Data Mining & Pattern Recognition

Versuch 6:
Gesichtserkennung

Datum:	
Namen:	
Unterschrift Betreuer:	

Prof. Dr. Johannes Maucher
Medieninformatik Bachelor (MIB)
WS 12/13

Inhaltsverzeichnis

1 Einführung

1.1 Lernziele

In diesem Versuch sollen Kenntnisse in folgenden Themen vermittelt werden:

- **Gesichtserkennung** mit der Eigenface Methode
- **Principal Component Analysis**

1.2 Protokoll

Abzugeben sind pro Gruppe spätestens vor Beginn des nächsten Versuchs:

- Antworten auf alle in dieser Anleitung gestellten Fragen. Die Antworten sollten in einem elektronischen Dokument verfasst und als .pdf bereitgestellt werden. Die Antworten sind entsprechend der Kapitelnummerierung in dieser Anleitung zu ordnen.
- Alle Dateien von denen in dieser Anleitung ein Abspeichern gefordert wird.

Die Durchführung und Dokumentation eigener Experimente, die über die Aufgabenstellungen in diesem Dokument hinausgehen, wird ausdrücklich unterstützt und in der Bewertung entsprechend berücksichtigt.

2 Theorie zur Vorbereitung

Die Gesichtserkennung kann mit unterschiedlichen Ansätzen realisiert werden. In diesem Versuch wird ausschließlich der *Eigenface*-Ansatz vorgestellt. Dieser Ansatz basiert auf der *Principal Component Analysis (PCA)* und wurde erstmals in [TP] vorgestellt. Die Eigenface-Methode weist eine gute Performance im Fall biometrisch aufgenommener Gesichtsbilder auf.

2.1 Das Prinzip der Eigenface Gesichtserkennung

Bilder mit C Pixeln in der Breite und R Pixeln in der Höhe können als $R \times C$ Matrizen abgespeichert werden. Handelt es sich um ein Schwarz-Weiß- oder Graustufen-Bild, dann wird pro Bild nur eine derartige Matrix benötigt. Der Eintrag in der i.ten Zeile und j.ten Spalte dieser Matrix definiert den Grauwert des entsprechenden Pixels. In Farbbildern werden je nach benutztem Farbraum mehre Matrizen pro Bild benötigt, wobei jede Matrix einen Farbkanal des Bildes repräsentiert. Für ein RGB-Bild werden z.B. 3 Matrizen für die Farbkanäle Rot, Grün und Blau benötigt.

Im Folgenden wird von quadratischen Graubildern mit $N \times N$ Pixeln ausgegangen. Wird jedes Pixel als ein Merkmal betrachtet, dann existieren insgesamt N^2 Merkmale, das Bild kann auch als ein Punkt im N^2-dimensionalen Raum betrachtet werden. Bilder der Auflösung 256×256 müßten also im 65536-dimensionalen Raum beschrieben werden. Entsprechend komplex wäre die notwendige Verarbeitung. Ist jedoch bekannt, dass in einer Menge von Bildern jeweils ein gleichartiges Objekt abgebildet ist, z.B. wenn alle Bilder ausschließlich je ein Gesicht enthalten, dann existieren große Abhängigkeiten zwischen diesen Bildern. Geometrisch ausgedrückt bedeutet dies, dass die Punkte, welche die Menge der gleichartigen Bilder beschreiben, nicht gleichmäßig über den N^2-dimensionalen Raum verteilt sind, sondern in einen relativ kleinen Unterraum mit $K << N^2$ Dimensionen nahezu vollständig beschrieben werden können. Jede dieser K Dimensionen beschreibt ein für die Kategorie (z.B. Gesichtsbilder) relevantes Merkmal. Im Fall der Gesichtserkennung werden die relevanten Merkmale auch als Eigenfaces bezeichnet. Jedes Eigenface kann als Bild dargestellt werden, welches ein bestimmtes Gesichtsmerkmal besonders

hervorhebt. Jedes individuelle Bild der Kategorie (d.h. jedes Gesicht) kann dann als Linearkombination der K relevanten Merkmale (der K Eigenfaces) beschrieben werden.

