Übungsaufgaben II, SBV1

Lisa Panholzer, Lukas Fiel

November 18, 2018

1 Übungsaufgaben II

1.1 Resampling und Interpolation

a) Implementierung Resampling

b) Implementierung Bi-Lineare Interpolation

c) Implementierung Checker-Board

1.2 Klassifizierung mittels Kompression

a) Klassifizierung von Texten

Idee Aus Texten in 8 verschiedenen Sprachen soll mittels Kompression eine Klassifizierung stattfinden. Dazu wurden folgende Datensätze vorbereitet:

- Abstract einer wissenschaftlichen Arbeit. Diese hatte den Vorteil dass es eine deutsche und englische Übersetzung gab. Alle weiteren Sprachen wurden aus der englischen Version mittels *google translate* generiert.
- Wörterbuch mit 10000 deutschen Wörtern. Dieser Datensatz wurde mittels *google translate* in alle anderen Sprachen übersetzt.
- Die erste Seite der Datenschutzrichtlinien von Facebook. Da die Datenschutzrichtlinien in sämtlichen Sprachen abruf bar sind, konnte für alle Sprachen ein passender Datensatz gefunden werden.
- Die erste Seite der Datenschutzrichtlinien von Google. Auch hier waren Daten in allen Sprachen verfügbar.
- Ein Witz der aus dem deutschen in alle andern Sprachen übersetzt wurde.

Da nach einer Übersetzung die Texte in verschiedenen Sprachen ungleich viele Buchstaben beinhalten ist auch die Dateigröße unterschiedlich. Dies könnte eventuell rechnerisch berücksichtigt werden. Viel einfacher aber ist es, die letzten Buchstaben jedes langen Textes zu ignorieren und so eine einheitliche Länge des Textes zu gewährleisten. Dies wurde mittels eines shell-Skripts erreicht, welches die ersten n Bytes eines Files speichert. So konnte für jeden Text eine Datei erzeugt werden die in allen Sprachen den

selben Speicherbedarf hat. Da es um Klassifizierung geht ist der Verlust der letzten Buchstaben bzw Wörter nicht wesentlich.

; columns

```
#!/bin/bash


# mkdir cutTestData
mkdir cutTestData/witzData/

for filename in TestData/witzData/*.txt; do
    dd bs=1 count=8200 if="$filename" of="cut$filename"

done
```

Nach einer solchen Normierung der Texte können diese miteinander verglichen werden. Dazu wurde ein Programm in *Octave* geschrieben, welches die Texte der einzelnen Datensätze miteinander vergleicht und in einer Matrix darstellt.

