Erkennung einer Emotion anhand von Sprache

Projektarbeit

für die Prüfung zum

Bachelor of Science

des Studiengangs Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Mesut Kuscu, Helge Brügner

Januar 2016

Bearbeitungszeitraum	29.09.15 - 07.01.2016
Matrikelnummer, Kurs	7429032, 3256960, TINF13A
Ausbildungsfirma	Hewlett Packard Enterprise GmbH, Böblingen
Betreuer	-
Gutachter	Prof. Dr. Dirk Reichardt

Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Projektarbeit mit dem Thema *Erkennung einer Emotion anhand von Sprache* ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Projektarbeit bei keiner anderen Prüfung vorgelegt habe;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Stuttgart, Januar 2016

Mesut Kuscu, Helge Brügner

Inhaltsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Voraussetzungen für die Anwendung der Evidenztheorie ist gegegeben, da folgendes gilt:

1. Jegliche Alternativmengen sind gegeben. Das bedeutet, dass davon ausgegangen wird, dass eine der Grundemotionen stets auf den Benutzer zu trifft und alle Grundemotionen vom Programm erkannt werden können.
2. Jegliche Alternativen schließen sich gegenseitig aus: Falls eine Grundemotion *erkannt* wurde, schließt diese andere aus; d.h. eine Tonaufnahme eines Benutzers beinhaltet immer nur eine der Grundemotionen.

Der Aufbau dieser Arbeit umfasst vorerst das Programmdesign. Hierbei werden die Ausprägungen der Emotionen konkretisiert, fehlende Werte für Konfidenzen hinzugefügt und mögliche Emotionsklassen mit den zugehörigen Merkmalen tabellarisiert. Daraufhin folgt die Dokumentation der Implementation und eine Beschreibung.

2 Abbildung der Evidenzen

Dieses Kapitel befasst sich mit der Zuordnung jeder Emotion und deren Attribute. Vorerst wird diese Zuordnung der Aufgabenstellung gemäß dargestellt. Daraufhin werden die Werte der Beispieldaten näher betrachtet und die zuvor vorgestellte Zuordnung der Emotionen konkretisiert.

2.1 Emotionen

Die Tabelle 2.1 stellt die Einstufung der Emotionen aus der Aufgabenstellung dar. Hierbei fällt auf, dass die Schallstärke nicht ausschlaggebend für die Emotion *Angst* ist und die Sprechgeschwindigkeit keinen Einfluss auf *Traurigkeit* hat. Außerdem fällt auf, dass keine Mittelstufe in den Ausprägungen der Merkmale existiert.Diese Tabelle wird im weiteren Verlauf konkretisiert und an die vorgegebenen Beispieldaten angepasst.

Emotion	Kürzel	Sprechgeschwindigkeit	Tonlage	Schallstärke
Angst	A	Schnell	Hoch	-
Überraschung	U	Schnell	Hoch	Hoch
Wut	W	Schnell	Hoch	Hoch
Freude	F	Schnell / Langsam	Hoch	Hoch
Ekel	E	Langsam	Tief	Schwach
Traurigkeit	T	-	Tief	Schwach

Tabelle 2.1: Abbildung der Emotionen nach Aufgabenstellung

2.2 Nutzung der Beispieldaten

Das Projekt benutzt die Beispielwerte aus den Dateien `E_004.csv` und `E_004b.csv`. In beiden Dateien sind jeweils 50 Takte angegeben, die zu analysieren sind.

In jedem Takt entstehen jeweils drei Evidenzen. In der Aufgabenstellung geht zwar hervor, dass jeweils immer nur 3 Levels (wenig, normal, viel) von Ausprägungen der jeweiligen Evidenz erkannt werden muss (so wie oben in der Tabelle gekennzeichnet), jedoch sind in

den Beispieldaten in der Tonlage und in der Schallstärke weitere Ausprägungen zu finden (sehr wenig, sehr viel). Die folgende Liste enthält alle Ausprägungen der Merkmale, die in den Beispieldaten zu finden sind.

Sprechgeschwindigkeit in Silben/Sekunden Werte von 2-7, keine Einteilung vorgegeben

Durchnschnittliche Tonlage sehr niedrig, niedrig, normal, hoch, sehr hoch

Schallstärke sehr niedrig, niedrig, normal, hoch, sehr hoch

Da die Anzahl der Ausprägungen in den Beispieldaten größer ist als in der Aufgabenstellung, befasst sich der verbleibende Teil dieses Kapitels mit der Nutzung und Einteilung der Beispieldaten.