Das Problem besteht nun zunächst darin, aus einer Menge von Bildern der gleichen Kategorie die relevanten Merkmale zu finden. Dieses Problem wird durch die Principal Component Analysis (PCA) gelöst. Die PCA, findet in einer Menge von Bildern der gleichen Kategorie die Hauptachsen, also die Richtungen im N^2-dimensionalen Raum, entlang derer die Varianz zwischen den gegebenen Bildern am stärksten ist. Der N^2-dimensionale Pixelraum wird dann in einen Raum, der durch die gefundenen Hauptachsen aufgespannt wird, transformiert. In diesem in der Anzahl der Dimensionen stark reduzierten Raum wird dann die Bilderkennung durchgeführt. Der hier skizzierte Ansatz der Eigenfaces für die Gesichtserkennung wurde erstmalig in [TP] beschrieben.

2.2 Genereller Ablauf

Die Bilderkennung besteht aus 2 Phasen. In der Trainingsphase werden die Gesichtsbilder der zu erkennenden Personen eingelesen und für diese mit der PCA der Eigenface-Raum berechnet. In der Erkennungsphase wird ein neu aufgenommenes Bild in den Eigenface-Raum transformiert und dort dem naheliegendsten Bild aus der Trainingsmenge zugeordnet.

2.2.1 Trainingsphase

1. Lese Gesichtsbilder der Personen, die erkannt werden sollen ein. Die Menge dieser Bilder definiert das Trainingsset
2. Berechne mit der PCA den Eigenface-Raum. Dabei werden nur die K Dimensionen, welche zu den Eigenvektoren mit den größten Eigenwerten gehören ausgewählt. Die zu den K Dimensionen (Eigenvektoren) gehörenden Bilder sind die Eigenfaces.
3. Transformiere jedes Bild der Trainingsmenge in den Eigenface-Raum und erhalte so die entsprechende Repräsentation des Bildes als Punkt im Eigenface-Raum.

2.2.2 Erkennungsphase

1. Transformiere das zu erkennende Bild in den Eigenface-Raum und berechne dort die Koordinaten des Bildes hinsichtlich aller K-Dimensionen (Eigenfaces)
2. Bestimme ob das zu erkennende Bild überhaupt eine Gesicht darstellt
3. Bestimme ob das Gesicht zu einer bekannten Person, deren Bild in der Trainingsmenge enthalten ist, gehört.

Update (optional)

1. Füge das erkannte Bild zur Menge der Trainingsbilder hinzu und führe die Schritte der Trainingsphase durch.

2.3 Bestimmung der Eigenfaces

Es werden zunächst M Gesichtsbilder der zu erkennenden Personen eingelesen (von jeder zu erkennenden Personen mindestens ein Bild). Es wird davon ausgegangen, dass jedes der Bilder C Pixel breit und R Pixel hoch ist. Das Bild kann dann als $R \times C$ Matrix dargestellt werden. Im Fall eines Graustufenbildes repräsentieren die Pixelwerte den entsprechenden Grauwert. Nach dem Einlesen werden die Bildmatrizen

als eindimensionale Vektoren dargestellt. Für diese Umformung werden die Zeilen jeder Matrix von oben nach unten ausgelesen und hintereinander gereiht. Jedes Bild wird dann durch einen Vektor der Länge $Z = R \cdot C$ repräsentiert und kann als Punkt im Z-dimensionalen Raum dargestellt werden. Die M Bildvektoren werden im folgenden mit $\Gamma_1, \Gamma_2, \ldots, \Gamma_M$ bezeichnet.
Im nächsten Schritt wird das Durchschnittsbild berechnet:

$$\overline{\Gamma} = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i \tag{1}$$

Dieses Durchschnittsbild wird von allen Bildern Γ_i abgezogen. Die Menge der so gewonnenen Bildrepräsentationen

$$\Phi_i = \Gamma_i - \overline{\Gamma} \tag{2}$$

ist dann mittelwertsfrei. Die Menge $\Phi_1, \Phi_2, \ldots, \Phi_M$ wird dann einer Principal Component Analysis (PCA) ([PCA]) unterzogen. Hierzu werden die mittelwertfreien Bildrepräsentationen Φ_i in die Spalten einer Matrix geschrieben. Diese Matrix wird im Folgenden mit X bezeichnet. Unter der Annahme, dass die Φ_i bereits als Spaltenvektoren vorliegen, ist die Matrix X definiert als:

