Methodische und Praktische Grundlagen der Informatik 4

Aufgabenblatt 2

mpgi4@cg.tu-berlin.de

WiSe 2014/2015

Allgemeine Hinweise:

- Die Aufgaben sollen in Gruppen bestehend aus zwei bis drei Personen bearbeitet werden. Ausnahmen müssen mit dem jeweiligen Tutor abgesprochen werden.
- Bitte reichen Sie Ihre Lösungen in Form einer ZIP Datei bis **Donnerstag, den 04.12.2014, um 12:00 Uhr** auf der ISIS Webseite der Vorlesung ein. Tragen Sie in jede abgegebene Datei Name und Email-Adresse aller Gruppenmitglieder ein.
- Wenn eine Aufgabe die Abgabe einer Grafik verlangt, dann muss ein vollständig funktionsfähiges Programm in der Lösung enthalten sein, welches bei der Ausführungen die Grafik erstellt.
- Verwenden Sie den vorgegebenen Skelettcode, um die Aufgaben umzusetzen.
- Ihr Lösung sollte in einem Ordner mit dem Namen `GruppeXY` abgegeben werden, wobei `XY` die Nummer Ihrer Gruppe ist.

Aufgabe 1: Potenzmethode für Eigenvektoren (3 Punkte)

Ziel der Aufgabe ist es, die Datei `power_iteration.py` zu vervollständigen und mit Hilfe der Potenzmethode zur Berechnung von Eigenvektoren die Gleichgewichtslösung einer Markow-Kette zu bestimmen.

Eine Markow-Kette beschreibt die Zeitentwicklung eines nicht-deterministischen Prozesses, wenn der Übergang zwischen aufeinander folgenden Zuständen durch eine Wahrscheinlichkeit beschrieben ist. Um das Konzept besser zu verstehen, werden wir eine Markov-Kette verwenden, um ein einfaches Modell zur Wettervorhersage zu erhalten. Wir beschränken uns dabei darauf, das Wetter in drei Kategorien $\{$ `Sonne`, `Wolken`, `Regen` $\}$ einzuteilen. Durch lange Beobachtungen haben wir festgestellt, dass mit einer Wahrscheinlichkeit von 0.25 auf einen sonnigen Tag ein regnerischer folgt, mit einer Wahrscheinlichkeit von 0.5 auf einen regnerischen Tag ein weiterer regnerischer folgt, usw. All diese Wahrscheinlichkeiten können wir in einer Tabelle darstellen:[1]

	`Regen`	`Wolken`	`Sonne`
`Regen`	0.5	0.25	0.25
`Wolken`	0.5	0.0	0.5
`Sonne`	0.25	0.25	0.5

Die Tabelle können wir als Matrix T auffassen, welche beschreibt, mit welcher Wahrscheinlichkeit ein bestimmtes Wetter morgen auftreten wird ausgehend vom heutigen Wetter. Nach unserem Modell hängt das Wetter für morgen also nur vom heutigen Wetter ab. Diese Abhängigkeit der Folgezustände nur vom aktuellen Zustand ist die definierende Eigenschaft einer Markow-Kette.

Eine nützliche Eigenschaft von Markow-Ketten ist, dass man von einem aktuellen Zustand neue Zustände erzeugen kann, welche dem Modell entsprechen. Dazu müssen wir nur die Transitionsmatrix T verwenden, um in neue Zustände entsprechen der Wahrscheinlichkeiten zu wechseln, welche die Element von T angeben. Dazu können wir einen Zufallsgenerator verwenden, welcher gleichverteilte Zahlen zwischen $[0, 1]$ erzeugt, und dann entsprechend der kumulativen Wahrscheinlichkeitsverteilung der Zeilen der Transitionsmatrix in neue Zustände wechseln. Zum Beispiel, wenn heute ein Regentag ist, dann wird der nächste Tag auch ein Regentag, wenn die Zufallszahl in $[0.0, 0.5)$ liegt, es wird ein Wolkentag mit der Zufallszahl in $[0.5, 0.75)$ und ansonsten ein sonniger Tag. Auf diese Art und Weise können wir

[1] Das Beispiel ist von (Grinstead, C. M., Snell, J. L. (2006). Introduction to Probability. American Mathematical Society) adaptiert.