2.3 Normierung der Ausprägungen der Merkmale

Jegliche Normierungen sind über eine externe Einstellungsdatei statisch festgelegt. Diese können, unabhängig vom Programmcode, vor Programmstart angepasst werden. Der Programmaufbau wird im nächsten Kapitel näher betrachtet.

2.3.1 Sprechgeschwindigkeit

Nach einer Analyse der Werte der Sprechgeschwindigkeit ist festgestellt worden, dass die Werte in beiden Dateien zwischen 2 und 7 variieren. Die folgende Tabelle 2.2 stellt die Analyse dar.

Dateiname	Minimum	Maximum	Mittelwert
`E_004.csv`	2.6	5.9	4
`E_004b.csv`	2.9	6.5	4.442

Tabelle 2.2: Ausprägung der Sprechgeschwindigkeit

Dadurch wurde folgende Norm für die Ausprägung der Sprechgeschwindigkeit aufgestellt, welche für beide Beispieldaten gilt:

langsam Der Wert ist kleiner als 2.5

normal Der Wert liegt im Intervall: 2.6 - 5.5

schnell Der Wert ist größer als 5.6

Die ausgewählten Werte orientieren sich von den Durchschnittswerten aus der Analyse der Beispieldatei. Die Analysefunktion (`printAverageSpeed()`) findet sich auch im Programmcode um spätere Analysen für andere Daten zu gewährleisten.

2.3.2 Durchschnittliche Tonlage und Schallstärke

Die Ausprägung für Schallstärke und durchschnittliche Tonlage wird wie folgt zusammengefasst:

niedrig falls sehr niedrig oder niedrig

normal falls normal

hoch falls hoch oder sehr hoch

Durch diese Normierung werden die Beispieldaten an die Aufgabenstellung angepasst: Es sind nur noch drei Ausprägungen zu analysieren.

2.4 Konkretisierung der Emotionen

Da nun weitere Ausprägungen durch die Beispieldaten dazugekommen sind, ist es notwendig, die Zuordnung der Emotionen zu konkretisieren. Die folgende Tabelle 2.3 stellt die finale Zuordnung der Emotionen dar. Unterschiede zu der vorherigen Tabelle 2.1 sind mit fetter Schrift makiert. In der Tabelle ist zu erkennen, dass die Ausprägung *Normal* hinzugefügt worden ist. Diese Ausprägung steht in der Meinung des Entwicklerteams im starken Zusammenhang mit der Emotion *Freude*.

Emotion	Sprechgeschwindigkeit	Tonlage	Schallstärke
Angst A	Schnell	Hoch	-
Überraschung U	Schnell	Hoch	Hoch
Wut W	**Normal**/Schnell	Hoch	Hoch
Freude F	Langsam/**Normal**/Schnell	**Normal**/Hoch	**Normal**/Hoch
Ekel E	Langsam	Tief	Schwach/**Normal**
Traurigkeit T	-	Tief	Schwach

Tabelle 2.3: Konkretisierung der Emotionen

2.5 Konfidenz der Messungen

Da keine Zuverlässigkeit der Messung in der Aufgabenstellung angegeben ist, müssen diese vom Entwicklerteam fiktiv geschätzt werden. Diese Werte können statisch vor Programmstart leicht in der Einstellungsdatei verändert werden. Die folgende Liste stellt die ausgesuchten Werte der Konfidenz der jeweiligen Messungen dar:

Sprechgeschwindigkeit liegt bei 0.6

Tonlage liegt bei 0.8

Schallstärke liegt bei 0.7

2.6 Ergebnisauswertung

Durch Akkumulation der Evidenzen führt das Programm zu einer Schlussfolgerung. Es wird die Emotion ausgewählt, dessen Plausabilität am höchsten ist. Da beide Dateien jeweils 50 Takte beinhalten, sollten im Idealfall insgesamt die 50 Emotionen ausgegeben werden, bei denen im jeweiligen Takt die Plausibilität am höchsten ist.

3 Implementierung

Das Programm ist komplett in C++ geschrieben und erfüllt die Anforderungen der Aufgabenstellung. Mit dem Programm ist es möglich, die gegebenen Beispieldaten einzulesen und mit dem Dempster Algorithmus eine Emotion (dessen Plausibilität am höchsten ist) für jeden Takt zu bestimmen. Die Ergebnisausgabe wird in der Konsole angezeigt und anschließend wird automatisch eine .CSV Datei geschrieben, in der die Ergebnisausgabe tabellarisiert ist.

Das Programm kann in vier Teile eingeteilt werden.