$$X = [\Phi_1, \Phi_2, \ldots, \Phi_M]\,. \tag{3}$$

Die entsprechende Kovarianzmatrix ist dann

$$CV = X \cdot X^T. \tag{4}$$

Für die PCA müssten als nächstes eigentlich die Eigenvektoren und Eigenwerte der Kovarianzmatrix CV berechnet werden. Für den vorliegenden Fall kann allerdings die hierfür notwendige Berechnung aus Komplexitätsgründen nicht realisiert werden. Man beachte dass die Matrix CV Z Spalten und Z Zeilen enthält (Z ist die Anzahl der Pixel in einem Bild) und für diese Z Eigenvektoren und Eigenwerte existieren. Wie in [TP] beschrieben existieren im Fall, dass die Anzahl der Bilder M wesentlich kleiner als die Anzahl der Pixel Z ist, nur $M-1$ relevante Eigenvektoren, die Eigenwerte aller anderen Eigenvektoren liegen nahe bei Null. Der in [TP] beschriebene Ansatz geht nun von der $M \times M$ Matrix

$$X^T \cdot X$$

aus, für welche die Eigenvektoren und Eigenwerte für eine moderate Bildanzahl M gut berechnet werden können. Per Definition gilt für die Eigenvektoren $\mathbf{v}_i$ und Eigenwerte μ_i dieser Matrix:

$$X^T \cdot X \cdot \mathbf{v}_i = \mu_i \mathbf{v}_i. \tag{5}$$

Werden beide Seiten dieser Matrix linksseitig mit der Matrix X multipliziert,

$$X \cdot X^T \cdot X \cdot \mathbf{v}_i = \mu_i X \mathbf{v}_i, \tag{6}$$

dann ist daraus zu erkennen, dass die M Vektoren

$$\mathbf{u}_i = X \mathbf{v}_i \tag{7}$$

die Eigenvektoren der Matrix $CV = X \cdot X^T$ sind. D.h. es können zunächst die M Eigenvektoren der relativ kleinen Matrix $X^T \cdot X$ bestimmt und aus diesen durch eine einfache Multiplikation mit der Matrix X die relevanten Eigenvektoren der Matrix CV berechnet werden. Da die Matrix X die M Bildrepräsentationen Φ_i als Spalten enthält, können die gesuchten Eigenvektoren auch als Linearkombination der M Bilder der Trainingsmenge beschrieben werden:

$$\mathbf{u}_i = \sum_{k=1}^{M} v_{i,k}\Phi_k \tag{8}$$

wobei mit $v_{i,k}$ die k.te Komponente des Vektors $\mathbf{v}_i$ bezeichnet wird. Die Eigenvektoren $\mathbf{u}_i$ werden auch Eigenfaces genannt. Per Definition sind die Eigenvektoren paarweise orthogonal. Jeder Eigenvektor ist ein Spaltenvektor mit Z (=Anzahl der Pixel) Komponenten.
Die M Eigenvektoren werden dann entsprechend der Größe der zugehörigen Eigenwerte μ_i geordnet. Für die weiteren Schritte kann zum Zwecke einer weiteren Komplexitätsreduktion eine Untermenge der K relevantesten Eigenvektoren benutzt werden (also der K Eigenvektoren mit den höchsten Eigenwerten). Beispielsweise haben in [TP] für die Erkennung von $M = 16$ Personen und eine Auflösung von 256×256 Pixel meist $K = 7$ Eigenvektoren für eine gute Erkennung ausgereicht.

2.4 Gesichtserkennung im Eigenspace

Die K ausgewählten Eigenvektoren $\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_K$ spannen einen $K-$dimensionalen Raum, den sogenannten Eigenspace auf. Die K Vektoren repräsentieren die K Merkmale hinsichtlich derer die Bilder der Trainingsdatenmenge am stärksten variieren.

Für die Bilderkennung wird jetzt jedes Bild, also sowohl die Bilder aus der Trainingsmenge als auch die zu erkennenden Bilder, in den Eigenspace transformiert. Jedes Bild stellt einen Punkt im Eigenspace dar. Für die Erkennung kann einfach die Distanz des zu erkennenden Bildes zu allen Bildern der Trainingsmenge berechnet werden. Das zu erkennende Bild wird der Person (Bildklasse) zugeordnet, deren Punkt im Eigenspace dem Punkt des zu erkennenden Bildes am nächsten liegt.

