Fakultät IV - Elektrotechnik und Informatik
Algorithmen und Datenstrukturen SoSe 2016
Brock / Deimel / Foucard / Mabrouk / Sieverling

Aufgabenblatt 10

letzte Aktualisierung: 01. July, 19:06 Uhr

Ausgabe: 3.07.2016
Abgabe: 13.07.2016 23:59

Thema: *Dynamic Programming*

Abgabe

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im svn Ordner
`Tutorien/txx/Studierende/deinname@TU-BERLIN.DE/Abgaben/`
eingecheckt sein:

Geforderte Dateien:	
`Blatt10/src/Matcher.java`	Aufgabe 1.6
`Blatt10/src/Database.java`	Aufgabe 1.7

Wichtige Ankündigungen

- Unit tests dürfen geteilt werden. Lösungen dürfen auf keinen Fall geteilt werden! Wir testen auf Plagiate.
- Verändert die Methodenköpfe vorgegebener Klassen nicht. Verändert keine Klassen die nicht Teil der Abgabe sind und fügt keine neuen hinzu.

1. Aufgabe: Dynamische Programmierung zur Spracherkennung

Spracherkennungssysteme müssen die Ähnlichkeit von gesprochenen Wörtern einschätzen. Eine Schwierigkeit dabei ist, dass das gleiche Wort unterschiedlich lang gesprochen werden kann. Das Problem wird noch komplizierter, da die zeitliche Variation in der Aussprache nicht alle Wortteile gleich betrifft. So bleiben "'t"' und "'p"' meistens unverändert, während "'a"' und "'s"' beliebig in die Länge gezogen werden können.
In dieser Aufgabe werden wir den Dynamic Time Warping Algorithmus implementieren um dieses Problem zu lösen. Dieser Algorithmus basiert auf dem Konzept der Dynamischen Programmierung.

1.1. (Tut) Repräsentation von Audiodateien im Computer

Überlegt, wie eine Audiodatei im Computer abgespeichert wird. Klärt dazu die folgenden Begriffe:

- Zeitreihe (time series)

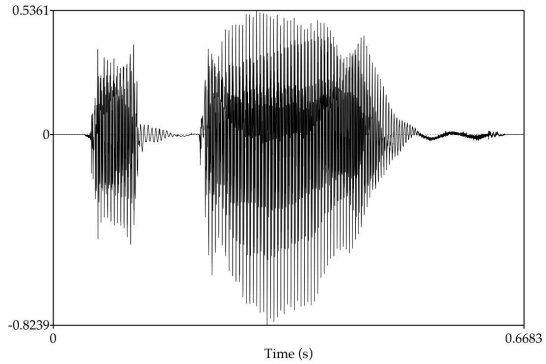

Abbildung 1: Audiodatei. Zu sehen ist die Aussprache des englischen Wortes "'above"' (Quelle: Wikimedia commons)

- Frame / Sample
- Abtastrate (sampling rate)

1.2. (Tut) Vergleich von Zeitreihen

Diskutiert wie ähnlich euch die folgenden beiden Zeitreihen erscheinen. Worin liegen die Unterschiede, worin die Gemeinsamkeiten? Was haltet Ihr für sinnvolle Möglichkeiten, zwei Zeitreihen aufeinander abzubilden?

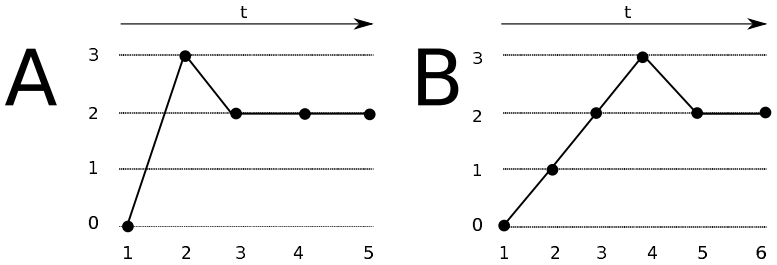

Abbildung 2: Zwei Beispielzeitreihen

Hinweis: Gegeben zwei Zeitreihen $(x_1, \ldots, x_n)$ und $(y_1, \ldots, y_n)$, so nehmen wir an, dass die Distanz zweier Frames x_i, y_j aus diesen Zeitreihen ihrer absoluten Differenz entspricht, also $d_{i,j} := |x_i - y_j|$ (x_i und y_i sind reelle Zahlen).

Wie können wir das Problem formalisieren? Klärt dazu, was ein Matchingpfad ist.

1.3. (Tut) Kumulierte Distanzmatrix berechnen

Um zwei Zeitreihen aufeinander abzubilden, müssen wir den optimalen Matchingpfad berechnen. Dynamische Programmierung gibt uns einen effizienten Algorithmus dafür.