1. Einlesen der Beispieldaten aus den CSV-Dateien,
2. Erstellen der Evidenzen für Sprechgeschwindigkeit (m_1), Tonlage (m_2) und Schallstärke (m_3),
3. Verarbeiten der Evidenzen (Bilden von m_{123} und Berechnung der Plausibilität) und
4. Ergebnisausgabe (Schreiben in eine CSV-Datei).

Dabei ist jegliches, relevanter Codesegment im Quellcode kommentiert.

Dieses Kapitel befasst sich vorerst mit der Dateistruktur der Implementation. Daraufhin wird jede Datei einzeln betrachtet und anschließend noch die Datenstruktur der Evidenzen erläutert.

3.1 Dateistruktur

Das Programm ist in drei Dateien aufgeteilt:

Main.cpp Diese Datei verlinkt die anderen beiden Dateien und dient als Startpunkt.

settings.h In dieser Datei finden sich jegliche voreingestellten Makros, globale Initialisierungen, Konstanten und Variablen für das Programm. Die Headerdatei wird in Main.cpp und dempster.h eingebunden.

dempster.h Diese Datei beinhaltet die Dempsterfunktion, welche von der Main.cpp aufgerufen wird.

Der verbleibende Teil dieser Dokumentation fokussiert sich nacheinander auf jede, genannte einzelne Datei und bespricht die implementierten Funktionen.

3.2 settings.h

Die Voreinstellungen des Programmes können in drei Teile eingeteilt werden.

Der erste Teil beinhaltet die Pfade zu den Beispiel-Eingabedaten (`E_004.csv` und `E_004b.csv`) und den jeweiligen Ausgabedaten. Zusätzlich kann noch ein Index angegeben werden, an welchem die Daten in den Dateien starten (um beispielsweise den Header zu überspringen).

Im zweiten Teil können die Konfidenzwerte (die Gewichtung der Evidenzen) festgelegt werden, welche in Kapitel 2.5 besprochen wurden.

Im dritten Teil sind jegliche Werte der Ausprägungen festgelegt. Beispielsweise kann hier festgelegt werden, welcher Wert für den unteren Teil der Sprechgeschwindigkeit ausschlaggebend ist (Kapitel 2.3.1) und wie die Ausprägungen der Merkmale in den vorgegeben Daten gespeichert sind.

3.3 dempster.h

In dieser Datei sind alle unmittelbar mit der Evidenzberechnung zusammenhängenden Funktionen zu finden. Die *Dempster*-Funktion erwartet alle Evidenzen (m_1, m_2 und m_3) mit Gewichtung und akkumuliert sie zur finalen Evidenz $m_(123)$. Des Weiteren beinhaltet sie eine Funktion zur Berechnung der *Plausibilitäten*.

Zur Berechnung der Evidenzen wird das folgende Strukt benutzt:

```
1 struct Evidence {
2   Evidence() {}
3   set<string> emotions;
4   double value = 0.0;
5 };
```

Listing 3.1: Strukt zur Evidenzberechnung

Entscheidend sind hier das Set und der double-Wert. *Evidence.emotions* beinhaltet alle zu dieser Evidenz gehörenden Emotionen (beispielsweise *Wut* und *Freude*) und kann auch leer sein, und *Evidence.value* speichert die dazugehörige Konfidenz. Das entsprechende Omega wird mit dem gleichen Strukt gespeichert; *emotions* beinhaltet hierbei alle Emotionen und *value* ist $1 - (KonfidenzderEvidenz)$. Die Evidenz und das dazugehörige Omega werden in einem Vektor mit mindestens zwei Komponenten gespeichert (oder mehr, falls Resultat einer Akkumulation). Dies ist in Diagramm 3.1 erkennbar.

