differenzgesichtern konstruiert. Dies wird im nächsten Kapitel behandelt.

4 Eigengesichter

Lernziele Kapitel 3

4.1 Die Lernenden können in Worten und Bildern die Konstruktion der Eigengesichter erklären.
(Aufgaben 4.1 und 4.2)

4.2 Die Lernenden können den Abstand eines Punktes von einer Gerade in auch höheren Dimensionen berechnen.
(Aufgaben 4.3, 4.4 und 4.5)

Die Eigengesichter werden nun aus den Differenzgesichtern $\vec{a}_1, \ldots, \vec{a}_K$ konstruiert. Um sich das Bildlich vorzustellen, betrachten wir diese Vektoren als Punkte $A_1, \ldots A_K$, wobei

$$\vec{a}_k = \overrightarrow{OA_k}, \qquad k = 1, \ldots, K.$$

Wie lesen also $\vec{a}_k$ als Vektor vom Ursprung O zum Punkt A_k. Nun stelle man sich die Differenzgesichter als die „Wolke“ von Punkten $A_1, \ldots A_K$ vor, wie links in Abbildung 6. Ausgehend davon konstruieren wir die Eigengesichter.

Schritt 1 Entlang einer gewissen Richtung weist diese Wolke die grösste Streuung auf. Entlang dieser grössten Streuung wählen wir einen Vektor $\vec{u}_1$ der Länge 1.

Schritt 2 Unter allen Vektoren die orthogonal zu $\vec{u}_1$ sind, wählen wir wieder einen, der in Richtung der grössten Streuung der Wolke zeigt. Diesen nennen wir $\vec{u}_2$ und er soll wieder Länge 1 haben.

Schritt 3 Unter allen Vektoren die orthogonal zu $\vec{u}_1$ und $\vec{u}_2$ sind, wählen wir wieder einen, der in Richtung der grössten Streuung der Wolke zeigt. Diesen nennen wir $\vec{u}_3$ und er soll wieder Länge 1 haben.

Schritt 4 Analog konstruieren wir $\vec{u}_4, \vec{u}_5, \ldots, \vec{u}_K$.

Die Vektoren $\vec{u}_1, \ldots, \vec{u}_K$ heissen *Eigengesichter*. Die genaue Berechnung dieser Vektoren ist nicht so einfach und ist darum schon implementiert. Man kann das ganze so auffassen: Mit diesem Verfahren „lernt“ man die Eigengesichter aus der Datenbank. Das Ganze ist links in Abbildung 6 stark vereinfacht visualisiert.

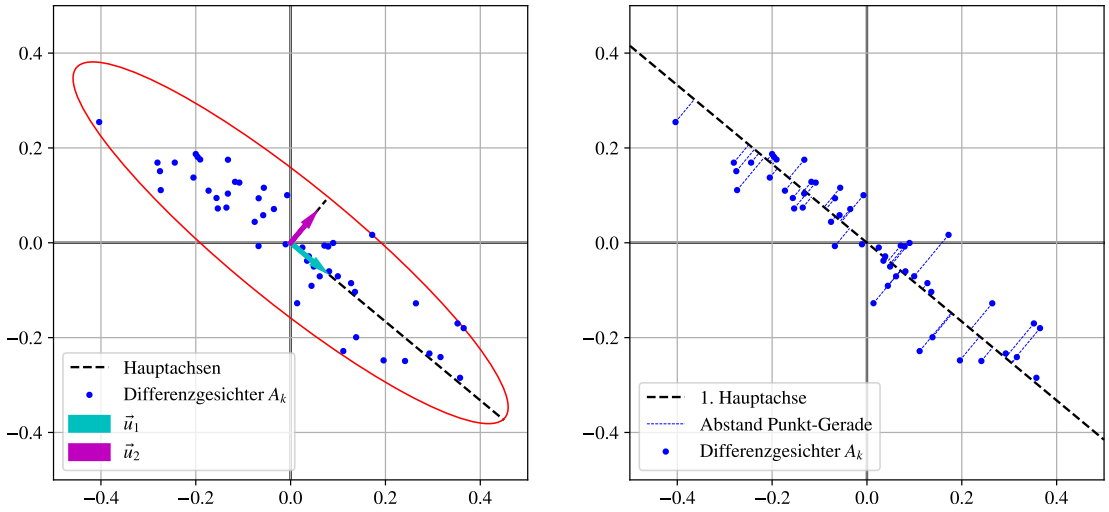


Abbildung 6: Die Eigengesichter sind die orthonormalen Vektoren entlang den Hauptachsen. Die Eigengesichter $\vec{u}_1$ und $\vec{u}_2$ hätten eigentlich Länge 1, sind aber hier der Übersicht wegen zu kurz abgebildet.

Aufgabe 4.1

Links in Abbildung 6 sehen wir die Konstruktion der Einfachheit halber nur in der Ebene. Nennen Sie zwei Unterschiede zwischen dieser vereinfachten Abbildung und unserer Situation. Nehmen Sie dabei an, dass wir Bilder der Auflösung $M \times N$ mit $M = 180$ und $M = 144$ betrachten.

Lösung

Die Abbildung entspricht dem Spezialfall $M \cdot N = 2$. Im unserer Situation ist aber $M \cdot N = 180 \cdot 144 = 25920$. Das heisst, alle involvierten Vektoren haben 25920 Komponenten nur nicht nur 2. Zudem sind in der Abbildung nur zwei Eigengesichter $\vec{u}_1$ und $\vec{u}_2$ gezeigt, da es in der Ebene nicht mehr als zwei Vektoren geben kann, die alle paarweise orthogonal sind. In unserem Fall, also in einem Raum der Dimension

25920, kann es 25920 paarweise orthogonale Vektoren geben. Aber auch dann werden nur so viele Eigengesichter konstruiert, wie wir Gesichter in der Datenbank haben (deren Anzahl haben wir mit K bezeichnet).

Aufgabe 4.2

In der obigen Konstruktion der Eigengesichter muss man in jedem Schritt „einen“ Vektor wählen, der folgende Bedingungen erfüllt:

(a) Er hat Länge 1.

(b) Er ist orthogonal zu den bereits konstruierten Vektoren.

(c) Er zeigt in Richtung der grössten Streuung (unter allen Vektoren die Bedingung (b) erfüllen).

Warum das Wort „einen“? Gibt es denn Mehrere?

Lösung

Ja es gibt Mehrere. Immer wenn ein Vektor $\vec{u}$ die Bedingungen (a), (b) und (c) erfüllt, so gilt das auch für $-\vec{u}$.

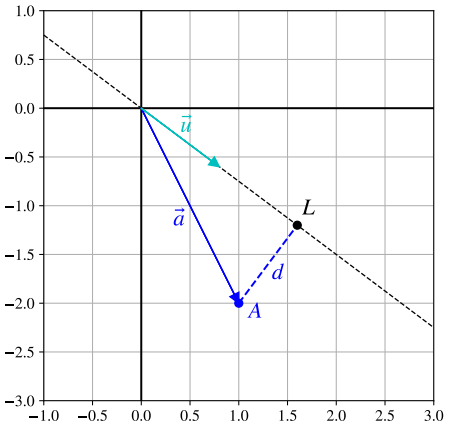
Wir werden nun das erste Eigengesicht $\vec{u}_1$ berechnen. Dazu müssen wir zuerst verstehen was es bedeutet, einen Vektor „entlang der grössten Streuung“ zu finden. Die erste Hauptachse wird wie folgt bestimmt: Sie ist diejenige Gerade, welche die Summe der Abstandsquadrate zu den Punkten $A_1, \ldots, A_K$ minimiert. Dies ist rechts in Abbildung 6 gezeigt. Um die Eigengesichter zu berechnen, müssen wir also den Abstand eines Punktes von einer Geraden berechnen können.

Aufgabe 4.3

Berechnen Sie den Abstand des Punktes A von der Geraden durch Null in Richtung $\vec{u}$, wobei $\vec{a} = \overrightarrow{OA}$ und

$$\vec{a} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \quad \vec{u} = \frac{1}{5}\begin{pmatrix} 4 \\ -3 \end{pmatrix}.$$

Hinweis: Der Vektor $\vec{u}$ hat Länge 1.



Lösung

Der Lotfusspunkt L bezeichnet den Punkt auf der Geraden, welcher A am nächsten liegt. Der Abstand von L zum Ursprung ist $\vec{a}\cdot\vec{u} = 2$. Dabei haben wir verwendet, dass $\vec{u}$ Länge 1 hat. Die gesuchte Distanz (blaue gestrichelte Linie im Bild) bezeichnen

wir mit d. Nach dem Satz von Pythagoras gilt dann

$$d^2 + (\vec{a} \cdot \vec{u})^2 = \|\vec{a}\|^2.$$

Durch umformen erhalten wir

$$d^2 = \|\vec{a}\|^2 - (\vec{a} \cdot \vec{u})^2 = 5 - 4 = 1.$$

Der Abstand von A zu Geraden ist also $d = 1$.