Dazu stellen wir die kumulierte Distanzmatrix D auf. Der Eintrag $D_{i,j}$ bezeichnet die Distanz des optimalen (Teil-)Matchingpfades zwischen $x_1, \ldots, x_i$ und $y_1, \ldots, y_j$, gegeben zwei Zeitreihen $(x_1, \ldots, x_n)$ und $(y_1, \ldots, y_m)$,

Die Intuition ist folgende: Wenn wir die Gesamtdistanz des optimalen Matchingpfades der Teilstrecken $x_1, \ldots, x_{i+1}$ und $y_1, \ldots, y_j$ für $i < n, j < m$ kennen, so muss eine kürzere Teilstrecke dieses Pfades wieder optimal sein – ähnlich wie bei Dijkstra, wo alle Teilpfade eines optimalen Pfades optimal sind.

Wir können das Prinzip der dynamischen Programmierung nutzen, um D aufzustellen, da das Prinzip der optimalen, überlappenden Teillösungen gilt: Die Distanz $D_{i,j}$ ergibt sich aus der geringsten kumulativen Distanz eines um eins kürzeren Matchingpfades zuzüglich der Distanz der Frames x_i und y_j. Formal bedeutet das:

$$D_{i,j} = \begin{cases} 0, & i = 0 \text{ und } j = 0 \\ \infty & \text{entweder } i = 0 \text{ oder } j = 0 \\ \min(D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1}) + d_{i,j} & \text{sonst } (i > 0, j > 0) \end{cases}$$

mit $d_{i,j} := |x_i - y_j|$.

Die ersten zwei Bedingungen wirken sich nur auf Zeile 0 und Spalte 0 der Distanzmatrix aus, können also zur Initialisierung gesetzt werden:

$$D_{0,0} = 0$$
$$D_{1,0} = \ldots = D_{n,0} = \infty$$
$$D_{0,1} = \ldots = D_{0,m} = \infty.$$

Da wir zur Berechnung von $D_{i,j}$ nur Elemente von links ($D_{i,j-1}$, $D_{i-1,j-1}$) oder oben ($D_{i-1,j}$, $D_{i-1,j-1}$) benötigen, kann die Distanzmatrix sukzessive von links oben nach rechts unten aufgebaut werden. Die folgende Illustration zeigt eine mögliche Reihenfolge:

	0	1	2	3
0	0	∞	∞	∞
1	∞	(1.)	(2)	(4.)
2	∞	(3.)	(5.)	(7.)
3	∞	(6.)	(8.)	(9.)

Berechnet die kumulierte Distanzmatrix für die beiden Zeitreihen aus der vorigen Aufgabe!

1.4. (Tut) Gesamtdistanz berechnen

Wie erhalten wir die Gesamtdistanz des besten Matchingpfades aus der kumulierten Distanzmatrix?

1.5. (Tut) Matchingpfad zweier Zeitreihen als kürzester Weg

Der folgende Algorithmus berechnet den Matchingpfad mit kürzester Distanz:

- Starte in der rechten unteren Zeile der kumulierten Distanzmatrix:
  $(i, j) := (n, m)$.
- Solange $(i, j) \neq (0, 0)$:
  - Gehe zum Minimum der oberen, linken und linken oberen Nachbarn:
    (i, j) := $\arg\min_{i,j}(D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1})$.

Indem wir uns die besuchten Zellen merken, erhalten wir alle Paare, die aufeinander abgebildet werden müssen.

Berechnet den Matchingpfad für die Zeitreihen aus den letzten beiden Aufgaben!

Wie können wir die beiden Zeitreihen mit Hilfe des berechneten Matchingpfades aufeinander abbilden?

Hinweis: Beachtet die richtige Reihenfolge des Matchingpfades! Achtet ebenso auf die Indizes, insbesondere ist $(0, 0)$ kein gültiges Paar und muss ignoriert werden (da wir bei 1 beginnen die Zeitreihen zu indizieren).

1.6. Signalvergleich implementieren (70 Punkte)

Implementiert den im Tutorium besprochenen Algorithmus zum Vergleich zweier Signale in der Klasse `Matcher`.

Implementiert die folgende Methoden:

`initializeAccDistanceMatrix()` Erstellt und initialisiert die kumulierte Distanzmatrix, so wie unter 1.3 definiert. D. h. der erste Index bezeichnet die Frames von ISignal x und der zweite die Frames von ISignal y.

`computeAccDistanceMatrix()` Berechnet die akkumulierte Distanzmatrix. Die Parameter sind die zwei Signale, sowie die (initialisierte) Distanzmatrix, welche ihr füllen sollt.