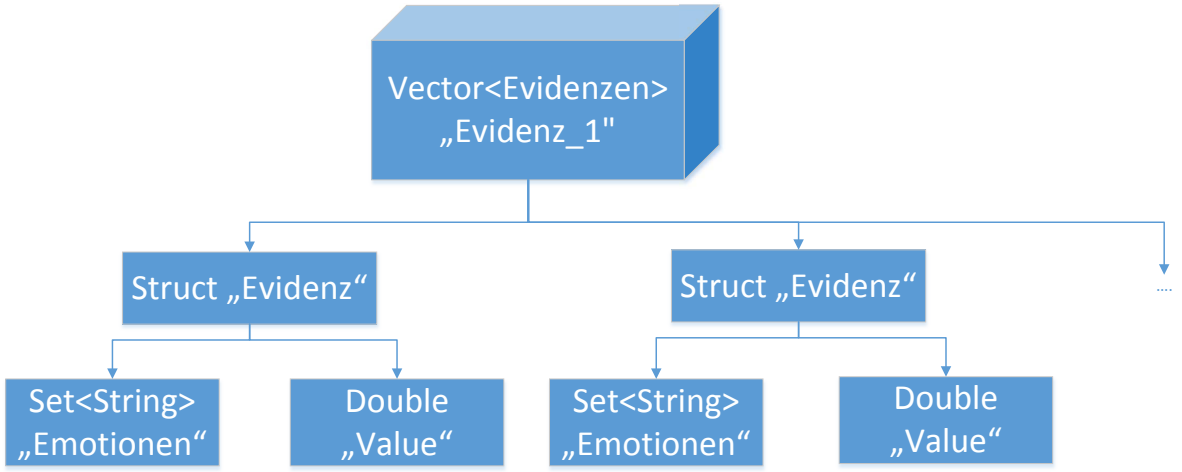

Abbildung 3.1: Struktur der Evidenzen

In der *Dempster*-Funktion wird zuerst aus m_1 und m_2 die Evidenz $m_(12)$ durch Vereinigung der Sets und Multiplikation der Konfidenzen gebildet, und es entstehen vier mögliche Kombinationen. Anschließend wird die Kombination aus Evidenz von m_1 und m_2 auf einen Konflikt untersucht; sollte ein Konflikt auftreten, wird k gebildet, die entsprechende leere Menge aus m_{12} entfernt und die übrigen Konfidenzen mit $\frac{1}{1-k}$ korrigiert. Durch Akkumulation von m_{12} und m_3 entsteht auf gleichem Wege m_{123}, und die resultierenden Mengen werden auf Konflikte untersucht, diese gegebenenfalls gelöscht und der Rest korrigiert. Die Funktion wird einmal pro Takt aufgerufen.

Die Plausibilitätsberechnung geschieht in der Funktion `plausibility()`. Als Eingabeparameter wird hier die Evidenz m_{123} erwartet. Durch Iterieren über alle Teilevidenzen werden für jede Emotion die Teilwerte aufsummiert und in einen neuen Vektor gespeichert. Anschließend wird die größte Plausibilität gesucht und zusätzlich abgespeichert.

3.4 Main.cpp

Die Main.cpp Datei beinhaltet die *Main-Funktion*, welche bei Programmstart aufgerufen wird. Zusätzlich zu der Main-Funktion sind hier weitere Funktionen zu finden. Der folgende Ausschnitt zeigt jegliche Funktionssignaturen der Main.cpp befindet sich im Abschnitt *Predefined Functions* dieser Datei:

```
1 double toDouble(string s);
2 void printFile(string filename);
3 vector<string> readFileInVector(string filename);
4 void writeResultsToFile(string filename,
5   vector<vector<Evidence>> evidences,
6   vector<map<string, double>> plausibilities);
7 void printAverageSpeed(vector<string> data);
```

```
8  set<string> getEmotionOfSpeed(double);
9  set<string> getEmotionOfPitch(string);
10 set<string> getEmotionOfIntensity(string);
11 void calculatePlausibilities(string input, string output);
```

Listing 3.2: Predefined classes/functions aus der Main.cpp

Die Funktionen `printFile()`, `readFileInVector` und `writeResultsToFile()` sind für Dateieingabe und -ausgabe zuständig. Mit `printAverageSpeed()` wurde die durchschnittliche Sprechgeschwindigkeit bestimmt. `getEmotionOfSpeed()`, `getEmotionOfPitch()` und `getEmotionOfIntensity()` liefern die Emotionen anhand der Eingabewerte entsprechend der Tabelle 2.3. In der Funktion `calculatePlausibilities()` befindet sich das Hauptprogramm bestehend aus Aufruf der Einlesefunktionen, einer Schleife, die für jeden Takt die *Dempster*-Funktion aufruft und dem Aufruf der Ausgabefunktion.

4 Starten der Applikation

Bei der Ausführung der Applikation ist zu beachten, dass die Eingabedateien sich mit dem korrekten Namen im korrekten Ordner befinden. Laut den Standardeinstellungen in der `settings.h`-Datei sollten sich die beiden Dateien im selben Ordner wie das Programm befinden und `E_004.csv` bzw. `E_004b.csv` genannt sein. Außerdem muss das Programm Schreibrechte auf die Ausgabedatei haben; laut Standardeinstellungen sind dies `resulta.csv` und `resultb.csv` im selben Ordner wie das Programm.