Was wir in Aufgabe 4.3 berechnet haben, funktioniert auch in höheren Dimensionen. Seien $\vec{v}$ und $\vec{w}$ zwei Vektoren mit $M \cdot N$ Komponenten, also

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{M \cdot N} \end{pmatrix}, \qquad \vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_{M \cdot N} \end{pmatrix}.$$

Deren Skalarprodukt ist dann definiert als

$$\vec{v} \cdot \vec{w} = v_1 w_1 + \ldots + v_n w_n.$$

Ganz analog ist auch das Quadrat Länge eines Vektors gegeben durch $\|\vec{v}\|^2 = \vec{v} \cdot \vec{v}$. Der Abstand eines Punktes zu einer Geraden berechnet sich dann nach der selben Formel wie in der Lösung von Aufgabe 4.3. Das probieren wir zuerst für $M \cdot N = 4$.

Aufgabe 4.4

Berechnen Sie den Abstand des Punktes A von der Geraden durch Null in Richtung $\vec{u}$, wobei $\vec{a} = \overrightarrow{OA}$ und

$$\vec{a} = \begin{pmatrix} 3 \\ -3 \\ 0 \\ 2 \end{pmatrix}, \quad \vec{u} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix}.$$

Hinweis: Der Vektor $\vec{u}$ hat Länge 1.

Lösung

Obwohl wir es mit vierdimensionalen Vektoren zu tun haben, spielt sich das Problem noch immer in einer Ebene ab, nämlich der Ebene, die von $\vec{a}$ und $\vec{u}$ aufgespannt wird. Wir verfahren daher gleich wie in Aufgabe 4.3. Der Abstand zum Lotfusspunkt L ist gegeben durch

$$\vec{a} \cdot \vec{u} = \tfrac{1}{2} \cdot 3 + \left(-\tfrac{1}{2}\right) \cdot (-3) + \tfrac{1}{2} \cdot 0 + \tfrac{1}{2} \cdot 2 = 4.$$

Dabei haben wir verwendet, dass $\vec{u}$ Länge 1 hat. Das Quadrat der Länge des Vektors $\vec{a}$ ist gegeben durch

$$\|\vec{a}\|^2 = \vec{a} \cdot \vec{a} = 3^2 + (-3)^2 + 0^2 + 2^2 = 22.$$

Die gesuchte Distanz bezeichnen wir mit d. Nach dem Satz von Pythagoras gilt dann

$$d^2 + (\vec{a} \cdot \vec{u})^2 = \|\vec{a}\|^2.$$

Durch umformen erhalten wir

$$d^2 = \|\vec{a}\|^2 - (\vec{a} \cdot \vec{u})^2 = 22 - 16 = 6.$$

Der Abstand von A zu Geraden ist also $d = \sqrt{6}$.

Aufgabe 4.5

Sei $\vec{u}$ ein Vektor mit $M \cdot N$ Komponenten und Länge 1. Dieser definiert die Gerade durch Null in Richtung $\vec{u}$. Finden Sie eine Formel für die Summe der Abstandsquadrate aller Differenzgesichter $A_1, \ldots, A_K$ zu dieser Geraden. Dies ist rechts in Abbildung 6 dargestellt.

Lösung

Wir konzentrieren uns zuerst auf ein beliebiges Differenzgesicht $\vec{a}_k = \overrightarrow{OA_k}$, wobei $1 \leq k \leq K$. Dessen Abstand zur Geraden bezeichnen wir mit d_k. Analog zu Aufgabe 4.3 gilt

$$d_k^2 = \|\vec{a}_k\|^2 - (\vec{a}_k \cdot \vec{u})^2 .$$

Die Summe dieser Abstandsquadrate $d_1^2 + \ldots + d_K^2$ ist demnach gegeben durch

$$\|\vec{a}_1\|^2 - (\vec{a}_1 \cdot \vec{u})^2 + \|\vec{a}_2\|^2 - (\vec{a}_2 \cdot \vec{u})^2 + \ldots + \|\vec{a}_{M \cdot N}\|^2 - (\vec{a}_{M \cdot N} \cdot \vec{u})^2 .$$

Die Formel aus Aufgabe 4.5 liefert für jedes $\vec{u}$ einen positiven Wert. Dasjenige $\vec{u}$, welches diesen Ausdruck minimiert, ist das erste Eigengesicht $\vec{u}_1$. Die Eigengesichter zu berechnen, heisst also ein Minimierungsproblem zu lösen. Dies ist nicht so einfach und ist daher bereits implementiert.

Die Eigengesichter wollen wir nun visualisieren, indem wir sie wieder als Bilder darstellen. Doch da gibt es noch ein Problem. Die Komponenten der Eigengesichter liegen nicht notwendigerweise zwischen 0 und 1 und können daher nicht als Graustufen interpretiert werden. Damit man sie als Bilder darstellen kann, müssen wir deren Komponenten zuerst in diese Teilmenge der reellen Zahl abbilden. Dazu multipliziert man jedes Eigengesicht mit einem gewissen Faktor, welcher vom Mass der Streuung entlang dieses Eigengesichtes abhängt. Dann entspricht das Eigengesicht ungefähr einem Differenzgesicht, welches man wieder durch Addition von $\vec{m}$ zu einem Vektor mit Einträgen zwischen 0 und 1 verschieben kann. Wir wollen hier nicht genauer darauf eingehen. Der Vollständigkeit halber geben wir aber in Abbildung 7 das Resultat an.

Im Grunde fangen die Eigengesichter charakteristische Gesichtszüge ein. Mit charakteristisch ist hier gemeint, dass genau diese Gesichtszüge für die grösste Streuung unter allen Bildern der Datenbank verantwortlich sind. Sie beschreiben die Merkmale, nach denen sich die Gesichter am meisten unterscheiden. Besser gesagt: Das erste Eigengesicht fängt den Gesichtszug mit der grössten Streuung ein. Die weiteren Eigengesichter fangen Gesichtszüge mit immer weniger Streuung ein. Dies spiegelt sich auch in deren Konstruktion wieder, welche die Eigengesichter ja gerade als Vektoren entlang der grössten Streuung definiert.

Abbildung 7: Die ersten 16 Eigengesichter wurden wieder als Bild dargestellt.

5 Projektion auf die Eigengesichter

Lernziele Kapitel 5

5.1 Die Lernenden können die Projektion eines Vektors auf einen Unterraum berechnen, wenn eine orthonormale Basis dieses Unterraumes gegeben ist. (Aufgaben 5.1 und 5.2)

5.2 Die Lernenden können das Skalarprodukt von Vektoren in Python berechnen. (Aufgabe 5.3)

In diesem Unterkapitel wollen wir allgemeine Bilder als Linearkombination der Eigengesichter darstellen. Man kann zeigen, dass alle Bilder der Datenbank exakt als Linearkombination der Eigengesichter geschrieben werden können. Neue Bilder können im Allgemeinen mit so einer Linearkombination nur angenähert werden. Die beste Näherung ist gerade die Projektion des neuen Bildes auf den Raum, der durch die Eigengesichter aufgespannt wird. Was das genau bedeutet, wollen wir zuerst im drei Dimensionen veranschaulichen.

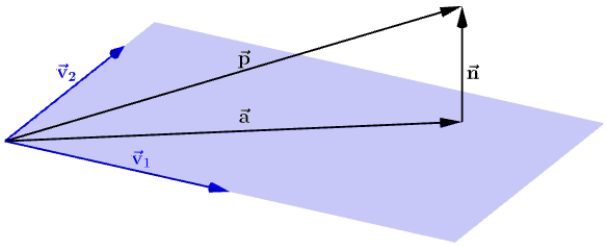


Abbildung 8: Projektion $\vec{a}$ von $\vec{p}$ auf die Ebene durch Null, die von $\vec{v}_1$ und $\vec{v}_2$ aufgespannt wird. Der Vektor $\vec{n}$ steht normal auf die Ebene und es gilt $\vec{p} = \vec{a} + \vec{n}$.

Aufgabe 5.1

Betrachten Sie die Vektoren

$$\vec{v}_1 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad \vec{v}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}.$$

(a) Sind die Vektoren $\vec{v}_1$ und $\vec{v}_2$ orthonormal? Begründen Sie.

(b) Die Vektoren $\vec{v}_1$ und $\vec{v}_2$ spannen eine Ebene durch den Nullpunkt auf. Berechnen Sie den Vektor $\vec{a}$, welcher der orthogonalen Projektion von $\vec{p}$ auf diese Ebene entspricht. Dazu können Sie zum Beispiel eine Zerlegung $\vec{p} = \vec{a} + \vec{n}$ wie in Abbildung 8 machen.

(c) Dann kann man $\vec{a}$ als Linearkombination vom $\vec{v}_1$ und $\vec{v}_2$ darstellen, also

$$\vec{a} = c_1\vec{v}_1 + c_2\vec{v}_2.$$

Berechnen Sie die Koeffizienten c_1 und c_2 in Termen von $\vec{v}_1, \vec{v}_2$ und $\vec{p}$.