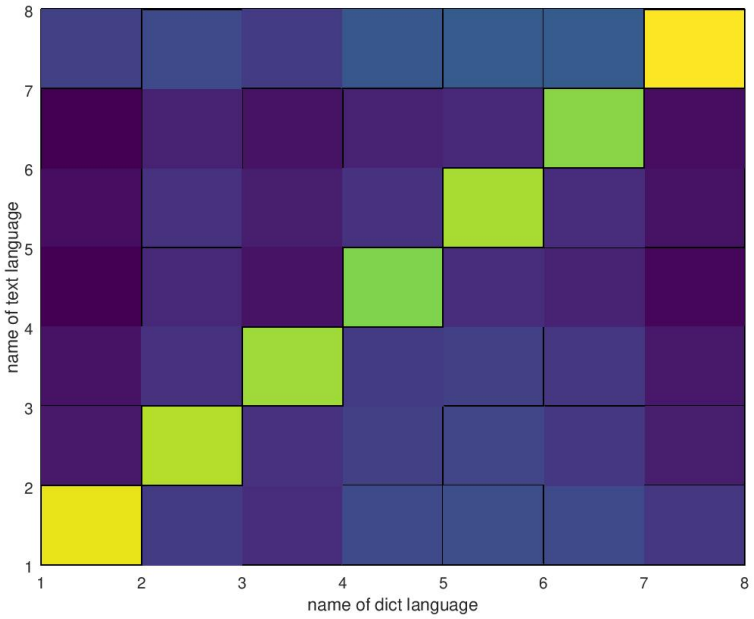


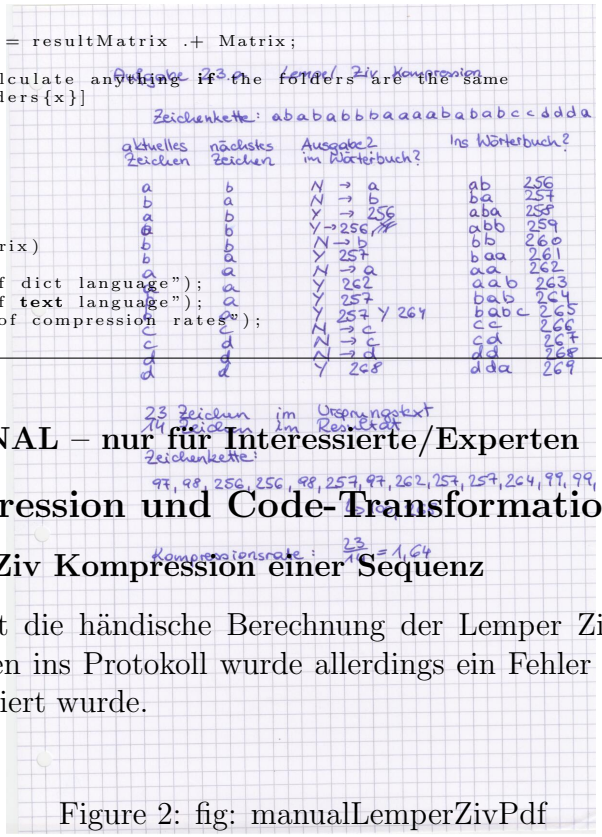
Figure 1: zipDataMatrix

; columns

```
# clear console, clear variables, close all open figures
clc
clear all
close all
```

```
# in this example we will use following languages
# languages = {"de","en","fr","es","po","un","bo","ne"};

# constants
# please note:copy-pasting windows paths: there are no backslashes in the path
folderPath = 'cutTestData/*';

# initialize result matrix
resultMatrix =zeros(8);


# loop through data folders
folders = glob(folderPath)
for x=1:numel(folders)
  [~, nameI] = fileparts (folders{x})
  for y=1:numel(folders)
    [~, nameI] = fileparts (folders{y})

    # just calculate matrix for different data sets
    if (!strcmp(folders{x},folders{y}))

      folderPath1 = [folders{x} , '/*'];
      folderPath2 = [folders{y} , '/*'];
      filesOfFolder1 = glob(folderPath1);
      filesOfFolder2 = glob(folderPath2);

      # for every element in data folder
      for i=1:numel(filesOfFolder1)
        [~, nameI] = fileparts (filesOfFolder1{i});
        #get file size
        [info, err, msg] = stat (filesOfFolder1{i});
        file1Size = info.size;

        #CREATE zip of file
        tmpFolderPath = nameI;
        mkdir(tmpFolderPath);
        copyfile(filesOfFolder1{i},tmpFolderPath);
        firstZipName = [nameI, '.zip']
        zip(firstZipName,[tmpFolderPath,'/*']);
        #get zip size
        [info, err, msg] = stat (firstZipName);
        file1file2ZipSize = info.size;

        # calculate compression rate
        file1CompressionRate = file1file2ZipSize / file1Size;

        #delete zip and folder as we just need the size for calculation
        delete([tmpFolderPath,'/',nameI,'.txt']);
        rmdir(tmpFolderPath);
        delete(firstZipName);

        for j=1:numel(filesOfFolder2)
          [~, nameJ] = fileparts (filesOfFolder2{j});
          zipName = [nameI,nameJ,'.zip'];
          #create tmp folder in testData folder
          tmpFolderPath = [nameI,nameJ];
          mkdir(tmpFolderPath);

          #copy filesOfFolder1 to folder
          ['copy␣filesOfFolder1␣', nameI, '␣,␣' nameJ , '␣to␣' , tmpFolderPath];
          copyfile(filesOfFolder1{i},tmpFolderPath);
          copyfile(filesOfFolder2{j},tmpFolderPath);

          # get folder size (add jokeFile size to filesOfFolder1ize)
          [info, err, msg] = stat (filesOfFolder2{i});
          file2Size = file1Size + info.size;

          #zip it and get size of zip
          zip(zipName,[tmpFolderPath,'/*']);
          [info, err, msg] = stat (zipName);
          file2ZipSize = info.size;

          #calculate compression rate
          compressionRate = file2ZipSize / file2Size ;

          #calculate exprected rate
          expectedfile2ZipSize = file2Size * file1CompressionRate;
```

```
          %Matrix(i,j) = (expectedfile2ZipSize - file2ZipSize) /  file2ZipSize;
          kompressionDict = (file1Size -file1file2ZipSize) / file1Size;
          kompressionBoth = (file2Size - file2ZipSize) / file2Size;
          kompressionsDelta = abs(kompressionBoth - kompressionDict);
          Matrix(i,j) = kompressionsDelta;


          #cleanup - remove tmp folder
          ['remove_' tmpFolderPath];
          delete([tmpFolderPath,'/',nameI,'.txt']);
          delete([tmpFolderPath,'/',nameJ,'.txt']);
          rmdir(tmpFolderPath);
          delete(zipName);
        endfor
      endfor

      resultMatrix = resultMatrix .+ Matrix;

    else # dont calculate anything if the folders are the same
      ["skip " folders{x}]
    endif


  endfor
endfor

    figure()
    surf(resultMatrix)
    view(2)
    xlabel("name of dict language");
    ylabel("name of text language");
    zlabel(" diff of compression rates");
    'SUCCESS'
```