Es gibt zwei Möglichkeiten, um das Programm zu starten. Zum Einen kann die schon kompilierte .EXE-Datei auf einem Windows-PC ausgeführt werden, und zum Anderen kann auf Basis des Quellcodes das Programm noch einmal neu kompiliert werden. Wenn `Main.cpp` kompiliert wird, ist darauf zu achten, dass `settings.h` und `dempster.h` verlinkt werden. Alternativ zu einer Kompilierung "von Hand"kann mit Microsoft Visual Studio die Solution (befindlich im Unterordner SSolution") geöffnet werden.

Anhang

In diesem Teil befindet sich noch der Code des Programmes. Dieser befindet sich auch in den zugehörigen Dateien in dem Ordner *Code*.

Main.cpp

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <set>
#include <map>

#include "settings.h"
#include "dempster.h"

using namespace std;

double toDouble(string s) {
  replace(s.begin(), s.end(), ',', '.');
  return atof(s.c_str());
}

void printFile(string filename) {
  ifstream file(filename);
  string value;

  while(file.good()) {
    getline(file, value, ';'); // read a string until next comma: http
        ://www.cplusplus.com/reference/string/getline/
    //cout << string(value, 1, value.length() - 2); // display value
        removing the first and the last character from it
    cout << string(value) << "-";
  }

  cout << endl;
}
```

```
33
34 vector<string> readFileInVector(string filename) {
35   vector<string> read;
36   ifstream file(filename);
37   string value;
38   int counter = 0; // Every 4th element has a '\n' afterwards.
39             // This needs to be removed from the element in the vector.
40
41   while(file.good()) {
42     counter++;
43
44     if((counter % 4) == 0) {
45       getline(file, value, CSV_ENDL); // read a string until next comma
46     }
47     else {
48       getline(file, value, CSV_SEP); // read a string until next comma
49     }
50
51     read.push_back(value);
52   }
53
54   if(read.empty()) {
55     cerr << " Error: Vector empty. Probably unreadable file";
56   }
57
58   read.pop_back(); //empty element at the end gave me headache
59
60   file.close();
61
62   return read;
63 }
64
65 void writeResultsToFile(string input, string filename, vector<vector<
       Evidence>> evidences, vector<map<string, double>> plausibilities) {
66   if(evidences.size() != plausibilities.size()) {
67     cout << " Error: Something went wrong while calculating evidences
           and plausibilities!" << endl;
68
69     return;
70   }
71
72   ofstream file(filename);
73
74   file << HEAD_CLK << CSV_SEP
75     << FEAR << CSV_SEP
76     << SURPRISE << CSV_SEP
77     << ANGER << CSV_SEP
```

```
78      << JOY << CSV_SEP
79      << DISGUST << CSV_SEP
80      << SORROW << CSV_SEP
81      << HEAD_MAX << CSV_SEP
82      << HEAD_MAXVAL << CSV_ENDL;
83
84    double crt;
85    double max;
86    string e;
87
88    for(size_t i = 0; i < evidences.size() && file.good(); i++) {
89      crt = 0.0;
90      max = 0.0;
91
92      file << (i + 1) << CSV_SEP;
93
94      for(const auto& emotion : OMEGA) {
95        file << (crt = plausibilities[i][emotion]) << CSV_SEP;
96
97        if(crt > max) {
98          e = emotion;
99          max = crt;
100       }
101     }
102
103     file << e << CSV_SEP << max << CSV_ENDL;
104   }
105
106   file.close();
107   std::cout << "Saved results of data " << input << " in file: " <<
          filename << "." << endl;
108 }
109
110 void printAverageSpeed(vector<string> data) {
111   int elem = 0;
112   int amountOfElements = 0;
113   double average = 0.0, min = 0.0, max = 0.0, temp = 0.0;
114   bool init = false;
115
116   for(int i = 4; i < data.size(); i += 4) { //empty
117     elem = i + 1; // Get specific element of Sprechgeschwindigkeit
118     temp = toDouble(data[elem]); // specific element of
          Sprechgeschwindigkeit
119     average += temp;
120     amountOfElements++;
121
122     //min max calculation
```