Lösung

(a) Die Vektoren sind orthogonal, denn ihr Skalarprodukt ist Null

$$\vec{v}_1 \cdot \vec{v}_2 = \frac{1}{\sqrt{3}} \cdot \frac{1}{\sqrt{2}} \cdot (1 \cdot (-1) + 1 \cdot 1 + 1 \cdot 0) = 0.$$

Sie sind sogar orthonormal, denn sie haben auch Länge 1, weil

$$\vec{v}_1 \cdot \vec{v}_1 = \frac{1}{3}\left(1^2 + 1^2 + 1^2\right) = 1$$

und

$$\vec{v}_2 \cdot \vec{v}_2 = \frac{1}{2}\left((-1)^2 + 1^2 + 0^2\right) = 1.$$

(b) Wir zerlegen $\vec{p} = \vec{a} + \vec{n}$ wie in Abbildung 8 in einen Vektor $\vec{a}$ der in der Ebene liegt und einen Vektor $\vec{n}$ der orthogonal zur Ebene steht. Gesucht ist der Vektor $\vec{a}$. Da er in der Ebene liegt, gibt es eindeutige Koeffizienten c_1 und c_2, so dass

$$\vec{a} = c_1\vec{v}_1 + c_2\vec{v}_2.$$

Wir addieren $\vec{n}$ auf beiden Seiten und erhalten

$$\vec{p} = c_1\vec{v}_1 + c_2\vec{v}_2 + \vec{n}.$$

Nun bilden wir auf beiden Seiten das Skalarprodukt mit $\vec{v}_1$, also

$$\vec{v}_1 \cdot \vec{p} = c_1 \underbrace{\vec{v}_1 \cdot \vec{v}_1}_{=1} + c_2 \underbrace{\vec{v}_1 \cdot \vec{v}_2}_{=0} + \underbrace{\vec{v}_1 \cdot \vec{n}}_{=0}.$$

Hier haben wir verwendet, dass $\vec{v}_1$ orthogonal ist zu $\vec{v}_2$ und $\vec{n}$. Wir erhalten

$$c_1 = \vec{v}_1 \cdot \vec{p} = \frac{6}{\sqrt{3}}.$$

In dem man stattdessen auf beiden Seiten das Skalarprodukt mit $\vec{v}_2$ bildet, erhält man analog

$$c_2 = \vec{v}_2 \cdot \vec{p} = -\frac{2}{\sqrt{2}}.$$

Damit berechnen wir die Projektion

$$\vec{a} = c_1\vec{v}_1 + c_2\vec{v}_2 = 2\begin{pmatrix}1\\1\\1\end{pmatrix} - 2\begin{pmatrix}-1\\1\\0\end{pmatrix} = \begin{pmatrix}4\\0\\2\end{pmatrix}.$$

(c) In der Lösung der vorherigen Teilaufgabe haben wir die Koeffizienten schon berechnet zu

$$c_1 = \frac{6}{\sqrt{3}} \qquad \text{und} \qquad c_2 = -\frac{2}{\sqrt{2}}.$$

Was wir in der vorherigen Teilaufgabe berechnet haben, funktioniert auch in höheren Dimensionen. Die Vektoren $\vec{v}, \vec{w}$ und $\vec{p}$ sollen nun jeweils $M \cdot N$ Komponenten haben. Wir sagen, die zwei Vektoren sind orthogonal, wenn $\vec{v}\cdot\vec{w} = 0$. Falls sie zusätzlich Länge 1 haben, also $\vec{v}\cdot\vec{v} = 1$ und $\vec{w}\cdot\vec{w} = 1$, dann heissen sie orthonormal. Eine Familie $\vec{v}_1, \ldots, \vec{v}_K$ heisst orthonormal, wenn die Vektoren alle paarweise orthonormal sind. Sie spannen dann einen K-dimensionalen Raum auf. Wir können wie in Aufgabe 5.1 die Projektion des Vektors $\vec{p}$ auf diesen Raum berechnen, nämlich

$$c_1\vec{v}_1 + \ldots + c_K\vec{v}_K,$$

wobei die Koeffizienten gegeben sind durch

$$c_k = \vec{v}_k \cdot \vec{p}, \qquad k = 1, \ldots, K.$$

Nun betrachten wir wieder die Eigengesichter $\vec{u}_1, \ldots, \vec{u}_K$. Gemäss der Konstruktion aus dem letzten Kapitel sind diese orthonormal. Wir betrachten zudem die Mona Lisa von Leonardo da Vinci, ein Bild das nicht in unserer Datenbank ist. Den entsprechenden Vektor nennen wir $\vec{p}$. Um dieses Bild in den Raum zu verschieben, der von den Differenzgesichtern (oder von den Eigengesichtern) aufgespannt wird, müssen wir noch das Durchschnittsgesicht $\vec{m}$ subtrahieren. Das entstehende Differenzgesicht kann dann näherungsweise als Linearkombination der Differenzgesichter $\vec{a}_1, \ldots, \vec{a}_K$ oder auch der Eigengesichter $\vec{u}_1, \ldots, \vec{u}_K$ dargestellt werden (diese spannen den selben Raum auf). Nur näherungsweise deswegen, weil wir lediglich eine Projektion auf den Raum der Differenzgesichter (oder Eigengesichter) berechnen. Das Differenzgesicht des neuen Gesichtes liegt aber nicht notwendigerweise in diesem Raum. Diese Projektion berechnet sich wie oben beschrieben.

Aufgabe 5.2

Sei $\vec{p}$ das Bild der Mona Lisa und $\vec{u}_1, \ldots, \vec{u}_K$ die Eigengesichter. Berechnen Sie die Koeffizienten $c_1, \ldots, c_K$ der Projektion von $\vec{p} - \vec{m}$ auf die Eigengesichter. Geben Sie die Koeffizienten in Termen von $\vec{p}, \vec{m}$ und den Eigengesichtern an.

Lösung

Wir verfahren analog zu Aufgabe 5.1, nur das wir jetzt nicht $\vec{p}$, sondern $\vec{p} - \vec{m}$ projizieren müssen. Das heisst, wir teilen diesen Vektor auf in eine noch unbekannte Linearkombination der Eigengesichter und einen dazu orthogonalen Teil $\vec{n}$, also

$$\vec{p} - \vec{m} = c_1\vec{u}_1 + c_2\vec{u}_2 + \ldots + c_K\vec{u}_K + \vec{n}.$$

Wir bilden das Skalarprodukt beider Seiten der obigen Gleichung mit u_1, also

$$\vec{u}_1 \cdot (\vec{p} - \vec{m}) = c_1 \underbrace{\vec{u}_1 \cdot \vec{u}_1}_{=1} + c_2 \underbrace{\vec{u}_1 \cdot \vec{u}_2}_{=0} + \ldots + c_K \underbrace{\vec{u}_1 \cdot \vec{u}_K}_{=0}.$$

Weil die Eigengesichter orthonormal sind, vereinfacht sich das zu

$$\vec{u}_1 \cdot (\vec{p} - \vec{m}) = c_1.$$

Wenn wir das auch noch mit $\vec{u}_2, \ldots, \vec{u}_K$ machen erhalten wir

$$\vec{u}_k \cdot (\vec{p} - \vec{m}) = c_k,$$

für alle $k \in \{1, \ldots, K\}$.

Hat man die Koeffizienten der Linearkombination berechnet, so kann die Projektion der Mona Lisa wieder als Linearkombination der Eigengesichter und des Durchschnittsgesichtes darstellen

$$\vec{m} + c_1\vec{u}_1 + c_2\vec{u}_2 + \ldots + c_K u_K.$$

Aufgabe 5.3

Seien `p` und `m` Vektoren mit $M \cdot N$ Komponenten, wobei `p` ein Gesicht zeigt (z.B. das der Mona Lisa) und `m` das Durchschnittsgesicht ist. Sei zudem `u_list` die Liste der Eigengesichter. Ergänzen Sie im Skript `chapter5.py` die Funktion `compute_coefficients(p, m, u_list)`, welche die Liste der Koeffizienten der Linearkombination aus Abbildung 9 zurück gibt. Um Ihre Lösung zu testen können Sie das Python Skript `chapter5.py` laufen lassen, welches die Projektion der Mona Lisa mit den von Ihnen berechneten Koeffizienten darstellt. Dazu müssen Sie auf der Kommandozeile den Pfad zum File `eigenfaces.dat` übergeben. *Hinweis:* Mit `np.dot(v,w)` lässt sich das Skalarprodukt zweier Vektoren `v` und `w` berechnen.