b) OPTIONAL – nur für Interessierte/Experten

1.3 Kompression und Code-Transformation

a) Lempel-Ziv Kompression einer Sequenz

Figure a) zeigt die händische Berechnung der Lemper Ziv Kompression. Beim Übertragen ins Protokoll wurde allerdings ein Fehler entdeckt, der in Tabelle 1 korrigiert wurde.

Figure 2: fig: manualLemperZivPdf

aktuelles Zeichen	nächstes Zeichen	Ausgabe (im Wörterbuch?)	ins Wörterbuch!	Speicher
a	b	N → a	ab	256
b	a	N → b	ba	257
a	b	Y → 256	aba	258
a	b	Y → 256	abb	259
b	b	N → b	bb	260
b	a	Y → 257	baa	261
a	a	N → a	aa	262
a	a	Y → 256	aab	263
b	a	Y → 257	bab	264
b	a	Y (257), Y → 264	babc	265
c	c	N → c	cc	266
c	d	N → c	cd	267
d	d	N → d	dd	268
d	d	Y → 268	dda	269
a		N → a		

Table 1: Level Ziv Kompression

In der korrigierten Version ergibt die resultierende Zeichenkette:

97	98	256	256	98	257	97	262	257	264	99	99	100	268	269

$$[H]KompressionsrateC = \frac{23}{15} = 1.5334 \tag{1}$$

b) Transformation einer Sequenz

c) Komprimierung einer Sequenz mittels Runlength Coding

Berechnung der Kompressionsrate

Die nachfolgende Sequenz soll anhand von Runlength Coding händisch komprimiert und die Kompressionsrate ausgegeben werden:

010101111100000111100010101111 (30 Stellen, n=2 Symbole:0,1)

Die Sequenz wird von links nach rechts codiert, und anstatt der eigentlichen Zeichen die Häufigkeit dieser ausgegeben. Nach der händischen Komprimierung weist die Frequenz folgende Lauflänge auf:

11111554311114 (14 Stellen)

Daraus ergibt sich folgende Kompressionsrate:
30/14 = **2,14**

Erweiterung der Symbolmenge
Die Komprimierung von Sequenzen anhand der RLC ist am effektivsten, je homogener der Informationsgehalt ist. Das bedeutet, das bei einer Steigerung der unterschiedlichen Zeichen die Komprimierungsmethode in manchen Fällen nicht mehr sinnvoll angewandt werden kann. Bei sehr kurzen Sequenzen kann es sogar zu einer Erhöhung der Zeichenanzahl in dieser kommen.

Wird nun die Anzahl der Symbole erhöht, muss beachtet werden, dass zusätzlich zu der Häufigkeit des Zeichens noch ein Trennsymbol, eine ID bzw. das Zeichen selbst mitgegeben werden muss. Dies wird anhand eines selbst gewählten Beispiels demonstriert.

Folgende Sequenz wird anhand von RLC komprimiert:
AABBBBBBBBCCCCCCCDEEEEEEEFFGHIIJJJKLMNNN
(40 Stellen, n=13 Symbole:A,B,C,D,E,F,G,H,I,J,K,L,M,N)

Die komprimierte Sequenz lautet:
A2B8C7D1E7F2H1I2J3K1L1M1N3 (26 Stellen)

Daraus ergibt sich folgende Kompressionsrate:
40/26=**1,54**

Wird eine sehr kurze Sequenz mit einer hohen Anzahl an Zeichen wiederholt, kann es aufgrund einer niedrigen Homogenität des Informationsgehalts dazu kommen, dass die codierte Sequenz länger ist, als die Originale.

Folgende Sequenz wird anhand von RLC komprimiert:
ABBCDEEFFFGHIIJJJKLLLMNOPQRSTU
(30 Stellen, n=21 Symbole:A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U)

Die komprimierte Sequenz lautet:
A1B2C1D1E2F3G1H1I2J3K1L3M1N1O1P1Q1R1S1T1U1 (42 Stellen)

Daraus ergibt sich folgende Kompressionsrate:
30/42=**0,71**

d) Entropieberechnung