```
123     if(init == false) { //this just once
124       init = true;
125       min = temp;
126       max = min;
127     }
128
129     if(min > temp) {
130       min = temp;
131     }
132     else if(max < temp) {
133       max = temp;
134     }
135   }
136
137   average = average / amountOfElements;
138
139   cout << "The average Speed is: " << average << endl;
140   cout << "The max value is: " << max << endl;
141   cout << "The min value is: " << min << endl;
142 }
143
144 set<string> getEmotionOfSpeed(double value) {
145   set<string> temp;
146
147   if(value <= SLOW) {
148     temp.insert(DISGUST); // Ekel
149     temp.insert(JOY); // Freude
150   }
151   else if(value >= FAST) {
152     temp.insert(FEAR); // Angst
153     temp.insert(SURPRISE); // Ueberraschung
154     temp.insert(ANGER); // Wut
155     temp.insert(JOY); // Freude
156   }
157   else { //Speed is normal - so it is between slow and fast (2.5 < x <
          6.6)
158     temp.insert(ANGER); // Wut
159     temp.insert(JOY); // Freude
160   }
161
162   return temp;
163 }
164
165 set<string> getEmotionOfPitch(string value) {
166   set<string> temp;
167
168   //string.compare returns 0 if equal
```

```
169   if((value.compare(LOW1) == 0) || (value.compare(LOW2) == 0)) {
170     temp.insert(DISGUST); // Ekel
171     temp.insert(SORROW); // Traurigkeit
172   }
173   else if(value.compare(MEDIUM) == 0) {
174     temp.insert(JOY); // Freude
175   }
176   else if((value.compare(HIGH1) == 0) || (value.compare(HIGH2) == 0)) {
177     temp.insert(FEAR); // Angst
178     temp.insert(SURPRISE); // Ueberraschung
179     temp.insert(ANGER); // Wut
180     temp.insert(JOY); // Freude
181   }
182   else {
183     cerr << " Error: Could not read the data. Tried to read the
          following: "
184       << value << " . Are you sure that you set all macros right?" <<
           endl;
185   }
186
187   return temp;
188 }
189
190 set<string> getEmotionOfIntensity(string value) {
191   set<string> temp;
192
193   //string.compare returns 0 if equal
194   if((value.compare(LOW1) == 0) || (value.compare(LOW2) == 0)) {
195     temp.insert(DISGUST); // Ekel
196     temp.insert(SORROW); // Traurigkeit
197   }
198   else if(value.compare(MEDIUM) == 0) {
199     temp.insert(JOY); // Freude
200     temp.insert(DISGUST); // Ekel
201   }
202   else if((value.compare(HIGH1) == 0) || (value.compare(HIGH2) == 0)) {
203     temp.insert(SURPRISE); // Ueberraschung
204     temp.insert(ANGER); // Wut
205     temp.insert(JOY); // Freude
206   }
207   else {
208     cerr << " Error: Could not read the data. Tried to read the
          following: "
209       << value << " . Are you sure that you set all macros right?" <<
           endl;
210   }
211
```

```
212   return temp;
213 }
214
215 map<string, double> plausibility(vector<Evidence> data) {
216   std::cout << "Plausibility:" << endl;
217
218   map<string, double> plausibility;
219
220   //init
221   for(const auto& emotion : OMEGA) {
222     plausibility[emotion] = 0.0;
223   }
224
225   plausibility[PL_MAX] = 0.0;
226
227   for(size_t i = 0; i < data.size(); i++) {
228     //for each emotion
229     for(const auto& emotion : OMEGA) {
230       if(data[i].emotions.find(emotion) != data[i].emotions.end()) {
231         //found element
232         plausibility[emotion] += data[i].value;
233       }
234     }
235   }
236
237   //print
238   string e;
239   bool init = false;
240   for(const auto& emotion : OMEGA) {
241     if(init == false) {
242       init = true;
243       plausibility[PL_MAX] = plausibility[emotion];
244       e = emotion;
245     }
246
247     if(plausibility[PL_MAX] < plausibility[emotion]) {
248       plausibility[PL_MAX] = plausibility[emotion];
249       e = emotion;
250     }
251
252     std::cout << "PI for Emotion: " << emotion << " : " << plausibility[
          emotion] << " " << endl;
253   }
254
255   std::cout << "Max PI found in Emotion: " << e << " : " << plausibility
        ["max"] << endl;
256
```