Lösung

Die Lösung könnte zum Beispiel so aussehen.

```
1 def compute_coefficients(p, m, u_list):
2     K = len(u_list)
3     c_list = np.zeros((K,))
4     for k in range(K):
5         c_list[k] = np.dot(u_list[k], p - m)
6     return c_list
```

./codes/solution/chapter5.py

Rekonstruktion:

Anscheinend sieht die Projektion fast gleich aus wie das Bild selber. Etwas informell ist das in in Abbildung 9 dargestellt. Dieses Phänomen bedeutet, dass das Differenzgesicht der Mona Lisa ganz oder in sehr guter Näherung im Raum liegt, er durch die Eigengesichter aufgespannt wird. Mit dieser Beobachtung wollen wir uns im nächsten Kapitel genauer auseinandersetzen.

Abbildung 9: Die Projektion der Mona Lisa auf die Eigengesichter entspricht einer Überlagerung (Linearkombination) der Eigengesichter und des Durchschnittsgesichtes. Die Vorfaktoren sind nur symbolisch und entsprechen nicht den echten Werten.

6 Bildkompression

Lernziele Kapitel 6

6.1 Die Lernenden können erklären, was die Eigengesichter im Vergleich zu anderen Basen auszeichnet und warum sie sich zur Bildkompression eignen.
(Aufgaben 6.1 und 6.3)

6.2 Die Lernenden können mit gegebenen Eigengesichtern eine Bildkompression in Python durchführen, falls alle Bilder als Vektoren gegeben sind.
(Aufgabe 6.2)

Wenn wir ein schwarz-weiss Bild mit der Auflösung $M = 144$ mal $N = 180$ Pixel für Pixel abspeichern, so müssen wir $M \cdot N = 25920$ Zahlen abspeichern. Unter Bildkompression versteht man Verfahren, die Bilder mit weniger Zahlen darstellen können. Das ist zum Beispiel nützlich wenn man Bilder über das Internet verschicken möchte (oder gleich ganze Filme). Die Datenmenge kann durch Kompression stark verringert werden, was zu viel kürzeren Download-Zeiten führt. Mit den Eigengesichtern lassen sich Bilder von Gesichtern komprimieren. Grund dafür ist eine spezielle Eigenschaft, welche die Eigengesichter auszeichnet, und dir wir nun untersuchen werden.

Im letzten Kapitel haben wir das Gemälde der Mona Lisa durch eine Projektion als Linearkombination der Eigengesichter und es Durchschnittsgesichtes angenähert. Dazu mussten wir mit geeigneten Skalarprodukten die Koeffizienten $c_1, \ldots, c_K$ dieser Linearkombination berechnen. Allerdings hätte man anstatt der Eigengesichter auch direkt die Differenzgesichter $\vec{a}_1, \ldots, \vec{a}_K$ der Bilder aus der Datenbank nehmen können. In diesem Fall würde man einfach andere Koeffizienten erhalten. Allerdings lassen sich diese nicht so einfach berechnen, da diese Differenzgesichter nicht orthonormal sind. Man betrachte Abbildung 10 für einen Vergleich der beiden Varianten. Ein weiterer Unterschied zwischen

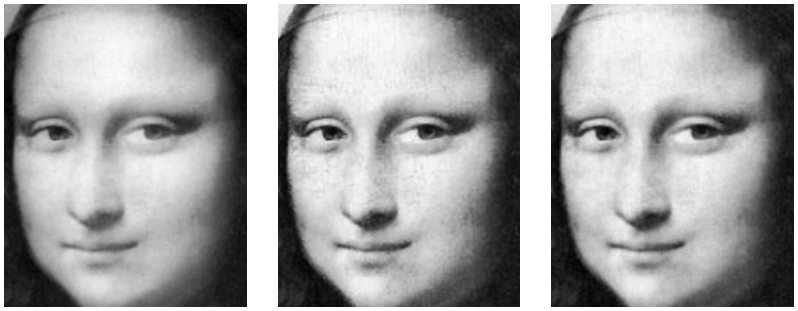


Abbildung 10: Rekonstruktion der Mona Lisa. Links wurden die Differenzbilder der Testgesichter $\vec{a}_1, \ldots, \vec{a}_K$ verwendet. Rechts wurden die Eigengesichter $\vec{u}_1, \ldots, \vec{u}_K$ verwendet. In der Mitte ist das Original.

diesen beiden Ansätzen ist die Verteilung der Beträge der Koeffizienten, also $|c_1|, \ldots, |c_K|$. Diese sind in Abbildung 11 gezeigt.

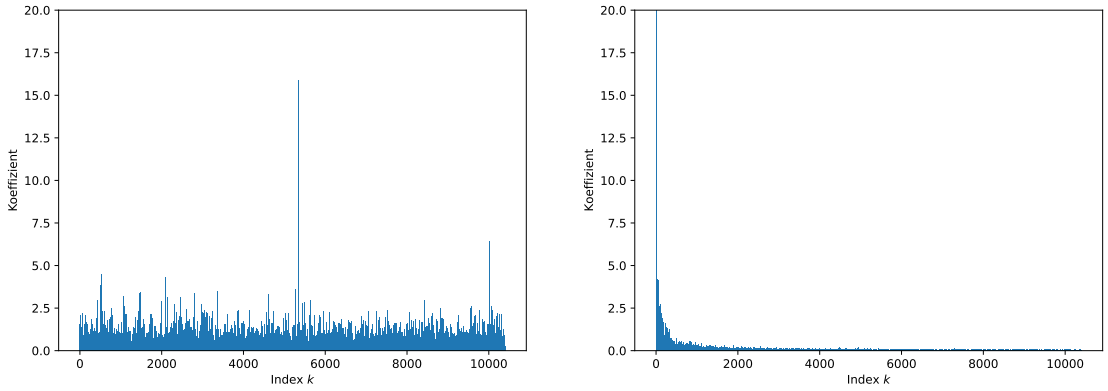


Abbildung 11: Der Absolutbetrag der Koeffizienten der Linearkombination des Differenzgesichtes der Mona Lisa, die in Abbildung 10 verwendet wurden. Links die Differenzgesichter der Datenbank und rechts die Eigengesichter verwendet.

Aufgabe 6.1

Betrachten Sie die Abbildung 11, welche die Koeffizienten der Linearkombinationen zeigt, die für die Bilder links bzw. rechts in Abbildung 10 verwendet wurden. Beschreiben Sie die Unterschiede in der Verteilung der Koeffizienten. Welche Rückschlüsse kann man dadurch ziehen?

Lösung

Die Koeffizienten der Eigengesichter fallen schnell ab. Dem rechten Bild kann man entnehmen, dass ungefähr die ersten 2000 Eigengesichter fast den ganzen Beitrag zur Linearkombination leisten. Die Koeffizienten der restlichen Eigengesichter ist fast Null. Bei der Basis der Differenzgesichter ist keine solche Struktur zu erkennen. Es ist nicht auszumachen, ob und welche Differenzgesichter besonders stark vertreten sind.

Was wir hier in Abbildung 11 beobachtet haben ist kein Einzelfall. Obwohl wir nur das Beispiel der Mona Lisa betrachtet haben, würden andere Gesichter ähnlich verteilte Koeffizienten liefern. Die Lösung der vorherigen Aufgabe ist zugleich die Grundidee der Bildkompression mittels Eigengesichter. Anstatt alle der K Eigengesichter, verwendet man nur die ersten $\tilde{K}$ Eigengesichter um ein Bild darzustellen. Die Abbildung 11 lässt vermuten, dass wenn $\tilde{K}$ gross genug gewählt ist, aber immer noch viel kleiner als K, sich die Bildqualität nicht wesentlich verschlechtert. Genauer gesagt, können wir ablesen, dass wohl die ersten $\tilde{K} = 2000$ Eigengesichter ausreichend sein sollten. Nun wollen ein Bild $\vec{p}$ rekonstruieren, indem wir nur die ersten $\tilde{K}$ Eigengesichter verwenden. Das rekonstruierte Gesicht bezeichnen wir mit $\vec{a}$, also

$$\vec{a} = \vec{m} + c_1\vec{u}_1 + c_2\vec{u}_2 + \ldots + c_{\tilde{K}}\vec{u}_{\tilde{K}},$$

wobei $\tilde{K} \leq K$ und $\vec{u}_1, \ldots, \vec{u}_{\tilde{K}}$ sind die ersten $\tilde{K}$ Eigengesichter. Wie man die Koeffizienten $c_1, \ldots, c_{\tilde{K}}$ berechnet, haben wir im letzten Kapitel gesehen.

Aufgabe 6.2

Ergänzen Sie im Skript `chapter6.py` die Funktion `compress(p, m, u_list, K_tilde)`, welche einen Vektor der Länge $M \cdot N$ zurück gibt, der dem Bild $\vec{a}$ entspricht. Hier bezeichnen `p` das ursprüngliche Bild $\vec{p}$ und `m` das Durchschnittsgesicht $\vec{m}$. Die Liste `u_list` enthält die Eigengesichter und `K_tilde` entspricht der Anzahl $\tilde{K}$ der Eigengesichter, die verwendet werden sollen. Testen Sie ihre Lösung indem Sie das Skript `chapter6.py` laufen lassen, welches die rekonstruierten Bilder ausgibt. Dazu müssen Sie in der Kommandozeile den Pfad zum File `eigenfaces.dat` übergeben. *Hinweis:* Verwenden Sie in ihrem Code die Funktion `compute_coefficients(p, m, u_list)` aus Aufgabe 5.3.