Die K Komponenten eines Trainingsbildes werden berechnet, indem das Bild auf den jeweiligen Eigenvektor projiziert wird. Demnach ist die k.te Komponente des i.ten Trainingsbildes Φ_i:

$$\omega_{k,i} = \mathbf{u}_k^T \Phi_i \tag{9}$$

Der dem Bild Φ_i entsprechende Punkt im Eigenspace ist dann

$$\mathbf{w}_i = [\omega_{1,i}, \omega_{2,i}, \ldots, \omega_{K,i}]. \tag{10}$$

Wird mit Γ das zu erkennende Bild und mit $\Phi = \Gamma - \overline{\Gamma}$ die um den Mittelwert der Trainingsbilder subtrahierte Version des Bildes bezeichnet, dann sind

$$\omega_k = \mathbf{u}_k^T \Phi \tag{11}$$

die Koordinaten der Projektion von Φ in den Eigenspace und der dieses Bild repräsentierende Punkt

$$\mathbf{w} = [\omega_1, \omega_2, \ldots, \omega_K]. \tag{12}$$

Das zu erkennende Bild wird dann dem Trainingsbild Φ_j zugeordnet, für welches gilt:

$$j = argmin_i \{d(\mathbf{w}, \mathbf{w}_i)\} \tag{13}$$

wobei mit $d(\mathbf{w}, \mathbf{w}_i)$ die euklidische Distanz zwischen den Projektionen von Φ und Φ_i bezeichnet wird.

Optional: Falls Φ_i nicht das einzige Bild einer Person in der Trainingsmenge ist, sondern für die entsprechende Person mehrere Trainingsbilder vorliegen, wird in der Distanzberechnung nicht Φ_i, sondern der Mittelwert über alle zu dieser Person gehörenden Bilder eingesetzt:

$$\overline{\Phi} = \frac{1}{|W|} \sum_{w \in W} \Phi_w. \tag{14}$$

Dabei bezeichnet W die Menge aller der Indizees w, für die die Φ_w zur gleichen Person gehören[1].

Für die Mindestdistanz

$$\epsilon = \min_i \{d(\Phi, \Phi_i)\} \tag{15}$$

wird in der Regel eine Schwelle T definiert. Wenn $\epsilon > T$ ist, also eine relativ große Distanz zwischen dem zu erkennenden Bild und dem nächstliegenden Bild aus der Trainingsmenge besteht, wird davon ausgegangen, dass es sich um ein unbekanntes Gesicht handelt. Optional könnte dieses unbekannte Gesicht in die Trainingsmenge aufgenommen werden.

Schließlich muss noch der Fall behandelt werden, dass das eingelesene Bild kein Gesicht darstellt. Aufgrund der starken Projektion vom ursprünglichen Bildraum in den Eigenspace kann dieser Fall nicht durch eine Schwelle auf den Fehler ϵ erkannt werden. Es kann durchaus sein, dass ein Nicht-Gesichtsbild in die Umgebung eines Gesichtsbild im Eigenspace projiziert wird. Ein Nicht-Gesichtsbild wird aber eine relativ große Distanz $d(\Phi, \Phi_f)$ zwischen

$$\Phi = \Gamma - \overline{\Gamma} \tag{16}$$

und der Repräsentation im Eigenspace

$$\Phi_f = \sum_{k=1}^{K} \omega_k \mathbf{u}_k \tag{17}$$

aufweisen. Durch die Definition einer weiteren Schwelle S auf $d(\Phi, \Phi_f)$ kann also erkannt werden, ob es sich überhaupt um ein Gesicht handelt [2].

2.5 Vor dem Versuch zu klärende Fragen

- Was sind Eigenvektoren und Eigenwerte?
- Was versteht man unter Eigenfaces?
- Die PCA ist u.a. im entsprechenden Kapitel des Dokuments [PCAJM] beschrieben. Wie kann mit der PCA eine Dimensionalitätsreduktion durchgeführt werden?
- Wie werden mit dem Python Modul *Image* Bilder in ein Python-Programm geladen?