```
257   return plausibility;
258 }
259
260 void calculate_plausibilities(string input, string output) {
261   vector<string> data = readFileInVector(input);
262
263   vector<vector<Evidence>> evidences;
264   vector<map<string, double>> plausibilities;
265
266   for(size_t i = DATA_START; i < data.size(); i = i + 4) {
267     cout << "Takt " << data[i] << endl;
268
269     set<string> speedEmotions = getEmotionOfSpeed(toDouble(data[i + 1]))
          ;
270     double speedConfidence = CONF_SPEED;
271
272     set<string> pitchEmotions = getEmotionOfPitch(data[i + 2]);
273     double pitchConfidence = CONF_PITCH;
274
275     set<string> intensityEmotions = getEmotionOfIntensity(data[i + 3]);
276     double intensityConfidence = CONF_INTENSITY;
277
278     evidences.push_back(
279       dempster(speedEmotions, speedConfidence,
280             pitchEmotions, pitchConfidence,
281             intensityEmotions, intensityConfidence));
282
283     plausibilities.push_back(plausibility(evidences.back()));
284
285     std::cout << endl;
286   }
287
288   writeResultsToFile(input, output, evidences, plausibilities);
289
290   cout << " Note: Done." << endl;
291 }
292
293 int main() {
294   global_init();
295
296   char pause;
297   std::cout << "Press 'enter' to start with the file " << FILE1 << "."
          << endl;
298   pause = getchar();
299   calculate_plausibilities(FILE1, OUTPUT1);
300
```

```
301   std::cout << "Press 'enter' to continue with the file " << FILE2 << ".
          " << endl;
302   pause = getchar();
303   calculate_plausibilities(FILE2, OUTPUT2);
304
305   std::cout << "To exit this programm press 'enter'." << endl;
306   pause = getchar();
307
308   return 0;
309 }
```

Listing 1: Code aus Main.cpp

dempster.h

```
1  #ifndef _DEMPSTER_H
2  #define _DEMPSTER_H
3
4  #include <vector>
5  #include <string>
6  #include <set>
7  #include "settings.h"
8
9  namespace std {
10
11   struct Evidence {
12     Evidence() {} // to dynamically add elements to a vector
13     set<string> emotions;
14     double value = 0.0;
15   };
16
17   vector<Evidence> dempster(set<string> speedEmotions, double
         speedConfidence,
18                 set<string> pitchEmotions, double pitchConfidence,
19                 set<string> intensityEmotions, double
                       intensityConfidence) {
20
21     // Initialization of the three evidences
22     vector<Evidence*> m_1;
23     Evidence m_1a;
24     Evidence m_1o; // Omega
25     vector<Evidence*> m_2;
26     Evidence m_2a;
27     Evidence m_2o; // Omega
28     vector<Evidence*> m_3;
```

```
29      Evidence m_3a;
30      Evidence m_3o; // Omega
31
32              // m_1: Speed
33      m_1a.emotions = speedEmotions;
34      m_1a.value = speedConfidence;
35      m_1o.emotions = OMEGA;
36      m_1o.value = (1 - m_1a.value);
37      m_1.push_back(&m_1a);
38      m_1.push_back(&m_1o); //OMEGA - index 1
39
40                  // m_2: Pitch
41      m_2a.emotions = pitchEmotions;
42      m_2a.value = pitchConfidence;
43      m_2o.emotions = OMEGA;
44      m_2o.value = (1 - m_2a.value);
45      m_2.push_back(&m_2a);
46      m_2.push_back(&m_2o); //OMEGA - index 1
47
48                  // m_3: Intensity
49      m_3a.emotions = intensityEmotions;
50      m_3a.value = intensityConfidence;
51      m_3o.emotions = OMEGA;
52      m_3o.value = (1 - m_3a.value);
53      m_3.push_back(&m_3a);
54      m_3.push_back(&m_3o); //OMEGA - index 1
55
56                  // m_1 U m_2 = m_12
57      vector<Evidence> m_12; // Call by copy required - variables are set
            by lower scope (for loop)
58
59      for(size_t i = 0; i < m_1.size(); i++) {
60        for(size_t j = 0; j < m_2.size(); j++) {
61          Evidence temp;
62          temp.value = m_1[i]->value * m_2[j]->value;
63
64          set_intersection(m_1[i]->emotions.begin(), m_1[i]->emotions.end
                (),
65                       m_2[j]->emotions.begin(), m_2[j]->emotions.end(),
66                       inserter(temp.emotions, temp.emotions.begin()));
67
68          m_12.push_back(temp);
69        }
70      }
71
72      // Check for empty set and correct if necessary
73      // (only possible once: in intersection m_1 U m_2 without Omega)
```