Lösung

Die Lösung könnte zum Beispiel so aussehen.

```
1 def compress(p, m, u_list, K_tilde):
2     # Reduzierte Liste der ersten K_tilde Eigengesichter
3     u_list_reduced = u_list[:K_tilde]
4     c_list_reduced = compute_coefficients(p, m, u_list_reduced)
5     a = np.zeros_like(p)
6     for k in range(K_tilde)[::-1]:
7         a += c_list_reduced[k] * u_list_reduced[k]
8     return a + m
```

./codes/solution/chapter6.py

Die durch `chapter6.py` generierten Bilder sind in Abbildung 12 gezeigt.

Was wir in Abbildung 12 beobachten ist eine Bildkompression. Sind die Eigengesichter bekannt, so lässt sich ein Bild rekonstruieren aus den Koeffizienten $c_1, \ldots, c_{\tilde{K}}$ der ersten $\tilde{K}$ Eigengesichter. Um ein Bild eines Gesichtes in leicht verminderter Qualität über das Internet zu versenden, wäre es ausreichend, nur $\tilde{K} = 2000$ Zahlen zu versenden, sofern der Empfänger über die Eigengesichter verfügt. Wir erinnern uns, dass ein Bild welches Pixelweise versendet wird, $M \cdot N = 25920$ Zahlen benötigt. Das ist etwa ein Faktor 13 mehr als die komprimierte Variante. Trotzdem sieht man für $\tilde{K} = 2000$ kaum einen Unterschied zum Original in Abbildung 12. Gleichzeitig sehen wir, dass die aggressivere Komprimierung mit $\tilde{K} = 200$ die Bildqualität doch sehr vermindert. Aber das konnte man aufgrund des rechten Verteilung in Abbildung 11 schon vermuten. Weiter beobachten wir, dass die Komprimierung für Bilder, welche kein Gesicht zeigen, weniger gut funktioniert. Der Grund dafür ist, das diese Differenzbilder nicht notwendigerweise nahe am Raum liegen, der durch die Differenz**gesichter** der Datenbank aufgespannt wird. Doch die Eigengesichter sind eine Basis eben dieses Unterraumes.

Aufgabe 6.3

Könnte man für die Bildkompression anstelle der Eigengesichter auch die Differenzgesichter $\vec{a}_1, \ldots, \vec{a}_K$ nehmen? *Hinweis*: Betrachten Sie Abbildung 11.

Abbildung 12: Rekonstruktion mit verschiedenen $\tilde{K}$. Gezeigt sind die Mona Lisa, Queen Elizabeth, ein Stuhl und das erste Space Shuttle „Columbia“. Gesichter werden generell besser rekonstruiert als andere Bilder.

Lösung

Nein, bei den Differenzgesichter ist nicht klar, welche Vektoren weggelassen werden können. Selbst wenn für ein konkretes Bild einige wenige ausreichen würden, so gibt es keine Möglichkeit zu bestimmen, welche das sind. Zudem sind diese Vektoren nicht orthonormal. Daher ist die Berechnung der Koeffizienten nur mittels Skalarprodukt nicht möglich.

7 Gesichtserkennung

Lernziele Kapitel 7

7.1 Die Lernenden können erklären, wie die Eigengesichter das „ähnlich aussehen“ von Gesichtern quantifizieren können.
(Aufgaben 7.1 und 7.2)

Nun wollen wir die Eigengesichter nutzen für eine Gesichtserkennung nutzen. Wie bereits in der Einleitung erklärt, ist Gesichtserkennung im Grunde eine Klassifizierung. Hier als Beispiel betrachten wir 8 verschiedene Klassen. Wir verwenden in diesem Kapitel eine Datenbank die nur aus Bilder dieser 8 Personen besteht. Diese kann hier heruntergeladen werden

https://people.math.ethz.ch/~rioliver/eigenfaces/datenbank_small.zip

Sie enthält von jeder Person genau 10 Bilder. Das heisst, sie besteht aus $K = 8 \cdot 10 = 80$ Bildern. Pro Klasse haben wir zusätzlich 3 Testbilder. Das sind Bilder die zwar jeweils

Tabelle 1: Die acht verschiedenen Klassen.

eine dieser 8 Personen zeigen, aber nicht in der Datenbank enthalten sind. Das Ziel ist nun die Testbilder möglichst der richtigen Klasse zuzuordnen. So können wir unsere Gesichtserkennung testen, daher der Name „Testbilder“. Zwar ist das für einen Menschen ganz

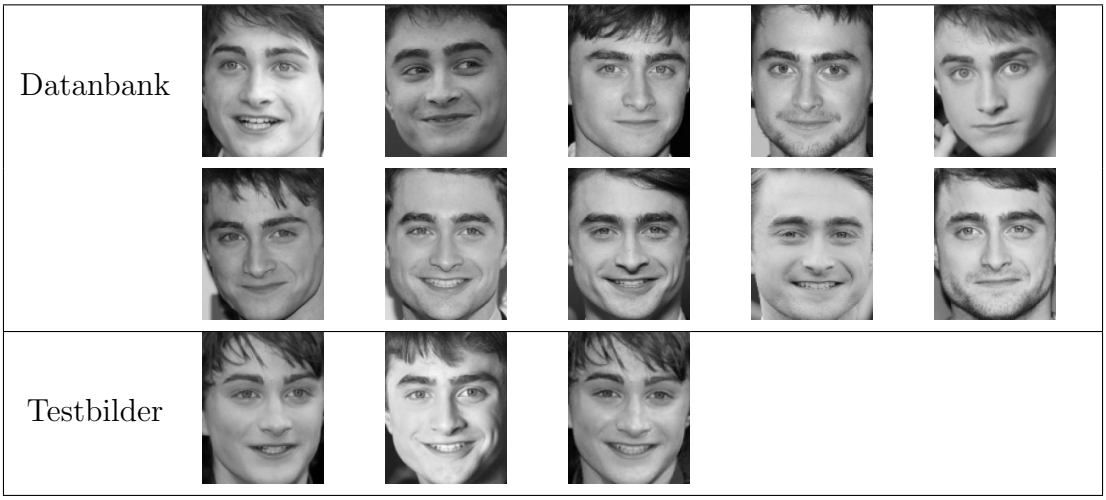


Abbildung 13: Datenbank- und Testbilder am Beispiel der Klasse „Daniel Radcliffe“

leicht, aber für einen Computer ist das überhaupt nicht einfach. Im Grunde müssen wir den Computer lehren, wann zwei Bilder ähnlich aussehen. Dazu brauchen wir ein Mass für die Distanz zweier Bilder. Es stellt sich heraus, dass die Eigengesichter ein Solches liefern. Seien nun $\vec{p}$ und $\vec{q}$ zwei Bilder von Gesichtern. Wir schreiben die entsprechenden Differenzgesichter als Linearkombination der Eigengesichter

$$\vec{p} - \vec{m} = c_1\vec{u}_1 + c_2\vec{u}_2 + \ldots + c_K\vec{u}_K$$

und

$$\vec{q} - \vec{m} = \tilde{c}_1\vec{u}_1 + \tilde{c}_2\vec{u}_2 + \ldots + \tilde{c}_K\vec{u}_K,$$

wobei nun $K = 80$ und $\vec{m}$ das Durchschnittsgesicht der Bilder der neuen Datenbank ist. Entsprechend sehen die Eigengesichter auch etwas anders aus, aber deren Eigenschaften sind die selben. Das besagte Mass der Distanz zwischen den Gesichtern $\vec{p}$ und $\vec{q}$ definieren wir als

$$D(\vec{p}, \vec{q}) = (c_1 - \tilde{c}_1)^2 + (c_2 - \tilde{c}_2)^2 + \ldots + (c_K - \tilde{c}_K)^2 .$$

Aufgabe 7.1

Warum entspricht dieser Begriff von Distanz dem „Ähnlich aussehen“ von Gesichtern? Erklären Sie in Worten. *Hinweis:* Rufen Sie sich die Konstruktion der Eigengesichter und die Diskussion am Ende von Kapitel 3 in Erinnerung.

Lösung

Die Eigengesichter fangen gewisse Gesichtszüge ein. Die Funktion $D(\vec{p}, \vec{q})$ vergleicht, wie fest sich $\vec{p}$ und $\vec{q}$ in eben diesen Gesichtszügen unterscheiden. Dabei werden die ersten Eigengesichter am stärksten gewichtet, da ihre Koeffizienten typischerweise am grössten sind (Abbildung 11). Das macht auch Sinn, denn es sind genau die ersten Eigengesichter (Gesichtszüge), in denen sich die Gesichter der Datenbank am meisten unterscheiden. Die Bilder nur Pixel für Pixel zu vergleichen macht keinen Sinn, denn kein Pixel kann alleine eine Information über ein Gesicht tragen. Anders gesagt können sich zwei Bilder in jedem Pixel stark unterscheiden und dennoch die selbe Person zeigen. Mit Gesichtszügen ist dies kaum möglich.