[1] Im Praktikumsversuch muss diese Option nicht implementiert werden. Die im folgenden Abschnitt beschriebene Versuchsdurchführung bezieht sich auf den Fall, dass nur die Distanz zu Einzelbildern berechnet wird und das nächstliegende Bild ausgegeben wird

[2] Im Versuch ist davon auszugehen, dass nur Gesichtsbilder verwendet werden, d.h. es muss nur der Test gegen die Schwelle ϵ implementiert werden

3 Durchführung

3.1 Vorbereitung

Für diesen Versuch sind je 3 verschiedene Bilder aller Praktikumsteilnehmer gegeben. Partitionieren Sie die Menge aller Bilder in

- ein Verzeichnis `training`, welches je 2 Bilder jedes Studenten enthält,
- ein Verzeichnis `test`, welches die nicht in `training` enthaltenen Bilder enthält (also ein Bild von jedem Studenten).

Ziehen Sie als Ausgangspunkt für diese Übung die auf dem Skripteserver bereitgestellte Datei `faceRecognitionTemplate.py` heran und erweitern Sie diese mit der im Folgenden beschriebenen Funktionalität. In der Datei bereits enthalten ist

- Der Aufruf eines Dialogs zur Auswahl
  - des Verzeichnisses, in welchem sich die Bilder für das Training befinden,
  - der Datei, welche das zu erkennende Bild enthält
- die Funktion `parseDirectory(TrainDir,Extension)`. Ihr wird das Verzeichnis der Trainingsbilder und die File Extension der Bilder (.png) übergeben. Die Funktion liefert eine Liste aller vollständigen Dateinamen mit Pfad, Dateiname und Extension aller Dateien im Verzeichnis `TrainDir` mit der Endung `Extension`

3.2 Phase Training

3.2.1 Image Akquisition

Implementieren Sie in dem gegebenen File `faceRecognitionTemplate.py` eine Funktion `generateListOfImgs(listOfTrainFiles)`. Der Funktion wird die Liste aller Dateinamen der Trainingsbilder übergeben. Die Funktion gibt eine Liste von Bildern zurück. Verwenden Sie hierfür das Python Bildverarbeitungspaket `Image`. Die Bilder der zurückzugebenden Liste sollen `Image`-Objekte sein (siehe [PIL]).
Zeigen Sie zur Kontrolle die Breite und Höhe der Bilder an.

3.2.2 Umwandlung in Numpy-Arrays

Numpy (siehe [NP]) ist ein Python Paket für mathematische Datenmodellierung. Das Paket stellt eine breite Palette von mathematischen Operationen und Funktionen zur Verfügung. Im Verbund mit weiteren Python-Bibliotheken wie *Scipy* und *Matplotlib* stellt Numpy eine interessante, kostenfreie Alternative zu Mathematik-Tools wie z.B. Matlab dar. Als Referenz für Numpy wird [NP] empfohlen.

Die im vorigen Schritt erzeugte Liste von `Image`-Objekten (`imgList`) soll nun mit der Methode `convertImgListToNumpyData(imgList)` in ein Numpy Array mit Float-Werten (Typ `numpy.asfarray`) umgewandelt werden. Jede Zeile des 2-dimensionalen Numpy-Arrays soll ein Bild, repräsentiert als 1-dimensionaler Vektor, enthalten. D.h. die Anzahl der Spalten des Numpy-Array ist die Anzahl der Pixel in einem Bild. Bei der Umwandlung muss ausserdem eine Normierung aller Werte in den Bereich zwischen 0 und 1 durchgeführt werden. Hierzu müssen alle Pixelwerte eines Bildes durch den im jeweiligen Bild vorkommenden Maximalwert geteilt werden.

3.2.3 Berechnung des Durschnittsbildes und Normierung

Die von der Funktion `convertImgListToNumpyData(imgList)` zurückgelieferte Matrix enthält in ihren Zeilen alle Trainingsbilder. Aus diesen Trainingsbildern ist nach Gleichung (1) das Durchschnittsbild zu berechnen, z.B. durch Anwendung der Numpy-Funktion `average`. Das Durchschnittsbild ist von allen Bildern abzuziehen (Gleichung (2)). Das daraus resultierende Numpy-Array enthält die mittelwertfreien Repräsentationen der Trainingsbilder und wird im Folgenden mit `NormedArrayOfFaces` bezeichnet.

Wichtiger Hinweis: Das Numpy-Array `NormedArrayOfFaces` ist die Transpornierte X^T der Matrix X aus Gleichung (3).