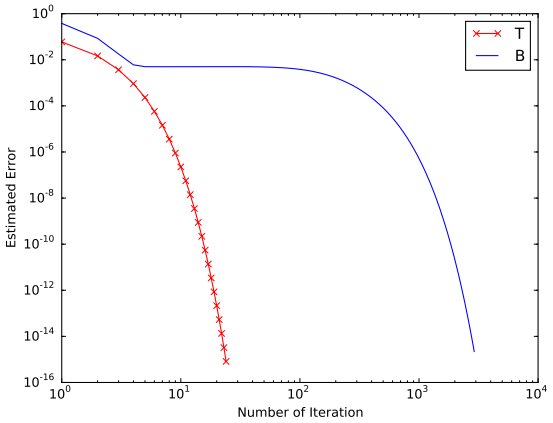
das Wetter für eine beliebige Anzahl von Tagen "vorhersagen". Wenn wir eine solche Vorhersage lange genug machen, dann werden wir für (die allermeisten) Markow-Ketten feststellen, dass sich eine stabile Häufigkeitsverteilung für die Zustände ergibt. In unserem Beispiel entspricht dies der Häufigkeit, mit welcher die drei Wetterzustände auftreten, zum Beispiel wenn wir über den Zeitraum eines Jahres eine tägliche Beobachtung durchführen. Da diese Verteilung stabil ist, wird sie als *Gleichgewichtsverteilung* bezeichnet. Mathematisch ist diese Verteilung als der *linke* Eigenvektor u_1 von T beschrieben, welche mit dem Eigenwert 1 assoziert ist, d.h. $u_1^T T = u_1^T$; da $b^T A = A^T b$ für eine beliebige Matrix A und einen beliebigen Vektor b so können wir u_1 auch als rechten Eigenvektor der transponierten Transitionsmatrix charakterisieren: $T^T u_1 = u_1$. Dieser Zusammenhang zwischen Gleichgewichtsverteilung und Eigenvektoren folgt in der Tat aus der Definition eines Eigenvektors als ein Vektor, welcher, bis auf eine mögliche Skalierung, invariant bezüglich der Matrix ist. Man kann darüber hinaus zeigen, dass eine Transitionsmatrix T keinen größeren Eigenwert hat. Damit können wir die Potenzmethode verwenden, um die Gleichgewichtsverteilung für unser Beispiel zu berechnen.

a) Implementieren Sie die Potenzmethode zur Berechnung des Eigenvektors welcher mit dem größten Eigenwert assoziert ist in der Funktion `powerIteration()` in `power_iteration.py`. Es soll angenommen werden, dass die übergebene Matrix die Voraussetzungen für die Methode erfüllt. Die Methode soll natürlich für beliebige Matrizen und nicht nur die obige Beispielmatrix funktionieren.

b) Berechnen Sie die Gleichgewichtslösung für die obige Markow-Kette mit Ihrer Implementierung der Potenzmethode, d.h. bestimmen sie die Häufigkeitsverteilung für das Wetter. Geben Sie den bei Ihrer Berechnung auftretenden Fehler an. Berechnen Sie mit Ihrer Implementierung zusätzlich den ersten Eigenvektor für die Matrix

$$B = \begin{pmatrix} 0.6552 & -0.3079 & -0.2479 \\ -0.3079 & 0.7251 & -0.2202 \\ -0.2479 & -0.2202 & 0.8098 \end{pmatrix}. \tag{1}$$

Erstellen Sie die Grafik `power_iteration_err.pdf`, welche für beide Matrizen den Fehler als ein Funktion des Iterationsschrittes darstellt. Die Grafik sollte qualitativ ähnlich zu unseren Ergebnis sein:



c) Warum ist die Konvergenz für die Matrix B wesentlich schlechter?

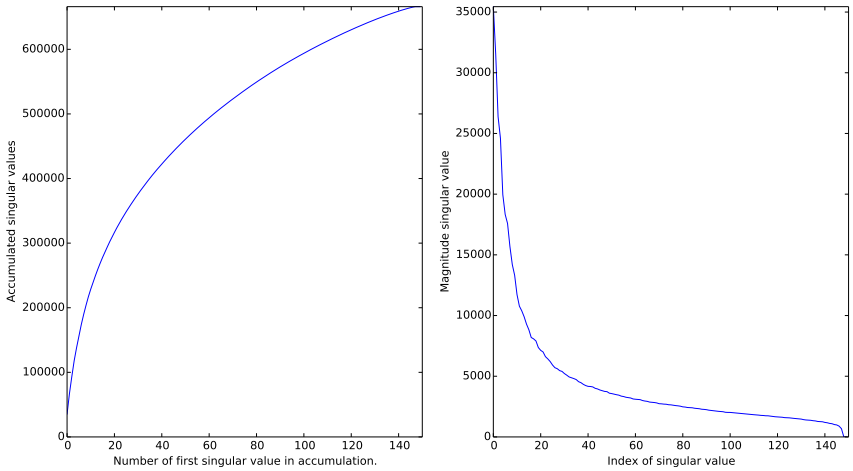
Aufgabe 2: Eigenfaces (6 Punkte)

Ziel der Aufgabe ist es, die Datei `eigenfaces.py` zu vervollständigen und den Eigenface-Algorithmus zur Gesichtserkennung zu implementieren, welcher in der Vorlesung vorgestellt wurde.