Sei nun ein Bild $\vec{p}$ gegeben, das nicht notwendigerweise in der Datenbank enthalten ist. Das Bild soll zudem eine der 8 Personen aus Tabelle 1 zeigen. Jedoch wissen wir nicht welche. Wir werden versuchen das mit folgendem Verfahren zu erraten.

Schritt 1: Seien $\vec{q}_1, \ldots, \vec{q}_{80}$ die Bilder der Datenbank. Wir berechnen die Distanz $D(\vec{p}, \vec{q}_i)$ für $i = 1, \ldots, 80$.

Schritt 2: Eines dieser Bilder wird die kleinste Distanz zu $\vec{p}$ haben, sagen wir das ist $\vec{q}_j$ für ein bestimmtes $1 \leq j \leq 80$.

Schritt 3: Wir schauen in der Datenbank nach, welche Person auf dem Bild $\vec{q}_j$ gezeigt ist. Da $\vec{p}$ ähnlich aussieht, zeigt es wohl die selbe Person und wir klassifizieren $\vec{p}$ entsprechend.

Aufgabe 7.2

Ergänzen Sie im Skript `chapter7.py` die Funktion `distance(p, q, m, u_list)` welche die Distanz $D(\vec{p}, \vec{q})$ zweier Bilder zurück gibt. Dabei ist `m` das Durchschnittsgesicht und `u_list` enthält die Liste der Eigengesichter. Die Variablen `p` und `q` enthalten die Vektoren $\vec{p}$ und $\vec{q}$. Testen Sie ihre Implementation indem Sie das Skript `chapter7.py` ausführen. Dazu müssen Sie in der Kommandozeile den Pfad zur reduzierten Datenbank angeben. *Hinweis:* Verwenden sie die Funktion `compute_coefficients(...)` aus Aufgabe 5.3.

Lösung

Die Lösung könnte zum Beispiel so aussehen.

```
def distance(p, q, m, u_list):
    cp = compute_coefficients(p, m, u_list)
    cq = compute_coefficients(q, m, u_list)
    d = (cp - cq)
    return np.dot(d, d)
```

./codes/solution/chapter7.py

Der Output von `recognition.py` ist in Tabelle 2 gelistet.

Klasse	**Anzahl Testbilder**	**davon richtig klassifiziert**
Daniel Radcliffe	3	2
Emma Stone	3	2
Emma Watson	3	2
Eva Green	3	2
Jennifer Lawrence	3	3
Orlando Bloom	3	2
Pierce Brosnan	3	1
Tom Cruise	3	2

Tabelle 2: Resultate der Gesichtserkennung.

Aus Tabelle 2 ist zu entnehmen, dass von den insgesamt $3 \cdot 8 = 24$ Testbildern 16 richtig klassifiziert wurden. Dies entspricht einer Erfolgsquote von $\frac{16}{24} = \frac{2}{3}$, also etwa 66%. Zum Vergleich: Würde man die Testbilder einfach zufällig den Klassen zuordnen, so wäre die erwartete Erfolgsquote $\frac{1}{8} = 0.125$, also 12.5%.

Teil II
Didaktische Methoden

Die Vektorgeometrie wir am Gymnasium im Anschauungsraum, also im $\mathbb{R}^3$, unterrichtet. Hier lassen sich die wichtigen Konzepte wie Vektoren, Linearkombination, Skalarprodukt, usw. in Bildern darstellen und dadurch leicht erklären. Während diese Konzepte sich auf natürliche Weise auf höhere Dimensionen verallgemeinern lassen, sind sie dort viel schwieriger darzustellen. Das Lernskript im zweiten Teil dieser Arbeit versucht genau das. Die Eigengesichter sind ein Weg, bestimmte Vektoren im $\mathbb{R}^n$ zu visualisieren. Vielleicht lässt sich die grundlegende lineare Algebra im $\mathbb{R}^n$ so auf ansprechende Weise präsentieren. Die nachfolgenden Kapitel behandeln jeweils eine didaktische Methode, die im Lernskript verwendet wird. Dabei wird die Methode erklärt, mit Referenz auf Ergebnisse der Lehr- und Lernforschung. Zudem wird erklärt, wie diese Methode im Lernskript verwendet wurde.

1 Interleaved Practice

Wir stellen uns vor, dass zwei Themen unterrichtet werden sollen, Thema A und Thema B. Das Naheliegendste wäre nun, zuerst mehrere Lektionen ausschliesslich zum Thema A zu unterrichten. Wenn dieses Thema abgeschlossen ist, werden mehrere Lektionen zum Thema B unterrichtet. Diesen Ansatz nennen wir *blocked study* [4]. Im Gegensatz dazu steht der Ausdruck *interleaved study* [4]. Damit ist gemeint, dass die Themen A und B jeweils abwechslungsweise unterrichtet werden. Symbolisch schreiben wir dafür

- blocked: AAAABBBB
- interleaved: ABABABAB

Diese Begriffe können sich auch auf die Reihenfolge von Übungsaufgaben beziehen anstatt auf die Reihenfolge von Unterrichtsblöcken. Wir nennen das entsprechend *blocked practice* und *interleaved practice*.

Generell scheint der Ansatz „interleaved“ langfristig besseren Lernerfolg zu versprechen als „blocked“. Für den Mathematikunterricht wurde dies zum Beispiel durch Rohrer und Taylor untersucht in [13]. Dabei wurden zwei Experimente durchgeführt, von denen wir hier das zweite kurz zusammenfassen. Die Lernenden in zwei Gruppen aufgeteilt. Eine lernte nach dem „interleaved“ Prinzip, die andere nach dem „blocked“. Dabei ging es um Volumenberechnungen dreidimensionaler Körper, genauer um Zylinder, Kugeln, Spheroid und Kegel. Die verschiedenen Themen entsprechen genau diesen vier Klassen von Körpern. Beide Gruppen erhielten den selben Unterricht und die selben Aufgaben. Die „blocked“-Gruppe löste allerdings alle Aufgaben zur selben Körperklasse hintereinander, währen die Aufgaben beider „interleaved“-Gruppe gemischt waren. Während der Übungsaufgaben schnitt die „blocked“-Gruppe besser ab. Doch beim anschliessenden Test, der gemischte Aufgaben enthielt, schnitt die „interleaved“-Gruppe deutlich besser ab. Dies ist in Abbildung 14 gezeigt.

Nun kommen wir zur Umsetzung im Lernskript. Hier zerfällt der Stoff grob in zwei Themen: Die lineare Algebra im $\mathbb{R}^n$ (Thema A) und die Implementierung in Python (Thema B). Der „blocked“-Ansatz würde zuerst die Theorie und Übungsaufgaben zur linearen Algebra der Eigengesichter unterrichten. Nachdem dieser rein theoretische Block abgehandelt ist, würde die Implementierung mit den entsprechenden Aufgaben folgen.

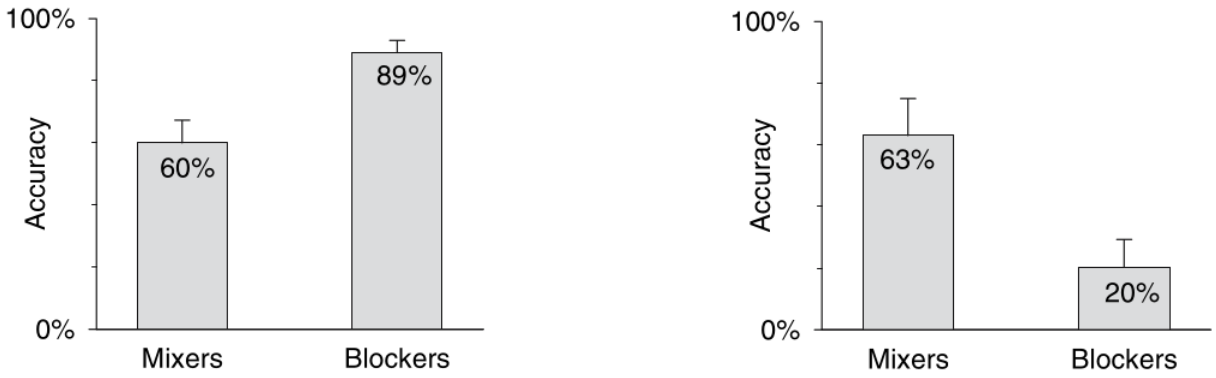


Abbildung 14: Resultate bei den Übungen (links) und den Test (rechts) [13]. „Mixed“ steht für „interleaved“.