3.2.4 Berechnung der Eigenfaces

Implementieren Sie die Funktion `calculateEigenfaces(adjfaces,width,height)`. Dieser Funktion werden die normierten Bilder `NormedArrayOfFaces` zusammen mit der Bildbreite und -Höhe übergeben. Zurück liefert die Funktion ein Numpy-Array, dessen Zeilen die berechneten normierten Eigenfaces sind. Die Berechnung der Eigenfaces ist im Theorieteil Abschnitt 2.3 beschrieben.
Für die Python-Implementierung können Sie folgende Hinweise berücksichtigen:

- Berechnung der transponierten eines Numpy-Arrays A mit der Numpy-Methode `transpose()`
- Matrixmultiplikation zweier Numpy-Arrays A und B mit der Numpy-Funktion `dot()`
- Berechnung der Eigenvektoren und Eigenvalues eines Numpy Arrays A mit der Numpy-Funktion `eigh()`
- Sortierung von Numpy-Arrays mit den Numpy-Funktionen `sort()` und `argsort()`.

Aus dem von der Funktion `calculateEigenfaces(adjfaces,width,height)` zurück gelieferten Array von Eigenfaces sind die K relevantesten auszuwählen. Dieses reduzierte Array wird im Folgenden mit `Usub` bezeichnet. Im Versuch kann $K = 6$ eingestellt werden.

3.2.5 Transformation der normierten Trainingsbilder in den Eigenface Raum

Die im vorigen Schritt angelegten K relevantesten Eigenfaces bilden den sogenannten *Eigenface-Raum*. Für jedes der normierten Trainingsbilder, also für jede Zeile aus `NormedArrayOfFaces`, sind die Koordinaten im Eigenface-Raum entsprechend der in Gleichung (9) definierten Transformation zu berechnen.

3.3 Phase Erkennung

Das zu erkennende Bild ist zunächst als Image-Objekt einzulesen und danach als ein Numpy-Array darzustellen. Eine Normierung der Pixelwerte in den Bereich zwischen 0 und 1 ist durchzuführen. Schließlich muss auch von diesem Bild das Durchschnittsbild aller Trainingsbilder abgezogen werden. Diese Prozessschritte entsprechen der oben beschriebenen Vorverarbeitung der Trainingsbilder. Das resultierende normierte und mittelwertfreie Bild wird im Folgenden mit `NormedTestFace` bezeichnet.

Danach sind die Koordinaten des `NormedTestFace` im Eigenface-Raum nach Gleichung (11) zu berechnen und das in diesem Raum nächstliegende Trainingsbild entsprechend Gleichung (13) zu bestimmen.

3.4 Ausgabe und Aufgaben

Anzuzeigen sind:

- das zu erkennende Bild und das gefundene nächstliegende Trainingsbild. Die Erzeugung und Anzeige eines neuen Image-Objektes ist in [PIL] beschrieben.
- die Distanz zwischen dem zu erkennenden Bild und dem nächstliegenden Trainingsbild.

Aufgaben:

1. Ab welcher Anzahl K von verwendeten Eigenvektoren treten Fehlklassifikationen ein?
2. Wie groß ist dann die Mindestdistanz zwischen Test- und nächstliegendem Trainingsbild?
3. Wie ändert sich die Distanz zwischen Bildern, wenn die Anzahl der Eigenvektoren reduziert wird?
4. Wie könnte dieser Einfluss der Eigenvektor-Anzahl auf die Mindestdistanz reduziert werden?
5. Nennen Sie zwei Algorithmus-unabhängige Parameter, die starken Einfluss auf Rate korrekter Gesichtserkennungen haben.

Literatur

[TP] M. Turk, A. Pentland; Eigenfaces for Recognition; Journal of Cognitive Neuroscience 3(1), 7186; `http://www.cs.ucsb.edu/%7Emturk/Papers/jcn.pdf`

[PCA] L.I. Smith; Tutorial on Principal Component Analysis `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`

[MKI] Johannes Maucher; Vorlesung Einführung in die Künstliche Intelligenz - Maschinelles Lernen Teil 1; `http://www.hdm-stuttgart.de/~maucher/Lecture_KI.html`

[PCAJM] Johannes Maucher; Beschreibung der Principal Component Analysis; `https://docs.google.com/document/d/13cc9RGeIvsV5JsC-MsymJ6aaJs_ZsUKEwqaOOO8IL8Q/edit`

[PIL] Python Imaging Library Handbook `http://www.pythonware.com/library/pil/handbook/`

[NP] Python Reference Guide; `http://docs.scipy.org/doc/numpy/reference/`