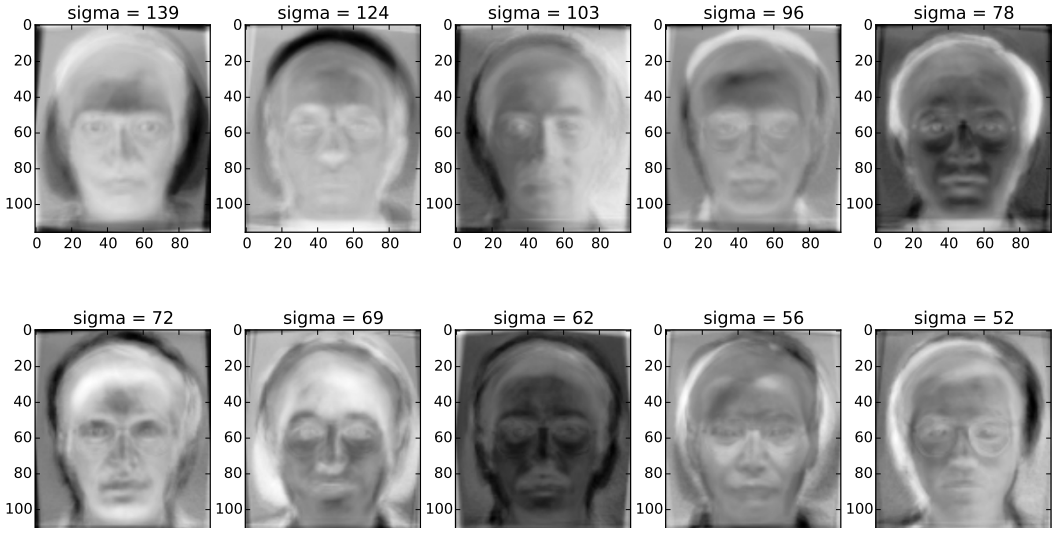
a) Implementieren Sie die Funktion `loadImages()` und laden Sie die Bilder aus dem Verzeichnis `./src/data/train/` in eine Python Liste.

b) Implementieren Sie die Funktion `dataMatrix()` und erzeugen Sie die Datenmatrix für die Bilder, welche Sie unter Teilaufgabe (a) geladen haben.

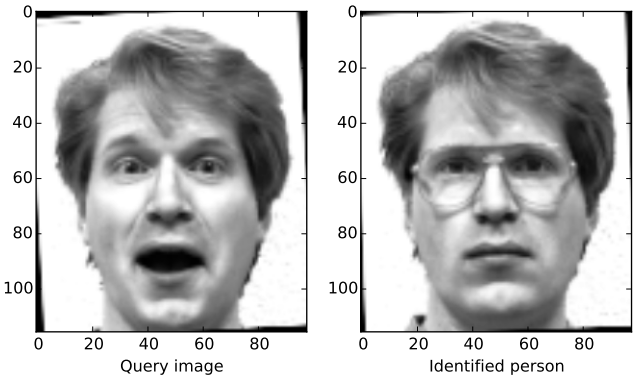
c) Implementieren Sie die Funktion `pca()` und berechnen Sie anhand der Datenmatrix aus Teilaufgabe (b) die Eigenfaces für den Datensatz. Stellen Sie die ersten zehn Eigenfaces mit Hilfe der Funktion `visualizeEigenFaces()` dar. Berechnen Sie den Index k, so dass 80% der akkumulierten Magnitude der Singulärwerte in den ersten k principal components enthalten ist. Verwenden Sie dazu `accumulatedEn()` und stellen sie das Ergebnis mit `plotSingularValuesEn()` graphisch dar.



Die ersten zehn Eigenfaces sind:



d) Laden Sie nun die Bilder in `./src/data/test/`. Verwenden Sie die Projektionen der Bilder auf die Eigenfaces, um in der Funktion `identifyPerson()` für jedes der Testbilder das ähnlichste Bild im ursprünglichen Datensatz zu finden. Als Ähnlichkeitsmaß soll der Winkel zwischen den Bildern im Eigenface-Raum dienen. Stellen Sie mit dem von uns zur Verfügung gestellten Code neben jedem Testbild das ähnlichste Trainingsbild dar. Wie groß ist Ihre Erfolgsquote? Vergleichen Sie die Erfolgsquote, wenn alle und wenn nur die ersten k Eigenfaces verwendet werden. Als Bild sollte man zum Beispiel erhalten:



e) Benutzen Sie Eigenfaces, um Bilder mit Gesichtern zu erkennen. Implementieren Sie dazu die Funktion `recognizeFace()`, welche folgende Fehlerfunktion verwendet:

$$\mathrm{err} = \frac{\|A - \hat{A}\|}{\|A\|} \tag{2}$$

wobei A ein Ausgangsbild ist und $\hat{A}$ die Rekonstruktion nach der Projektion in die Eigenface-Basis. Für die Berechnung von err sind A und $\hat{A}$ als Vektoren aufzufassen. Erstellen Sie mit dem vorgegebenen Code eine Visualisierung der Rekonstruktion und des ursprünglichen Bildes, wie unten für ein Beispiel dargestellt. Was wäre ein guter Grenzwert um zwischen Gesichtern und Nicht-Gesichtern zu unterscheiden?