Unter dem obigen Gesichtspunkt wurde aber der „interleaved“-Ansatz gewählt. Das heisst, es gibt jeweils einen kurzen Theorie-Block und/oder eine Aufgabe zur linearen Algebra. Danach folgt meist eine Implementierungsaufgabe.

Allerdings entspricht dies nicht exakt der „interleaved practice“ wie sie in [13] untersucht wurde. Nicht nur bei den Übungsaufgaben, sondern auch bei der Vermittlung des Stoffes wechseln die Themen A und B ab. Zudem bauen die Implementierungsaufgaben oft auf den Theorieaufgaben auf, sind also nicht unabhängig. Andererseits sind die Aufgaben beim Lernskript Teil des Theorie-Inputs und nicht etwa reine Übungsaufgaben wie in der Studie.

2 Scaffolding

Selbstständiges Entdecken ist gerade in der Mathematik wünschenswert und wichtig. Gerade bei komplexen Themen ist das aber schwierig umzusetzen. In diesem Fall können gewisse Hilfestellungen gegeben werden, die es den Lernenden erlauben, Hindernisse zu überwinden, an denen sie sonst gescheitert wären. Diese Hilfestellungen bilden sinnbildlich ein „Gerüst“. Der Einsatz so eines Gerüstes wird entsprechend als *Scaffolding* bezeichnet [14]. So eine Hilfestellung ist im Lernskript einerseits durch die Anleitung selbst gegeben, andererseits aber auch durch den zur Verfügung gestellten Python Code. Dieser nimmt den Lernenden insbesondere folgende Hindernisse aus dem Weg, da die entsprechenden Operationen schon implementiert sind.

- Das einlesen der Bilder aus der Datenbank, d.h. die Konvertierung einer Bilddatei in eine Matrix aus Zahlen, wie in Abbildung 3 dargestellt.
- Das erstellen von Bildern, d.h. die Konvertierung einer Matrix aus Zahlen in eine Bilddatei.
- Die numerisch stabile Berechnung der Eigengesichter mittel Singulärwertzerlegung.
- Das einlesen und abspeichern der Metadaten, das heisst der Information, welches Bild der Datenbank zu welcher Person gehört.

Ohne diese Hilfestellungen wäre das Thema der Eigengesichter nur schwer zugänglich, denn die Implementierung dieser Operationen kann sehr frustrierend sein. Andererseits sind sie notwendig um einen graphischen Output zu generieren. Zudem haben sie (bis

auf die Singulärwertzerlegung) nicht viel mit Mathematik zu tun. Indem sie als Blackbox zur Verfügung gestellt werden, könne die Lernenden sich auf die wesentliche Mathematik konzentrieren.

3 Lernziele

Zu Beginn jedes Kapitel (bis auf das erste) werden im Lernskript jeweils die Lernziele, sowie die Aufgaben welche diese abdecken, aufgelistet. Dies zeigt den Lernenden auf, was wichtig ist und was von ihnen erwartet wird. So können Sie ihre Lernzeit gezielter investieren. Damit die Lernenden ihren eigenen Fortschritt prüfen können, wird zu jedem Lernziel mindestens eine Aufgabe gestellt.

Lernziele können gemäss der revidierten Taxonomie von Bloom klassifizert werden [2], [1]. Die verschiedenen Klassen sind in Abbildung 15 aufgelistet. Alle Lernziele aus dem Lern-

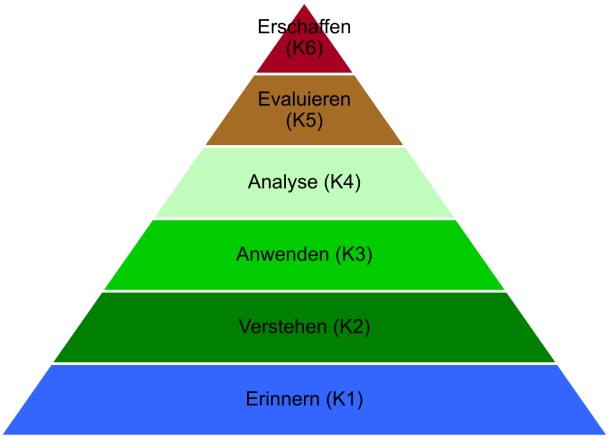


Abbildung 15: Blooms revidierte Taxonomie von Lernzielen [1].

skript sind in Tabelle 3 nochmals aufgelistet und gemäss dieser Taxonomie klassifiziert.

Klassifizierung	Lernziele
Wissen	
Verstehen	3.3, 4.1, 6.1, 7.1
Anwenden	2.1, 2.2, 3.1, 3.2, 4.2, 5.1, 5.2, 6.2
Analyse	
Evaluieren	
Erschaffen	

Tabelle 3: Klassifizierung der Lernziele aus dem Skript.

4 Gruppenarbeit: Erstellung der Datenbank

Jedes Programm zur Gesichtserkennung braucht Bilder von Gesichtern, aus denen es „lernen“ kann, wie neue Gesichter zu klassifizieren sind. Das Lernskript kann mit der bereitgestellten Datenbank bearbeitet werden. Alternativ kann eine eigene Datenbank erstellt und verwendet werden. Letzteres lässt sich in als Gruppenarbeit, zum Beispiel

mit einer Schulklasse, auslegen. Die Anleitung dazu befindet sich in Kapitel 1. In dieser Gruppenarbeit wird noch keine Mathematik vermittelt. Allerdings bietet sie einen Einstieg in das sonst sehr abstrakte Lernskript, welcher auch der sozialen Komponente des Lernens Rechnung trägt. Für eine gelungene Gruppenarbeit sind gemäss [3] folgende fünf Erfolgsfaktoren entscheidend:

1. *Positive Interdependenz*: Das Ziel kann nur gemeinsam erreicht werden.
2. *Individuelle Verantwortlichkeit*: Alle müssen ihre Aufgaben erledigen.
3. *Förderliche Interaktionen*: Aufgaben müssen soziale Interaktionen erfordern.
4. *Kooperative Arbeitstechniken*: Führen und sich führen lassen, Konflikte lösen, Kompromisse schliessen.
5. *Reflexive Prozesse*: Erwerb inhaltlicher und sozialer Kompetenzen.

Bei der gemeinsamen Erstellung einer Datenbank kommen einige dieser Faktoren zu tragen. Darauf und auch auf einige Probleme wird im Folgenden eingegangen.

1. *Positive Interdependenz*: Je mehr Bilder die Datenbank zu Verfügung hat, desto mächtiger werden die Anwendungen wie zum Beispiel die Bildkompression. Dementsprechend werden die Resultate schlechter, wenn nicht jeder seinen Teil zur Datenbank beiträgt. Das Problem ist, dass dies nicht sofort, sondern erst später in der Bearbeitung des Lernskriptes sichtbar wird.
2. *Individuelle Verantwortlichkeit*: Wie im vorherigen Punkt erklärt, wird das Resultat der Ganzen Klasse schlechter wenn jemand seinen/ihren Beitrag nicht leistet. Zudem müssen alle Anweisungen in Kapitel 1 ganz genau befolgt werden. Wenn nur ein einziges Bild nicht das richtige Format hat, wird das Programm nicht laufen und niemand kann das Lernskript weiter bearbeiten.
3. *Förderliche Interaktionen*: Eine soziale Interaktion ist nur zwingend erforderlich, wenn die Resultate zusammengetragen werden. Dabei müssen alle Files gemäss Abbildung 2 zusammengetragen werden und die richtigen `.csv` Dateien müssen aufgesetzt werden.
4. *Kooperative Arbeitstechniken*: Diese sind nur nötig beim Zusammentragen, wie im vorherigen Punkt beschrieben.
5. *Reflexive Prozesse*: In der Gruppenarbeit wird zwar keine Mathematik unterrichtet, aber das zuschneiden der Bilder kann helfen, ein Bild als eine $M \times N$ „Matrix“ von Pixeln zu verstehen.

In Kapitel 1 wird nicht darauf eingegangen wie man das Bildbearbeitungsprogramm oder das Tabellen-Kalkulationsprogramm genau verwenden muss. Der Grund dafür ist einerseits, dass jedes dieser Programme anders funktioniert. Andererseits sind die Lernenden so gezwungen einander zu helfen. Sobald jemand herausfindet, wie er z.B. ein Bild zu schwarz-weiss konvertieren kann, sollte er/sie es den anderen zeigen. Denn die Datenbank kann erst erstellt werden, wenn alle ihren Teil erledigt haben.