`computeDistance()` Extrahiert die Distanz der beiden Audiodateien aus der kumulierten Distanzmatrix.

`computeMatchingPath()` Findet den kürzesten Pfad durch die kumulierte Distanzmatrix

Mit der Klasse `MatcherTest` könnt ihr eure Implementierung testen. Testsignale könnt ihr entweder aus .wav Dateien laden, oder mittels eines Array definieren.

`MatcherTest` enthält auch eine Methoden zur Ausgabe kleiner Matrizen, und eine Methode zum Speichern von Signalen im .wav Format. [1]

Hinweis: Da wir relativ große Arrays verwenden (mit ca. 1-2GB Speicherbedarf), ist es möglich dass Java auf eurem Rechner an die Grenzen seines Heap-Speichers kommt. Ihr bekommt dann eine HeapError Exception. Falls euer Rechner genug Hauptspeicher zur Verfügung hat, könnt ihr die erlaubte Maximalgröße des Heaps manuell erhöhen:

1. Geht auf den kleinen schwarzen Pfeil neben dem grünen Run-Button in der oberen Leiste
2. Klickt auf Run configurations...
3. Wählt links unter Java Application die Anwendung, die Ihr ausführen wollt, z. B. WavMatchExample
4. Klickt auf den Registerreiter Arguments
5. Im Feld VM Arguments fügt Ihr folgenden Befehl ein: `-Xmx3072m` (dies bedeutet, Java hat bis zu 3GB Heap Memory – ihr könnt diese Zahl auch erhöhen, wenn Ihr mehr RAM zur Verfügung habt)

[1] Zum einfachen Einlesen und Schreiben von WAV-Audiodateien verwenden wir in der Vorgabe die Java Wav File IO mit freundlicher Genehmigung von Andrew Greensted.
(http://www.labbookpages.co.uk/audio/javaWavFiles.html).

1.7. Datenbankanwendung implementieren (30 Punkte)

Zusätzlich haben wir eine kleine Anwendung vorbereitet: ein primitives Spracherkennungssystem. Die Hauptkomponente der Anwendung ist eine Datenbank mit annotierten Audiodateien.

Jede in `./data/database_small.txt` gelistete Audiodatei stellt ein Referenzsignal für ein bestimmtes Wort dar, wobei für das selbe Wort auch mehrere Referenz-Signale gelistet werden können:

```
da data/da2.wav
dort data/dort2.wav
fast data/fast2.wav
hallo data/hallo2.wav
```

Die erste Zeichenkette (ohne Leerzeichen!) ist die Annotation, die zweite Zeichenkette der Pfad der Datei, relativ zum Vorgabenordner.

Vervollständigt die Datenbankapplikation, so dass sie für ein Signal den ähnlichsten Eintrag in der Datenbank zurückgibt. Implementiert die folgenden Methoden:

`lookup()` Implementiert die Methode `Database.lookup(ISignal query)`, welches ein Signal als Anfrage (engl. "'query"') übergeben bekommt und den Namen des wahrscheinlichsten Wortes zurückgibt, und das Matcher-Objekt, dass nähere Details enthält (z.B. welches Referenzsignal, berechnete Distanz).

Für die Berechnung des wahrscheinlichsten Wortes verwenden wir den so genannten Nearest-Neighbour-Algorithmus: Wir suchen in allen Referenzsignalen jenes mit der geringsten Distanz zum gegebenen Signal (den nächsten Nachbarn).

Hinweis: Verwendet zum Vergleich zweier Signale in `lookup()` ausschließlich die Methode `compare()`!

In der Klasse `DatabaseTest` findet ihr ein paar Beispiele für erfolgreiche Suchen. Die Klasse `SpeechRecognition` implementiert ausserdem eine Kommandozeilenapplikation um einzelne Wörter zu erkennen.

Hinweis: Ihr könnt auch eigene Soundfiles zur Datenbank hinzufügen und ausprobieren. Beachtet aber, dass die Soundfiles im WAV-Format sein sollten und eine niedrige Abtastrate aufweisen sollten (die mitgelieferten Beispiele haben eine sample rate von 8 kHz). Ansonsten übersteigt der Speicherbedarf der Distanzmatrix die Möglichkeiten der Memory Allocation!

Hinweis: Manchmal ist die von uns berechnete Distanz kein guter Indikator für die tatsächliche Ähnlichkeit der Audiodatien. Wenn Ihr zum Beispiel die Datei `hallo1.wav` mit allen anderen mitgelieferten Dateien vergleicht, werdet Ihr feststellen, dass die Distanz zu `da1.wav` geringer ist als zu `hallo2.wav`.

Woran könnte das liegen?