```
74      for(size_t i = 0; i < m_12.size(); i++) {
75        if(m_12[i].emotions.empty()) {
76          cout << " Note: Found empty set in m_12" << endl;
77
78          double k = 1 / (1 - m_12[i].value); // calculate correction
                factor k
79
80          m_12.erase(m_12.begin() + i); // delete empty element i
81
82          for(size_t a = 0; a < m_12.size(); a++) { // correct remaining
                elements with k
83            m_12[a].value *= k;
84          }
85
86          cout << " Note: Corrected m_12 with value k=" << k << endl;
87
88          break; // leave for loop since there can only be one correction
89        }
90      }
91
92      // m_12 U m_3 = m_123
93      vector<Evidence> m_123; //call by copy required here - because the
            variables are created in a loop dynamically - do not save only
            pointers in this vector
94
95      for(size_t i = 0; i < m_12.size(); i++) {
96        for(size_t j = 0; j < m_3.size(); j++) {
97          Evidence temp; //empty element
98          temp.value = (m_12[i].value) * (m_3[j]->value);
99
100         set_intersection(m_12[i].emotions.begin(), m_12[i].emotions.end
                (),
101                  m_3[j]->emotions.begin(), m_3[j]->emotions.end(),
102                  inserter(temp.emotions, temp.emotions.begin()));
103
104         m_123.push_back(temp);
105       }
106     }
107
108     // Check for empty set and correct if necessary
109     // (multiple empty sets possible)
110     bool correction = false;
111     double sum_empty = 0.0;
112     vector<size_t> index_to_delete;
113
114     for(size_t k = 0; k < m_123.size(); k++) {
115       if(m_123[k].emotions.empty()) {
```

```
116         cout << " Note: Found empty set in m_123" << endl;
117
118         correction = true;
119         sum_empty += m_123[k].value;
120
121         index_to_delete.push_back(k); //remember which element needs to
                be removed
122       }
123     }
124
125     // correct if empty set found
126     if(correction) {
127       double k = 1 / (1 - sum_empty);
128
129       // delete empty elements
130       for(size_t b = 0; b < index_to_delete.size(); b++) {
131         m_123.erase(m_123.begin() + index_to_delete[b]);
132       }
133
134       // correct remaining elements with k
135       for(size_t a = 0; a < m_123.size(); a++) {
136         m_123[a].value *= k;
137       }
138
139       cout << " Note: Corrected m_123 with k=" << k << endl;
140     }
141
142     return m_123;
143   }
144
145 }
146
147 #endif // _DEMPSTER_H
```

Listing 2: Code dempster.h

settings.h

```
1 /*
2   This file contains all settings
3 */
4
5 #ifndef _SETTINGS_H
6 #define _SETTINGS_H
7
```

```
8  // I/O File Settings
9  #define FILE1 "E_004.csv" // Path of 1st file
10 #define FILE2 "E_004b.csv" // Path of 2nd file
11 //#define FILE3 "E_004c.csv" // Path of 3rd file - Not applieable for
       this project
12
13 #define OUTPUT1 "resulta.csv" // Path of Output File
14 #define OUTPUT2 "resultb.csv" //Path of second output file
15
16 #define DATA_START 4 //the data starts at element of 4, or index 4 (to
       skip header)
17
18 // Definitions of Emotions
19 #define FEAR "A" // Angst
20 #define SURPRISE "U" // Ueberraschung
21 #define ANGER "W" // Wut
22 #define JOY "F" // Freude
23 #define DISGUST "E" // Ekel
24 #define SORROW "T" // Traurigkeit
25
26 // Definitions of Confidences
27 #define CONF_SPEED 0.6
28 #define CONF_PITCH 0.8
29 #define CONF_INTENSITY 0.7
30
31 // Definitions
32 // ...for Speed
33 #define SLOW 2.5 //untere Grenze
34 #define FAST 5.6 //obere Grenze
35
36 // ...for Pitch and Intensity
37 #define LOW1 "sehr niedrig"
38 #define LOW2 "niedrig"
39 #define MEDIUM "normal"
40 #define HIGH1 "hoch"
41 #define HIGH2 "sehr hoch"
42
43 // Definitions for Plausibility
44 #define PL_MAX "max"
45
46 // Definitions for Output File
47 #define CSV_SEP ';'
48 #define CSV_ENDL '\n'
49 #define HEAD_CLK "Takt"
50 #define HEAD_MAX "Max"
51 #define HEAD_MAXVAL "PL of Max"
52
```

```
53 namespace std {
54
55   set<string> OMEGA;
56
57   void global_init() {
58     OMEGA.insert(FEAR);
59     OMEGA.insert(SURPRISE);
60     OMEGA.insert(ANGER);
61     OMEGA.insert(JOY);
62     OMEGA.insert(DISGUST);
63     OMEGA.insert(SORROW);
64   };
65
66 }
67
68 #endif // _SETTINGS_H
```

Listing 3: Code settings.h