5 Problem basiertes Lernen

Hierbei handelt es sich um eine Methode zur kognitiven Aktivierung. Den Lernenden wird eine Aufgabe zu einem neuen Thema gestellt, ohne dass sie zuvor einen Theorie-Input zu diesem Thema erhalten haben [7]. Dieser folgt erst nach der Bearbeitung dieser Aufgabe. Dabei soll das Interesse der Lernenden geweckt und ihr Vorwissen aktiviert werden. Aufgrund des fehlenden Theorie-Inputs mag dieses Vorwissen nicht ausreichend sein um die Aufgabe zu lösen. In jedem Fall sollen die Lernenden sich aber ihrer Wissenslücken bewusst werden und damit die Notwendigkeit des nachfolgenden Unterrichts besser verstehen [7]. Der Effekt dieses *Problem basierten Lernens* (PBL) wurde in [7] untersucht. Dabei wurden die Newtonschen Bewegungsgesetze mit drei verschiedenen Methoden an drei verschiedene Gruppen unterrichtet. Diese waren PBL, Selbststudium und ein Lehrervortrag. Der Inhalt war bei allen Methoden der Selbe. Aus den nachfolgenden Test geht hervor, dass PBL viel eher zur Überwindung der Messkonzepte führte wie die anderen Methoden. Zudem war dieser Konzeptwandel insbesondere nach einer Woche beim PBL am effektivsten.

Im Lernskript wurde PBL in Kapitel 6 eingesetzt. Das neue Konzept, welches hier eingeführt wird, ist die Bildkompression mittels der Eigengesichter. Die kognitiv aktivierende Aufgabe des PBL Ansatzes ist Aufgabe 6.1. Genauer: Die Grundidee dieser Kompression ist in Abbildung 11 enthalten. Wenn ein Differenzgesicht als Linearkombination der Eigengesichter dargestellt wird, so fallen die Beträge dieser Koeffizienten rasch ab. Doch bevor erklärt wird, wie nun die Bildkompression mit Eigengesichtern genau funktioniert, sollten die Lernenden Aufgabe 6.1 bearbeiten. Dabei vergleichen sie die Linearkombination mittels Eigengesichter mit derjenigen einer anderen Basis des selben Unterraumes. Der Kontrast dieser beiden Beispiele erlaubt den Lernenden hoffentlich, die speziellen Eigenschaften der Eigengesichter herauszuarbeiten. Die Frage in Aufgabe 6.1 ist bewusst offen gestellt, so dass eine Diskussion möglich ist. Dabei können die Lernenden hoffentlich eine kognitive Aktivierung erfahren, die ihnen die Auseinandersetzung mit der nachfolgenden Theorie erleichtert.

6 Vom $\mathbb{R}^3$ in den $\mathbb{R}^n$

Die Vektorgeometrie wird am Gymnasium hauptsächlich im Anschauungsraum $\mathbb{R}^3$ unterrichtet. Um die Methode der Eigengesichter zu verstehen, müssen die Konzepte der Vektorgeometrie auf den $\mathbb{R}^n$ verallgemeinert werden. Dies birgt insbesondere die zwei folgenden Schwierigkeiten:

1. Es können kaum mehr Bilder gezeichnet werden, um die Konzepte zu veranschaulichen.

2. Die Komponenten der Vektoren können nicht mehr explizit aufgelistet werden.

Um diese Konzepte effizient zu lernen, muss irgendwie an das Vorwissen im $\mathbb{R}^3$ angeknüpft werden. Um dies zu erreichen, wurden im Lernskript die Aufgaben 3.1, **??** (Kapitel 4) und 5.1 (Kapitel 5) hinzugefügt. Alle diese Aufgaben behandeln ein Konzept in $\mathbb{R}^3$, welches anschliessend auf den $\mathbb{R}^n$ verallgemeinert wird. Als Beispiel schauen wir uns Aufgabe 5.1 aus Kapitel 3 genauer an. In diesem Kapitel geht es darum, einen Vektor $\vec{p}$ auf einen Unterraum zu projizieren, der von einer Familie von orthonormalen Vektoren $\vec{v}_1, \ldots, \vec{v}_n$ aufgespannt wird. Eine Projektion im $\mathbb{R}^n$ ist relativ abstrakt. Als Unterstützung

wurde daher zuerst Aufgabe 5.1 gestellt. Diese behandelt die analoge Projektion mit Vektoren im $\mathbb{R}^3$. Die Komponenten der beteiligten Vektoren sind dabei explizit gegeben und Abbildung 8 veranschaulicht die Situation. Damit können die Lernenden hoffentlich an ihr Vorwissen im $\mathbb{R}^3$ anknüpfen und den Rest des Kapitels bearbeiten.

7 Holistic mental model confrontation

Nachdem ein neues, komplexes Modell unterrichtet wurde bietet es sich an, eine Selbsterklärungsaufgabe zu stellen. Dabei sollen die Lernenden dieses neue Konzept selber erklären. Allerdings kann eine weitere Methode der Wissensicherung unter Umständen noch erfolgreicher sein, die *holistic mental model confrontation* (kurz HMMC). Dabei werden die Lernenden aufgefordert, das (korrekte) Expertenmodell mit einem (fehlerhaften) Laienmodell zu vergleichen. Dabei sollten im Laienmodell typische Misskonzepte abgebildet sein. In [6] wurde diese Methode mit einer gewöhnlichen Selbsterklärungsaufgabe verglichen. Thema war der Blutkreislaufes des Menschen. Dieser wurde mit einem fehlerhaften und einem exakten Modell dargestellt. Eine Gruppe von Lernenden musste die zwei Modelle vergleichen, während die andere lediglich das exakte Modell erklären musste. Es hat sich herausgestellt, dass die Gruppe mit der HMMC das exakte Modell besser verstanden hatte.

Die Auffassung von Bildern als Punkte oder Vektoren im $\mathbb{R}^n$ sind eine effektive, Modellvorstellung um die Konstruktion der Eigengesichter aus Kapitel 3 zu verstehen. Dieses Modell ist aber sehr abstrakt, da es sich um Vektoren in $\mathbb{R}^n$ mit $n \gg 3$ handelt. Andererseits können diese Vektoren für $n = 2$ einfach in ein Koordinatensystem eingetragen werden, auch wenn die Dimensionen dann nicht mehr unserem Anwendungsfall entsprechen. Genau diese extreme Vereinfachung fassen wir als Laienmodell auf. In Aufgabe 3.6 soll dieses Laienmodell mit mit dem exakten Modell verglichen werden.

Literatur

[1] Lorin W Anderson, David R Krathwohl, Peter W Airasian, Kathleen A Cruikshank, Richard E Mayer, Paul R Pintrich, James Raths, and Merlin C Wittrock. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives, abridged edition*. 2000.

[2] B. S. Bloom, M. B. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. *Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain*. Longmans Green, New York, 1956.

[3] Frank Borsch. *Kooperatives Lernen : Theorie - Anwendung - Wirksamkeit*. Lehren und Lernen. Verlag W. Kohlhammer, Stuttgart, 3., aktualisierte auflage. edition, 2019.

[4] Paulo F. Carvalho and Robert L. Goldstone. Effects of interleaved and blocked study on delayed test of category learning generalization. *Frontiers in Psychology*, 5:936, 2014.

[5] Bor-Chun Chen, Chu-Song Chen, and Winston H. Hsu. Cross-age reference coding for age-invariant face recognition and retrieval. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[6] Soniya Gadgil, Timothy J. Nokes-Malach, and Michelene T.H. Chi. Effectiveness of holistic mental model confrontation in driving conceptual change. *Learning and Instruction*, 22(1):47–61, 2012.

[7] Sofie M.M. Loyens, Suzanne H. Jones, Jeroen Mikkers, and Tamara van Gog. Problem-based learning as a facilitator of conceptual change. *Learning and Instruction*, 38:34–42, 2015.

[8] J. Mckenzie. Beyond technology: Questioning, research, and the information literate school. 2000.

[9] Kevin Miller. Representational tools and conceptual change: The young scientist's tool kit. *Journal of Applied Developmental Psychology*, 21(1):21–25, 2000.

[10] Claudia Mähler and Elsbeth Stern. Transfer. *Handwörterbuch Pädagogische Psychologie*, pages 782–793, 2006.

[11] D. Rohrer and H. Pashler. Recent research on human learning challenges conventional instructional strategies. *Educational Researcher*, 39:406 – 412, 2010.

[12] Doug Rohrer, Robert F. Dedrick, and Kaleena Burgess. The benefit of interleaved mathematics practice is not limited to superficially similar kinds of problems. 2014.

[13] Doug Rohrer and Kelli Taylor. The shuffling of mathematics problems improves learning. *Instructional Science*, 35(6):481–498, 2007.

[14] Wolfgang Schnotz. *Pädagogische Psychologie*, volume 284. Beltz, PVU Weinheim, 2006.

[15] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America. A, Optics and image science*, 4:519–24, 04 1987.

[16] Elsbeth Stern, Carmela Aprea, and Hermann G. Ebner. Improving cross-content transfer in text processing by means of active graphical representation. *Learning and Instruction*, 13(2):191–203, 2003. External and Internal Representations in Multimedia Learning.

[17] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[18] Murat Taskiran, Nihan Kahraman, and Cigdem Eroglu Erdem. Face recognition: Past, present and future (a review). *Digital Signal Processing*, 106:102809, 2020.

[19] Matthew Turk and Alex Pentland. Face recognition using eigenfaces. *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991.